Open Digital HDL to Synthesized Layout Flow for Mixed-Signal IC's

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

James Robert Copus,

* * * * *

The Ohio State University

2003

Master's Examination Committee:

Dr. Steven Bibyk, Adviser

Dr. Joanne DeGroat

Approved by

Adviser

Department of Electrical Engineering

ABSTRACT

As mixed-signal circuits become larger and more complex, manual creation of the digital parts of these designs becomes more difficult. To simplify and speed up the design process, synthesis and place-and-route tools are used to automate much of the design of the digital components. A design flow was established and analyzed using BuildGates Extreme for synthesis, and Silicon Ensemble for automatic cell placement and routing. The methodology starts with a hardware model created in either VHDL or Verilog, and the final product of the flow is a layout that can be imported into Cadence for integration into a mixed-signal design.

To support the design flow, a digital standard cell library was created using the AMI 0.5 μm process. The details of standard cell library generation are discussed, including the establishment standard cell rules and standard cell inclusion choices.

Three other university-built standard cell libraries were compared against the OSU standard cell library. All four libraries were used as a target of the design flow, and three different sample designs were created using each standard cell library. The OSU library proved to be the most flexible in terms of different types of cells, and designs created using the library usually had the smallest layout area. The library and design flow is a viable option for rapid design and layout of the digital components of mixed-signal integrated circuits in the research setting.

Open Digital HDL to Synthesized Layout Flow for Mixed-Signal IC's

By

James Robert Copus, M.S.

The Ohio State University, 2003

Dr. Steven Bibyk, Adviser

As mixed-signal circuits become larger and more complex, manual creation of the digital parts of these designs becomes more difficult. To simplify and speed up the design process, synthesis and place-and-route tools are used to automate much of the design of the digital components. A design flow was established and analyzed using BuildGates Extreme for synthesis, and Silicon Ensemble for automatic cell placement and routing. The methodology starts with a hardware model created in either VHDL or Verilog, and the final product of the flow is a layout that can be imported into Cadence for integration into a mixed-signal design.

To support the design flow, a digital standard cell library was created using the AMI 0.5 μm process. The details of standard cell library generation are discussed, including the establishment standard cell rules and standard cell inclusion choices.

Three other university-built standard cell libraries were compared against the OSU standard cell library. All four libraries were used as a target of the design flow,

and three different sample designs were created using each standard cell library. The OSU library proved to be the most flexible in terms of different types of cells, and designs created using the library usually had the smallest layout area. The library and design flow is a viable option for rapid design and layout of the digital components of mixed-signal integrated circuits in the research setting.

To my family and friends, who provided support and encouragement throughout my education.

ACKNOWLEDGMENTS

I would like to thank Dr. Stephen Bibyk for the opportunity to work in his research group and his assistance with my research and thesis, and also for agreeing to serve on my examination committee.

I would also like to thank Dr. Joanne DeGroat for her insight on digital VLSI techniques and agreeing to serve on my examination committee.

Thanks go to the Information Electronics group, especially Jason Abele, Todd James, Jason Parry, and John Fisher, for ideas, assistance, and feedback.

Thanks also go to the FEH Program, especially to Dr John Demel and Dr. Rick Freuler, for providing a place for me during a time of many changes. The faculty, staff, and students involved in the program helped make my time in graduate school enjoyable and educational.

I would also like to thank all my friends for listening to my rants and frustrations, for being there when I needed a break, and understanding when I could not take a break. Along the same lines, praise needs to go to the Cartoon Network and their late-night Adult Swim lineup for providing the background noise during long nights of writing my thesis.

Finally, I would like to thank George, Lucille, and Angela Copus for their loving support when it looked like I was becoming a permanent student.

VITA

October 21, 1977 ..... Born - Springfield, OH

June 8, 2001 ..... B.S. Electrical & Computer Eng.,
The Ohio State University,
Columbus, OH

September 2001 to present ..... Graduate Teaching Associate,
The Ohio State University,
Columbus, OH

James Robert Copus was born in Springfield, Ohio on October 21, 1977 to Reverend George and Lucille Copus. After graduation from Riverdale High School in 1996, he attended The Ohio State University, where he pursued various academic interests and became involved in multiple extra-curricular groups. After obtaining a Bachelors degree in Electrical and Computer Engineering in 2001, his interests led to graduate school research with the Information Electronics research group and teaching with the Fundamentals of Engineering Honors Program as a Graduate Teaching Associate while pursuing a Masters Degree in Electrical Engineering.

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in Information Electronics: Dr. Stephen Bibyk

TABLE OF CONTENTS

Page

Abstract ii

Dedication iii

Acknowledgments iv

Vita v

List of Tables x

List of Figures xi

Chapters:

1. Introduction to Digital VLSI Circuit Design 1

    1.1 Introduction 1
    1.2 Synthesis 3
    1.3 Circuit Layout Generation 4
        1.3.1 Programmable Logic Devices 5
        1.3.2 Gate Arrays 5
        1.3.3 Standard Cell Libraries 7
    1.4 Full-Custom vs. Automated Synthesis and Place & Route 7
    1.5 IP Reuse 10

2. Automatic Synthesis and Place & Route Methodology 12

    2.1 Design Flow 12

3. Design Flow Implementation 17

3.1 Standard Cell Library Design 17
3.2 Standard Cell Design Rules 20
3.2.1 Standard Cell Height 20
3.2.2 Inter-Cell Routing Rules 21
3.2.3 Intra-Cell Routing Rules 25
3.2.4 Power and Ground Rails 25
3.2.5 Feedthroughs 26
3.2.6 Filler Cells 26
3.3 Cell Schematic Design 26
3.3.1 Cell Inclusion Choices 27
3.3.2 Naming Conventions 28
3.3.3 Individual Cell Testing 30
3.4 Abstract Generation 30
3.4.1 Separate Abstract Library 33
3.4.2 Verilog Gate List 35
3.4.3 Synthesis Library File Creation 36
3.5 Synthesis Steps 37
3.6 BuildGates Scripting 39
3.7 Limitations of Synthesis Software 41
3.7.1 Single Signal Transition 41
3.7.2 Clauses in IF Statement 42
3.7.3 WAIT Statements 43
3.8 Place & Route Steps 43
3.8.1 Software Initialization 43
3.8.2 Floorplan Initialization 44
3.8.3 Power Planning 45
3.8.4 Pin and Cell Placement 46
3.8.5 Routing 47
3.8.6 Export 48
3.8.7 DEF 49
3.8.8 GDSII 49
3.8.9 Silicon Ensemble Scripting 50
3.9 Import Into Cadence 52
3.9.1 Import Verilog as Schematic 52
3.9.2 Import DEF as Layout 53
3.9.3 Import GDS II as Layout 54
3.9.4 Design Checking 55
3.10 Cell Libraries Used As Reference 56
3.10.1 IIT06_STDCELLS 56

3.10.2 UT_LP_AMI06 . . . 57
3.10.3 MSU_Jennings . . . 57
3.11 VHDL Sample Models . . . 58
3.11.1 8-bit Bidirectional Bus . . . 58
3.11.2 Mini-Universal Asynchronous Receiver-Transmitter . . . 58
3.11.3 AVR Microprocessor core . . . 59

4. Results . . . 60

4.1 Comparison of Standard Cell Libraries . . . 60
4.2 Synthesis Results . . . 61
4.2.1 Bidirectional Bus . . . 64
4.2.2 MiniUART . . . 64
4.2.3 AVR Microprocessor Core . . . 65
4.2.4 Post-Synthesis, Pre-Routing Analysis . . . 65
4.3 Placement and Routing . . . 66
4.3.1 Design Checking . . . 69

5. Conclusions . . . 72

5.1 Contributions . . . 72
5.2 Future Work . . . 73
5.2.1 Standard Cell Library Improvements . . . 73
5.2.2 Software Tools . . . 74
5.3 Final Remarks . . . 75

Appendices:

A. VHDL/Verilog Synthesis, Place & Route Flow Tutorial . . . 76

A.1 Synthesis . . . 76
A.2 Place & Route Design . . . 77

B. Abstract Generator Tutorial for Standard Cells . . . 80

C. Synopsis Library . . . 82

D. Symbol Library . . . 92

E. LEF Conversion for Silicon Ensemble . . . 94

F. LEF Conversion to Verilog 96

G. LEF Conversion to Synopsis LIB 99

H. BuildGates Synthesis Scripts 101

H.1 Master Script 101
H.2 Bidirectional Bus Synthesis Script 102
H.3 MiniUART Synthesis Script 102
H.4 AVR Core Synthesis Script 102

I. Silicon Ensemble Pin Constraint Format 103

J. Silicon Ensemble Place & Route Scripts 105

J.1 Shell Script 105
J.2 Bidirectional Bus Place & Route Script 106

K. Synthesis Summary Reports 108

K.1 OSU AVR Microprocessor 108
K.2 OSU Bidirectonal Bus 109
K.3 OSU miniUART 110
K.4 UT AVR Microprocessor 111
K.5 UT miniUART 112
K.6 IIT Bidirectional Bus 113
K.7 IIT miniUART 114
K.8 MSU AVR Microprocessor 115
K.9 MSU miniUART 116

L. OSU Standard Cell Library Contents 118

M. OSU Standard Cell Library 119

N. Sample Layouts 191

Bibliography 195

LIST OF TABLES

Table **Page**

4.1 Comparison of Standard Cell Libraries. 62

4.2 Comparison of Number of Cells in the Standard Cell Libraries. 63

4.3 Area Comparison of OSU Cells with Smallest in Other Library. . . . 63

4.4 Synthesis Results of Bidirectional Bus. 65

4.5 Synthesis Results of MiniUART Design. 65

4.6 Synthesis Results of AVR Microprocessor. 66

4.7 Final Design Layout Areas. 68

4.8 Core Row Utilization of all placed-and-routed designs. 70

LIST OF FIGURES

Figure **Page**

2.1 HDL to Layout Design Flow 14

2.2 Software Tools Used in HDL to Layout Flow 16

3.1 Standard Cell Library Generation 19

3.2 Different Minimum Spacing Methods 22

3.3 How Grid Offset Increases Grid Intersections. 23

3.4 Routing Grid 24

D.1 BuildGates Graphical Symbol Library 93

M.1 Symbol of ao22x1 120

M.2 Schematic of ao22x1 121

M.3 Layout of ao22x1 122

M.4 Symbol of aoi22x1 123

M.5 Schematic of aoi22x1 124

M.6 Layout of aoi22x1 125

M.7 Symbol of bufx1 126

M.8 Schematic of bufx1 127

M.9 Layout of bufx1 128
M.10 Symbol of bufx4 129
M.11 Schematic of bufx4 130
M.12 Layout of bufx4 131
M.13 Symbol of bufxz1 132
M.14 Schematic of bufzx1 133
M.15 Layout of bufzx1 134
M.16 Symbol of dff 135
M.17 Schematic of dff 136
M.18 Layout of dff 137
M.19 Symbol of dffpc 138
M.20 Schematic of dffpc 139
M.21 Layout of dffpc 140
M.22 Layout of fill1 141
M.23 Layout of fill2 142
M.24 Symbol of invx1 143
M.25 Schematic of invx1 144
M.26 Layout of invx1 145
M.27 Symbol of invx4 146
M.28 Schematic of invx4 147
M.29 Layout of invx4 148

M.30 Symbol of invzx1 149
M.31 Schematic of invzx1 150
M.32 Layout of invzx1 151
M.33 Symbol of lat 152
M.34 Schematic of lat 153
M.35 Layout of lat 154
M.36 Symbol of latpc 155
M.37 Schematic of latpc 156
M.38 Layout of latpc 157
M.39 Symbol of mux21x1 158
M.40 Schematic of mux21x1 159
M.41 Layout of mux21x1 160
M.42 Symbol of nand2x1 161
M.43 Schematic of nand2x1 162
M.44 Layout of nand2x1 163
M.45 Symbol of nand3x1 164
M.46 Schematic of nand3x1 165
M.47 Layout of nand3x1 166
M.48 Symbol of nand4x1 167
M.49 Schematic of nand4x1 168

M.50 Layout of nand4x1 169
M.51 Symbol of nor2x1 170
M.52 Schematic of nor2x1 171
M.53 Layout of nor2x1 172
M.54 Symbol of nor3x1 173
M.55 Schematic of nor3x1 174
M.56 Layout of nor3x1 175
M.57 Symbol of nor4x1 176
M.58 Schematic of nor4x1 177
M.59 Layout of nor4x1 178
M.60 Symbol of tiehigh 179
M.61 Schematic of tiehigh 180
M.62 Layout of tiehigh 181
M.63 Symbol of tielow 182
M.64 Schematic of tielow 183
M.65 Layout of tielow 184
M.66 Symbol of xnor2x1 185
M.67 Schematic of xnor2x1 186
M.68 Layout of xnor2x1 187
M.69 Symbol of xor2x1 188
M.70 Schematic of xor2x1 189

M.71 Layout of xor2x1 190

N.1 Layout of Bidirectional Bus on OSU Digital Standard Cell Library. . 192

N.2 Layout of MiniUART on OSU Digital Standard Cell Library. 193

N.3 Layout of AVR Microprocessor Core on OSU Digital Standard Cell Library. 194

CHAPTER 1

Introduction to Digital VLSI Circuit Design

1.1 Introduction

Over the last few decades, the semiconductor industry has been able to constantly miniaturize their fabrication technologies, allowing more transistors to fit on a single silicon wafer. This higher transistor density has allowed larger and more complicated digital circuits to be created. However, as integrated circuits are designed on a larger scale, it becomes almost impossible to design and verify the circuits manually. This task becomes even more difficult during the creation of mixed-signal integrated circuits, since the analog part of the design takes up a large part of the design resources.

The current methodology for creating Very Large Scale Integrated (VLSI) circuits, which are defined as a circuit containing hundreds of thousands of transistors, is to use a hardware description language, such as VHDL or Verilog, to define the behavior of a circuit. This model is then verified for functionality, and then synthesized using a software tool such as Synopsis Design Compiler or Ambit BuildGates. The synthesized circuit is usually in the form of a gate-level netlist that can be mapped to physical layouts contained in a standard cell library. A place-and-route tool, such as Synopsis Physical Compiler or Cadence Silicon Ensemble, takes the netlist and

creates a physical layout containing the standard cells and the proper wire routing between the cells. This finished layout can then be tested, either through netlist extraction and verification or through a layout- versus- schematic test that can verify that both the layout and a schematic are equivalent.

This type of design methodology allows a top-down design flow, where the overall specifications can be created, and large digital blocks can be instantiated at a behavioral, dataflow, register transfer level, or structural level. These blocks can then be synthesized and easily reused in other designs to even further speed up the design process. The low-level problems of cell placement and routing can be handled by automated software, and further refinement can be performed to achieve timing convergence with the original specifications.

This document outlines a design flow to generate a working semiconductor layout from a hardware description language model. The layout could then be easily integrated into a mixed-signal layout using Cadence, a widely-used tool in analog and mixed-signal circuit design. A digital standard cell library was created in support of this design flow. The flow Chapter 1 provides a brief background of digital VLSI circuit design techniques. Chapter 2 outlines an automatic synthesis and place-and-route methodology, and Chapter 3 describes in detail the steps used to create and use this flow. The construction of the standard cell library is also described. Chapter 4 discusses the results of some analyses of the standard cell library and design flow. Chapter 5 draws some conclusions and offers ideas for future work in this area.

1.2 Synthesis

Hardware description languages are used to model and simulate hardware designs at different levels of abstraction: from a behavioral or algorithmic level to a gate-level logic or register transfer level (RTL). Two languages are commonly used to model hardware: VHDL and Verilog. VHDL was first developed for the Department Of Defense VHSIC (Very High Speed Integrated Circuits) groups. It eventually became an IEEE and ANSI standard. The language was updated in 1993 to add more features and clarifications of previous features [1]. The Verilog Hardware Description Language was created by Gateway Design Automation in 1984. Gateway Design Automation was later bought out by Cadence Design Systems in 1989, and Cadence later made the Verilog language an open standard. Verilog became an IEEE standard in 1995. [2] Both of these languages are widely used, with VHDL used heavily in defense work, and Verilog in industry, although both languages are used in both sectors.

Synthesis is the process of construction a gate-level netlist from a model of a circuit described in a hardware description language. Since VHDL is primarily designed to be a hardware simulation language, not all VHDL models can be synthesized into a netlist, or even realized into hardware. Different synthesis tools support different subsets of VHDL, and may require a specific modeling style to properly synthesize a circuit. [1]

Current synthesis software can take a model in a hardware description language and synthesize it into a physical design. The tool can then optimize that design for minimal area or speed requirements, according to the timing and size characteristics of the target technology. Many synthesis tools can also add automatic test circuitry such as scan chains or built-in self-test logic.

1.3 Circuit Layout Generation

Automatic placement and routing algorithms for digital standard cells advanced to take advantage of more complex semiconductor processes. When only one or two metal layers were available for routing, the software tools usually required that all routing occurred around the digital cells, and connection pins had to be at the cell boundaries. Cells were usually of different cell heights, so care had to be taken so that power and ground connections lined up correctly. Since all inter-cell routing had to occur between the cells (as opposed to through or over the cells), a large proportion of the layout area was consumed with routing. As more routing layers became available, more complex algorithms were developed to utilize space more efficiently, and over-the-cell routing became widely used.

More than one way has been developed to speed up the VLSI circuit design and production process. Programmable logic devices, such as field-programmable gate arrays (FPGA's) and complex programmable logic devices (CPLD's) are a fast way to make small amounts of digital chips, at the expense of circuit speed and area. Gate arrays depend on part of the semiconductor being prefabricated to allow fast turnaround times. Standard cell based designs provide the most flexibility, and highest circuit speeds of the automated methods, at the expense of slow turnaround times and high initial fabrication costs. These methods provide different speed, production times, and price points. The choice depends on the specifications and constraints of the design and project.

1.3.1 Programmable Logic Devices

Programmable Logic Devices are integrated circuits that contain logic blocks or gate arrays that can be connected in different ways to implement logic gates and memory to produce digital circuits. The devices can be mass-produced, and then programmed by the end user. This provides a cheap way to produce a small number of devices and eliminates fabrication time, since parts can be kept on hand and programmed when needed. However, they are slower and less area-efficient than ASICs. They are ideal for individual projects, design testing for ASICs, and low volume production. Many software tools exist to easily synthesize HDL models and automatically program FPGA's and CPLD's.

1.3.2 Gate Arrays

The gate array method of producing integrated circuits attempts to significantly shorten fabrication time by prefabricating part of the circuit, and adding custom routing to the chip to produce a unique design. A layout is initially covered in pairs of transistors spread out at predefined intervals that can be interconnected with wires on multiple layers. Depending on how the wires are connected, different designs can be produced from a prefabricated transistor image. In many cases, this can reduce design time and fabrication cost, since the metal layers can be added to the partially fabricated chip kept in stock relatively quickly. This decreases the turnaround time for producing custom ASICs, at the expense of lower transistor densities and slower circuit speeds compared to custom layouts or standard cell layouts. The gate array method is also useful because transistors can be connected quickly without much

knowledge of the design rules of the process. This makes design significantly faster than full-custom layout.

The defining characteristic of the standard gate array method, as opposed to the sea of gates or path programmable logic method, is that standard gate arrays have channels on the silicon wafer left vacant between rows of transistors. These channels provide room for routing. This was important for processes that only had a few metal routing layers. However, the channels decreased the density of transistors on the chip.[3]

A way of increasing the layout density of a gate array is by using the "sea of gates" method. This method does not use channels between rows of transistor pairs. Usually, a transistor pattern is created, and over-the-cell routing is used to connect them to form gates. The prefabricated transistor image can make production cheaper and faster, in the same manner as the standard gate array method. [4] [5]

A similar approach to the "sea of gates" methodology is the "sea of wires" or path programmable logic design approach [5]. Instead of a predefined transistor layout that is routed to create the circuit, the layout is initially covered with a horizontal and vertical grid of wires that can be connected together with transistors. A section of the grid that contains a subset of the wires is called the unit cell. The unit cells are considered blank cells when they do not contain any transistors. These blank cells are replaced with cells that add functions, similar to the standard cells used in standard cell logic design. These logic cells have predefined pins that connect to the wire grids, and depending on how the transistors are placed, different logic functions are created. The benefits of this methodology is that a layout can be quickly be created, and have functionality only mildly worse than full-custom design. The drawback

is that, as silicon process technology has progressed, there are more metal routing layers, making routing more complex. Other methods have proved more popular and efficient since this technique was first proposed.

1.3.3 Standard Cell Libraries

The approach most widely used method for producing large high-speed digital designs is the standard cell methodology. Custom blocks are produced that implement common logic functions and memory elements. Large, complex circuits are made up of combinations of these blocks, which are then connected with metal routing. Standard cell design methodologies will be discussed in more detail in Chapter 3.1. The advantage is that cells can easily be placed and routed, either by hand or using automatic tools. The standard cells have high transistor densities and good speed and power characteristics. Automatic cell placement and routing allows much higher speeds in a smaller area than reprogrammable devices or gate array technologies. Manual design and layout methods are still superior, at the expense of an extremely large design time.

1.4 Full-Custom vs. Automated Synthesis and Place & Route

Before automatic synthesis and routing tools became common, it was common to spend many man-years producing a large working digital chip using full-custom design and layout tools. Automatic synthesis and routing software tools have drastically decreased the design and verification time for integrated circuits, at the expense of speed, area, and power dissipation. As tools have improved, however, the gap between custom circuit design and layout and automatic circuit design and layout has narrowed.

Studies have been published over the years comparing automated logic synthesis and routing to full-custom circuit design. One such study used simple gates from a standard cell library to synthesize some small logic circuits such as ripple-carry adders and reduction-tree multipliers using BuildGates and placed and routed the design using Silicon Ensemble. [6] The results were compared to full-custom designs of the same circuits. It was found that although full-custom designs are generally smaller, faster and more power efficient than automated designs, the design time using automated software tools was 50% to 75% shorter. This is a significant benefit. It was also found that Silicon Ensemble could perform placement and routing as efficiently as a custom designer for designs with inter-bitslice connection, but full-custom designs could reach higher performance through the use of advanced circuit techniques. This increased performance could lower the slowest-path delay by 50% to 70%. The place and route tools, such as Silicon Ensemble, were expected to perform better at exploiting the freedom and complexity of using a large number of metal layers, such as in a six- or eight-layer metal process technology.

Intel discussed automatic routing tools in 1988 that handled the merging of channels for better channel utilization [7]. The placing back-to-back of abutted rows of standard cells was called "doubleback." Every other row was inverted, so that power and ground rails could be shared, and well areas could be combined. Previously, in silicon process technologies that contained only a single metal layer, all cell connection points were placed at the borders of the cell. The addition of more layers to the process technologies allowed over-the-cell routing, so pins could be placed at any point in the cell. The Dense Auto Place and Route (DAPR) software tool was able to route to pins anywhere inside of a cell, and could handle multiple pin connection

points. The routing technique used by the software tool was called Channel Routing, and it was gridless, with pins that were any size or location, Metal2 and Poly pins could overlap, and there was no standard cell height. Even though there was no standard cell height, the power and ground lines had to be continuous when the cells abutted and overlapped. Analysis of the routing software results showed that their techniques produced designs with 20% to 25% larger area than full custom designs, but the automatically generated circuits had much greater throughput.

Another more recent study compared full-custom design to standard cell design, but with an emphasis on low power VLSI [8]. A cross between full-custom and automated design was analyzed by exploiting the fact that digital filters have regular structures, so the "full-custom" design can be readily automated with the help of layout generators and scripts. It was found that the ability to introduce custom tailored cells in critical places was advantageous in the reduction of power dissipation. The custom designed circuit using low power techniques was found to consume about one-fourth the power of a standard-cell circuit. However, custom designed circuits were found to take more time to produce, even with the automatic generation of regular structures.

These studies have shown that, although full-custom designs are usually superior in speed, power dissipation, and area optimization, automated techniques greatly speed up the design time by up to a factor of four. Also, it is possible to produce much more complex designs while using less design resources than full-custom techniques. As synthesis and place-and-route tools have improved, the disadvantages of using standard-cell synthesis have decreased. In fact, automated routing tools can

take better advantage of the availability of multiple metal routing layers, making automated designs superior to full-custom designs in some cases. Continuing research work focuses on further improving automatic design, synthesis, and layout generation methodology and tools. Work has been done to improve peak performance, reduce timing overhead, lower power dissipation, and develop new synthesis and layout algorithms [9].

1.5 IP Reuse

The ability to reuse intellectual property (IP) in chip designs speeds up overall production times. Companies generally reuse large blocks of digital or analog circuits in many designs. This lowers design times, and gives greater confidence in the correctness of the circuit, since many components have been successfully verified and fabricated in other circuits. If an IP block has been properly characterized for power and timing information, it becomes easy to put together various blocks for certain purposes with a reasonable confidence in the outcome. However, companies usually only reuse IP internally, or patent and license the designs to other companies. This means that many people have to redesign common digital cores from scratch if they cannot obtain a commercial equivalent.

A free source of open digital IP cores is the OpenCores Project, found on the World Wide Web [10]. The object of the OpenCores Project is "to design and publish core designs under a license for hardware modeled on the General Public License (GPL) [11] for software. " The cores are primarily designed for use in FPGAs, which are relatively cheap for small scale use. These cores should also work well for ASIC designs. Each design has its own license, which is usually modeled after the GNU

GPL or another open license. The AVR microprocessor core [12] and the miniUART [13] circuit, which have been synthesized as part of the analysis of the Ohio State University digital standard cell library, are both published on the OpenCores website.

The advantage of an open repository of IP cores is that digital cores can be quickly added to a larger design without having to design the implementation of that core, or paying another company to provide the design. Of course, there is no assurance the core will work, even though someone else may have tested it. Nor is it necessarily the most efficient design, since individuals, companies, students, and hobbyists of varying abilities and motivations provide the designs without charge. However, since the designs are provided in a hardware description language, anyone can verify the design and check to see if it is usable for their purposes. Another problem is that project documentation may be scarce or non-existent. Some projects have quite detailed documentation and data sheets, such as with the miniUART project. Others may have only a couple pages describing the basic functionality of the design.

The licensing schemes of the designs provided by Open Cores also vary. Some have no limitations, meaning that one can copy or modify the design code freely, and produce digital hardware with no restrictions. Other licenses may require that modifications be resubmitted to the Open Cores project so that the core can possibly include the modifications. One has to carefully decide if the license is acceptable before using the specific core.

CHAPTER 2

Automatic Synthesis and Place & Route Methodology

This chapter outlines the HDL-to-layout design flow, which will be implemented in Chapter 3.

2.1 Design Flow

The design flow proposed and analyzed in this document assumes that a functionally verified hardware model has been created in a language such as VHDL or Verilog. The design flows used in the current research environment at The Ohio State University have well-established methods of design, simulation, and verification of hardware descriptions using HDLs. The Mentor software suite, including ModelSim, is a widely used and well-understood toolset for designing and simulating hardware descriptions, although Cadence also contains various tools for VHDL and Verilog design and simulation.

The layout end of the design flow also has a well-established and well-understood set of tools for design and analysis. Magic and Cadence Virtuoso can produce custom semiconductor layouts. Various derivatives of Berkeley Spice, Irsim, and Cadence Analog Environment can simulate and test transistor-level designs.

The high-level methods used in digital ASIC design flows and the low-level tools used in digital ASIC design flows are both highly used in the OSU research environment. However, the tool-flow needed to automatically produce suitable layouts from hardware descriptions has not been well-documented in this research environment.

A working design flow was developed to bring the two separate flows together. Software tools were configured and the appropriate libraries were developed so that a working hardware description written in either VHDL or Verilog could be transformed into a working transistor layout ready for fabrication. An overview of the design flow is shown in Figure 2.1. A hardware description goes through the process of synthesis, which turns the hardware description into a physically-realizable circuit design. A layout is then generated from the design, and the layout has all necessary logic cells placed and wires routed between them, producing a finished layout.

A more detailed description of the design flow is shown in Figure 2.2. This toolflow starts where a typical large-scale HDL design flow ends: with a valid design written in a language such as VHDL or Verilog. The design is put through a software synthesis package, in this case BuildGates Extreme. The synthesis stage analyzes the design and forms a gate-level circuit that is equivalent to the original design. This new circuit can be optimized and mapped to a standard cell library so that a physical circuit can eventually be created. The synthesized design is exported as a Verilog netlist, which contains a listing of all inputs and outputs, all logical cells used, and how the cell pins are connected. This netlist is imported into the place-and-route software, called Silicon Ensemble. Silicon Ensemble creates a layout containing all the standard cells used in the design, plans power, ground, and any clock information provided to it. Then the tool routes all necessary wiring to connect the cells and input and output

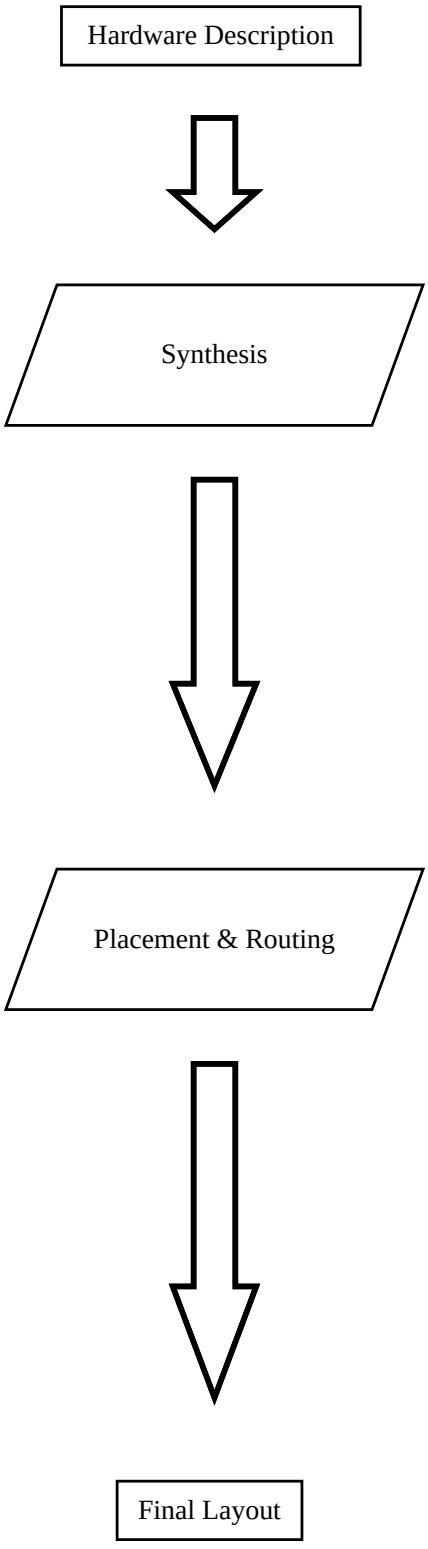

Figure 2.1: HDL to Layout Design Flow

pins. When the design is successfully routed without any errors, exported using a special Design Exchange Format, and imported into Cadence's Virtuoso software to verify the layout, or combine with any custom-designed layouts. The finished circuit can then be sent to a foundry for fabrication.

The specific target of this flow is the AMI 0.5 micron process provided as part of the MOSIS Educational Program [14]. This process is the smallest process available for free to integrated circuit design classes at educational institutions. A standard cell library was created to use with the target process, and libraries based on the standard cell library provided the physical cells the synthesis and place-and-route tools used in the design flow. The standard cell library was compared against three other research-oriented digital standard cell libraries using the same toolflow.

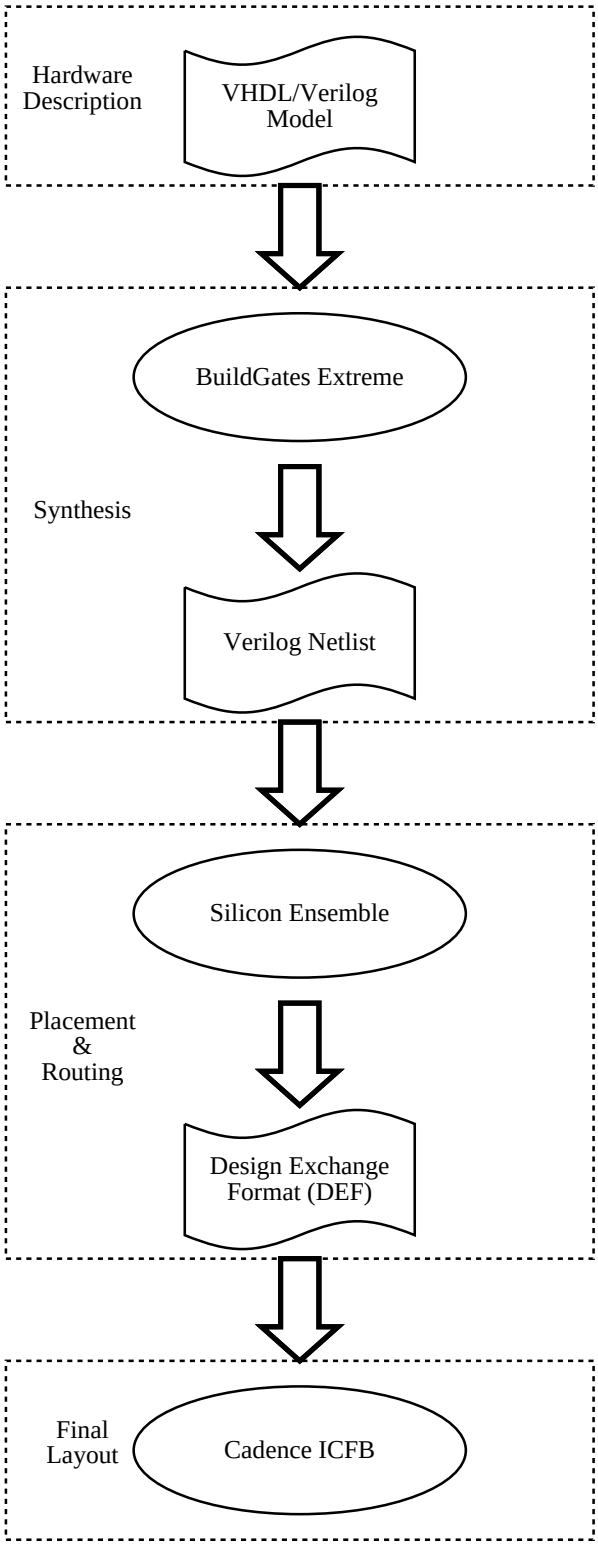

Figure 2.2: Software Tools Used in HDL to Layout Flow

CHAPTER 3

Design Flow Implementation

To implement the design flow outlined in Chapter 2, a standard cell library was created, as described in section Section 3.1. Support files were created so that the library could be used with the software tools needed for synthesizing a hardware model and generating a semiconductor layout. This is described in Section 3.4. Then, the step-by-step operation of the software tools was documented in Sections 3.5 and 3.8. Finally, other standard cell libraries to be used for comparison to the newly created library were chosen, along with sample designs. These are described in Sections 3.10 and 3.11.

3.1 Standard Cell Library Design

There were certain goals to be met in the design of a new digital standard cell library. The first was to minimize the physical design of the cells, since the primary use for the library was to be used in the MOSIS TinyChip program for student projects. The AMI 0.5 micron process is the smallest process available for student projects, and the die size available is 1500 microns on a side, and only 900 microns on a side after a standard padframe is placed. The small cell area was achieved by using a minimum-sized standard-cell height and aggressive intra-cell routing that utilized

two metal layers to minimize the cell widths. The second goal was for the library to be reliable, since the cells may be re-used in many different projects. To increase the reliability, the original cell schematics were based on the Tanner Library [15], on which the University of Tennessee based their standard cell libraries.

To be useful, the standard cell library had to be available for use in different software tools, such as the synthesis tools and place-and-route tools. The steps used in the standard library file generation is shown in Figure 3.1. Schematics are generated for each logic cell used in the library. A layout is also created according to the standard cell design rules discussed later. Once the schematic and layout for every cell is proved equivalent using the Layout-Versus-Schematic (LVS) tool in Cadence, abstracts can be generated from the layout. The Abstract Generator tool extracts only the information necessary for the place-and-route tool, and creates a version of the layout that only contains this information. The abstracts are automatically saved in the Cadence ICFB library containing the standard cells. The abstracts, along with information about the process technology, are exported into a Library Exchange Format (LEF). A script were created to convert this LEF file into a slightly different LEF file compatible with the Silicon Ensemble Place & Route tool. A script was also created to take the LEF file information and generate a Verilog gate listing for use in the Silicon Ensemble tool. A third script takes the LEF file information and creates most of the information needed to generated a Synopsis synthesis library (LIB) file. Once the rest of the logic information about the library is manually entered, it can be compiled into the Ambit Library Format (ALF) that BuildGates Extreme requires for synthesis to a standard cell library.

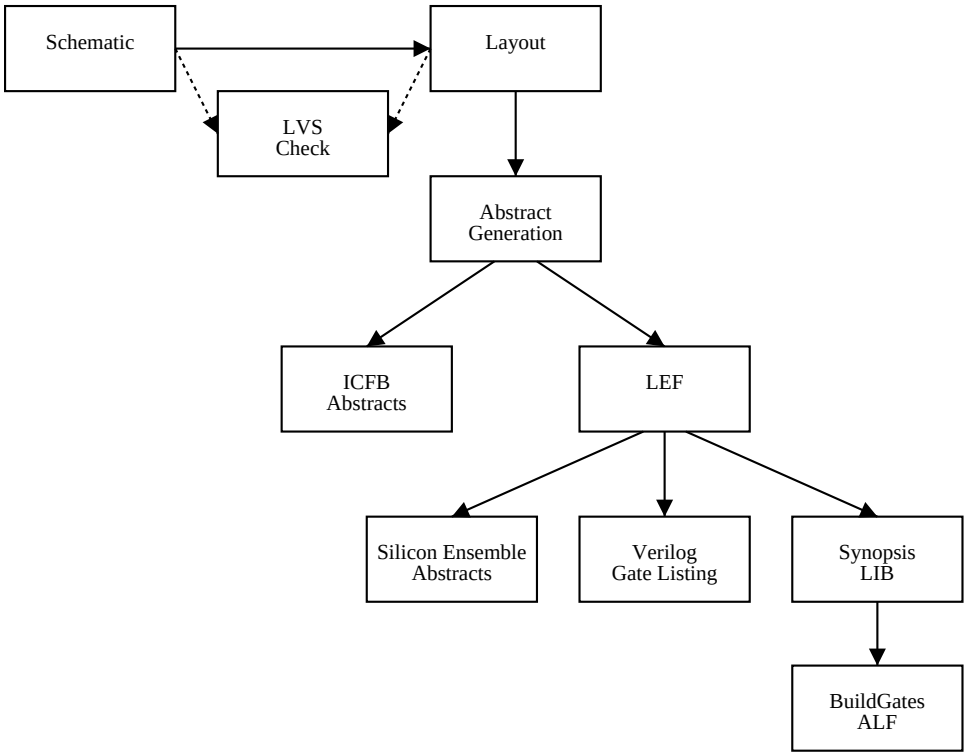

Figure 3.1: Standard Cell Library Generation

Certain rules had to be set up to be able to produce a set of consistent cells that could be abutted and flipped upside down, thus requiring no space between cell rows and allowing cells to share power and ground rails with the cells above and below. Also, the AMI 0.5 micron design rules, available through the MOSIS foundry, had to be followed. Cell design strategies used by Virginia Tech [16], Mississippi State [17], and the University of Tennessee were studied in the formulation of the Ohio State standard cell library.

As can be seen, there are multiple digital standard cell libraries available for the research community built for the AMI 0.5 micron process. The goal in designing yet another cell library was to produce a full-featured set of logic and minimize the size of each individual cell. The main target of this standard cell library is student class and research project that can be fabricated using the MOSIS TinyChip educational program [14]. The limitations of the other standard cell libraries became evident when they were compared to the Ohio State standard cell library. To minimize the size of the standard cells, certain strategies were developed, such as limited internal use of a second metal routing layer. The standard cell library design rules are described in detail in the next section.

3.2 Standard Cell Design Rules

3.2.1 Standard Cell Height

A single standard cell height was defined to allow simple placement of cells. Having a single cell height makes it possible to place cells side-by-side and have the power and ground rails automatically line up at the top and bottom edges, respectively. The height was defined to make room for NMOS and PMOS transistors and the intra-cell

routing to connect the transistors. The height was also devised to be a multiple of the horizontal routing grid spacing, which will be discussed next.

3.2.2 Inter-Cell Routing Rules

To make wire routing faster and easier for automatic routing software, a regular routing grid was devised with horizontal and vertical grid spacing optimized so that parallel wires and their vias are at a minimum distance according to the process design rules. The routing rules were defined so that Metal1 and Metal3 were to be used for horizontal routing, and Metal2 was to be used for vertical routing. The metal layers have different spacing and pitch rules, so different spacing was required for the horizontal and vertical gridlines.

The minimum spacing between parallel routes on the same metal layer could be defined in different ways. The spacing depends on how, or if, vias need to be used in metal routes. The different ways to define wire spacing are shown in Figure 3.2. The actual spacing between the wires is called *line-on-line* spacing, meaning the minimum spacing is defined by the minimum pitch between two wires, assuming no vias. If there were no need to connect between layers of metal, line-on-line spacing could be used. This method is not feasible for a standard cell library, since vias need to be used during routing. The second type of spacing is *line-on-via* spacing. This is the minimum spacing between a via and a metal route, according to the process technology design rules.. This could be used for routing in a standard cell library, but the vias on parallel sets of wires would have to be staggered so that the distance between vias did not violate the design rules for the process. This also could be done, but would complicate routing algorithms and possibly make standard cell design more

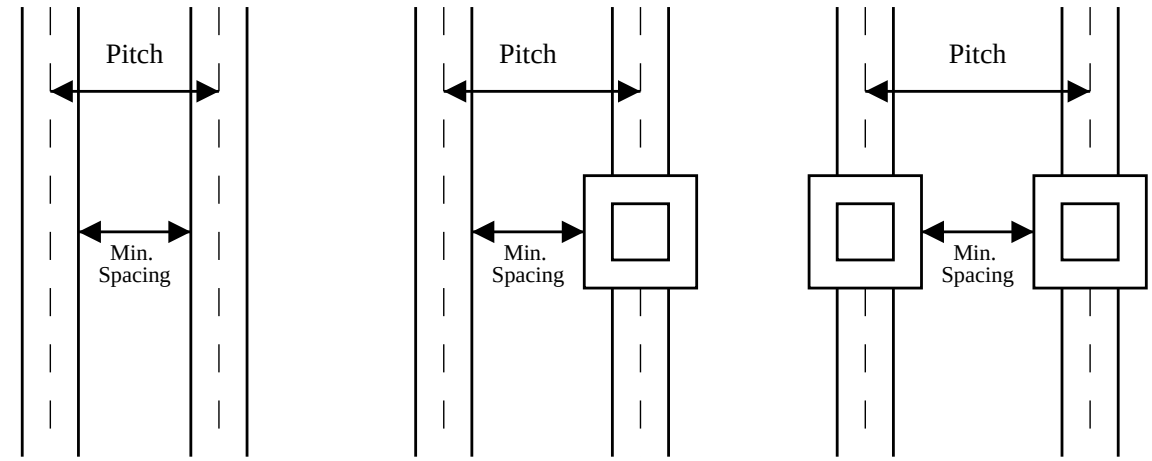

Figure 3.2: Different Minimum Spacing Methods

complex. The third way of defining the minimum spacing between metal routes is *via-on-via* spacing. In this case, the minimum spacing is defined by the minimum distance allowed between two vias on a layer, according to the design rules. Via-on-via spacing is used for most standard cell libraries, since it allows vias to be placed next to each other on parallel routes. The downside of this method is that it makes the wiring slightly less dense than with line-on-via spacing.

Using the minimum via spacing rules defined for the AMI 0.5 micron process, the via-on-via pitch determined that the vertical gridlines on the standard cells would be spaced 2.4 microns apart and the horizontal gridlines would be spaced 3.0 microns apart. The standard cell height is then a multiple of the horizontal grid spacing. Each individual cell may have a different cell width, but it must be a multiple of the vertical grid spacing.

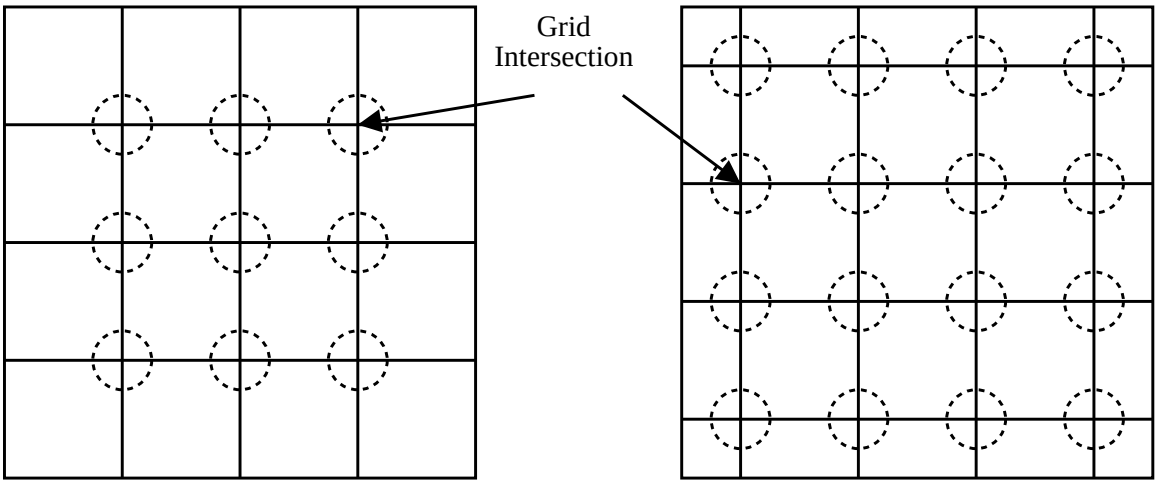

Figure 3.3: How Grid Offset Increases Grid Intersections.

An offset was added to the standard cell grid in order to allow more room for grid intersections inside the cell, and therefore more locations for pins on the grid. Figure 3.3 shows how the number of intersections are increased with the addition of an offset. In the figure, both grids are of the same area, but the second grid has a half-grid offset in each directions. This increased the number of available intersections from nine to sixteen. Each intersection of the vertical and horizontal gridlines is a location on which a pin or via can be located. The via-on-via spacing for both directions allowed unlimited via stacking at those points, meaning that one metal layer can be connected to any other at a grid intersection using multiple vias.

A diagram of the routing grid, along with the power and ground rails, is shown in Figure 3.4.

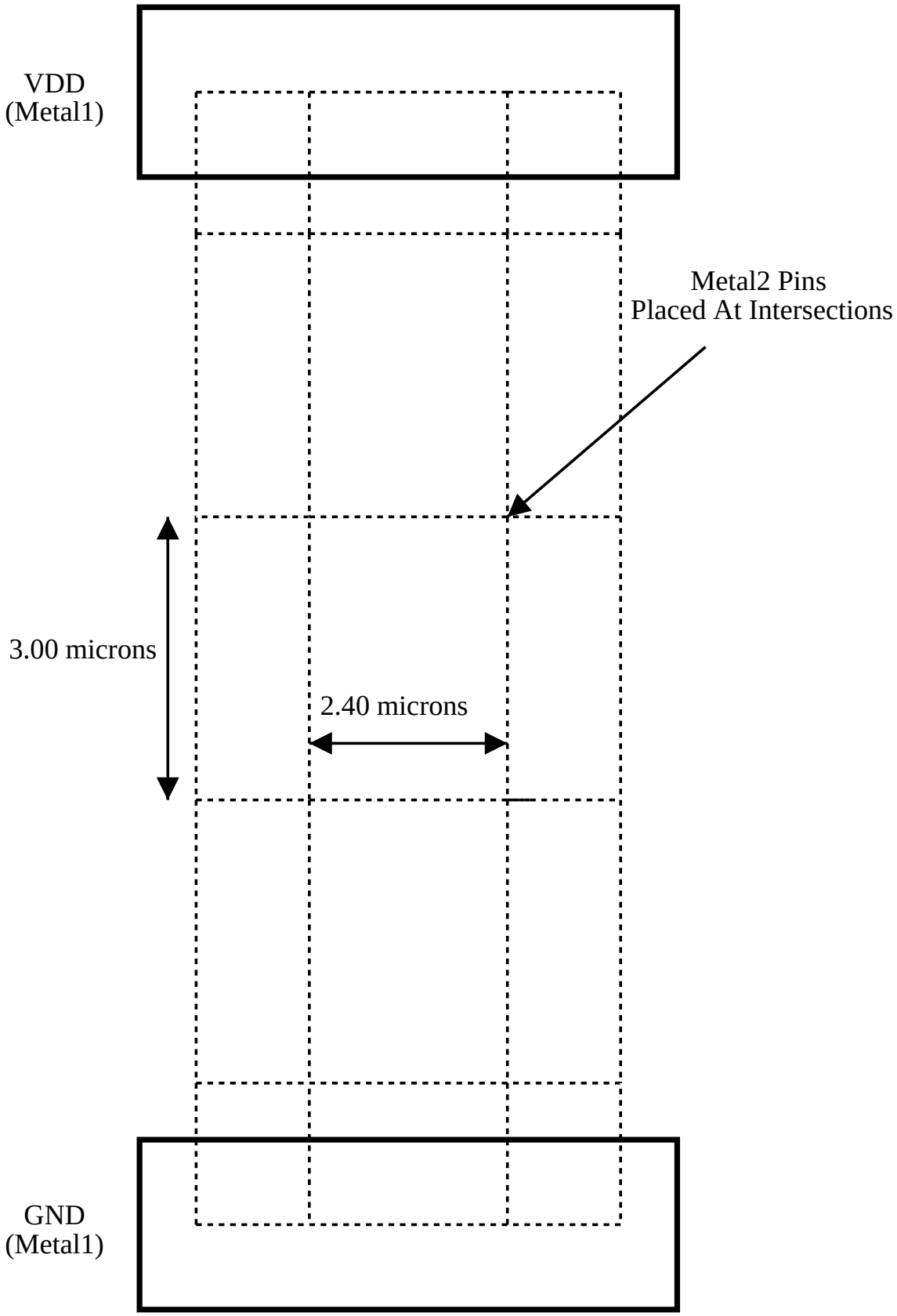

Figure 3.4: Routing Grid

3.2.3 Intra-Cell Routing Rules

Routing between transistors inside of a cell do not require that metal layers go along the routing grids. In fact, some standard cell libraries choose to route a metal layer in the opposite direction inside of a cell from the routing direction between cells. This makes placing vias easier for the automatic routing tool, and is a preferred method when many metal routing layers are available. [16] Routing in this manner can cause over-the-cell routing congestion for a process with three or less metal routing layers, however. This technique was avoided to allow as many paths through a cell as possible.

To maximize the amount of routing room between cells, Metal3 was not used in any of the standard cells, and Metal2 was used as little as possible. Metal1 was used extensively inside the cells, so the automatic routing tool will rarely use Metal1 to route inside of a cell, except in the case of routing power and ground signals and the connections of cells in adjacent rows. In many other university standard cell libraries that use a three-metal layer process, the second routing layer is completely avoided to maximize the inter-cell routing paths[16]. It was decided to allow some Metal2 inside the cells in an attempt to minimize the cell sizes, at the expense of vertical feedthrough paths on the routing grid. This tradeoff of less routing paths for smaller cell area was acceptable for certain designs that required a small layout area and had relatively few inter-cell wires.

3.2.4 Power and Ground Rails

Power and Ground rails were placed horizontally at the top and bottom, respectively, of all standard cells. These rails actually extend outside of the defined standard

cell boundary so that the cells in rows above and below overlap and share common power and ground rails.

N-taps and P-taps are placed in a regular pattern under the rails so that NMOS transistors have their bulk tied to ground and PMOS transistors have their bulk tied to power. These taps also are shared between cells in abutting rows.

3.2.5 Feedthroughs

Some standard cell libraries add extra room for dedicated feedthrough paths through the cells [18]. These are paths through the cells with no obstructions or logic pins. This is important for processes with only a few metal layers, but was eliminated for this process, since minimizing cell area was a major goal.

3.2.6 Filler Cells

Not all cells placed in a layout will be abutted with another logical cell. To ensure N-well and power and ground rail continuity, filler cells are usually placed between other standard cells cells in a layout. Filler cells contain no transistors, but do contain an N-well and power and ground rails using the same design rules as the rest of the standard cell library. The width of the standard fill cell is the same as the minimum spacing of the vertical routing grid, 2.4 microns. Fill cells can be added to the layout by the automatic place-and-route software after the placement of the logic cells and before wire routing occurs.

3.3 Cell Schematic Design

Schematics were created using Cadence Virtuoso for each logic cell used in the standard cell library. The schematics became the basis for the cell layouts that

were created according to the previously specified standard cell design rules. The design methodology for the cells were taken from standard complementary CMOS logic design methods commonly used in digital design [19].

3.3.1 Cell Inclusion Choices

To properly synthesize circuits, certain logic cells had to be included in the library. The 2-input NAND gate could be used to synthesize any other combinatorial logic, and a clocked memory element, such as a flip-flop, would be required to implement a state machine.

To provide more efficient cell synthesis, NAND gates with 2, 3, and 4 inputs were added, along with 2, 3, and 4 input NOR gates. Two-input XOR and XNOR gates were also included. More complex combinational logic functions were created, such as the AND-OR and AND-OR-INVERTER gates that act as compound gates. A two-input multiplexor was included to provide further flexibility.

