An Automated Method for Data Path Construction

by

Darren A. Schmidt

Submitted to the Department of Electrical Engineering

and Computer Science in Partial Fulfillment of the Requirements

for the Degrees of Master of Engineering and Bachelor of Science

in Electrical Engineering and Computer Science at the

Massachusetts Institute of Technology

May 1998

Author_____

Department of Electrical Engineering and Computer Science
May 20, 1998

Certified by_____

Chris Terman
Thesis Supervisor

Accepted by_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

An Automated Method for Data Path Construction

by

Darren A. Schmidt

Submitted to the Department of Electrical Engineering

and Computer Science in Partial Fulfillment of the Requirements

for the Degrees of Master of Engineering and Bachelor of Science

in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

Abstract

The construction of custom data paths is a time-consuming and difficult problem. Even with a library of standard cells, custom hand layout can take months. In addition, minor design changes or revisions for next generation designs can be tedious and difficult. By creating an automated system, data path design can be greatly simplified and dramatically sped up. Not only does this aid in the initial design, but it also enhances the designers ability to make design changes. Now these revisions involve simply modifying lines of code, rather than the painful process of redoing a custom layout. In addition, changes to new technology are easily accommodated. Rather than redo the entire design, only the standard cells and the affected constants in the program need to be revised. Then, the program can be rerun and a new design can be created. One of the drawbacks of automation is that it reduces the designer's flexibility in layout. In addition, many existing tools are complex and difficult to use. The goal of this project is to create an easy-to-use, automated design tool that results in all of the benefits previously mentioned, while at the same time giving the user maximum flexibility in design. This thesis describes the implementation of such a system, as well as the results of using this tool to create a substantial design.

Thesis Supervisor: Chris Terman
Title: Professor

Acknowledgements

There are many people without whom this thesis would not have been possible and to them I give my thanks:

To Professor Gerry Sussman and Amorphous Computing, for giving me the opportunity to work in their group.

To Professor Chris Terman, for not only being a great thesis advisor, but also a great teacher and friend.

To my family, for providing me with their continued love and support.

To my roommates and friends, for helping me to enjoy my MIT experience.

Contents

List of Figures

Chapter 1

Introduction

This thesis describes the implementation and test drive of a standard cell layout tool for data paths. This chapter begins with a summary of previous work in the area of automated placement and routing tools and a description of the goals of this project. It then provides an introduction to the larger project for which this work is being done and concludes with an outline of this thesis.

1.1 Background

The automated conversion of a VLSI design to layout has been an active area of research for the last 15 to 20 years. This task entails three main phases, cell design, placement and routing. A number of algorithms have been developed to solve the last two of these problems.

There have been two main approaches to the problem of placement, the task of placing modules adjacent to each other in order to minimize area or cycle time. The Min-cut algorithm[1] performs placement by partitioning. It divides the blocks to be placed at the top level of the design into two groups of approximately equal area, while minimizing the number of connections required across this divide. This process is then repeated for the two halves, splitting the layout into quarters, eighths, and so on, until the leaf cells are reached. Another widely used technique is one in which the movement of modules is likened to thermal annealing.[2] Modules are initially moved randomly, and the "temperature" of the layout is evaluated by applying some measure such as routing area

or timing. As the layout "cools", the routing and/or timing improves. For each proposed subblock movement, the resulting temperature is calculated. If it is higher than the current temperature, the move is not completed. To avoid local minima, the "melt" is reheated and then recooled according to an "annealing schedule". The lowest temperature configuration found so far is saved before each movement, and the algorithm stops when after a given number of tries a new minimum is not found. Both of these methods have been widely used, although they require significant runtime.

A variety of routing tools have been developed as well. Routing is the task of taking a module placement and a list of connections and connecting the modules with wires. One common type of router is a channel router, which routes rectangular channels. The greedy channel router[3] wires up the channel in a left-to-right, column-by-column manner. Within each column the router tries to maximize the utility of the wiring produced, using simple "greedy" heuristics. Global routers[4] are channel routers that attempt to minimize the total number of wiring tracks required by examining the entire design and generating all possible net segments for each net. Then, it chooses from among all possible net connections in order to minimize the total channel density. There are also more complicated routers, such as maze routers. These can route just about any configuration, but have comparatively long running times. They are usually reserved for very difficult routing problems. These routing methods have also been widely used, and there are many custom routing tools as well.

There currently exist a number of tools that perform both routing and placement for standard cell designs. One such tool that is in wide use is TimberWolf.[5] This tool uses simulated annealing to perform the placement, and then a global router to perform

the routing. This package provides good results and substantial area savings, compared to other standard cell layout methods.

The above mentioned placement and routing tools provide reasonable solutions to the general VLSI design problem. However, they fall short in a few areas when it comes to data path design. The current automated systems minimize the role of the designer. In addition, they don't group standard cells by function, but rather to optimize area or timing. Thus, it can be difficult to understand the resulting layout. Finally, they are complicated to use and often require considerable amounts of run-time.

1.2 Objectives

The goal of this project is to create a tool that provides the best of both worlds. That is, it should provide the benefits of automation and reduce design time. But, it should do this while still giving the designer complete control over the layout. It should also result in a logical grouping of standard cells, so that designs can be easily followed and debugged. Finally, it should be easy to use and relatively fast. Since the designs of data paths and their resulting layouts tend to be regular, placement and routing are often quite straightforward from the designer's point of view. Thus, a designer-directed layout strategy should perform very well without too much effort on the part of the designer. In addition, this strategy would result in a logical placement of standard cells that could be easily followed by the designer. And unlike many of the more complicated algorithms, it would be relatively easy to use and have a very short run-time.

This thesis attempts to illustrate the usefulness of an automated system of the type previously described. First, a prototype of such a system will be created. This will

involve the layout of the standard cells that are necessary for data path construction, as well as implementing a method for building with these cells. This method of construction will be created in the form of a Lisp program. Finally, a sample data path will be constructed using this prototypical tool. While by no means the optimal implementation, this project is instead intended to prove the usefulness of such a tool. Once the basic tool is built and tested, it can then be refined and the user-interface can be improved upon at a later date.

1.3 The Amorphous Computing Project

One of the major current activities of the Mathematics and Computation Group of the Artificial Intelligence Lab is the study of amorphous computing. Drawing from natural phenomenon such as a swarm of bees cooperating to construct a hive, this group is attempting to address the fundamental organization of computer systems. Two fundamental questions they have raised are:

1) How do we obtain coherent behavior from the cooperation of large numbers of unreliable parts that are interconnected in unknown, irregular, and time-varying ways?

2) What are the methods for instructing myriads of programmable entities to cooperate to achieve particular goals?

The objective of this research is to create the system-architectural, algorithmic, and technological foundations for exploiting programmable materials. These "programmable materials" are materials that incorporate vast numbers of programmable elements that react to each other and the environment. In order to do this, the group is attempting to identify engineering principles for organizing and instructing a myriad of

programmable entities to cooperate to achieve pre-established goals, even though the individual entities are connected in unknown, irregular, and time-varying ways.

Amorphous computing is inspired by the recent developments in molecular biology and in microfabrication. Each of these is the basis of a kernel technology that makes it possible to build or grow huge numbers of almost-identical information-processing units, with integral actuators and sensors, at almost no cost. Microelectronic components are so inexpensive that the group can imagine mixing them into materials that are produced in bulk, such as paints, gels, and concrete. Such ''smart materials" will be used in structural elements and in surface coatings, such as skins or paints.[6]

The role that the work of this thesis will play is to help create the second and third generation of the amorphous computing hardware. The group hopes to incorporate all of the necessary hardware for each individual element onto a single chip. This includes the processor, memory, and a radio for communications. The tool developed and used for this thesis will be used to create the processor portion of this chip. For the second generation of hardware, an existing processor design will be created in our new design environment. This will then be combined with memory and the radio, and put onto a single chip. It is hoped that this tool will eventually be used to implement a microprocessor similar to the Hitachi SH-3, which the group is currently developing. This enhanced processor would then be used for the third-generation of amorphous computing hardware.

1.4 Thesis Outline

This chapter has presented a summary of past research in automated layout tools, as well as an overview of this project.

Chapter 2 delves into the first part of this project, the creation of a library of standard cells.

Chapter 3 then turns to the implementation of the layout tool that is used to put these standard cells together to create circuits.

Chapter 4 focuses on using this tool to create larger data path elements.

Chapter 5 then describes the use of this tool to create a substantial data path design and assesses the performance of this tool in performing this task.

Finally, Chapter 6 concludes this thesis with the author's comments on this experience.

Chapter 2

The Building Blocks: Standard Cells

The first part of this thesis involved the creation of a standard cell library. This chapter details the design of the standard cells, which form the building blocks for data path construction. These are the only parts of any given design that are laid out by hand, which was done using the Magic layout editor[7]. The first section of this chapter gives a brief overview of the new design tool, which will aid in the understanding of the standard cell design. The second section details the common set of design rules to which all standard cells adhere. The third section details how the design of the standard cells facilitates placement and routing, while the fourth section describes the components that are included in the library.

2.1 System Overview

In order to understand the standard cells, it is useful to have a general knowledge of the system in which they will be used. The designer will write a program that utilizes this tool to generate their design. Within this program, they will specify the layout by tiling and copying existing standard cells in a specific fashion. Thus, all placement will be completely controlled by the designer. The designer will then connect these standard cells by specifying which nodes they wish to connect, as well as the channel(s) in which this connection should be made. The process currently being used has four independent routing layers: one poly and three metal. The poly and metal1 layers are reserved for in-cell routing. Thus, the general inter-cell routing method used will be to divide the design

into a large grid of horizontal and vertical channels. Top-level routing will be done on this two-dimensional grid, with metal2 runs in the vertical direction and metal3 runs in the horizontal direction. Vertical metal2 runs will begin and end each wire by connecting to the desired node within a standard cell. They will then be connected by a horizontal metal3 run, in the horizontal channel specified by the user. In this way, the designer will specify both the placement and the routing of their design. An example of the routing scheme can be seen in Figure 2-1.

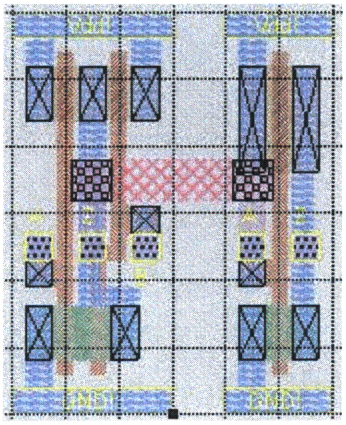

Figure 2-1: Basic Routing Plan

2.2 A Basic Standard Cell

A basic standard cell is illustrated in Figure 2-2. This 2-input NAND gate demonstrates some of the main properties of all standard cells. Each standard cell includes power and ground rails, which border the remaining circuitry. A row of PFET's lines the top of the cell, while a row of NFET's complements it at the bottom. In between, the appropriate connections are made via poly or metal1. Nodes that need to be

accessed in the next level up in a design are connected to metal2 contacts centered at the intersection of a horizontal and a vertical channel, to allow for over-the-cell routing.

Transistors were sized with the following points in mind. For most data path circuitry, transistors can be fairly small. As a result, many transistors are minimum size. Since PFET's are weaker than NFET's, they are generally twice as wide as the corresponding NFET. Gates were also designed to match the speed of a minimum sized inverter. For example, if two transistors are in parallel, then their width is doubled, as can be seen in the NAND gate. In most cases, this is the most that was done to optimize transistor sizes.

This particular processor design is most concerned with functionality, so most standard cell circuitry is kept simple and reliable. Most standard cells consist mainly of complementary logic and transmission gates. There are no dynamic logic gates, so that recharging is not an issue.

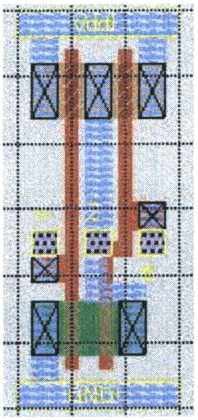

Figure 2-2: Basic Gate Layout

2.3 Standard Cell Layout

The standard cells were laid out in a way to facilitate two main functions. First, features have been incorporated to facilitate the tiling of multiple standard cells. Second, they were laid out in such a way as to ease the routing of wires between standard cells.

2.3.1 Facilitating Tiling

The main concern in the tiling of standard cells is the connection of power and ground rails. This problem is addressed by incorporating a few features into the standard cells. First, all standard cells have a common height. In addition, each has an identical set of power and ground rails, as described in the previous section. The width of these rails is half the minimum width, or 4 lambda. This is so that when cells are stacked together vertically, a minimum width power line will be created. The layout tool has features that place the power/ground contacts as well as the ability to space out stacked cells if necessary. Finally, these power and ground rails extend two lambda past their standard cell circuitry, so that when two cells are placed next to each other the rails make a connection, with no design rule errors.

2.3.2 Simplifying the Routing

The most important aspect considered in coming up with a set of standard cell design rules was how to make routing as simple as possible. In order to prevent conflicts with wires running between cells, the standard cells do not contain any metal2 or metal3 wires. In addition, to make it easy for the program to connect wires, all of the metal2 contacts are placed in the center of an intersection of a vertical and a horizontal channel.

Finally, the height and width of all standard cells are multiples of the horizontal channel height and vertical channel width, respectively. This is so that when multiple cells are tiled together either vertically or horizontally, each cell remains "on grid" and the contacts remain at a channel intersection, as described above. For our particular process, the minimum width of contacted metal2 wires is 4, while the minimum width of contacted metal3 wires is 6. In addition, the minimum spacing between metal2 or metal3 wires is 4. Thus, these standard cells were designed around vertical channels with a width of 8 lambda and horizontal channels with a height of 10 lambda. Note that in order to make the 8 lambda vertical channels work error-free, m2/m3 contacts in the same horizontal channel and adjacent vertical channels must be offset to the left or right by one. This offsetting is specified by the user, when necessary.

2.4 Standard Cell Library

The library of standard cells includes all leaf nodes that are necessary for the construction of a processor data path. This includes basic complementary gates, registers, and muxes. As mentioned previously, no dynamic logic is used and most gates are kept simple in order to insure functionality. It is hoped that this library can also be used to implement other data paths as well.

2.4.1 Basic Gates

A considerable portion of the library consists of simple, complementary gates. This includes inverters, NAND, NOR, AND, OR, and-or-invert, or-and-invert, and buffer gates. Most of these are made up of minimum-width transistors, although there are also

larger versions. These gates with larger FET's are used mainly in places where large drive is necessary.

2.4.2 XOR and XNOR

Rather than one of the smaller, exotic XNOR circuits that are widely known, a more reliable, slightly larger design was chosen. The XNOR circuit used can be seen in Figure 2-3. The feature that makes it attractive is that the input nodes are connected only to gates. In addition, it is made up of only complementary logic, with no transmission gates. Thus, it is fully restoring. This is important for standard cell design, since it makes this issue transparent to the system level designer. In addition, it is capable of driving a more substantial load than many of the other pass gate XNOR circuits.

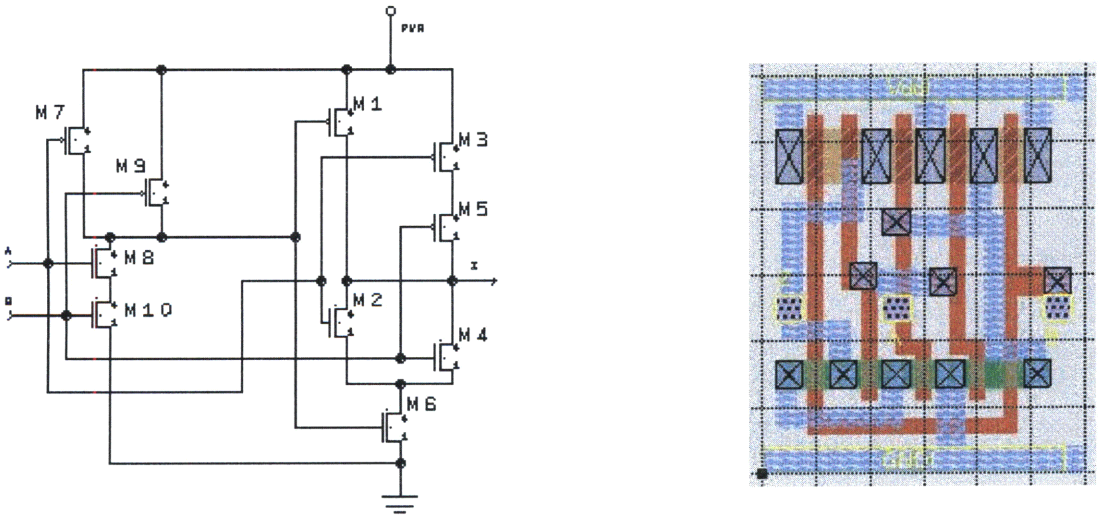

Figure 2-3: XNOR Schematic and Layout

2.4.3 Registers

Once again simplicity and reliability were chosen over more complicated circuitry, and a static "jam" register was chosen.[8] No dynamic gates are used, so charging is not an issue. This design also employs weak feedback inverters, which reduces the load on the clock. A schematic of this register can be seen in Figure 2-5.

2.4.4 Multiplexor

The standard two-input multiplexor was designed using transmission gates. It can be seen in Figure 2-4.

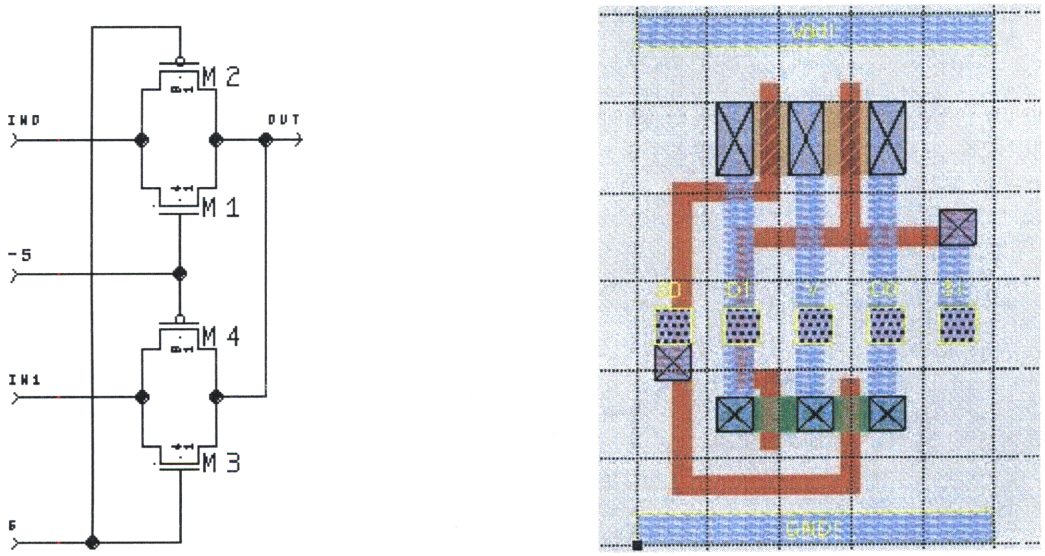

Figure 2-4: 2:1 Mux Schematic and Layout

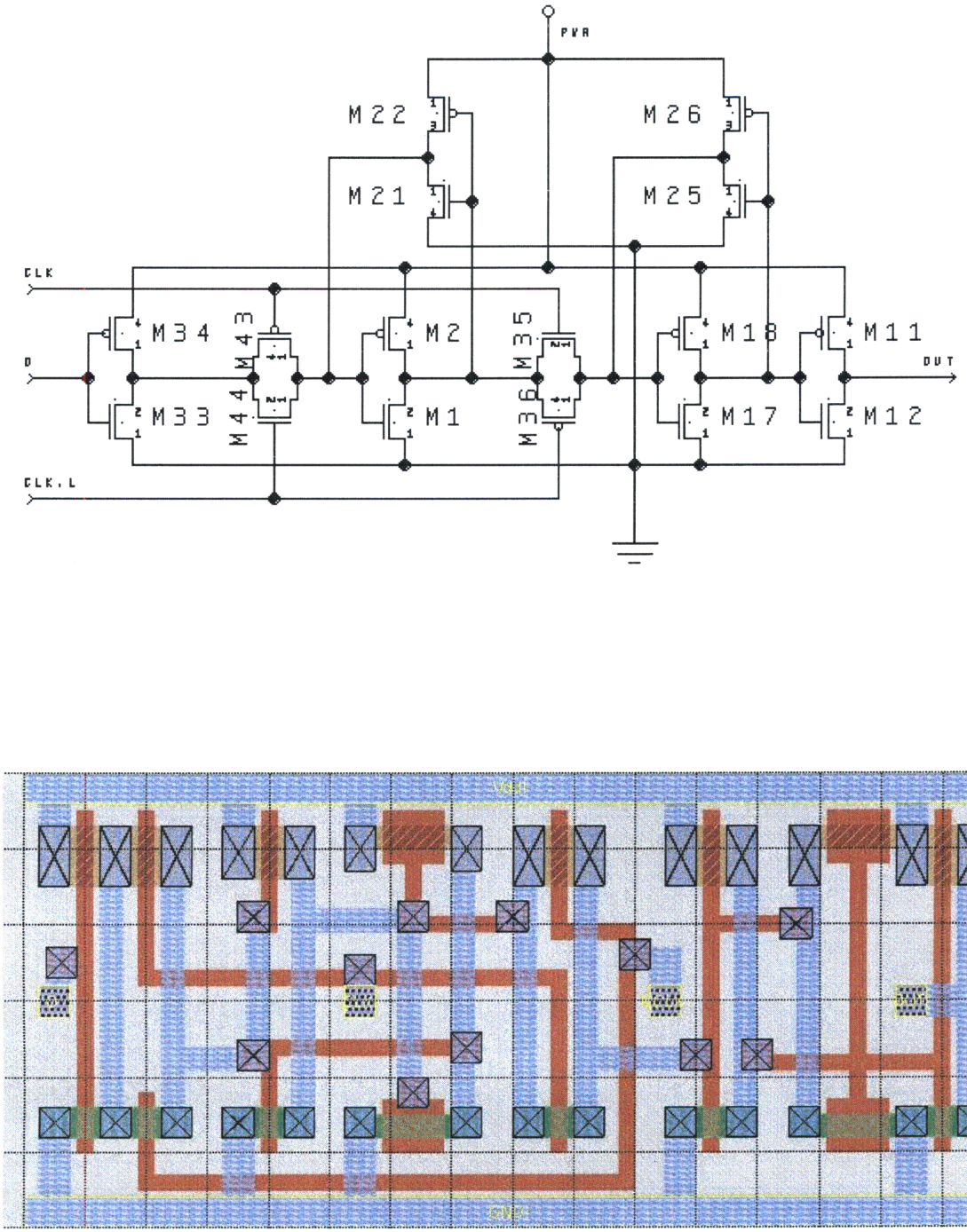

Figure 2-5: Jam Register Schematic and Layout

Table 2-1: Standard Cell List

Standard Cell	Number of Transistors	Width (channels)
Inverter – 4/2	2	2
Inverter – 8/4	2	2
Inverter – 16/8	2	3
Inverter – 32/16	2	5
Inverter – weak	2	4
Buffer – 4/2, 8/4	4	4
Buffer – 8/4, 16/8	4	4
Buffer – 16/8, 32/16	4	4
Driver	2	7
Nand2	4	3
Nor2	4	3
And2	6	5
Or2	6	5
Aoi21	6	4
Aoi22	8	5
Aoi31	12	7
Oai21	6	4
Oai22	8	5
Jam Register	16	16
T-Gate	2	4
2:1 Mux	4	5
Tristate Driver	4	3
Tristate Inverter	4	4
Xnor	10	6
Xor	10	6

