

# Design and Simulation of Cryogenic Test Circuits

by

Kyle Vickers Owen

A thesis submitted to the Graduate Faculty of  
Auburn University  
in partial fulfillment of the  
requirements for the Degree of  
Master of Science

Auburn, Alabama  
August 6, 2016

Keywords: electronics, integrated circuit, ring oscillator, multiply-accumulate, SRAM

Copyright 2016 by Kyle Vickers Owen

Approved by

Michael C. Hamilton, Chair, Associate Professor of Electrical and Computer Engineering  
Mark L. Adams, Assistant Professor of Electrical and Computer Engineering  
Victor P. Nelson, Professor of Electrical and Computer Engineering  
Charles D. Ellis, Research Associate

## Abstract

New technologies are being introduced in anticipation of the end of Moore's Law, such as quantum computing, in hopes that the speed gains from the different topologies will offset the increase in size of the devices. Fast digital logic families have also evolved, including reciprocal quantum logic, rapid single flux quantum logic, and several more, that operate using quantum effects while being founded on conventional digital logic. One main issue with these logic families, and quantum computing for that matter, is their reliance on cryogenic temperatures to operate. Though these logic families are viable technologies, they are not very efficient in terms of data throughput unless the bandwidth in and out of the cryogenic dewar is sufficiently high. This requires high speed level shifters and generally protocol translators, such as serializer/deserializers, in order to interface to current generation digital logic. Thus, there is much interest in the operation of currently available low power CMOS circuits at cryogenic and near-cryogenic temperatures. Test circuits in a variety of current generation CMOS processes have been designed and simulated, in hopes to show functioning circuits at cryogenic temperatures. Test circuits range from simple ring oscillators to static and dynamic memory structures to more complicated synthesized ALU-like circuits, such as a multiply-accumulate unit.

## Acknowledgments

I would like to thank Dr. Michael Hamilton for his encouragement and guidance as my professor, advisor, and mentor during the past two years.

I would also like to thank Dr. Charles Ellis for his several years of mentoring me in the microfabrication lab and helping me develop a career path that is both rewarding and fun.

I would like to thank Drs. Victor Nelson and Mark Adams for their review of this thesis.

I would like to thank the individuals at Lincoln Laboratory, especially Drs. Peter Grossmann and Phillip Bailey, for their insight into the intricacies of Cadence Virtuoso, Encounter, and other related tools.

I would like to thank my undergraduate research assistants, including SueAnne Griffith, Hunter Burch, Andrew McCrabb, Thomas Seitz, Hayden Burch, Shane Williams, and Robert Christiansen. Their efforts in the research group have been truly invaluable.

I would like to also thank Dr. Thomas Burch and Patti Burch, for helping me stay motivated, and for being my family away from home.

Finally, I am very thankful for my own family, Dr. John Owen, Nancy Owen, and Evan Owen, for their words of encouragement throughout the years, particularly in pursuing higher education.

This thesis is dedicated to my grandmother, Ann Jackson, who has always believed in me, but may not understand most of this thesis.

## Table of Contents

Abstract . . . . .	ii
Acknowledgments . . . . .	iii
List of Figures . . . . .	vii
List of Tables . . . . .	ix
List of Abbreviations . . . . .	x
1 Introduction . . . . .	1
2 Ring Oscillators . . . . .	3
2.1 Background . . . . .	3
2.2 32 nm Ring Oscillators . . . . .	5
2.2.1 Schematic . . . . .	6
2.2.2 Layout . . . . .	7
2.2.3 DRC and LVS . . . . .	9
2.2.4 Parasitic Extraction . . . . .	10
2.3 90 nm Ring Oscillators . . . . .	12
2.3.1 Schematic . . . . .	12
2.3.2 Layout . . . . .	14
2.3.3 DRC and LVS . . . . .	18
2.3.4 Parasitic Extraction . . . . .	18
3 32-bit Multiply-Accumulate Unit . . . . .	20
3.1 Background . . . . .	20
3.2 Hardware Description Language . . . . .	20
3.3 Simulation . . . . .	20
3.4 Layout . . . . .	21

3.5	DRC and LVS . . . . .	23
4	Static Random Access Memory . . . . .	24
4.1	Background . . . . .	24
4.2	Schematic . . . . .	25
4.3	Simulation . . . . .	29
4.4	Layout . . . . .	32
4.5	DRC and LVS . . . . .	33
5	Conclusion and Future Work . . . . .	34
5.1	Conclusion . . . . .	34
5.2	Future Work . . . . .	34
	Bibliography . . . . .	35
	Appendices . . . . .	36
A	MATLAB Code . . . . .	37
A.1	SRAM Static Noise Margin Analysis . . . . .	37
A.1.1	Parser . . . . .	37
A.1.2	Grapher . . . . .	39
A.1.3	Analyzer . . . . .	40
A.1.4	Inscribed Square - Read . . . . .	41
A.1.5	Inscribed Square - Write . . . . .	42
B	IC Manage Tutorial . . . . .	44
B.1	Adding a new user . . . . .	45
B.2	Adding a new PDK project . . . . .	48
B.3	Add variant to existing project . . . . .	51
B.4	Add libraries to existing variant . . . . .	55
B.5	Adding unmanaged libraries . . . . .	59
B.6	Adding a new workspace . . . . .	62
B.7	Adding library properties . . . . .	65