Inverters with different drive strengths were added to allow complementary inputs and outputs. Buffers were also created with different drive strengths to provide better circuit performance. To allow tri-state circuits, an inverter and a buffer were included that were able to have a high-impedance output, so that devices such as busses could be synthesized under the standard cell library.

State-holding elements were designed for the standard cell library. D flip-flops were included so that the library would have clocked memory elements, and latches were included for designs that do not need the memory elements to be clocked. Two types of D flip-flops were added: one with a preset and clear and one without. Two types of latches were added: one with a preset and clear and one without. The added

circuitry for preset and clear on both the flip-flops and latches increase the cell area, so the synthesis tool will try to use the versions of the cells without preset and clear, if possible. There was no random access memory (RAM) element included in the cell library, since a separate RAM compiler is used to generate large blocks of memory more efficiently than a typical synthesis tool. The memory elements included with the standard cell library were designed to be used in such memory elements as state machines and data registers.

Cells were also added to tie an input to logical '1' or '0'. The "tie high" and "tie low" cells force the input of another cell to logical '1' or '0' without shorting the input directly to the power or ground rails. This is useful when the inputs to a WHY?

3.3.2 Naming Conventions

Naming conventions were specified in order to create a consistent set of logic cells. The naming conventions were applied to standard cell names, along with the names of all input and output pins.

Standard cells were named according to their function, and in some cases the names specified their relative drive strength. For example, a two-input NAND gate with minimum-sized transistors (and therefore single drive strength) was named **nand2x1**.

Compound gates, such as the AND-OR-INVERTER compound gate, had simplified names that implied the function of the cell and the layout of the inputs and outputs, such as **aoi22x1**.

Memory cells, such as flip-flops and latches, had the cell type and any extra functionality included in the cell names. A D flip-flop with preset and clear was named **dffpc**, while a D flip-flop without preset or clear was named **dff**.

The input and output names for all the standard cells also followed standard naming conventions. Inputs to simple logic gates were named alphabetically, starting with *A* and increasing in alphabetical order for each additional input. Outputs of simple logic gates were labelled *Y*.

The power and ground pins were labelled *vdd!* and *gnd!*, respectively. The exclamation point after the pin names signifies that they are global pins to Cadence Virtuoso. A global pin has an implied connection to all other global pins of that name in a circuit. Global pin names were needed so that Cadence would recognize the power and ground pins in schematics made from an imported Verilog netlist. When those pins were declared as being global, the power and ground nets did not have to be explicitly connected in the original Verilog netlist. This was important because the imported schematics were used to verify that the final design layouts matched the synthesized Verilog design.

Special-purpose pins were given names that implied their purpose. These were:

CLK Clock signal input pin.

CLR Clear signal input pin.

D Data input pin of a memory cell.

EN Input pin that enables an output.

HI Pin with constant output of logical "1.'

LO Pin with constant output of logical "0.'

PRE Preset signal input pin.

QN Negative output pin of a memory cell.

QP Positive output pin of a memory cell.

Sel Select input pin on a multiplexor.

TriState Input pin that enables a high-impedance output.

A complete listing of all standard cells is found in Appendix L. The complete standard cell library, including schematics and layouts is found in Appendix M

3.3.3 Individual Cell Testing

Each cell layout in the library was checked for design rule violations using the DRC check included with Cadence Virtuoso. Then, each layout had a netlist extracted and compared to the original schematics. The Cadence Layout Versus Schematic (LVS) tool was used to verify that the extracted layout matched the schematic. Some informal testing occurred by placing layouts back-to-back in abutted rows to check for DRC violations arising from improper overlaps and spacing between cells.

3.4 Abstract Generation

The Cadence Abstract Generator was used to create abstract cells of the layouts. Abstract cells contain only the information that the place-and-route tool needs to properly connect pins and avoid obstructions while routing. The NCSU AMI06 Technology Library available and widely used with the OSU Cadence system does not

natively handle abstract generation with the Abstract Generator tool. Some changes specific to placement-and-routing parameters found in the technology file had to be made. The Cadence documentation provided background information on what technology file options existed [20]. The original NCSU technology library was dumped into and ASCII .tf format techfile. Place and Route rules were appended to the original file. The classes added to the technology file were:

```
prRoutingLayers(
;  ( layer                    preferredDirection  )
;  ( -----                    ------------------  )
   ( poly                     "vertical"   )
   ( metal1                   "horizontal" )
   ( metal2                   "vertical"   )
   ( metal3                   "horizontal" )
) ;prRoutingLayers

prViaTypes(
 ; ( (device    cellViewName)   viaType )
   ( (M1_POLY   symbolic)       "default" )
   ( (M2_M1     symbolic)       "default" )
   ( (M3_M2     symbolic)       "default" )
   ( (via       symbolic)       "default" )
   ( (via2      symbolic)       "default" )
 ) ; prViaTypes

 prRoutingPitch(
 ;( layer                pitch )
  ;( -----                ----- )
   ( poly                 1.800 )
   ( metal1               3.000 )
   ( metal2               2.400 )
   ( metal3               3.000 )
  ) ;prRoutingPitch
```

The prRoutingLayers() class defined the routable layers and specified the preferred routing direction for each layer. The polysilicon layer had to be included for compatibility with the abstract generation tool, but was not listed as a valid routing layer in the final library files used in the place-and-route tool. The horizontal-vertical-horizontal or "HVH, routing preference was specified for the metal layers, meaning that the Metal1 layer would route in the horizontal direction, the Metal2 layer in the vertical direction, and the Metal3 layer in the horizontal direction.

The prViaTypes() class listed all the valid vias between routing layers, such as M1_POLY, M2_M1, and M3_M2 vias. The prRoutingPitch() defined the minimum pitch between parallel routes of the same metal type. This pitch depends on the minimum distance between two different metal routes in the same layer as defined in the design rules for the technology, and also the minimum width of the routing wires and vias in the technology. These classes were defined in the technology file, along with the removal of outdated and unused *cp*, *ca*, and *ce* via types, which also caused some compatibility problems with Abstract Generator.

The altered technology file was uploaded as a new library called *ami05_abstract_friendly* and the digital cell library properties were changed to be referenced to the new technology library, instead of NCSU_TechLib_ami06. The original technology file could have been altered to be compatible, but many different projects depend on this technology library. It was safer to create a new temporary library for the abstract generation process, in case the changes to the technology file broke compatibility with other projects using the library. Abstracts could be created from the finished digital cell library using the algorithm shown in Appendix B derived from the Cadence Documentation [21].

The abstract generation process resulted in a Library Exchange Format (LEF) file. However, this file was still not compatible with Silicon Ensemble. A Perl script was developed to alter the abstract LEF file so that it would import correctly into Silicon Ensemble. This script, found in Appendix E, was inspired by the Rice University's Cadence Tutorials [22], however, the types of changes the LEF required were almost completely different than what Rice's scripts provided. First, nwell, nactive, and pactive information needed to be added to to the LEF, for software compatibility

reasons. The polysilicon layer was changed from a routing layer to a masterslice layer, meaning Silicon Ensemble would know the layer existed, but would not try to route with it. This was to prevent long, high-resistance polysilicon wires from being used to connect pins, which would potentially hurt the circuit performance. Since the elec layer was not used for any of the standard cells, the M1_ELEC via was removed from the library for Silicon Ensemble compatibility reasons. Silicon Ensemble did not know the ELEC layer existed, nor was it desired to use the layer for routing. Via generation rules were added to the file to allow the generation of more complex vias, such as for the power ring, which is usually wider than a standard routing wire, and needs more than one simple via for Silicon Ensemble to generate the ring correctly. Finally, the sites "Core" and "CoreSite" were added to the LEF. Sites define the type of cell in a library. The abstract generator created cells of those types, and the cells type was listed in each of the cell definitions. Therefore, the definitions of Core and CoreSite, along with the symmetry of the cell types and the base sizes of the cell types was included for Silicon Ensemble.

The final LEF file could be imported into Silicon Ensemble for the placement-and-routing phase of the design flow. A few more library files were created from this LEF, such as a Verilog file containing a standard cell gate listing and a Synopsis-compatible synthesis library.

3.4.1 Separate Abstract Library

Due to some odd quirks in Cadence, it was difficult to re-import a finished layout back from Silicon Ensemble. The DEF file that gets imported into Cadence from

Silicon Ensemble refers to the abstract cells in the standard cell library. Cadence automatically puts the abstract cells in the layout during the import process. However, these abstract cells do not have all the information required to produce a full layout. The abstract cells need to be replaced with the standard layout cells. Previous versions of Cadence had simple methods of replacing the cell types, but these methods were removed in the most recent version. In the current version, the abstract cells in the design could be replaced by their layout equivalent by performing a *Search and Replace* on the layout, by searching for all cell types of "abstract" and replacing them with cell types of "layout." Unfortunately, this method would cause Cadence to crash randomly. Yet another alternative was devised to replace the abstracts with layouts.

The abstract cells in the Cadence library could simply contain the layouts, since there was no need to have the abstracts in the library, except for during the import process. Unfortunately, another problem would occur if the layouts were simple copied to the locations of the abstracts in the standard cell library. The DEF import process would happen successfully, but the final layout produced would have the cells in the wrong places. It turned out that the cell origins were different between the abstract and layout versions of the standard cell library cells.

Finally, a working solution was developed. Two separate libraries would be created, each contained the standard cell library. The first had abstracts that actually contained the layouts of the standard cells. The second, called the *Import* library, had the original abstracts from the standard cell library. The DEF file could be imported using the *Import* library, and the imported layout would contain properly oriented abstract cells. Then, the *Rename Reference Library* function could be used to change

all instances in the imported library from referring to the *Import* library to referring to the main standard cell library. This caused the final imported layout to contain the correct standard cell layouts, with the correct orientation. The only problem with this method was that Cadence did not always remove the "lock file" that kept the imported layout from being changed. The lock file had to be manually deleted at the UNIX command line so that the *Rename Reference Library* process could write to the imported layout.

After much work, a successful strategy for importing the final layout into Cadence was devised. A second standard cell library had to be created in Cadence, called *OSU_digital_ami05_import*. This second library increased the complexity of the design flow, but this method proved to be the most stable and consistent way of importing designs from Silicon Ensemble into Cadence Virtuoso.

3.4.2 Verilog Gate List

When Silicon Ensemble imports a Verilog netlist of a synthesized design, it requires a listing of the cells that make up the standard cell library. This list tells Silicon Ensemble what cells in the Verilog netlist are part of the standard cell library, and what pin names to expect. The gate listing contains a module declaration of each cell, along with a listing of the input and output pins of the cell. A Perl script was created to read in the cell information from the LEF file, and generate the Verilog module declarations. The Perl script can be found in Appendix F. At this point, all libraries required to successfully place-and-route with Silicon Ensemble had been generated for the standard cell library.

3.4.3 Synthesis Library File Creation

The target synthesis environment for this design flow was BuildGates Extreme, provided in the Cadence suite of tools. BuildGates uses a synthesis library format that may contain cell logic, area, power dissipation, capacitance, and detailed timings. This library format is the ALF, or Ambit Library Format. This binary format can be created using the *libcompile* program that is distributed with BuildGates Extreme. The source file format is the ASCII-text and well-documented Synopsis Library (.LIB) format. The use of the Synopsis format also provides for easy portability to new synthesis tools as they become available.

The Synopsis library was created by using a Perl script to read in the standard cell library data from the LEF file created during the abstract generation process. The script can be found in Appendix G. It reads the cell names, pin names, pin directions, and cell areas from the LEF and converts the data into the LIB format. The script also generates generic maximum fanout and fanout load data based on cell data observed in other libraries. The cell logic was added manually into the new file, along with any state information for latches and flip-flops. Timing and capacitance information could be added to the library in the future; however, it was not vital for simple minimum-area synthesis and place-and-route circuit design. The complete LIB file was then converted to the ALF binary format using *libcompile*.

A graphical symbol library makes the graphical user interface for BuildGates Extreme more useful. It contains graphical gate-type views of all the standard cells. BuildGates Extreme maps the synthesized schematic instances to the gates found in the symbol library to make the diagrams more clear. Most symbols in the .sym library were manually created using generic symbols found in the BuildGates distribution.

Other symbols were created using the *symedit* tool found in the BuildGates software. The symbol library for the OSU standard cell library can be found in Appendix D.

3.5 Synthesis Steps

BuildGates Extreme was used to synthesize the sample VHDL models and map the synthesized circuits to the standard cell library. On the HP-UX machines found in the Electrical Engineering department, BuildGates Extreme was started by first sourcing the startup file and calling the name of the program at the command line:

```
>source /opt/local/cadence/StartupSPR.EE
>bgx_shell -gui &
```

The *-gui* option starts the graphical user interface for BuildGates Extreme. This is not necessary when running automatic synthesis scripts, which will be discussed later. After starting the program, the VHDL file was imported by using the menubar:

```
File -> Open
```

The "File Type" was selected as *VHDL*, and the VHDL model file was selected. The imported design was then synthesized into a set of generic digital gates using the menu:

```
Commands -> Build Generic
```

The Build Generic menu contains a few options: *All, Group all processes, Group named processes, Group all subprograms,* and *Extract FSM.* For a simple design, with

one one top-level circuit, none of these options needed to be checked. If more than one top-level design exists, then the *All* option must be checked. If more than one process or subprogram exists, they can be grouped using the other options. The *Extract FSM* option extracts a finite state machine from the design.

After the design was synthesized, the design could be optimized. The optimization step requires a physical standard cell library so that area and timing optimizations can occur. To import the library, the options were selected from the menu:

```
File -> Open
```

The "File Type" was selected as *ALF library*, and the digital cell library file was selected. After the library was imported, the library was made the target technology by using the menu:

```
Commands -> Set Target Technology
```

The appropriate technology library was then selected. The options used in the optimization process depended on the target optimizations that needed to occur. The menu command that *should* have started the optimization was:

```
Commands -> Optimize
```

Many options are brought up in a menu. However, a quirk with the graphical user interface in BuildGates caused a TCL script error:

```
Error: can't read "vgb_optimize(insert_clocktree)":
no such element in array
```

The workaround was to type the optimization command into the BuildGates command window:

```
> do_optimize
```

All of the options available in the graphical menus are also available in command-line equivalents. After optimization, the synthesized design was exported as a Verilog netlist by selecting from the menu:

```
File -> Save
```

The "File Type" was selected as *Verilog* and a file name was entered. The Verilog netlist could then be imported into Silicon Ensemble for placement and routing.

3.6 BuildGates Scripting

It was not necessary to use the graphical version of BuildGates Extreme to perform synthesis. The whole process could be synthesized using a script written in the TCL scripting language. There are command-line equivalents for all the steps used in the synthesis process in BuildGates Extreme. The scripts were used to automate the synthesis of multiple circuits under multiple standard cell libraries. A sample synthesis script is shown:

```
read_alf ~/synthesis/OSU_diglib_ami06/OSU_diglib_ami06.alf
read_vhdl ~/nc/bus/bus.vhd
do_build_generic
do_optimize -priority area
write_verilog -hierarchical OSU_diglib_ami06_bus.v
exit
```

The *read_alf* command loads the digital standard cell library into the system. The *read_vhdl* command loads the VHDL design file into the system. The *do_build_generic*

command synthesizes the loaded design and maps it to generic gates. The *do_optimize* command performs the optimization of the design for the target technology. Note that the *-priority area* option is called, which optimizes the circuit for minimum area. The *write_verilog* command exports the optimized design as a Verilog netlist. The *-hierarchical* option preserves the block hierarchy found in the original VHDL design. The *exit* command ends the script and exits BuildGates Extreme.

A report containing information about the synthesized circuit can be written by adding the following line to the TCL script:

```
report_area -cells -summary filename.txt
```

The *-cells* option reports information about the cells, and the *-summary* option provides a summary of the entire synthesized design.

The TCL script can be called at the UNIX command line:

```
> bgx_shell script.tcl
```

A separate script was generated for each individual design that was synthesized. This was because BuildGates Extreme scripting tools did not easily synthesize multiple designs under multiple libraries during the same synthesis session. A "master" shell script called individual BuildGates instances for every single design that was synthesized under different libraries. This also helped localize and identify problems in the synthesis process for a single design. The TCL scripts and the master shell script is found in Appendix H. Note that the *nohup* command preceding the *bgx_shell* command allow the process to run independent of the terminal, even if the user is logged out [23]. The *nice* command runs the *bgx_shell* command at a lower priority, so that other users can easily share processor time. The use of scripting greatly

automated the synthesis process, allowing a large batch of designs to be synthesized without user interaction. A tutorial that goes step-by-step through the process of synthesizing a sample design in BuildGates extreme is shown in Appendix A.1.

3.7 Limitations of Synthesis Software

Current synthesis tools, such as Synopsis and BuildGates are able to synthesize a wide variety of circuits. However, they still have limitations. The limitations discussed next mostly result from the synthesis software not being able to handle certain VHDL constructs. This list is not exhaustive; it merely shows important synthesis limitations observed while examining various VHDL models and synthesis tools.

3.7.1 Single Signal Transition

A synthesizable process can only look for a signal transition on one signal at a time. This is because the flip-flop that the synthesis tools use can only be sensitive to one clock or other signal transition. A different process in the same model can be sensitive to a different signal transition, or a single process can be sensitive to the VALUES of multiple signals.

An example of a process that cannot be synthesized in VHDL because of two signal transitions is shown below:

```
process(clk1, clk2)
     IF (clk1'EVENT AND clk2'EVENT )
          <VHDL Code>
     END IF;
```

```
end process;
```

The code must be rewritten so that either the separate clock transitions are handled by different processes, or the second clock transition is handled inside the process without being part of the process sensitivity list.

3.7.2 Clauses in IF Statement

Another limitation of synthesis tools is the use of multiple clauses in an IF statement. For example:

```
IF  (clk'EVENT AND clk = '1' AND start='1')
     <VHDL Code>
END IF;
```

This type of statement cannot be synthesized, simply because of limitations in the synthesis tools. Separating the code into two "IF" statements allows the code to be synthesized:

```
IF (start= '1') THEN
     IF  (clk'EVENT AND clk = '1')  THEN
          <VHDL Code>
     END IF;
END IF;
```

Note that the IF statement that contains an 'EVENT has to be the second if statement in to loop to synthesize properly in BuildGates [24]. This also makes the *start* signal asynchronous.

3.7.3 WAIT Statements

Not all types of WAIT statements are synthesizable. The WAIT FOR *time* statement cannot be synthesized. However, the WAIT UNTIL *condition* statement can be synthesized by some tools. [1].

3.8 Place & Route Steps

The placement step of automatic digital VLSI design takes a synthesized netlist of the design and lays out the standard cell equivalents in a manner conducive to easy routing within the area and timing constraints specified for the design. The routing step wires the cells together as specified in the netlist, according to routing rules specified in the technology library. Usually, these steps are performed by the same software tool. The automatic cell placement and wire routing tool used was Silicon Ensemble, provided by Cadence.

3.8.1 Software Initialization

The Silicon Ensemble software was started by sourcing the startup script for the package, and then calling the program name:

```
> source /opt/local/cadence/Startup.EE
> sedsm -m=500 &
```

The *-m=500* option allocates 500 Megabytes of memory to Silicon Ensemble. Other amounts of memory can be allocated depending on the need. Failure to allocate enough memory for a complex design will cause Silicon Ensemble to crash. The graphical user interface automatically starts unless the *-gd=ansi* option is called with the *sedsm* command.

After the Silicon Ensemble graphical interface started, the LEF library file containing the digital standard cell library and the process technology information was imported using the menu:

```
File -> Import -> LEF
```

The LEF file name was entered, and the *Clear Existing Design Data* option was checked, to make sure that no previous saved designs existed. If the option was not checked and previous design files existed in the working directory, the import would not succeed. Next, the synthesized design was imported as a Verilog netlist using the menu:

```
File -> Import -> Verilog
```

The Verilog netlist was entered in the *Verilog Source Files* option of the "Import Verilog" dialog box, along with the Verilog gate-listing file that was generated for the digital standard cell library. The name of the top-level block was entered in the *Verilog Top Module* option. The power and ground options were left at their defaults.

3.8.2 Floorplan Initialization

Next, the floorplan of the design was initialized using the menu:

```
Floorplan -> Initialize Floorplan
```

The "Initialize Floorplan" dialog box popped up, and the *Flip Every Other Row* and *Abut Rows* options were checked. These options allowed the placement tools to flip the cells of every other row so that the power and ground rails were shared, and the cells abutted each other. The *IO To Core Distance* values were inputted; for example, 20 microns each. The *Row Utilization (%)* option could be changed to make sure there

was enough room for the cells to be placed and routing wires connected between the cells. The *Calculate* button gave an estimated Width, Height, Chip Area, and Core Row Utilization. Changing the options in the window can change the expected results. The Core Row Utilization shown in the "Expected Results" box must be below 100 % for the cells to be placed correctly. Even if the Core Row Utilization is below 100 %, the routing steps may not complete successfully, because there may not be enough room to route wires. After these options were set up, the floorplan was initialized, and the basic chip layout appeared, which showed the rows that cells could be placed in as blocks.

3.8.3 Power Planning

This step could have been done before or after cell placement. The power planning stage was started by using the menu:

```
Route -> Plan Power
```

The "Plan Power" dialog box then appeared. Selecting the *Add Rings* option brought up the dialog box that allowed power and ground rings around the design to be specified. There must be enough space specified for the rings to fit in the area between the edge of the layout and the standard cell rows. If not, the rings cannot be placed. This was specified in the previous *IO to Core Distance* settings from the original floorplanning. The *Core Ring Width* was specified for both the Horizontal and Vertical directions; for example. 2.40 microns. When done, the power and ground rings appeared. The next option in the "Plan Power" dialog box, *Add Stripes*, could be used to add vertical power and ground stripes through the circuit. The Width, Spacing, and Stripe Set-to-Set Spacing could all be entered to specify the stripes. Stripes are

really only necessary for large designs, to provide consistent power and ground supplies to the cells in the middle of the layout block. For example, the *Width* of a strip could be specified as 1.2 microns, and the *Set-to-Set Spacing* could be specified as 100 microns. This means that every 100 microns there would be a pair of vertical power and ground stripes. When these options were entered, the vertical stripes appeared. However, the rings and stripes are still not connected to each other or the cells in the block. The power and ground paths were connected using the menu option:

```
Route -> Connect Ring
```

Various options exist to connect the ring together, including switches to connect stripes, blocks, rings, IO pads, and cell pins. The width of the connections could also be specified. However, for most cases, the options already selected by default are fine.

3.8.4 Pin and Cell Placement

The next step was to place the input and output pins specified in the Verilog netlist. This was done using the menu option:

```
Place -> IOs
```

The default *Random* option simply places them clockwise around the circuit in the order they are specified in the netlist. Another option, *IO Constraint File*, places the pins according to a constraint file. This allows manual placement of pins for maximum flexibility. A sample constraint file is shown in Appendix I. There is also an option to *Refine Pin Placement* after the cells have been placed or routing has occurred, which can improve the design timing or lessen the amount of routing required in the layout. The pins can also be assigned to be on different layers using the *Pin Layer*

Assignment option. The spacing between pins can also be specified in this dialog, and the options are to be spaced Evenly, Abutted, or Center Abutted. The default option, *Random* was the only option necessary to successfully place the pins.

After pin placement, cell placement occurred using the menu option:

```
Place -> Cells
```

The defaults were again used in the resulting dialog box. There were several options included that could change cell placement depending on timing or pin placement, and generate timing analyses and congestion maps.

After cell placement, there were still gaps between the cells. To ensure n-well continuity and power and ground rail continuity, filler cells were added. This was done using the menu:

```
Place -> Filler Cells -> Add Cells
```

The model name, which was the name of the fill cell in the standard cell library, was entered, along with a prefix name, which could be any text. The *Placement* options found in the dialog box had to be set to have only *North* and *Flip South* check. This was because the fill cells could only be in those two orientations to place successfully without overlap. *Special Pins* and *Special Nets* were also added to the appropriate dialog box, by adding the proper names of the power and ground pins to the dialog box. After the settings were entered, fill cells were added to the design.

3.8.5 Routing

Since all cells had been placed, routing could now be started. This was done using the menu:

`Route -> WRoute`

The WRoute option allowed a few different options. The defaults, *Global and Final Route* with *Auto Search and Repair* also selected, was used. The steps could also be broken up to only do global routing by checking *Global Route Only*, or redo the final routing by checking *Incremental Final Route*. Using that option, it was also possible to redo the global routing for violations using the appropriate checkbox. *Timing Driven Routing* could also be specified. After being set up, the final routing step placed wires using the technology and routing constraints given to it The blockages inside the cells were avoided if necessary, and each metal layer wire was usually routed in the direction specified in the LEF file. In some cases, wires on a metal layer would route in the direction opposite to the directions specified in the LEF. This is because Silicon Ensemble will break some of the routing rules in order to properly route a layout without producing design errors or geometry violations.

If geometry violations existed in the routed layout, a few options existed. First, the final routing stage could be rerun, to try and reroute the wires and remove the errors. Also, the floorplan could be re-initialized, using different settings. The IO to Core distance could be increased, or the Row Core Utilization percentage could be lowered. Lowering the row core utilization spreads out the cells, and gives more room for wire routing. This was usually the best option.

3.8.6 Export

After the routed layout was completed without errors, the design could be exported. The layout could be exported into the following formats: Encrypted LEF,

DEF, Verilog, LEF Block, PDEF, SDF, and GDS II. The only options successfully explored were the DEF and GDS II formats.

3.8.7 DEF

The Design Exchange Format (DEF) was mostly used to exchange the design between Silicon Ensemble and Cadence's Virtuoso layout software. The DEF was exported using the menu:

```
File -> Export -> DEF
```

The exported filename was entered and the following options checked: Cells, Nets, Special Nets, Vias, Groups, Modifications, and Layout Modifications. This provided enough information for Cadence to successfully import the DEF file.

3.8.8 GDSII

Some digital standard cell libraries did not contain the required abstracts to be able to import the design into Cadence using the DEF file format. An alternate method of exchanging files between Silicon Ensemble and Cadence is using the GDSII file format. The layer map files needed to use this method are taken from the Illinois Institute of Technology standard cell library support files [25]. These files are named *gd2_icfb.map* and *gds2_seultra.map*.

First, the finished design was exported from Silicon Ensemble using the menu:

```
File -> Export -> GDS II...
```

The name of the output file was entered into the *GDS-II File* box. The *gds2_seultra.map* map file was entered into the *Map File* box. Finally, the name of the top module of the design was entered into the *Structure Name* box.

A quick tutorial that takes a sample design through this part of the design flow can be found in A.2.

3.8.9 Silicon Ensemble Scripting

There are limited scripting capabilities included with Silicon Ensemble. Every time a Silicon Ensemble session is started, a journal file is created, listing all the commands used in the session. Renaming the extension of the journal file from .jnl to .mac turns the file into a Silicon Ensemble command script file. This allows for a session to be run *exactly* as a previous session was run. Editing the script commands allows for more flexibility, but there are some major limitations with the script commands. Some commands require layout area information, which is not available until other commands have been executed. This means that more complex layouts cannot be successfully automated, because the command structure in the script will not execute properly if the area values are incorrect.

An example problem occurs when adding fill cells to the layout, to ensure n-well continuity in the circuit layout. The command to do this is:

```
SROUTE ADDCELL MODEL fill1 PREFIX fill1 NO FS SPIN
                    vdd! NET vdd! SPIN gnd! NET gnd!
                    AREA ( -84240 -84300 ) ( 84480 84600 ) ;
```

The values after the "AREA" option in the command represent the coordinate values of the layout corners. The size of the floorplan is not established until after the floorplan is initialized by a previous command. The fill cells will not be added correctly if the coordinates are not entered correctly.

Another problem caused by incorrect area information occurs when trying to add rows of a different standard cell height to the circuit. This is done when a standard cell library contains cells of multiple heights. The ADD ROW command requires coordinate values that are not available until the floorplan has been initialized. If the area is too small, there may not be enough room to place the cells of the different height. If the area is too large, the coordinates will be out of coordinate boundaries of the floorplan, which will cause an error.

Another problem that occurs with trying to create Silicon Ensemble scripts is that the core row utilization percentage may be too high to be able to route the layout successfully. In this case, the percentage specified in the script can be lowered, and the script can be re-run. Other such script parameters can be modified manually, such as the IO to Core Distance, or the width and spacing of power and ground rings and stripes. The manual modification of these scripts limits the usefulness of the scripting tools, since the time savings of scripting over use of the graphical interface are minimal.

Despite these problems, some semi-autonomous place-and-route scripts were created. These scripts can be found in Appendix J. A master shell script called these Silicon Ensemble scripts, and redirected some of the script output messages to files. These helped identify problems or errors in the script generation. Each script was called using the command:

```
nohup nice sedsm -gd=ansi -m=500 "EXECUTE script.mac"
                | grep -e violations -e ERROR > errors.txt
```

As with the synthesis scripts, the program is run in the background at a low priority. The *-gd=ansi* option runs Silicon Ensemble in the terminal window instead of having a graphical user interface. The *"EXECUTE script.mac"* command executes the place-and-route script. The rest of the command redirects any output lines containing the words *violations* or *ERROR* to a file named *errors.txt*. Most relevant messages about script execution problems or geometry violations in the layout contain those words. When an error occurred, the script was manually altered, and then re-run.

3.9 Import Into Cadence

The designs exported from Silicon Ensemble were imported into Cadence Virtuoso for final layout verification. The layout was checked for design errors, and then compared to a schematic of the synthesized design to prove they were equivalent.

3.9.1 Import Verilog as Schematic

Schematics based on the synthesized Verilog netlist created by BuildGates Extreme were imported into Cadence. The cells used in the netlist were referenced to the schematics contained in the Cadence version of the standard cell library.

First, a Cadence library was created to contain the design. It was mapped to the *NCSU_TechLib_ami06* reference library. Then, the Verilog netlist was imported using the ICFB menu:

```
File -> Import -> Verilog
```

The newly created design library was entered as the *Target Library Name*, and "basic" and "digital_lib_ami06" were entered as the Reference Libraries. The verilog files to be imported were entered in the *Verilog Files To Import* box. If this was not the first time importing files into the new library, the *Overwrite Existing Views* box needed to be checked. When properly set up, executing this dialog box imported all cells in the Verilog netlist, and properly referenced cells in the design to the standard cell library schematics and symbols already found in the *OSU_digital_ami05* library. If there is more than one module in the netlist hierarchy, schematics will be created and properly referenced to each other. If a module is referenced that is not defined in the standard cell library or in the netlist itself, then a "blackbox" symbol will be created, without any reference to a schematic.

3.9.2 Import DEF as Layout

To import the DEF created by Silicon Ensemble, the Cadence ICFB menu was used:

```
File -> Import -> DEF
```

The library name was filled into the *Library Name* box, and the name of the top level of the design was filled into the *Cell Name* box. The *View Name* was filled in as "layout." The *Use Ref. Library Names:* box was filled in with "OSU_digital_ami05_import." This was because the proper abstracts referred to in the DEF file are contained in that library, rather than the normal *OSU_digital_ami05* library. The *DEF File Name:* was also entered before executing the dialog box. A new cellview was created called *layout* in the top module of the design library. It consisted of all the placed abstracts taken from the standard cell library, and all the routing created by Silicon Ensemble.

The layout is not quite finished. The abstract standard library cells had to be converted into layout cells. This was done using the *Rename Reference Library* functions. However, due to a quirk with Cadence, a file lock was usually placed on the directory containing the layout. While this lock exists, the file cannot be edited by a Cadence program.

To remove the file lock, the lock file was removed from the directory:

```
> rm <Cadence Directory>/<Library>/<Cell>/layout/layout.cdb.cdslck
```

The lock could also be removed most of the time by exiting Cadence and restarting the program. After the lock file was removed, the reference library could be changed using the Design Manager Window:

```
Edit -> Rename Reference Library
```

The *From Library:* box was filled with "OSU_digital_ami05_import" and the *To Library:* was filled with "OSU_digital_ami05." Executing this command changed all cells that were referenced to the import version of the library, which contained the abstracts, to the regular version of the library, which had the abstract cells replaced with the normal layouts. The layout cell was now a valid semiconductor layout that could be tested in the same manner as a custom-built layout.

3.9.3 Import GDS II as Layout

To import a GDS II stream created by Silicon Ensemble into Cadence, a library first had to be created to contain the imported file. Then, all of the standard cells

that would be used in the final design had to be copied from the standard cell library into the new library. This was so that the GDS II file could reference the proper files during import.

The GDS II File could be imported into Cadence using the ICFB menu:

```
File -> Import -> Stream
```

The path and name of the input stream file was entered into the *Input File* box. The name of the top module in the design was entered into the *Top Cell Name* box. The name of the Cadence Library that would contain the imported design was entered into the *Library Name* box. Next, the name of the layer map table was entered by clicking on the button: *User Defined Data*. In the dialog box that popped up, the name of the layer map, in this case "gds2_icfb.map," was entered into the *Layer Map Table* box. After these options were set, the file would import into the directory, and reference the proper layouts copied from the standard cell library. The layout, if the import was successful, could be tested in the same manner as a custom-built layout.

3.9.4 Design Checking

The design checking followed standard procedures common to layout creation in Cadence. The design was checked using the DRC tool to find design rule violations. If the design was imported properly, no design errors should have existed. Then the layout was extracted into a transistor-level netlist, so that a Layout-Versus-Schematic check could be performed. The LVS check compares the schematics derived from the imported Verilog netlist to the imported layout. Functionally, this test can only prove that the layout matches the post-synthesis design. Therefore, the Verilog netlist

should also be simulated against the original hardware description to verify that the Verilog description is correct.

The layout can be combined with any analog or mixed-signal designs and exported in a foundry-compatible format in the same manner as any other Cadence layout. A full tutorial that goes from HDL to a Cadence layout can be found in Appendix A.

3.10 Cell Libraries Used As Reference

Once the standard cell library had been created and proper libraries set up, it could be compared to other similar research standard cell libraries. The other standard cell libraries were provided by The Illinios Institute of Technology [25], The University of Tennessee [26], and The Mississippi State University [17]. Eventually, these three libraries were used in the HDL-to-layout design flow, so each library had some minor modifications so that it was compatible with the software used in the flow.

3.10.1 IIT06_STDCELLS

The Illinois Institute of Technology created this standard cell library for use in student projects at the university [27]. It is freely available for educational use. Library files are available for Synopsis synthesis, Verilog and VHDL simulation, Silicon Ensemble placement-and-routing [25]. Layouts of 26 different cells are included. Various scripts to automate synthesis, place-and-route, and conversion to different software environments, such as Cadence, SignalStorm, Magic, and IRSIM are also available with the library. A *RuleVia* symbol must be added to the Cadence ICFB instance of the library, to allow proper importing of Metal2 to Metal1 vias from the completed Silicon Ensemble layout.

3.10.2 UT_LP_AMI06

This library was created at the University of Tennessee as part of a class that developed a low-power library of standard cells [28]. These cells were derived from the Tanner AMI-0.5u Standard Cell Library [15]. The library cells are available in the Cadence design library format at the University of Tennessee website [26]. A verilog cell description and LEF file were included for use in Silicon Ensemble. Synthesis libraries were not included with the library, so the same techniques that were used to develop the OSU synthesis library files were utilized to create synthesis libraries.

3.10.3 MSU_Jennings

The Mississippi State University standard cell library was created by former MSU student Scott Jennings and is used for course work at MSU [17]. This minimal library only contains six digital cells. This library also contains a double-height cell, meaning that the height of the cell is twice the height of the normal standard-cell library height. The D flip-flop is double-height because it is much larger than the rest of the cells in the library. Increasing the height of the cell allows more efficient intra-cell routing, which may decrease the overall area compared to a single-height version of the same cell.

The MSU library and relevant design flows were well-documented on the MSU website. The Synopsis synthesis libraries were provided, along with the LEF file needed for placement and routing. A Verilog gate list was created from the LEF file using the scripts developed for the OSU library. Also, a *RuleVia* symbol was added to the Cadence ICFB instance of the library, to allow proper importing of Metal2 to Metal1 vias from the completed Silicon Ensemble layout.

3.11 VHDL Sample Models

Three different test designs were synthesized using four different digital standard cell libraries. Not all designs were able to be synthesized using all cell libraries, due to the differences in functions available in the libraries. The next sections briefly describe each of the sample models.

3.11.1 8-bit Bidirectional Bus

The VHDL model for this design was produced by Altera, which produces programmable logic devices. It is a sample design meant for use on a CPLD, but can easily be synthesized by most of the standard cell libraries. The model contains a behavioral representation of an 8-bit bidirectional bus. [29]

3.11.2 Mini-Universal Asynchronous Receiver-Transmitter

This design was obtained from the OpenCores Project website. The MiniUART is much more complex than the bidirectional bus sample design, however still relatively simple. The MiniUART is used to interface an 8-bit data bus to an RS-232 serial line. It can handle multiple baudrates, but contains no control handshaking so it cannot be used to interface with a modem. It also does not support FIFO input or output. This design is a larger and more useful circuit to test than the bidirectional bus design. It was found that a synthesized design used many different types of cells, but had much less internal wiring than a complex system, such as a microprocessor. This allowed a denser layout to be created, which helped identify standard cell design errors. [13]

3.11.3 AVR Microprocessor core

The AVR microprocessor is the most complex design synthesized during the analysis of the standard cell libraries. This particular core is compatible with the ATmega103 [30]. It contains 32 8-bit registers, and allows most of the instructions found in the ATmega103 core, except for the *SLEEP* and *CLRWDT* instructions. The AVR RISC architecture is designed for low power operation, and can execute most instructions in a single clock cycle. This processor core design was chosen because of the relative popularity of the AVR architecture, making it a good "real world" type of test for the standard cell libraries. It also provides a larger and more complex design than the other two models used for analysis. [12]

CHAPTER 4

Results

The design flow that was developed was used to produce layouts from three different hardware descriptions. The three layouts were produced using the Ohio State standard cell library, along with three other similar standard cell libraries for comparison. In this chapter, the standard cell libraries will be compared. Then, the results of the synthesis and layout stages of the design flow will be discussed.

4.1 Comparison of Standard Cell Libraries

The four standard cell libraries compared had many similar features. Since they were all based on the AMI 0.5 micron process, they all had to follow the same design rules. However, they all followed different standard cell design rules, with different routing grid sizes, standard cell heights, and size requirements. The types of cells chosen for each library also differed.

A comparison of the standard cell libraries is shown in Table 4.1. Cells that are functionally identical are contained in the same row. The library with the largest number of cells was the Illinois Institute of Technology library, although many of these cells have the same logic, but have different drive strengths. A comparison of the total number of cells in each library, and also a comparison of the number of cells

that don't include repeated cells of different drive strengths is included in Table 4.2. After discounting the functionally identical cells with different drive strengths, it was found that the OSU library had slightly more cells than the IIT library.

Since one of the design goals of the OSU standard cell lib was to minimize cell area, the relative sizes of equivalent cells were compared. Table 4.3 shows all cells in the OSU library that have an equivalent cell in another library, and the area of the smallest cell in the other library compared to the OSU cell. Most of the OSU cells were 16.7 % smaller than the next smallest equivalent cell found in another library. This was due to the fact that the second smallest library was made by the University of Tennessee, and their standard cell height was 16.7 % taller than the standard cell height of the OSU library. Only one cell in the OSU standard cell library was actually larger than the smallest equivalent cell from another standard library. The *dff* cell was 0.88 % larger than the D flip-flop found in the UT library. Also, the *mux21x1* cell was only 0.88 % smaller than the the multiplexor found in the UT library. In general, though, the OSU cells were significantly smaller than the cells found in the other libraries.

4.2 Synthesis Results

Synthesis of the three sample cores under the four different standard cell libraries was achieved using the software tool BuildGates Extreme provided by Cadence. The summary files produced by BuildGates as a result of synthesizing the designs can be found in Appendix K. Not every library was able to synthesize every sample core. The University of Tennessee library and the Mississippi State library lacked a tri-state device, which made them unsuitable for synthesizing the bidirectional bus

MSU		IIT		UT		OSU	
Cell	Area	Cell	Area	Cell	Area	Cell	Area
		AND2X1	288				
		AND2X2	288				
		AOI21X1	288				
						ao22x1	252
		AOI22X1	360	AOI22	216	aoi22x1	180
		BUFX2	216	BUF1	129.6	bufx1	108
		BUFX4	288	BUF4	259.2	bufx4	216
		DFFNEGX1	864				
		DFFPOSX1	864	DFF_S	820.8	dff	828
DFFSRX1	1209.6			DFFPC_S	1166.4	dffpc	1116
		FAX1	1080				
		HAX1	720				
INVX1	86.4	INVX1	144	INV	86.4	invx1	72
		INVX2	144				
		INVX4	216			invx4	180
		INVX8	360				
						invzx1	216
						lat	360
						latpc	540
		MUX2X1	432	MUX2	345.6	mux21x1	342.72
NAND2X1	129.6	NAND2X1	216	NAND2	129.6	nand2x1	108
		NAND3X1	324	NAND3	172.8	nand3x1	144
				NAND4	216	nand4x1	180
NOR2X1	129.6	NOR2X1	216	NOR2	129.6	nor2x1	108
		NOR3X1	576	NOR3	172.8	nor3x1	144
						nor4x1	180
		OAI21X1	207				
		OAI22X1	360				
		OR2X1	288				
		OR2X2	288				
		TBUFX1	360			bufzx1	288
		TBUFX2	504				
TIEHI	86.4			TIEHI	86.4	tiehigh	72
TIELO	86.4			TIELO	86.4	tielow	72
						xnor2x1	252
		XOR2X1	504	XOR2	302.4	xor2x1	252

Table 4.1: Comparison of Standard Cell Libraries.

	MSU	IIT	UT	OSU
Total Number Cells	6	26	15	23
Unique Cells	6	20	14	21

Table 4.2: Comparison of Number of Cells in the Standard Cell Libraries.

OSU Cell	Area (microns)	Other Library	Area (microns)	% Difference
aoi22x1	180	UT	216	-16.7 %
bufx1	108	UT	129.6	-16.7 %
dff	828	UT	820.8	+0.88 %
dffpc	1116	UT	1166.4	-4.3 %
invx1	72	UT/MSU	86.4	-16.7 %
invx4	180	IIT	216	-16.7 %
mux21x1	342.72	UT	345.6	-0.88 %
nand2x1	108	UT/MSU	129.6	-16.7 %
nand3x1	144	UT	172.8	-16.7 %
nand4x1	180	UT	216	-16.7 %
nor2x1	108	UT/MSU	129.6	-16.7 %
nor3x1	144	UT	172.8	-16.7 %
bufzx1	280	IIT	360	-20 %
xor2x1	252	UT	302.4	-16.7 %

Table 4.3: Area Comparison of OSU Cells with Smallest in Other Library.

design. The Illinois Institute of Technology's library lacked a flip-flop that contained preset or clear, which made it unable to synthesize the AVR microprocessor core. The Ohio State library was the only library capable of synthesizing all three designs. This was an unintended consequence of the varied sample designs, but it shows how a large number of available cells in a library can greatly improve the usefulness of a standard cell library.

4.2.1 Bidirectional Bus

The two libraries that successfully synthesized the bidirectional bus design were the IIT and the OSU libraries. A comparison of the two synthesized designs is found in Table 4.4. The others lacked a tri-state device required to implement the type of bus the synthesis tool created. The IIT version of the synthesized circuit was created a bit differently than the OSU version, due to the slightly different composition of the cell libraries. The IIT version contained an extra seven inverters. Combined with the fact that the OSU library cells are significantly smaller, the total unrouted cell area of the OSU implementation is 15.7 % smaller than the IIT implementation. Note that the value of 15,048 microns is the area of just the cells, and is not the final area of the placed and routed circuit. The improvements in cell density in the OSU library could be offset by lack of feedthrough paths through the cells, which would cause the cells to be placed farther apart to increase routing paths in the final layout.

4.2.2 MiniUART

BuildGates was able to successfully synthesize the miniUART sample design using all four standard cell libraries. A table comparing the number of cells used and the total circuit area is shown in Table 4.5. The OSU library again came out smallest

	IIT	OSU
Total Number of Instances	32	25
Total Cell Area (microns ^{2})	17,856	15,048

Table 4.4: Synthesis Results of Bidirectional Bus.

	MSU	UT	IIT	OSU
Total Number of Instances	693	413	478	478
Total Cell Area (microns^2)	175,478.4	129,384	159,993	122,505.12

Table 4.5: Synthesis Results of MiniUART Design.

in total area, although the UT and IIT libraries used fewer cells. The OSU library was 5.6% smaller than the next smallest library, and 30.4% smaller than the largest library.

4.2.3 AVR Microprocessor Core

The design of the AVR microprocessor core required a register with preset or clear to properly synthesize, so the IIT library was unable to be utilized to synthesize this core. A comparison of the remaining three libraries is shown in Table 4.6. Again, the OSU library has more cells than the UT library, but a smaller total area. The OSU library is 8.6% smaller than the UT library, and 23.6% smaller than the MSU library.

4.2.4 Post-Synthesis, Pre-Routing Analysis

It is clear that the strategy of minimizing cell area seems to be advantageous, but it is also clear that a large, full-featured library of cells is even more important.

	MSU	UT	OSU
Total Number of Instances	6,797	3,453	3,598
Total Cell Area (microns^2)	1,343,822.42	1,122,249.62	1,024,963.2

Table 4.6: Synthesis Results of AVR Microprocessor.

The synthesized circuits used a fewer number of total cells when mapped to the UT library than the OSU library. However, that library lacked a tri-state buffer or tri-state inverter, so the bidirectional bus design could not be synthesized. In the IIT library, the lack of a complex memory element such as the *dffpc* element in the OSU library kept it from being synthesized for the AVR microprocessor core. The OSU library had all of the essential elements for successful synthesis, but the library could be expanded to make the synthesis more efficient by using more complex function blocks in a single cell to perform the same function taken up by multiple combinational logic cells. This assumes that the compound logic blocks would take up less area than stringing together multiple simple logic blocks.

The synthesis analysis showed that the OSU library was a very good option for synthesis, since it contained a wide variety of cells, and had, on average, significant area advantages over the other cell libraries. The area advantages were not so important at this stage, since wire routing could take up a large part of the final design.

4.3 Placement and Routing

The place-and-route stage of the design flow depended less on the types of cells and types of designs being used, and more on the number of cells, physical cell layouts,

and amount of wiring in the design. Thus, the final layout areas would differ greatly from the synthesized estimation of the design area.

The software tool used for the place-and-route stage of the design flow was Silicon Ensemble. This software, provided by Cadence, is widely used in the university research environment. The graphical user interface for Silicon Ensemble is useful for the detailed place-and-route of a single design, because many placement and routing settings can be tweaked and altered to generate the desired layout. The scripting commands that are used with the software are not as flexible as those used in the synthesis tools. A good knowledge of the expected layout area and density is required because the script commands require area-knowledgeable parameters.

To achieve desired automated placement and routing with Silicon Ensemble, a sample design was put through the graphical version of the Silicon Ensemble and successfully placed and routed. Silicon Ensemble automatically writes all the commands used in a session to a journal file. Changing the file extension of the journal file from ".jnl" to ".mac" and executing the file with Silicon Ensemble reruns all the commands used in the previous session. This journal file was used as the basis for all the scripts used to produce layouts from the synthesized designs for all the libraries.

The Silicon Ensemble execution scripts used to place-and-route some of the sample designs can be found in Appendix J. Various changes had to be implemented for each sample design and each library. Filenames and library references were changed for the appropriate designs. Power and Ground declarations in the script had to be changed to match the references in the relevant standard cell library. An extra command had to be added for the MSU_Jennings library designs to add rows of double cell height in addition to the single cell height rows. This was to allow the placement of

Design	MSU	UT	IIT	OSU
Bidirectional Bus Area (microns^2)	N/A	N/A	45,624.96	38,025
MiniUART Area (microns^2)	288,369	205,752.96	257,049	199,809
AVR Core Area (microns^2)	2,493,872.64	2,424,249	N/A	3,721,041

Table 4.7: Final Design Layout Areas.

the D flip-flop onto the layouts. Each design required a different core row utilization percentage to allow enough room for the routing between the cells. This was achieved by running a master shell script that would run each of the designs sequentially, and create text files with a quick summary of Silicon Ensemble errors or layout geometry violations. If the place-and-route stage successfully passed, the command to execute that script was commented out manually, and the core row utilization percentage was commented next to the appropriate execution command. If the design did not execute successfully, then the core row utilization percentage was decreased by five percent in the relevant script, and the master shell script was rerun. This process was repeated until all scripts completed successfully.

The final layout areas of all the circuits that were created are shown in Table 4.7. Screenshots of the final layouts produced by the OSU_digital_ami05 library can be found Appendix N. As was expected, the OSU library was the smallest for the bidirectional bus design and the miniUART sample. In fact, the bus design was 16.7 % smaller than the other design,about the same amount that the individual standard library cells were smaller than the other libraries, in general. The MiniUART design was only about 2.9 % smaller than the next smallest design, which used the UT library. However, the AVR core design using the OSU standard cell library was much

larger than the other designs. In fact, the layout was more than 50 % larger than the smallest design, created using the UT library. The limiting factor was the D flip-flop with preset and clear, which was widely used in all the layouts. The area of the OSU version of the D flip-flop was comparable to or smaller than the other libraries, but the wide use of Metal2 to route between transistors in the cell made routing vertical paths over the cell very difficult. This was because Metal2 was the only metal layer allowed to route vertically in the automatic place-and-route software.

After all layouts were successfully created, it became clear that as the designs became more complex, the core row utilization percentage dropped dramatically, as seen in Table 4.8. The small designs had a row utilization of about 80-85 %, while the AVR microprocessor core had row utilizations that varied quite a bit.

The new Ohio State library seems comparable to the other libraries with respect to core row utilization, except with the AVR core, which was only at 35 %. When using the place-and-route software with the graphical user interface, certain "search and repair" techniques to fix geometry violations can bring the core row utilization of the OSU library up to about 45 % for the AVR core. This is more difficult to do when batch scripting multiple designs, because it requires feedback on the success or failure of the routing process. This could be difficult to implement using the Silicon Ensemble scripting environment.

4.3.1 Design Checking

All of the synthesized designs were able to be placed-and-routed, but some of the reference standard cell libraries had trouble being re-imported into Cadence. This was mainly due to the other libraries not having quite the same setup as the OSU

Design	MSU	UT	IIT	OSU
Bidirectional Bus (%)	N/A	N/A	85	85
MiniUART (%)	85	85	85	80
AVR Core (%)	60	50	N/A	35

Table 4.8: Core Row Utilization of all placed-and-routed designs.

library. The IIT library did not contain abstract cells in the Cadence version of the library. These would have to be created to use the DEF import process used by the OSU library. The MSU library imported without problems, but didn't pass the design rule check, possibly due to spacing errors involving the boundaries of the D flip-flop. This might be due to different boundary rules that may not have been communicated properly to Silicon Ensemble, although this difference was never established as the problem. The UT library also imported properly, but the Search & Replace option in Cadence was needed to replace the abstract cells in the designs with layout views. The only problem with this approach was that it sometimes tended to cause Cadence to crash abruptly. The UT library also did not pass DRC, due to a spacing issue between an active contact in the D flip-flop and any abutting active layer from another cell.

The designs built using the OSU library all passed DRC. The designs were all successfully extracted into a netlist for layout-versus-schematic verification. The non-OSU libraries were not checked for LVS equivalency because of the previously mentioned problems with import into Cadence and DRC errors. It also was not a high priority because the focus of the design flow was the custom-built standard cell library.

The bidirectional bus and miniUART layouts both easily passed LVS, however, the AVR microprocessor core had a problem with the schematic that had been imported from the Verilog netlist. One of the modules created by the synthesis tool was *AWRS_partition_2*. This schematic contained terminals that were "shorted together," causing errors when trying to check and save the schematic in Cadence. This issue must be taken into account when trying to verify a layout by checking it against a schematic imported from BuildGates.

CHAPTER 5

Conclusions

The HDL to Layout design flow allows simple ASIC design generation, and easy integration into current digital and analog design methodology. The design can be created, simulated, and verified using standard digital design tools, and the results can be automatically synthesized and mapped to a standard cell library. The synthesized design can be easily turned into a layout, and imported into the same environment that is heavily used for analog and mixed-signal design, simulation, and layout. Once in this environment, digital and analog blocks can be placed side-by-side, allowing production of mixed-signal integrated circuits.

The support libraries, files, and documentation outlined proved successful when used to process hardware descriptions into finished layouts. The layouts all passed design-rule checking and in most cases could be proved equivalent to their schematic counterparts.

5.1 Contributions

Implementing a methodology that automated generation of digital integrated circuits and allowed integration of these designs with analog and mixed-signal layouts was the focus of this research. To that end, a flexible digital standard cell library

was created, along with all necessary support files. The information gleaned from comparison of the standard cell library will hopefully assist others in deciding how to use this flow, depending on their design requirements.

5.2 Future Work

5.2.1 Standard Cell Library Improvements

The standard cell library used in this design flow proved successful, improvements would improve the power and flexibility of the software tools. Performing power and timing characterization and adding this information to the synthesis files would open up many more synthesis options. The synthesis tools would then be able to perform timing-knowledgeable synthesis and low power synthesis. The resulting synthesized netlists would better match design requirements. Timing and power characterization were not a high priority during initial standard cell library design, since the AMI 0.5 micron process is not a cutting-edge process and would not likely be used for high-speed designs.