Chapter 3

The Place and Route Tool

Once the standard cell library was in place, the next step was to create the algorithm that would use these standard cells to create a design. This chapter focuses on the Lisp program that was created by several people in order to perform this task. This program, *magic.scm*, can be found in the appendix. An overview of the basic strategy was presented in chapter 2, so one is not given here. The first section of this chapter discusses the method adopted to solve this problem. Section two focuses on how this algorithm performs placement. Section three then details how routing is done.

3.1 Program Methodology

The members of this group decided that the most natural way to think about the various parts of this problem was as objects. Thus, the basic strategy employed is to convert magic files into data objects, manipulate them to create a design, and then convert this back into a Magic file. Finally, an attempt was made to make this tool functional in any process, rather than just the particular one used by this project. As a result, there is a section that assigns design rule parameters that are essential to this tool to the corresponding values for a specific process. So if the designer changes processes, they can simply change the value of these global variables, and the program will still work optimally, without creating design errors.

The first section of *magic.scm* contains the object definitions, as well as procedures to access these objects. One of the major objects is a magic-file. This object

consists of additional objects: layers, one of the possible paint layers in Magic, such as poly; uses, or subcells; and labels, which are names used to label nodes in a circuit. Each of these objects also has several procedures associated with it, such as instantiate-use, which is used to add a subcell to the current layout. As the reader will see shortly, this procedure is used in performing layout. Another object is a magic-box, which is a vector consisting of coordinates for the four corners of a box. This object is used to specify where to add paint to the layout, such as in wiring. Finally, there are two procedures, read-magic-file and write-magic-file, which are used to convert between this data representation of a layout and the Magic representation. These are basically parsing routines, which extract the desired information and format it appropriately.

3.2 Performing Placement

The next section of *magic.scm* deals with placement. There are two main procedures used to perform placement. The basic idea, as described previously, is to tile together subcells. The approach taken here is to first create a horizontal row of cells, which usually make up a single bit-slice. Then, these slices are tiled vertically, to create the data path. This is done with two main procedures. First, instantiate-and-tile is used to create a single bit, by specifying the horizontal direction. Then, one of two things is done. If the circuit being designed consists of identical bit-slices, then instantiate-vertical-array can be used to create an array of the subcircuit just designed. If the bits are different, then instantiate-and-tile can be used in the vertical direction. Instantiate-and-tile-vertical-array was added after some use of this tool, in order to simplify the the task of replicating a single subcircuit a large number of times.

Instantiate-and-tile takes in a number of arguments, including the direction in which to tile. Most of the arguments are lists: uses-to-instantiate, or the subcells that the designer wishes to tile together, in the order in which they will be tiled; instance-names, or the names to be given to the particular instances of the uses you are tiling; transform-generators, or transforms that you wish to apply to the instances you are tiling, such as flipping them upsidedown or sideways; xoffsets and yoffsets, which are used to offset the instances you are tiling in the x or y direction. The list of transforms is defined earlier in this section of the code, and they are performed by transforming the coordinates of a use appropriately. This algorithm then applies instantiate-use to each set of corresponding entries of these lists. It also calculates the correct position at which to place the next instance, by setting the variables xpos and ypos appropriately. Instantiate-and-tile-vertical-array is very similar, except that it does some things to simplify the process of creating an array. It takes in the name of the use you wish to instantiate and the number of times you wish to instantiate it. It then creates the lists described above of the appropriate length and calls instantiate-use on each set of the corresponding entries of these lists. It does assume several things, such as the name for each instantiated use, which is simply the name of the subcell with a number appended to it that corresponds to the number of that particular instance in the array. By using these procedures, the user is able to specify the exact placement of the subcells in their design.

3.3 Performing Wiring

The remainder of *magic.scm* deals with the task of connecting the placed uses together. As described before, this tool divides the layout into vertical and horizontal

routing channels in order to perform the routing. There are two important routing tasks. The first one is routing signals, or routing from one label to another. The second one is routing power and ground via a large grid.

3.3.1 Routing Signals

Routing of signals is done via the procedure instantiate-wire. This procedure takes as arguments the magic file being edited, the name of the use and the label in that use from which the wire will start, the name of the use and the label in that use where the wire will terminate, and a horizontal channel index. A label is a term Magic uses to give a name to a rectangular area on a particular layer. They are used to indicate where the "pins" for each cell are located. The procedure then routes a wire consisting of a metal3 horizontal wire in the specified channel, and metal2 vertical wires from the specified labels to this horizontal wire, with contacts at the intersections of these wires. These metal2 wires are painted in the vertical channel in which their labels occur. As mentioned previously, wires are created by making boxes and filling them in with the appropriate metal.

Instantiate-wire performs the above task by calling several other procedures. It uses instantiate-vertical-wire-segment to make the vertical wires. This procedure creates a box that spans from the center of the specified starting horizontal channel to the center of the specified ending horizontal channel in the specified vertical channel. This box is centered in the vertical channel, and has a width equal to the minimum metal2 width. This width is one of the previously mentioned global variables that is process-dependent. The box is then filled with metal2. Next, instantiate-horizontal-wire-segment is used to

make the horizontal wire and the contacts. This routine is similar to instantiate-vertical-wire, as it creates boxes and then fills them with the appropriate paint. The only difference is that this box spans from the center of the starting vertical channel to the center of the ending vertical channel, in the specified horizontal channel. Again, this box is centered in the horizontal channel, and has a width equal to the minimum metal3 width. This procedure also uses instantiate-contact to place the contacts at the ends of this wire. Instantiate-contact takes a layer, height, width, and vertical and horizontal channel indexes as arguments. It then creates a box of the desired height and width centered in the intersection of the specified channels and fills it with the specified layer.

There are several other wiring procedures that allow for more complicated wiring. Instantiate-five-segment-wire is very similar to instantiate-wire, except it contains two more jogs. In other words, the user specifies a starting horizontal channel, a vertical channel, and then an ending horizontal channel. As before, the procedure routes from the specified labels to this wire with vertical wires. It works very similar to the way instantiate-wire does. This procedure can be used when the vertical channel that one of the labels is in is already in use by another wire. Instantiate-single-layer-wire is another wiring procedure, and it can be used to make a wire on a single layer. This works similar to instantiate-wire, except it uses instantiate-specified-vertical-wire-segment and instantiate-specified-horizontal-wire-segment rather than instantiate-vertical-wire and instantiate-horizontal-wire. This procedure is useful for routing between adjacent cells in metal1 to save routing channels, where this is possible. There are two other procedures instantiate-vertical-strap and instantiate-horizontal-strap that are very similar to one another. They can be used to create a straight wire from one specified channel to the

next, in either the horizontal or vertical direction. These procedures create a horizontal/vertical wire on the specified layer, in the specified horizontal/vertical channel, and runs it from the left edge of the specified starting vertical/horizontal channel to the right edge of the specified ending vertical/horizontal channel. This is useful for doing things like tying a node to ground or VDD. These procedures enhance the basic instantiate-wire command by allowing the designer to route several different types of wires.

3.3.2 Routing Power and Ground

Power and ground are distributed across the chip by a grid whose properties are set by the designer. As explained in section 2, all standard cells include metal1 power and ground rails bordering the top and bottom, respectively. Thus, when cells are tiled together horizontally and vertically, these create horizontal power and ground wires, which are shared by adjacent rows. In order to handle large power requirements, rather than butting each vertical row up against its neighbor, these can be spaced out. Then, the space can be filled in with metal1, effectively creating wider power and ground rails. Since this program relies on the contacts within standard cells being on a grid in order to perform signal wiring, all standard cells have to remain on grid. Thus, the width of the horizontal rails must be increased in increments equal to the horizontal channel height. In order to make power distribution even more efficient, the designer also has the ability to add vertical metal2 wires to connect these horizontal power and ground rails. The designer therefore has several options when routing power: they have the flexibility to set the width of both the horizontal and vertical wires, as well as how frequently to place the vertical connections. The final task that needs to be handled is the placement of

substrate contacts, which are placed in the center of the horizontal power and ground rails, the minimum distance apart. Note that this happens to correspond to the vertical grid spacing.

The task of routing power and ground is performed by the function instantiate-power-grid. This procedure performs the three functions described above: it creates the horizontal rails of the desired width, it places substrate contacts, and it routes the vertical interconnections of the desired width and spacing. In order to do this, instantiate-power-grid takes in three arguments: horizontal-channel-span is used to specify the width of the horizontal wires, vertical-rail-spacing is used to specify the frequency of vertical rails, and vertical-rail-width is used to specify the width of the vertical rails. The procedure can be divided into three main parts according to the task that each part performs.

The first part of this procedure creates the horizontal rails. This is done with the named-let instantiate-horizontal-rails. This routine walks down the specified layout from top to bottom, painting horizontal metal1 wires at the appropriate places. This is done by calculating the topmost channel of the design, and then moving down the appropriate number of channels in between wires, depending on the desired horizontal-channel-span. In order to do the painting, it calls the previously mentioned procedure instantiate-horizontal-strap. The starting and ending vertical channels are the leftmost and rightmost channels of the design, which are also calculated by instantiate-power-grid.

The second part of this procedure is interleaved with the painting of these vertical wires, and that is the placing of contacts. There are actually two types of contacts that need to be placed, the substrate contacts and the metal2 contacts that will connect the horizontal and vertical power wires together. These contacts are placed on a particular

horizontal channel after it has been painted. This is done with the named-let instantiate-substrate-contacts. This section of the code walks across the particular horizontal wire in minimum contact spacing increments, determines which type of contact to paint at each location, and paints that contact. The determination of contact type is based on the vertical-channel-spacing argument, which tells the program where vertical wires will be going and thus where metal2 contacts need to be placed. The actual painting of the contacts is done with the append-box routine, which creates a box at the specified location and fills it with paint from the specified layer.

The final part of this routine is the painting of the vertical wires where specified by the designer. These are painted by the named-let instantiate-vertical-rails. This code once again walks across the entire layout horizontally. Only this time it paints pairs of adjacent vertical metal2 wires, one from the topmost power rail to the bottommost power rail and one from the topmost ground rail to the bottommost ground rail, with each pair of wires spaced apart by the specified vertical-rail-spacing. The painting of the vertical wires is done with instantiate-vertical-strap. The leftmost and rightmost vertical rails are created separately, by another let function called instantiate-vertical-boundary. This procedure walks down the entire design and paints two pairs of metal2 wires at each standard cell row location. It uses instantiate-vertical-strap to add wires from the current set of horizontal power and ground rails to the power and ground rails just below, on both sides of the design. This in effect creates vertical wires that run the length of the design and are connected to every horizontal power or ground rail. It is in this way that the power grid is created on a given layout.

Chapter 4

Data Path Elements

After developing the standard cell library and place-and-route tool, the next step was to use these to construct the components necessary for data path design. The idea was to create a set of data path elements that would be used in the processor, as well as future data path designs. These included elements such as a 32-bit adder and a 32-bit register. Rather than describe each of these blocks, this chapter will instead focus on the design and construction of one such component, the 32-bit adder. It is hoped that this will give the reader a better understanding of how this tool is used to construct circuits. The first section will describe the types of adder designs investigated for this project, and explain the design of the adder circuit that was finally chosen. Section two will then explain how the new tool was used to construct this adder out of standard cells. Section three concludes the chapter by commenting on other blocks that were constructed in a similar fashion.

4.1 32-Bit Adder Design

Addition is the basis for many processing operations. As a result, an adder is an important part of a data path cell library. A wide variety of implementations exist, and the choice as to which one to use depends on speed and density requirements. A variety of adders were designed and simulated for this project, and in the end a binary look-ahead carry adder was chosen.

4.1.1 Binary Full Adder

The familiar equations for a binary adder, where A and B are the summands, C_{in} is the carry input, S is the sum output, and C_{out} is the carry output are:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + C_{in}(B+A)$$

These can be factored into the alternative form:

$$S = P \oplus C_{in}$$

$$C_{out} = G + PC_{in} \quad \text{where}$$

$$G = AB \qquad \textit{generate signal}$$

$$P = A \oplus B \qquad \textit{propagate signal}$$

4.1.2 Ripple-Carry Adder

The simplest 32-bit adder design consists of 32 full adders chained together. The above full adder equations can be implemented with the following gates:

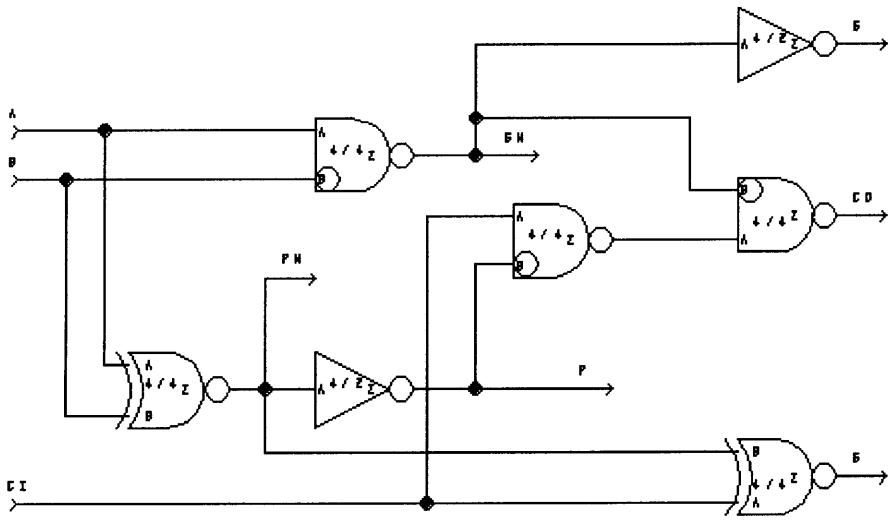

Figure 4-1: Full Adder Schematic

While small and simple, this adder is slow, due to the long carry chain. However, it does provide a base on which to compare the speed of more complicated adders.

4.1.3 Carry-Skip Adder

The worst-case delay for a ripple-carry adder occurs when a carry is generated in the lowest order bit and must propagate through the entire adder. One way to improve this delay is to "skip" the carry over a block of full adders if the propagate signals for each bit in the block are high.[9] This design consists of a ripple-carry adder divided into blocks, where a special circuit associated with each block quickly detects if all the bits to be added are different ($P_i = 1$ for all bits). In this case, the carry entering into the block may directly bypass it and be transmitted into the next block. It is also worth noting that if $A_i=B_i$ for some i in the block ($P_i = 0$), no block is skipped, but a carry is generated in that block. Thus, the carries are propagated in parallel, and the total time of computation is bound by the time of carry propagation in the largest block.

The carry-skip block is actually just the AND of every P_i in a given block. Since the fan-in of this AND gate equals the number of bits in a block, optimal block sizes are 3 or 4 bits. Additional improvement can be achieved by adding additional layers of skip. These layers would then skip multiple blocks, and they are conveniently identical in construction to lower skip levels. An optimal scheme consisting of two levels of skips was determined and tested. It was found that the speed of this adder was limited primarily by the speed of the skips, which become slow with high fan-in.

4.1.4 Manchester Adder

Another method commonly used to improve the speed of addition is a Manchester carry chain.[10] The objective behind this design is to propagate the carry as fast as possible, by using P_i and C_i to either propagate or generate a carry for each bit. One way to do this is with a multiplexor in the following configuration:

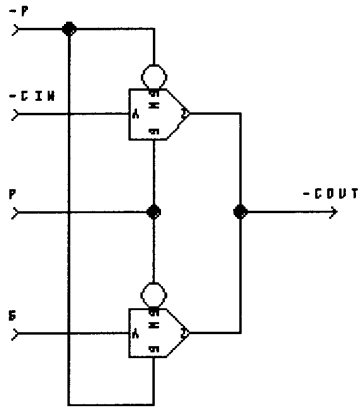

Figure 4-2: Manchester Carry Adder Schematic

However, such circuits rapidly slow down as bits are chained together, due to the resulting chain of transmission gates. So once again, the adder is divided into blocks and separated with restoring inverters. With the gates used for this project, the optimal block size was found to be 3. A further improvement can also be gained by applying the skip idea from before. Once again, blocks of bits can be skipped if all of the bits in a block are propagating a carry. The skip circuitry chosen for this design was a "conflict-free" circuit, which improves the speed by using a 3-input multiplexer that prevents conflicts at the wired OR node in the adder. This can be seen in Figure 4-3. The control signals T1, T2, and T3 are generated as follows: $T1 = -(P_0P_1)P_2$, $T2 = -P_2$, $T3 = P_0P_1P_2$.

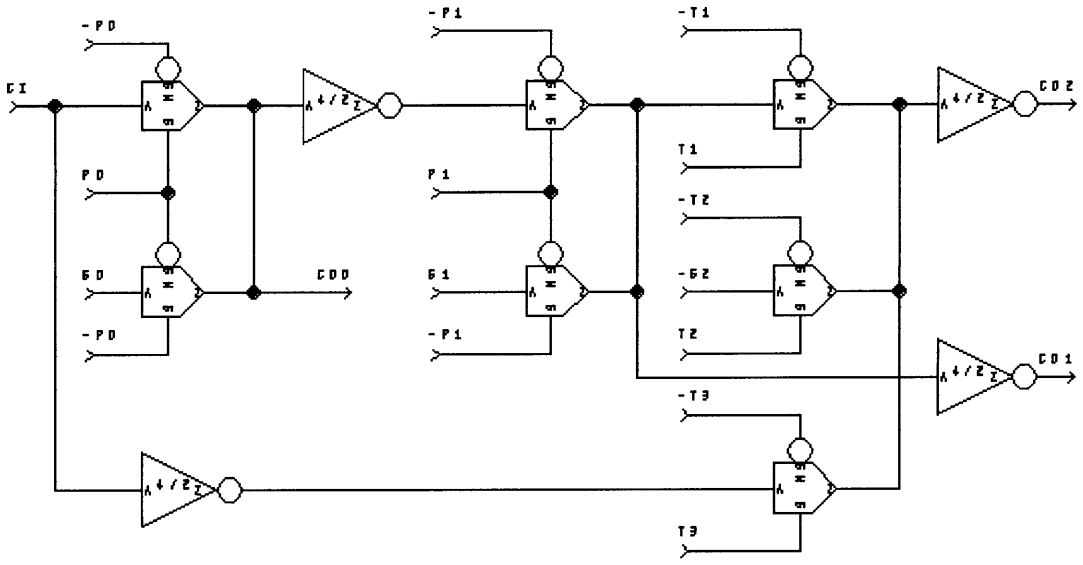

Figure 4-3: Conflict-Free Manchester Carry Skip Circuitry

This idea of skipping bits can also be applied to the resulting blocks. A 32-bit manchester adder with carry skip as described was constructed, and it turned out to be quite fast. This design consists of ten 3-bit manchester-skip blocks in series, with a full adder tacked onto the front and rear of this array. Then, block-skip circuitry is added on top of this, as can be seen in Figure 4-4. Once again, the rate-determining element turned out to be the skip circuitry, which became slow for large fan-in or significant loads. Thus, the buffering inverters must be kept small.