More logic cells could also be added to the library to improve flexibility. Different types of custom-built compound logic gates could be added that would minimize cell area, since the gates would be optimized for area. Also, less routing would be needed, since the cells would replace groups of logic gates wired together in the layout. Additional types of memory element, such as flip-flops and latches, could be added to help minimize area. For example, a flip-flop that can only be cleared could replace one with preset and clear if the design did not need the preset functionality.

Cells currently in the standard cell library could be improved through redesign. Specifically, the D flip-flop with preset and clear is large and contains a large amount

of Metal2. The cell proves to be a large obstacle when trying to route large designs. Rebuilding this cell, even at the expense of increasing the cell area, would improve the routability of large designs, such as the AVR microprocessor core. Improving the routability would allow a higher row utilization, which allow denser designs, more than offsetting an increase in cell area of the D flip-flop.

5.2.2 Software Tools

Many features and options pertaining to the software tools used in the design flow have not been fully explored. These tools could be better documented, and different features could prove useful for generating layouts from a hardware description language.

In particular, the scripting abilities of Silicon Ensemble are very limited. The scripting language is only useful for repeating the same routing process for the same design. The scripting tools with Silicon Ensemble should be further explored to see if more advanced scripting options exist.

Various inconsistencies were found in the software tools, from broken library compatibility between programs to user interface errors within some tools. At the very least, better workarounds for these problems could be explored, such as fixing the problems that prevented one of the Verilog-generated schematics from being used in LVS testing.

Other software tools exist for HDL synthesis and design placement-and-routing. The library files needed by the Synopsis Design Compiler have already been created, so adding this software as an option in the design flow should be trivial. The tools

provided by other companies should also be explored and analyzed for suitability in the design flow.

5.3 Final Remarks

Advances in automatic creation of digital integrated circuits has allowed more of the design time to be spend on the analog portions of mixed-signal circuits. It is hoped that the design flow developed during the course of this research can be used to introduce others to the types of methods common in modern digital VLSI circuit production. It is also hoped that the digital standard cell library will serve as the basis for other projects, and that the library itself will be expanded and improved in the future.

APPENDIX A

VHDL/Verilog Synthesis, Place & Route Flow Tutorial

This is a tutorial showing how to take a VHDL hardware model and synthesize it into a Verilog netlist, then take the design and produce a fully placed and routed layout.

```
Sample Design: Mini-UART
Universal Asynchronous Receiver Transmitter
http://www.opencores.org/cvsweb.shtml/MiniUART/

miniUART.vhd (VHDL descriptions)
RxUnit.vhd
TxUnit.vhd
 clkUnit.vhd

Target Technology: OSU_diglib_ami06
```

A.1 Synthesis

```
Files needed for Synthesis:
OSU_diglib_ami06.alf
OSU_diglib_ami06.sym (for graphical use)

1) source /opt/local/cadence/StartupSPR.EE
2) bgx_shell -gui &
3) Import VHDL Design
- File -> Open
- Select File Type: VHDL, select:
miniUART.vhd
RxUnit.vhd
TxUnit.vhd
clkUnit.vhd
- Click OK
command line equivalent: read_vhdl /rcc4/student/copusj/flow_tutorial/RxUnit.vhd
       read_vhdl /rcc4/student/copusj/flow_tutorial/TxUnit.vhd
       read_vhdl /rcc4/student/copusj/flow_tutorial/clkUnit.vhd
       read_vhdl /rcc4/student/copusj/flow_tutorial/miniUART.vhd
```

```
4) Synthesize Design
- Commands -> Build_Generic
- Click OK
command line equivalent: do_build_generic

You can view the schematic at this point by right-clicking "miniUART" in the
Logical Tab and selecting Open Schematic -> Main Window.

5) Optimize and Map to Desired Library
- File -> Open
- Select File Type: ALF library, browse to proper directory and select OSU_diglib_ami06.alf
- Click OK
command line equivalent: read_alf /rcc4/student/1/OSU_diglib_ami06.alf

- Commands -> Set Target Technology, select OSU_diglib_ami06 Click OK
- Commands -> Optimize, select Priority Area Click OK
command line equivalent: do_optimize -priority area

6) Save as Verilog Netlist
- File -> Save As, select Verilog, type in file name
command line equivalent: write_verilog miniUART_built.v


-Note, this whole process can be scripted in a .tcl file:
read_alf OSU_diglib_ami06.alf
set_global hdl_vhdl_environment common

read_vhdl  RxUnit.vhd
read_vhdl  TxUnit.vhd
read_vhdl  clkUnit.vhd
read_vhdl  miniUART.vhd

do_build_generic
do_optimize -priority area
write_verilog -hierarchical miniUART_built.v

-Then, run bgx_shell <filename.tcl> to synthesize the circuit.
```

A.2 Place & Route Design

```
Files Needed for Place & Route:
OSU_diglib_ami06.lef : Library Exchange Format of digital cell library
OSU_diglib_ami06.v : Verilog modules of digital cell library gates
miniUART_built.v  : Verilog design from previous step

1) Create a Silicon Ensemble work directory and start the software
- mkdir se
- sedsm -m=500 &
starts Silicon Ensemble and allocates 500 MB of memory (good for very large
designs)

2) Import LEF:
- File -> Import -> LEF, select LEF file
- Click OK.

3) Import Verilog:
- File -> Import -> Verilog
- Browse and add miniUART_built.v (your design) AND OSU_diglib_ami06_gates.v
- Add the Verilog Top Module (bidir) to the form
- Click OK.
```

```
4) Initialize Floorplan:
- Floorplan -> Initialize Floorplan
- Select "Flip Every Other Row" and "Abut Rows"
- Change the IO To Core Distance to 20 microns for each
- Change Cell Density to 75%
- Click OK

5) Place IO's and Cells:
- Place -> IOs, Click OK
- Place -> Cells, Click OK
- Place -> Filler Cells -> Add
Model: fill1
Prefix: fill1
Placement: ONLY North and Flip South checked.

Special Pins:
Special Pin     Special Net
vdd!            vdd!
gnd!            gnd!
- Click OK.

6) Routing:
- Route -> Plan Power
- Select Add Rings

Fill in the Core Ring Width to the desired width (3.00 microns for both,
for example) Sizing is important when importing the vias back into cadence?

- Click OK
- Click Close
- Route -> Connect Ring
- Click OK
- Route -> WRoute, Select Global and Final Route and Auto Search and Repair
- Click OK

7) Export Design:
- File -> Export -> DEF
- Fill in name (miniUART.def)
Select:
Cells              Modifications
Nets               Layout Modifications
Special Nets
Vias
Groups
- Click OK
- Exit Silicon Ensemble

Import into Cadence:
1) Create a new library (miniUART) attached to the NCSU_TechLib_ami06 library.
2) Import the Verilog netlist as a schematic
- File -> Import -> Verilog
Target Library Name : miniUART
Reference Libraries : basic OSU_diglib_ami06
Verilog Files To Import : miniUART_built.v
Check "Overwrite Existing Views" if necessary.
- Click OK.

3) Import Layout
- File -> Import -> DEF
Library Name: miniUART
Cell Name: miniUART
View Name: layout
Use Ref. Library Names: OSU_diglibimport_ami06
DEF File Name: <path/filename> (~/flow_tutorial/se/miniUART.def)
```

```
Make sure you select the new cellview and save it.

4) To properly turn the "abstract" cells into "layout" cells in the imported
layout view, go to the Library Manager and select Edit -> Rename Reference
Library.

You may need to go to the location of the layout file, as in

~/<Cadence Directory>/miniUART/miniUART/layout/

and delete the layout.cdb.cdslck file if it exists.

In Library: miniUART
From Library: OSU_diglibimport_ami06
To Library: OSU_diglib_ami06
Click OK.
```

APPENDIX B

Abstract Generator Tutorial for Standard Cells

```
This tutorial assumes you have Cadence version IC50 with Abstract Generator
version 5.0.32.11.  The purpose of this document is to show how to take standard
cell library layouts created with  Cadence Virtuoso and create a set of abstract
cells that can be exported as a .LEF file and used in Silicon Ensemble for
automatic place & route.

(If you have pins residing in other layers of the cell (such as our vdd! and
gnd! pins), be sure to select all areas of the cell in Virtuoso, select:

Edit -> Hierarchy -> Flatten.
Flatten Mode : one level.
Flatten Pcells and Preserve Pins checked.

To start the abstract generation process:

-Open one of the cell layouts in Virtuoso
-Select:
Tools -> Abstract Editor.

-There should now be a new option in the menu: Abstract.
-Select:
Abstract -> Create Abstract.


-In the "Create Advanced Abstract" window, select Library.


-All cells should be imported into the Core bin.  If the cell has a valid layout,
there should be a checkmark beside the cell in the Layout column.  For any cells
that you do not wish to create abstracts (pads, test cells, etc),
you can move to the Ignore bin.

-To move a cell to the Ignore bin:
    -Click on the cell.
    -Select Cells -> Move.
    -Select the Ignore bin and click "OK".
    -Multiple cells can be selected at a time and moved in this manner.

We can now start creating abstracts for the cells remaining in the Core bin.

-Select Cells -> Select All to highlight all the cells.
- Select Flow -> Pins
- The default options should be fine.
- Click  "RUN"
```

```
-Select Cells -> Extract
-DESELECT: Extract Signal Nets
-DESELECT: Extract Power Nets
 This is because only premade pins should be extracted.

WARNING:
Due to a bug in the Abstract Generator, the Abstract options window is
almost 1000 pixels tall with no option to resize the window to get to
the run button.  You may need to adjust the resolution of your screen
to use this function.

-Select Cells -> Abstract
-Blockage:
Layer Assignment for Blockage: select metal1, metal2, metal3, via, and via2.

Check any warnings you receive.  Some may not be a problem, however, some
warnings could result from an error in the layout or in the abstract
generation process.


Currently, the Verify function does not work, due to compatibility problems between
the Abstract Generator and Silicon Ensemble

Export as LEF:

File -> Export -> LEF

-Choose a filename (default is abstract.lef)
-Make sure Export Geometry LEF Data and Export Tech LEF Data are checked.
-Make sure LEF Version is set to: 5.3
-Export LEF for Bin is set to: ALL
```

APPENDIX C

Synopsis Library

Synopsis Library for OSU Standard Cell Library

```
library (OSU_diglib_ami06) {

cell(dffpc) {
area : 1116 ;
vhdl_name : "dffpc" ;

pin(CLK) {
direction : input;
fanout_load :4;
}

pin (D) {
direction : input;
fanout_load : 1;
}

pin(CLR) {
direction : input;
fanout_load : 3;
}

pin (PRE) {
direction : input;
fanout_load : 4;
}

ff("IQ", "IQN"){
next_state : "D"
clocked_on : "CLK"
clear : "CLR'"
preset : "PRE"
clear_preset_var1 : L;
clear_preset_var2 : H;
}

pin(QP) {
direction : output;
max_fanout : 50;
function : "IQ";
internal_node : "QP";
```

```
}

pin(QN) {
direction : output;
max_fanout : 50;
function : "IQN";
internal_node : "QN";
}

}


cell(xor2x1) {
area : 252 ;
vhdl_name : "xor2x1" ;

pin(B) {
direction : input;
fanout_load : 1;
}
pin(A) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "(A*B')+(A'*B)";
}
}
cell(xnor2x1) {
area : 252 ;
vhdl_name : "xnor2x1" ;

pin(B) {
direction : input;
fanout_load : 1;
}
pin(A) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "((A*B')+(A'*B))'";
}
}

/*--------------*/

cell(nor4x1) {
area : 180 ;
vhdl_name : "nor4x1" ;

pin(Y) {
direction : output;
max_fanout : 50;
function : "(A+B+C+D)'";
}
pin(C) {
direction : input;
fanout_load : 1;
}
pin(B) {
```

```
direction : input;
fanout_load : 1;
}
pin(D) {
direction : input;
fanout_load : 1;
}
pin(A) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(nor3x1) {
area : 144 ;
vhdl_name : "nor3x1" ;

pin(C) {
direction : input;
fanout_load : 1;
}
pin(B) {
direction : input;
fanout_load : 1;
}
pin(A) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "(A+B+C)'";
}
}

/*--------------*/

cell(nor2x1) {
area : 108 ;
vhdl_name : "nor2x1" ;

pin(A) {
direction : input;
fanout_load : 1;
}
pin(B) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "(A+B)'";
}
}

/*--------------*/

cell(nand4x1) {
area : 180 ;
vhdl_name : "nand4x1" ;
```

```
pin(D) {
direction : input;
fanout_load : 1;
}
pin(B) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "(A*B*C*D)'";
}
pin(C) {
direction : input;
fanout_load : 1;
}
pin(A) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(nand3x1) {
area : 144 ;
vhdl_name : "nand3x1" ;

pin(A) {
direction : input;
fanout_load : 1;
}
pin(B) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "(A*B*C)'";
}
pin(C) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(nand2x1) {
area : 108 ;
vhdl_name : "nand2x1" ;

pin(B) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "(A*B)'";
}
pin(A) {
direction : input;
```

```
fanout_load : 1;
}
}

/*--------------*/

cell(mux21x1) {
area : 342.72 ;
vhdl_name : "mux21x1" ;

pin(Y) {
direction : output;
max_fanout : 50;
function : "(A*Sel)+(B*Sel')";
}
pin(A) {
direction : input;
fanout_load : 1;
}
pin(Sel) {
direction : input;
fanout_load : 1;
}
pin(B) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(invx1) {
area : 72 ;
vhdl_name : "invx1" ;

pin(A) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "A'";
}
}

/*--------------*/

cell(dff) {
area : 828 ;
vhdl_name : "dff" ;

pin(D) {
direction : input;
fanout_load : 1;
}
pin(CLK) {
direction : input;
fanout_load : 1;
}
ff("IQ", "IQN"){
next_state : "D"
clocked_on : "CLK"
}
pin(QN) {
```

```
direction : output;
max_fanout : 50;
function : "IQN";
}

pin(QP) {
direction : output;
max_fanout : 50;
function : "IQ";
}
}

/*--------------*/

cell(bufx1) {
area : 108 ;
vhdl_name : "bufx1" ;

pin(Y) {
direction : output;
max_fanout : 50;
function : "A";
}
pin(A) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(aoi22x1) {
area : 180 ;
vhdl_name : "aoi22x1" ;

pin(A) {
direction : input;
fanout_load : 1;
}
pin(B) {
direction : input;
fanout_load : 1;
}
pin(C) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "((A*B)+(C*D))'";
}
pin(D) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(ao22x1) {
area : 252 ;
vhdl_name : "ao22x1" ;

pin(Y) {
```

```
direction : output;
max_fanout : 50;
function : "(A*B)+(C*D)";
}
pin(B) {
direction : input;
fanout_load : 1;
}
pin(C) {
direction : input;
fanout_load : 1;
}
pin(D) {
direction : input;
fanout_load : 1;
}
pin(A) {
direction : input;
fanout_load : 1;
}
}

/*--------------*/

cell(bufzx1) {
area : 288 ;
vhdl_name : "bufzx1" ;

pin(A) {
        direction : input;
        fanout_load : 1;
}
pin(TriState) {
        direction : input;
        fanout_load : 1;
}

pin(Y) {
        direction : output;
        max_fanout : 50;
        function : "A";
        three_state : "TriState";
}

}
/*--------------*/

cell(tiehigh) {
area : 72 ;
vhdl_name : "tiehigh" ;

pin(HI) {
        direction : output;
        max_fanout : 50;
        function : "0";
}

}
/*--------------*/



cell(tielow) {
area : 72 ;
vhdl_name : "tielow" ;
```

```
pin(LO) {
        direction : output;
        max_fanout : 50;
        function : "1";
}
}
/*--------------*/

cell(bufx4) {
area : 216 ;
vhdl_name : "bufx4" ;

pin(Y) {
direction : output;
max_fanout : 50;
function : "A";
}
pin(A) {
direction : input;
fanout_load : 1;
}
}

cell(lat) {
area : 360 ;
vhdl_name : "lat" ;

pin(EN) {
direction : input;
fanout_load : 1;
}

pin(D) {
direction : input;
fanout_load : 1;
}

latch("IQ", "IQN") {
enable : "EN"
data_in : "D"
}

pin(QP) {
direction : output;
max_fanout : 50;
function : "IQ";
}

pin(QN) {
direction : output;
max_fanout : 50;
function : "IQN";
}
}


cell(latpc) {
area : 540 ;
vhdl_name : "latpc" ;

pin(EN) {
direction : input;
fanout_load : 1;
}
```

```
pin(PRE) {
direction : input;
fanout_load : 1;
}

pin(CLR) {
direction : input;
fanout_load : 1;
}
pin(D) {
direction : input;
fanout_load : 1;
}

latch("IQ", "IQN") {
enable : "EN"
data_in : "D"
clear : "CLR"
preset : "PRE"
clear_preset_var1 : L;
clear_preset_var2 : H;
}

pin(QP) {
direction : output;
max_fanout : 50;
function : "IQ";
}

pin(QN) {
direction : output;
max_fanout : 50;
function : "IQN";
}
}

/*--------------*/

cell(invx4) {
area : 180 ;
vhdl_name : "invx4" ;

pin(A) {
direction : input;
fanout_load : 1;
}
pin(Y) {
direction : output;
max_fanout : 50;
function : "A'";
}
}

/*--------------*/

cell(invzx1) {
area : 216 ;
vhdl_name : "invzx1" ;

pin(A) {
        direction : input;
        fanout_load : 1;
}
pin(TriState) {
```

```
        direction : input;
        fanout_load : 1;
}

pin(Y) {
        direction : output;
        max_fanout : 50;
        function : "A'";
        three_state : "TriState";
}
}



/*--------------*/
}
```

APPENDIX D

Symbol Library

The symbol library used in BuildGates Extreme is found in Figure D.1. All logic cells used by the synthesis tool are mapped to a symbol for use in the schematic view of the graphical version of BuildGates.

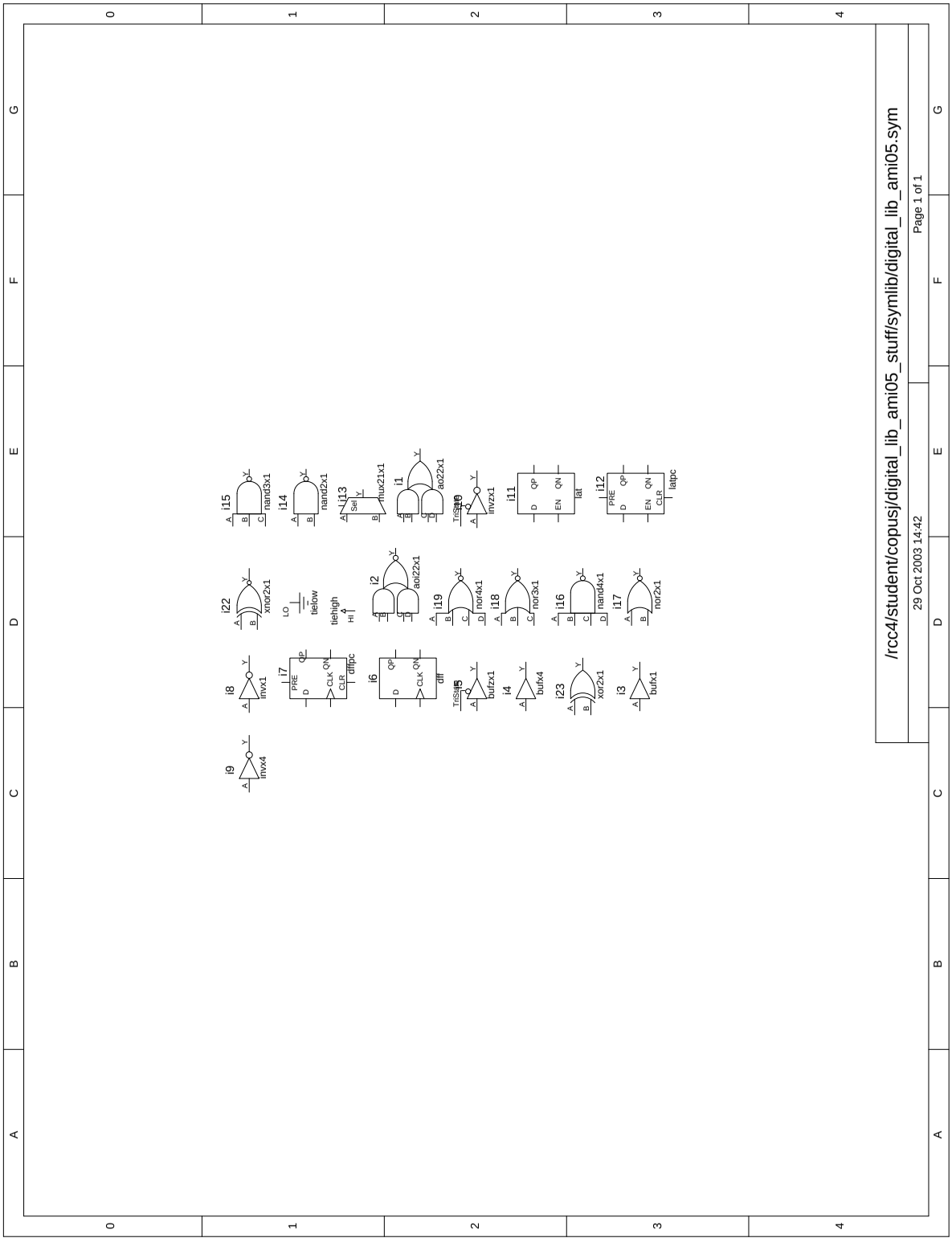

Figure D.1: BuildGates Graphical Symbol Library

APPENDIX E

LEF Conversion for Silicon Ensemble

```
#!/usr/local/bin/perl
# This script changes the LEF file created by Cadence's Abstract Generator
# so that it will import into Silicon Ensemble correctly.
#
# ./lef_abstract_to_se.perl input_file.lef > output_file.lef

while(<>) {

# add extra layer information (for some reason, SE wants this)
    if ($_ =~ /END UNITS/)
    {
print "END UNITS\n\n";
print "LAYER nwell\n TYPE  MASTERSLICE ;\nEND nwell\n\n";
print "LAYER nactive\n TYPE  MASTERSLICE ;\nEND nactive\n\n";
print "LAYER pactive\n TYPE  MASTERSLICE ;\nEND pactive\n\n";
    }
# change poly from routing layer to masterslice
    elsif($_ =~ /LAYER poly/ &! $one_instance)
    {
$poly_flag =1;
print "LAYER poly\n\tType MASTERSLICE ;\n";
    }
    elsif($_ =~ /TYPE ROUTING ;/ && $poly_flag){}
    elsif($_ =~ /WIDTH 0.60 ;/ && $poly_flag){}
    elsif($_ =~ /SPACING 0.90 ;/ && $poly_flag){}
    elsif($_ =~ /OFFSET 0.90 ;/ && $poly_flag){}
    elsif($_ =~ /PITCH 1.80 ;/ && $poly_flag){}
    elsif($_ =~ /DIRECTION VERTICAL ;/ && $poly_flag){$poly_flag = 0; $one_instance=1;}


# REMOVE VIA M1_ELEC information
    elsif ($_ =~ /VIA M1_ELEC/)
    { $flag = 1;    }
    elsif ($_ =~ /LAYER elec ;/ && $flag){}
    elsif ($_ =~ /RECT -0.90 -0.90 0.90 0.90 ;/ && $flag){}
    elsif ($_ =~ /LAYER cc ;/ && $flag){}
    elsif ($_ =~ /RECT -0.30 -0.30 0.30 0.30 ;/ && $flag){}
    elsif ($_ =~ /LAYER metal1 ;/ && $flag){}
    elsif ($_ =~ /RECT -0.60 -0.60 0.60 0.60 ;/ && $flag){}
    elsif ($_ =~ /END M1_ELEC/ && $flag){$flag = 0;}

# Add VIARULES, Site Core and Site CoreSite
    elsif ($_ =~ /END M3_M2/)
```

```
    {
print "END M3_M2\n\n";

print "VIARULE viagen21 GENERATE\n\tLAYER metal1 ;
                \n\t\tDIRECTION HORIZONTAL ;\n\t\tWIDTH 1.2 TO 120 ;\n\t\tOVERHANG 0.3;
                \n\t\tMETALOVERHANG 0 ;\n\tLAYER metal2 ;\n\t\tDIRECTION VERTICAL ;
                \n\t\tWIDTH 1.2 TO 120 ;\n\t\tOVERHANG 0.3 ;\n\t\tMETALOVERHANG 0 ;
                \n\tLAYER via ;\n\t\tRECT -0.3 -0.3 0.3 0.3 ;\n\t\tSPACING 1.5 BY 1.5 ;
                \nEND viagen21\n\n";
print "VIARULE TURN1 GENERATE\n\tLAYER metal1 ;
                \n\t\tDIRECTION HORIZONTAL ;
                \n\tLAYER metal1 ;\n\t\tDIRECTION VERTICAL ;
                \nEND TURN1\n\n";
print "VIARULE TURN2 GENERATE\n\tLAYER metal2 ;\n\t\tDIRECTION HORIZONTAL ;
                \n\tLAYER metal2 ;\n\t\tDIRECTION VERTICAL ;\nEND TURN2\n\n";
print "VIARULE TURN3 GENERATE\n\tLAYER metal3 ;\n\t\tDIRECTION HORIZONTAL ;
                \n\tLAYER metal3 ;\n\t\tDIRECTION VERTICAL ;\nEND TURN3\n\n";

print "SITE Core\n\tCLASS CORE ;\n\tSYMMETRY Y ;
                \n\tSIZE 2.400 BY 15.000 ;\nEND Core\n\n";
print "SITE CoreSite\n\tCLASS CORE ;\n\tSYMMETRY Y ;
                \n\tSIZE 2.400 BY 15.000 ;\nEND CoreSite\n\n";
    }
    else
    {
print $_;
    }

}
```

APPENDIX F

LEF Conversion to Verilog

```
#!/usr/local/bin/perl
# This script converts the LEF file containing a standard cell library into
# a verilog module listing for use with Silicon Ensemble.
#
# ./lef_to_verilog.perl input_file.lef > output_file.v

while(<>) {

    if ($_ =~ /MACRO ([^ ]*)\n/){
        $cell_name = $1;
        print "module $cell_name (";
        $flag = 1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_1_flag){
        $pin_1_data = $1;
        $pin_1_flag = 1;
    }
    elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_1_flag &! $pin_2_flag){
        $pin_1_direction = $1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_2_flag){
        $pin_2_data = $1;
        $pin_2_flag = 1;
    }
    elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_2_flag &! $pin_3_flag){
        $pin_2_direction = $1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_3_flag){
        $pin_3_data = $1;
        $pin_3_flag = 1;
    }
    elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_3_flag &! $pin_4_flag){
        $pin_3_direction = $1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_4_flag){
        $pin_4_data = $1;
        $pin_4_flag = 1;
    }
    elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_4_flag &! $pin_5_flag){
        $pin_4_direction = $1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_5_flag){
        $pin_5_data = $1;
        $pin_5_flag = 1;
```

```
    }
    elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_5_flag &! $pin_6_flag){
        $pin_5_direction = $1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_6_flag){
        $pin_6_data = $1;
        $pin_6_flag = 1;
    }
    elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_6_flag &! $pin_7_flag){
        $pin_6_direction = $1;
    }
    elsif (($_ =~ /PIN ([^ ]*)\n/) && $flag &! $pin_7_flag){
        $pin_7_data = $1;
        $pin_7_flag = 1;
    }
     elsif (($_ =~ /DIRECTION ([^ ]*) ;\n/) && $pin_7_flag){
        $pin_7_direction = $1;
    }
    elsif($_ =~ /END $cell_name\n/ && flag){
if ($pin_1_flag)
{
    print $pin_1_data;
}
if ($pin_2_flag)
{
    print ", ".$pin_2_data;
}
if ($pin_3_flag)
{
    print ", ".$pin_3_data;
}
if ($pin_4_flag)
{
    print ", ".$pin_4_data;
}
if ($pin_5_flag)
{
    print ", ".$pin_5_data;
}
if ($pin_6_flag)
{
    print ", ".$pin_6_data;
}
if ($pin_7_flag)
{
    print ", ".$pin_7_data;

}
print ") ;\n";
print "\t\tinput ";
if ($pin_1_flag && ($pin_1_direction =~ /INPUT/ || $pin_1_direction =~ /INOUT/)){
    print $pin_1_data;
}
if ($pin_2_flag && ($pin_2_direction =~ /INPUT/ || $pin_2_direction =~ /INOUT/)){
    if ($pin_1_direction =~ /INPUT/ || $pin_1_direction =~ /INOUT/){print ",";}
    print " ".$pin_2_data;
}
if ($pin_3_flag && ($pin_3_direction =~ /INPUT/ || $pin_3_direction =~/INOUT/)){
    if ($pin_1_direction =~ /INPUT/ || $pin_1_direction =~ /INOUT/ ||$pin_2_direction =~/INPUT/ || $pin_2_direction =~
    {print ",";}
    print " ".$pin_3_data;
}
if ($pin_4_flag && ($pin_4_direction =~ /INPUT/ || $pin_4_direction =~ /INOUT/)){
    if ($pin_1_direction =~ /INPUT/ || $pin_1_direction =~ /INOUT/||$pin_2_direction =~/INPUT/ || $pin_2_direction =~/I
    {print ",";}
```

```
    print " ".$pin_4_data;
}
if ($pin_5_flag && ($pin_5_direction =~ /INPUT/ || $pin_5_direction =~ /INOUT/)){
    if ($pin_1_direction =~ /INPUT/ || $pin_1_direction =~ /INOUT/||$pin_2_direction =~/INPUT/ || $pin_2_direction =~ /
    {print ",";}
    print " ".$pin_5_data;
}
if ($pin_6_flag && ($pin_6_direction =~ /INPUT/ || $pin_6_direction =~/INOUT/)){
    if ($pin_1_direction =~ /INPUT/ || /INOUT/||$pin_2_direction =~/INPUT/ || /INOUT/||$pin_3_direction =~/INPUT/ || /I
    {print ",";}
    print " ".$pin_6_data;
}
if ($pin_7_flag && ($pin_7_direction =~ /INPUT/ || $pin_7_direction =~ /INOUT/)){
    if ($pin_1_direction =~ /INPUT/ || /INOUT/||$pin_2_direction =~/INPUT/ || /INOUT/||$pin_3_direction =~/INPUT/ || /I
    {print ",";}
    print " ".$pin_7_data;
}
print ";\n";
print "\t\toutput ";

if ($pin_1_flag && ($pin_1_direction =~ /OUTPUT/)){
    print $pin_1_data;
}
if ($pin_2_flag && ($pin_2_direction =~ /OUTPUT/)){
    if ($pin_1_direction =~ /OUTPUT/){print ",";}
    print " ".$pin_2_data;
}
if ($pin_3_flag && ($pin_3_direction =~ /OUTPUT/)){
    if ($pin_1_direction =~ /OUTPUT/||$pin_2_direction =~/OUTPUT/)
    {print ",";}
    print " ".$pin_3_data;
}
if ($pin_4_flag && ($pin_4_direction =~ /OUTPUT/)){
    if ($pin_1_direction =~ /OUTPUT/||$pin_2_direction =~/OUTPUT/||$pin_3_direction =~/OUTPUT/)
    {print ",";}
    print " ".$pin_4_data;
}
if ($pin_5_flag && ($pin_5_direction =~ /OUTPUT/)){
    if ($pin_1_direction =~ /OUTPUT/||$pin_2_direction =~/OUTPUT/||$pin_3_direction =~/OUTPUT/||$pin_4_direction =~ /OU
    {print ",";}
    print " ".$pin_5_data;
}
if ($pin_6_flag && ($pin_6_direction =~ /OUTPUT/)){
    if ($pin_1_direction =~ /OUTPUT/||$pin_2_direction =~/OUTPUT/||$pin_3_direction =~/OUTPUT/||$pin_4_direction =~ /OU
    {print ",";}
    print " ".$pin_6_data;
}
if ($pin_7_flag && ($pin_7_direction =~ /OUTPUT/)){
    if ($pin_1_direction =~ /OUTPUT/||$pin_2_direction =~/OUTPUT/||$pin_3_direction =~/OUTPUT/||$pin_4_direction =~ /OU
    {print ",";}
    print " ".$pin_7_data;
}

print ";\nendmodule\n\n";
$flag=0;
$pin_1_flag=0;
$pin_2_flag=0;
$pin_3_flag=0;
$pin_4_flag=0;
$pin_5_flag=0;
$pin_6_flag=0;
$pin_7_flag=0;
    }
}
```

APPENDIX G

LEF Conversion to Synopsis LIB

```
#!/usr/local/bin/perl
# This script converts the LEF file containing a standard cell library into
# the Synopsys LIB format.  (From there, the .lib can be converted to
# the binary .alf format used by Ambit BuildGates using the libcompile
# command.
#
# ./lef_to_lib.perl input_file.lef > output_file.lib


while(<>) {

# Write file header information

if($_ =~/VERSION 5.3 ;/){
    print "library (OSU_diglib_ami06) {\n\n";
}


# Look for first macro standard cell
    elsif ($_ =~ /MACRO ([^ ]*)\n/){
        $cell_name = $1;
        $cell_flag = 1;
        print "cell(".$cell_name.") {\n";

    }
elsif(($_ =~/SIZE ([^ ]*) BY ([^ ]*) ;/) && $cell_flag) {
        $area = $1 * $2;
        print "area : ".$area." ;\n";
        print "vhdl_name : \"".$cell_name."\" ;\n\n";
}
elsif(($_ =~ /PIN ([^ ]*)\n/) && $cell_flag
      && ($_ !~ /PIN vdd\!/) && ($_ !~ /PIN gnd\!/)) {
        print "pin(".$1.") {\n";
        $valid_pin = 1;
}
elsif(($_ =~ /DIRECTION ([^ ]*) ;/) && $cell_flag && $valid_pin) {
    $direction = $1;
    $direction  =~ tr/A-Z/a-z/;
    print "\tdirection : ".$direction.";\n";
    if($direction =~ /output/) {
        print "\tmax_fanout : 50;\n";
        print "\tfunction : \"INSERT FUNCTION\";\n}\n";
    }
    else {
        print "\tfanout_load : 1;\n}\n";
```

```
    }
    $valid_pin = 0;

}
elsif(($_ =~ /END $cell_name/) && $cell_flag) {
    $cell_flag = 0;
    print "}\n\n";
    print "/*--------------*/\n\n";
}
elsif($_ =~ /END LIBRARY/) {
    print "\n}\n";
}
else{}
}
```

APPENDIX H

BuildGates Synthesis Scripts

The master script synthesizes designs for all standard cell libraries, but TCL scripts are only included for the OSU library.

H.1 Master Script

Shell script that calls multiple instances of BuildGates Extreme. Each instance of BuildGates uses its own TCL script.

```
# Calls multiple instances of BuildGates Extreme
# with .tcl scripts controlling the action

source /opt/local/cadence/StartupSPR.EE

# UT_LP_AMI06
nohup nice bgx_shell ~/synthesis/UT_LP_AMI06/synth_UT_LP_AMI06_avr.tcl
nohup nice bgx_shell ~/synthesis/UT_LP_AMI06/synth_UT_LP_AMI06_bus.tcl
nohup nice bgx_shell ~/synthesis/UT_LP_AMI06/synth_UT_LP_AMI06_miniUART.tcl

# iit_stdcells
nohup nice bgx_shell ~/synthesis/iit_stdcells/synth_iit_avr.tcl
nohup nice bgx_shell ~/synthesis/iit_stdcells/synth_iit_bus.tcl
nohup nice bgx_shell ~/synthesis/iit_stdcells/synth_iit_miniUART.tcl

# msu_jennings
nohup nice bgx_shell ~/synthesis/msu_jennings/synth_msu_bus.tcl
nohup nice bgx_shell ~/synthesis/msu_jennings/synth_msu_miniUART.tcl
nohup nice bgx_shell ~/synthesis/msu_jennings/synth_msu_avr.tcl

# OSU_diglib_ami06
nohup nice bgx_shell ~/synthesis/OSU_diglib_ami06/synth_osu_bus.tcl
nohup nice bgx_shell ~/synthesis/OSU_diglib_ami06/synth_osu_miniUART.tcl
nohup nice bgx_shell ~/synthesis/OSU_diglib_ami06/synth_osu_avr.tcl
```

H.2 Bidirectional Bus Synthesis Script

Targetted for the OSU Standard Cell Library

```
# OSU_diglib_ami06 library synthesis
read_alf ~/synthesis/OSU_diglib_ami06/OSU_diglib_ami06.alf

# Bidirectional Bus
read_vhdl ~/nc/bus/bus.vhd
do_build_generic
do_optimize -priority area
write_verilog -hierarchical OSU_diglib_ami06_bus.v
report_area -cells -summary OSU_diglib_ami06_bus.summary
exit
```

H.3 MiniUART Synthesis Script

Targetted for the OSU Standard Cell Library

```
# OSU_diglib_ami06 library synthesis
read_alf ~/synthesis/OSU_diglib_ami06/OSU_diglib_ami06.alf
# miniUART
read_vhdl ~/flow_tutorial/RxUnit.vhd
read_vhdl ~/flow_tutorial/TxUnit.vhd
read_vhdl ~/flow_tutorial/clkUnit.vhd
read_vhdl ~/flow_tutorial/miniUART.vhd
do_build_generic
do_optimize -priority area
write_verilog -hierarchical OSU_diglib_ami06_miniUART.v
report_area -cells -summary OSU_diglib_ami06_miniUART.summary
exit
```

H.4 AVR Core Synthesis Script

Targetted for the OSU Standard Cell Library

```
# OSU_diglib_ami06 library synthesis
read_alf ~/synthesis/OSU_diglib_ami06/OSU_diglib_ami06.alf
# AVR Microprocessor Core
read_vhdl ~/nc/avr/AVRuCPackage.vhd
read_vhdl ~/nc/avr/reg_file.vhd
read_vhdl ~/nc/avr/io_reg_file.vhd
read_vhdl ~/nc/avr/io_adr_dec.vhd
read_vhdl ~/nc/avr/bit_processor.vhd
read_vhdl ~/nc/avr/pm_fetch_dec.vhd
read_vhdl ~/nc/avr/alu_avr.vhd
read_vhdl ~/nc/avr/avr_core.vhd
do_build_generic
do_optimize -priority area
write_verilog -hierarchical OSU_diglib_ami06_avr.v
report_area -cells -summary OSU_diglib_ami06_avr.summary
exit
```

APPENDIX I

Silicon Ensemble Pin Constraint Format

Sample constraint file that Silicon Ensemble uses to place pins. This is the pin listing for the ALU module in the AVR microprocessor core.

```
# for alu_avr module

LEFT ( # ordered from BOTTOM to TOP

   (IOPIN alu_c_flag_in );
   (IOPIN alu_z_flag_in );
   (IOPIN idc_add );
   (IOPIN idc_adc );
   (IOPIN idc_adiw );
   (IOPIN idc_sub );
   (IOPIN idc_subi );
   (IOPIN idc_sbc );
   (IOPIN idc_sbci );
   (IOPIN idc_sbiw );
   (IOPIN adiw_st );
   (IOPIN sbiw_st );
   (IOPIN idc_and );
   (IOPIN idc_andi );
   (IOPIN idc_or );
   (IOPIN idc_ori );
   (IOPIN idc_eor );
   (IOPIN idc_com );
   (IOPIN idc_neg );
   (IOPIN idc_inc );
   (IOPIN idc_dec );
   (IOPIN idc_cp );
   (IOPIN idc_cpc );
   (IOPIN idc_cpi );
   (IOPIN idc_cpse );
   (IOPIN idc_lsr );
   (IOPIN idc_ror );
   (IOPIN idc_asr );
   (IOPIN idc_swap );
)

TOP ( # ordered from left to right
   (IOPIN alu_data_r_in[0] );
   (IOPIN alu_data_r_in[1] );
```

```
   (IOPIN alu_data_r_in[2] );
   (IOPIN alu_data_r_in[3] );
   (IOPIN alu_data_r_in[4] );
   (IOPIN alu_data_r_in[5] );
   (IOPIN alu_data_r_in[6] );
   (IOPIN alu_data_r_in[7] );

   (IOPIN alu_data_d_in[0] );
   (IOPIN alu_data_d_in[1] );
   (IOPIN alu_data_d_in[2] );
   (IOPIN alu_data_d_in[3] );
   (IOPIN alu_data_d_in[4] );
   (IOPIN alu_data_d_in[5] );
   (IOPIN alu_data_d_in[6] );
   (IOPIN alu_data_d_in[7] );



)

BOTTOM ( # ordered from left to right
   (IOPIN alu_data_out[0] );
   (IOPIN alu_data_out[1] );
   (IOPIN alu_data_out[2] );
   (IOPIN alu_data_out[3] );
   (IOPIN alu_data_out[4] );
   (IOPIN alu_data_out[5] );
   (IOPIN alu_data_out[6] );
   (IOPIN alu_data_out[7] );
   (IOPIN alu_c_flag_out );
   (IOPIN alu_z_flag_out );
   (IOPIN alu_n_flag_out );
   (IOPIN alu_v_flag_out );
   (IOPIN alu_s_flag_out );
   (IOPIN alu_h_flag_out );
)
```

APPENDIX J

Silicon Ensemble Place & Route Scripts

Included is the master shell script that calls the Silicon Ensemble program, which uses its own command script. Only one command script is included, since the command structure is almost identical, with only varying file name, module name, and area information commands, in most cases.

J.1 Shell Script

The master shell script calls multiple instances of Silicon Ensemble.

```
# P&R OSU Designs

echo "P&R OSU bus"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE osu_bus.mac"
      | grep -e violations -e ERROR > osu_bus_errors.txt # 85%
echo "P&R OSU miniUART"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE osu_miniUART.mac"
      | grep -e violations -e ERROR > osu_miniUART_errors.txt # 80%
echo "P&R OSU avr"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE osu_avr.mac"
      | grep -e violations -e ERROR > osu_avr_errors.txt # 35%
echo "Finished OSU designs"

# P&R IIT Designs
echo "P&R IIT bus"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE iit_bus.mac"
      | grep -e violations -e ERROR > iit_bus_errors.txt #85%
echo "P&R IIT miniUART"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE iit_miniUART.mac"
      | grep -e violations -e ERROR > iit_miniUART_errors.txt # 85%
echo "Finished IIT designs"

# P&R MSU Designs
echo "P&R MSU miniUART"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE msu_miniUART.mac"
```

```
      | grep -e violations -e ERROR > msu_miniUART_errors.txt # 85%
echo "P&R MSU avr"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE msu_avr.mac"
      | grep -e violations -e ERROR > msu_avr_errors.txt
echo "Finished MSU Designs"



# P&R UT Designs
echo "P&R UT miniUART"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE ut_miniUART.mac"
      | grep -e violations -e ERROR > ut_miniUART_errors.txt # 85 %
echo "P&R UT avr"
nohup nice sedsm -gd=ansi -m=500 "EXECUTE ut_avr.mac"
      | grep -e violations -e ERROR > ut_avr_errors.txt # 50%
echo "Finished UT Designs"
```

J.2 Bidirectional Bus Place & Route Script

Targetted for the OSU Standard Cell Library

```
 set v draw.row.at ON;

 FINPUT LEF FILENAME "~/synthesis/OSU_diglib_ami06/OSU_diglib_ami06.lef"
        REPORTFILE "importlef.rpt" ;
 SET VAR INPUT.VERILOG.POWER.NET "vdd!";
 SET VAR INPUT.VERILOG.GROUND.NET "gnd!";
 SET VAR INPUT.VERILOG.LOGIC.1.NET "vdd!";
 SET VAR INPUT.VERILOG.LOGIC.0.NET "gnd!";
 SET VAR INPUT.VERILOG.SPECIAL.NETS "vdd! gnd!";
 INPUT VERILOG FILE "OSU_diglib_ami06/OSU_diglib_ami06_gates.v OSU_diglib_ami06_bus.v"
        LIB "cds_vbin" REFLIB "cds_vbin " DESIGN "cds_vbin.bidir:hdl" ;
 set v USERLEVEL EXPERT;
 set v PLAN.REPORT.STAT "  ";
 FINIT FLOOR;
 FINIT FLOORPLAN  rowu 0.85 rowsp 0 blockhalo 2000 f a 1 abut xio 3000 yio 3000 ;
 set v UPDATECOREROW.BLOCKHALO.GLOBAL 2000;
 IOPLACE AUTOMATIC STYLE EVEN ;

 SET VAR QPWR.RSPF ""
;
 SET VAR QPLACE.PLACE.GROUTE.ANALYSIS ""
;
 SET VAR QPLACE.OPT.TIMING.TYPE ""
;
 SET VAR QPLACE.PLACE.PIN ""
;
 QPLACE NOCONFIG
;
 SROUTE ADDCELL MODEL fill1 PREFIX fill1 NO FS SPIN vdd!
       NET vdd! SPIN gnd! NET gnd! AREA ( -21840 -21600 ) ( 22080 21900 ) ;
 SET VAR DRAW.SWIRE.AT ON ;
 SET VAR DRAW.CHANNEL.AT ON ;
 BUILD CHANNEL ;
 CONSTRUCT RING
 NET "gnd!"
 NET "vdd!"
 LAYER metal1 CORERINGWIDTH 240 SPACING CENTER BLOCKRINGWIDTH 0
 LAYER metal2 CORERINGWIDTH 240 SPACING CENTER BLOCKRINGWIDTH 0
  ;
```

```
;
 DISPOSE CHANNEL ;
 SET VAR DRAW.CHANNEL.AT OFF ;
 SET VAR DRAW.SWIRE.AT SMALL ;
 CONNECT RING NET "gnd!" NET "vdd!"
        STRIPE BLOCK ALLPORT IOPAD ALLPORT IORING FOLLOWPIN ;
 SET VAR WROUTE.FINAL TRUE
;
 SET VAR WROUTE.GLOBAL TRUE
;
 SET VAR WROUTE.INCREMENTAL.FINAL FALSE
;
 WROUTE NOCONFIG
;

 OUTPUT DEF  CELLS  NETS  SPECIALNETS  VIAS
        GROUPS  MODIFICATIONS  EXTPIN  LAYOUTMODIFICATION  FILENAME  "osu_bus.def" ;
 FQUIT ;
```

APPENDIX K

Synthesis Summary Reports

The summary files produced by BuildGates Extreme show such things as the number of cells in each stage of the design hierarchy, the number of nets in the design, and the total cell area of the design.

K.1 OSU AVR Microprocessor

```
+-----------------------------------------------------+
| Report  | report_area                               |
|---------+-------------------------------------------|
| Options | -cells -summary OSU_diglib_ami06_avr.summary |
+---------+-------------------------------------------+
| Date    | 20031016.202758                           |
| Tool    | bgx_shell                                 |
| Release | v5.11-s079                                |
| Version | Aug 19 2003 04:51:32                      |
+---------+-------------------------------------------+
| Module  | avr_core                                  |
+-----------------------------------------------------+

Summary Area Report
Source of Area : Timing Library
-------------------
+-------------------------------------------------------------------------+
|                 Module |   Wireload |  Cell Area |   Net Area | Total Area |
|------------------------+------------+------------+------------+------------|
|               avr_core |       NONE | 1024963.20 |       0.00 | 1024963.20 |
|                alu_avr |       NONE |   36092.16 |       0.00 |   36092.16 |
|          bit_processor |       NONE |   26308.80 |       0.00 |   26308.80 |
|             io_adr_dec |       NONE |    6228.00 |       0.00 |    6228.00 |
|            io_reg_file |       NONE |   74629.44 |       0.00 |   74629.44 |
|           pm_fetch_dec |       NONE |  378146.88 |       0.00 |  378146.88 |
| reg_file_ResetRegFile1 |       NONE |  503557.92 |       0.00 |  503557.92 |
|            AWDP_DEC_00 |       NONE |    8028.00 |       0.00 |    8028.00 |
|            AWDP_INC_11 |       NONE |    9216.00 |       0.00 |    9216.00 |
|             AWDP_ADD_0 |       NONE |    9504.00 |       0.00 |    9504.00 |
|            AWDP_INC_98 |       NONE |    9216.00 |       0.00 |    9216.00 |
|       AWRS_partition_2 |       NONE |   15930.72 |       0.00 |   15930.72 |
|       AWDP_partition_2 |       NONE |   11952.00 |       0.00 |   11952.00 |
```

```
|             AWDP_DEC_0 |       NONE |    8028.00 |       0.00 |    8028.00 |
|            AWDP_INC_10 |       NONE |    9216.00 |       0.00 |    9216.00 |
+-------------------------------------------------------------------------------+

+-------------------------------------------------------------+
|  Block report for module 'avr_core'  | Current | Cumulative |
|                                      |  Module |            |
|--------------------------------------+---------+------------|
| Number of combinational instances    |       0 |       3116 |
| Number of noncombinational instances |       0 |        468 |
| Number of hierarchical instances     |       6 |         14 |
| Number of blackbox instances         |       0 |          0 |
| Total number of instances            |       6 |       3598 |
| Area of combinational cells          |    0.00 |  502675.20 |
| Area of non-combinational cells      |    0.00 |  522288.00 |
| Total cell area                      |    0.00 | 1024963.20 |
| Number of nets                       |     336 |       3750 |
| Area of nets                         |    0.00 |       0.00 |
| Total area                           |    0.00 | 1024963.20 |
+-------------------------------------------------------------+

+-----------------------------------------------------------------------------------------+
|                                    Cell Usage Table                                     |
|-----------------------------------------------------------------------------------------|
|                Cellref |    Library |  Number of |  Cell Type |  Cell Area | Total Area |
|                        |            |  Instances |            |            |            |
|------------------------+------------+------------+------------+------------+------------|
|                alu_avr |    netlist |          1 |       hier |   36092.16 |   36092.16 |
|          bit_processor |    netlist |          1 |       hier |   26308.80 |   26308.80 |
|             io_adr_dec |    netlist |          1 |       hier |    6228.00 |    6228.00 |
|            io_reg_file |    netlist |          1 |       hier |   74629.44 |   74629.44 |
|           pm_fetch_dec |    netlist |          1 |       hier |  378146.88 |  378146.88 |
| reg_file_ResetRegFile1 |    netlist |          1 |       hier |  503557.92 |  503557.92 |
+-----------------------------------------------------------------------------------------+
```

K.2 OSU Bidirectonal Bus

```
+-------------------------------------------------------+
| Report  | report_area                                 |
|---------+---------------------------------------------|
| Options | -cells -summary OSU_diglib_ami06_bus.summary  |
+---------+---------------------------------------------+
| Date    | 20031016.202509                             |
| Tool    | bgx_shell                                   |
| Release | v5.11-s079                                  |
| Version | Aug 19 2003 04:51:32                        |
+---------+---------------------------------------------+
| Module  | bidir                                       |
+-------------------------------------------------------+

Summary Area Report
Source of Area : Timing Library
-------------------
+-----------------------------------------------------------------+
|     Module |   Wireload |  Cell Area |   Net Area | Total Area |
|------------+------------+------------+------------+------------|
|      bidir |       NONE |   15048.00 |       0.00 |   15048.00 |
+-----------------------------------------------------------------+

+-------------------------------------------------------+
```

```
|   Block report for module 'bidir'    |  Current | Cumulative |
|                                      |  Module  |            |
|--------------------------------------+----------+------------|
| Number of combinational instances    |        1 |          1 |
| Number of noncombinational instances |       24 |         24 |
| Number of hierarchical instances     |        0 |          0 |
| Number of blackbox instances         |        0 |          0 |
| Total number of instances            |       25 |         25 |
| Area of combinational cells          |    72.00 |      72.00 |
| Area of non-combinational cells      | 14976.00 |   14976.00 |
| Total cell area                      | 15048.00 |   15048.00 |
| Number of nets                       |       35 |         35 |
| Area of nets                         |     0.00 |       0.00 |
| Total area                           | 15048.00 |   15048.00 |
+--------------------------------------------------------------+

+-------------------------------------------------------------------------------------+
|                                  Cell Usage Table                                   |
|-------------------------------------------------------------------------------------|
|    Cellref |           Library |  Number of |  Cell Type |  Cell Area | Total Area |
|            |                   |  Instances |            |            |            |
|------------+-------------------+------------+------------+------------+------------|
|      invx1 | OSU_diglib_ami06 |          1 |       comb |      72.00 |      72.00 |
|     invzx1 | OSU_diglib_ami06 |          8 |    noncomb |     216.00 |    1728.00 |
|        dff | OSU_diglib_ami06 |         16 |    noncomb |     828.00 |   13248.00 |
+-------------------------------------------------------------------------------------+
```

K.3 OSU miniUART

```
+-------------------------------------------------------------+
| Report  | report_area                                       |
|---------+---------------------------------------------------|
| Options | -cells -summary OSU_diglib_ami06_miniUART.summary  |
+---------+---------------------------------------------------+
| Date    | 20031016.202524                                   |
| Tool    | bgx_shell                                         |
| Release | v5.11-s079                                        |
| Version | Aug 19 2003 04:51:32                              |
+---------+---------------------------------------------------+
| Module  | miniUART                                          |
+-------------------------------------------------------------+

Summary Area Report
Source of Area : Timing Library
-------------------
+-------------------------------------------------------------------+
|     Module |   Wireload |  Cell Area |   Net Area | Total Area |
|------------+------------+------------+------------+------------|
|   miniUART |       NONE |  122505.12 |       0.00 |  122505.12 |
|    ClkUnit |       NONE |   23654.88 |       0.00 |   23654.88 |
|     RxUnit |       NONE |   41312.16 |       0.00 |   41312.16 |
|     TxUnit |       NONE |   35248.32 |       0.00 |   35248.32 |
| AWDP_DEC_1 |       NONE |    3708.00 |       0.00 |    3708.00 |
+-------------------------------------------------------------------+

+--------------------------------------------------------------+
|  Block report for module 'miniUART'  |  Current | Cumulative |
|                                      |  Module  |            |
```

Number of combinational instances	51	389
Number of noncombinational instances	18	85
Number of hierarchical instances	3	4
Number of blackbox instances	0	0
Total number of instances	72	478
Area of combinational cells	7385.76	52125.12
Area of non-combinational cells	14904.00	70380.00
Total cell area	22289.76	122505.12
Number of nets	101	502
Area of nets	0.00	0.00
Total area	22289.76	122505.12

Cell Usage Table

Cellref	Library	Number of Instances	Cell Type	Cell Area	Total Area
ClkUnit	netlist	1	hier	23654.88	23654.88
RxUnit	netlist	1	hier	41312.16	41312.16
TxUnit	netlist	1	hier	35248.32	35248.32
nand4x1	OSU_diglib_ami06	1	comb	180.00	180.00
tiehigh	OSU_diglib_ami06	1	comb	72.00	72.00
nor3x1	OSU_diglib_ami06	3	comb	144.00	432.00
nand3x1	OSU_diglib_ami06	6	comb	144.00	864.00
mux21x1	OSU_diglib_ami06	8	comb	342.72	2741.76
nor2x1	OSU_diglib_ami06	9	comb	108.00	972.00
invx1	OSU_diglib_ami06	10	comb	72.00	720.00
nand2x1	OSU_diglib_ami06	13	comb	108.00	1404.00
dff	OSU_diglib_ami06	18	noncomb	828.00	14904.00

K.4 UT AVR Microprocessor

Report	report_area
Options	-cells -summary UT_LP_AMI06_avr.summary
Date	20031016.201856
Tool	bgx_shell
Release	v5.11-s079
Version	Aug 19 2003 04:51:32
Module	avr_core

Summary Area Report
Source of Area : Timing Library

Module	Wireload	Cell Area	Net Area	Total Area
avr_core	NONE	1122249.62	0.00	1122249.62
alu_avr	NONE	43113.60	0.00	43113.60
bit_processor	NONE	29030.40	0.00	29030.40
io_adr_dec	NONE	7646.40	0.00	7646.40
io_reg_file	NONE	83635.20	0.00	83635.20
pm_fetch_dec	NONE	416923.21	0.00	416923.21
reg_file_ResetRegFile1	NONE	541900.81	0.00	541900.81

```
|            AWDP_DEC_00 |       NONE |   10497.60 |       0.00 |   10497.60 |
|            AWDP_INC_11 |       NONE |   11318.40 |       0.00 |   11318.40 |
|       AWRS_partition_2 |       NONE |   19137.60 |       0.00 |   19137.60 |
|       AWRS_partition_3 |       NONE |   18230.40 |       0.00 |   18230.40 |
|       AWDP_partition_2 |       NONE |   14428.80 |       0.00 |   14428.80 |
|       AWDP_partition_3 |       NONE |   11404.80 |       0.00 |   11404.80 |
|             AWDP_DEC_0 |       NONE |   10497.60 |       0.00 |   10497.60 |
|            AWDP_INC_10 |       NONE |   11318.40 |       0.00 |   11318.40 |
+-------------------------------------------------------------------------+

+-----------------------------------------------------------+
|  Block report for module 'avr_core'  | Current | Cumulative |
|                                      |  Module |            |
|--------------------------------------+---------+------------|
| Number of combinational instances    |       0 |       2971 |
| Number of noncombinational instances |       0 |        468 |
| Number of hierarchical instances     |       6 |         14 |
| Number of blackbox instances         |       0 |          0 |
| Total number of instances            |       6 |       3453 |
| Area of combinational cells          |    0.00 |  576374.41 |
| Area of non-combinational cells      |    0.00 |  545875.21 |
| Total cell area                      |    0.00 | 1122249.62 |
| Number of nets                       |     336 |       3612 |
| Area of nets                         |    0.00 |       0.00 |
| Total area                           |    0.00 | 1122249.62 |
+-----------------------------------------------------------+

+-------------------------------------------------------------------------------------------+
|                                    Cell Usage Table                                       |
|-------------------------------------------------------------------------------------------|
|                Cellref |    Library |  Number of |  Cell Type |  Cell Area | Total Area |
|                        |            |  Instances |            |            |            |
|------------------------+------------+------------+------------+------------+------------|
|                alu_avr |    netlist |          1 |       hier |   43113.60 |   43113.60 |
|          bit_processor |    netlist |          1 |       hier |   29030.40 |   29030.40 |
|             io_adr_dec |    netlist |          1 |       hier |    7646.40 |    7646.40 |
|            io_reg_file |    netlist |          1 |       hier |   83635.20 |   83635.20 |
|           pm_fetch_dec |    netlist |          1 |       hier |  416923.21 |  416923.21 |
| reg_file_ResetRegFile1 |    netlist |          1 |       hier |  541900.81 |  541900.81 |
+-------------------------------------------------------------------------------------------+
```

K.5 UT miniUART

```
+---------------------------------------------------+
| Report  | report_area                             |
|---------+-----------------------------------------|
| Options | -cells -summary UT_LP_AMI06_miniUART.summary |
+---------+-----------------------------------------+
| Date    | 20031016.201918                         |
| Tool    | bgx_shell                               |
| Release | v5.11-s079                              |
| Version | Aug 19 2003 04:51:32                    |
+---------+-----------------------------------------+
| Module  | miniUART                                |
+---------------------------------------------------+

Summary Area Report
Source of Area : Timing Library
-------------------
+-------------------------------------------------------------+
|     Module |   Wireload |  Cell Area |   Net Area | Total Area |
```

```
|------------+------------+------------+------------+------------|
|   miniUART |       NONE |  129384.00 |       0.00 |  129384.00 |
|    ClkUnit |       NONE |   25920.00 |       0.00 |   25920.00 |
|     RxUnit |       NONE |   44064.00 |       0.00 |   44064.00 |
|     TxUnit |       NONE |   36288.00 |       0.00 |   36288.00 |
| AWDP_DEC_1 |       NONE |    4622.40 |       0.00 |    4622.40 |
+-----------------------------------------------------------------+

+-------------------------------------------------------------+
|  Block report for module 'miniUART'  |  Current | Cumulative |
|                                      |  Module  |            |
|--------------------------------------+----------+------------|
| Number of combinational instances    |       51 |        324 |
| Number of noncombinational instances |       18 |         85 |
| Number of hierarchical instances     |        3 |          4 |
| Number of blackbox instances         |        0 |          0 |
| Total number of instances            |       72 |        413 |
| Area of combinational cells          |  8337.60 |   59616.00 |
| Area of non-combinational cells      | 14774.40 |   69768.00 |
| Total cell area                      | 23112.00 |  129384.00 |
| Number of nets                       |      101 |        446 |
| Area of nets                         |     0.00 |       0.00 |
| Total area                           | 23112.00 |  129384.00 |
+-------------------------------------------------------------+

+-------------------------------------------------------------------------------+
|                              Cell Usage Table                                 |
|-------------------------------------------------------------------------------|
|    Cellref |     Library |  Number of |  Cell Type |  Cell Area | Total Area |
|            |             |  Instances |            |            |            |
|------------+-------------+------------+------------+------------+------------|
|    ClkUnit |     netlist |          1 |       hier |   25920.00 |   25920.00 |
|      NAND4 | UT_LP_AMI06 |          1 |       comb |     216.00 |     216.00 |
|     RxUnit |     netlist |          1 |       hier |   44064.00 |   44064.00 |
|      TIELO | UT_LP_AMI06 |          1 |       comb |      86.40 |      86.40 |
|     TxUnit |     netlist |          1 |       hier |   36288.00 |   36288.00 |
|       NOR3 | UT_LP_AMI06 |          3 |       comb |     172.80 |     518.40 |
|      NAND3 | UT_LP_AMI06 |          6 |       comb |     172.80 |    1036.80 |
|       MUX2 | UT_LP_AMI06 |          8 |       comb |     345.60 |    2764.80 |
|       NOR2 | UT_LP_AMI06 |          9 |       comb |     129.60 |    1166.40 |
|        INV | UT_LP_AMI06 |         10 |       comb |      86.40 |     864.00 |
|      NAND2 | UT_LP_AMI06 |         13 |       comb |     129.60 |    1684.80 |
|      DFF_S | UT_LP_AMI06 |         18 |    noncomb |     820.80 |   14774.40 |
+-------------------------------------------------------------------------------+
```

K.6 IIT Bidirectional Bus

```
+-----------------------------------------+
| Report  | report_area                   |
|---------+-------------------------------|
| Options | -cells -summary iit_bus.summary |
+---------+-------------------------------+
| Date    | 20031016.202144               |
| Tool    | bgx_shell                     |
| Release | v5.11-s079                    |
| Version | Aug 19 2003 04:51:32          |
+---------+-------------------------------+
| Module  | bidir                         |
+-----------------------------------------+

Summary Area Report
Source of Area : Timing Library
```

```
------------------
+-------------------------------------------------------------+
|     Module |   Wireload |  Cell Area |   Net Area | Total Area |
|------------+------------+------------+------------+------------|
|      bidir |       NONE |   17856.00 |       0.00 |   17856.00 |
+-------------------------------------------------------------+

+-------------------------------------------------------------+
|   Block report for module 'bidir'    |  Current | Cumulative |
|                                      |  Module  |            |
|--------------------------------------+----------+------------|
| Number of combinational instances    |        8 |          8 |
| Number of noncombinational instances |       24 |         24 |
| Number of hierarchical instances     |        0 |          0 |
| Number of blackbox instances         |        0 |          0 |
| Total number of instances            |       32 |         32 |
| Area of combinational cells          |  1152.00 |    1152.00 |
| Area of non-combinational cells      | 16704.00 |   16704.00 |
| Total cell area                      | 17856.00 |   17856.00 |
| Number of nets                       |       42 |         42 |
| Area of nets                         |     0.00 |       0.00 |
| Total area                           | 17856.00 |   17856.00 |
+-------------------------------------------------------------+

+-------------------------------------------------------------------------------------------+
|                                    Cell Usage Table                                       |
|-------------------------------------------------------------------------------------------|
|    Cellref |             Library |  Number of |  Cell Type |  Cell Area | Total Area |
|            |                     |  Instances |            |            |            |
|------------+---------------------+------------+------------+------------+------------|
|      INVX1 | iit06_stdcells_pads |          8 |       comb |     144.00 |    1152.00 |
|     TBUFX1 | iit06_stdcells_pads |          8 |    noncomb |     360.00 |    2880.00 |
|   DFFPOSX1 | iit06_stdcells_pads |         16 |    noncomb |     864.00 |   13824.00 |
+-------------------------------------------------------------------------------------------+
```

K.7 IIT miniUART

```
+-----------------------------------------------+
| Report  | report_area                         |
|---------+-------------------------------------|
| Options | -cells -summary iit_miniUART.summary |
+---------+-------------------------------------+
| Date    | 20031016.202158                     |
| Tool    | bgx_shell                           |
| Release | v5.11-s079                          |
| Version | Aug 19 2003 04:51:32                |
+---------+-------------------------------------+
| Module  | miniUART                            |
+-----------------------------------------------+

Summary Area Report
Source of Area : Timing Library
------------------
+-------------------------------------------------------------+
|     Module |   Wireload |  Cell Area |   Net Area | Total Area |
|------------+------------+------------+------------+------------|
|   miniUART |       NONE |  159993.00 |       0.00 |  159993.00 |
|    ClkUnit |       NONE |   32868.00 |       0.00 |   32868.00 |
|     RxUnit |       NONE |   54567.00 |       0.00 |   54567.00 |
|     TxUnit |       NONE |   44055.00 |       0.00 |   44055.00 |
| AWDP_DEC_1 |       NONE |    5976.00 |       0.00 |    5976.00 |
+-------------------------------------------------------------+
```

Block report for module 'miniUART'	Current Module	Cumulative
Number of combinational instances	60	389
Number of noncombinational instances	18	85
Number of hierarchical instances	3	4
Number of blackbox instances	0	0
Total number of instances	81	478
Area of combinational cells	12951.00	86553.00
Area of non-combinational cells	15552.00	73440.00
Total cell area	28503.00	159993.00
Number of nets	110	491
Area of nets	0.00	0.00
Total area	28503.00	159993.00

Cell Usage Table

Cellref	Library	Number of Instances	Cell Type	Cell Area	Total Area
AOI21X1	iit06_stdcells_pads	1	comb	288.00	288.00
ClkUnit	netlist	1	hier	32868.00	32868.00
RxUnit	netlist	1	hier	54567.00	54567.00
TxUnit	netlist	1	hier	44055.00	44055.00
OR2X1	iit06_stdcells_pads	2	comb	288.00	576.00
AND2X1	iit06_stdcells_pads	6	comb	288.00	1728.00
NAND3X1	iit06_stdcells_pads	6	comb	324.00	1944.00
DFFPOSX1	iit06_stdcells_pads	7	noncomb	864.00	6048.00
DFFNEGX1	iit06_stdcells_pads	11	noncomb	864.00	9504.00
NAND2X1	iit06_stdcells_pads	12	comb	216.00	2592.00
INVX1	iit06_stdcells_pads	16	comb	144.00	2304.00
OAI21X1	iit06_stdcells_pads	17	comb	207.00	3519.00

K.8 MSU AVR Microprocessor

Report	report_area
Options	-cells -summary msu_avr.summary
Date	20031016.202500
Tool	bgx_shell
Release	v5.11-s079
Version	Aug 19 2003 04:51:32
Module	avr_core

Summary Area Report
Source of Area : Timing Library

Module	Wireload	Cell Area	Net Area	Total Area
avr_core	NONE	1343822.42	0.00	1343822.42
alu_avr	NONE	57456.00	0.00	57456.00
bit_processor	NONE	33998.40	0.00	33998.40

```
|             io_adr_dec |       NONE |   13003.20 |       0.00 |   13003.20 |
|            io_reg_file |       NONE |   94478.40 |       0.00 |   94478.40 |
|           pm_fetch_dec |       NONE |  512481.61 |       0.00 |  512481.61 |
| reg_file_ResetRegFile1 |       NONE |  632404.81 |       0.00 |  632404.81 |
|            AWDP_DEC_00 |       NONE |   14342.40 |       0.00 |   14342.40 |
|            AWDP_INC_11 |       NONE |   11620.80 |       0.00 |   11620.80 |
|       AWRS_partition_2 |       NONE |   33350.40 |       0.00 |   33350.40 |
|       AWRS_partition_3 |       NONE |   24883.20 |       0.00 |   24883.20 |
|       AWDP_partition_2 |       NONE |   26870.40 |       0.00 |   26870.40 |
|       AWDP_partition_3 |       NONE |   17280.00 |       0.00 |   17280.00 |
|             AWDP_DEC_0 |       NONE |   14342.40 |       0.00 |   14342.40 |
|            AWDP_INC_10 |       NONE |   11620.80 |       0.00 |   11620.80 |
+------------------------------------------------------------------------------+

+-------------------------------------------------------------+
|  Block report for module 'avr_core'  | Current | Cumulative |
|                                      |  Module |            |
|--------------------------------------+---------+------------|
| Number of combinational instances    |       0 |       6315 |
| Number of noncombinational instances |       0 |        468 |
| Number of hierarchical instances     |       6 |         14 |
| Number of blackbox instances         |       0 |          0 |
| Total number of instances            |       6 |       6797 |
| Area of combinational cells          |    0.00 |  777729.63 |
| Area of non-combinational cells      |    0.00 |  566092.79 |
| Total cell area                      |    0.00 | 1343822.42 |
| Number of nets                       |     336 |       6951 |
| Area of nets                         |    0.00 |       0.00 |
| Total area                           |    0.00 | 1343822.42 |
+-------------------------------------------------------------+

+-------------------------------------------------------------------------------------------+
|                                    Cell Usage Table                                       |
|-------------------------------------------------------------------------------------------|
|                Cellref |    Library |  Number of |  Cell Type |  Cell Area | Total Area |
|                        |            |  Instances |            |            |            |
|------------------------+------------+------------+------------+------------+------------|
|                alu_avr |    netlist |          1 |       hier |   57456.00 |   57456.00 |
|          bit_processor |    netlist |          1 |       hier |   33998.40 |   33998.40 |
|             io_adr_dec |    netlist |          1 |       hier |   13003.20 |   13003.20 |
|            io_reg_file |    netlist |          1 |       hier |   94478.40 |   94478.40 |
|           pm_fetch_dec |    netlist |          1 |       hier |  512481.61 |  512481.61 |
| reg_file_ResetRegFile1 |    netlist |          1 |       hier |  632404.81 |  632404.81 |
+-------------------------------------------------------------------------------------------+
```

K.9 MSU miniUART

```
+-----------------------------------------------+
| Report  | report_area                         |
|---------+-------------------------------------|
| Options | -cells -summary msu_miniUART.summary |
+---------+-------------------------------------+
| Date    | 20031016.202220                     |
| Tool    | bgx_shell                           |
| Release | v5.11-s079                          |
| Version | Aug 19 2003 04:51:32                |
+---------+-------------------------------------+
| Module  | miniUART                            |
+-----------------------------------------------+

Summary Area Report
Source of Area : Timing Library
```

```
------------------
+-----------------------------------------------------------------+
|     Module |   Wireload |  Cell Area |   Net Area | Total Area |
|------------+------------+------------+------------+------------|
|   miniUART |       NONE |  175478.40 |       0.00 |  175478.40 |
|    ClkUnit |       NONE |   35078.40 |       0.00 |   35078.40 |
|     RxUnit |       NONE |   59961.60 |       0.00 |   59961.60 |
|     TxUnit |       NONE |   48513.60 |       0.00 |   48513.60 |
| AWDP_DEC_1 |       NONE |    6652.80 |       0.00 |    6652.80 |
+-----------------------------------------------------------------+

+-----------------------------------------------------------+
|  Block report for module 'miniUART'  |  Current | Cumulative |
|                                      |  Module  |            |
|--------------------------------------+----------+------------|
| Number of combinational instances    |       85 |        604 |
| Number of noncombinational instances |       18 |         85 |
| Number of hierarchical instances     |        3 |          4 |
| Number of blackbox instances         |        0 |          0 |
| Total number of instances            |      106 |        693 |
| Area of combinational cells          | 10152.00 |   72662.40 |
| Area of non-combinational cells      | 21772.80 |  102816.00 |
| Total cell area                      | 31924.80 |  175478.40 |
| Number of nets                       |      131 |        702 |
| Area of nets                         |     0.00 |       0.00 |
| Total area                           | 31924.80 |  175478.40 |
+-----------------------------------------------------------+

+-------------------------------------------------------------------------------------+
|                                 Cell Usage Table                                    |
|-------------------------------------------------------------------------------------|
|    Cellref |            Library |  Number of |  Cell Type |  Cell Area | Total Area |
|            |                    |  Instances |            |            |            |
|------------+--------------------+------------+------------+------------+------------|
|    ClkUnit |            netlist |          1 |       hier |   35078.40 |   35078.40 |
|     RxUnit |            netlist |          1 |       hier |   59961.60 |   59961.60 |
|      TIEHI | jennings_pads_noqn |          1 |       comb |      86.40 |      86.40 |
|      TIELO | jennings_pads_noqn |          1 |       comb |      86.40 |      86.40 |
|     TxUnit |            netlist |          1 |       hier |   48513.60 |   48513.60 |
|    DFFSRX1 | jennings_pads_noqn |         18 |    noncomb |    1209.60 |   21772.80 |
|      INVX1 | jennings_pads_noqn |         18 |       comb |      86.40 |    1555.20 |
|     NOR2X1 | jennings_pads_noqn |         18 |       comb |     129.60 |    2332.80 |
|    NAND2X1 | jennings_pads_noqn |         47 |       comb |     129.60 |    6091.20 |
+-------------------------------------------------------------------------------------+
```

APPENDIX L

OSU Standard Cell Library Contents

Name	Cell Type	Function
ao22x1	And-Or	Y=(A*B)+(C*D)
aoi22x1	And-Or-Inv	Y=((A*B)+(C*D))'
bufx1	Buffer (1x drive strength)	Y=A
bufx4	Buffer (4x drive strength)	Y=A
bufzx1	Tri-State Buffer	Y=A*TriState:'Z'*Tristate'
dff	D Flip-Flop	D=Data;CLK=Clock; QP=Output; QN=Output'
dffpc	D Flip-Flop	D=Data;CLK=Clock;PRE=Preset;CLR= Clear;P=Output;QN=Output'
fill1	Single-Width Fill Cell	(none)
fill2	Double-Width Fill Cell	(none)
invx1	Inverter (1x drive strength)	Y=A'
invx4	Inverter (4x drive strength)	Y=A'
invzx1	Tri-State Inverter	Y=A'*TriState:'Z'*Tristate'
lat	Latch	D=Data; EN=Enable; QP=Output; QN=Output'
latpc	Latch	D=Data;EN=Enable;PRE=Preset;CLR=Clear;QP=Output;QN=Output'
mux21x1	Multiplexor	Y=(A*Sel)+(B*Sel')
nand2x1	2-Input NAND	Y=(A*B)'
nand3x1	3-Input NAND	Y=(A*B*C)'
nand4x1	4-Input NAND	Y=(A*B*C*D)'
nor2x1	2-Input NAND	Y=(A+B)'
nor3x1	3-Input NAND	Y=(A+B+C)'
nor4x1	4-Input NAND	Y=(A+B+C+D)'
tiehigh	Tie High	Y='1'
tielow	Tie Low	Y='0'
xnor2x1	Exclusive NOR	Y=((A*B')+(A'*B))'
xor2x1	Exclusive OR	Y=(A*B')+(A'*B)