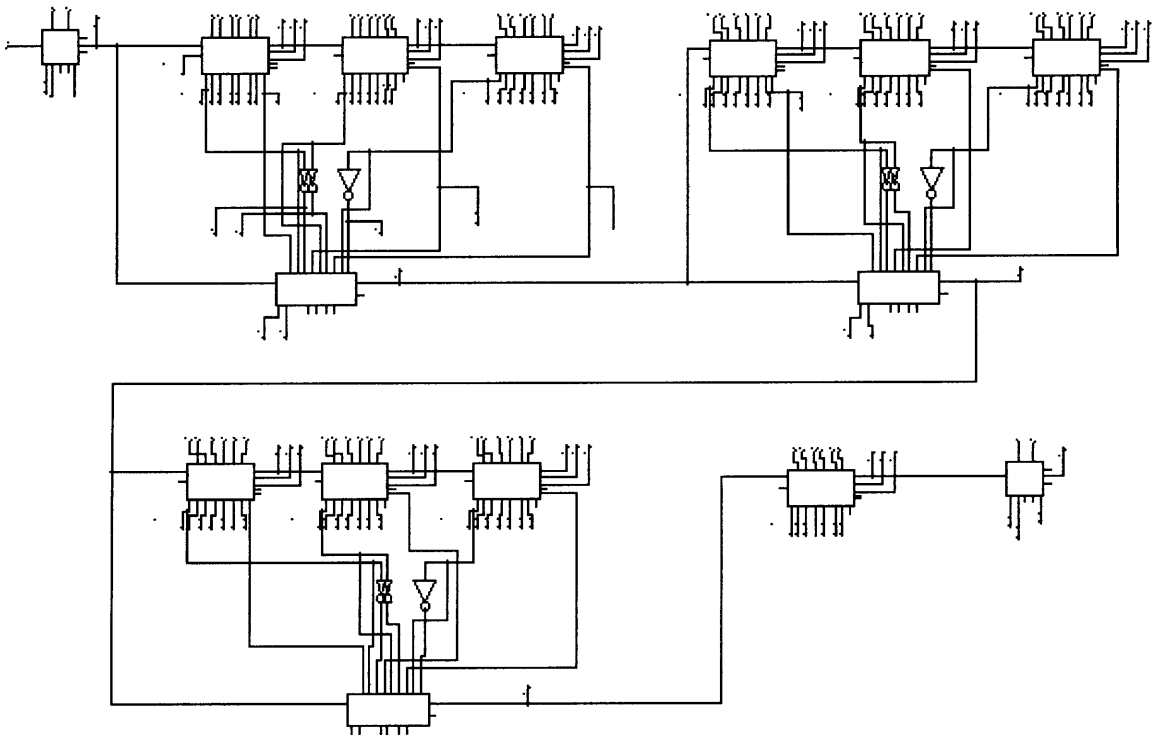

Figure 4-4: 32-Bit Manchester Carry-Skip Adder Schematic

4.1.5 Binary Lookahead Carry Adder

The idea behind this adder is to improve the linear growth of carry delay with the size of the input for an n-bit adder by calculating the carries for each stage in parallel. The carry for the ith stage, C_i, may be expressed as

$$C_i = G_i + P_iC_{i-1}$$

Expanding this yields

$$C_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \ldots + P_i\ldots P_1C_0$$

The size and fan-in of the gates needed to implement this carry-lookahead scheme becomes quite large as the number of bits increases. One alternative to this method is to compute the carries in a binary fashion.[11] Define a new operator # such that : $(g_1,p_1) \# (g_2,p_2) = (g_1 + p_1g_1, p_1p_2)$. It can be seen that the carry signals can be determined by G_i, where

$$(G_i,P_i) = \begin{cases} (g_1,p_1) & \text{if } i=1 \\ (g_i,p_i) \ \ldots \# \ldots (G_{i-1},P_{i-1}) & \text{if } 2 < i < n \\ (g_i , p_i) \# (g_{i-1},p_{i-1}) \ldots \# \ldots (g_1,p_1) & \end{cases}$$

In other words, by combining the p_i's and g_i's from two input bits, or the results from two previous combinations of such signals, the carry bits are constructed in a binary fashion. Such an adder can be divided up into sections, as shown in Figure 4-5.

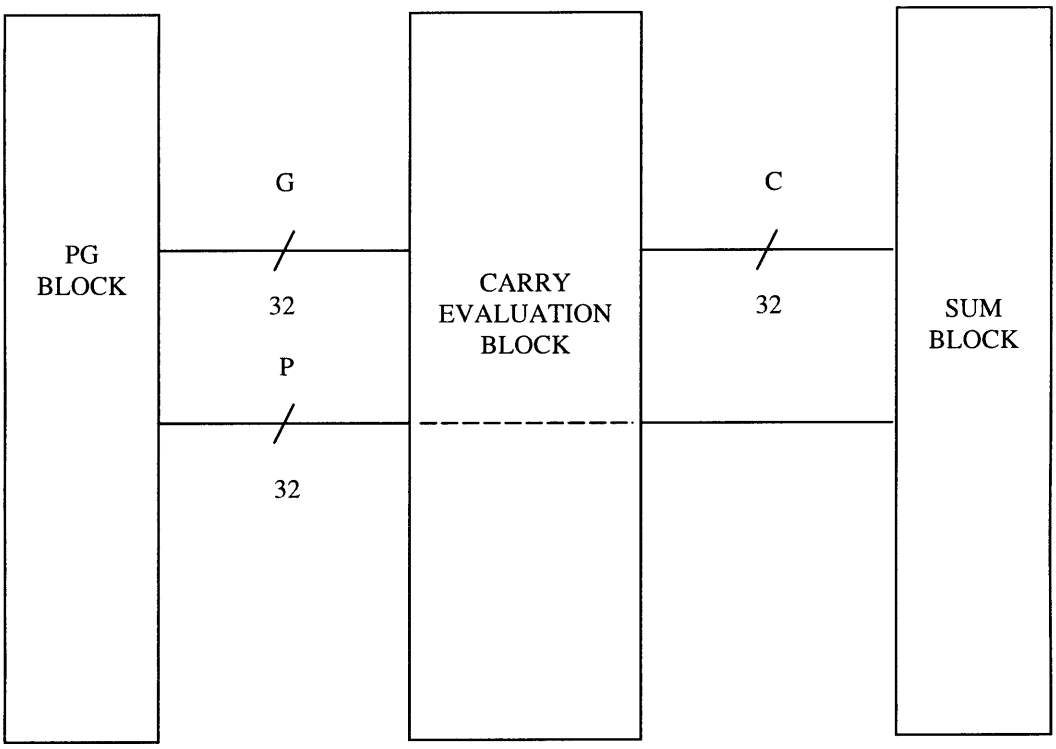

Figure 4-5: Binary Look-Ahead Carry Adder Block Diagram

The generate/propagate block simply consists of the logic required to compute P and G and can be seen in Figure 4-6. The sum block consists of an array of XNOR's. The carry block consists of the logic required to implement the # function. In order to improve speed and reduce the number of gates, we can complement alternating # blocks. These blocks can be seen in Figure 4-7, as BA and BB.

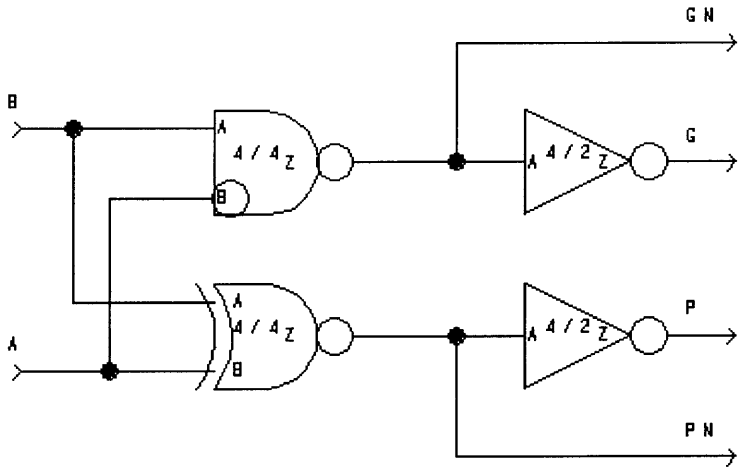

Figure 4-6: GP Block Schematic

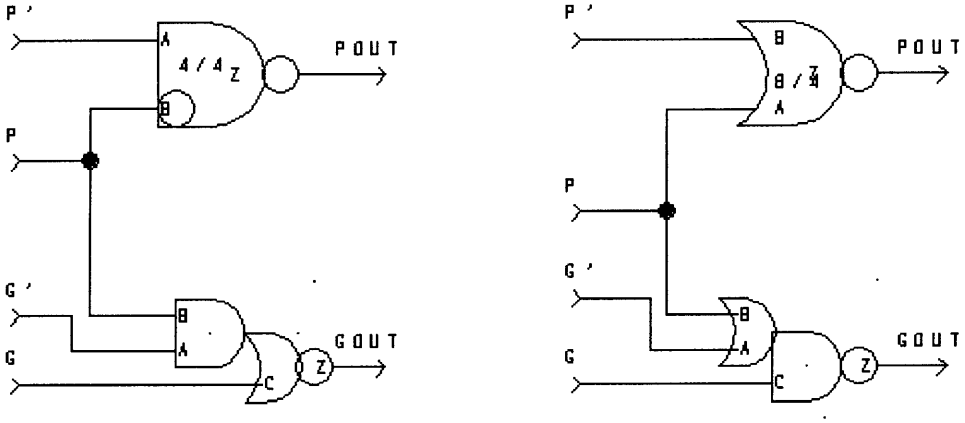

Figure 4-7: BA and BB Block Schematics

The carry evaluation block then looks like Figure 4-8 for a four-bit adder. This pattern is then iterated to create the carry block for an adder of the desired size. Since two bits are combined at each stage, a binary tree is formed, and thus the speed of this adder is proportional to $\log_2(n)$. Another nice feature of this adder is that it is very regular, and thus not too difficult to layout, especially with multiple layers of signal

wiring. Such an adder was designed and built, and it was the fastest of the previously mentioned adders.

One of the limiting factors of such an n-bit adder is that the most significant '#' operator in each of the j columns of the carry evaluation block must drive 2^j "#" blocks in the next stage. As n gets large, this can be a considerable load, as $j=\log_2(n)$. Thus, buffering of such signals becomes very important. One approach to this problem is to run these outputs directly to the most-significant "#" block in the next stage, while simultaneously buffering them to the remaining "#" blocks. This will maximize the speed at which these outputs reach the most significant block, which turns out to be the critical path. In the design used for this project, various sizes of inverters were used to perform this buffering. In the second to last stage, for example, a buffer consisted of a 16/8 inverter followed by a 32/16 inverter. A schematic for the 32-bit BLCA Adder can be seen in Figure 4-9.

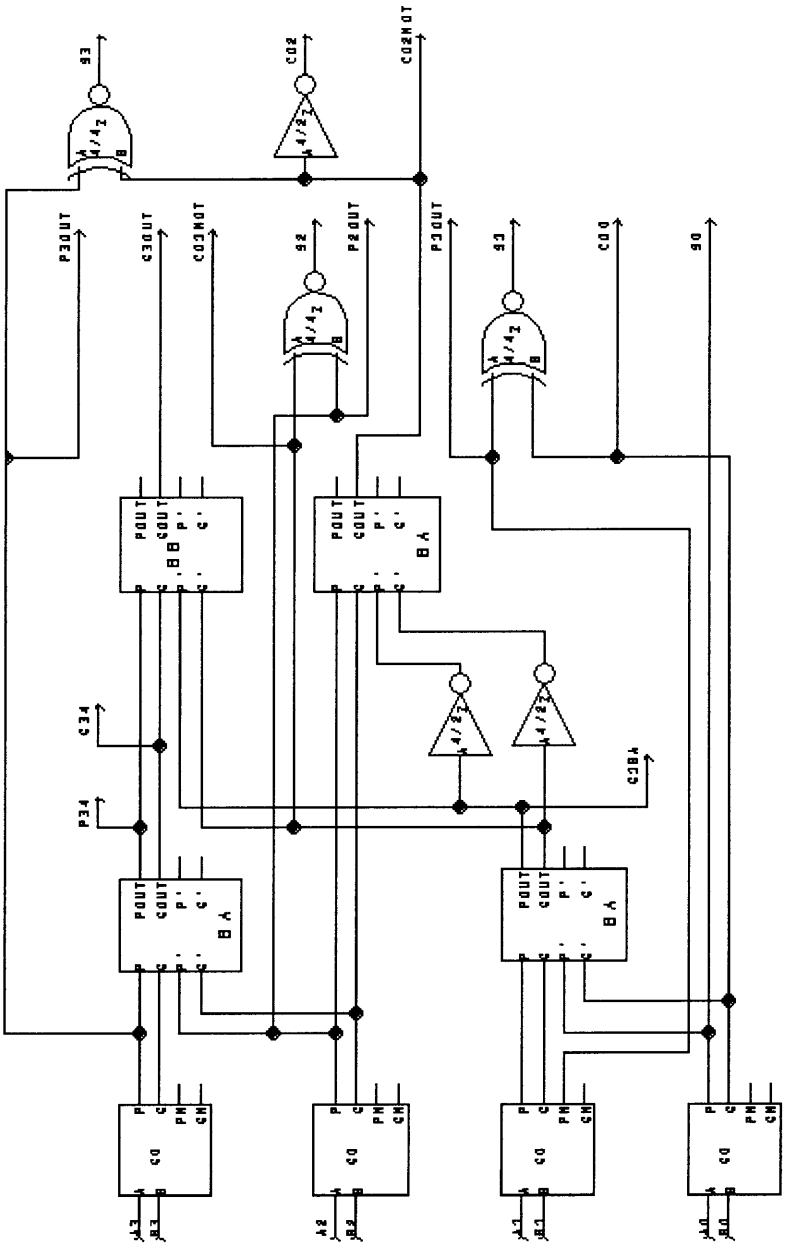

Figure 4-8: 4-Bit BLCA Adder Schematic

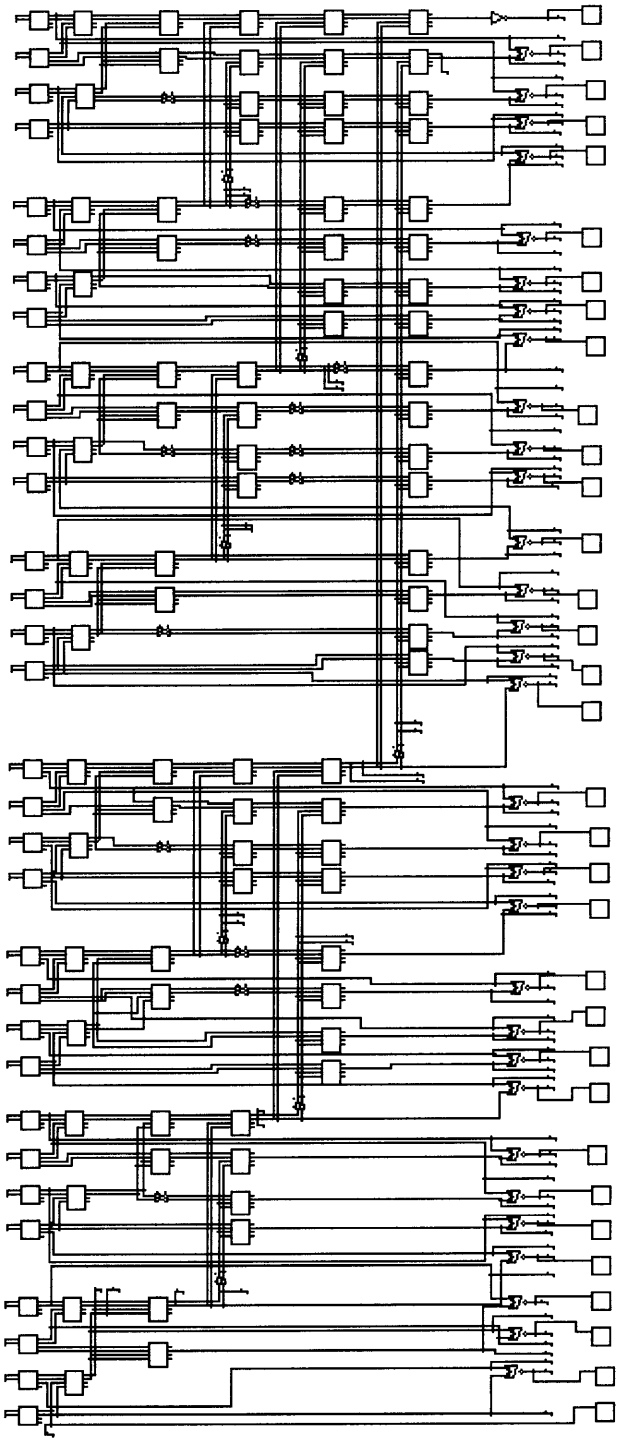

Figure 4-9: 32-Bit BLCA Adder Schematic

4.1.6 Final Results

Table 4-1: 32-Bit Adder Results

32-Bit Adder Style	Computation Time (ns)
Ripple-Carry Adder	8.5
Carry-Skip	3.5
Manchester Carry Chain	6
Manchester Carry Chain with skip	2.7
Binary Carry-Lookahead	2.1

These results are from a schematic-capture program, which was then used to generate Spice files. This program does not take all capacitances into account. Each circuit was tested in the bottom left process corner, with a Vdd of 3V and a temperature of 125 C.

4.2 Layout of the 32-Bit Adder

The organization of the adder in layout corresponds very closely to the schematic. The generate/propagate logic lines the far left, the array of XNOR's is on the far right, and the columns of the carry block are in the middle. One convenient feature of this adder is that the carry buffers fit nicely under the subsequent columns of the carry block, as can be seen in the Figure 4-10.

All of the necessary leaf cells already existed in the standard cell library, so the only thing that needed to be done was to place and route them. The 32-bit adder contains large amounts of redundancy, so it was constructed in such a way as to maximize the power of the place-and-route tool. First, the GP, BA, and BB blocks were constructed. One feature worth pointing out is the overlap of the two cells that make up BA, which is a nice feature supported by the place-and-route algorithm. Then, the carry block for a two-bit adder was constructed out of these pieces and connected to two GP blocks. As

can be seen from the schematics, the carry block for a four-bit adder is simply a pair of two-bit adders, with an additional column of BA's on the top two bits. Moreover, an eight-bit adder consists of a pair of four-bit adders, with an additional column of BB's on the top four bits, and so on. This is the method that was used to construct the GP and carry block sections of the 32-bit adder. One detail that must be attended to is the polarity of the inputs to the BA and BB blocks. If there are ever two consecutive BA or BB blocks, then the outputs of the first one must be inverted before they are input into the second one. These inverters were put wherever there was space, which was usually underneath the carry buffers. Finally, an array of 31 XNOR's and an inverter were appended to the right of the carry block, in order to generate the SUM and C31 outputs (only 31 XNOR's are required because S0 is just P0 when there is no Cin).

4.3 Additional Blocks

The other necessary blocks were constructed in a similar manner. However, the adder was the one block that involved significant research in the area of circuit design. Unlike the adder, the other blocks were based on the schematics for a previously-designed processor. The only major difference between these is transistor sizes, due to the switch from custom hand layout to standard cells.

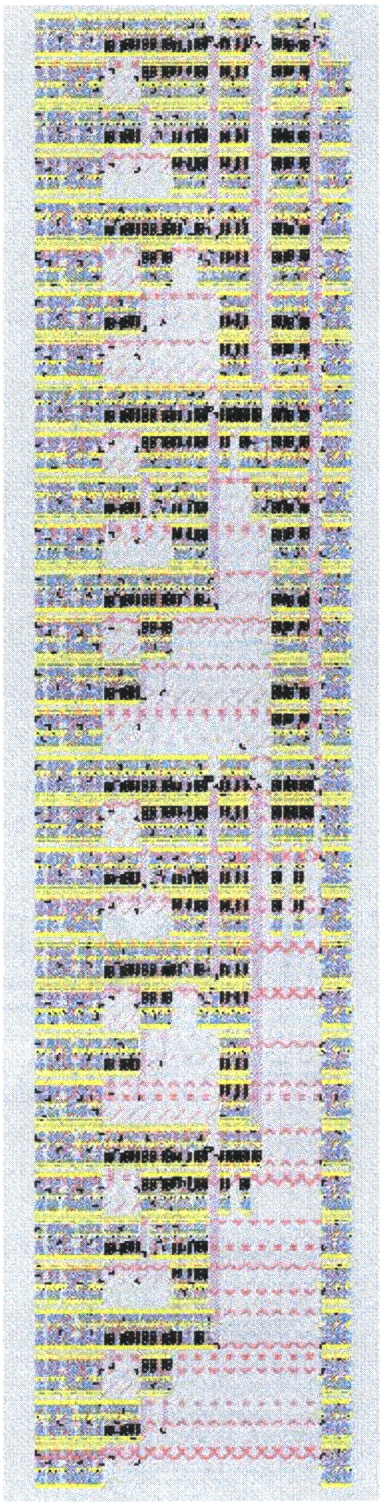

Figure 4-10: 32-Bit BLCA Adder Layout

Chapter 5

Taking the Tool for a Test Drive

Once a standard cell library and a functional tool were in place, the next step was to test them out on a substantial design. By using this new design methodology to create an example design, it was hoped that its performance could be evaluated. It was also hoped that this exercise would lead to the identification of improvements for future versions of this tool. This chapter documents the testing of this new system. The first section describes the design that was built. Section two comments on how this circuit was built. Finally, section three suggests some possible improvements that could be implemented in the future.

5.1 A 32-Bit Counter

Rather than building a complete data path, it was decided to first test out this tool on a smaller, but still substantial design. A 32-bit counter was chosen for this task. A schematic for this counter can be seen in Figure 5-1. It consists of a 32-bit 2:1 multiplexor, the previously described 32-bit adder, a 32-bit register, and some control logic. The output of the mux, which is either the value in the register or 0 depending on the output of the control logic, is input into the adder as addend A. Addend B is hard-wired to be 1. The result of this addition is then stored in the register. The control logic detects when the value 2^{31} is in the register. Thus, this counter counts up by 1 until it reaches 2^{31}, at which point it is reset to 0.

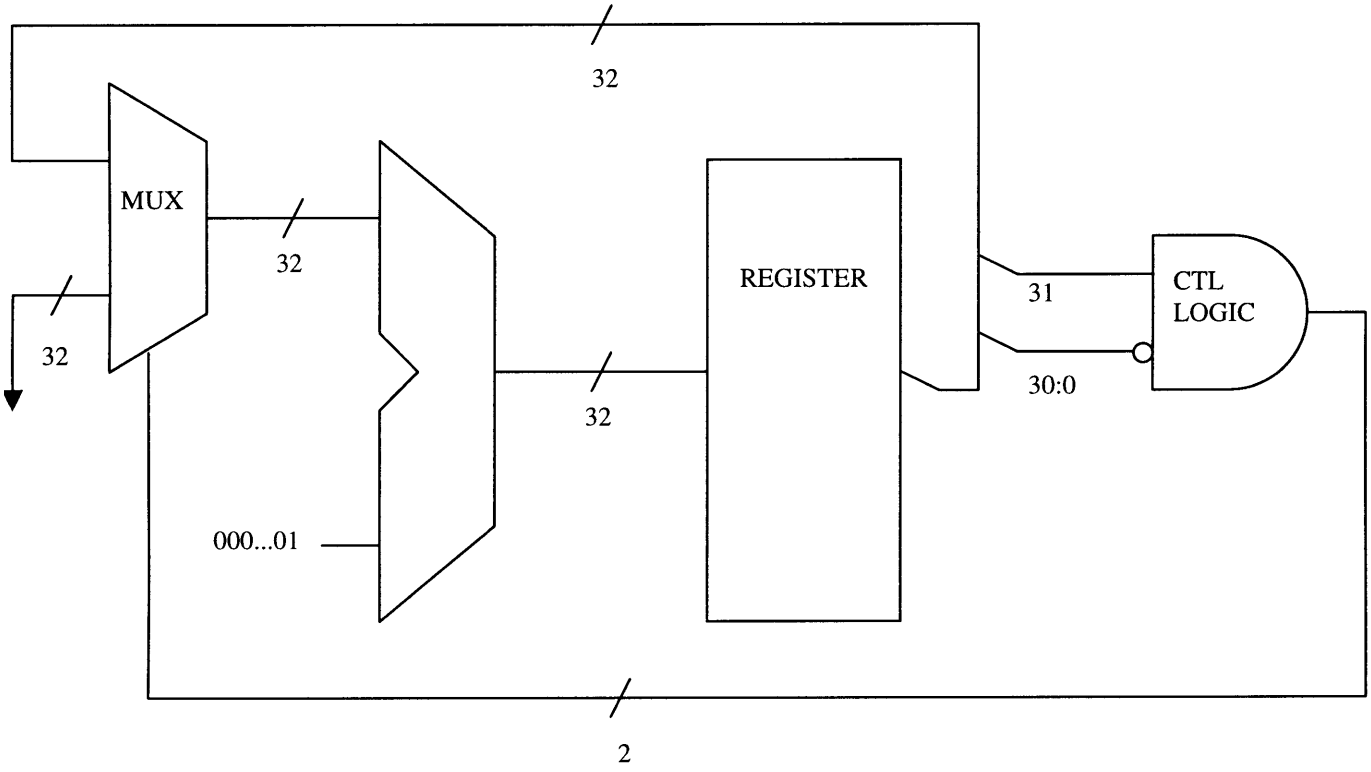

Figure 5-1: 32-Bit Counter Schematic

5.2 Building the Counter

The basic approach used in constructing the counter is the one that would be taken when constructing any data path. Each block was built from the bottom up, and then the blocks were put together. The file containing the counter design, *counter.scm*, can be found in the appendix.

5.2.1 32-bit Multiplexor

As mentioned previously, all blocks were constructed from the bottom up. Thus, a single bit-slice of a particular block was constructed out of standard cells. This bit-slice also included as much of the wiring as possible. Then, this slice was copied to construct a 32-bit unit. Including wires where possible allows them to be duplicated when the larger array is built, rather than having to route them for each slice. One of the keys to maximizing the utility of this tool is to take advantage of replication wherever possible. When building bit-slices, and doing any wiring, the basic approach taken is to have Magic and Scheme running side by side. Then, you can look at the layout you have just done and make changes, and then view them, on the fly. This is especially helpful in choosing which channels to use for routing.

This is the approach that was used to build the mux, in which all 32 bit-slices are identical. First, a single bit slice was built, tgmux_gnd. This is one of the standard cells, tgmux, with one of its inputs wired to ground. This was then duplicated, using instantiate-and-tile-vertical-array, in order to create a 32-bit mux. Finally, all of the S nodes and all of the –S nodes were wired together, by running a wire from these nodes in the topmost bit to the corresponding nodes in the bottommost bit.

5.3.2 32-bit Register

The 32-bit register also consists of 32 identical bit-slices, and it was constructed in a similar fashion. A single bit-slice was replicated with instantiate-and-tile-vertical-array to create a 32-bit register.

5.2.3 Counter Logic

The logic required for resetting the counter is basically a big AND tree. This was constructed out of a vertical array of NAND and NOR gates, which was built from the bottom up. First, a small two-level tree was built out of two NOR gates and a NAND gate. Then, this was copied twice and another NOR was added to create a three-level tree. This same process was used until eventually a five-level, 32-input tree was constructed. Since most of the inputs to this tree need to be inverted, an array of 32 inverters was tiled horizontally with this logic.

5.2.4 Putting It All Together

Once all of the individual blocks were constructed, the next step was to put them all together. The counter was laid out the same way it is drawn in the schematic, as can be seen in Figure 5-2. All of the blocks were tiled together using instantiate-and-tile. Since wires need to run across the data path and there are no free horizontal channels across the adder, the bit-slices were all spaced apart by one channel width. This was done by assigning a global variable, row-spacing, to be 1. Using this as an argument for instantiate-and-tile allows the entire design to be spaced appropriately by simply changing this variable. Next, all of the necessary connections were made. These constitute the bulk of the *counter.scm* file, as well as the bulk of work that went into building the counter.

The A inputs to the adder were tied to VDD and GND with instantiate-vertical-strap. All of the wires for even-numbered bits are nearly identical, as are all of the wires for odd-numbered bits. The difference between these two groups of bits is the direction

from them to the nearest GND rail. The only difference between nearly identical bits was the specified horizontal channel. So, it was possible to wire bit1 and bit2 and then copy the description of these wires sixteen times each. Then, the horizontal-channel-index just had to be changed for each bit to the appropriate value. Finally, bit0 had to be wired, since it is the one bit that actually needs to be connected to Vdd.

The output of the mux was wired to the other input of the adder using instantiate-wire. These wires were more time-consuming since the ending-use-ids were all different. The starting-use-ids were identical, except for the number of the mux. The only other difference was again the horizontal-channel-index. But changing these was fairly easy, since there is a pattern to them.

The output of the adder was wired to the register similar to the way the mux was wired to the adder input. This was done with instantiate-wire. The descriptions of most of these wires are also very similar to one another. In this case, they are even more similar than before. The only differences in use-ids are the numbers of the xnor in the adder and the reg_bit in the register, and of course the horizontal-channel-index. So it was possible to copy one wire 31 times, and then change the numbers on the xnor and reg_bit appropriately. Once again, there is a pattern to the horizontal-channel-index, with each subsequent wire being seven channels above the previous one. The only different wire is the first one, from C0 to the register. As described previously, this is generated in the GP block of the adder, and doesn't come from an XNOR.

Again, the output from the register to the inverter array of the logic block was done in a similar manner. This time, though, since the two contacts to be connected were adjacent, it was possible to conserve metal-2 and metal-3 channels by wiring these in

metal-1. This was done with the instantiate-single-layer-wire command. Again, the only differences are the number indexes on the starting and ending use-ids, and the horizontal-channel-indexes. These are handled the same way as before – one wire is done and then copied 32 times, and then the appropriate numbers are changed for each wire.

The output of the inverters were then wired to the NOR gates in the first level of the logic tree. For half of the inputs, the inverter is adjacent to the nor gate, so instantiate-single-layer-wire can be used as before. The other half of the inputs are in different horizontal rows, so they must be wired with instantiate-wire. Again, the first two inputs were wired, one of each type, and these were then copied. These wires were similar to those from the mux to the adder, in that the ending-use-ids were significantly different from wire to wire, and they required more attention.

The output of the register also had to be fed back to the input of the mux. This was done with instantiate-wire, and these wires were placed in the space between horizontal rows of standard cells in the design. They were done similar to all previous wires. The final wire that had to be added, was from the two outputs of the logic block to the select bits (S and –S) on the mux. These were placed at the bottom of the design.

The last thing that had to be done was the creation of the power grid. The horizontal width was set by row-spacing, to be 1, since all rows of standard cells were spaced 1 channel apart, as previously mentioned. It was decided to distribute power with one set of vertical wires of width 4, based on some rough calculations.

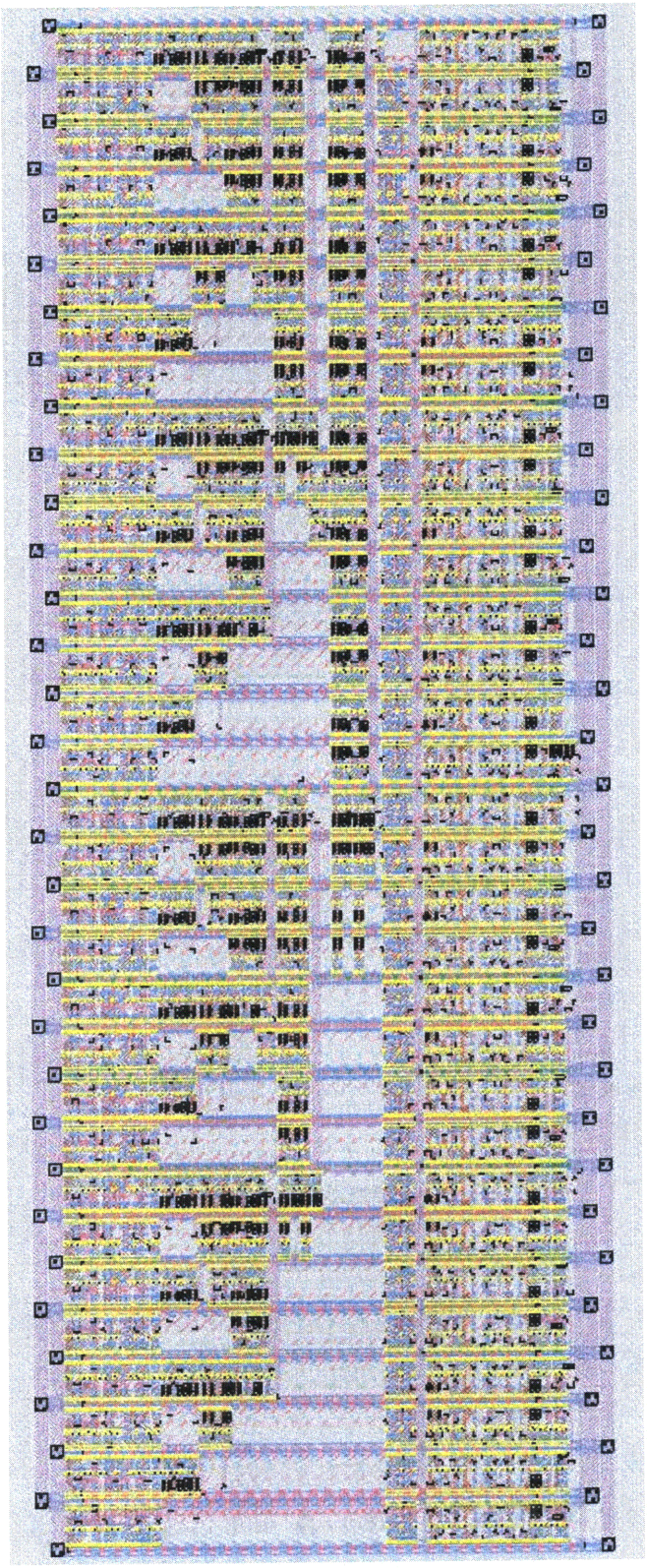

Figure 5-2: 32-Bit Counter Layout

The final part of *counter.scm* is several "make" definitions. These call the previously written procedures, which build the necessary blocks. This makes it simple to rebuild things, if changes are made. The file can simply be reloaded and these "make" procedures can be called, rather than calling each procedure individually. For example, if a change is made at a very low level, calling make-counter rebuilds the entire counter from the bottom up and the counter will therefore include this change. It takes about a minute to rerun counter.scm and rebuild the entire counter.

5.3 Possible Improvements

The process of designing and using this tool has offered valuable insight into ways in which it could be improved. Like most projects numerous decisions had to be made as to how to implement certain things. Since this was my first time working with CAD tools, many of these were made with little experience on which to base them. Looking back, I think that some things could be done differently in order to improve performance. In addition, some features could be further augmented.

5.3.1 Improving Wiring

As mentioned in chapter 4, the most time-consuming part of building the counter was wiring between the individual 32-bit blocks. This difficult task presents several challenges. One thing that slows down this wiring is the fact that the designer has to enter all of the use-ids. And as was the case with the counter, these can be different from one bit to the next. One possible way to deal with this would be to create a function that takes a given wire and copies it, based on its geometric properties. So, you could make a

wire and then copy it, but move this copy to a different horizontal channel. Then, you wouldn't have to enter all of the use-ids, but could instead just copy the wire in the appropriate places.

Another problem with the wiring is that changing the amount of space between rows of standard cells can result in wires violating design rules. This is because the designer places wires by specifying a horizontal channel. While this channel may be free for one row spacing, it may not be for another spacing. This results in the designer having to go back and change the horizontal-channel-index for wires if they don't plan out the appropriate row-spacing before they do the wiring. One way this problem might be dealt with is to add more "intelligence" to the wiring routines. It may not be necessary to specify the horizontal channel for all wires, since this may not be that important. Adding something like a "greedy" channel router to automatically route these in open channels, and routing only the more important wires explicitly might be a good alternative. This would probably save considerable time and effort at little cost.

5.3.2 Enhancing the Routing of Power and Ground

Another area in which additional work could be done is in the routing of power and ground. The current routine requires the user to calculate the power requirements of the various parts of the design and then specify the horizontal wire width and vertical wire spacing. More "intelligence" might also be added to this routine in the future, so that it could estimate the power requirements of the design and then pick the appropriate values for horizontal wire width and vertical wire spacing. This would save time and effort and help speed up the design process even more.

5.3.3 Accepting Additional Input Styles

The current tool requires the user to input their design in the Lisp-like form described in chapter 3. This form is fairly simple, but it might be nice to allow this tool to take in other forms of input as well. With the widespread use of VHDL, one possibility would be to allow it to take in VHDL descriptions and then convert them into layout. Another possibility might be that it could take in a netlist that is generated by some schematic capture tool such as ViewLogic. This might also save substantial effort, as the designer would no longer have to enter the bulk of the information by hand.

Chapter 6

Conclusion

The design, implementation, and test drive of a tool for data path construction have been described in this thesis. Most of the goals for this project were met. This tool provides an automated method for design that reduces design time over custom hand layout. At the same time, it provides design control by allowing the designer to specify placement and routing. This enables standard cells to be placed in logical groupings, resulting in readable layouts. In addition, this tool is fairly easy to use. Finally, this program runs relatively quickly, as once a design file is written it can be converted to a Magic layout in a few minutes at most. Although these goals were met, there is plenty of room for improvement. Additional functions could be added to further reduce design time, as mentioned in chapter 5.

I have learned several things from this project. As my first experience in CAD tool design, it has given me insight into how these tools work. It has also given me exposure to standard cell design, and I have learned how to make the implementation of these cells as transparent as possible to the designer at the next level. This work has also shown me the complexity involved in the tasks placement and routing, and the many different approaches to performing these tasks. This has reminded me that there are often many different ways to solve a problem, all of which may be viable solutions.