APPENDIX M

OSU Standard Cell Library

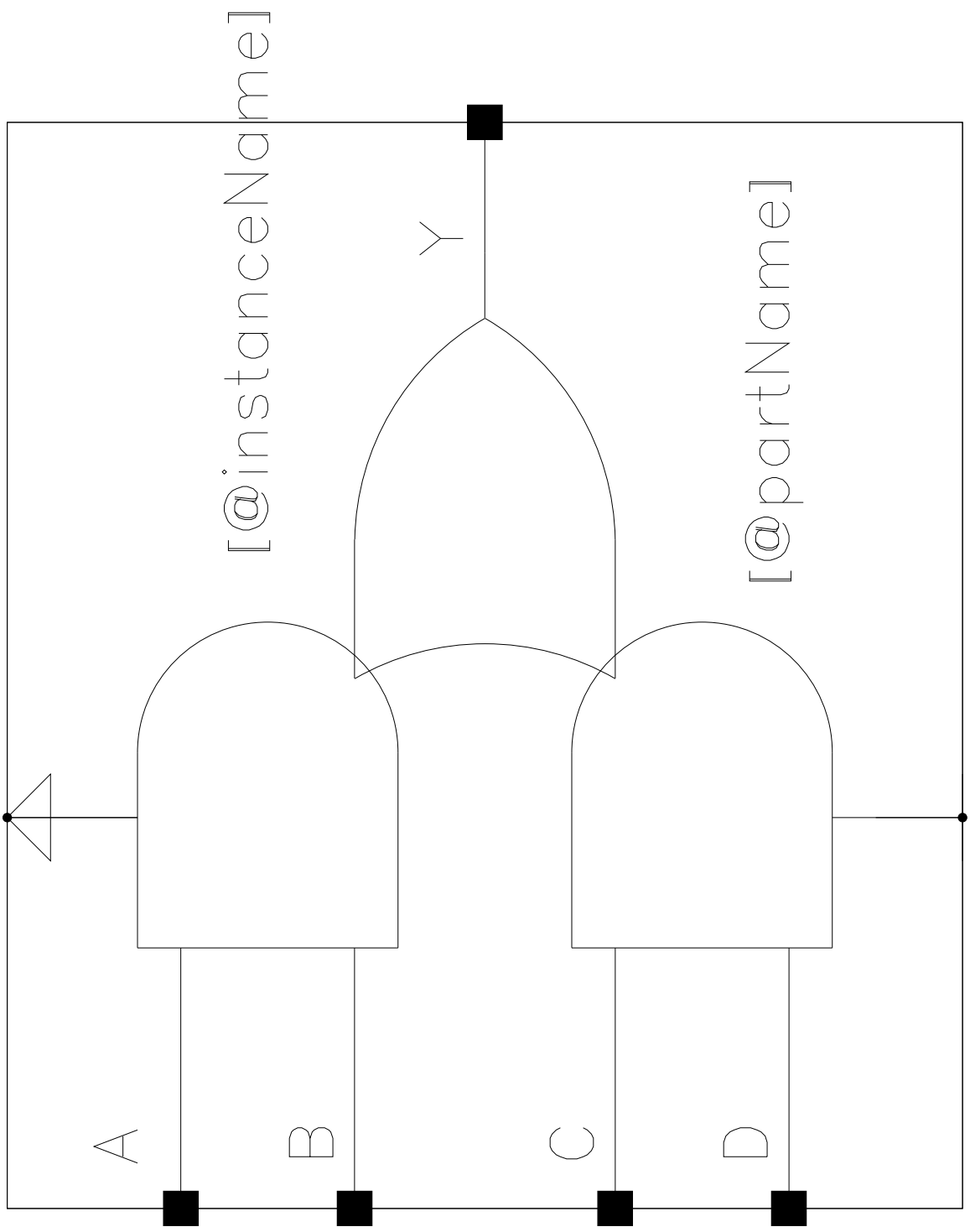

Figure M.1: Symbol of ao22x1

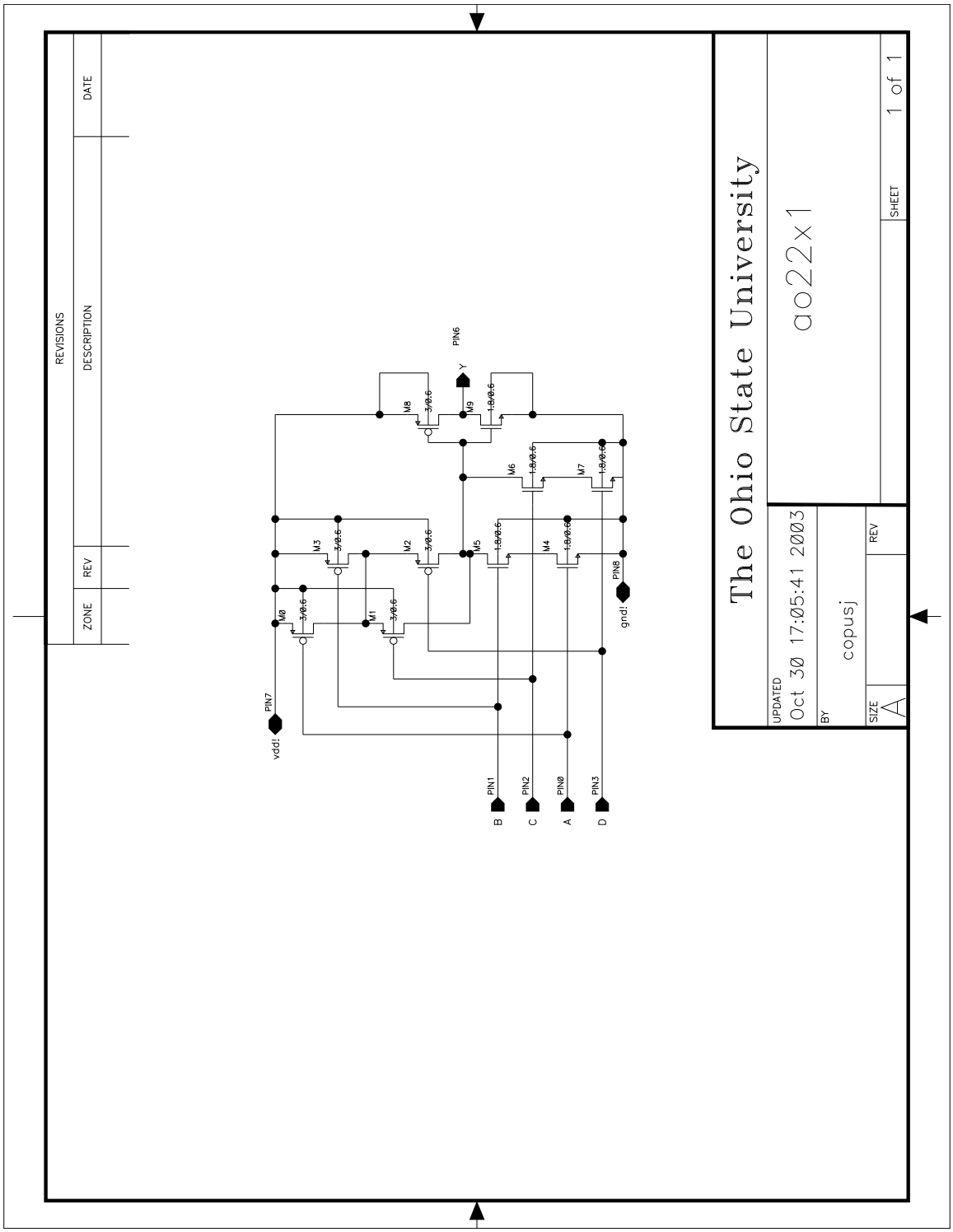

Figure M.2: Schematic of ao22x1

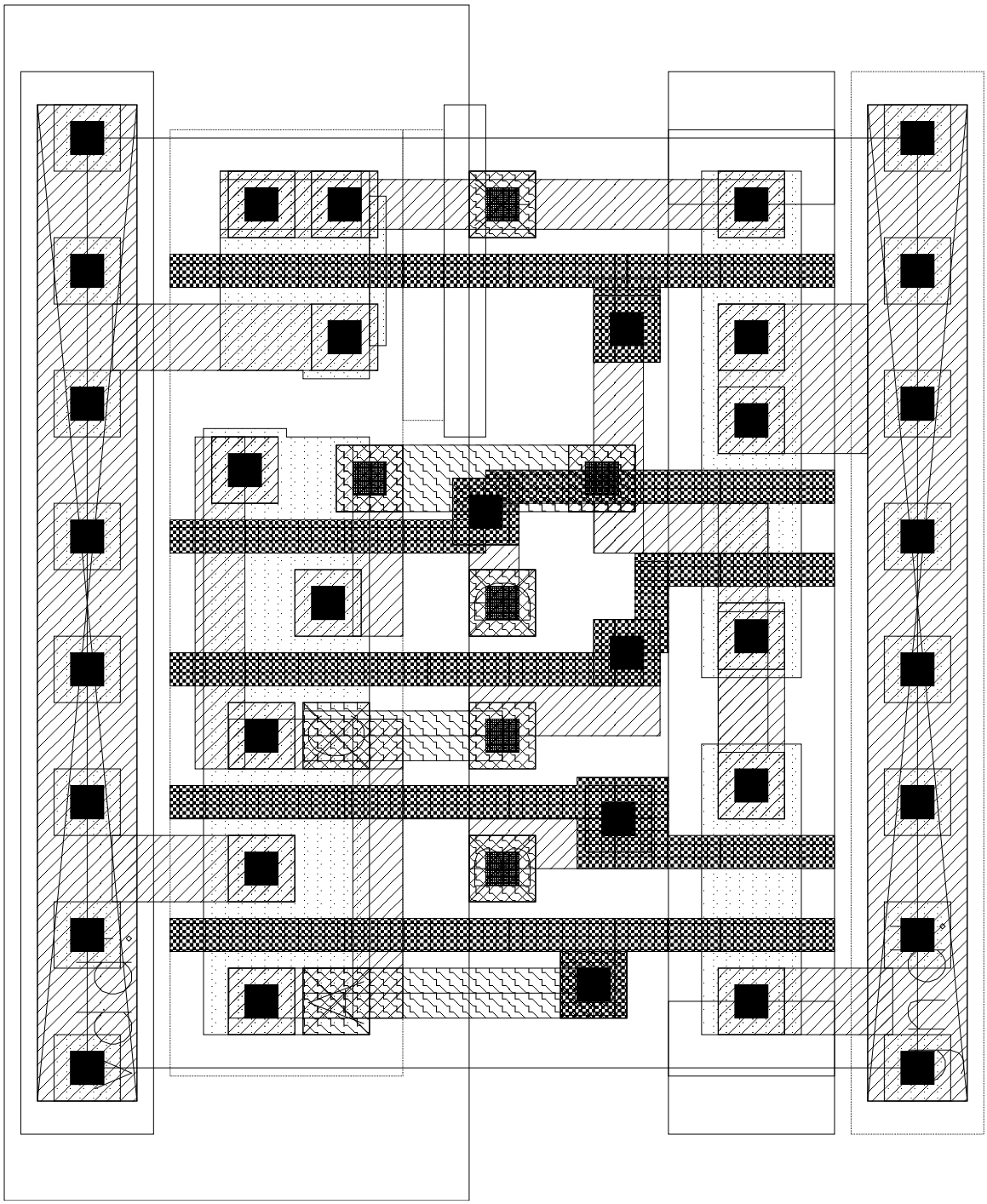

Figure M.3: Layout of ao22x1

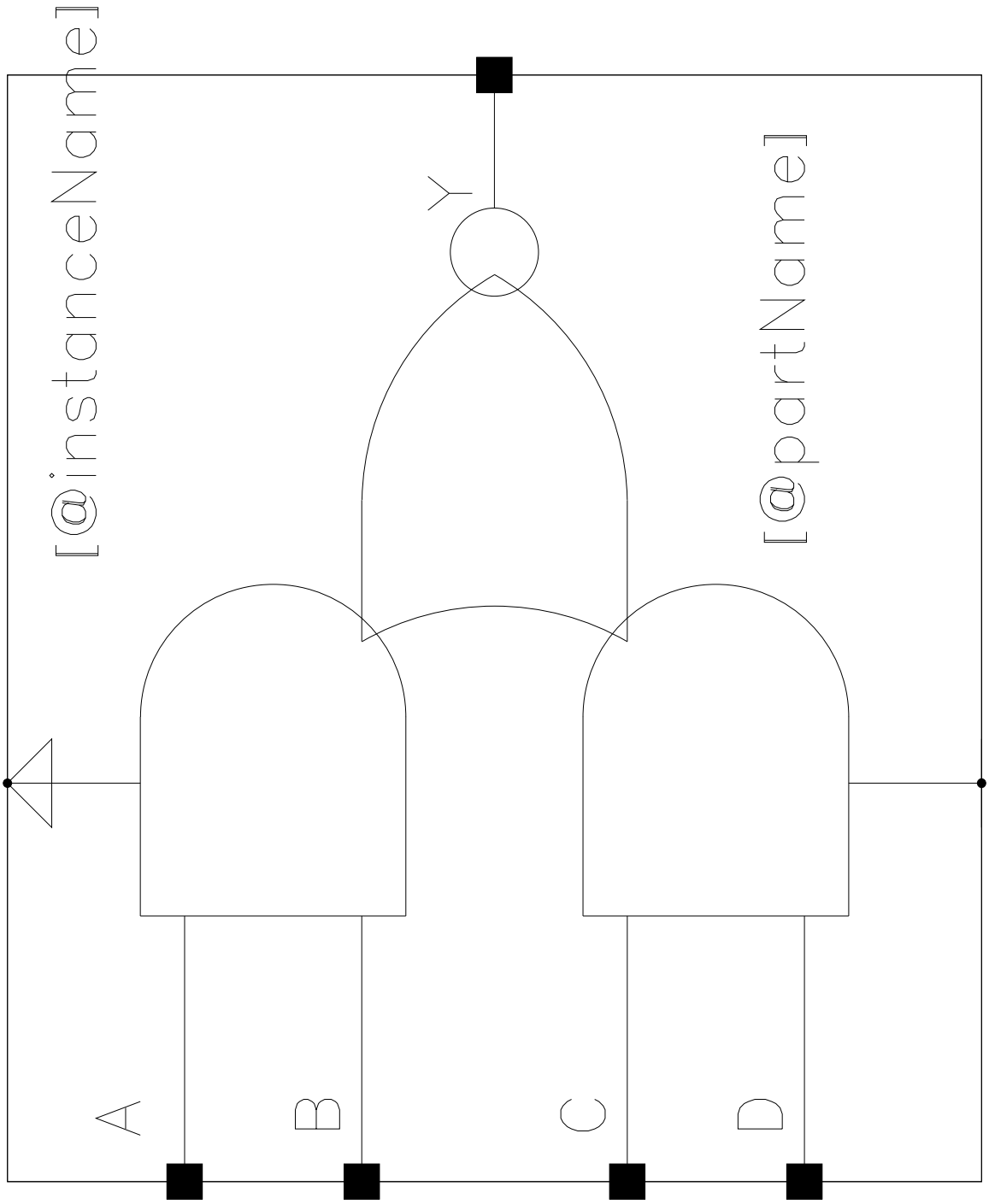

Figure M.4: Symbol of aoi22x1

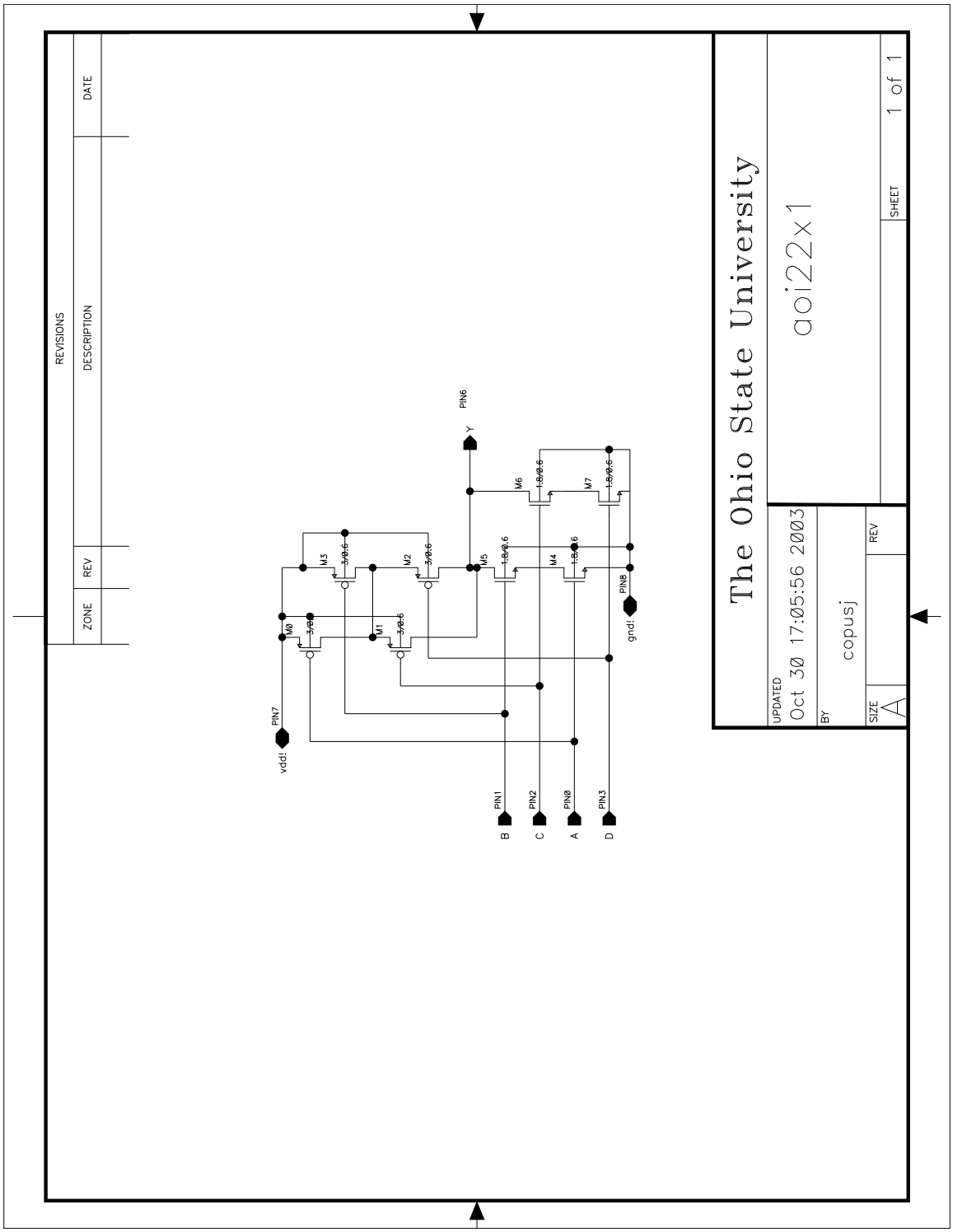

Figure M.5: Schematic of aoi22x1

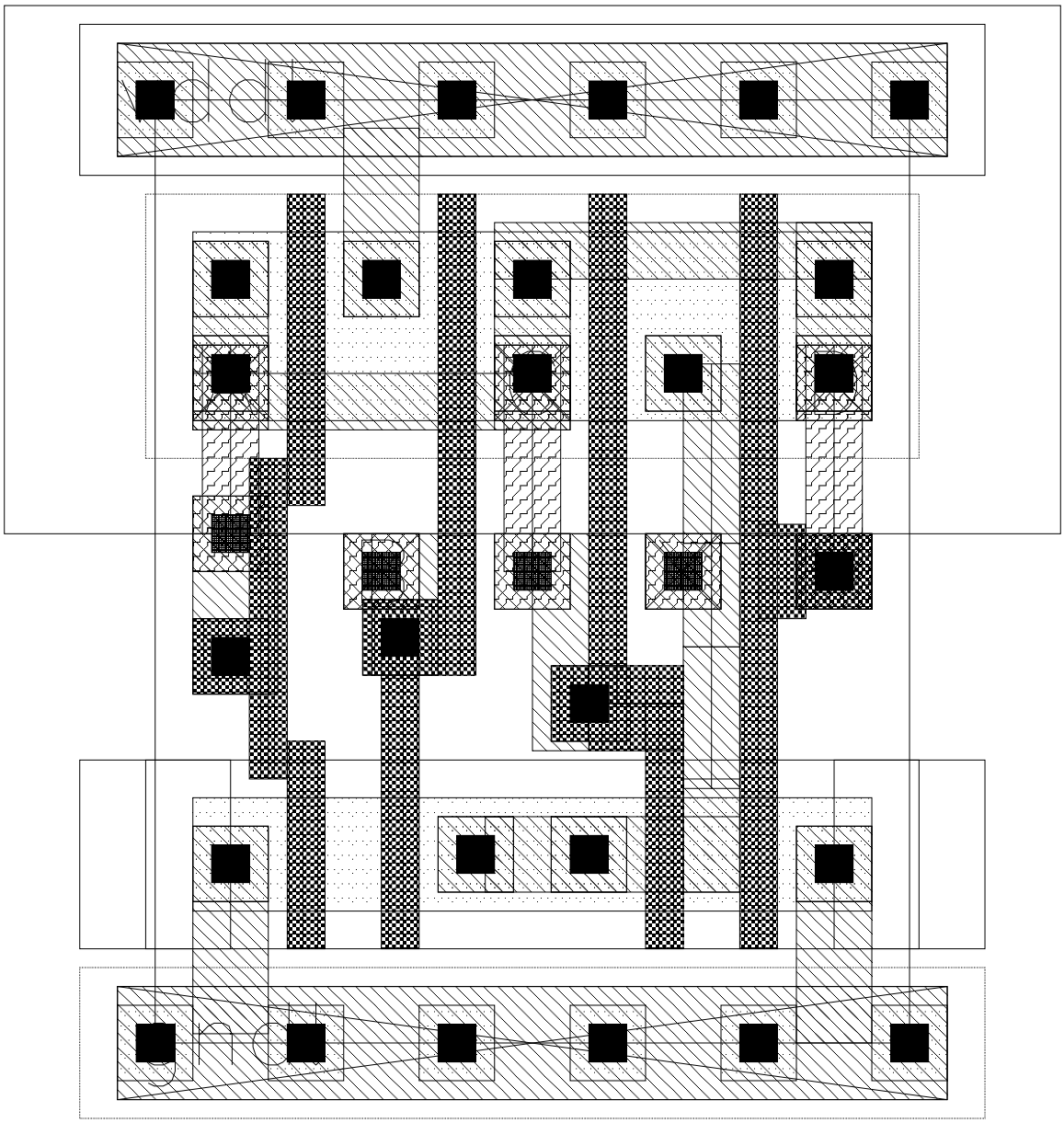

Figure M.6: Layout of aoi22x1

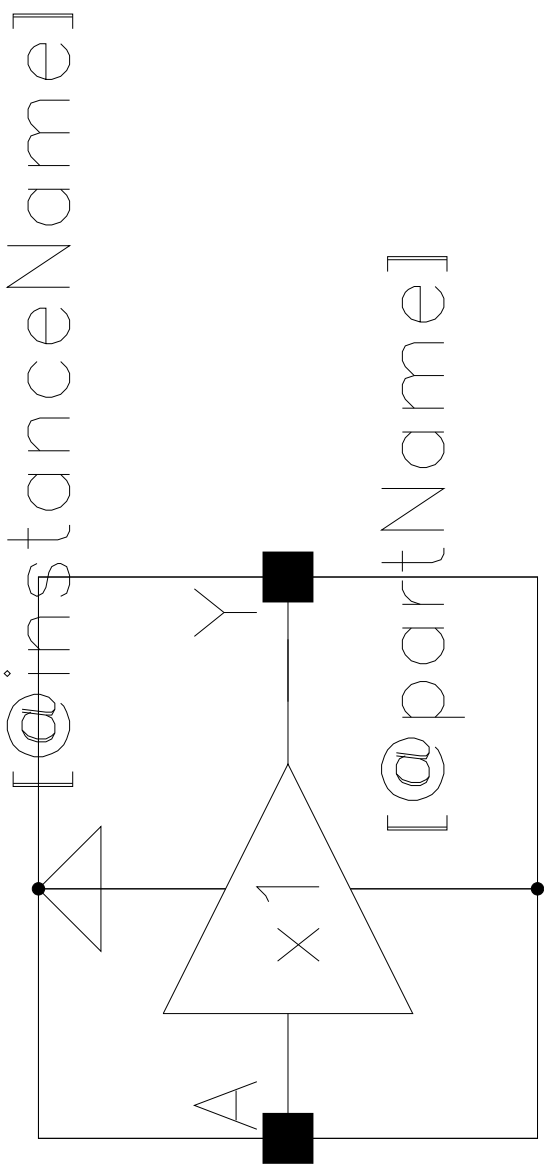

Figure M.7: Symbol of bufx1

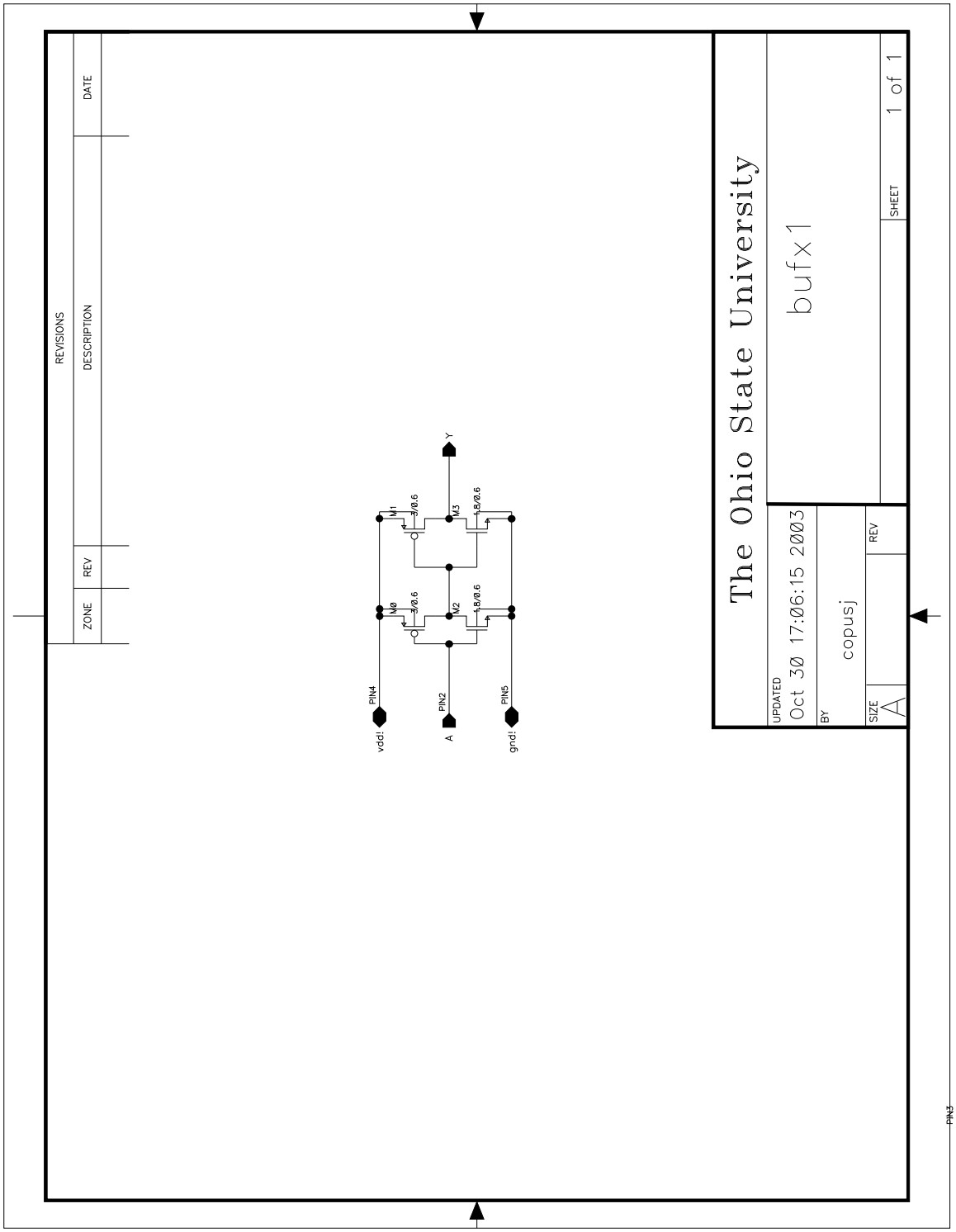

Figure M.8: Schematic of bufx1

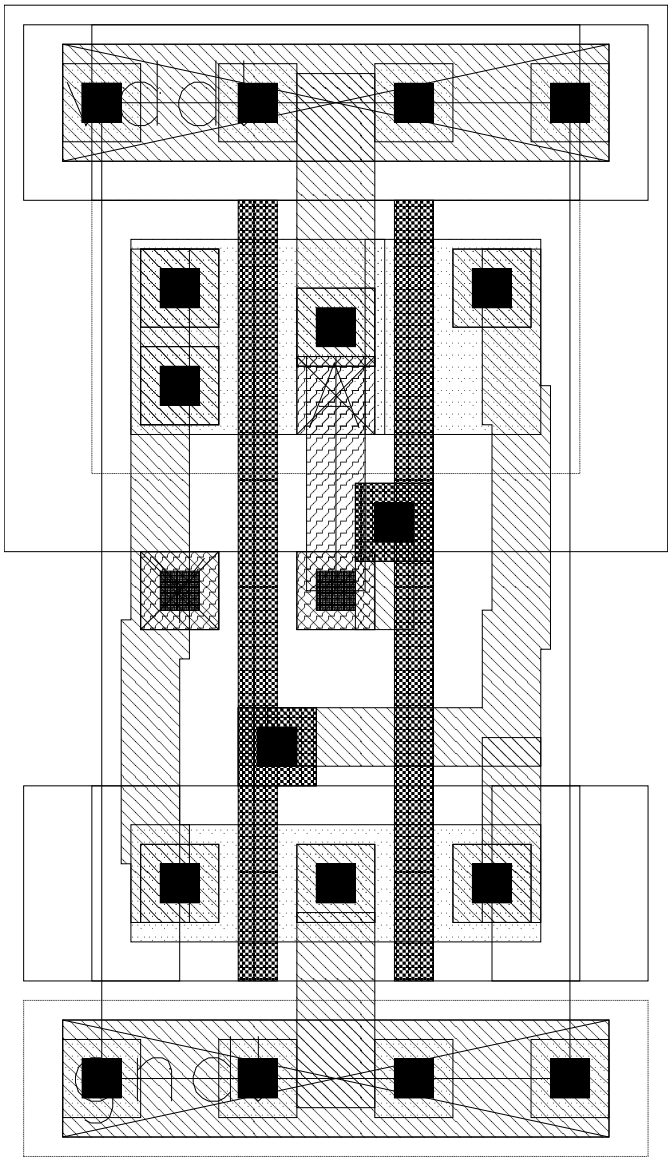

Figure M.9: Layout of bufx1

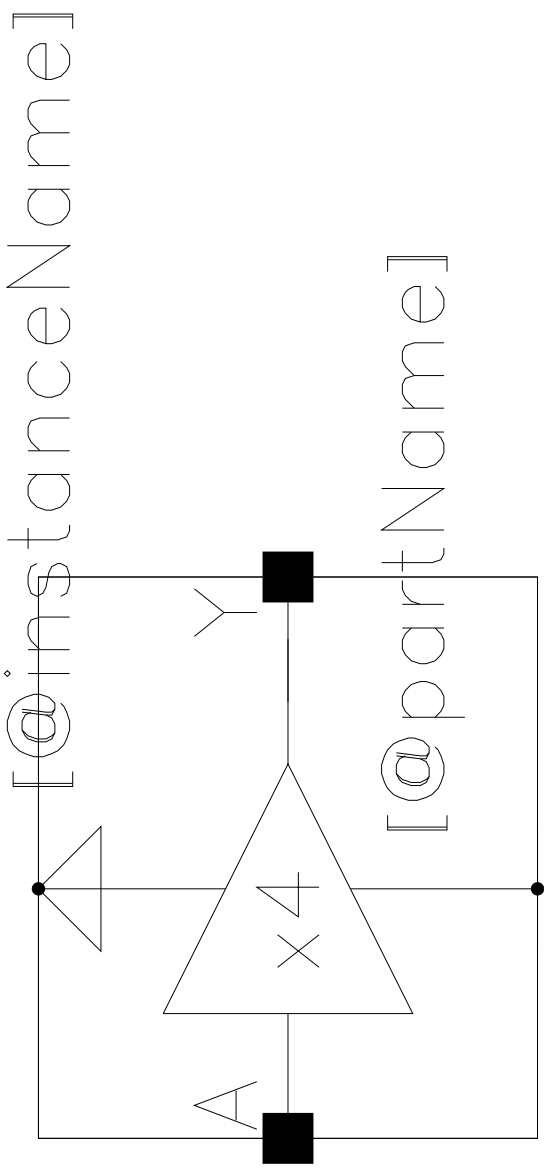

Figure M.10: Symbol of bufx4

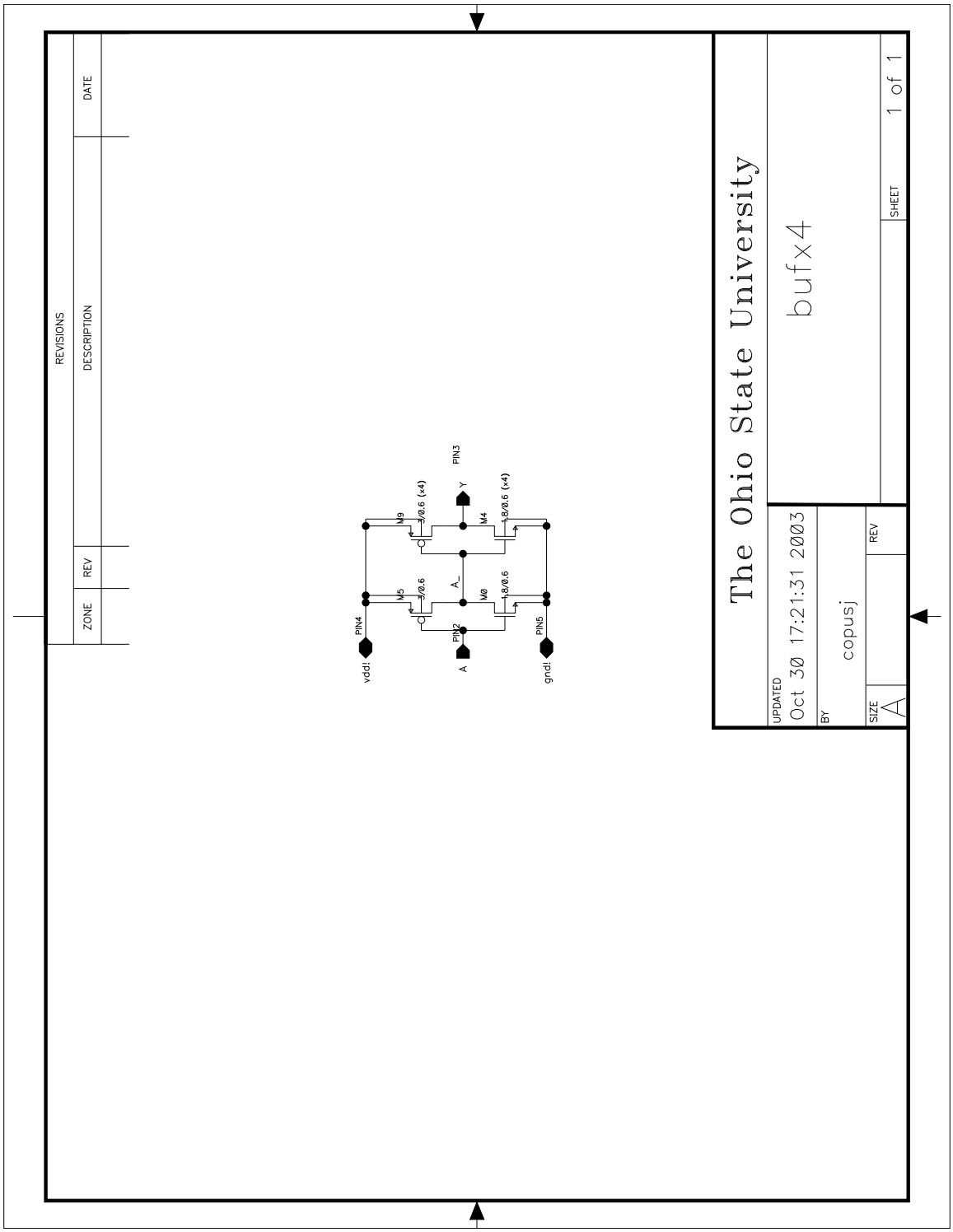

Figure M.11: Schematic of bufx4

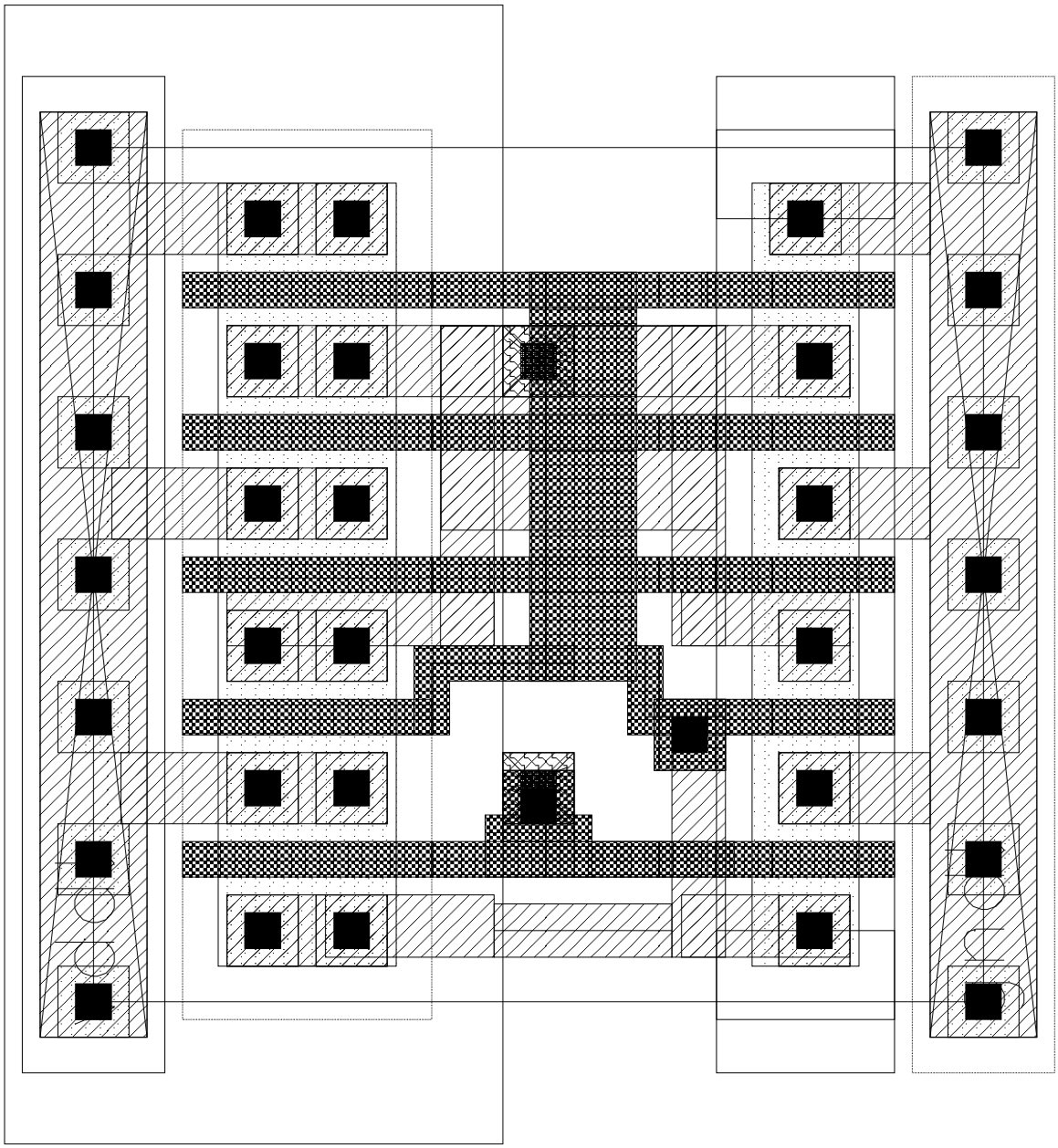

Figure M.12: Layout of bufx4

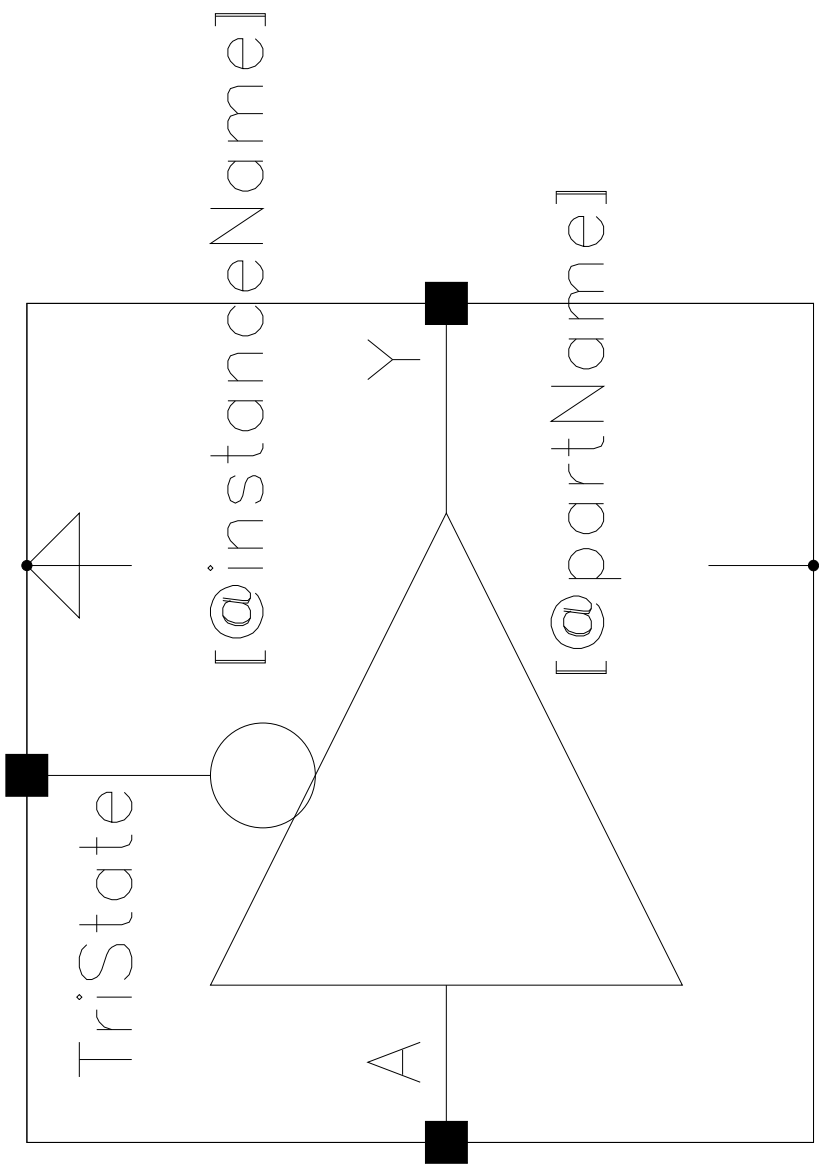

Figure M.13: Symbol of bufxz1

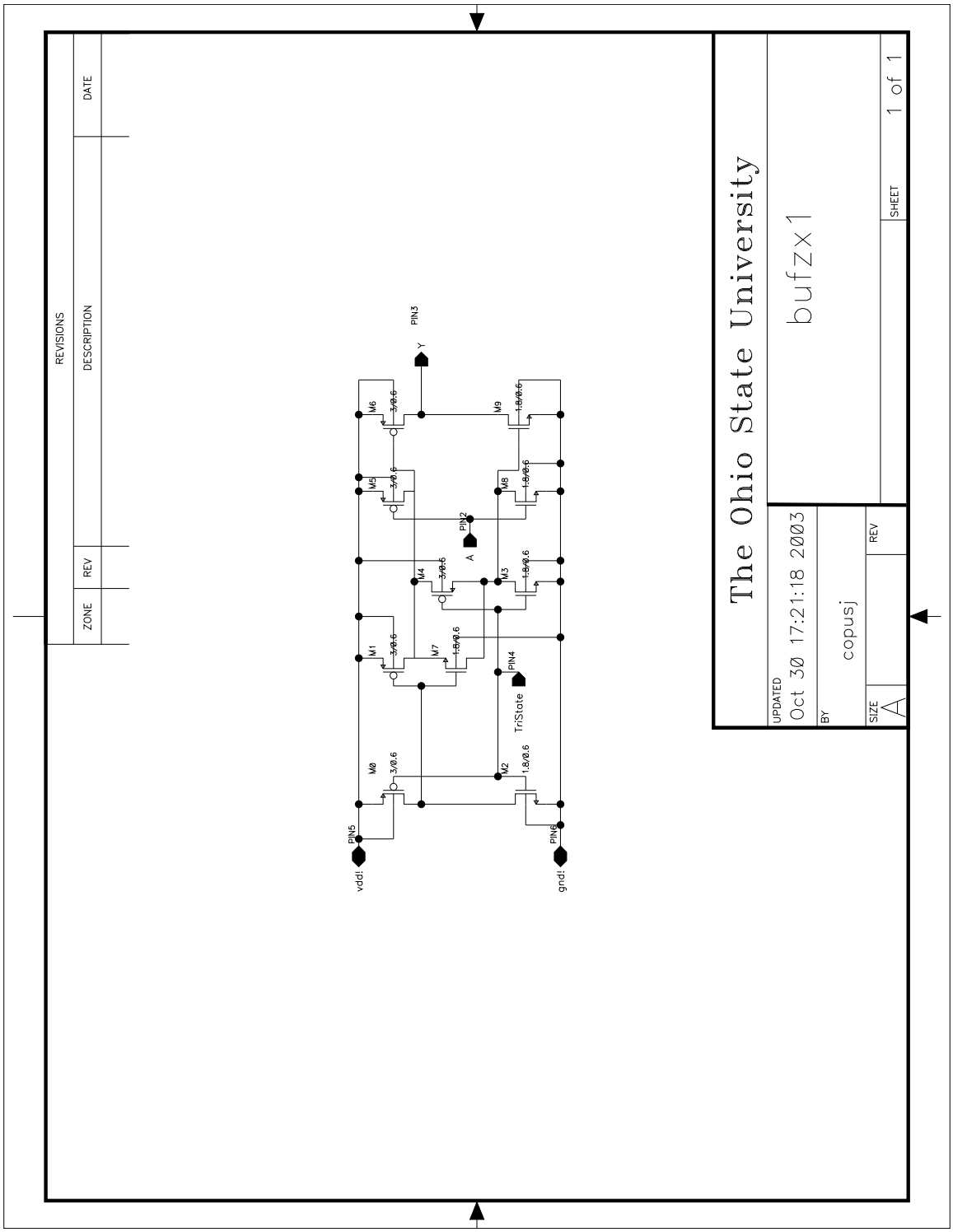

Figure M.14: Schematic of bufzx1

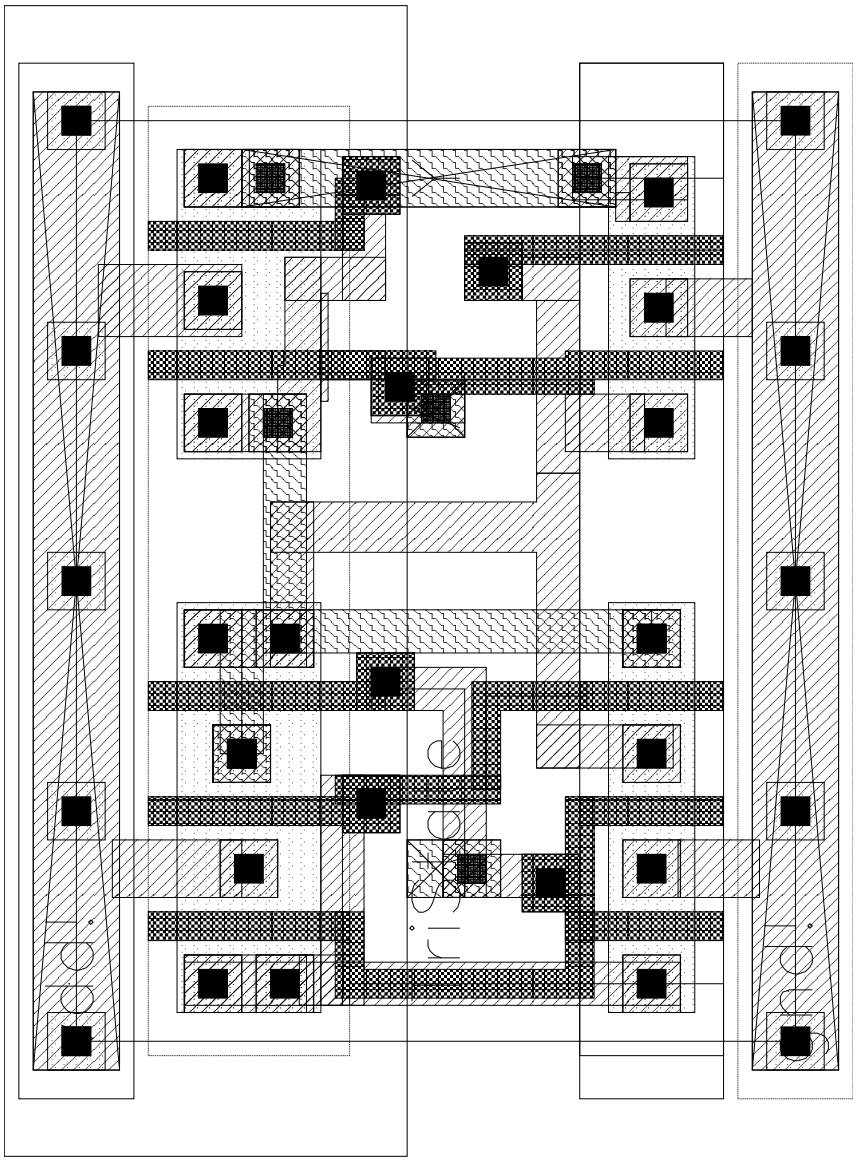

Figure M.15: Layout of bufzx1

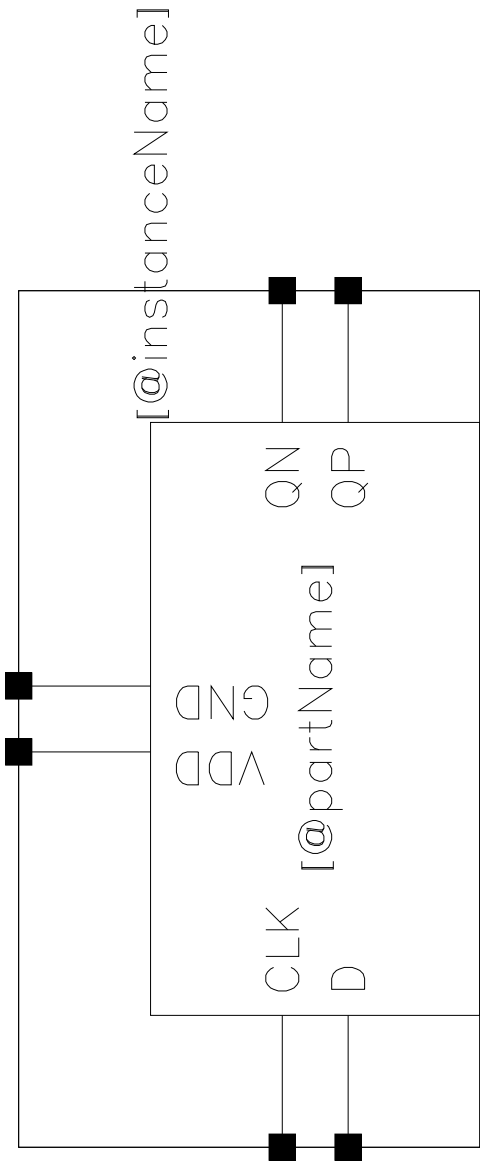

Figure M.16: Symbol of dff

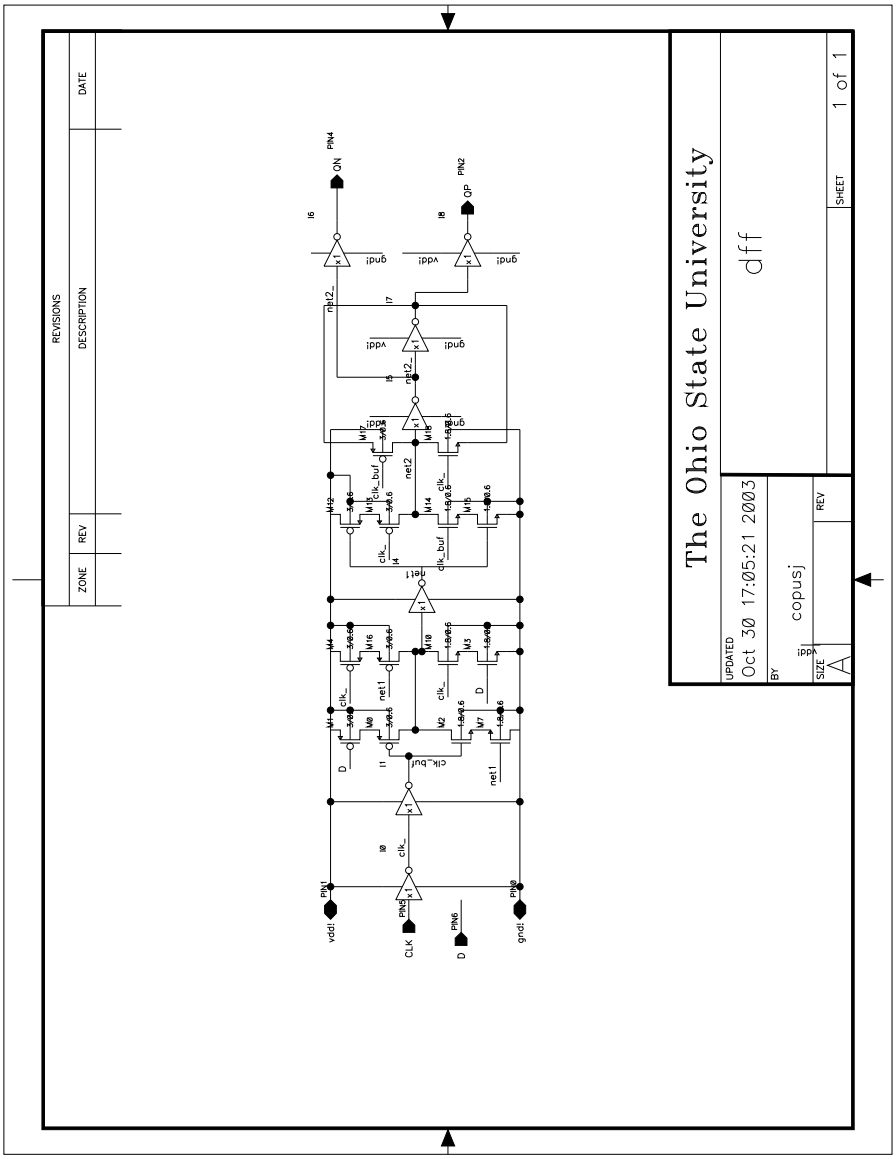

Figure M.17: Schematic of dff

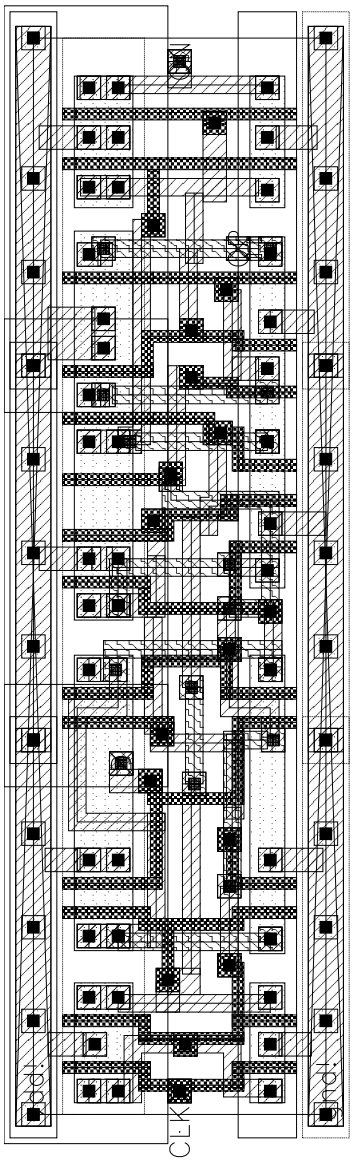

Figure M.18: Layout of dff

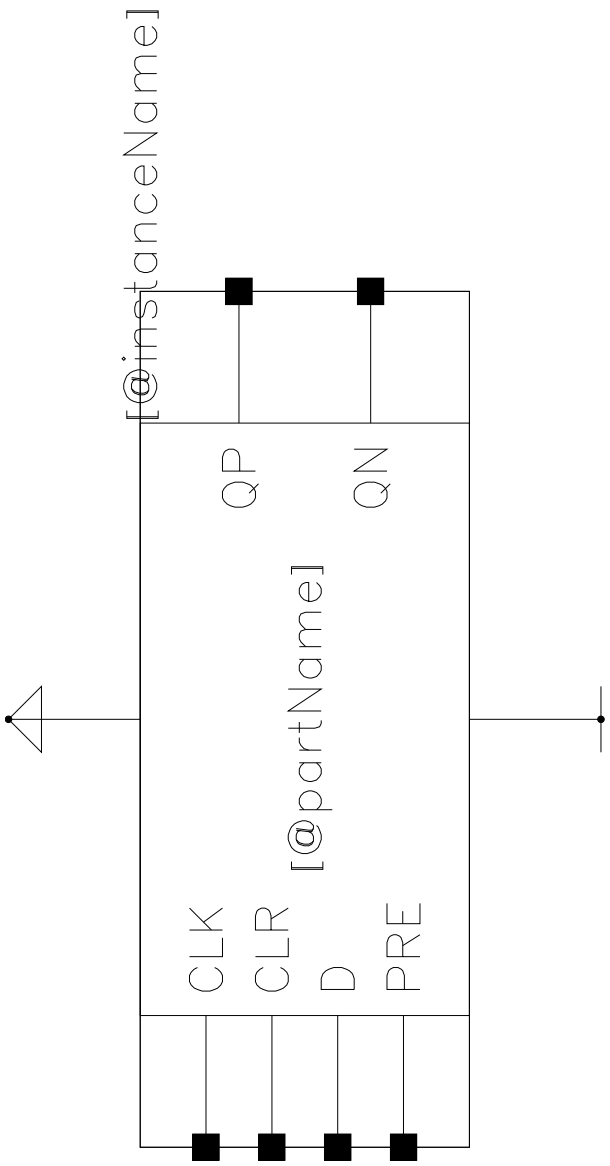

Figure M.19: Symbol of dffpc

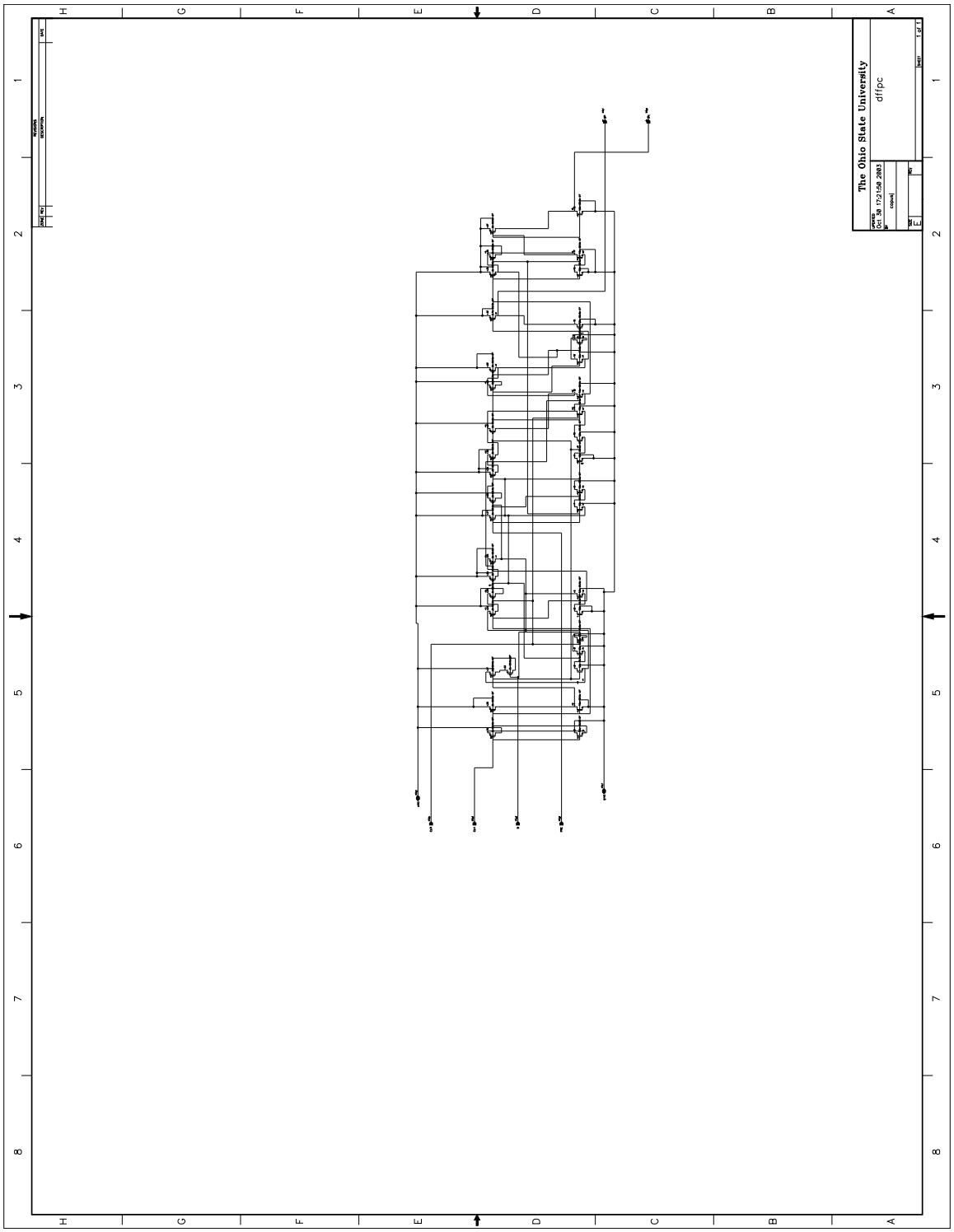

Figure M.20: Schematic of dffpc

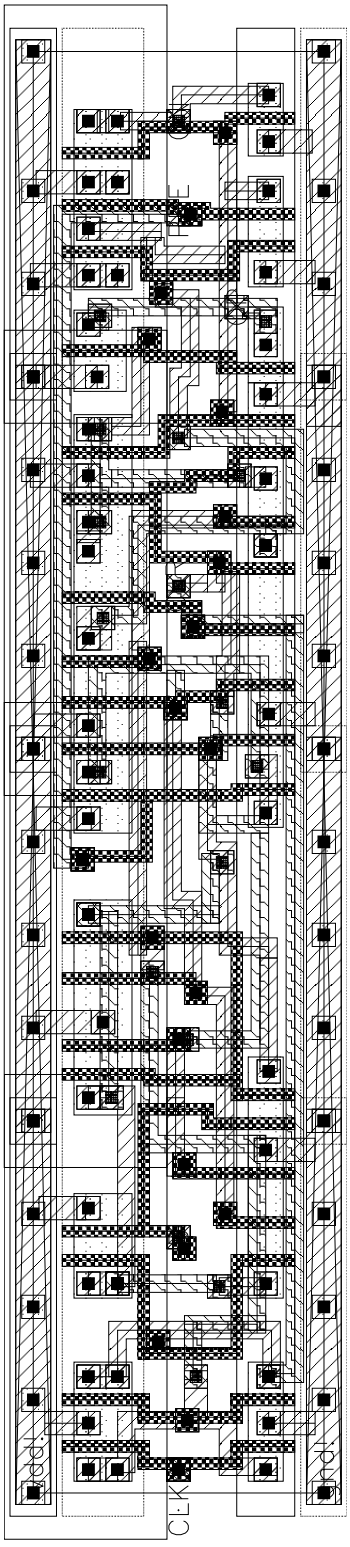

Figure M.21: Layout of dffpc

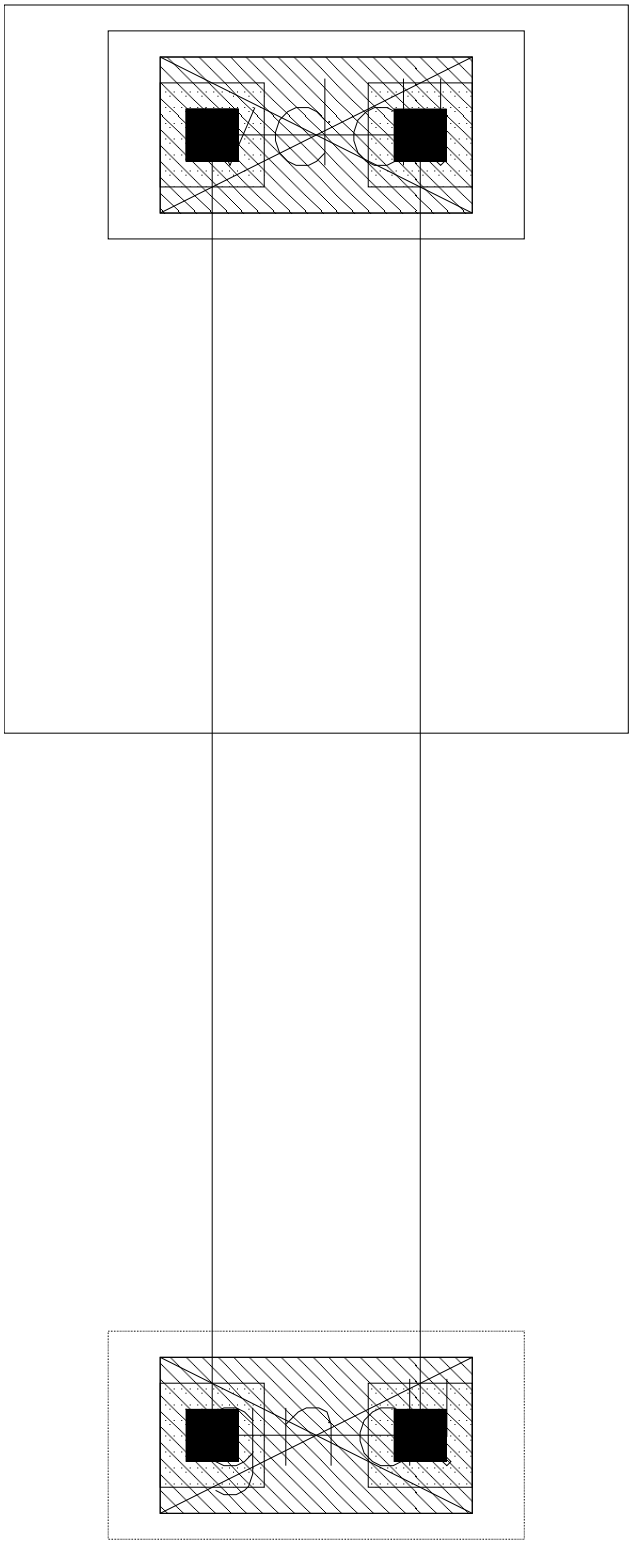

Figure M.22: Layout of fill1

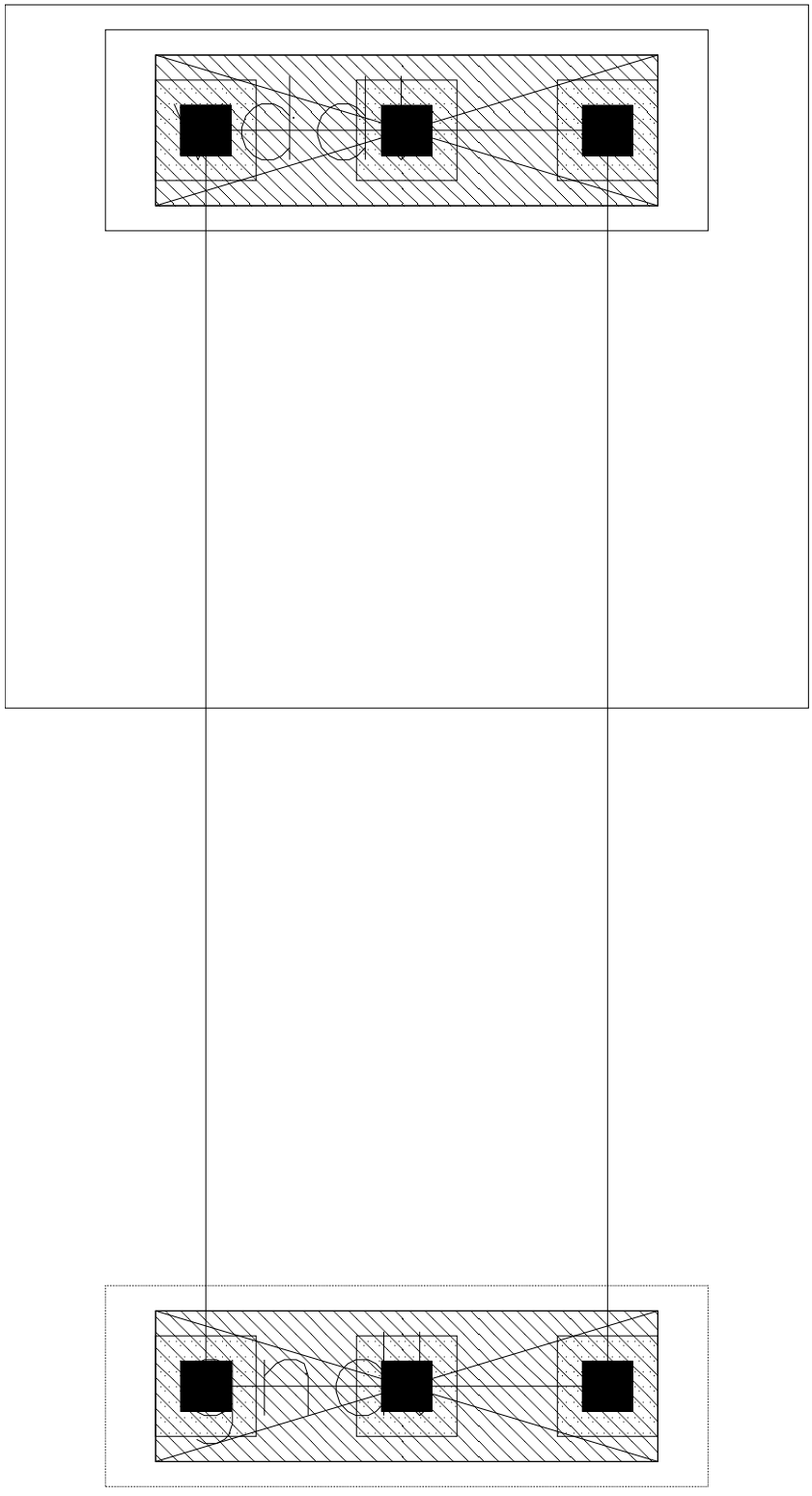

Figure M.23: Layout of fill2

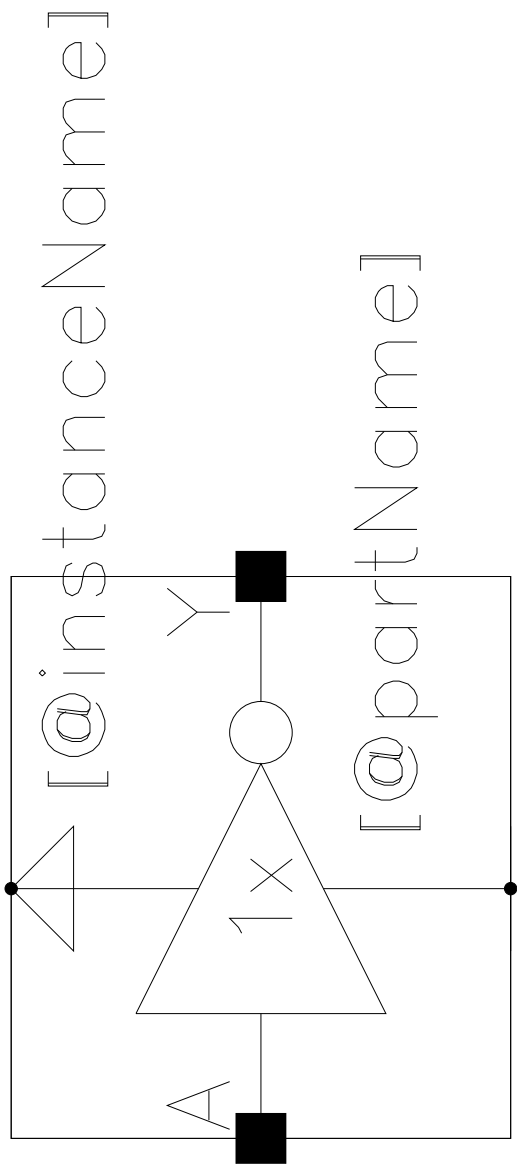

Figure M.24: Symbol of invx1

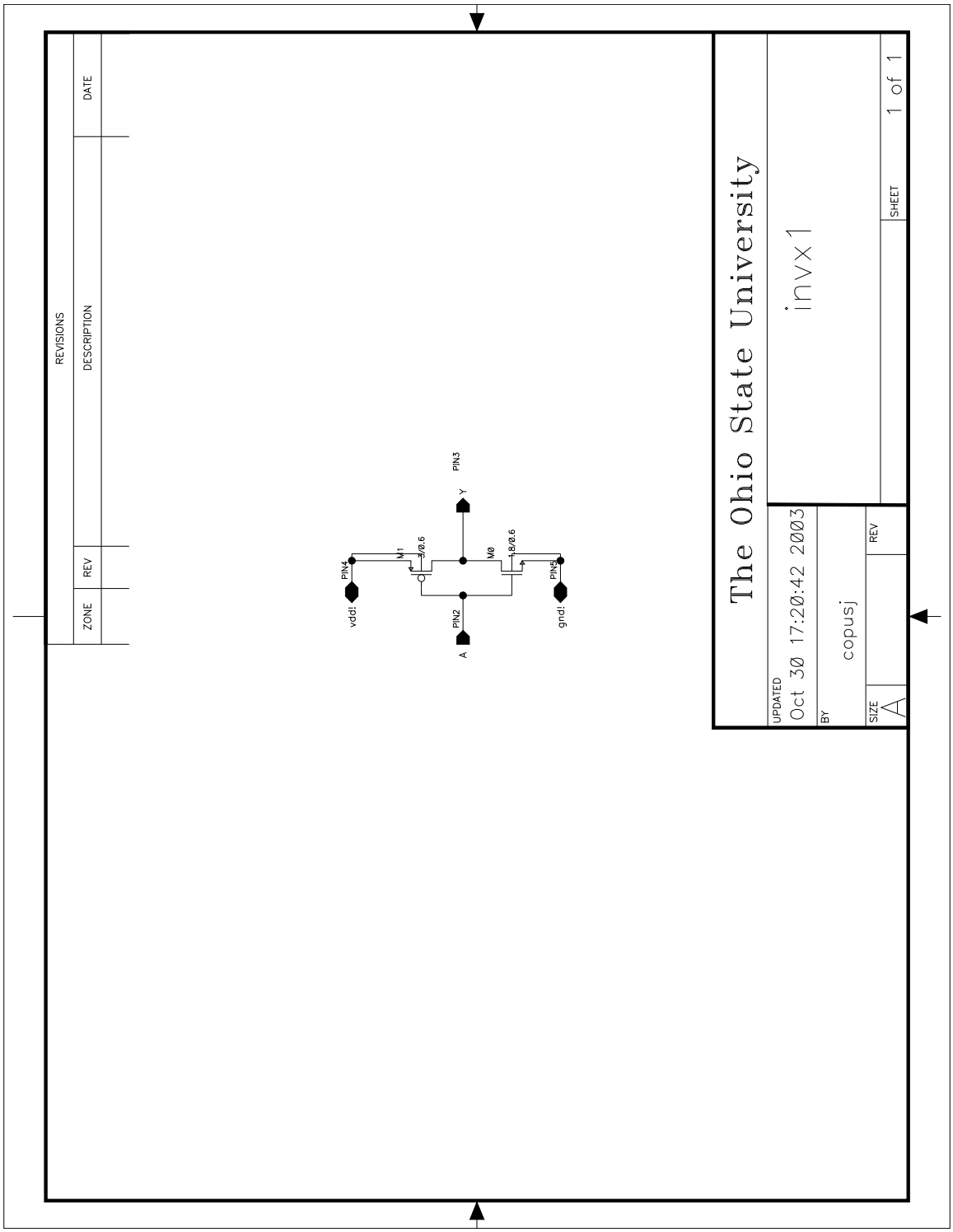

Figure M.25: Schematic of invx1

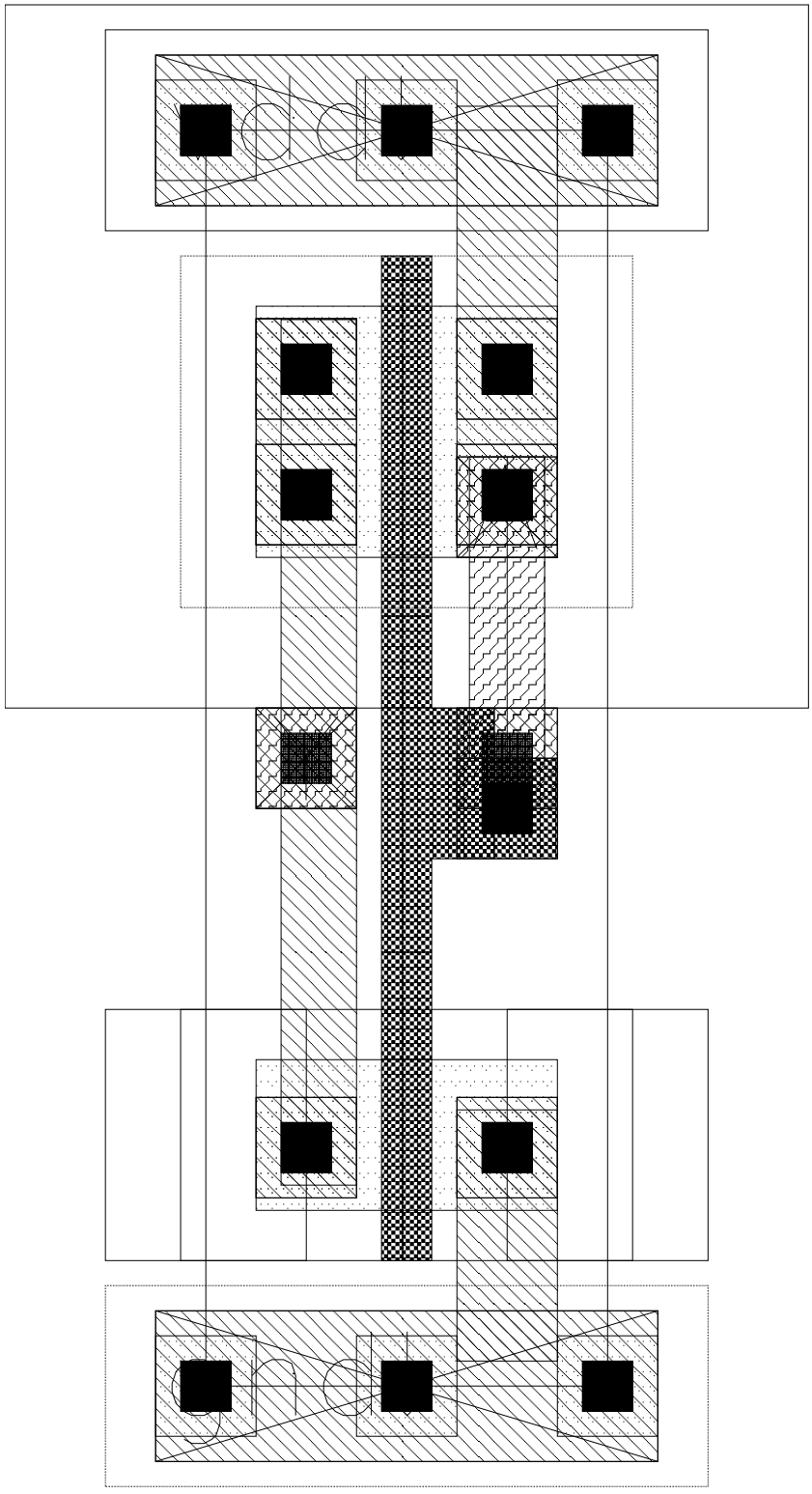

Figure M.26: Layout of invx1

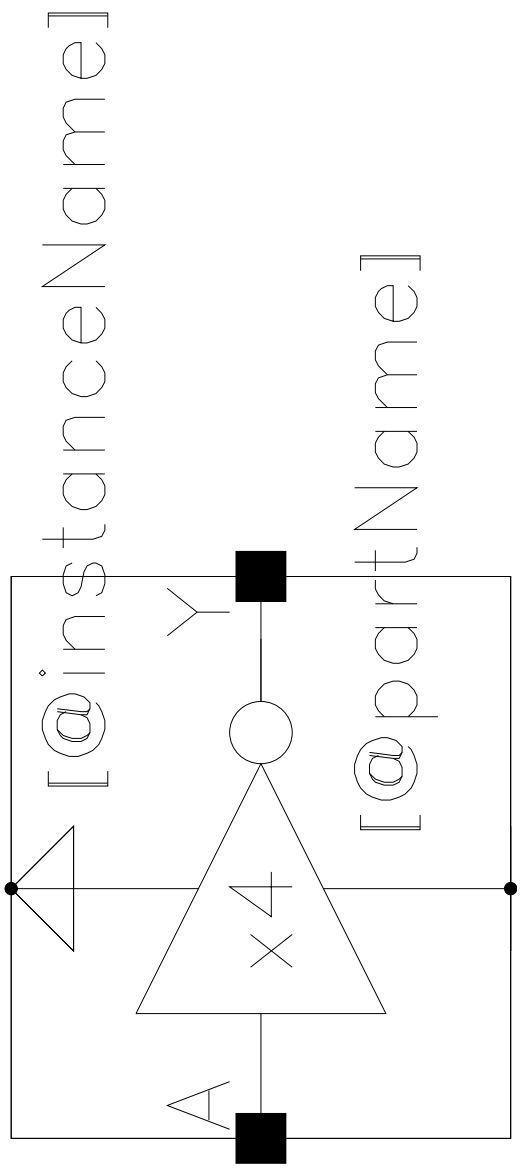

Figure M.27: Symbol of invx4

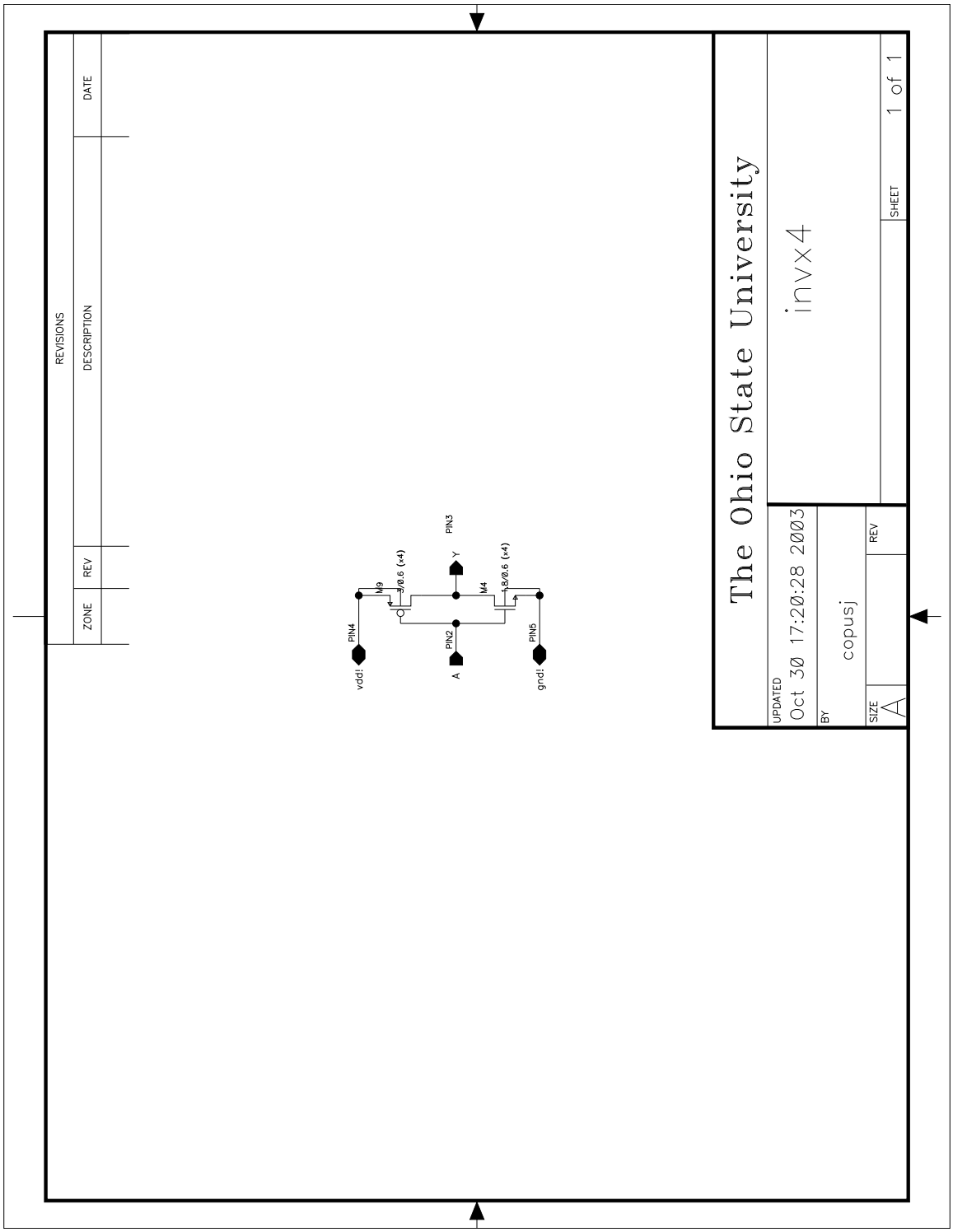

Figure M.28: Schematic of invx4

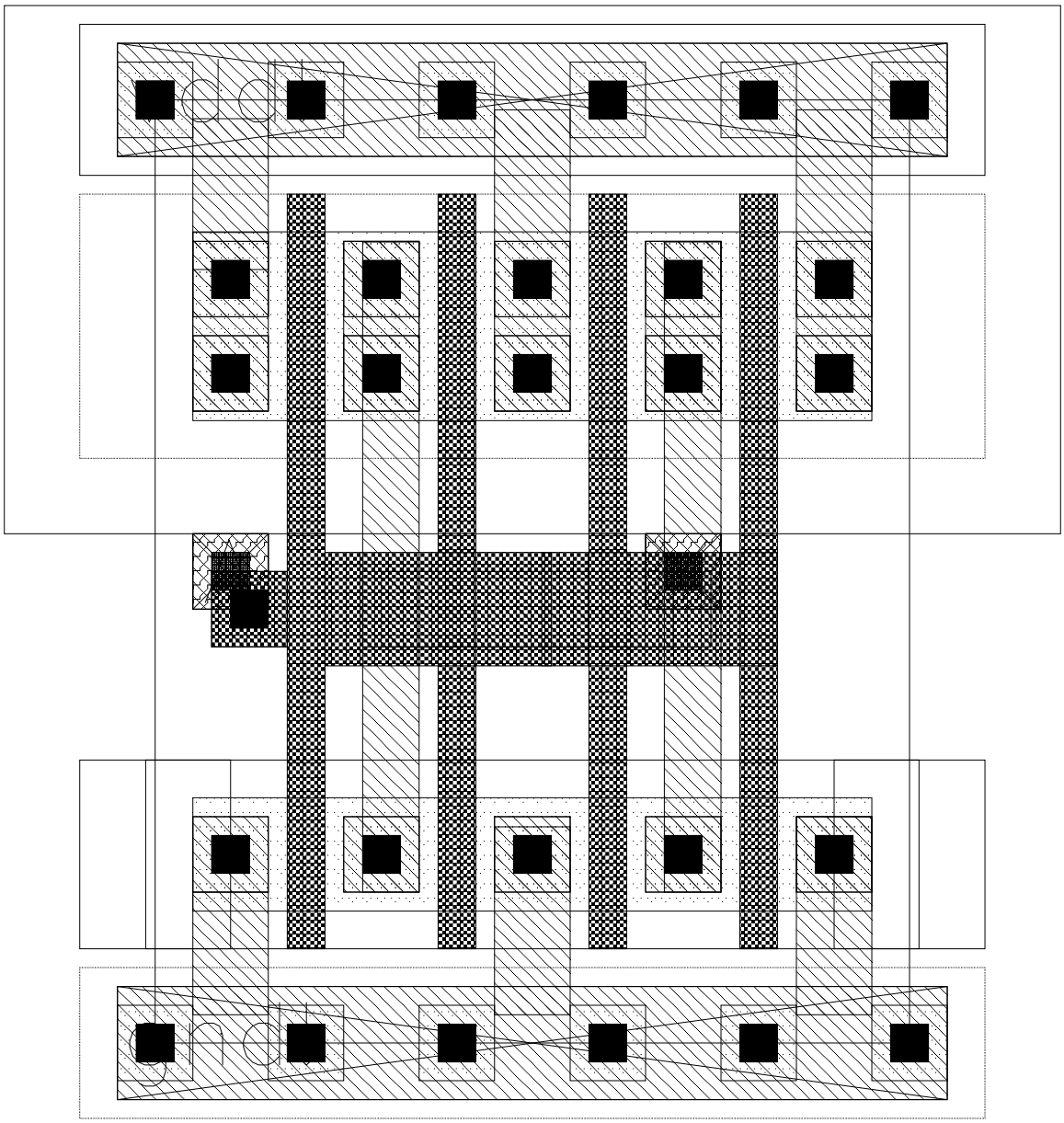

Figure M.29: Layout of invx4

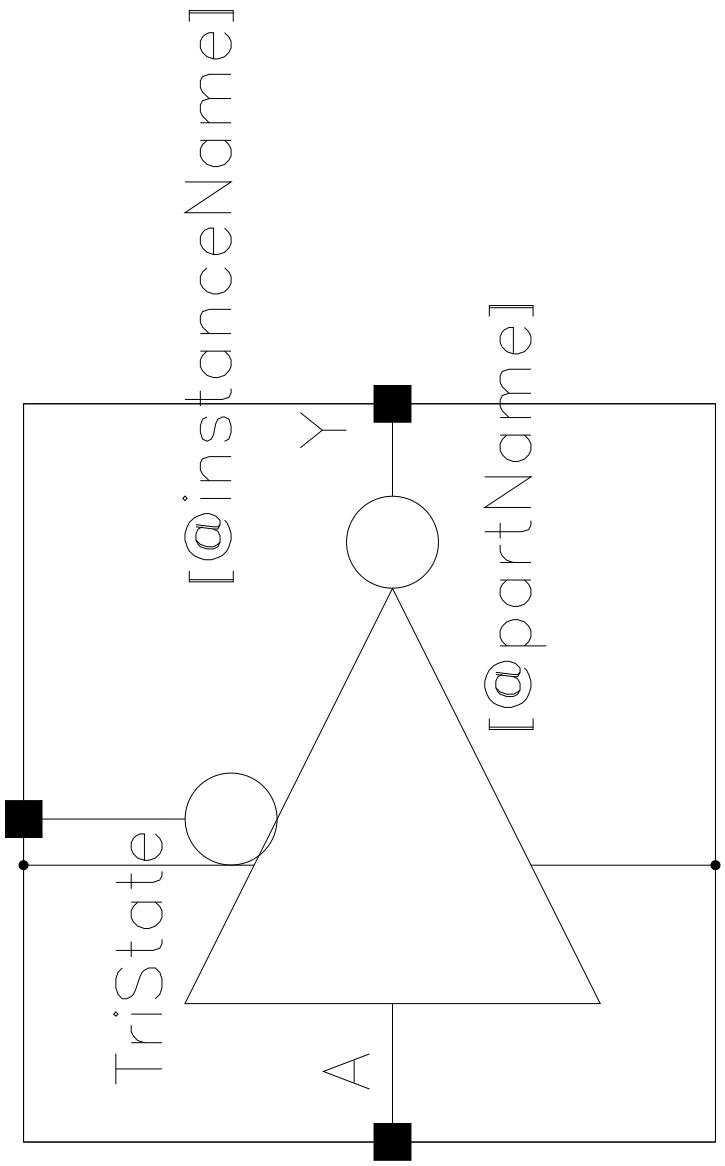

Figure M.30: Symbol of invzx1

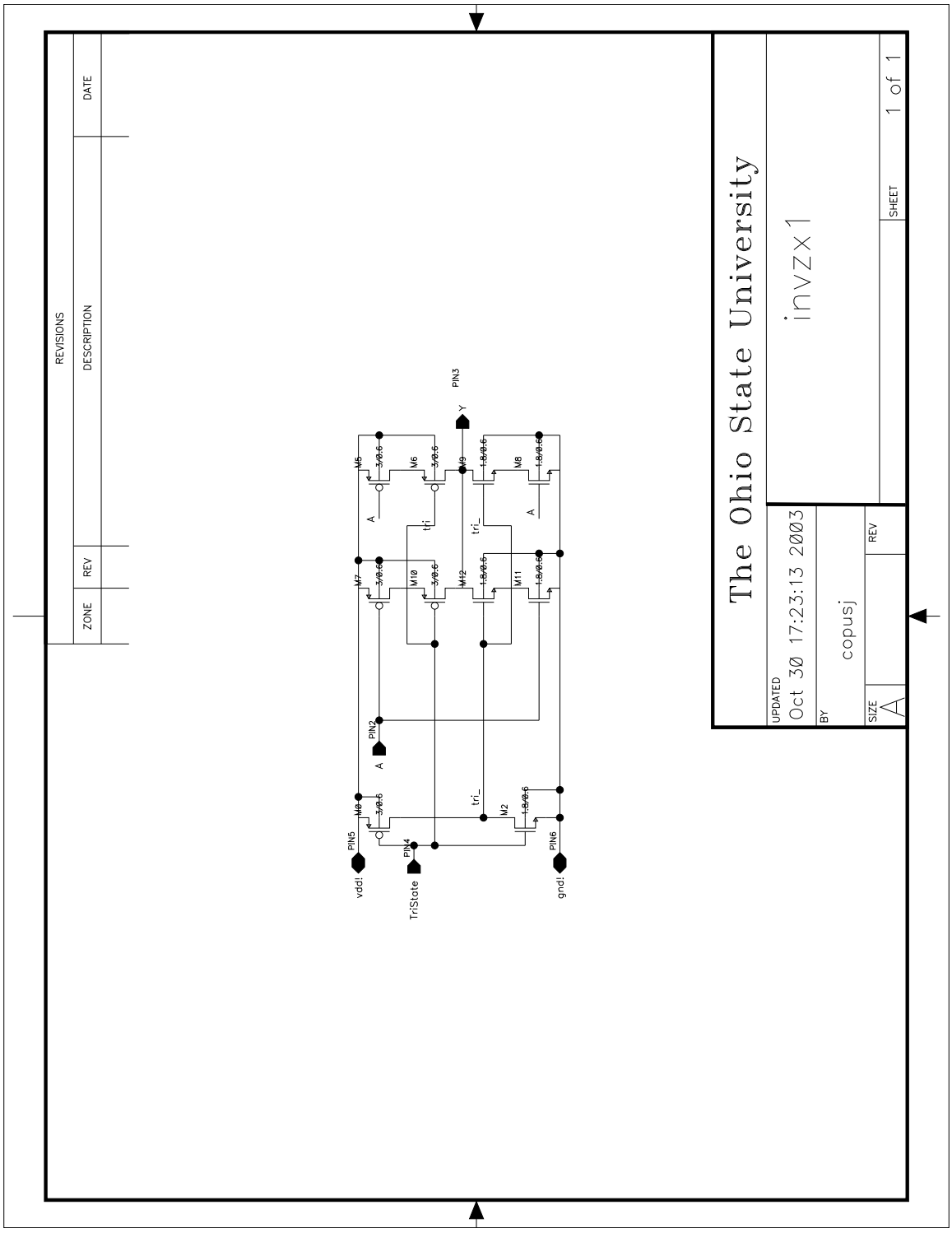

Figure M.31: Schematic of invzx1

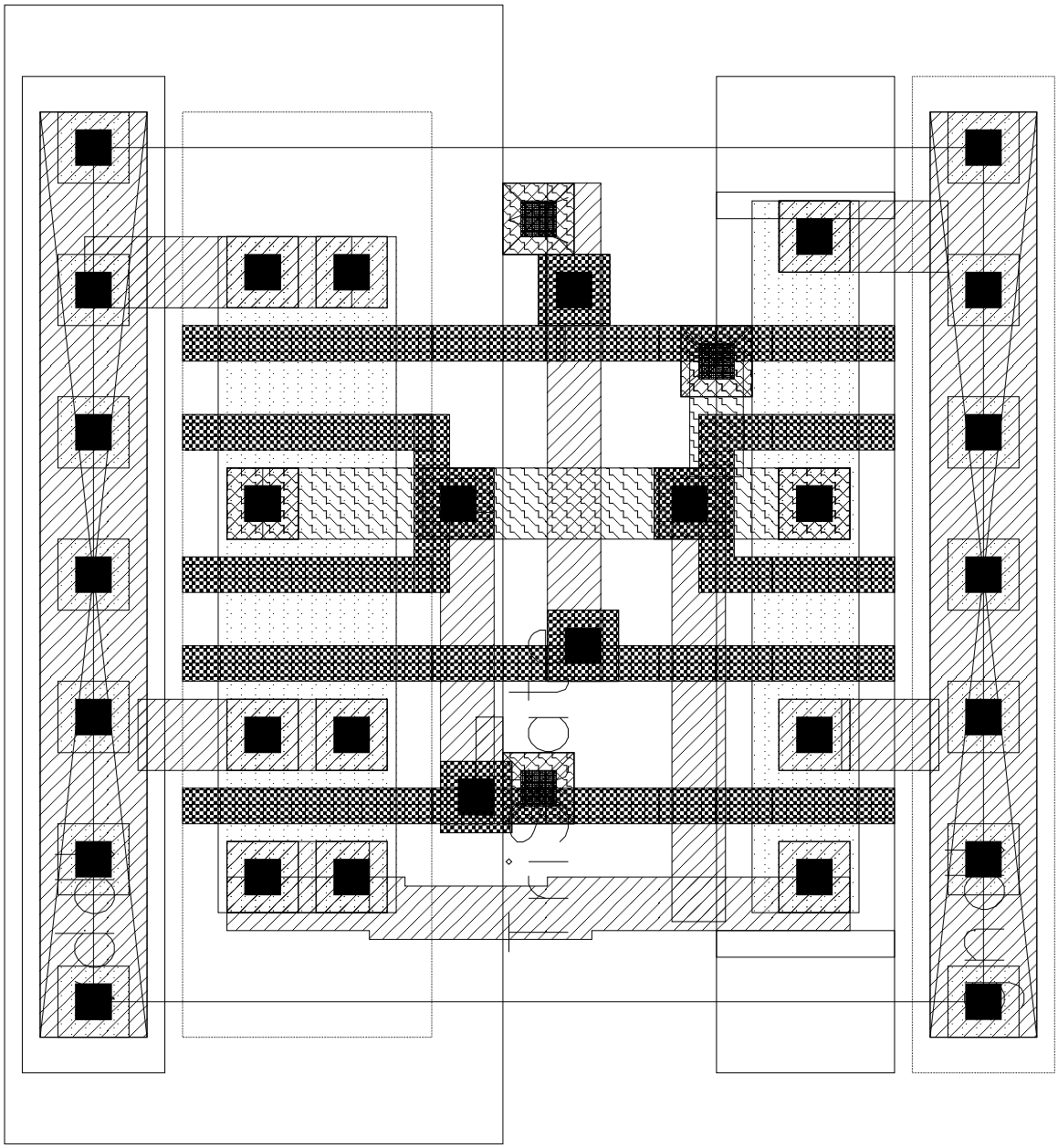