Appendix

A.1 magic.scm

```
(declare (usual-integrations))

;Useful constants
(define infinity 999999999)
(define minus-infinity (- infinity))

;Design rule parameters
(define metal1-spacing 8)
(define metal1-width 4)
(define half-metal1-width (/ metal1-width 2))
(define horizontal-metal1-channel-height metal1-spacing)
(define metal1-height 4)
(define half-metal1-height (/ metal1-height 2))
(define metal2-spacing 8)
(define vertical-channel-width metal2-spacing)
(define metal2-width 4)
(define half-metal2-width (/ metal2-width 2))
(define metal3-spacing 10)
(define horizontal-channel-height metal3-spacing)
(define metal3-width 6)
(define half-metal3-width (/ metal3-width 2))
(define metal3-height 6)
(define half-metal3-height (/ metal3-height 2))
(define m3contact-width 6)
(define half-m3contact-width (/ m3contact-width 2))
(define m3contact-height 6)
(define half-m3contact-height (/ m3contact-height 2))
(define m2contact-width 4)
(define half-m2contact-width (/ m2contact-width 2))
(define m2contact-height 4)
(define half-m2contact-height (/ m2contact-height 2))

;Object definitions
(define-structure (magic-file
                (keyword-constructor make-magic-file)
                (conc-name magic-file/)
                (print-procedure
                 (standard-unparser-method
                  'MAGIC-FILE
                  (lambda (mf port)
                    (write-char #\space port)
                    (write (magic-file/filename mf) port)))))
  (filename #f)
  (tech #f)
  (timestamp #f)
  (layers '())
  (uses '())
  (labels '()))
```

```
(define (find-use mf use-id)
  (let ((result #f))
    (let loop ((uses (magic-file/uses mf)))
      (if (not (equal? uses '()))
        (if (equal? use-id (magic-use/use-id (car uses)))
            (set! result (car uses))
            (loop (cdr uses)))
        (error "Unable to find requested use")))
    result))

(define (find-label mf label-name)
  (let ((result #f))
    (let loop ((labels (magic-file/labels mf)))
      (if (not (equal? labels '()))
        (if (equal? label-name (magic-label/label-name (car labels)))
            (set! result (car labels))
            (loop (cdr labels)))
        (error "Unable to find requested label")))
    result))


(define-structure (magic-layer
               (keyword-constructor make-magic-layer)
               (conc-name magic-layer/)
               (print-procedure
                (standard-unparser-method
                 'MAGIC-LAYER
                 (lambda (layer port)
                   (write-char #\space port)
                   (display (magic-layer/layer-name layer) port)
                   (write-char #\space port)
                   (write (magic-layer/boxes layer) port)))))
  (layer-name #f)
  (boxes '()))

(define (get-layer magic-file layer-name)
  (let loop ((layers (magic-file/layers magic-file)))
    (if (not (equal? layers '()))
      (if (eq? layer-name (magic-layer/layer-name (car layers)))
          (car layers)
          (loop (cdr layers)))
      ;didn't find it, make one
      (instantiate-layer magic-file layer-name))))

(define (instantiate-layer magic-file layer-name)
  (let ((layer (make-magic-layer 'layer-name
                         layer-name)))
    (set-magic-file/layers! magic-file
                      (cons layer
                          (magic-file/layers magic-file)))
    layer))

(define-structure (magic-box (type vector) (constructor #f)
                       (conc-name magic-box/))
  (xleft #f)
  (ybot #f)
```

```
  (xright #f)
  (ytop #f))

(define (make-magic-box x1 y1 x2 y2)
  (vector (min x1 x2) (min y1 y2) (max x1 x2) (max y1 y2)))

(define (magic-box/width box)
  (- (magic-box/xright box) (magic-box/xleft box)))

(define (magic-box/height box)
  (- (magic-box/ytop box) (magic-box/ybot box)))

(define (horizontal-center magic-box)
  (/ (+ (magic-box/xright magic-box) (magic-box/xleft magic-box)) 2))

(define (vertical-center magic-box)
  (/ (+ (magic-box/ytop magic-box) (magic-box/ybot magic-box)) 2))

(define (append-box layer box)
  (set-magic-layer/boxes!
    layer
    (cons box
        (magic-layer/boxes layer)))
   box)

(define (instantiate-box layer x1 y1 x2 y2)
  (let ((box (make-magic-box x1 y1 x2 y2)))
    (append-box layer box)))

(define-structure (magic-use (conc-name magic-use/)
                        (keyword-constructor make-magic-use)
                        (print-procedure
                         (standard-unparser-method 'MAGIC-USE
                         (lambda (use port)
                           (write-char #\space port)
                           (display (magic-use/use-id use) port)
                           (write-char #\space port)
                           (write (magic-use/box use) port)))))
  (filename #f)
  (use-id #f)
  (array #f)
  (timestamp #f)
  (transform #f)
  (box #f)
  (transform-generator #f))

(define (magic-use/width use)
  (magic-box/width (magic-use/box use)))

(define (magic-use/height use)
  (magic-box/height (magic-use/box use)))

(define (append-use magic-file use)
    (set-magic-file/uses! magic-file
                    (cons use
                        (magic-file/uses magic-file)))
    use)
```

```
(define (instantiate-use magic-file
                   use-file
                   use-id
                   transform-generator
                   xorg
                   yorg
                   #!optional array)
  (let ((new-use (make-magic-use
              'filename use-file
              'use-id use-id
              'array (if (default-object? array)
                         #f
                         array)
              'transform-generator transform-generator
              ))
      (subcell-mf (read-magic-file use-file)))

                              ;Set bounding box
    (set-magic-use/box! new-use (magic-file-bounding-box subcell-mf))

                              ;Set transform
    (set-magic-use/transform! new-use
                        (compose-magic-transforms
                         (transform-generator new-use)
                         (magic-translate
                        (- xorg (magic-box/xleft
                                (magic-use/box new-use)))
                        (- yorg (magic-box/ybot
                                (magic-use/box new-use))))))
    (append-use magic-file new-use)

    ))

(define-structure (magic-label (conc-name magic-label/))
  (layer #f)
  (box #f)
  (position #f)
  (label-name #f))

(define (instantiate-label magic-file layer-name box position label-
name)
  (let ((label (make-magic-label layer-name box position label-name)))
    (set-magic-file/labels! magic-file
                      (cons label
                          (magic-file/labels magic-file)))
    label))



(define-structure (magic-use-array (conc-name magic-use-array/))
  (xlo #f)
  (xhi #f)
  (xsep #f)
  (ylo #f)
  (yhi #f)
  (ysep #f))
```

```
(define-structure (magic-transform (conc-name magic-transform/))
  (a #f)
  (b #f)
  (c #f)
  (d #f)
  (e #f)
  (f #f))

(define magic-label-position-names
  '#(CENTER NORTH NORTHEAST EAST SOUTHEAST SOUTH SOUTHWEST WEST
NORTHWEST))

(define (read-magic-file filename)
  (let ((pathname (pathname-new-type (merge-pathnames filename)
"mag")))
    (call-with-input-file pathname
      (lambda (port)
      (let ((magic-file (make-magic-file
                  'filename (->namestring
                            (if (equal? "mag"
                                    (pathname-type filename))
                              (pathname-new-type filename #f)
                              filename)))))

        (define (parse-tech line)
          (parse-optional line "tech " parse-timestamp
                      (lambda (line)
                        (set-magic-file/tech! magic-file
                                              (string-tail line 5)))))

        (define (parse-timestamp line)
          (parse-optional line "timestamp " parse-body
                      (lambda (line)
                        (set-magic-file/timestamp!
                         magic-file
                         (parse-time (string-tail line 10))))))
        (define (parse-body line)
          (cond ((string=? "<< end >>" line)
                 (done))
                ((string=? "<< labels >>" line)
                 (parse-label (get-line)))
                ((and (string-prefix? "<< " line)
                      (string-suffix? " >>" line))
                 (parse-layer line))
                ((string-prefix? "use " line)
                 (parse-use line))
                (else
                 (syntax-error line "unknown line type")))
        (define (parse-layer line)
          (let ((layer-name
                 (intern (substring line 3 (fix:- (string-length line)
3)))))
            (let loop ()
            (let ((line (get-line)))
              (if (string-prefix? "rect " line)
                  (begin
                  (apply
```

```
                instantiate-box
                (get-layer magic-file layer-name)
                (map parse-integer
                     (burst-line line 4)))
               (loop))
               (parse-body line))))))

        (define (parse-use line)
          (let ((use (make-magic-use)))

            (define (parse-array line)
            (parse-optional line "array " parse-timestamp
                        (lambda (line)
                          (set-magic-use/array!
                           use
                           (apply make-magic-use-array
                                (map parse-integer
                                     (burst-line line 6)))))))

            (define (parse-timestamp line)
            (parse-optional line "timestamp " parse-transform
                        (lambda (line)
                          (set-magic-use/timestamp!
                           use
                           (parse-time (string-tail line 10))))))

            (define (parse-transform line)
            (parse-required line "transform " parse-box
                        (lambda (line)
                          (set-magic-use/transform!
                           use
                           (apply make-magic-transform
                                (map parse-integer
                                     (burst-line line 6)))))))

            (define (parse-box line)
            (parse-required line "box " continue
                        (lambda (line)
                          (set-magic-use/box!
                           use
                           (apply make-magic-box
                                (map parse-integer
                                     (burst-line line 4)))))))

            (define (continue line)
            (append-use
             magic-file
             use)
            (parse-body line))

            (let ((strings (burst-line line #f)))
            (if (not (and (pair? strings)
                        (or (null? (cdr strings))
                          (and (pair? (cdr strings))
                               (null? (cddr strings))))))
                (syntax-error line "wrong number of arguments on
line"))
```

```
    (set-magic-use/filename! use (car strings))
    (set-magic-use/use-id!
     use
     (if (null? (cdr strings))
         #f
         (cadr strings))))
    (parse-array (get-line))))

(define (parse-label line)
  (let loop ((line line))
    (if (string-prefix? "rlabel " line)
      (let ((strings (burst-line line 7)))
        (instantiate-label
         magic-file
         (intern (car strings))
         (make-magic-box
          (parse-integer (cadr strings))
          (parse-integer (caddr strings))
          (parse-integer (cadddr strings))
          (parse-integer (car (cddddr strings))))
         (translate-position
          (parse-integer (cadr (cddddr strings))))
         (caddr (cddddr strings)))
        (loop (get-line)))
      (parse-body line))))

(define (translate-position position)
  (if (not (< -1 position
           (vector-length magic-label-position-names)))
    (syntax-error position "illegal label position"))
  (vector-ref magic-label-position-names position))

(define (done)
  magic-file)

(define read-line
  (let ((eol (char-set #\newline)))
    (lambda (port)
    (let ((return (read-string eol port)))
      (read-char port)
      return)
    )))

(define (get-line)
  (let ((line (read-line port)))
    (if (eof-object? line)
      (syntax-error line "premature end-of-file"))
    line))

(define (syntax-error line message)
  (error (string-append "Syntax error in magic file ("
                  message
                  "):")
       line))

(define (parse-optional line prefix next-state parser)
  (next-state
```

```
            (if (string-prefix? prefix line)
              (begin
                (parser line)
                (get-line))
              line)))

        (define (parse-required line prefix next-state parser)
          (if (not (string-prefix? prefix line))
            (syntax-error line (string-append "expected " prefix)))
          (parser line)
          (next-state (get-line)))

        (define (burst-line line n)
          (let ((end (string-length line)))
            (let loop
              ((start (fix:+ (string-find-next-char line #\space) 1))
               (segments '()))
              (let ((space
                     (substring-find-next-char line start end #\space)))
                (if space
                    (loop (fix:+ space 1)
                        (cons (substring line start space) segments))
                    (let ((segments
                         (reverse!
                          (cons (substring line start end) segments))))
                    (if (and n (not (fix:= n (length segments))))
                        (syntax-error line "incorrect burst number"))
                    segments))))))

        (define (parse-time string)
          (+ epoch (parse-integer string)))

        (define (parse-integer string)
          (let ((n (string->number string)))
            (if (not (exact-integer? n))
              (syntax-error string "not an integer"))
            n))

        (let ((line (get-line)))
          (if (not (equal? "magic" line))
            (syntax-error line "not a magic file")))
        (parse-tech (get-line))))))

(define (write-magic-file mf)
  ;; Note: this code assumes that the caller will provide an
  ;; appropriate "checkpaint" layer if needed.
  (define (write-box box port)
         (write (magic-box/xleft box) port)
         (write-char #\space port)
         (write (magic-box/ybot box) port)
         (write-char #\space port)
         (write (magic-box/xright box) port)
         (write-char #\space port)
         (write (magic-box/ytop box) port))
  (define (translate-position position)
         (let ((end (vector-length magic-label-position-names)))
           (let loop ((i 0))
```

```
            (if (fix:= i end)
              (error "Illegal label position:" position))
            (if (eq? position (vector-ref magic-label-position-names
i))
              i
              (loop (fix:+ i 1))))))

  (let ((pathname (pathname-new-type (merge-pathnames (magic-
file/filename mf)) "mag")))
    (call-with-output-file pathname
      (lambda (port)
      (write-string "magic" port)
      (newline port)
      (if (magic-file/tech mf)
          (begin
            (write-string "tech " port)
            (write-string (magic-file/tech mf) port)
            (newline port)))
      (if (magic-file/timestamp mf)
          (begin
            (write-string "timestamp " port)
            (write (- (magic-file/timestamp mf) epoch) port)
            (newline port)))
      (for-each (lambda (layer)
                (if (not (null? (magic-layer/boxes layer)))
                  (begin
                    (write-string "<< " port)
                    (write (magic-layer/layer-name layer) port)
                    (write-string " >>" port)
                    (newline port)
                    (for-each (lambda (box)
                                (write-string "rect " port)
                                (write-box box port)
                                (newline port))
                              (magic-layer/boxes layer)))))
              (magic-file/layers mf))
      (for-each (lambda (use)
                (write-string "use " port)
                (write-string
                 (->namestring
                  (let ((filename (magic-use/filename use)))
                  (if (pathname=? (directory-pathname "")
                                  (directory-pathname filename))
                      (file-pathname filename)
                      (merge-pathnames
                       (file-pathname filename)
                       (relative-pathname (directory-pathname filename)
                                          pathname)))))
                 port)
                (if (magic-use/use-id use)
                  (begin
                    (write-char #\space port)
                    (write-string (magic-use/use-id use) port)))
                (newline port)
                (let ((array (magic-use/array use)))
                  (if array
                    (begin
```

```
                (write-string "array " port)
                (write (magic-use-array/xlo array) port)
                (write-char #\space port)
                (write (magic-use-array/xhi array) port)
                (write-char #\space port)
                (write (magic-use-array/xsep array) port)
                (write-char #\space port)
                (write (magic-use-array/ylo array) port)
                (write-char #\space port)
                (write (magic-use-array/yhi array) port)
                (write-char #\space port)
                (write (magic-use-array/ysep array) port)
                (newline port))))
          (if (magic-use/timestamp use)
            (begin
              (write-string "timestamp " port)
              (write (- (magic-use/timestamp use) epoch) port)
              (newline port)))
          (let ((t (magic-use/transform use)))
            (write-string "transform " port)
            (write (magic-transform/a t) port)
            (write-char #\space port)
            (write (magic-transform/b t) port)
            (write-char #\space port)
            (write (magic-transform/c t) port)
            (write-char #\space port)
            (write (magic-transform/d t) port)
            (write-char #\space port)
            (write (magic-transform/e t) port)
            (write-char #\space port)
            (write (magic-transform/f t) port)
            (newline port))
          (write-string "box " port)
          (write-box (magic-use/box use) port)
          (newline port))
        (magic-file/uses mf))
(if (not (null? (magic-file/labels mf)))
    (begin
      (write-string "<< labels >>" port)
      (newline port)
      (for-each (lambda (label)
                (write-string "rlabel " port)
                (write (magic-label/layer label) port)
                (write-char #\space port)
                (write-box (magic-label/box label) port)
                (write-char #\space port)
                (write (translate-position
                       (magic-label/position label))
                     port)
                (write-char #\space port)
                (write-string (magic-label/label-name label) port)
                (newline port))
              (magic-file/labels mf))))
(write-string "<< end >>" port)
(newline port)))))
```

```
(define (transform-magic-label t label)
  (make-magic-label (magic-label/layer label)
                (transform-magic-box t (magic-label/box label))
                (magic-label/position label)
                (magic-label/label-name label)))

(define (transform-magic-box t box)
  (call-with-values
      (lambda ()
      (transform-magic-coordinates t
                              (magic-box/xleft box)
                              (magic-box/ybot box)))
    (lambda (x1 y1)
      (call-with-values
        (lambda ()
          (transform-magic-coordinates t
                                (magic-box/xright box)
                                (magic-box/ytop box)))
      (lambda (x2 y2)
        (make-magic-box (min x1 x2) (min y1 y2) (max x1 x2) (max y1
y2)))))))

(define (transform-magic-coordinates t x y)
  (values (+ (* x (magic-transform/a t))
             (* y (magic-transform/b t))
             (magic-transform/c t))
          (+ (* x (magic-transform/d t))
             (* y (magic-transform/e t))
             (magic-transform/f t))))

(define (compose-magic-transforms . transforms)
  (reduce
   (lambda (f g)
     ;; Create a transform that is equivalent to (LAMBDA (X) (G (F
X))).
     (make-magic-transform (+ (* (magic-transform/a f) (magic-
transform/a g))
                              (* (magic-transform/d f) (magic-transform/b
g)))
                           (+ (* (magic-transform/b f) (magic-transform/a g))
                              (* (magic-transform/e f) (magic-transform/b
g)))
                           (+ (* (magic-transform/c f) (magic-transform/a g))
                              (* (magic-transform/f f) (magic-transform/b g))
                              (magic-transform/c g))
                           (+ (* (magic-transform/a f) (magic-transform/d g))
                              (* (magic-transform/d f) (magic-transform/e
g)))
                           (+ (* (magic-transform/b f) (magic-transform/d g))
                              (* (magic-transform/e f) (magic-transform/e
g)))
                           (+ (* (magic-transform/c f) (magic-transform/d g))
                              (* (magic-transform/f f) (magic-transform/e g))
                              (magic-transform/f g))))
   magic-rotate-0
   transforms))
(define magic-rotate-0   (make-magic-transform  1  0  0  0  1  0))
```

```
(define magic-rotate-90  (make-magic-transform  0  1  0 -1  0  0))
(define magic-rotate-180 (make-magic-transform -1  0  0  0 -1  0))
(define magic-rotate-270 (make-magic-transform  0 -1  0  1  0  0))
(define magic-reflect-x  (make-magic-transform -1  0  0  0  1  0))
(define magic-reflect-y  (make-magic-transform  1  0  0  0 -1  0))

(define (magic-translate dx dy) (make-magic-transform 1 0 dx 0 1 dy))

(define (generate-identity-transform #!optional new-use)
  magic-rotate-0)

(define identity generate-identity-transform)

(define (generate-sideways-transform new-use)
  (compose-magic-transforms
   magic-reflect-x
   (magic-translate (+ (* 2 (magic-box/xleft (magic-use/box new-use)))
                       (magic-use/width new-use))
                    0)))

(define sideways generate-sideways-transform)

(define (generate-upsidedown-transform new-use)
  (compose-magic-transforms
   magic-reflect-y
   (magic-translate 0 (+ (* 2 (magic-box/ybot (magic-use/box new-use)))
                         (magic-use/height new-use)))))

(define upsidedown generate-upsidedown-transform)


(define (generate-upsideways-transform new-use)
  (compose-magic-transforms
   magic-rotate-180
   (magic-translate (+ (* 2 (magic-box/xleft (magic-use/box new-use)))
                       (magic-use/width new-use))
                    (+ (* 2 (magic-box/ybot (magic-use/box new-use)))
                       (magic-use/height new-use)))))

(define upsideways generate-upsideways-transform)


(define (magic-file-bounding-box mf)
  (let ((xleft #f)
        (ybot #f)
        (xright #f)
        (ytop #f))
    (let ((do-box
           (lambda (box)
             (if (not xleft)
               (begin
                 (set! xleft (magic-box/xleft box))
                 (set! ybot (magic-box/ybot box))
                 (set! xright (magic-box/xright box))
                 (set! ytop (magic-box/ytop box)))
               (begin
                 (if (fix:< (magic-box/xleft box) xleft)
```

```
                    (set! xleft (magic-box/xleft box)))
                (if (fix:< (magic-box/ybot box) ybot)
                    (set! ybot (magic-box/ybot box)))
                (if (fix:> (magic-box/xright box) xright)
                    (set! xright (magic-box/xright box)))
                (if (fix:> (magic-box/ytop box) ytop)
                    (set! ytop (magic-box/ytop box))))))))
      (for-each (lambda (layer)
              (for-each (lambda (box)
                        (do-box box))
                      (magic-layer/boxes layer)))
            (magic-file/layers mf))
      (for-each (lambda (use)
              (do-box (transform-magic-box (magic-use/transform use)
                                           (magic-use/box use))))
            (magic-file/uses mf))
      (for-each (lambda (label)
              (do-box (magic-label/box label)))
            (magic-file/labels mf)))
    (make-magic-box xleft ybot xright ytop)))

(define (instantiate-and-tile-horizontally magic-file
                                   uses-to-instantiate
                                   instance-names
                                   transform-generators
                                   xorg
                                   xoffset
                                   yorg)
  (let ((new-instance)
      (xpos xorg))
    (for-each (lambda (use-to-instantiate instance-name transform-
generator)
            (set! new-instance (instantiate-use
                            magic-file
                            use-to-instantiate
                            instance-name
                            transform-generator
                            xpos
                            yorg))
            (set! xpos (+ xpos xoffset (magic-use/width new-instance)))

            )
            uses-to-instantiate
            instance-names
            transform-generators)))


(define (instantiate-and-tile      magic-file
                           uses-to-instantiate
                           instance-names
                           tile-direction ; horizontal or vertical
                           transform-generators
                           xorg
                           yorg
                           xoffsets
                           yoffsets)
  (let ((new-instance)
```

```
      (xpos  xorg)
        (ypos yorg))

    (cond ((equal? tile-direction 'horizontal)
         (for-each (lambda (use-to-instantiate instance-name transform-
generator xoffset yoffset)

                   (set! new-instance (instantiate-use
                                magic-file
                                use-to-instantiate
                                instance-name
                                transform-generator
                                xpos
                                ypos))
                   (set! xpos (+ xoffset xpos (magic-use/width new-
instance)))
                   (set! ypos (+ ypos yoffset))
                   )
                  uses-to-instantiate
                  instance-names
                  transform-generators
                  xoffsets
                  yoffsets))

        ((equal? tile-direction 'vertical)
         (for-each (lambda (use-to-instantiate instance-name transform-
generator xoffset yoffset)
                   (set! new-instance (instantiate-use
                                magic-file
                                use-to-instantiate
                                instance-name
                                transform-generator
                                xpos
                                ypos))
                   (set! ypos (+ yoffset ypos (magic-use/height new-
instance)))
                   (set! xpos (+ xpos xoffset))
                   )
                  uses-to-instantiate
                  instance-names
                  transform-generators
                  xoffsets
                  yoffsets))
        (else error "Invalid tile direction"))))


; creates a vertical array made up of the specified number of
; copies of a given cell

(define (instantiate-and-tile-vertical-array  magic-file
                                    use-to-instantiate
                                    number-of-copies
                                    xorg
                                    yorg
                                    xoffset
                                    yoffset)
```

```
  (define (rename->vector name size)
    (make-initialized-vector
     size
     (lambda (i)
       (string-append name "-" (number->string i)))))


  (let ((new-instance)
      (xpos  xorg)
        (ypos yorg)
      (uses-to-instantiate (make-list number-of-copies use-to-
instantiate))
      (instance-names
       (vector->list
        (rename->vector use-to-instantiate number-of-copies)))

      (transform-generators (apply append
                             (make-initialized-list
                              (/ number-of-copies 2)
                              (lambda (i) (list identity
upsidedown)))))

      (xoffsets (make-list number-of-copies xoffset))
      (yoffsets (make-list number-of-copies yoffset)))


         (for-each (lambda (use-to-instantiate instance-name transform-
generator xoffset yoffset)
                     (set! new-instance (instantiate-use
                                         magic-file
                                         use-to-instantiate
                                         instance-name
                                         transform-generator
                                         xpos
                                         ypos))
                     (set! ypos (+ yoffset ypos (magic-use/height new-
instance)))
                     (set! xpos (+ xpos xoffset))
                     )
                   uses-to-instantiate
                   instance-names
                   transform-generators
                   xoffsets
                   yoffsets)))


;Wiring stuff

;Given a vertical wiring channel index (0-<nchans-1>), returns an
infinitely
;tall box that covers that channel. This is assumed to be taking place
in
;the coordinates of the cell under construction, the origin of whose
;channels is assumed to be (0,0).

(define (vertical-channel-box channel-index)
  (let ((xleft (* vertical-channel-width channel-index)))
```

```
    (make-magic-box xleft
                    minus-infinity
                    (+ xleft vertical-channel-width)
                    infinity)))



;Given a horizontal wiring channel index (0-<nchans-1>), returns an
;infinitely wide box that covers that channel. This is assumed to be
;taking place in the coordinates of the cell under construction,
;the origin of whose channels is assumed to be (0,0).

(define (horizontal-channel-box channel-index)
  (let ((ybot  (* horizontal-channel-height channel-index)))
    (make-magic-box minus-infinity
                    ybot
                    infinity
                    (+ ybot horizontal-channel-height))))

(define (intersect-channel-boxes vertical-box horizontal-box)
  (make-magic-box (magic-box/xleft vertical-box)
                  (magic-box/ybot horizontal-box)
                  (magic-box/xright vertical-box)
                  (magic-box/ytop horizontal-box)))

(define (channel-intersection-box vertical-channel-index
                                  horizontal-channel-index)
  (let ((vertical-channel-box
         (vertical-channel-box vertical-channel-index))
        (horizontal-channel-box
         (horizontal-channel-box horizontal-channel-index)))
    (intersect-channel-boxes vertical-channel-box horizontal-channel-
box)))

(define (point->channel-indices x y)
  (cons (quotient
         x
         vertical-channel-width)
        (quotient
         y
         horizontal-channel-height)))

(define (channel-indices/vertical-channel-index channel-indices)
  (car channel-indices))

(define (channel-indices/horizontal-channel-index channel-indices)
  (cdr channel-indices))

#|This routine creates a vertical strap on the specified layer in the
specified
vertical channel. It is assumed that the starting-horizontal-channel is
below the ending-horizontal-channel. The strap will extend from the
bottom edge
of the starting-horizontal-channel to the top edge of the
ending-horizontal-channel.|#
```

```
(define (instantiate-vertical-strap
       mf layer width
       vertical-channel-index
       starting-horizontal-channel-index
       horizontal-channel-index xoffset yoffset yoffsettop)

  ;These boxes are in the coordinate system of the top-level cell
  (let ((vertical-channel-box
       (vertical-channel-box vertical-channel-index))
      (starting-horizontal-channel-box
       (horizontal-channel-box starting-horizontal-channel-index))
      (ending-horizontal-channel-box
       (horizontal-channel-box horizontal-channel-index))
      (half-width (/ width 2)))
    (let ((starting-box (intersect-channel-boxes
                  vertical-channel-box
                  starting-horizontal-channel-box))
        (ending-box (intersect-channel-boxes
                  vertical-channel-box
                  ending-horizontal-channel-box)))
      (append-box (get-layer mf 'metal2)
              (make-magic-box
               (- (horizontal-center starting-box)
                  half-width (- xoffset))
               (- (magic-box/ybot starting-box) yoffset)
               (+ (horizontal-center ending-box)
                  half-width xoffset)
               (+ (magic-box/ytop ending-box) yoffset yoffsettop)))

;add contacts
      (append-box (get-layer mf 'm2contact)
              (make-magic-box
               (- (horizontal-center starting-box)
                  half-width (- xoffset))
               (- (magic-box/ybot starting-box) yoffset half-width)
               (+ (horizontal-center ending-box)
                  half-width xoffset)
               (+ (- (magic-box/ybot starting-box) yoffset) half-
width)))

      (append-box (get-layer mf 'm2contact)
              (make-magic-box
               (- (horizontal-center starting-box)
                  half-width (- xoffset))
               (- (+ (magic-box/ytop ending-box) yoffset yoffsettop)
half-width)
               (+ (horizontal-center ending-box)
                  half-width xoffset)
               (+ (magic-box/ytop ending-box) yoffset yoffsettop half-
width)))
)))
```

```
#|This routine creates a horizontal strap on the specified layer in the
specified horizontal channel. It is assumed that the starting-vertical-
channel is left of the ending-vertical-channel. The strap will extend
from the left edge of the starting-vertical-channel to the right edge
of the ending-vertical-channel.|#

(define (instantiate-horizontal-strap mf layer width starting-vertical-
channel-index
                              ending-vertical-channel-index
                              horizontal-channel-index
                              yoffset)
  (let ((horizontal-channel-box (horizontal-channel-box
                          horizontal-channel-index))
      (starting-channel-box (vertical-channel-box
                          starting-vertical-channel-index))
      (ending-channel-box (vertical-channel-box
                        ending-vertical-channel-index))
      (half-width (/ width 2)))
    (append-box
     (get-layer mf layer)
     (make-magic-box (magic-box/xleft starting-channel-box)
                 (+ yoffset
                  (- (vertical-center horizontal-channel-box)
                     half-width))
                 (magic-box/xright ending-channel-box)
                 (+ (vertical-center horizontal-channel-box)
                  half-width yoffset)))))

(define (instantiate-channel-intersection-label
       mf
       layer
       vertical-channel-index
       horizontal-channel-index
       height
       width
       position
       text)
  (let ((channel-intersection-box
       (channel-intersection-box vertical-channel-index horizontal-
channel-index))
      )
    (instantiate-label mf
                  layer
                  (make-magic-box (- (horizontal-center
                                      channel-intersection-box)
                                     (/ width 2))
                                  (- (vertical-center
                                      channel-intersection-box)
                                     (/ height 2))
                                  (+ (horizontal-center
                                      channel-intersection-box)
                                     (/ width 2))
                                  (+ (vertical-center
                                      channel-intersection-box)
                                     (/ height 2)))
                  position
                  text)))
```

```
(define (instantiate-contact mf layer height width vertical-channel-
index horizontal-channel-index xoffset yoffset)
  (let ((contact-box (channel-intersection-box vertical-channel-index
horizontal-channel-index))
      (half-height (/ height 2))
      (half-width (/ width 2)))
    (append-box (get-layer mf layer)
            (make-magic-box
             (- (+ xoffset (horizontal-center contact-box))
                half-width)
             (- (+ yoffset (vertical-center contact-box))
                half-height)
             (+ (+ xoffset (horizontal-center contact-box))
                half-width)
             (+ (+ yoffset (vertical-center contact-box))
                half-height)))))

;Creates a vertical wire in metal2
(define (instantiate-vertical-wire-segment
       mf
       vertical-channel-index
       starting-horizontal-channel-index
       horizontal-channel-index)

  ;These boxes are in the coordinate system of the top-level cell
  (let ((vertical-channel-box
       (vertical-channel-box vertical-channel-index))
      (starting-horizontal-channel-box
       (horizontal-channel-box starting-horizontal-channel-index))
      (ending-horizontal-channel-box
       (horizontal-channel-box horizontal-channel-index))
      )
    (let ((starting-box (intersect-channel-boxes
                  vertical-channel-box
                  starting-horizontal-channel-box))
        (ending-box (intersect-channel-boxes
                  vertical-channel-box
                  ending-horizontal-channel-box)))
      (append-box (get-layer mf 'metal2)
              (make-magic-box
                   (- (horizontal-center starting-box)
                    half-metal2-width)
                   (vertical-center starting-box)
                   (+ (horizontal-center ending-box)
                    half-metal2-width)
                   (vertical-center ending-box))))))

;Creates a horizontal wire in metal3, optionally with metal3 contacts
;at each end (the default is to supply the contacts if the optional
;arguments are omitted)

(define (instantiate-horizontal-wire-segment
       mf
       starting-vertical-channel-index
       ending-vertical-channel-index
       horizontal-channel-index #!optional starting-contact? ending-
contact?)
```

```
  ;These boxes are in the coordinate system of the top-level cell
  (let ((starting-vertical-channel-box
       (vertical-channel-box starting-vertical-channel-index))
      (ending-vertical-channel-box
       (vertical-channel-box ending-vertical-channel-index))
      (horizontal-channel-box
        (horizontal-channel-box horizontal-channel-index))
      )

    (let ((starting-box (intersect-channel-boxes
                  starting-vertical-channel-box
                  horizontal-channel-box))
        (ending-box (intersect-channel-boxes
                  ending-vertical-channel-box
                  horizontal-channel-box)))

      (let ((starting-box-horizontal-center (horizontal-center
starting-box))
          (starting-box-vertical-center (vertical-center starting-box))
          (ending-box-horizontal-center (horizontal-center ending-box))
          (layer)
          (half-height))
      (cond ((and (= (abs (- (magic-box/xleft starting-box) (magic-
box/xleft ending-box)))
                  vertical-channel-width)
              (or (default-object? starting-contact?) starting-
contact?)
              (or (default-object? ending-contact?) ending-contact?)
;The wire spans adjacent vertical channels, and both contacts have
;been requested.  To avoid a bogus design rule error in Magic,
;just make one big metal3 contact, rather than two contacts and a short
;wire.  So, substitute this contact fragment for the wire

           (set! layer 'm3contact)
           (set! half-height half-m3contact-height))
          (else
           (set! layer 'metal3)
           (set! half-height half-metal3-height)))

      (append-box (get-layer mf layer)
                  (make-magic-box
                   starting-box-horizontal-center
                   (- starting-box-vertical-center
                      half-height)
                   ending-box-horizontal-center
                   (+ starting-box-vertical-center
                      half-height)))

      ;Now do the contacts at the ends of the wire
      (if (or (default-object? starting-contact?) starting-contact?)
        (instantiate-contact mf 'm3contact m3contact-height m3contact-
width starting-vertical-channel-index
                          horizontal-channel-index 0 0))
      (if (or (default-object? ending-contact?) ending-contact?)
        (instantiate-contact mf 'm3contact m3contact-height m3contact-
width ending-vertical-channel-index
```

```
                        horizontal-channel-index 0 0))

      ))))


;Runs a wire from one label to another, using a vertical run from a
;starting label, a horizontal run
;in the specified horizontal channel, and then a
;vertical run to the ending label

(define (instantiate-wire mf starting-use-ids starting-label-name
                    ending-use-ids ending-label-name
                    horizontal-channel-index)
  (define (label->box mf use-ids label-name)
    (let ((transforms (list (identity)))
          (use))
      (for-each (lambda (use-id)
              (set! use (find-use mf use-id))
              (set! transforms
                  (cons (magic-use/transform use) transforms))
              (set! mf (read-magic-file (magic-use/filename use)))
              )
            use-ids)
      (transform-magic-box
       (apply compose-magic-transforms transforms)
       (magic-label/box (find-label mf label-name)))))

  (if (not (or (pair? starting-use-ids) (null? starting-use-ids)))
      (set! starting-use-ids (list starting-use-ids)))

  (if (not (or (pair? ending-use-ids) (null? ending-use-ids)))
      (set! ending-use-ids (list ending-use-ids)))

 (let ((starting-label-box (label->box mf starting-use-ids
                              starting-label-name))
      (ending-label-box (label->box mf ending-use-ids
                              ending-label-name)))

    (let ((starting-channel-indices
         (point->channel-indices
          (magic-box/xleft
           starting-label-box)
          (magic-box/ybot
           starting-label-box)))
        (ending-channel-indices
         (point->channel-indices
          (magic-box/xleft
           ending-label-box)
          (magic-box/ybot
           ending-label-box)))
        )
      ;Hokay, we're finally ready to make the wire
      (cond ((eq? (channel-indices/vertical-channel-index starting-
channel-indices)
                  (channel-indices/vertical-channel-index ending-
channel-indices))
                              ;the wire only has a vertical segment
```

```
            (instantiate-vertical-wire-segment
             mf
             (channel-indices/vertical-channel-index
              starting-channel-indices)
             (channel-indices/horizontal-channel-index
              starting-channel-indices)
             (channel-indices/horizontal-channel-index
              ending-channel-indices))
            )
           (else
                               ;Do the starting vertical wire
            (instantiate-vertical-wire-segment
             mf
             (channel-indices/vertical-channel-index
              starting-channel-indices)
             (channel-indices/horizontal-channel-index
              starting-channel-indices)
             horizontal-channel-index)
                               ;and while we're at it, do the ending
vertical wire
            (instantiate-vertical-wire-segment
             mf
             (channel-indices/vertical-channel-index
              ending-channel-indices)
             (channel-indices/horizontal-channel-index
              ending-channel-indices)
             horizontal-channel-index)
                               ;and now the horizontal wire

            (instantiate-horizontal-wire-segment
             mf
             (channel-indices/vertical-channel-index
              starting-channel-indices)
             (channel-indices/vertical-channel-index
              ending-channel-indices)
             horizontal-channel-index
             )))
      )))


;Runs a wire from one use to another, using a vertical run from a
;starting label, a horizontal run in the specified horizontal channel,
;a vertical run in the specified vertical channel, a horizontal run
;in the other specified horizontal channel, and then a vertical run
;to the ending label


(define (instantiate-five-segment-wire mf starting-use-ids starting-
label-name
                    ending-use-ids ending-label-name
                    starting-horizontal-channel-index
                    ending-horizontal-channel-index
                    middle-vertical-channel-index)
  (define (label->box mf use-ids label-name)
    (let ((transforms '())
          (use))
      (for-each (lambda (use-id)
```

```
              (set! use (find-use mf use-id))
              (set! transforms
                  (cons (magic-use/transform use) transforms))
              (set! mf (read-magic-file (magic-use/filename use)))
              )
            use-ids)
       (transform-magic-box
        (apply compose-magic-transforms transforms)
        (magic-label/box (find-label mf label-name)))))

  (if (not (pair? starting-use-ids))
      (set! starting-use-ids (list starting-use-ids)))

  (if (not (pair? ending-use-ids))
      (set! ending-use-ids (list ending-use-ids)))

  (let ((starting-label-box (label->box mf starting-use-ids
                                starting-label-name))
      (ending-label-box (label->box mf ending-use-ids
                                ending-label-name)))

    (let ((starting-channel-indices
         (point->channel-indices
          (magic-box/xleft
           starting-label-box)
          (magic-box/ybot
           starting-label-box)))
        (ending-channel-indices
         (point->channel-indices
          (magic-box/xleft
           ending-label-box)
          (magic-box/ybot
           ending-label-box)))
        )
      ;Hokay, we're finally ready to make the wire

                                ;Do the starting vertical wire
          (instantiate-vertical-wire-segment
           mf
           (channel-indices/vertical-channel-index
            starting-channel-indices)
           (channel-indices/horizontal-channel-index
            starting-channel-indices)
           starting-horizontal-channel-index)

;and while we're at it, do the ending vertical wire

          (instantiate-vertical-wire-segment
           mf
           (channel-indices/vertical-channel-index
            ending-channel-indices)
           (channel-indices/horizontal-channel-index
            ending-channel-indices)
           ending-horizontal-channel-index)
```

```
;and now the first horizontal wire

           (instantiate-horizontal-wire-segment
            mf
            (channel-indices/vertical-channel-index
             starting-channel-indices)
            middle-vertical-channel-index
            starting-horizontal-channel-index
            )

;followed by the second horizontal wire

           (instantiate-horizontal-wire-segment
            mf
            middle-vertical-channel-index
            (channel-indices/vertical-channel-index
             ending-channel-indices)
            ending-horizontal-channel-index
            )

;finally, add the connecting vertical wire

           (instantiate-vertical-wire-segment
            mf
            middle-vertical-channel-index
              starting-horizontal-channel-index
              ending-horizontal-channel-index))))

#| This routine creates a horizontal wire on the specified layer in the
specified horizontal channel.  It assumes you wire left to right  |#

(define (instantiate-specified-horizontal-wire-segment
       mf layer width
       starting-vertical-channel-index
       ending-vertical-channel-index
       horizontal-channel-index)
  ;These boxes are in the coordinate system of the top-level cell
  (let ((starting-vertical-channel-box
       (vertical-channel-box starting-vertical-channel-index))
      (ending-vertical-channel-box
       (vertical-channel-box ending-vertical-channel-index))
      (horizontal-channel-box
        (horizontal-channel-box horizontal-channel-index))
      )

    (let ((starting-box (intersect-channel-boxes
                  starting-vertical-channel-box
                  horizontal-channel-box))
        (ending-box (intersect-channel-boxes
                  ending-vertical-channel-box
                  horizontal-channel-box)))

      (let ((starting-box-horizontal-center (horizontal-center
starting-box))
          (starting-box-vertical-center (vertical-center starting-box))
          (ending-box-horizontal-center (horizontal-center ending-box))
          (half-height))
```

```
      (append-box (get-layer mf layer)
                  (make-magic-box
                   (- starting-box-horizontal-center
                      (/ width 2))
                   (- starting-box-vertical-center
                      (/ width 2))
                   (+ ending-box-horizontal-center
                      (/ width 2))
                   (+ starting-box-vertical-center
                      (/ width 2))))

      ))))


#| This routine creates a vertical wire on the specified layer in the
specified vertical channel. |#

(define (instantiate-specified-vertical-wire-segment
       mf layer width
       vertical-channel-index
       starting-horizontal-channel-index
       horizontal-channel-index)
  ;These boxes are in the coordinate system of the top-level cell
  (let ((vertical-channel-box
       (vertical-channel-box vertical-channel-index))
      (starting-horizontal-channel-box
       (horizontal-channel-box starting-horizontal-channel-index))
      (ending-horizontal-channel-box
       (horizontal-channel-box horizontal-channel-index))
      )
    (let ((starting-box (intersect-channel-boxes
                  vertical-channel-box
                  starting-horizontal-channel-box))
        (ending-box (intersect-channel-boxes
                  vertical-channel-box
                  ending-horizontal-channel-box)))
      (append-box (get-layer mf layer)
              (make-magic-box
                   (- (horizontal-center starting-box)
                    (/ width 2))
                   (vertical-center starting-box)
                   (+ (horizontal-center ending-box)
                    (/ width 2))
                   (vertical-center ending-box))))))

#| This routine creates a wire on the specified layer in the specified
horizontal channel between two contacts.  It uses instantiate-
horizontal-strap
and instantiate-vertical-strap and assumes that you are wiring from
left to
right |#

(define (instantiate-single-layer-wire mf starting-use-ids starting-
label-name
                    ending-use-ids ending-label-name
                    layer width
                    horizontal-channel-index)
```

```
  (define (label->box mf use-ids label-name)
    (let ((transforms (list (identity)))
          (use))
      (for-each (lambda (use-id)
                (set! use (find-use mf use-id))
                (set! transforms
                    (cons (magic-use/transform use) transforms))
                (set! mf (read-magic-file (magic-use/filename use)))
                )
              use-ids)
      (transform-magic-box
       (apply compose-magic-transforms transforms)
       (magic-label/box (find-label mf label-name)))))

  (if (not (or (pair? starting-use-ids) (null? starting-use-ids)))
      (set! starting-use-ids (list starting-use-ids)))

  (if (not (or (pair? ending-use-ids) (null? ending-use-ids)))
      (set! ending-use-ids (list ending-use-ids)))

 (let ((starting-label-box (label->box mf starting-use-ids
                                starting-label-name))
      (ending-label-box (label->box mf ending-use-ids
                                ending-label-name)))

    (let ((starting-channel-indices
         (point->channel-indices
          (magic-box/xleft
           starting-label-box)
          (magic-box/ybot
           starting-label-box)))
        (ending-channel-indices
         (point->channel-indices
          (magic-box/xleft
           ending-label-box)
          (magic-box/ybot
           ending-label-box)))
        )

      ;Hokay, we're finally ready to make the wire
      (cond ((eq? (channel-indices/vertical-channel-index starting-
channel-indices)
                  (channel-indices/vertical-channel-index ending-
channel-indices))
            ;the wire only has a vertical segment
            (instantiate-specified-vertical-wire-segment
             mf
             layer width
             (channel-indices/vertical-channel-index
              starting-channel-indices)
             (channel-indices/horizontal-channel-index
              starting-channel-indices)
             (channel-indices/horizontal-channel-index
              ending-channel-indices))
            )
           (else
```

```
                          ;Do the starting vertical wire
            (instantiate-specified-vertical-wire-segment
             mf layer width
             (channel-indices/vertical-channel-index
              starting-channel-indices)
             (channel-indices/horizontal-channel-index
              starting-channel-indices)
             horizontal-channel-index)

            ;and while we're at it, do the ending vertical wire

            (instantiate-specified-vertical-wire-segment
             mf layer width
             (channel-indices/vertical-channel-index
              ending-channel-indices)
             (channel-indices/horizontal-channel-index
              ending-channel-indices)
             horizontal-channel-index)

            ;and now the horizontal wire

            (instantiate-specified-horizontal-wire-segment
             mf layer width
             (channel-indices/vertical-channel-index
              starting-channel-indices)
             (channel-indices/vertical-channel-index
              ending-channel-indices)
             horizontal-channel-index
             )))
      )))


; routes a power grid on the specified magic file, consisting of the
number of
; specified vertical rails and horizontal rails of the specified width
; Also places substrate contacts
; This routine assumes that the topmost and bottommost rails are both
GND

(define (instantiate-power-grid mf horizontal-channel-span contact-
spacing vertical-rail-spacing vertical-rail-width)
  (let* ((boundary-box (magic-file-bounding-box mf))
       (left-boundary-channel (/ (vector-first boundary-box) 8))
       (bottom-boundary-channel (/ (vector-second boundary-box) 10))
       (right-boundary-channel (- (/ (vector-third boundary-box) 8) 1))
       (top-boundary-channel (- (/ (vector-fourth boundary-box) 10) 1))
       (horizontal-rail-index (+ top-boundary-channel 1))
       (substrate-contact-type 'ndc)
       (contact-type 'ndc)
       (xposition (/ contact-spacing 2))
       (rail-yoffset (* 5 (- horizontal-channel-span 1)))
       (rail-xoffset 0)
       (left-channel left-boundary-channel)
       (right-channel right-boundary-channel)
       (vertical-channel-number 0)
       (vertical-channel-position vertical-rail-spacing)
       )
```

```
    (if (integer? right-boundary-channel)
      #f
      (set! right-boundary-channel (ceiling right-boundary-channel)))

    (let instantiate-horizontal-rails ((horizontal-rail-index
horizontal-rail-index))
      (cond
       ((eqv? substrate-contact-type 'ndc)
      (set! left-channel (- left-boundary-channel 1))
      (set! right-channel (+ right-boundary-channel (* 2 (+ horizontal-
channel-span 1)))))
       ((eqv? substrate-contact-type 'pdc)
      (set! left-channel (- left-boundary-channel (* 2 (+ horizontal-
channel-span 1))))
      (set! right-channel (+ right-boundary-channel 1))))

      (if (> horizontal-rail-index (- bottom-boundary-channel
                                (+ 2 (* 3 horizontal-channel-span))))
        (begin
          (instantiate-horizontal-strap mf 'metal1 (+ (* 10 horizontal-
channel-span) 8) left-channel right-channel horizontal-rail-index rail-
yoffset)
          (let instantiate-substrate-contacts ((xposition xposition))
            (let ((channel-box (horizontal-channel-box horizontal-rail-
index)))
            (if (< xposition (* (+ right-boundary-channel 1) 8))
                (begin
                  (cond ((eqv? vertical-channel-number vertical-rail-
spacing)
                         (if (eqv? substrate-contact-type 'ndc)
                           (set! contact-type substrate-contact-type)
                           (set! contact-type 'm2contact))
                         (set! vertical-channel-number (+ vertical-
channel-number 1)))
                        ((eqv? vertical-channel-number (+ vertical-rail-
spacing 1))
                         (if (eqv? substrate-contact-type 'ndc)
                           (set! contact-type 'm2contact)
                           (set! contact-type substrate-contact-type))
                         (set! vertical-channel-number 0))

                        (else
                         (set! contact-type substrate-contact-type)
                         (set! vertical-channel-number (+ vertical-
channel-number 1))))

                  (append-box (get-layer mf contact-type)
                          (make-magic-box
                           xposition
                           (- (+ rail-yoffset (vertical-center channel-
box))
                              2)
                           (+ xposition 4)
                           (+ rail-yoffset (vertical-center channel-
box)
                              2)))
```

```
                  (instantiate-substrate-contacts (+ xposition 4
contact-spacing))))))
          (if (eqv? substrate-contact-type 'ndc)
            (set! substrate-contact-type 'pdc)
            (set! substrate-contact-type 'ndc))
          (set! vertical-channel-number 0)
          (instantiate-horizontal-rails (- horizontal-rail-index (+ 6
horizontal-channel-span))))
        #f))

    (let instantiate-vertical-rails ((vertical-channel-position
vertical-channel-position))
      (if (< vertical-channel-position right-boundary-channel)
        (begin
          (instantiate-vertical-strap mf 'metal2 vertical-rail-width
vertical-channel-position 6 (- top-boundary-channel 6) 0 (- (* 5
horizontal-channel-span)) 0)
          (instantiate-vertical-strap mf 'metal2 vertical-rail-width (+
vertical-channel-position 1) 0  top-boundary-channel 0 (* 5 horizontal-
channel-span) 0)
          (instantiate-vertical-rails (+ vertical-channel-position 2
vertical-rail-spacing)))
        #f))


    (let instantiate-vertical-boundary ((horizontal-rail-index
horizontal-rail-index))
      (if (> horizontal-rail-index 18)
        (begin
              (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10
horizontal-channel-span)) -1 (+ horizontal-rail-index 1) (- horizontal-
rail-index  1 (* 2 (+ 6 horizontal-channel-span))) (- (+ 2 (/ (* 10
horizontal-channel-span) 2)))(* 5 horizontal-channel-span) 0)

              (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10
horizontal-channel-span)) (+ 2 right-boundary-channel)  (+ horizontal-
rail-index 1) (- horizontal-rail-index 1 (* 2 (+ 6 horizontal-channel-
span))) (+ 6 (* 15 horizontal-channel-span)) (* 5 horizontal-channel-
span) 0)

        (if (> horizontal-rail-index 11)
            (begin
            (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10
horizontal-channel-span)) -2 (- horizontal-rail-index 6 horizontal-
channel-span) (- horizontal-rail-index (* 3 (+ 6 horizontal-channel-
span))) (- (+ 6 (* 15 horizontal-channel-span)))(- (* 5 horizontal-
channel-span)) 0)
            (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10
horizontal-channel-span)) (+ 1 right-boundary-channel) (- horizontal-
rail-index 6 horizontal-channel-span) (- horizontal-rail-index  (* 3 (+
6 horizontal-channel-span)))(+ 2 (/ (* 10 horizontal-channel-span) 2))
(- (* 5 horizontal-channel-span)) 0))

            #f)
```

```
          (instantiate-vertical-boundary (- horizontal-rail-index (* 2
(+ 6 horizontal-channel-span)))))

        #f))

      (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10 horizontal-
channel-span)) -2 6 (- top-boundary-channel 6) (- (+ 6 (* 15
horizontal-channel-span)))(- (* 5 horizontal-channel-span)) 0)

    (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10 horizontal-
channel-span)) -1 0  top-boundary-channel (- (+ 2 (/ (* 10 horizontal-
channel-span) 2)))(* 5 horizontal-channel-span) 0)

    (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10 horizontal-
channel-span)) (+ 1 right-boundary-channel) 6 (- top-boundary-channel
6) (+ 2 (/ (* 10 horizontal-channel-span) 2)) (- (* 5 horizontal-
channel-span)) 0)

    (instantiate-vertical-strap mf 'metal2 (+ 8 (* 10 horizontal-
channel-span)) (+ 2 right-boundary-channel) 0  top-boundary-channel (+
6 (* 15 horizontal-channel-span)) (* 5 horizontal-channel-span) 0)
    ))
```