Figure M.32: Layout of invzx1

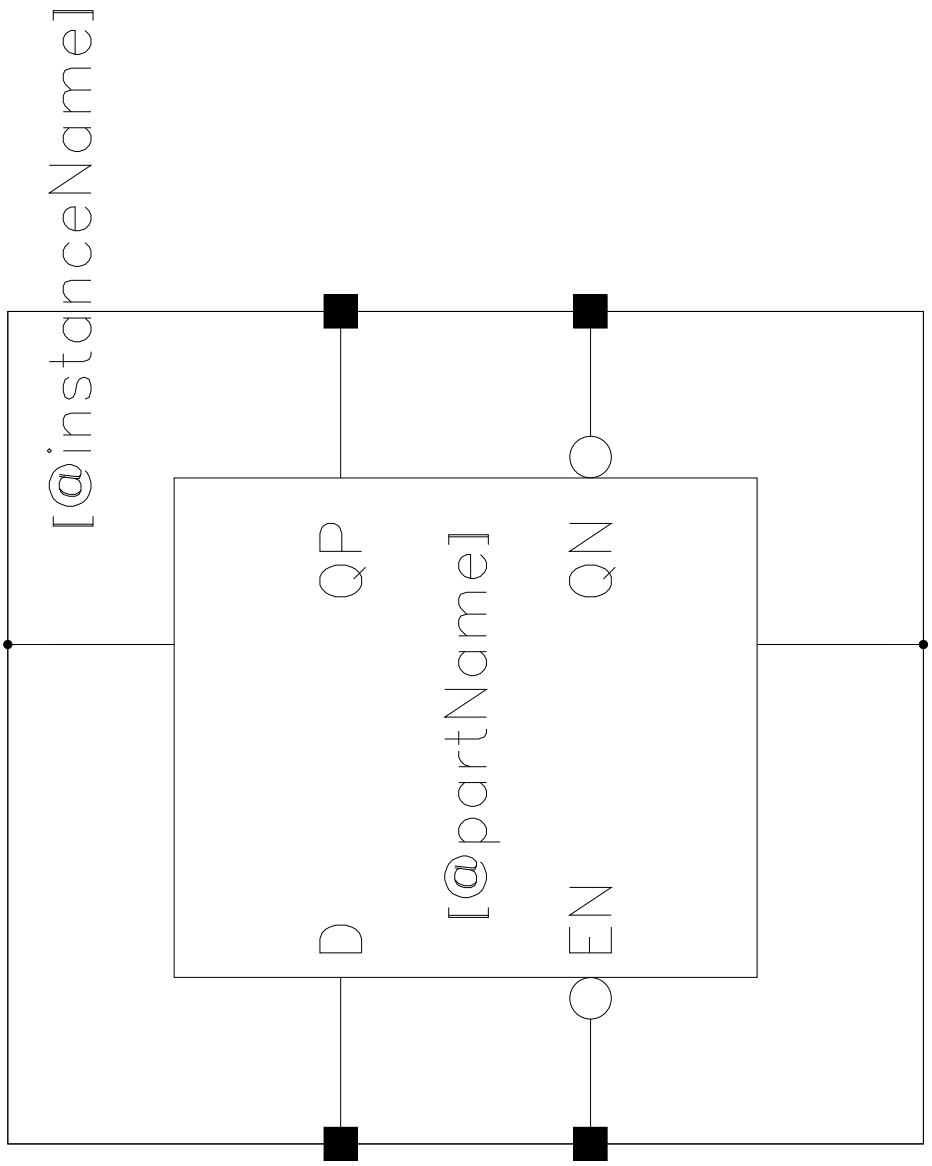

Figure M.33: Symbol of lat

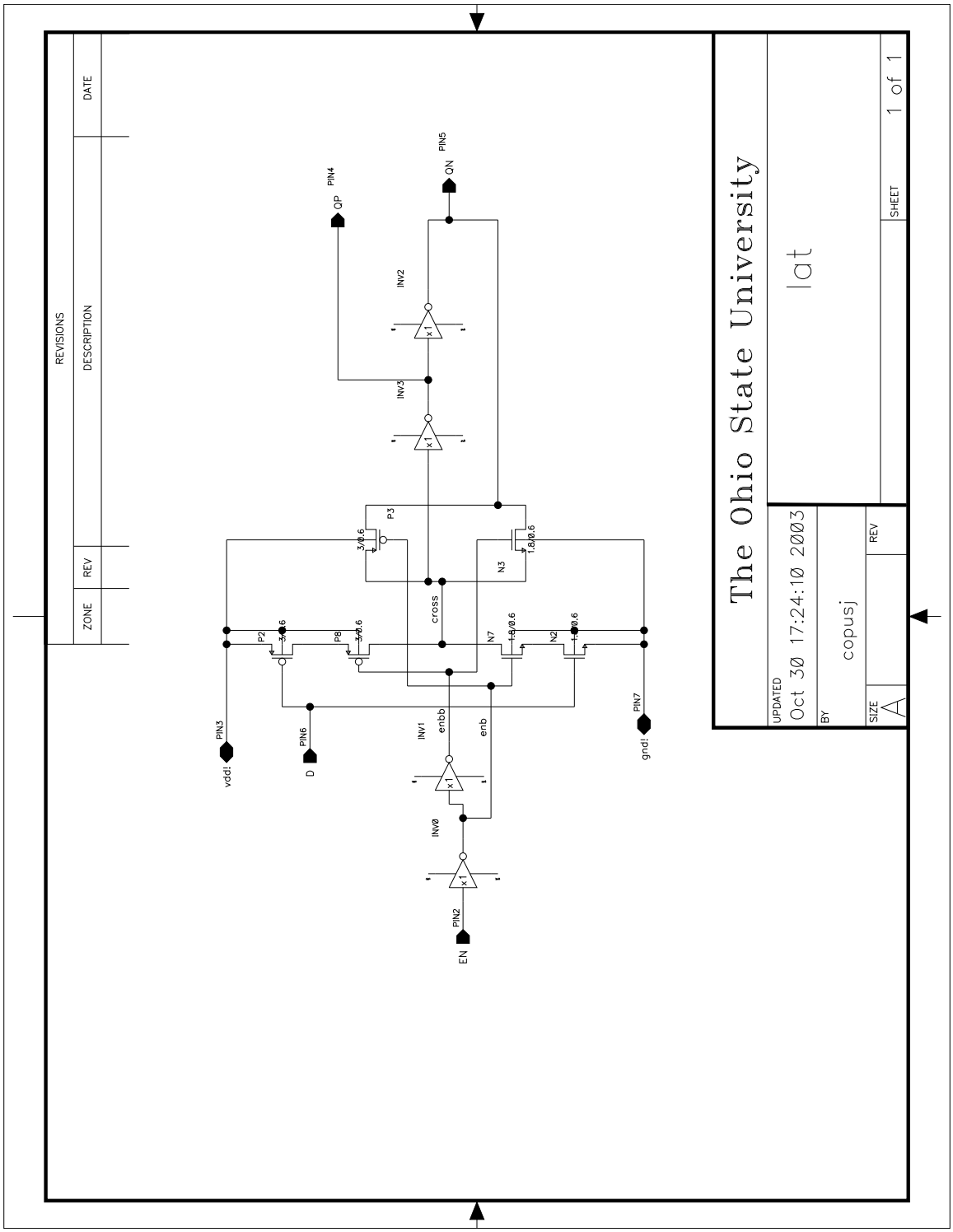

Figure M.34: Schematic of lat

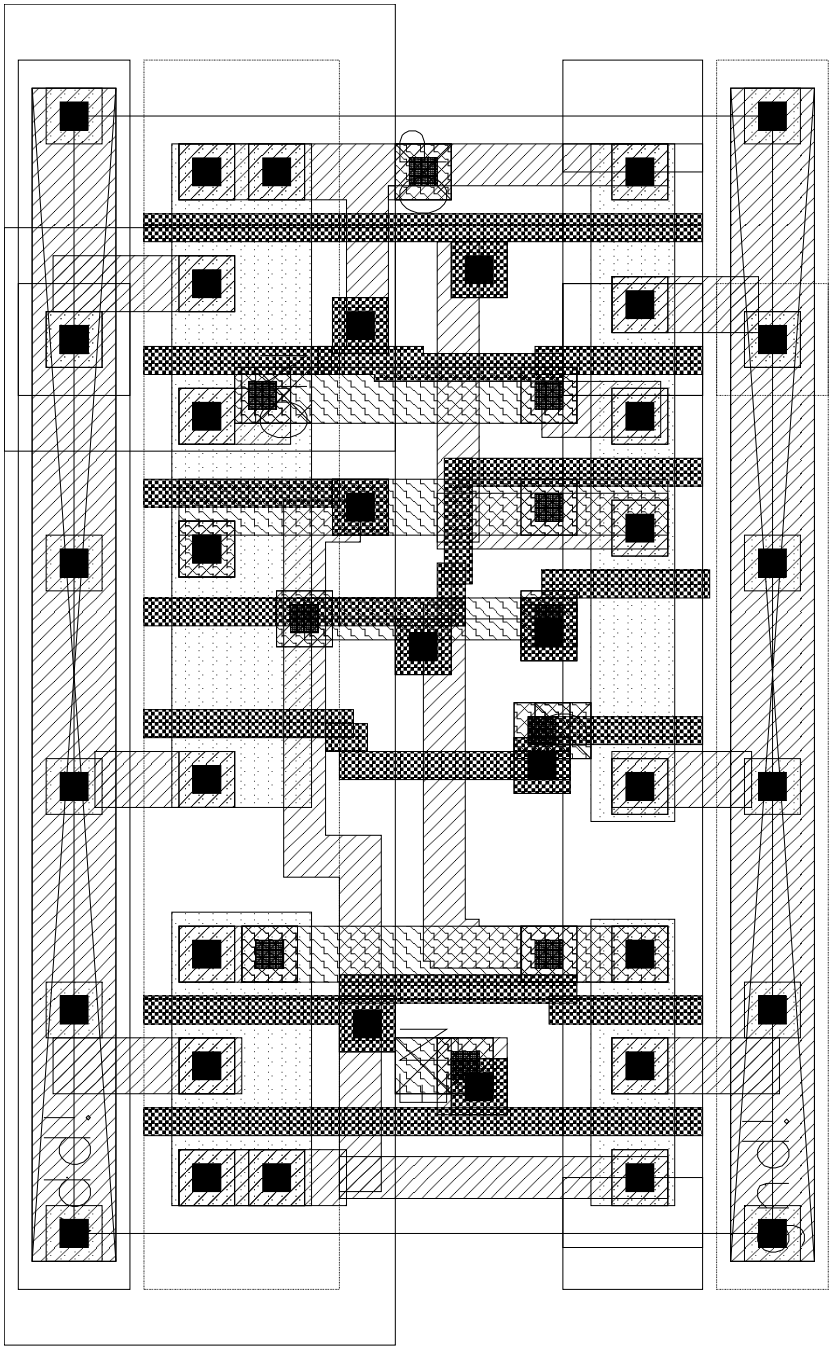

Figure M.35: Layout of lat

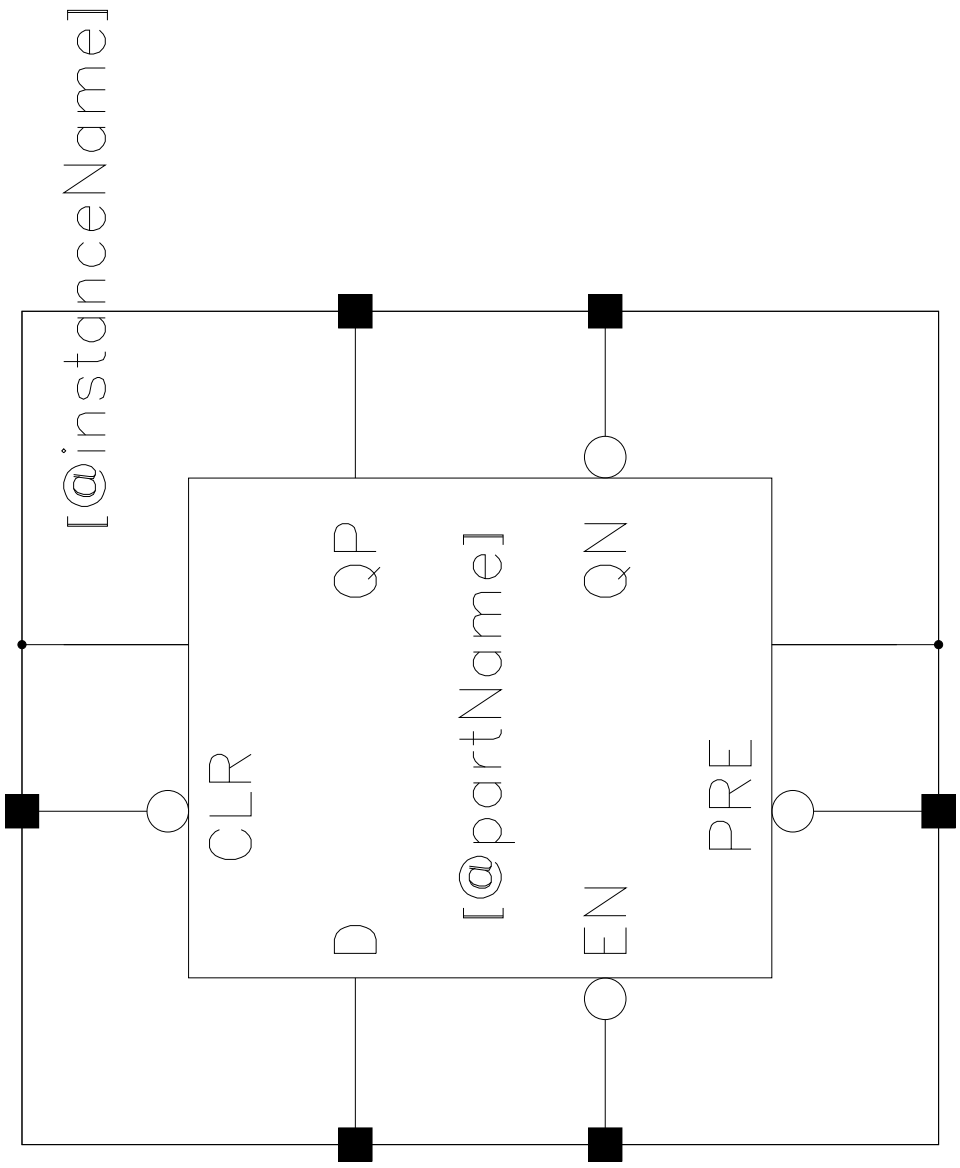

Figure M.36: Symbol of latpc

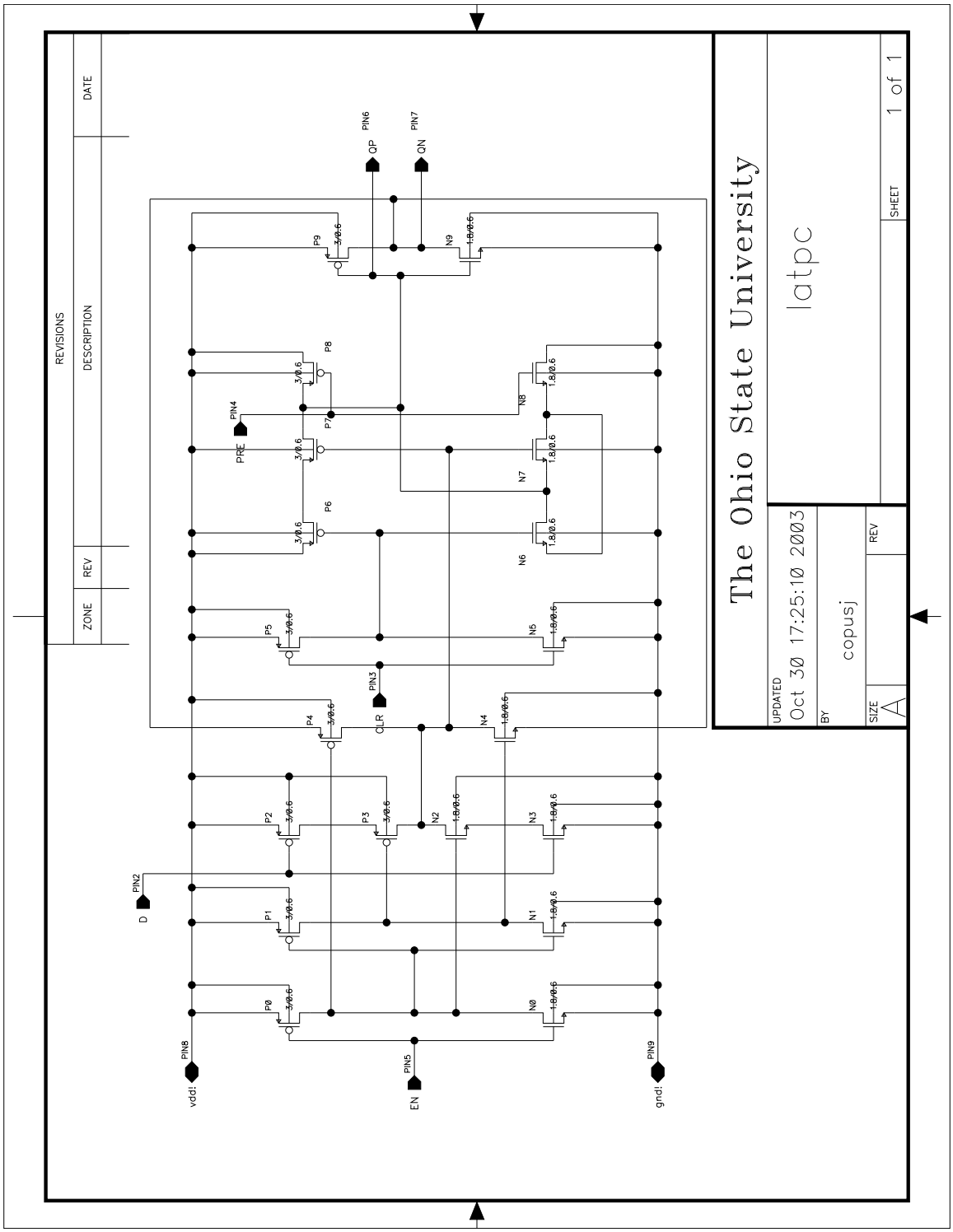

Figure M.37: Schematic of latpc

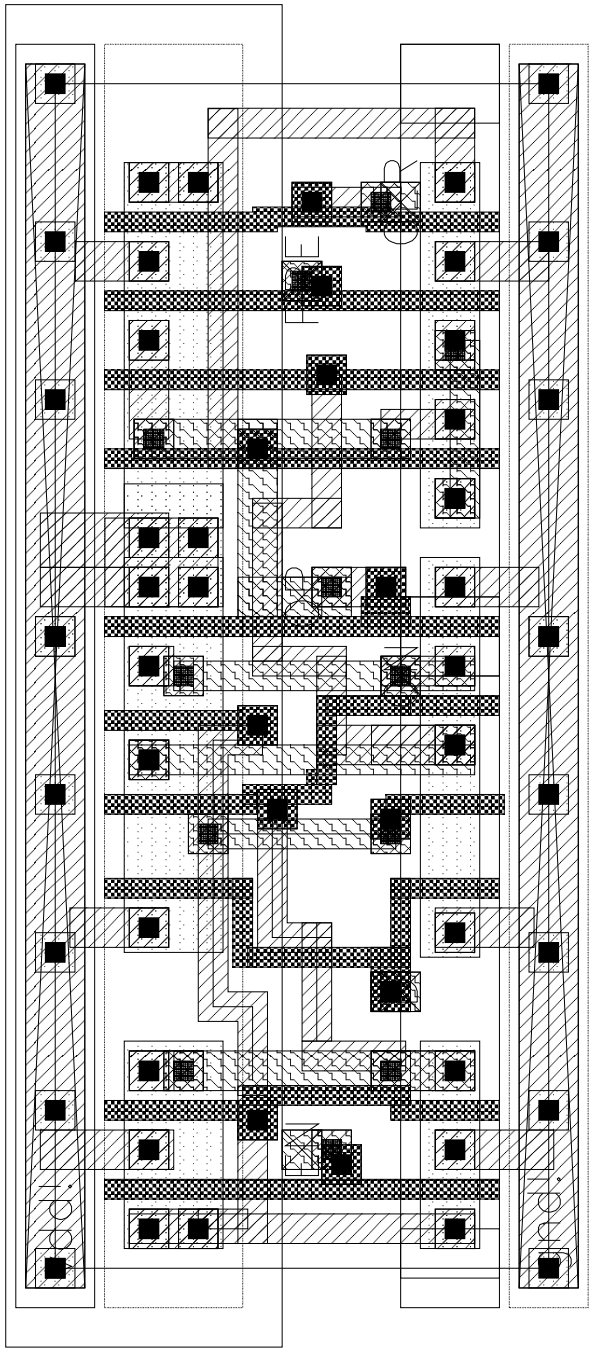

Figure M.38: Layout of latpc

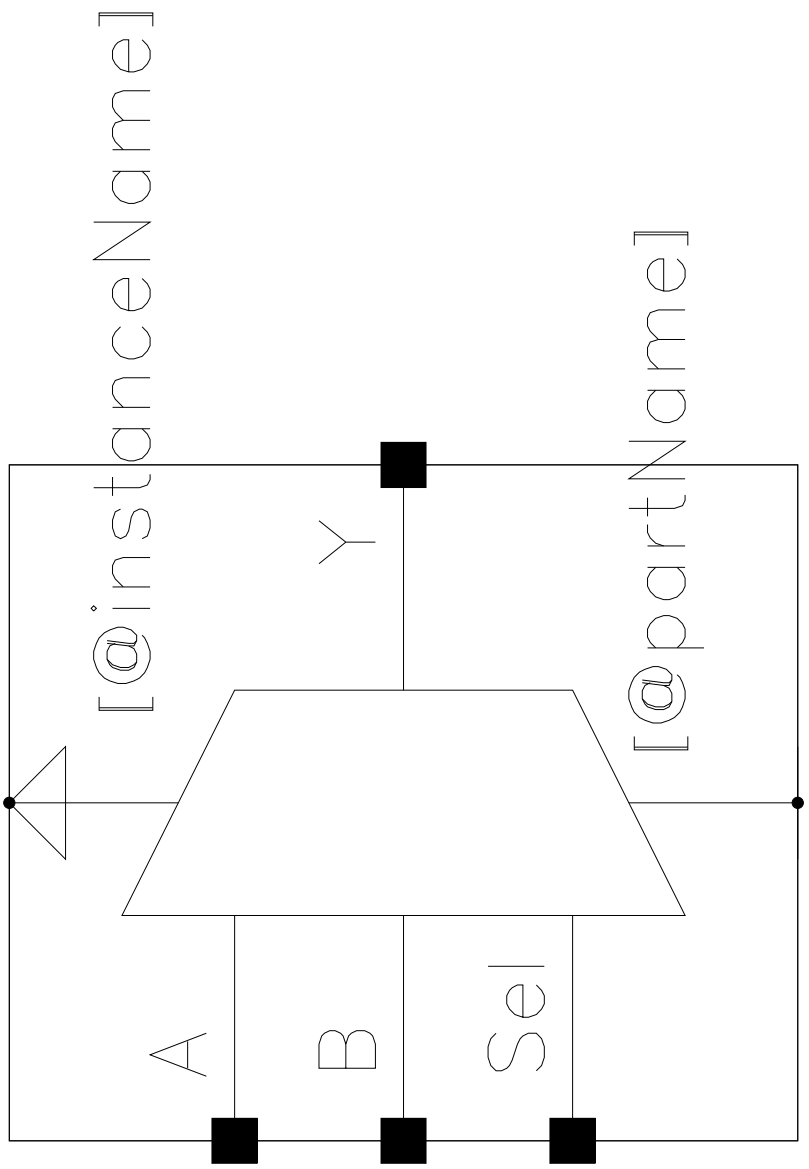

Figure M.39: Symbol of mux21x1

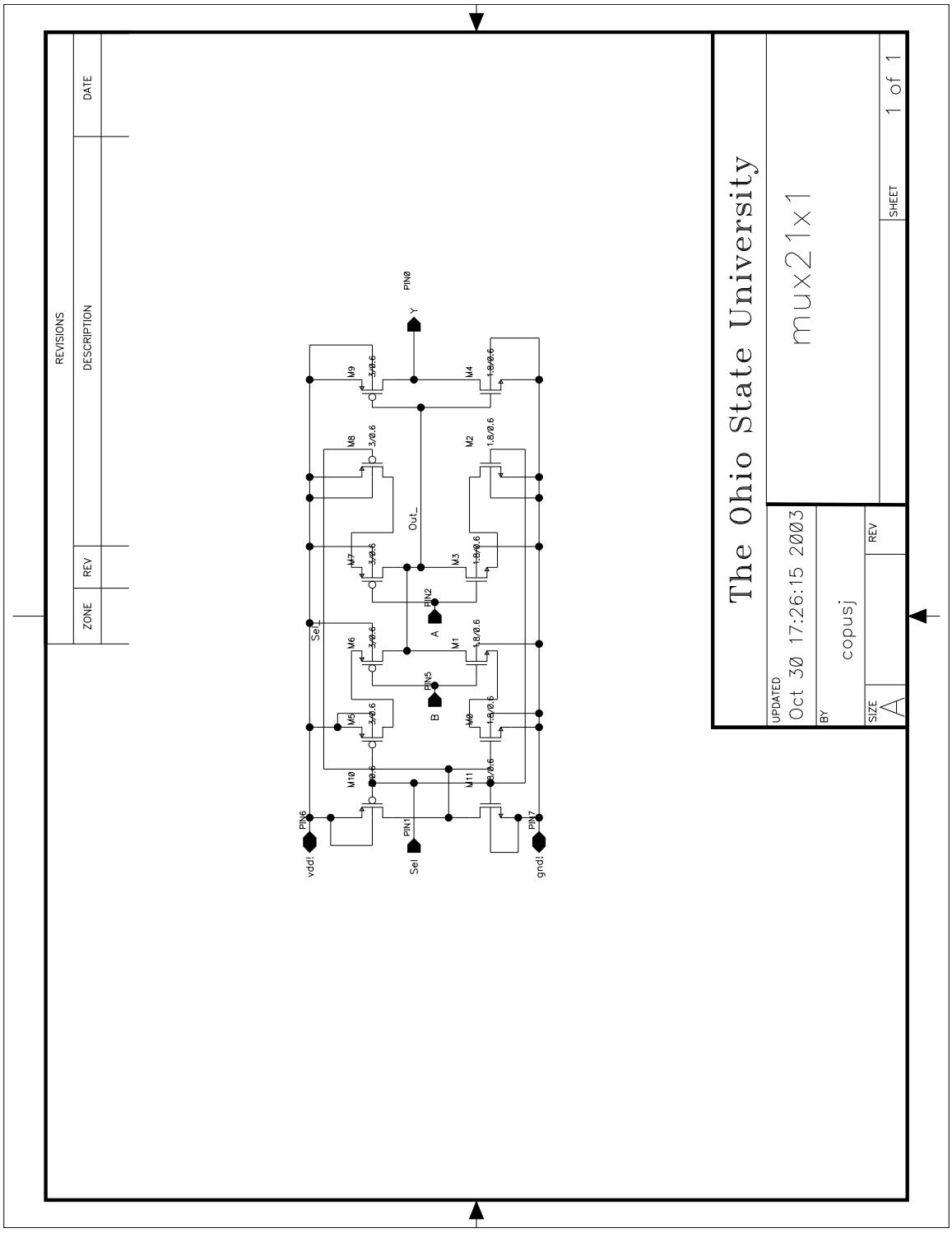

Figure M.40: Schematic of mux21x1

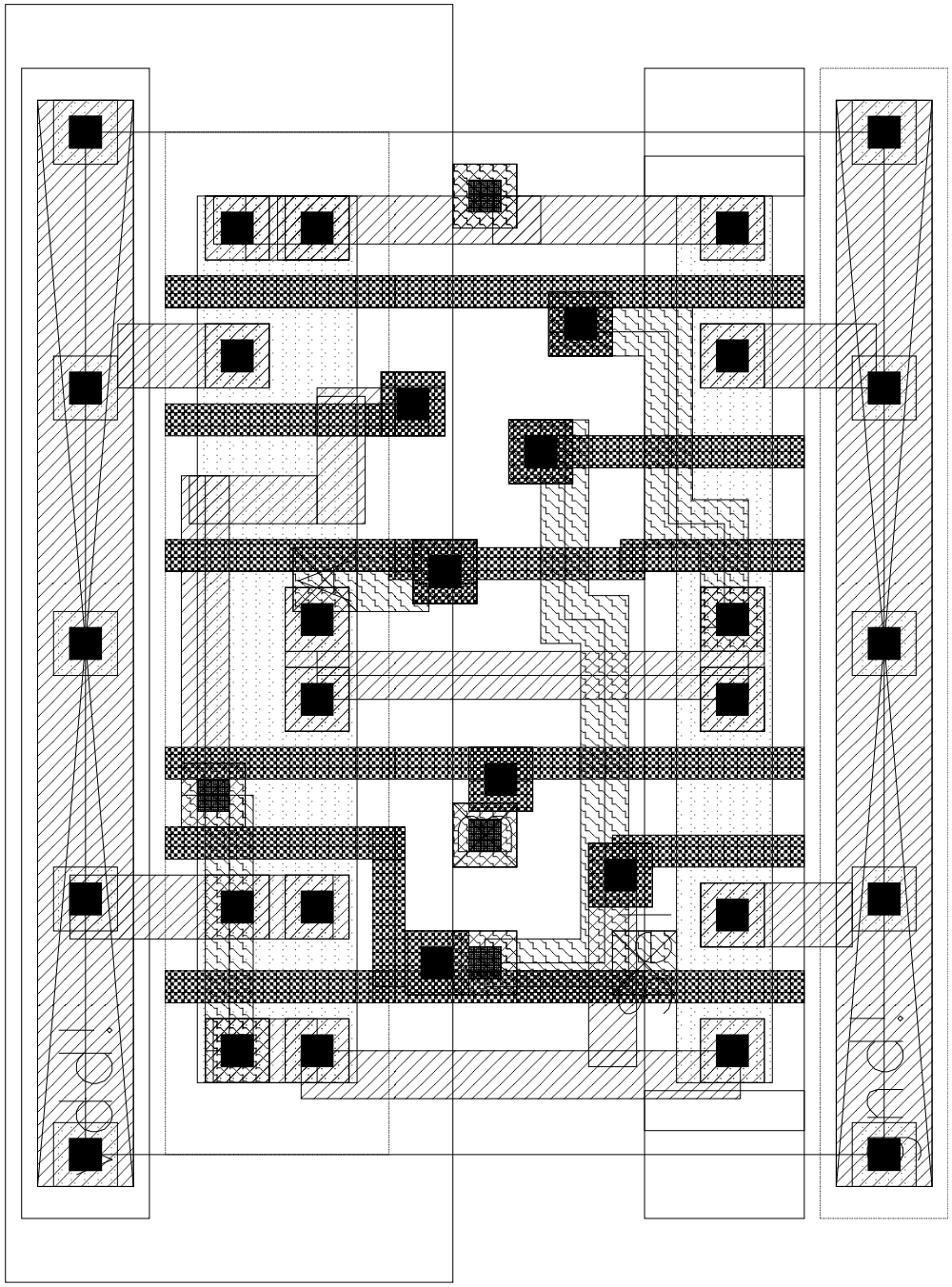

Figure M.41: Layout of mux21x1

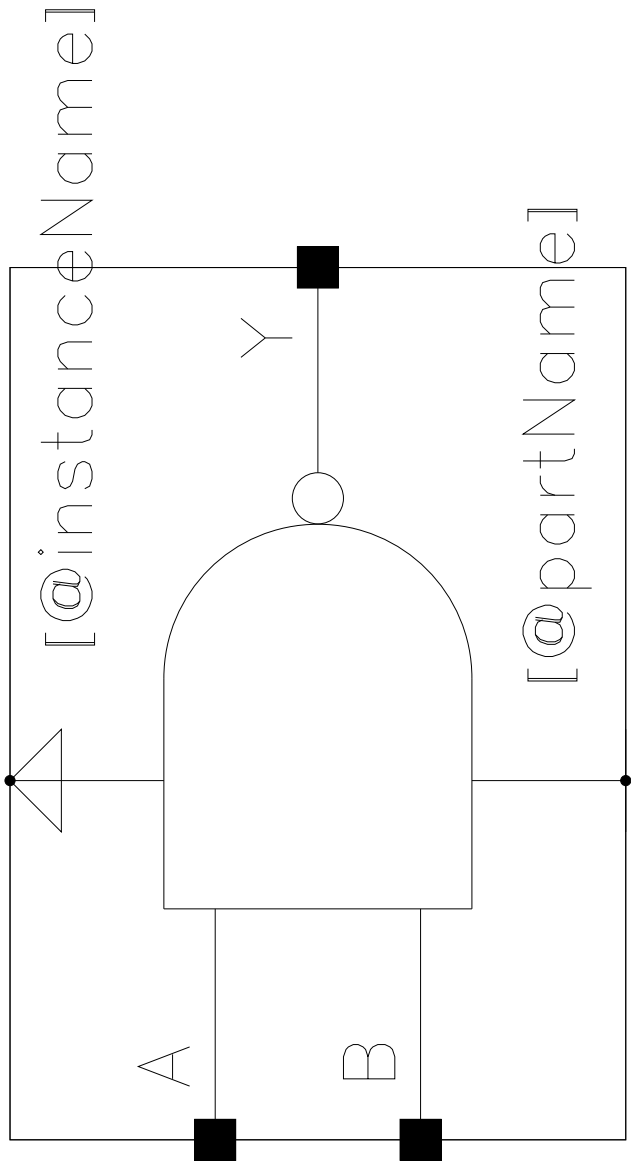

Figure M.42: Symbol of nand2x1

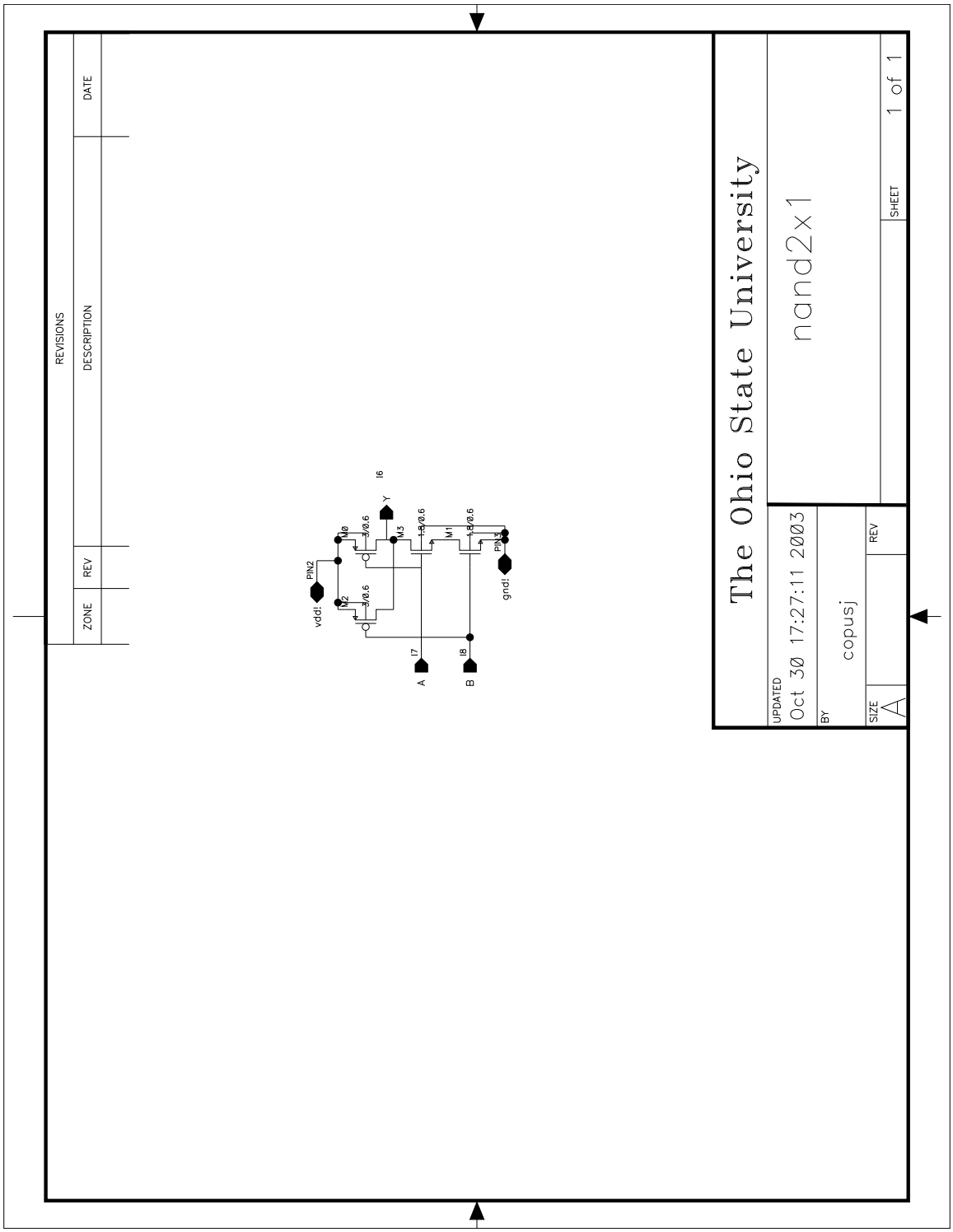

Figure M.43: Schematic of nand2x1

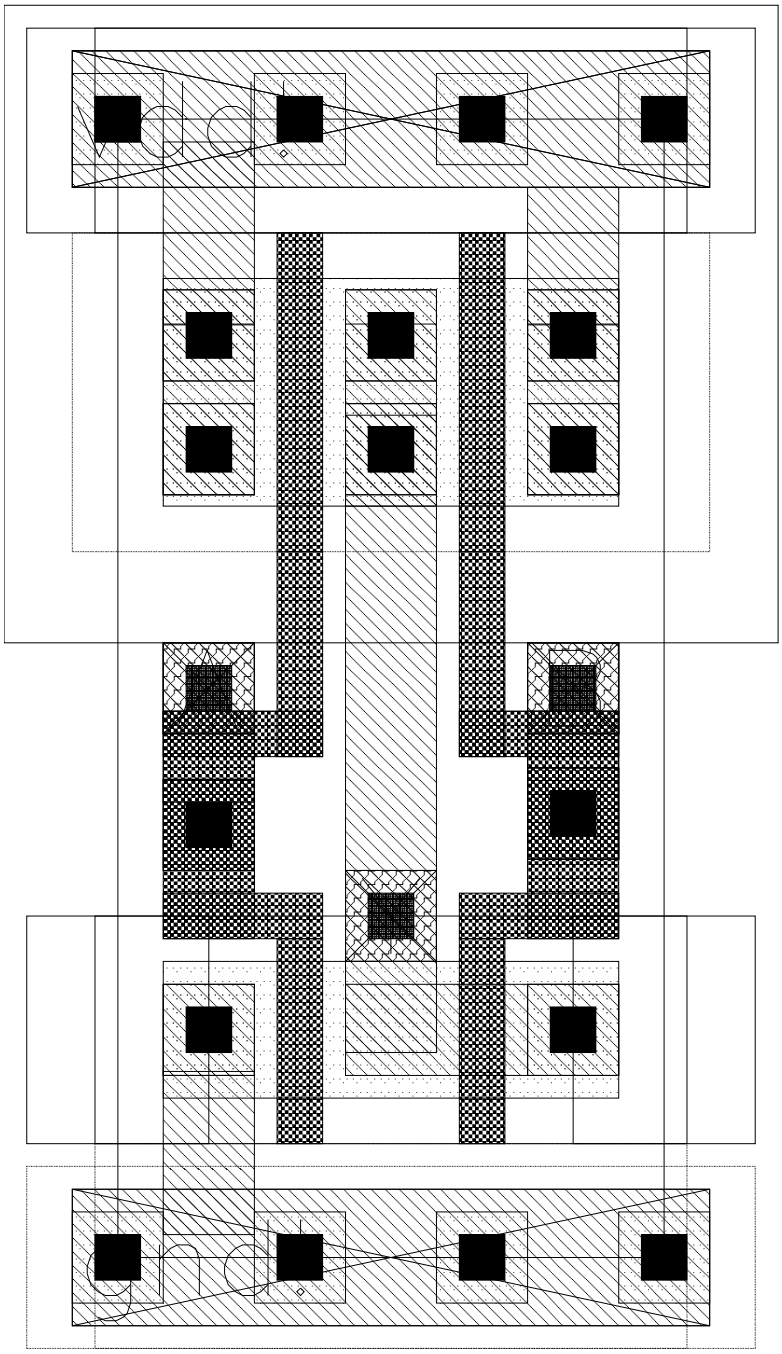

Figure M.44: Layout of nand2x1

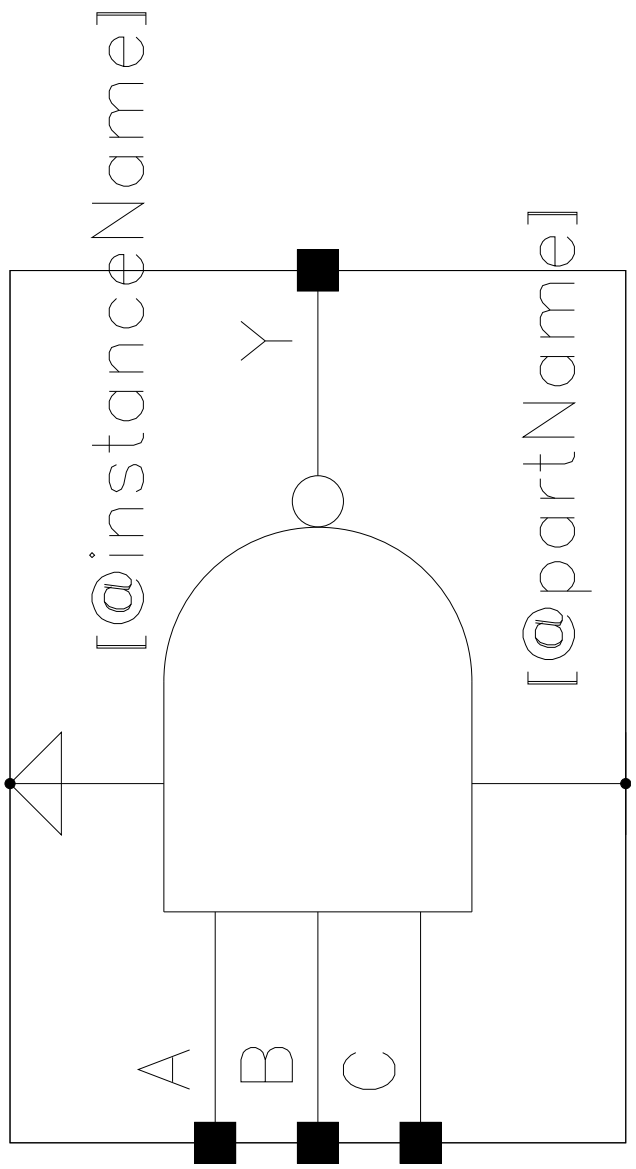

Figure M.45: Symbol of nand3x1

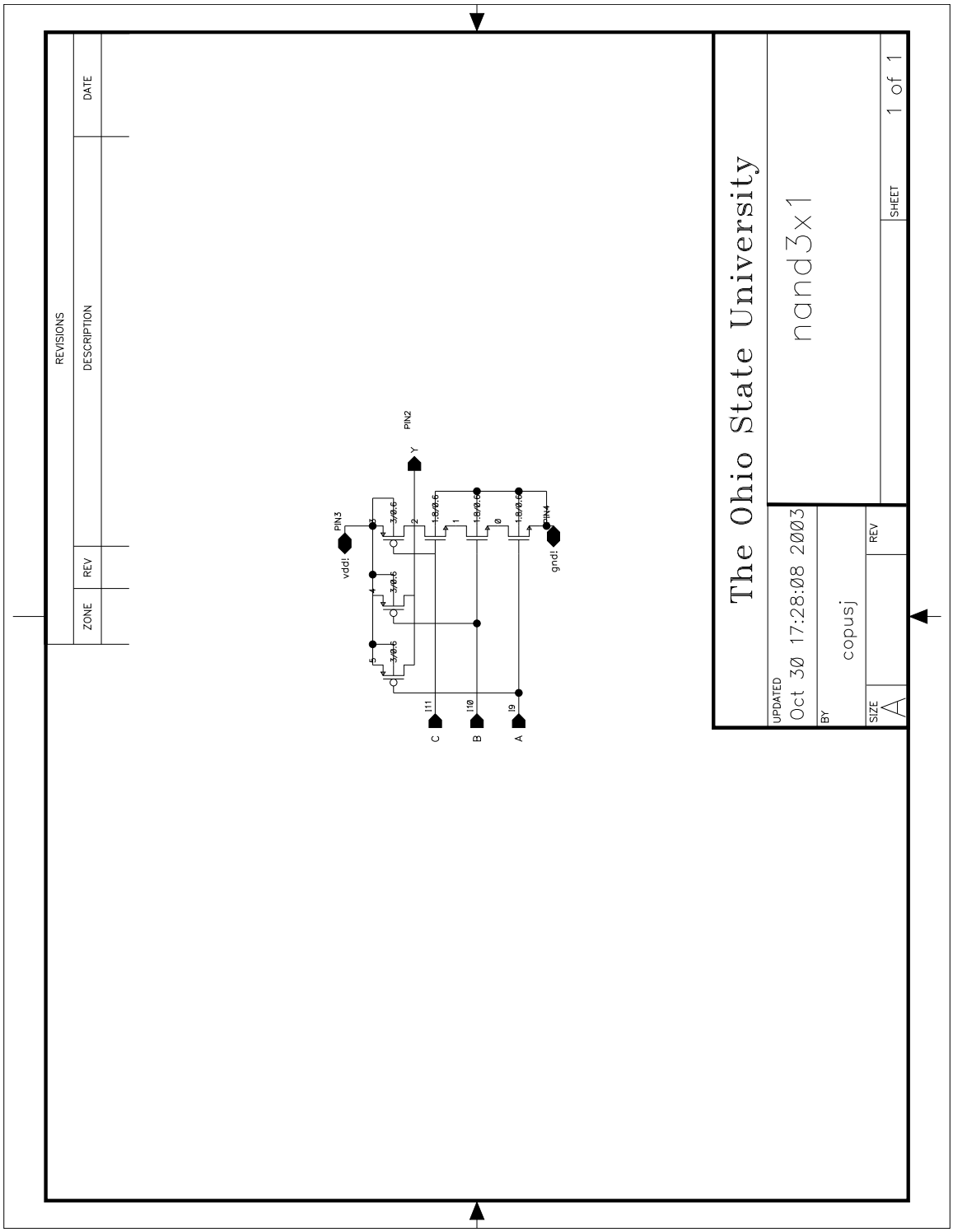

Figure M.46: Schematic of nand3x1

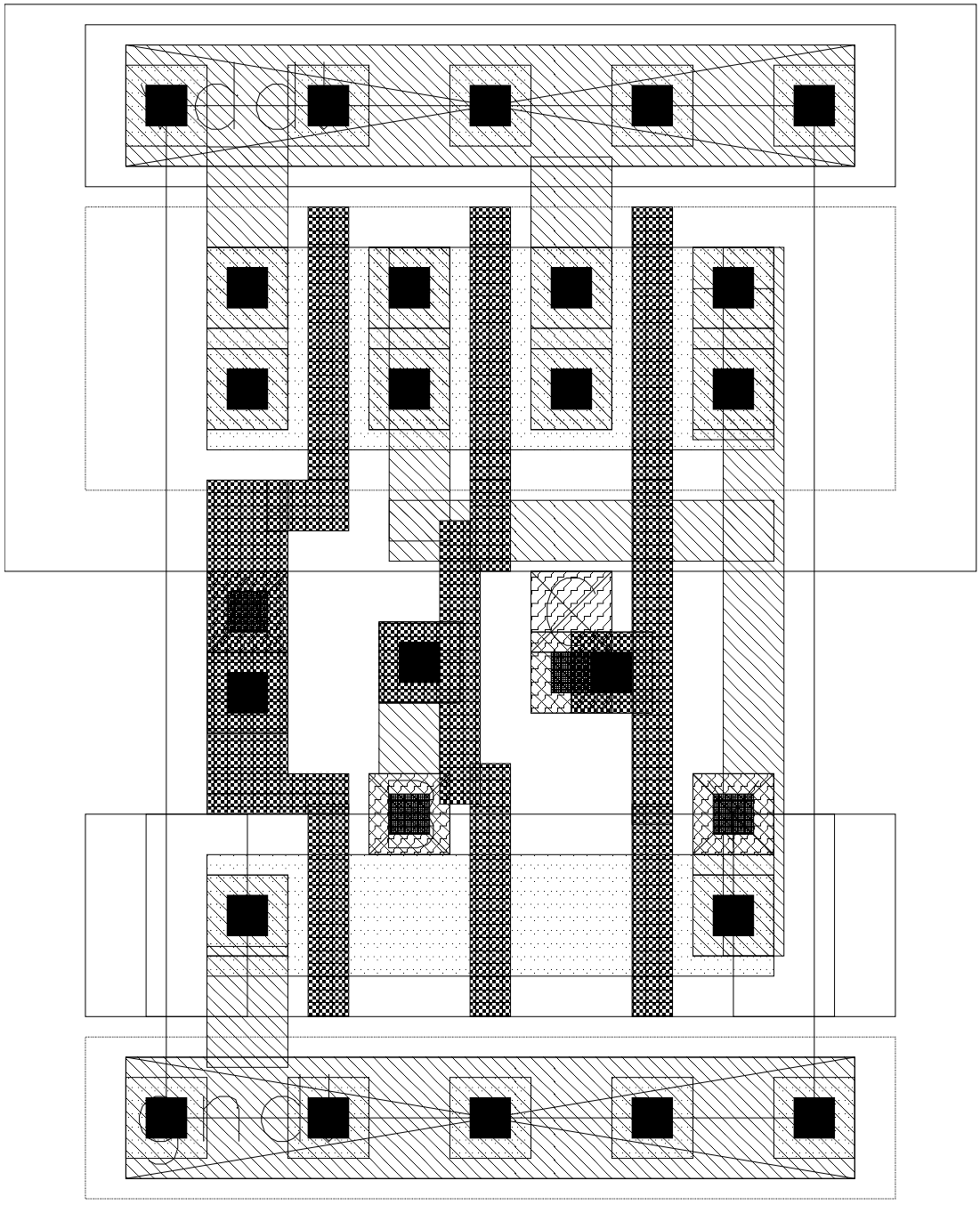

Figure M.47: Layout of nand3x1

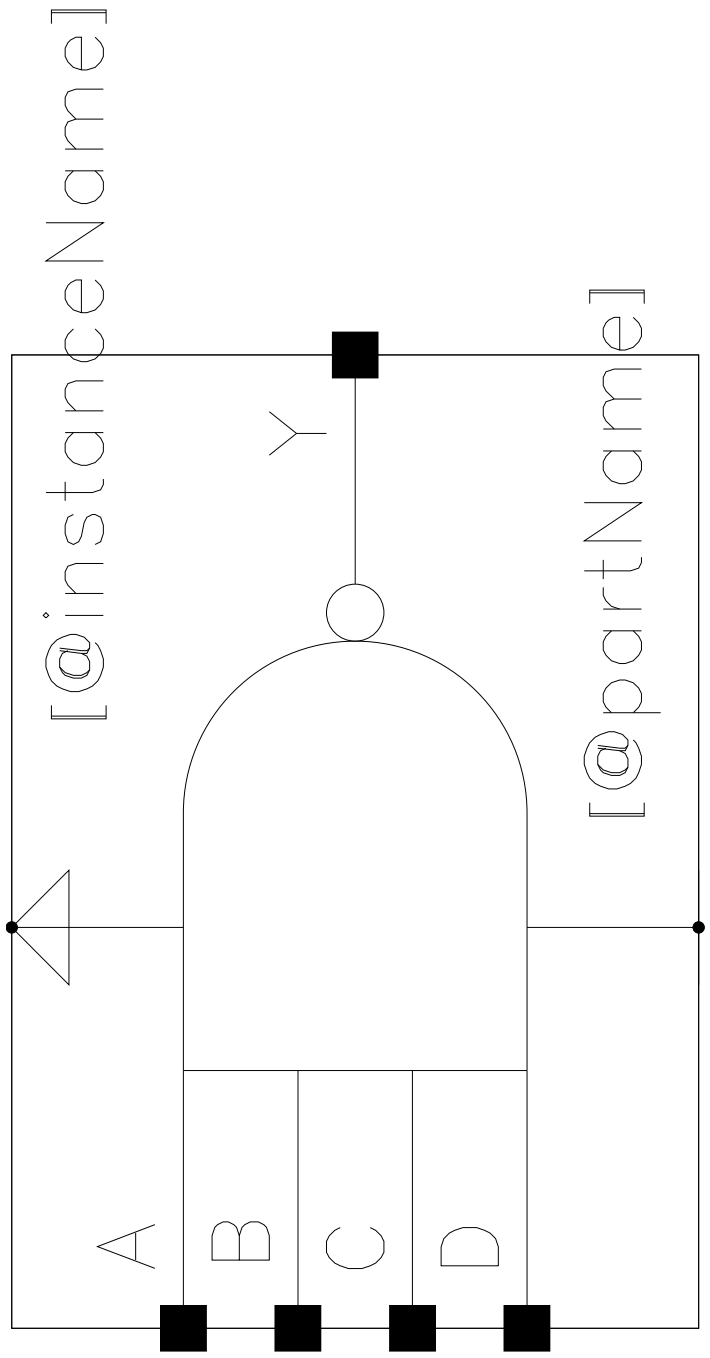

Figure M.48: Symbol of nand4x1

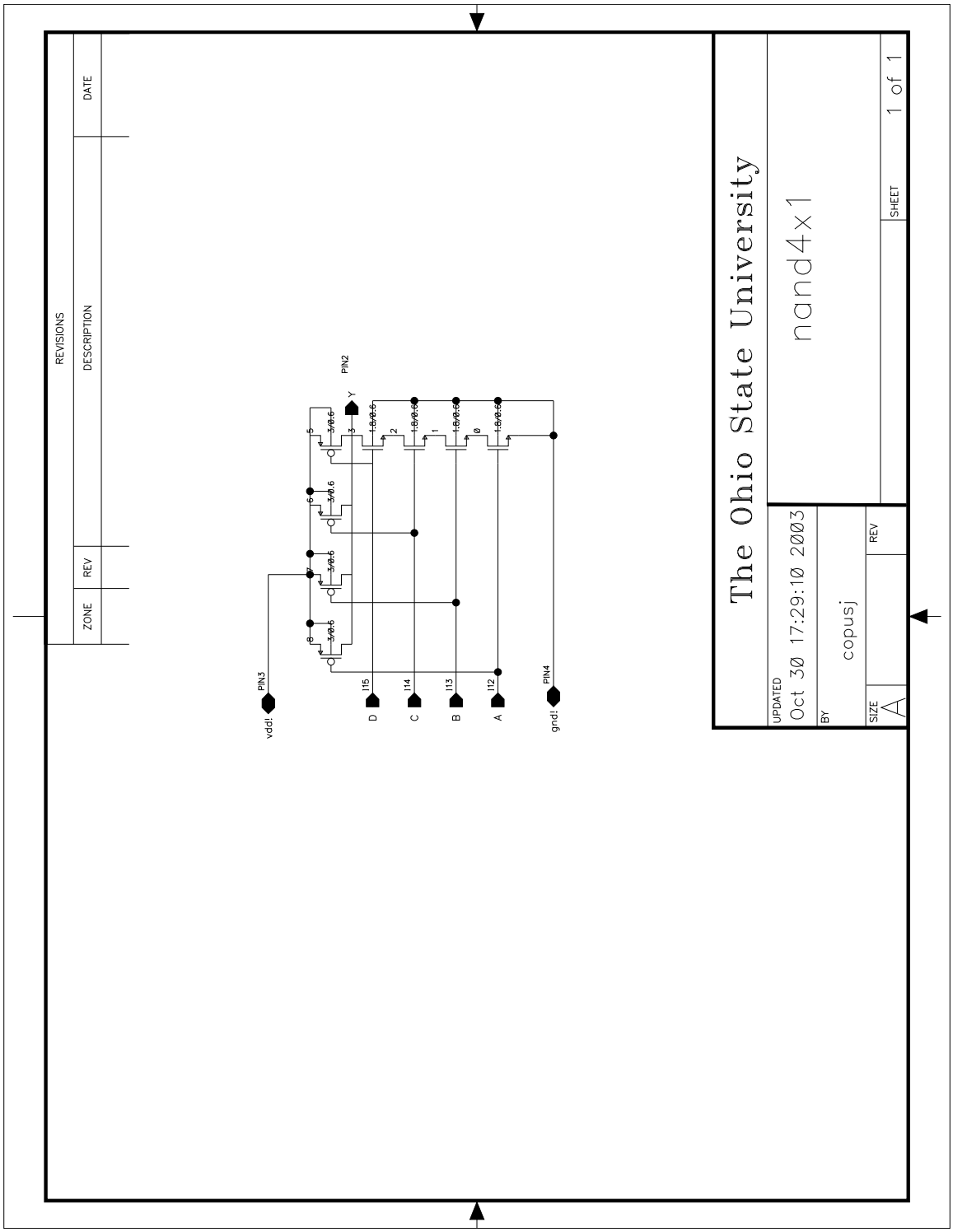

Figure M.49: Schematic of nand4x1

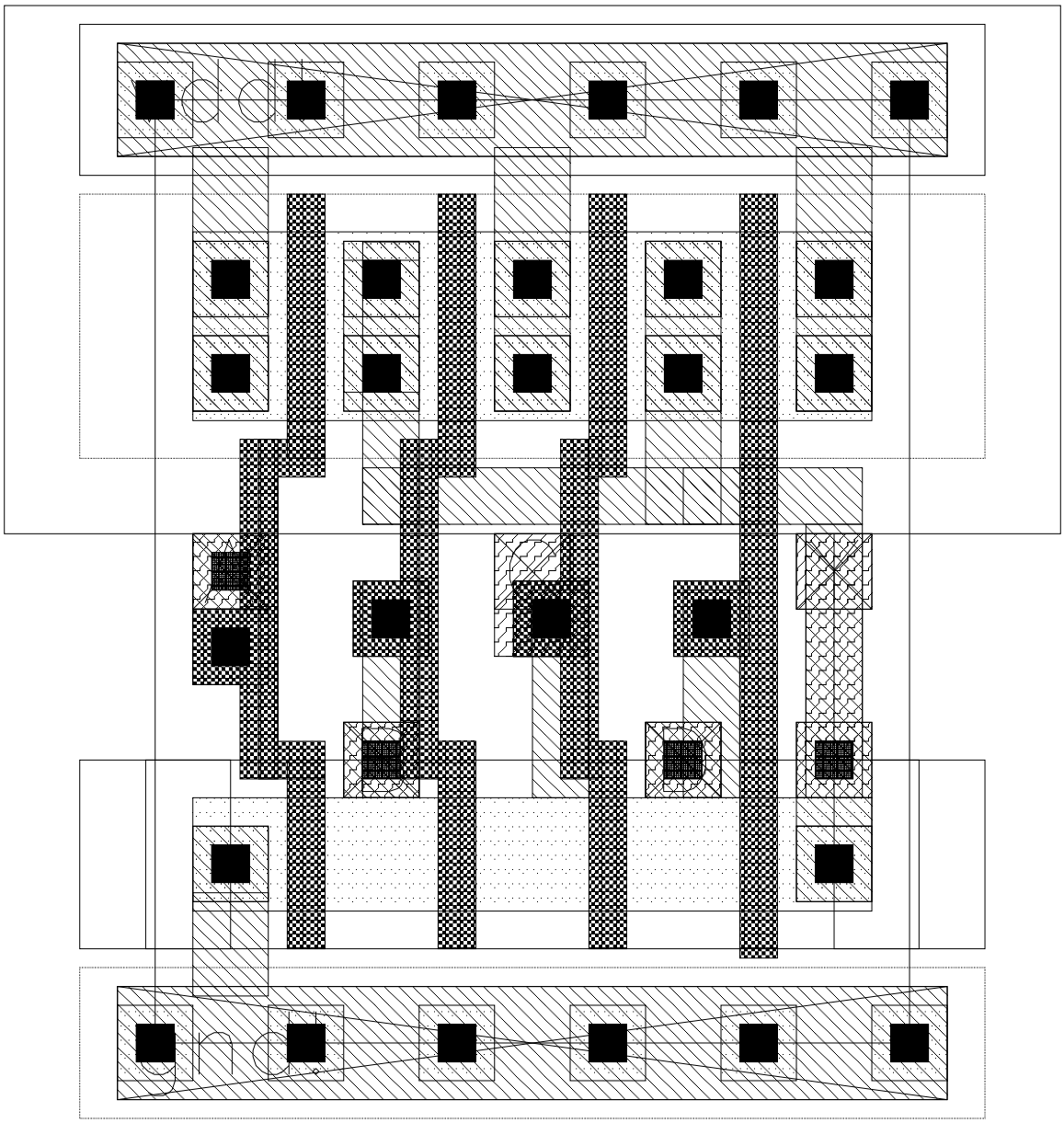

Figure M.50: Layout of nand4x1

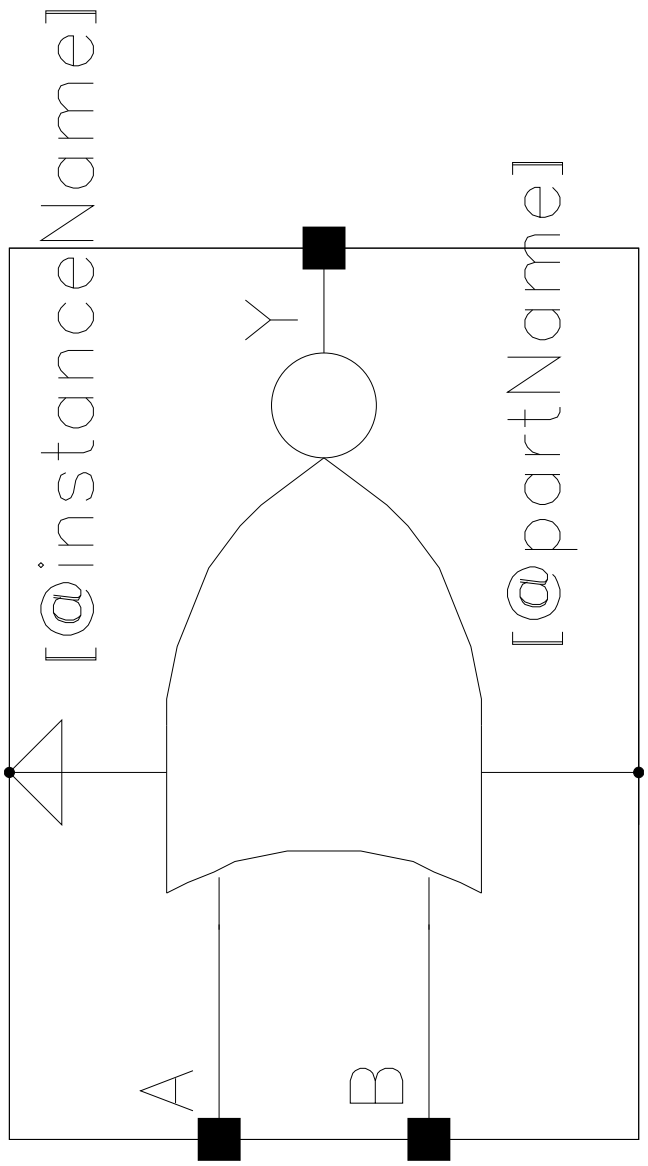

Figure M.51: Symbol of nor2x1

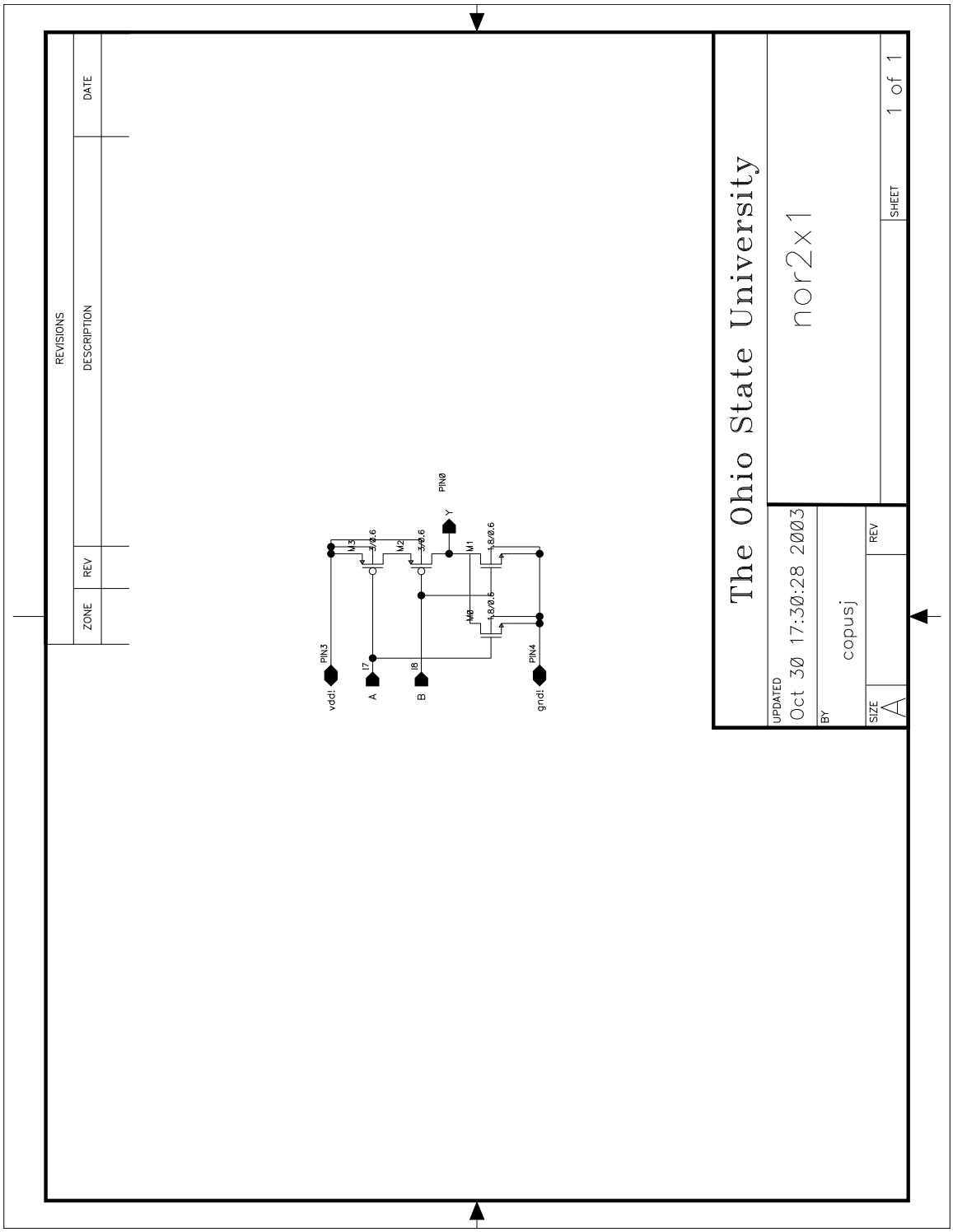

Figure M.52: Schematic of nor2x1

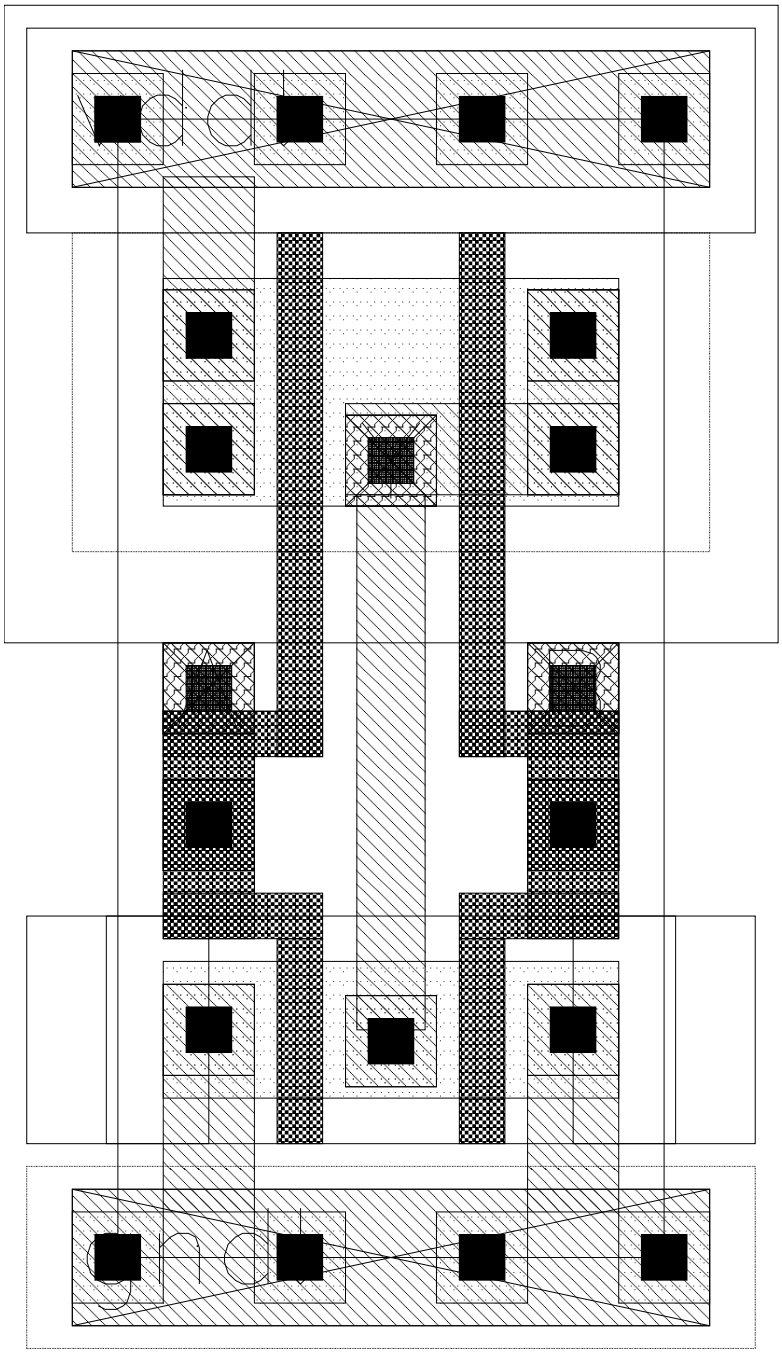

Figure M.53: Layout of nor2x1

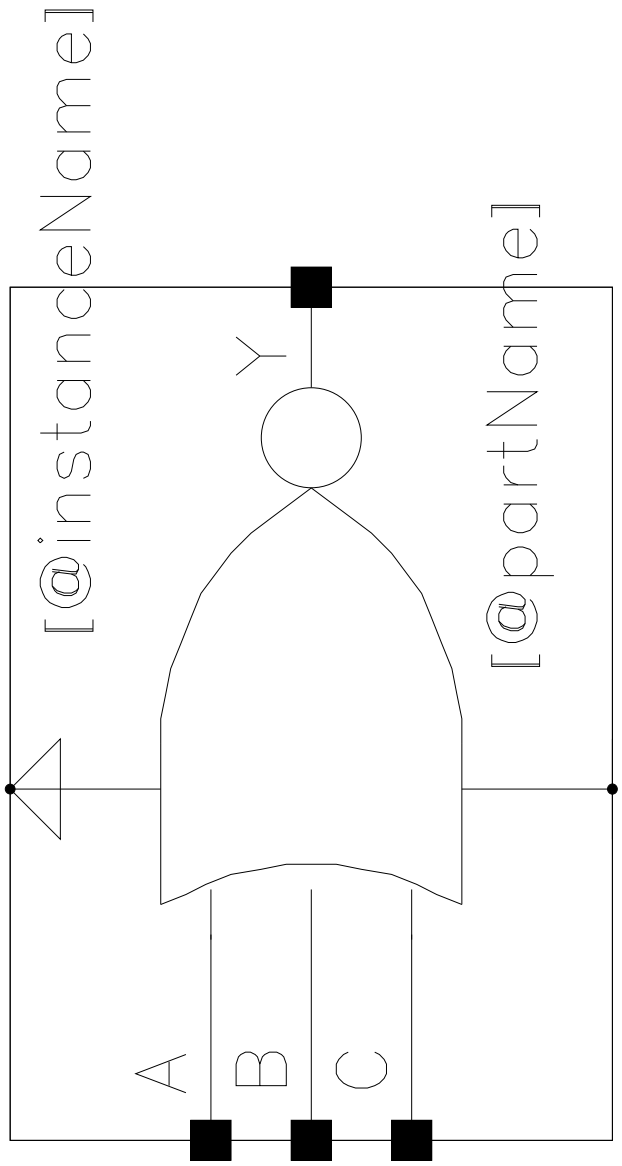

Figure M.54: Symbol of nor3x1

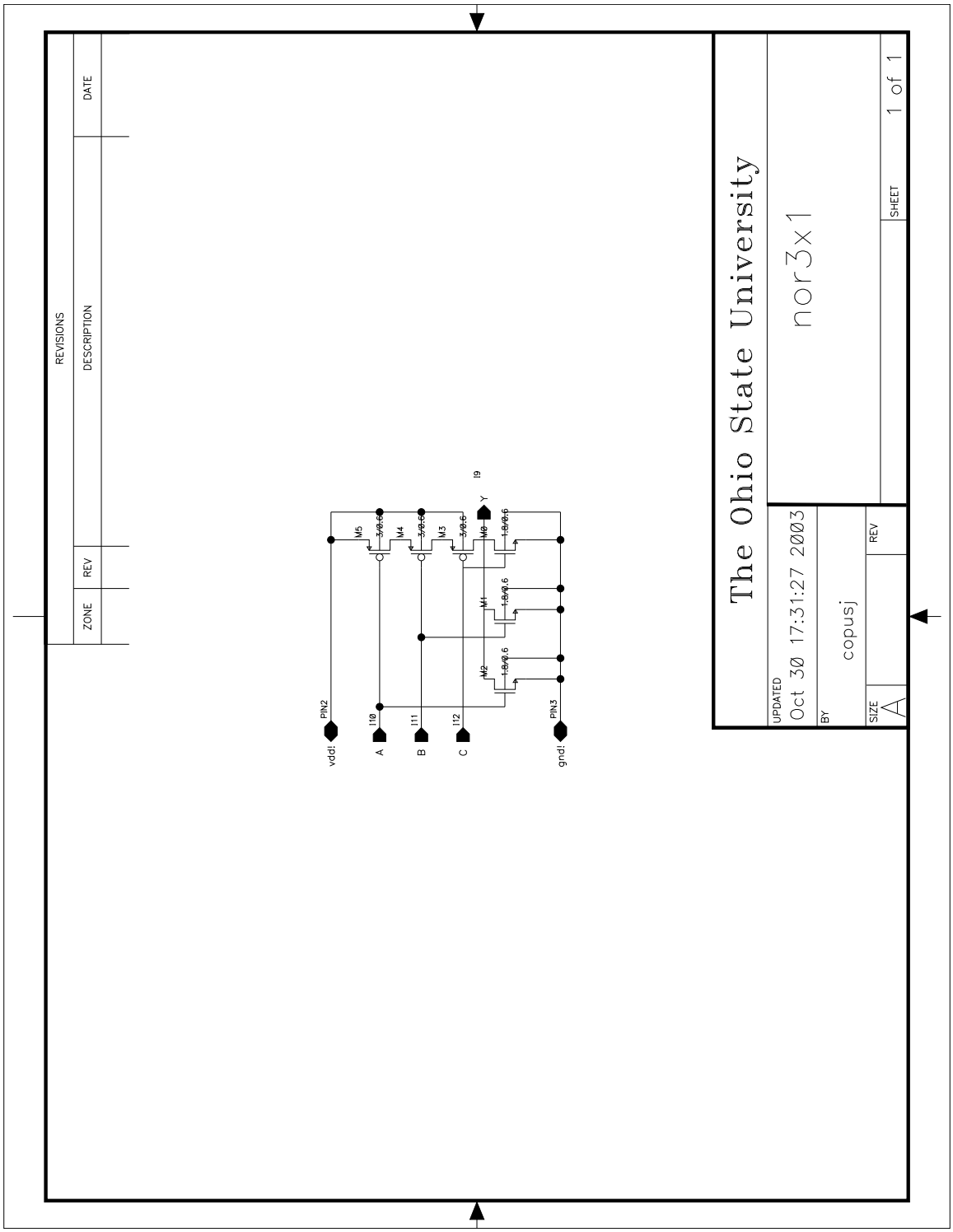

Figure M.55: Schematic of nor3x1

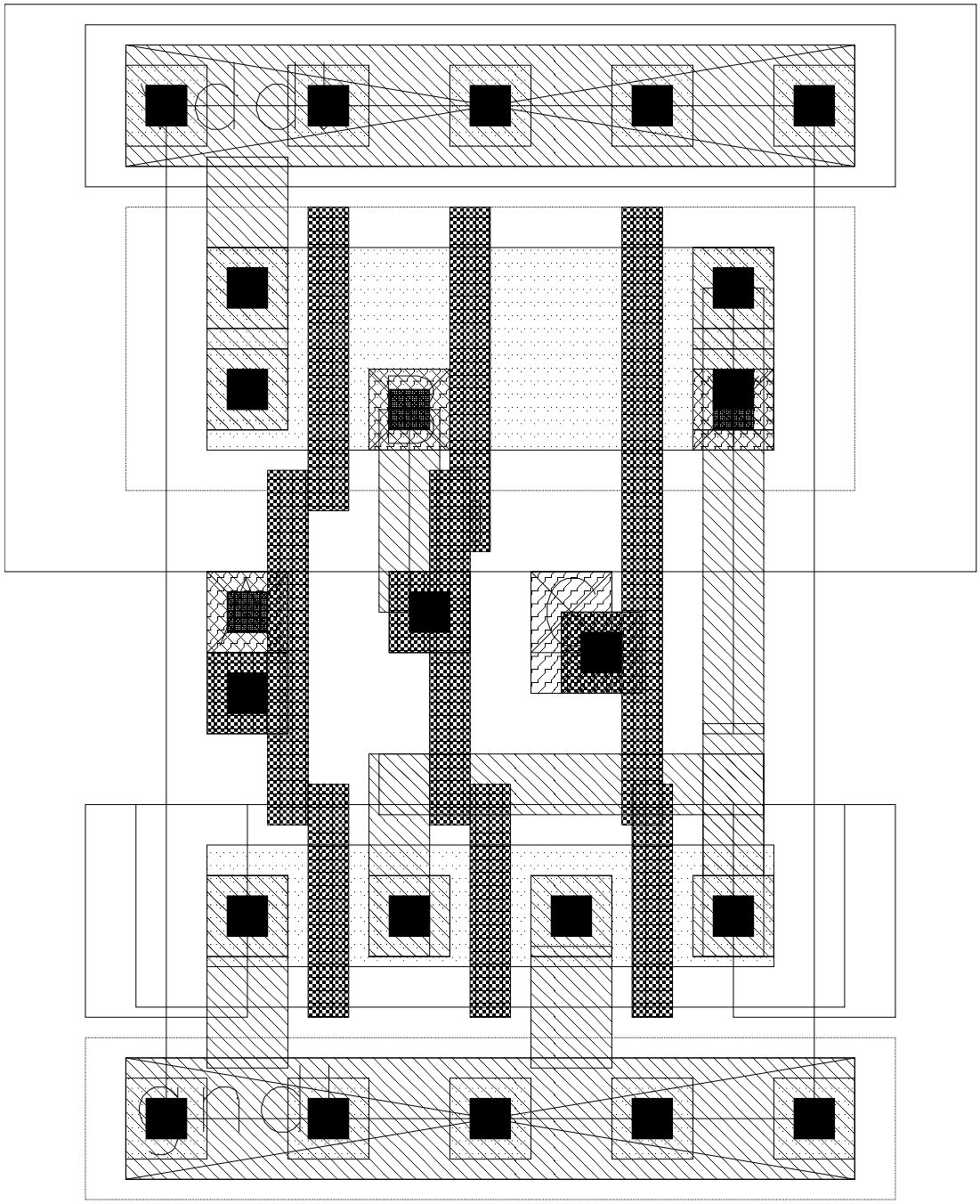

Figure M.56: Layout of nor3x1

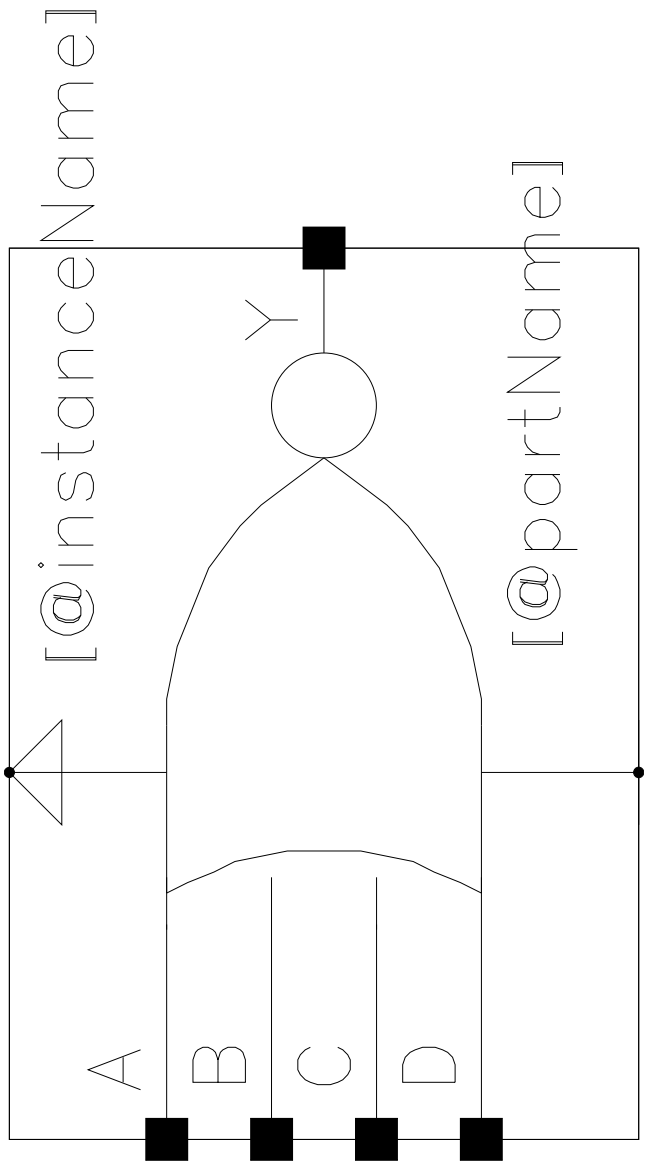

Figure M.57: Symbol of nor4x1

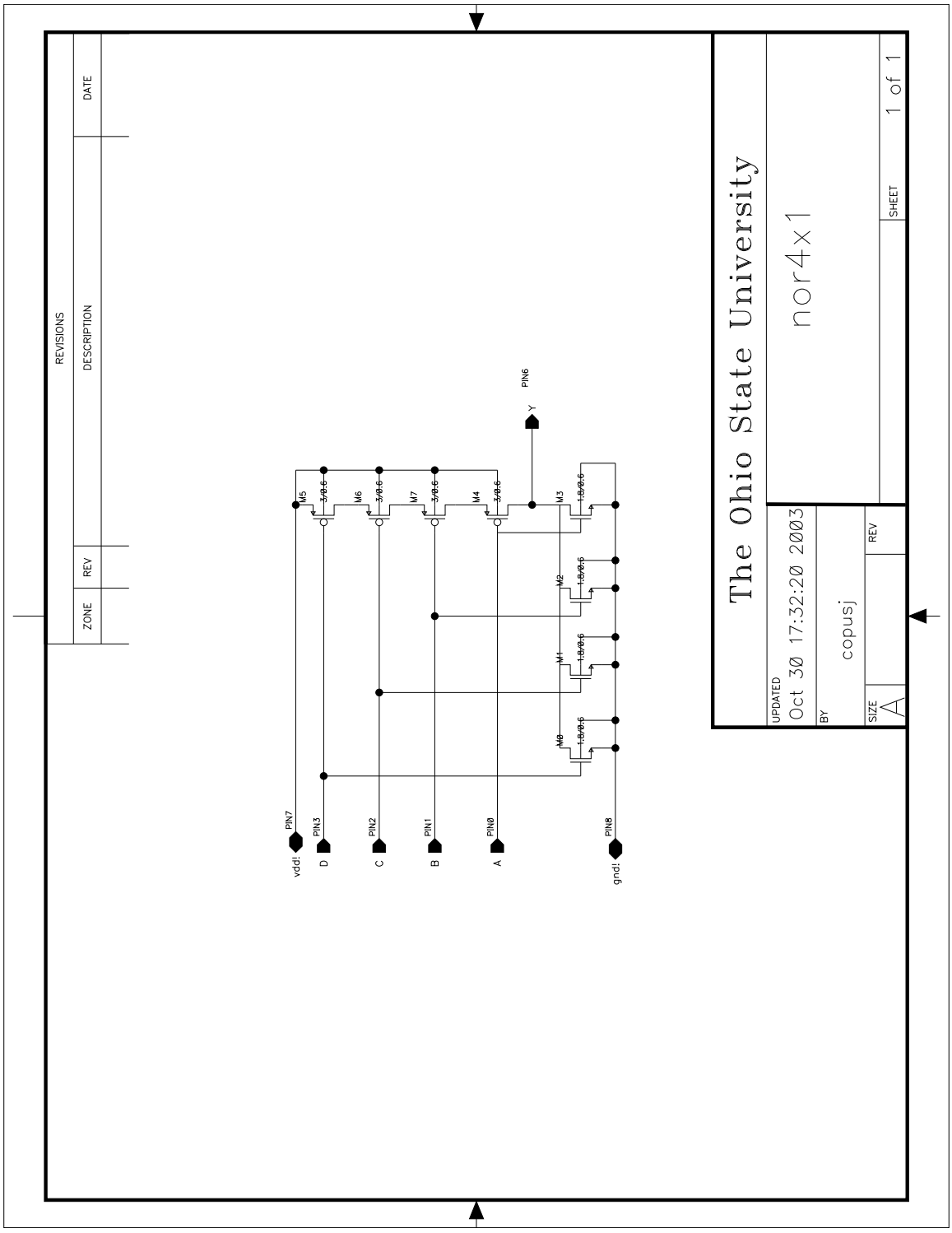

Figure M.58: Schematic of nor4x1

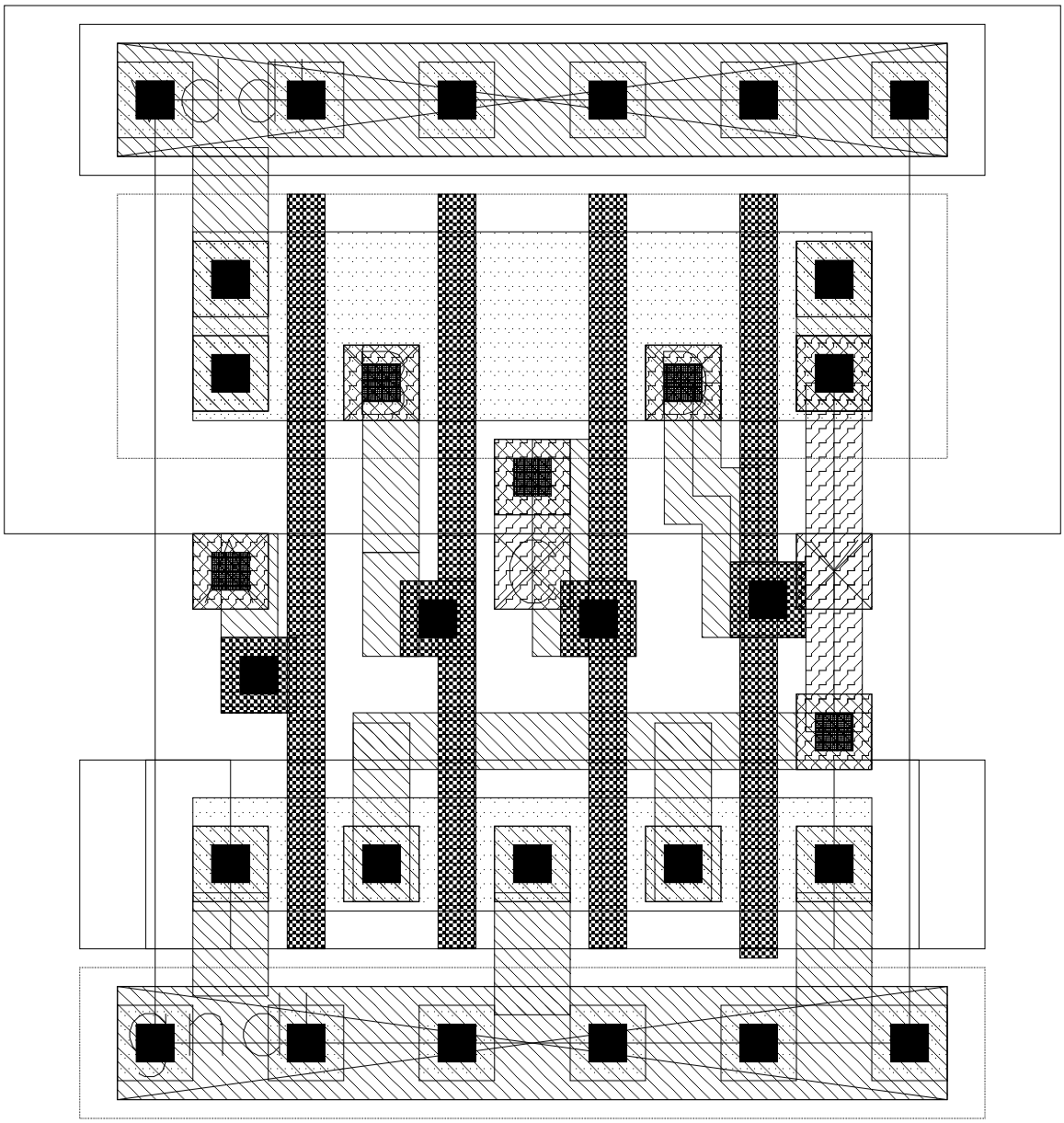

Figure M.59: Layout of nor4x1

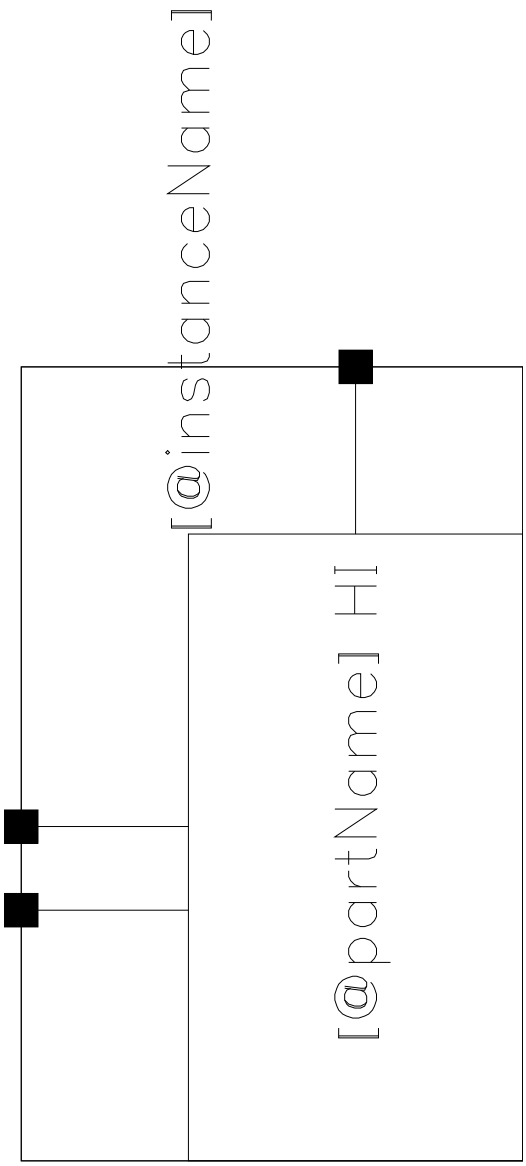

Figure M.60: Symbol of tiehigh

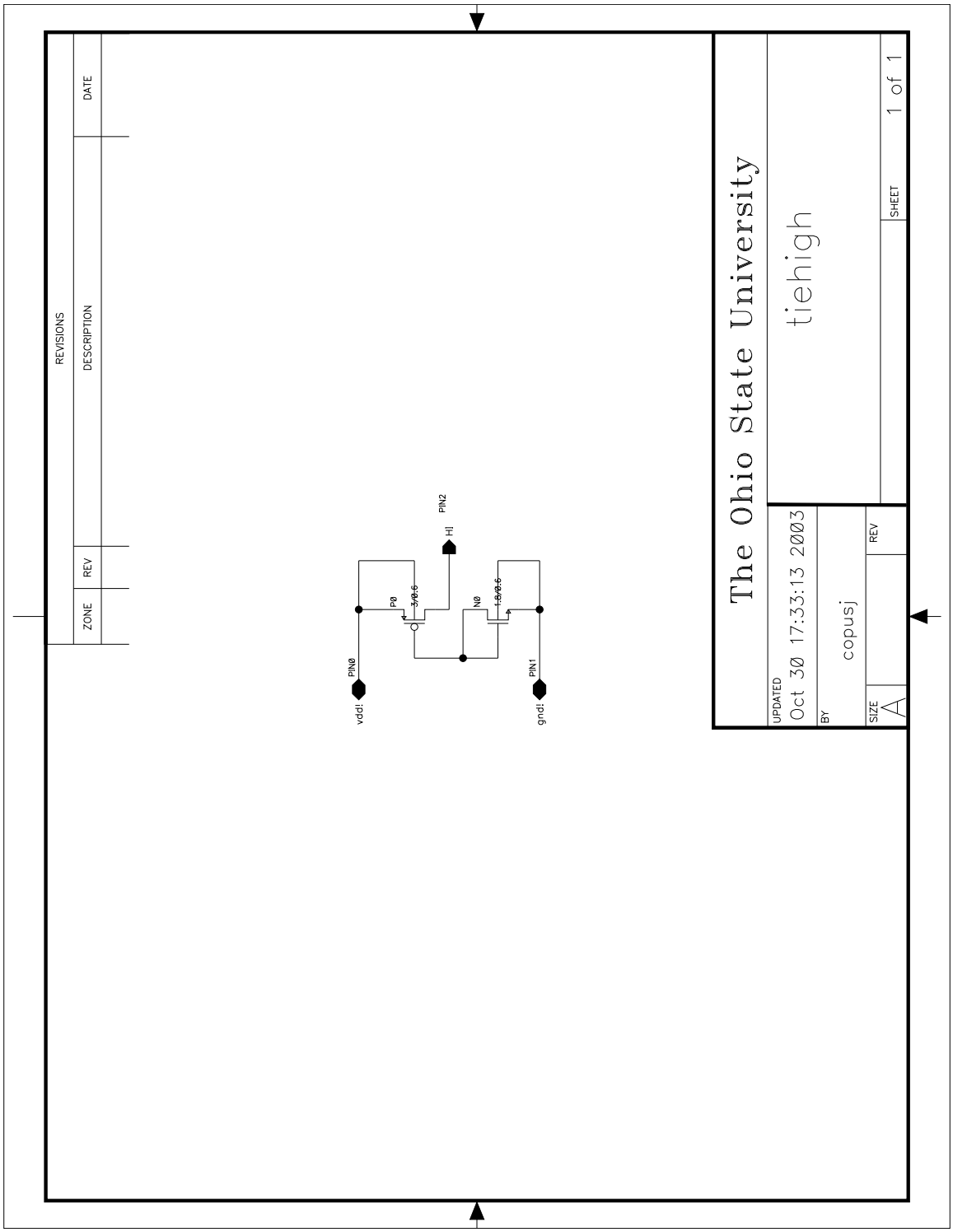

Figure M.61: Schematic of tiehigh

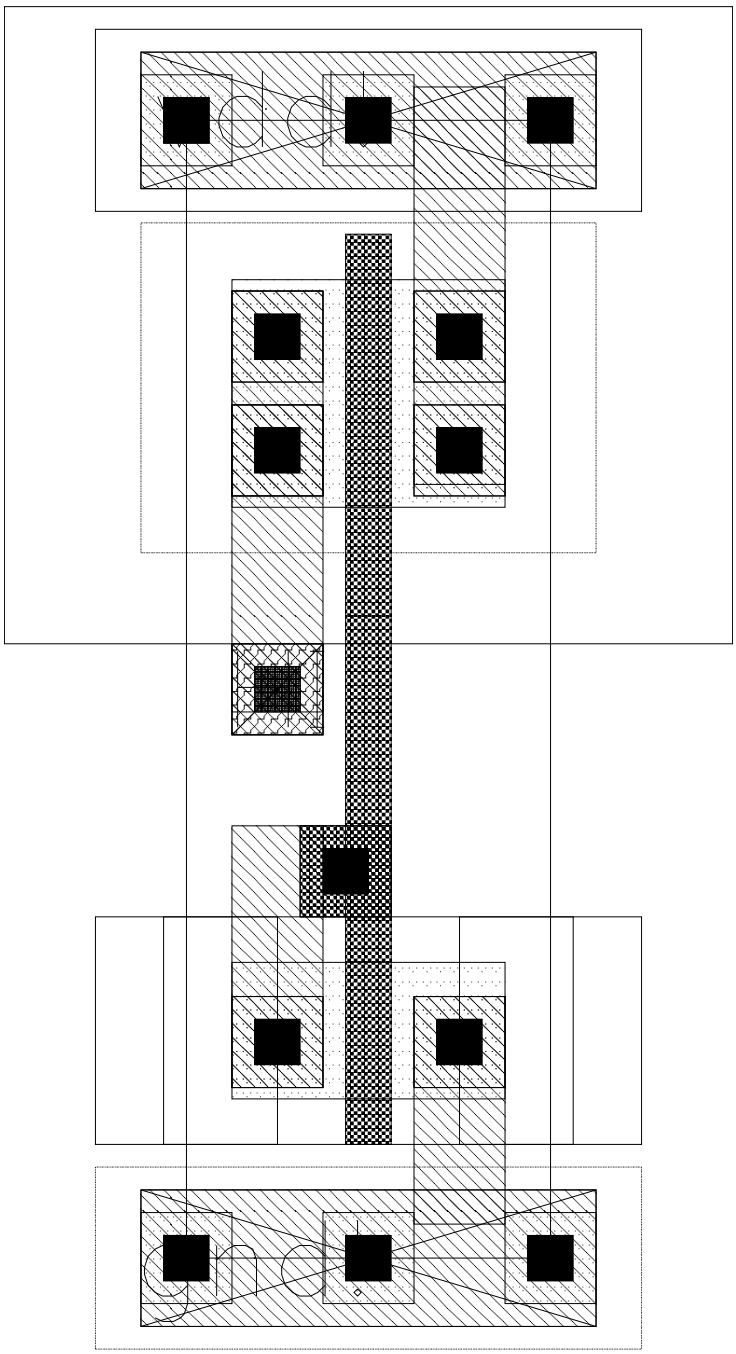

Figure M.62: Layout of tiehigh

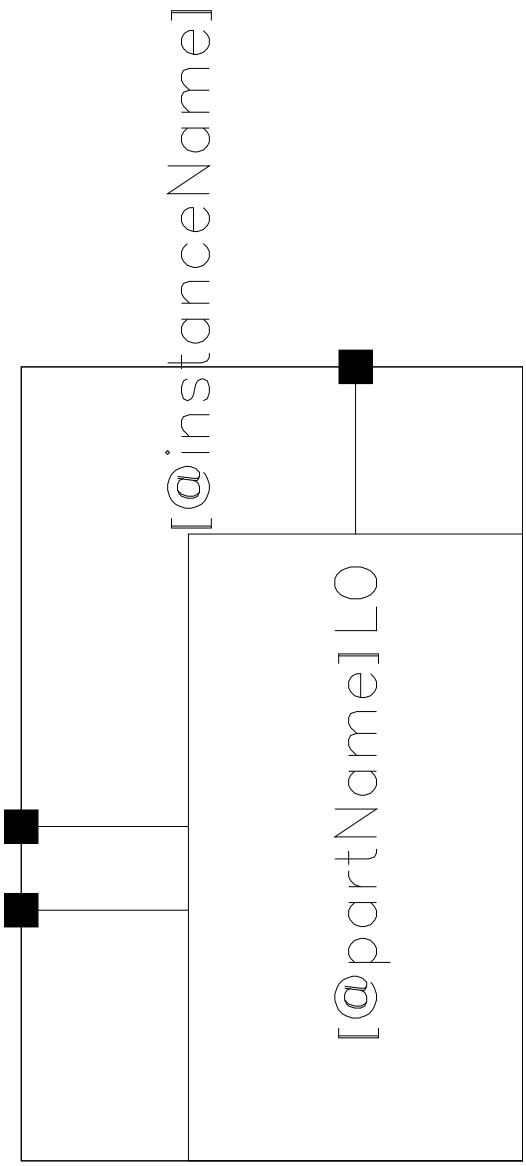

Figure M.63: Symbol of tielow

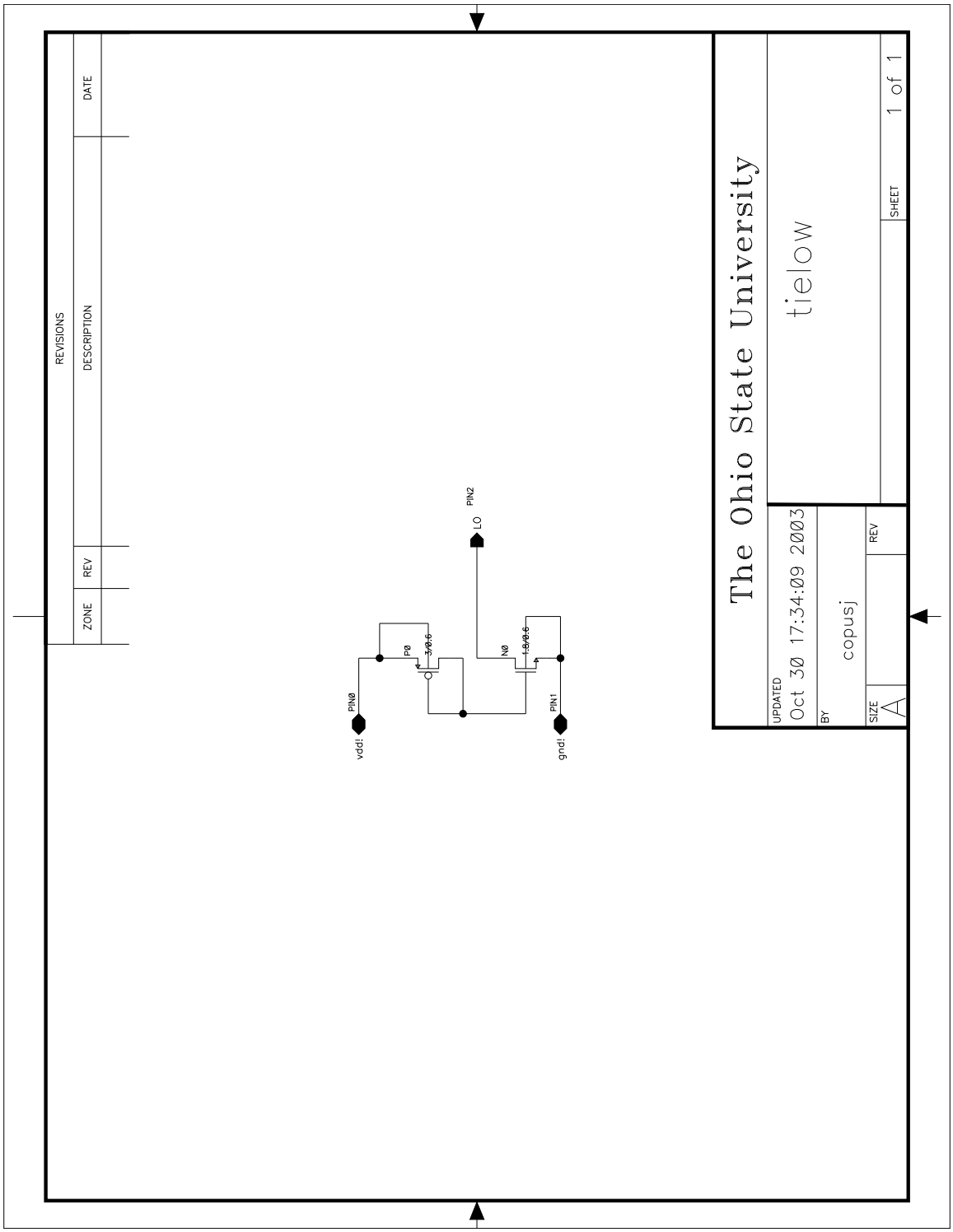

Figure M.64: Schematic of tielow

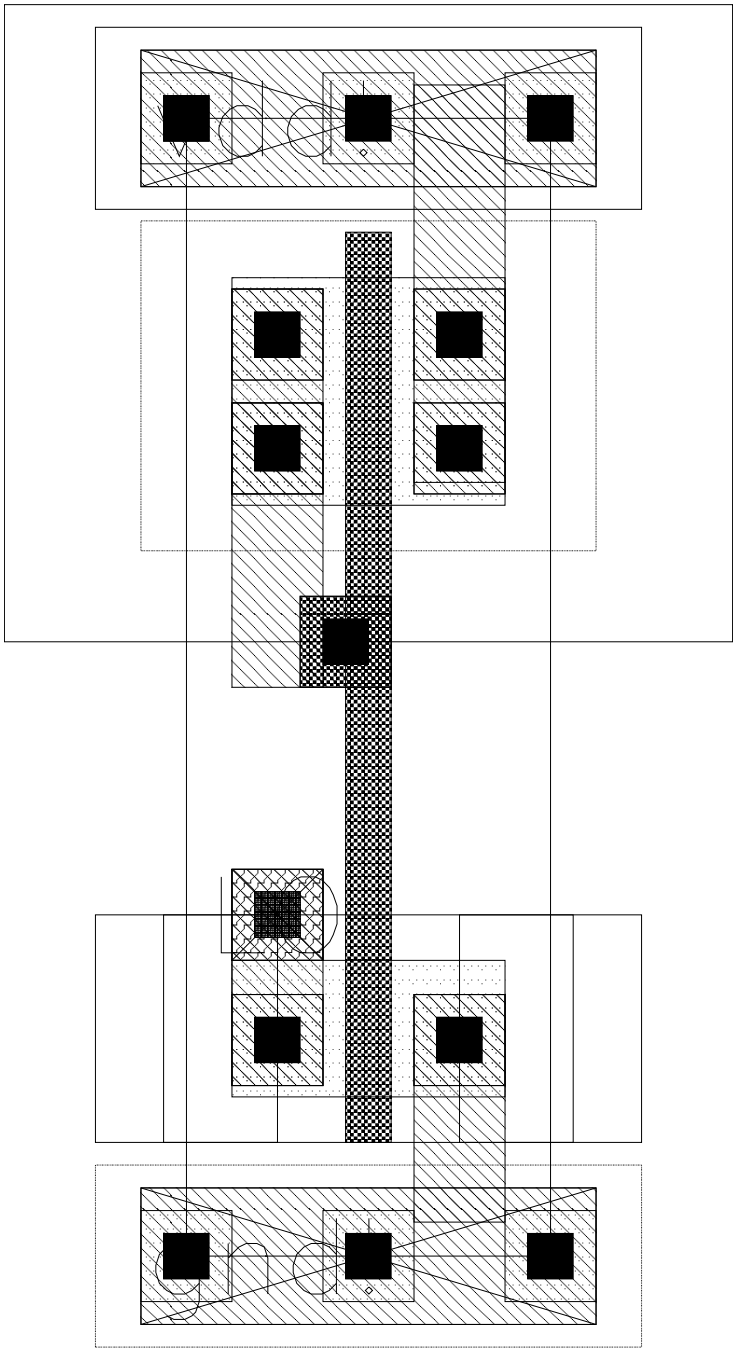

Figure M.65: Layout of tielow

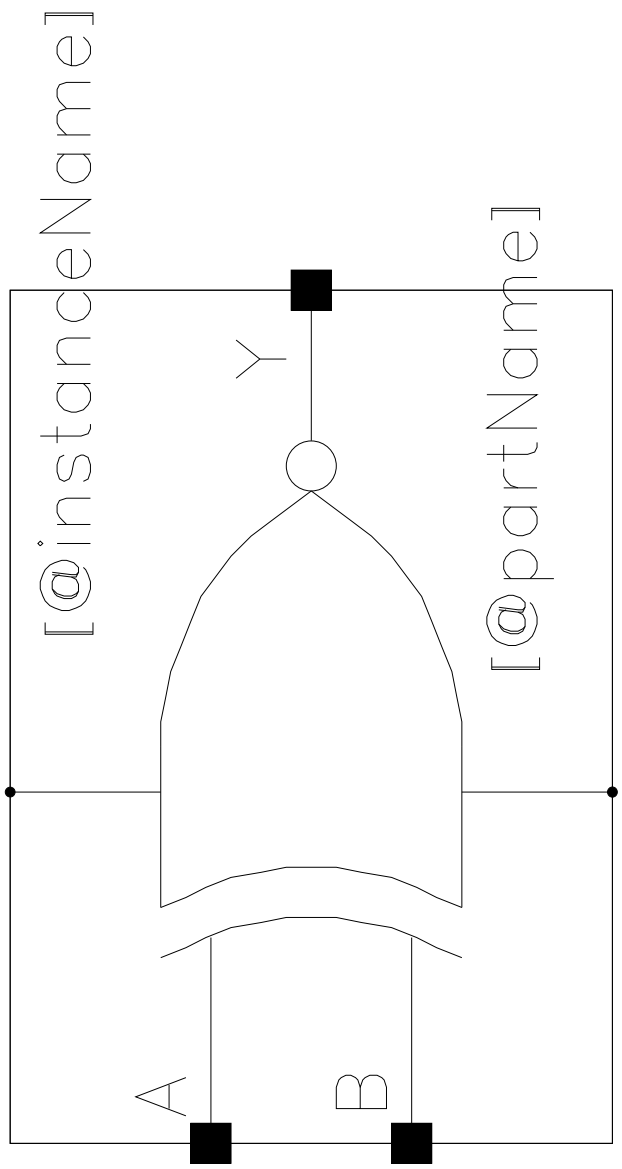

Figure M.66: Symbol of xnor2x1

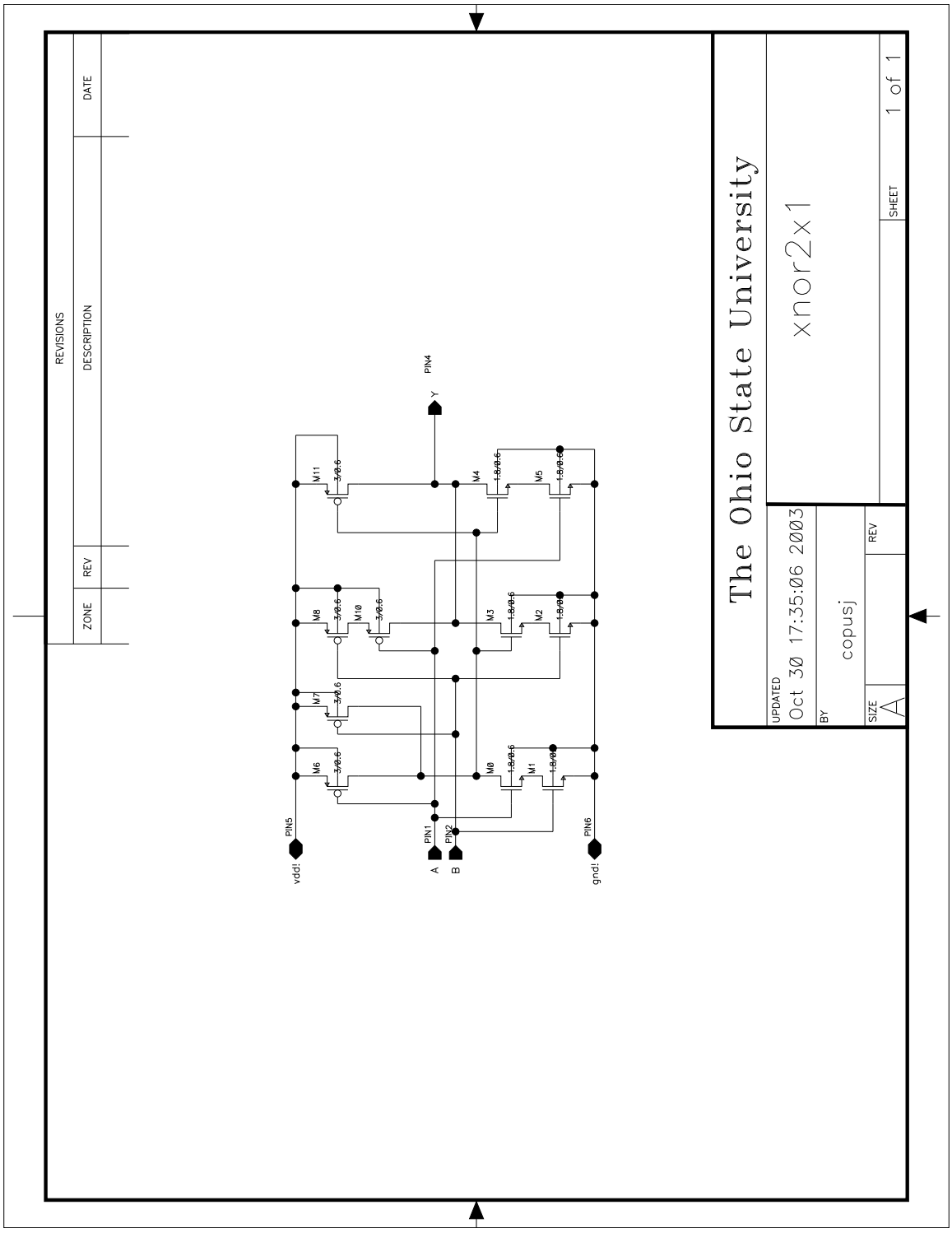

Figure M.67: Schematic of xnor2x1

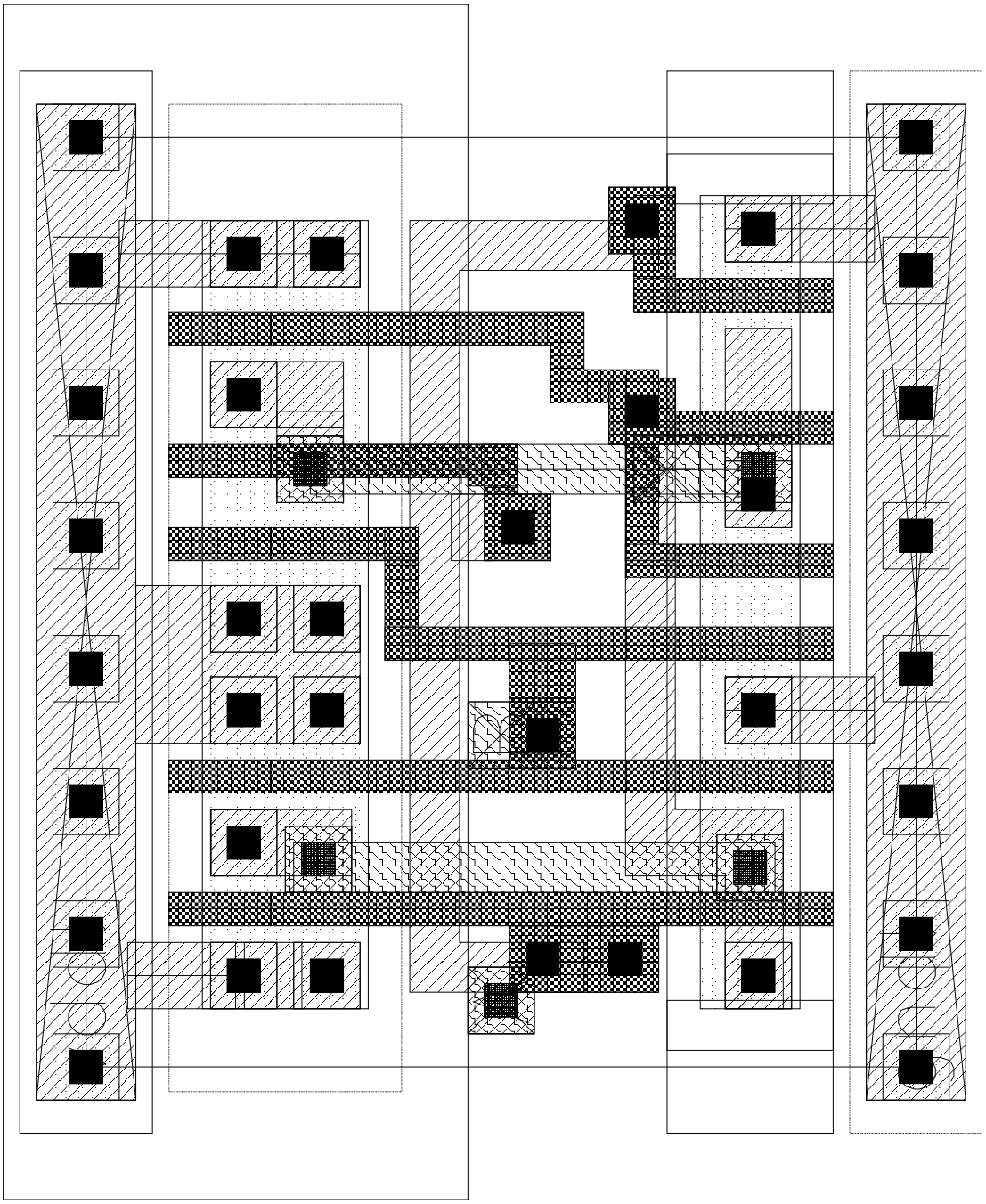

Figure M.68: Layout of xnor2x1

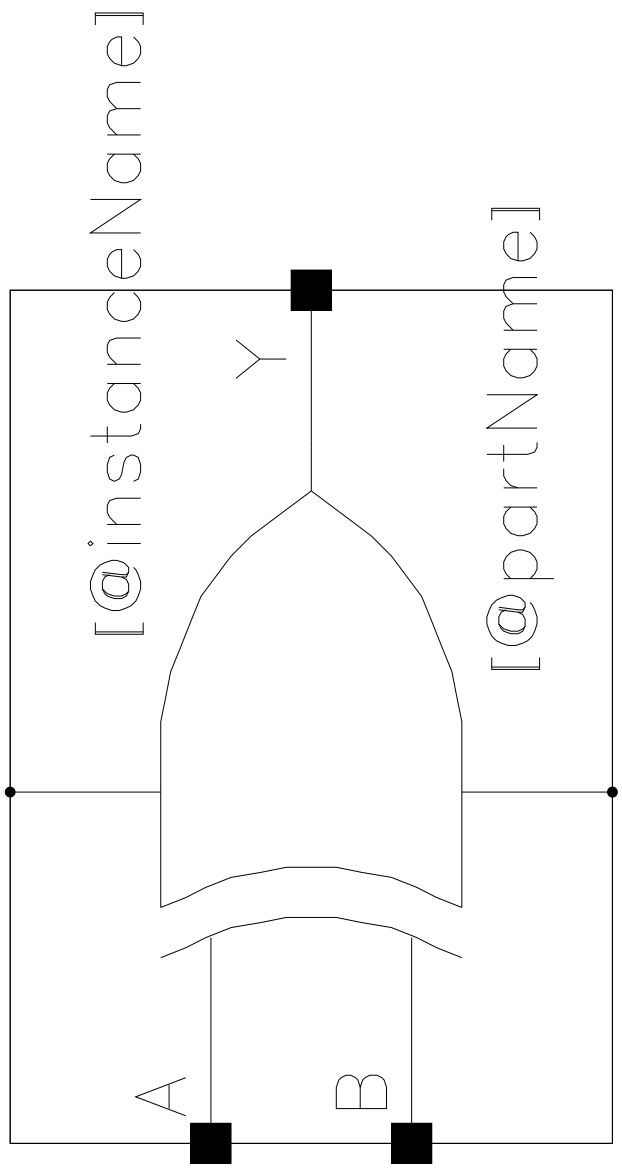

Figure M.69: Symbol of xor2x1

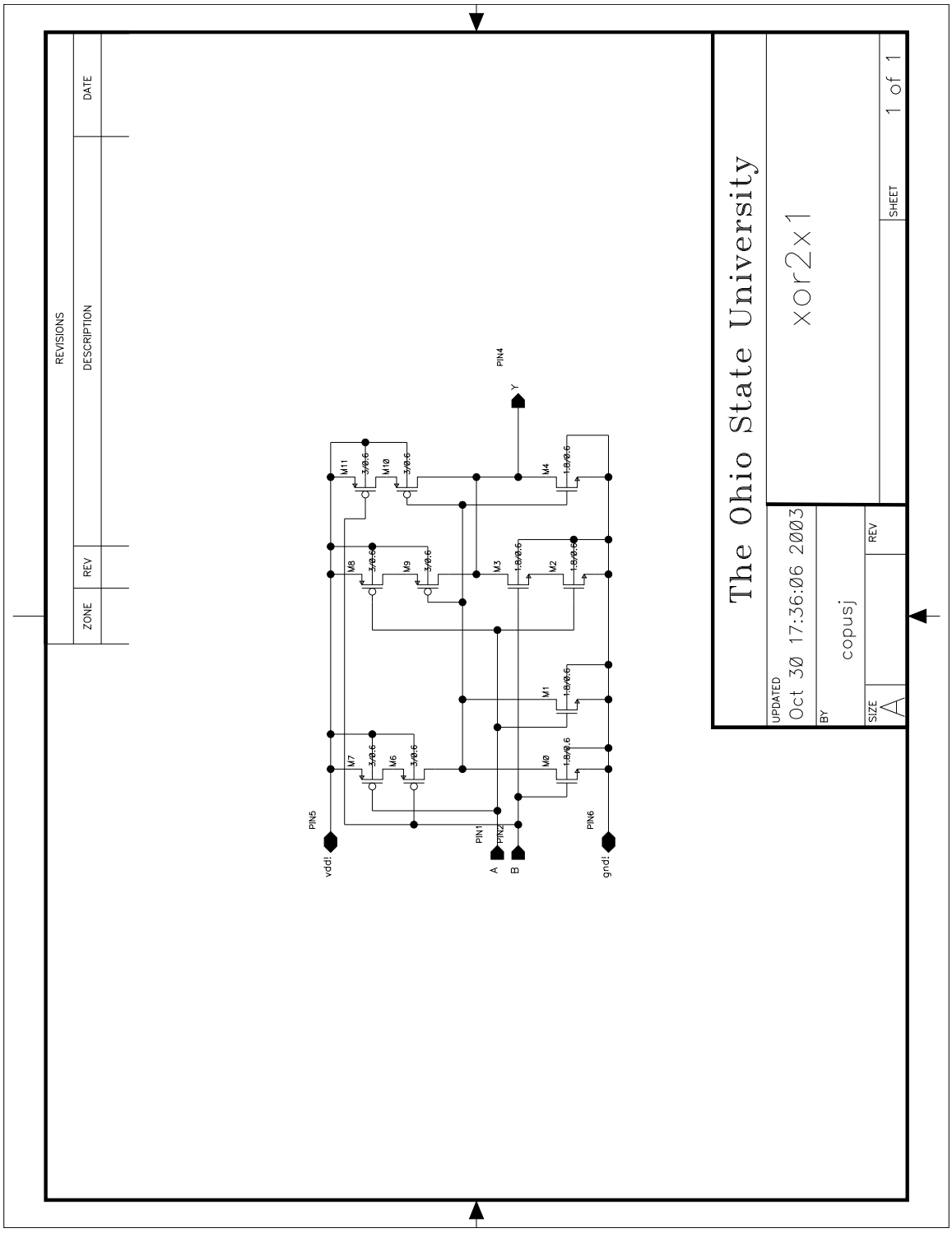

Figure M.70: Schematic of xor2x1

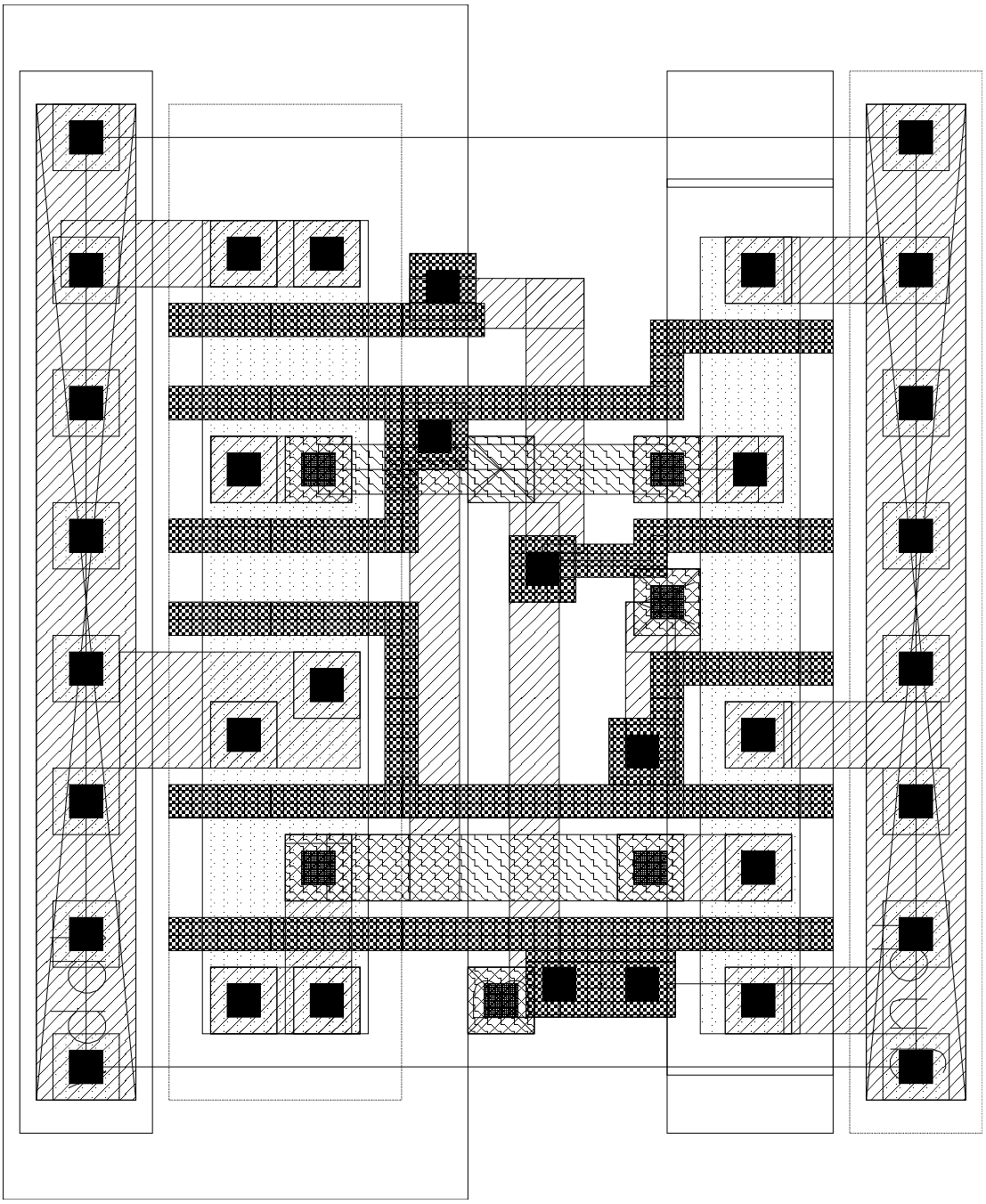

Figure M.71: Layout of xor2x1

APPENDIX N

Sample Layouts

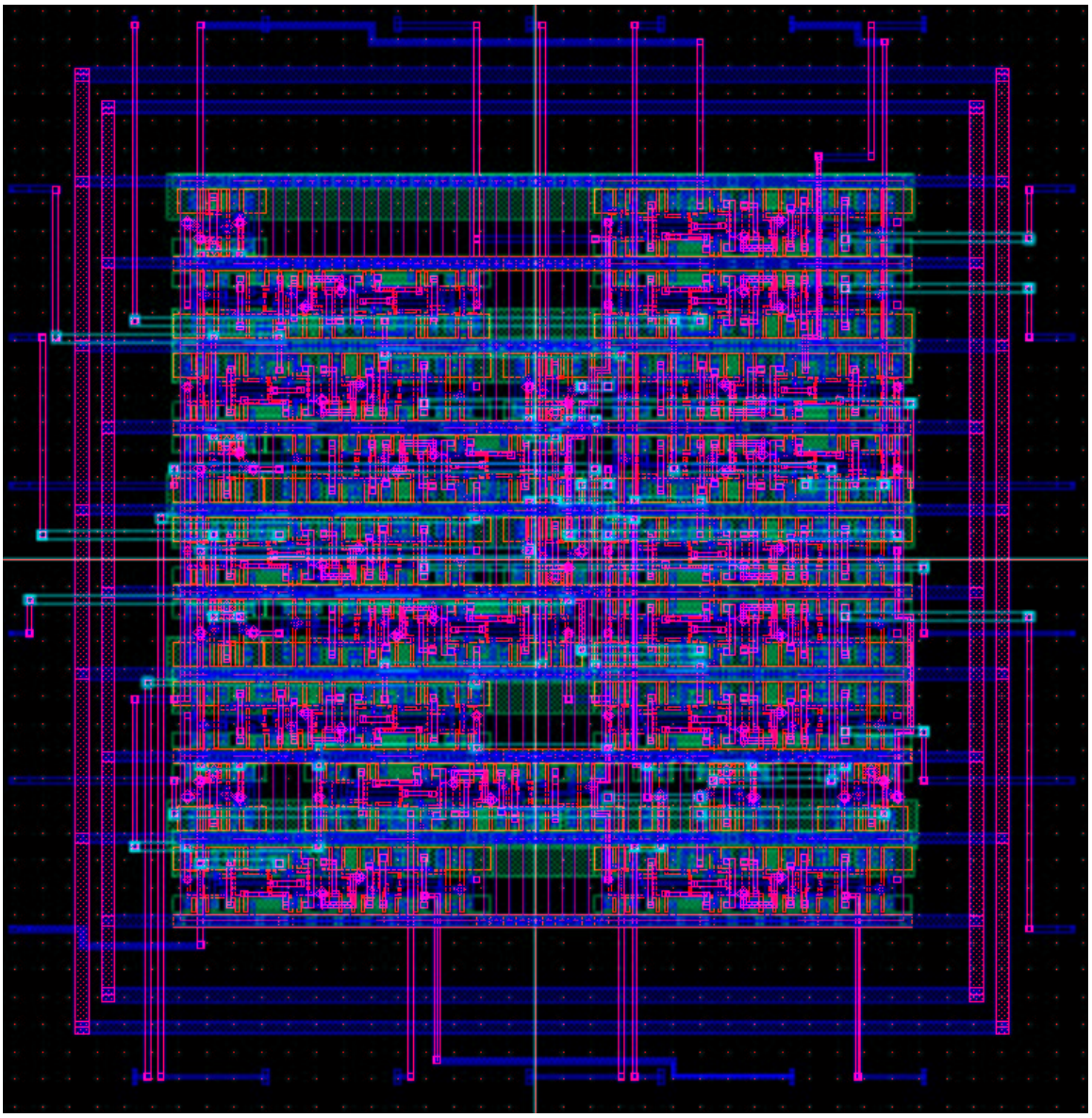

Figure N.1: Layout of Bidirectional Bus on OSU Digital Standard Cell Library.

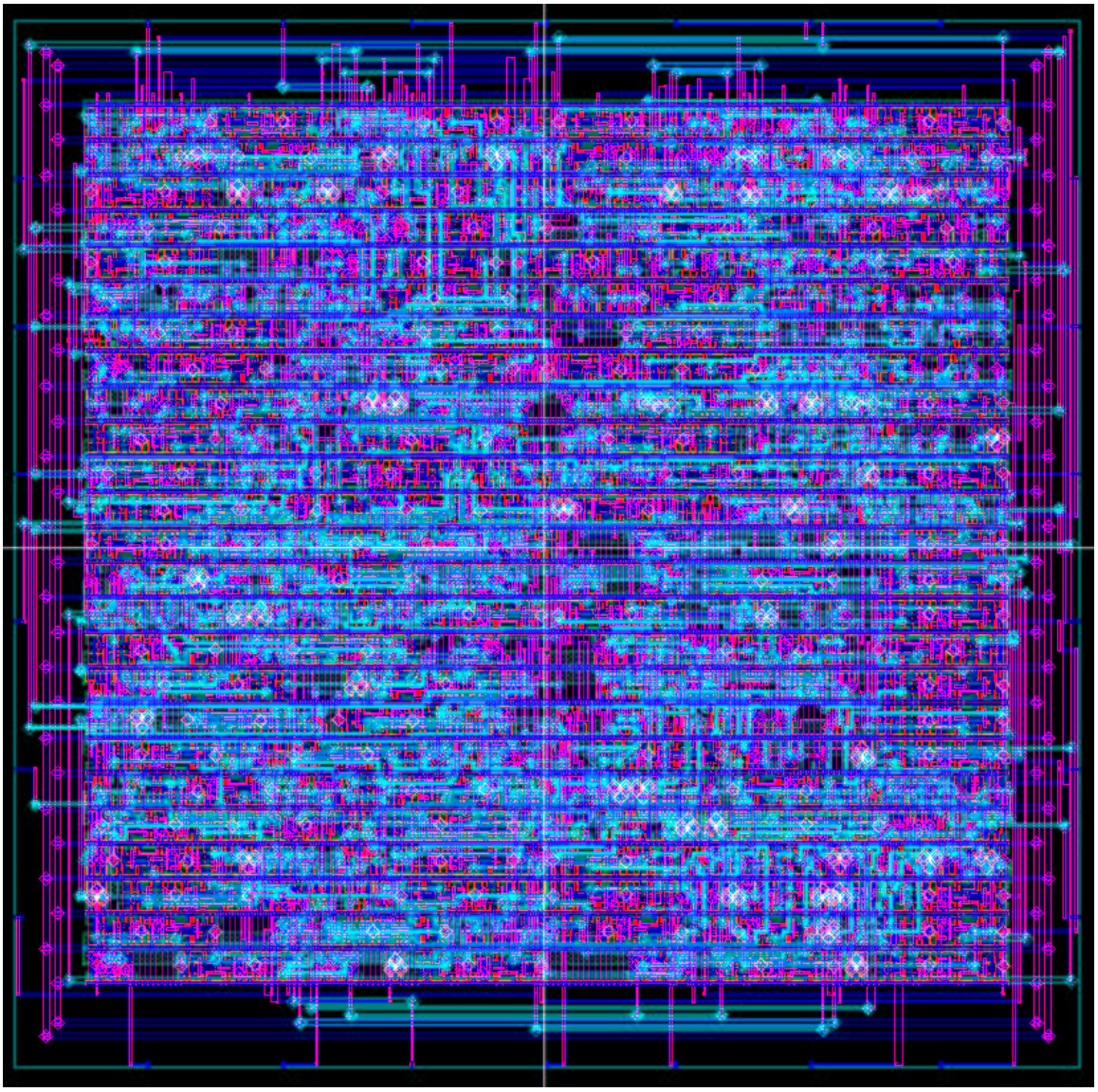

Figure N.2: Layout of MiniUART on OSU Digital Standard Cell Library.

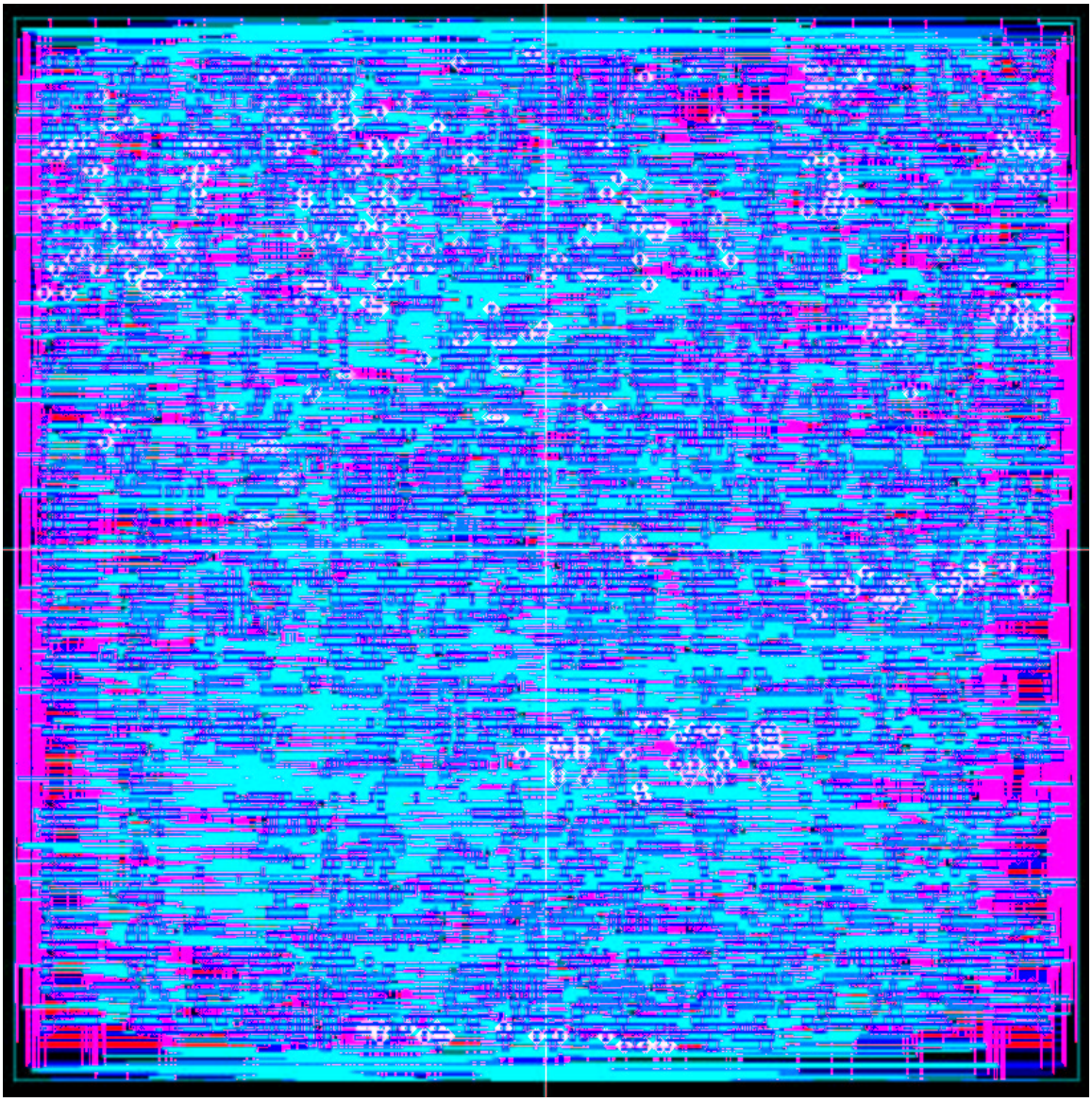

Figure N.3: Layout of AVR Microprocessor Core on OSU Digital Standard Cell Library.

BIBLIOGRAPHY

[1] J. Bhasker, *A VHDL Synthesis Primer*, Star Galaxy Publishing, 1996.

[2] "Verilog History," URL: `http://www.vol.webnexus.com/Sample/overview/history.html`.

[3] A. Cataldo and R. Wilson, "Embattled gate array players pull out an ace," URL: `http://www.commsdesign.com/news/tech_beat/OEG20020318S0011`.

[4] "Why Sea-of-Gates?," URL: `http://cas.et.tudelft.nl/~patrick/docs/wwman/section2.3.1.html`.

[5] T. Carter and K. Smith, "Path-programmable logic," in *Second Annual IEEE ASIC Seminar and Exhibit*, Sept. 1989, pp. P14–5/1–4.

[6] H. Eriksson and P. Larsson-Edefors, "Full-custom vs. standard-cell design flow - an adder case study," in *Proceedings of the ASP-DAC 2003 Asia and South Pacific Design Automation Conference*, Jan. 2003, pp. 507–510.

[7] M. Rose, M. Wiesel, D. Kirkpatrick, and N. Nettleton, "Dense, performance directed, auto place and route," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1988, pp. 11.1/1–11.1/4.

[8] M. Kontiala, A. Heinonen, and J. Nurmi, "Low-power methodology issues in digital circuit design," in *IEEE International Symposium on Circuits and Systems*, May 2002, pp. 1I–493–I–496.

[9] D. Chinnery and K.Keutzer, *Closing The Gap Between ASIC & Custom*, Kluwer Adademic Publishers, 2002.

[10] "The OpenCores Project," URL: `http://www.opencores.org`.