A.2 counter.scm

```
; 32-bit counter example to test out magic.scm

(declare (usual-integrations))

(define row-spacing 1)
(define empty-space (* 10 row-spacing))


; mux
(define (tgmux_gnd)
  (let ((new-file (make-magic-file
                'filename "tgmux_gnd"
                'tech "scmos"))
       )
    (instantiate-and-tile
     new-file
     '("tgmux")
     '("tgmux-0")
     'horizontal
     (list identity)
     0
     0
     '(0)
     '(0))

    (instantiate-vertical-strap new-file
                         'metal2
                         4
                         1
                         0
                         2
                         0
                         -2
                         -3)
    (write-magic-file new-file)))


(define (mux_32)
  (let ((new-file (make-magic-file
                'filename "mux_32"
                'tech "scmos"))
       )
    (instantiate-and-tile-vertical-array
     new-file
     "tgmux_gnd"
     32
     0
     0
     0
     empty-space)

    (instantiate-wire new-file
                 '("tgmux_gnd-0")
```

```
                    "S"
                    '("tgmux_gnd-31")
                    "S"
                    3)

    (instantiate-wire new-file
                    '("tgmux_gnd-0")
                    "-S"
                    '("tgmux_gnd-31")
                    "-S"
                    3)
    (write-magic-file new-file)))

; register
(define (reg_bit)
  (let ((new-file (make-magic-file
                'filename "reg_bit"
                'tech "scmos"))
      )
    (instantiate-and-tile
     new-file
     '("tsinv" "inv_1" "inv_weak" "tg" "inv_1" "inv_weak_big_contacts"
"inv_2")
     '("tsinv-0" "inv_1-0" "inv_weak-0" "tg-0" "inv_1-1" "inv_weak-1"
"inv_2-0")
     'horizontal
     (list identity
         identity
         identity
         identity
         sideways
         identity)
     0
     0
     '(0 0 0 0 0 -8 0)
     '(0 0 0 0 0 0 0)
     )

    (instantiate-wire new-file
                    '("tsinv-0")
                    "Z"
                    '("inv_1-0")
                    "A"
                    1)

    (instantiate-wire new-file
                    '("inv_1-0")
                    "Z"
                    '("tg-0")
                    "D1"
                    3)

    (instantiate-single-layer-wire new-file
                    '("inv_1-0")
                    "Z"
                    '("inv_weak-0")
```

```
                "A"
                'metal1
                4
                2)

    (instantiate-wire new-file
                '("inv_weak-0")
                "Z"
                '("inv_1-0")
                "A"
                1)

    (instantiate-wire new-file
                '("tsinv-0")
                "EN"
                '("tg-0")
                "S0"
                0)

    (instantiate-wire new-file
                '("tsinv-0")
                "-EN"
                '("tg-0")
                "S1"
                4)

    (instantiate-wire new-file
                '("tg-0")
                "Y"
                '("inv_1-1")
                "A"
                1)

    (instantiate-wire new-file
                '("tg-0")
                "Y"
                '("inv_weak-1")
                "Z"
                1)

    (instantiate-wire new-file
                '("inv_1-1")
                "Z"
                '("inv_weak-1")
                "A"
                0)

    (write-magic-file new-file)))


(define (basic_reg)
  (let ((new-file (make-magic-file
              'filename "basic_reg"
              'tech "scmos"))
    )
    (instantiate-and-tile-vertical-array
     new-file
```

```
     "reg_bit"
     32
     0
     0
     0
     (* 10 row-spacing))



    (write-magic-file new-file)))


;inverter array
(define (inv_1_array)
  (let ((new-file (make-magic-file
                'filename "inv_1_array"
                'tech "scmos"))
      )
    (instantiate-and-tile-vertical-array
     new-file
     "inv_1"
     32
     0
     0
     0
     (* 10 row-spacing))

    (write-magic-file new-file)))


; counter logic
(define (nor_nand_tree_triple)
  (let ((new-file (make-magic-file
                'filename "nor_nand_tree_triple"
                'tech "scmos"))
      )
    (instantiate-and-tile
     new-file
     '("nor2" "nand2" "nor2")
     '("nor2-0" "nand2-0" "nor2-1")
     'vertical
     (list identity
         upsidedown
         identity)
     0
     0
     '(0 0 0)
     (list empty-space empty-space empty-space))

    (instantiate-wire new-file
                  '("nor2-0")
                  "Z"
                  '("nand2-0")
                  "A"
                  3)
```

```
(instantiate-wire new-file
                  '("nor2-1")
                  "Z"
                  '("nand2-0")
                  "B"
                  12)
    (write-magic-file new-file)))


(define (nor_nand_nor_tree)
  (let ((new-file (make-magic-file
                'filename "nor_nand_nor_tree"
                'tech "scmos"))
      )
    (instantiate-and-tile
     new-file
     '("nor_nand_tree_triple" "nor2" "nor_nand_tree_triple")
     '("nor_nand_tree_triple-0" "nor2-0" "nor_nand_tree_triple-1")
     'vertical
     (list identity
         upsidedown
         identity)
     0
     0
     '(0 0 0)
     (list empty-space empty-space empty-space))

    (instantiate-five-segment-wire new-file
                           '("nor_nand_tree_triple-0" "nand2-0")
                           "Z"
                           '("nor2-0")
                           "A"
                           11
                           22
                           3)

    (instantiate-five-segment-wire new-file
                           '("nor_nand_tree_triple-1" "nand2-0")
                           "Z"
                           '("nor2-0")
                           "B"
                           36
                           25
                           3)
    (write-magic-file new-file)))


(define (nor_nand_nor_nand_tree)
  (let ((new-file (make-magic-file
                'filename "nor_nand_nor_nand_tree"
                'tech "scmos"))
      )
    (instantiate-and-tile
     new-file
     '("nor_nand_nor_tree" "nand2" "nor_nand_nor_tree")
     '("nor_nand_nor_tree-0" "nand2-0" "nor_nand_nor_tree-1")
     'vertical
```

```
     (list identity
         upsidedown
         identity)
     0
     0
     '(0 0 0)
     (list empty-space empty-space empty-space))

    (instantiate-five-segment-wire new-file
                    '("nor_nand_nor_tree-0" "nor2-0")
                    "Z"
                    '("nand2-0")
                    "A"
                    23
                    51
                    4)

    (instantiate-five-segment-wire new-file
                    '("nor_nand_nor_tree-1" "nor2-0")
                    "Z"
                    '("nand2-0")
                    "B"
                    79
                    53
                    4)
    (write-magic-file new-file)))


(define (inv_2and4)
  (let ((new-file (make-magic-file
                'filename "inv_2and4"
                'tech "scmos"))
      )
    (instantiate-and-tile-horizontally
     new-file
     '("inv_2" "inv_4")
     '("inv_2-0" "inv_4-0")
     (list identity
         identity)
     0
     0
     0
     )

    (instantiate-single-layer-wire new-file
                    '("inv_2-0")
                    "Z"
                    '("inv_4-0")
                    "A"
                    'metal1
                    4
                    2)
    (write-magic-file new-file)))


(define (counter_logic)
  (let ((new-file (make-magic-file
```

```
                'filename "counter_logic"
                'tech "scmos"))
      )
    (instantiate-and-tile
     new-file
     '("nor_nand_nor_nand_tree" "nor2" "inv_2and4"
"nor_nand_nor_nand_tree")
     '("nor_nand_nor_nand_tree-0" "nor2-0" "inv_2and4-0"
"nor_nand_nor_nand_tree-1")
     'vertical
     (list identity
         upsidedown
         identity
         upsidedown)
     0
     0
     '(0 0 0 0)
     (list empty-space empty-space empty-space empty-space))

   (instantiate-five-segment-wire new-file
                           '("nor_nand_nor_nand_tree-0" "nand2-0")
                           "Z"
                           '("nor2-0")
                           "A"
                           54
                           107
                           5)

    (instantiate-five-segment-wire new-file
                    '("nor_nand_nor_nand_tree-1" "nand2-0")
                    "Z"
                    '("nor2-0")
                    "B"
                    168
                    109
                    5)


    (instantiate-wire new-file
                  '("nor2-0")
                  "Z"
                  '("inv_2and4-0" "inv_2-0")
                  "A"
                  112)
 (write-magic-file new-file)))


(define (counter)
  (let ((new-file (make-magic-file
                'filename "counter"
                'tech "scmos"))
      )
    (instantiate-and-tile
     new-file
     '("mux_32" "addthirtytwo" "basic_reg" "inv_1_array"
"counter_logic")
```

```
     '("mux_32-0" "addthirtytwo-0" "basic_reg-0" "inv_1_array-0"
"counter_logic-0")
     'horizontal
     (list identity
         identity
         identity
         identity
         identity)
     0
     0
     '(0 16 0 0 0)
     '(0 0 0 0 0)
     )

;adder inputs (1)
;bit0
    (instantiate-vertical-strap new-file
                          'metal2
                          4
                          10
                          2
                          5
                          0
                          -5
                          -1)
;bit2
    (instantiate-vertical-strap new-file
                          'metal2
                          4
                          10
                          14
                          16
                          0
                          -6
                          1)
;bit4
    (instantiate-vertical-strap new-file
                          'metal2
                          4
                          10
                          28
                          30
                          0
                          -6
                          1)
;bit6
    (instantiate-vertical-strap new-file
                          'metal2
                          4
                          10
                          42
                          44
                          0
                          -6
                          1)
```

```
;bit8
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        56
                        58
                        0
                        -6
                        1)
;bit10
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        70
                        72
                        0
                        -6
                        1)
;bit12
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        84
                        86
                        0
                        -6
                        1)
;bit14
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        98
                        100
                        0
                        -6
                        1)
;bit16
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        112
                        114
                        0
                        -6
                        1)
```

```
;bit18
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        126
                        128
                        0
                        -6
                        1)
;bit20
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        140
                        142
                        0
                        -6
                        1)
;bit22
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        154
                        156
                        0
                        -6
                        1)
;bit24
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        168
                        170
                        0
                        -6
                        1)
;bit26
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        182
                        184
                        0
                        -6
                        1)
;bit28
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        196
                        198
```

```
                        0
                        -6
                        1)
;bit30
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        10
                        210
                        212
                        0
                        -6
                        1)
;bit1
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        10
                        12
                        0
                        -5
                        -1)
;bit3
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        24
                        26
                        0
                        -5
                        -1
                        )
;bit5
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        38
                        40
                        0
                        -5
                        -1)
;bit7
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        52
                        54
                        0
                        -5
                        -1)
;bit9
    (instantiate-vertical-strap new-file
                        'metal2
```

```
                        4
                        12
                        66
                        68
                        0
                        -5
                        -1)
;bit11
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        80
                        82
                        0
                        -5
                        -1)
;bit13
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        94
                        96
                        0
                        -5
                        -1)
;bit15
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        108
                        110
                        0
                        -5
                        -1)
;bit17
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        122
                        124
                        0
                        -5
                        -1)
;bit19
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        136
                        138
                        0
                        -5
                        -1)
```

```
;bit21
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        150
                        152
                        0
                        -5
                        -1)
;bit23
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        164
                        166
                        0
                        -5
                        -1)
;bit25
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        178
                        180
                        0
                        -5
                        -1)
;bit27
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        192
                        194
                        0
                        -5
                        -1)
;bit29
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        206
                        208
                        0
                        -5
                        -1)
;bit31
    (instantiate-vertical-strap new-file
                        'metal2
                        4
                        12
                        220
                        222
```

```
                        0
                        -5
                        -1)

; output of mux to adder

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-0")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryfour-0" "carrytwo-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    3)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-1")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryfour-0" "carrytwo-0" "carryone-0" "gp-0" "and2-
0" "nand2-0")
                    "A"
                    9)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-2")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryfour-0" "carrytwobb-0" "carrytwo-0" "gp-0"
"and2-0" "nand2-0")
                    "A"
                    17)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-3")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryfour-0" "carrytwobb-0" "carrytwo-0" "carryone-
0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    23)


  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-4")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwo-0"
"gp-0" "and2-0" "nand2-0")
                    "A"
                    31)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-5")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwo-0"
"carryone-0" "gp-0" "and2-0" "nand2-0")
```

```
                  "A"
                  37)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-6")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwobb-0"
"carrytwo-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  45)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-7")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwobb-0"
"carrytwo-0" "carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  51)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-8")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryfour-0" "carrytwo-0"
"gp-0" "and2-0" "nand2-0")
                  "A"
                  59)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-9")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryfour-0" "carrytwo-0"
"carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  65)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-10")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryfour-0" "carrytwobb-0"
"carrytwo-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  73)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-11")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryfour-0" "carrytwobb-0"
"carrytwo-0" "carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  79)
```

```
  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-12")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryeight_tophalf-0"
"carryfour-0" "carrytwo-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    87)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-13")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryeight_tophalf-0"
"carryfour-0" "carrytwo-0" "carryone-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    93)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-14")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryeight_tophalf-0"
"carryfour-0" "carrytwobb-0" "carrytwo-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    101)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-15")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carrysixteen_tophalf-0" "carryeight-0" "carryeight_tophalf-0"
"carryfour-0" "carrytwobb-0" "carrytwo-0" "carryone-0" "gp-0" "and2-0"
"nand2-0")
                    "A"
                    107)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-16")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0" "carryfour-
0" "carrytwo-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    115)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-17")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0" "carryfour-
0" "carrytwo-0" "carryone-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    121)

  (instantiate-wire new-file
```

```
                  '("mux_32-0" "tgmux_gnd-18")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0" "carryfour-
0" "carrytwobb-0" "carrytwo-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  129)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-19")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0" "carryfour-
0" "carrytwobb-0" "carrytwo-0" "carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  135)


  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-20")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0"
"carryeight_tophalf-0" "carryfour-0" "carrytwo-0" "gp-0" "and2-0"
"nand2-0")
                  "A"
                  143)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-21")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0"
"carryeight_tophalf-0" "carryfour-0" "carrytwo-0" "carryone-0" "gp-0"
"and2-0" "nand2-0")
                  "A"
                  149)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-22")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0"
"carryeight_tophalf-0" "carryfour-0" "carrytwobb-0" "carrytwo-0" "gp-0"
"and2-0" "nand2-0")
                  "A"
                  157)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-23")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carryeight-0"
"carryeight_tophalf-0" "carryfour-0" "carrytwobb-0" "carrytwo-0"
"carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  163)
```

```
  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-24")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryfour-0" "carrytwo-0" "gp-0" "and2-0" "nand2-0")
                    "A"
                    171)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-25")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryfour-0" "carrytwo-0" "carryone-0" "gp-0" "and2-0"
"nand2-0")
                    "A"
                    177)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-26")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryfour-0" "carrytwobb-0" "carrytwo-0" "gp-0" "and2-
0" "nand2-0")
                    "A"
                    185)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-27")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryfour-0" "carrytwobb-0" "carrytwo-0" "carryone-0"
"gp-0" "and2-0" "nand2-0")
                    "A"
                    191)


  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-28")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwo-0" "gp-0"
"and2-0" "nand2-0")
                    "A"
                    199)

  (instantiate-wire new-file
                    '("mux_32-0" "tgmux_gnd-29")
                    "Y"
                    '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
```

```
"carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwo-0"
"carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  205)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-30")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwobb-0"
"carrytwo-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  213)

  (instantiate-wire new-file
                  '("mux_32-0" "tgmux_gnd-31")
                  "Y"
                  '("addthirtytwo-0" "carrythirtytwo-0"
"carrythirtytwo_tophalf-0" "carrysixteen-0" "carrysixteen_tophalf-0"
"carryeight-0" "carryeight_tophalf-0" "carryfour-0" "carrytwobb-0"
"carrytwo-0" "carryone-0" "gp-0" "and2-0" "nand2-0")
                  "A"
                  219)

; adder outputs to register

  (instantiate-wire new-file
                  '("addthirtytwo-0" "carrythirtytwo-0" "carrysixteen-
0" "carryeight-0" "carryfour-0" "carrytwo-0" "gp-0" "and2-0" "inv_1-0")
                  "Z"
                  '("basic_reg-0" "reg_bit-0" "tsinv-0")
                  "A"
                  4)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-0" "xnor-0")
                  "Z"
                  '("basic_reg-0" "reg_bit-1" "tsinv-0")
                  "A"
                  8)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-0" "xnor-1")
                  "Z"
                  '("basic_reg-0" "reg_bit-2" "tsinv-0")
                  "A"
                  15)
    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-0" "xnor-2")
                  "Z"
                  '("basic_reg-0" "reg_bit-3" "tsinv-0")
                  "A"
                  22)
```

```
    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-0" "xnor-3")
                    "Z"
                    '("basic_reg-0" "reg_bit-4" "tsinv-0")
                    "A"
                    29)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-1" "xnor-3")
                    "Z"
                    '("basic_reg-0" "reg_bit-5" "tsinv-0")
                    "A"
                    36)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-1" "xnor-2")
                    "Z"
                    '("basic_reg-0" "reg_bit-6" "tsinv-0")
                    "A"
                    43)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-1" "xnor-1")
                    "Z"
                    '("basic_reg-0" "reg_bit-7" "tsinv-0")
                    "A"
                    50)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-1" "xnor-0")
                    "Z"
                    '("basic_reg-0" "reg_bit-8" "tsinv-0")
                    "A"
                    57)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-2" "xnor-0")
                    "Z"
                    '("basic_reg-0" "reg_bit-9" "tsinv-0")
                    "A"
                    64)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-2" "xnor-1")
                    "Z"
                    '("basic_reg-0" "reg_bit-10" "tsinv-0")
                    "A"
                    71)
```

```
    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-2" "xnor-2")
                    "Z"
                    '("basic_reg-0" "reg_bit-11" "tsinv-0")
                    "A"
                    78)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-2" "xnor-3")
                    "Z"
                    '("basic_reg-0" "reg_bit-12" "tsinv-0")
                    "A"
                    85)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-3" "xnor-3")
                    "Z"
                    '("basic_reg-0" "reg_bit-13" "tsinv-0")
                    "A"
                    92)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-3" "xnor-2")
                    "Z"
                    '("basic_reg-0" "reg_bit-14" "tsinv-0")
                    "A"
                    99)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-3" "xnor-1")
                    "Z"
                    '("basic_reg-0" "reg_bit-15" "tsinv-0")
                    "A"
                    106)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-3" "xnor-0")
                    "Z"
                    '("basic_reg-0" "reg_bit-16" "tsinv-0")
                    "A"
                    113)

    (instantiate-wire new-file
                    '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-4" "xnor-0")
                    "Z"
                    '("basic_reg-0" "reg_bit-17" "tsinv-0")
                    "A"
                    120)
```

```
    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-4" "xnor-1")
                  "Z"
                  '("basic_reg-0" "reg_bit-18" "tsinv-0")
                  "A"
                  127)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-4" "xnor-2")
                  "Z"
                  '("basic_reg-0" "reg_bit-19" "tsinv-0")
                  "A"
                  134)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-4" "xnor-3")
                  "Z"
                  '("basic_reg-0" "reg_bit-20" "tsinv-0")
                  "A"
                 141)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-5" "xnor-3")
                  "Z"
                  '("basic_reg-0" "reg_bit-21" "tsinv-0")
                  "A"
                 148)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-5" "xnor-2")
                  "Z"
                  '("basic_reg-0" "reg_bit-22" "tsinv-0")
                  "A"
                 155)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-5" "xnor-1")
                  "Z"
                  '("basic_reg-0" "reg_bit-23" "tsinv-0")
                  "A"
                 162)

    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-5" "xnor-0")
                  "Z"
                  '("basic_reg-0" "reg_bit-24" "tsinv-0")
                  "A"
                 169)
```

```
    (instantiate-wire new-file
                  '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-6" "xnor-0")
                   "Z"
                   '("basic_reg-0" "reg_bit-25" "tsinv-0")
                   "A"
                  176)

    (instantiate-wire new-file
                   '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-6" "xnor-1")
                   "Z"
                   '("basic_reg-0" "reg_bit-26" "tsinv-0")
                   "A"
                  183)

    (instantiate-wire new-file
                   '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-6" "xnor-2")
                   "Z"
                   '("basic_reg-0" "reg_bit-27" "tsinv-0")
                   "A"
                  190)

    (instantiate-wire new-file
                   '("addthirtytwo-0" "xnor_array_with_inv-0"
"xnor_quad-6" "xnor-3")
                   "Z"
                   '("basic_reg-0" "reg_bit-28" "tsinv-0")
                   "A"
                  197)

    (instantiate-wire new-file
                   '("addthirtytwo-0" "xnor_array_with_inv-0" "xnor-0")
                   "Z"
                   '("basic_reg-0" "reg_bit-29" "tsinv-0")
                   "A"
                  204)

    (instantiate-wire new-file
                   '("addthirtytwo-0" "xnor_array_with_inv-0" "xnor-1")
                   "Z"
                   '("basic_reg-0" "reg_bit-30" "tsinv-0")
                   "A"
                  211)

    (instantiate-wire new-file
                   '("addthirtytwo-0" "xnor_array_with_inv-0" "xnor-2")
                   "Z"
                   '("basic_reg-0" "reg_bit-31" "tsinv-0")
                   "A"
                   218)


; outputs of register to inverters

    (instantiate-single-layer-wire new-file
```

```
                        '("basic_reg-0" "reg_bit-0" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-0")
                        "A"
                        'metal1
                        4
                        2)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-1" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-1")
                        "A"
                        'metal1
                        4
                        10)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-2" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-2")
                        "A"
                        'metal1
                        4
                        16)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-3" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-3")
                        "A"
                        'metal1
                        4
                        24)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-4" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-4")
                        "A"
                        'metal1
                        4
                        30)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-5" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-5")
                        "A"
                        'metal1
                        4
                        38)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-6" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-6")
```

```
                        "A"
                        'metal1
                        4
                        44)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-7" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-7")
                        "A"
                        'metal1
                        4
                        52)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-8" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-8")
                        "A"
                        'metal1
                        4
                        58)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-9" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-9")
                        "A"
                        'metal1
                        4
                        66)

(instantiate-single-layer-wire new-file
               '("basic_reg-0" "reg_bit-10" "inv_2-0")
               "Z"
               '("inv_1_array-0" "inv_1-10")
               "A"
               'metal1
               4
               72)

(instantiate-single-layer-wire new-file
               '("basic_reg-0" "reg_bit-11" "inv_2-0")
               "Z"
               '("inv_1_array-0" "inv_1-11")
               "A"
               'metal1
               4
               80)
(instantiate-single-layer-wire new-file
               '("basic_reg-0" "reg_bit-12" "inv_2-0")
               "Z"
               '("inv_1_array-0" "inv_1-12")
               "A"
               'metal1
               4
               86)
```

```
(instantiate-single-layer-wire new-file
              '("basic_reg-0" "reg_bit-13" "inv_2-0")
              "Z"
              '("inv_1_array-0" "inv_1-13")
              "A"
              'metal1
              4
              94)

(instantiate-single-layer-wire new-file
              '("basic_reg-0" "reg_bit-14" "inv_2-0")
              "Z"
              '("inv_1_array-0" "inv_1-14")
              "A"
              'metal1
              4
              100)

(instantiate-single-layer-wire new-file
              '("basic_reg-0" "reg_bit-15" "inv_2-0")
              "Z"
              '("inv_1_array-0" "inv_1-15")
              "A"
              'metal1
              4
              108)

(instantiate-single-layer-wire new-file
              '("basic_reg-0" "reg_bit-16" "inv_2-0")
              "Z"
              '("inv_1_array-0" "inv_1-16")
              "A"
              'metal1
              4
              114)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-17" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-17")
              "A"
              'metal1
              4
              122)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-18" "inv_2-0")
              "Z"
              '("inv_1_array-0" "inv_1-18")
              "A"
              'metal1
              4
              128)

(instantiate-single-layer-wire new-file
              '("basic_reg-0" "reg_bit-19" "inv_2-0")
```

```
                "Z"
                '("inv_1_array-0" "inv_1-19")
                "A"
                'metal1
                4
                136)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-20" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-20")
                "A"
                'metal1
                4
                142)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-21" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-21")
                "A"
                'metal1
                4
                150)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-22" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-22")
                "A"
                'metal1
                4
                156)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-23" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-23")
                "A"
                'metal1
                4
                164)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-24" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-24")
                "A"
                'metal1
                4
                170)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-25" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-25")
                "A"
```

```
                'metal1
                4
                178)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-26" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-26")
                "A"
                'metal1
                4
                184)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-27" "inv_2-0")
                        "Z"
                '("inv_1_array-0" "inv_1-27")
                "A"
                'metal1
                4
                192)

(instantiate-single-layer-wire new-file
                        '("basic_reg-0" "reg_bit-28" "inv_2-0")
                        "Z"
                        '("inv_1_array-0" "inv_1-28")
                "A"
                'metal1
                4
                198)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-29" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-29")
                "A"
                'metal1
                4
                206)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-30" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-30")
                "A"
                'metal1
                4
                212)

(instantiate-single-layer-wire new-file
                '("basic_reg-0" "reg_bit-31" "inv_2-0")
                "Z"
                '("inv_1_array-0" "inv_1-31")
                "A"
                'metal1
                4
                220)
```

```
; output of inverters to logic

    (instantiate-single-layer-wire new-file
                            '("inv_1_array-0" "inv_1-0")
                            "Z"
                            '("counter_logic-0" "nor_nand_nor_nand_tree-
0" "nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-0")
                            "A"
                            'metal1
                            4
                            2)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-1")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-0")
                    "B"
                    4)

    (instantiate-single-layer-wire new-file
                            '("inv_1_array-0" "inv_1-2")
                            "Z"
                            '("counter_logic-0" "nor_nand_nor_nand_tree-
0" "nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-1")
                            "A"
                            'metal1
                            4
                            16)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-3")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-1")
                    "B"
                    18)

    (instantiate-single-layer-wire new-file
                            '("inv_1_array-0" "inv_1-4")
                            "Z"
                            '("counter_logic-0" "nor_nand_nor_nand_tree-
0" "nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-0")
                            "A"
                            'metal1
                            4
                            30)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-5" "inv_2-0")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-0")
                    "B"
                    32)
```

```
#|    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-5")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-0")
                  "B"
                  32)  |#

    (instantiate-single-layer-wire new-file
                           '("inv_1_array-0" "inv_1-6")
                           "Z"
                           '("counter_logic-0" "nor_nand_nor_nand_tree-
0" "nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-1")
                           "A"
                           'metal1
                           4
                           44)

    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-7")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-1")
                  "B"
                  46)

    (instantiate-single-layer-wire new-file
                           '("inv_1_array-0" "inv_1-8")
                           "Z"
                           '("counter_logic-0" "nor_nand_nor_nand_tree-
0" "nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-0")
                           "A"
                           'metal1
                           4
                           58)

    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-9")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-0")
                  "B"
                  60)

    (instantiate-single-layer-wire new-file
                  '("inv_1_array-0" "inv_1-10")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-1")
                  "A"
                  'metal1
                  4
                  72)

    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-11")
                  "Z"
```

```
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-1")
                    "B"
                    74)

    (instantiate-single-layer-wire new-file
                    '("inv_1_array-0" "inv_1-12")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-0")
                    "A"
                    'metal1
                    4
                    86)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-13")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-0")
                    "B"
                    88)

    (instantiate-single-layer-wire new-file
                    '("inv_1_array-0" "inv_1-14")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-1")
                    "A"
                    'metal1
                    4
                    100)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-15")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-0"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-1")
                    "B"
                    102)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-16")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-1")
                    "B"
                    116)
    (instantiate-single-layer-wire new-file
                    '("inv_1_array-0" "inv_1-17")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-1")
                    "A"
                    'metal1
                    4
                    122)
```

```
    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-18")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-0")
                  "B"
                  134)

    (instantiate-single-layer-wire new-file
                  '("inv_1_array-0" "inv_1-19")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-1" "nor2-0")
                  "A"
                  'metal1
                  4
                  136)

    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-20")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-1")
                  "B"
                  148)

    (instantiate-single-layer-wire new-file
                  '("inv_1_array-0" "inv_1-21")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-1")
                  "A"
                  'metal1
                  4
                  150)

    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-22")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-0")
                  "B"
                  162)

    (instantiate-single-layer-wire new-file
                  '("inv_1_array-0" "inv_1-23")
                  "Z"
                  '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-1" "nor_nand_tree_triple-0" "nor2-0")
                  "A"
                  'metal1
                  4
                  164)

    (instantiate-wire new-file
                  '("inv_1_array-0" "inv_1-24")
```

```
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-1")
                    "B"
                    176)

    (instantiate-single-layer-wire new-file
                    '("inv_1_array-0" "inv_1-25")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-1")
                    "A"
                    'metal1
                    4
                    178)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-26")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-0")
                    "B"
                    190)

    (instantiate-single-layer-wire new-file
                    '("inv_1_array-0" "inv_1-27")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-1" "nor2-0")
                    "A"
                    'metal1
                    4
                    192)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-28")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-1")
                    "B"
                    204)

    (instantiate-single-layer-wire new-file
                    '("inv_1_array-0" "inv_1-29")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-1")
                    "A"
                    'metal1
                    4
                    206)

    (instantiate-wire new-file
                    '("inv_1_array-0" "inv_1-30")
                    "Z"
                    '("counter_logic-0" "nor_nand_nor_nand_tree-1"
"nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-0")
```

```
                    "B"
                    218)

    (instantiate-single-layer-wire new-file
                            '("inv_1_array-0" "inv_1-31")
                            "Z"
                            '("counter_logic-0" "nor_nand_nor_nand_tree-
1" "nor_nand_nor_tree-0" "nor_nand_tree_triple-0" "nor2-0")
                            "A"
                            'metal1
                            4
                            220)
; output of register to mux input

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-0" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-0")
                    "D0"
                    6)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-1" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-1")
                    "D0"
                    7)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-2" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-2")
                    "D0"
                    13)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-3" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-3")
                    "D0"
                    20)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-4" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-4")
                    "D0"
                    27)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-5" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-5")
                    "D0"
                    34)

    (instantiate-wire new-file
```

```
                '("basic_reg-0" "reg_bit-6" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-6")
                "D0"
                41)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-7" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-7")
                "D0"
                48)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-8" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-8")
                "D0"
                55)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-9" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-9")
                "D0"
                62)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-10" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-10")
                "D0"
                69)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-11" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-11")
                "D0"
                76)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-12" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-12")
                "D0"
                83)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-13" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-13")
                "D0"
                90)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-14" "inv_2-0")
```

```
                "Z"
                '("mux_32-0" "tgmux_gnd-14")
                "D0"
                97)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-15" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-15")
                "D0"
                104)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-16" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-16")
                "D0"
                111)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-17" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-17")
                "D0"
                118)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-18" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-18")
                "D0"
                125)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-19" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-19")
                "D0"
                132)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-20" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-20")
                "D0"
                139)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-21" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-21")
                "D0"
                146)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-22" "inv_2-0")
                "Z"
```

```
                '("mux_32-0" "tgmux_gnd-22")
                "D0"
                153)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-23" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-23")
                "D0"
                160)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-24" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-24")
                "D0"
                167)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-25" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-25")
                "D0"
                174)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-26" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-26")
                "D0"
                181)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-27" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-27")
                "D0"
                188)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-28" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-28")
                "D0"
                195)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-29" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-29")
                "D0"
                202)

(instantiate-wire new-file
                '("basic_reg-0" "reg_bit-30" "inv_2-0")
                "Z"
                '("mux_32-0" "tgmux_gnd-30")
```

```
                    "D0"
                    209)

    (instantiate-wire new-file
                    '("basic_reg-0" "reg_bit-31" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-31")
                    "D0"
                    216)

; selector bits from logic to mux

    (instantiate-five-segment-wire new-file
                    '("counter_logic-0" "inv_2and4-0" "inv_2-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-0")
                    "S"
                    113
                    -1
                    95)

(instantiate-power-grid new-file row-spacing 4 65 4)

    (instantiate-five-segment-wire new-file
                    '("counter_logic-0" "inv_2and4-0" "inv_4-0")
                    "Z"
                    '("mux_32-0" "tgmux_gnd-31")
                    "-S"
                    115
                    223
                    95)
    (write-magic-file new-file)))

(define (make-basic_reg)
  (reg_bit)
  (basic_reg))

(define (make-counter_logic)
  (nor_nand_tree_triple)
  (nor_nand_nor_tree)
  (nor_nand_nor_nand_tree)
  (inv_2and4)
  (counter_logic))

(define (make-counter)
  (make-basic_reg)
  (mux_32)
  (make-counter_logic)
  (inv_1_array)
  (counter))
```

References

1. Ulrich Lauther, "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Proceedings of the 16th Design Automation Conference*, 1979, pp. 1-10.

2. Mark Hortoog, "Analysis of Placement Procedures for VLSI Standard Cell Layout," *Proceedings of the 23rd Design Automation Conference*, 1986, pp. 314-319.

3. Ronald Rivest and Charles Fidducia, "A 'Greedy' Channel Router," *Proceedings of the 19th Design Automation Conference*, 1982, pp. 418-424.

4. J. Soukup, "Global Router," *Proceedings of the 16th Design Automation Conference*, 1978, pp. 481-484.

5. C. Sechen and A. Sangiovanni-Vincentelli, "Timberwolf3.2: A New Standard Cell Placement and Global Routing Package," *Proceedings of the International Conference on Computer-Aided Design*, 1984, pp. 72-75.

6. http://www.swiss.ai.mit.edu/~switz/amorphous/index.html

7. Magic, Version 6. DECWRL/Livermore, 1990.

8. Weste and Eshragian, *Principles of CMOS VLSI Design*, Reading, Mass.: Addison-Wesley, 1993, pp. 326-327

9. Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, San Francisco, CA: Morgan Kaufmann Publishers, 1996, pp. A43-A44

10. Weste and Eshragian, *Principles of CMOS VLSI Design*, Reading, Mass.: Addison-Wesley, 1993, pp. 528-530

11. Weste and Eshragian, *Principles of CMOS VLSI Design*, Reading, Mass.: Addison-Wesley, 1990, pp. 326-331