[11] "GNU general public license," URL: `http://www.gnu.org/licenses/gpl.txt`.

[12] "Project: AVR core," URL: `http://www.opencores.org/projects/avr_core`, May 2003.

[13] "Project: Serial uart," URL: http://www.opencores.org/projects/miniuart2, Jan. 2003.

[14] "MOSIS FAQ: Educational program," URL: http://www.mosis.org/Faqs/faq-education.html, May 2003.

[15] "Standard cell library for MOSIS SCMOS," URL: http://www.mosis.org/Technical/Designsupport/std-cell-library-scmos.html.

[16] J. Sulistyo, "Development of CMOS Standard Cell Library," URL: http://www.ee.vt.edu/~jgtront/ece5546/standard_cells.pdf.

[17] "Mississippi State University Standard Cell Library," URL: http://www.erc.msstate.edu/mpl/education/cadence/standard_cell/downloads.html.

[18] R. Tsui, A. Shenoy, J. Tampone, and S. Taylor, "A three-layer router for standard cell VLSI circuits," in *IEEE International Symposium on Circuits and Systems*, June 1988, pp. 1441–1444.

[19] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley, second edition, 1992.

[20] Cadence Design Systems Inc., *Technology File and Display Resource File User Guide*, Apr. 2001.

[21] Cadence Design Systems Inc., *Cadence Abstract Generator User Guide, Product Version 5.0*, June 2003.

[22] T. Haunsperger, "Tutorial help for cadence," URL: http://www.owlnet.rice.edu/~watkinst/tutorial/index.html, Rice University.

[23] "The Single UNIX Specification, Version 2: Nohup," URL: http://www.opengroup.org/onlinepubs/007908799/xcu/nohup.html, 1997.

[24] "VHDL Synthesis," URL: http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.syn/html/node3.html.

[25] "Standard cells for use with magic and cadence/synopsis," URL: http://www.ece.iit.edu/~cad/scells/.

[26] "Ami-0.6 standard-cell library for cadence," URL: http://vlsi1.engr.utk.edu/ece/bouldin_courses/ut-lp-ami06.html.

[27] J. Grad and J.E. Stine, "A standard cell library for student projects," in *International Conference on Microelectronic Systems Education*. IEEE Computer Society, 2003, pp. 98–99.

[28] D. Bouldin, "Microelectronic system courses," URL: http://vlsi1.engr.utk.edu/ece/bouldin_courses/.

[29] "VHDL: Bidirectional bus," URL: http://www.altera.com/support/examples/vhdl/v_bidir.html.

[30] ATMEL, "8-bit AVR microcontroller with 128K in-system programmable flash: ATmega103, ATmega103L," Datasheet.