B.8	Managed vs. unmanaged libraries . . . . .	68
B.9	Editing a view . . . . .	70
B.10	Canceling a checkout . . . . .	79
B.11	Reverting to an older version . . . . .	82
C	Plotting Schematics Tutorial . . . . .	87
C.1	.cdsplotinit Contents . . . . .	87
C.2	schematic_beautify.sed Contents . . . . .	88
D	ADE Tutorial . . . . .	92
E	LVS Tutorial . . . . .	114
E.1	lvs.runset Contents . . . . .	121
F	DRC Tutorial . . . . .	123
F.1	cell_noden.runset Contents . . . . .	130
F.2	chip_den.runset Partial Contents . . . . .	132
G	PEX Tutorial . . . . .	133
G.1	90 nm PEX Setup . . . . .	134
G.2	32 nm PEX Setup . . . . .	139
G.3	pex.runset Contents . . . . .	145
H	Encounter Tutorial . . . . .	147
H.1	Behavioral Verilog . . . . .	147
H.1.1	mac32_dual_wrapper.v Contents . . . . .	147
H.2	Tcl Scripts . . . . .	149
H.2.1	rtl_rvt.tcl Contents . . . . .	149
H.2.2	top_level.tcl Contents . . . . .	149
H.2.3	pr.tcl Contents . . . . .	150

## List of Figures

2.1	One of sixteen ROs in the 32 nm core. . . . .	6
2.2	The completed 16-RO chip. . . . .	7
2.3	The completed 16-RO core. . . . .	8
2.4	A single RO with schematic overlay. . . . .	9
2.5	A single RO extracted from the core, used for PEX. . . . .	10
2.6	The measured RO frequency vs. PEX frequency from simulation at RT. . . . .	11
2.7	The RO schematic as used for the 90 nm chip. . . . .	12
2.8	The divider schematic as used for the 90 nm chip. . . . .	12
2.9	The completed 90 nm RO chip. . . . .	14
2.10	One of two full speed ROs on the 90 nm chip. . . . .	15
2.11	One of two divided ROs on the 90 nm chip. . . . .	16
2.12	“Plaid” capacitor used for bypass across power rails. . . . .	17
2.13	The 90 nm layout used for PEX. . . . .	18
2.14	Expected frequencies of the 90 nm RO chip at 4.2K. . . . .	19
3.1	The completed 32 nm 32-bit MAC chip. . . . .	21

3.2	The completed 32 nm 32-bit MAC core. . . . .	22
4.1	The 14 nm butterfly test schematic. . . . .	25
4.2	The 14 nm 3, 2, 2 bit cell schematic. . . . .	26
4.3	The 14 nm sense amplifier schematic. . . . .	27
4.4	The 14 nm top level SRAM circuits schematic. . . . .	28
4.5	The 14 nm SRAM simulation results showing successful reads and writes. . . . .	29
4.6	Static noise margin for “322” bit cell. . . . .	30
4.7	Static noise margin for “332” bit cell. . . . .	30
4.8	Static noise margin for “542” bit cell. . . . .	31
4.9	The completed 14 nm SRAM test chip. . . . .	32
4.10	The completed 14 nm SRAM test circuits. . . . .	33



List of Tables

4.1 Read and Write SNM Summary (0.8 V) . . . . . 31

## List of Abbreviations

ADE Analog Design Environment

BEOL back end of line

CDL circuit design language

CDS Cadence Design Systems, Inc.

CSV comma-separated values

DRAM dynamic random access memory

DRC design rule checking

FD fully depleted

FEOL front end of line

GDSII graphical database system, version II

HVT high voltage threshold

IC integrated circuit

LFSR linear feedback shift register

LVS layout versus schematic

LVT low voltage threshold

MAC multiply-accumulate unit

PCell parameterized cell

PD partially depleted

PDK process design kit

PEX parasitic extraction

RO ring oscillator

RQL reciprocal quantum logic

RSFQ rapid single flux quantum

RT room temperature

RTL register-transfer level

RVT regular voltage threshold

SerDes serializer/deserializer

SNM static noise margin

SOI silicon on insulator

SRAM static random access memory

SVT super-high voltage threshold

## Chapter 1

### Introduction

The state of current-generation electronics is truly staggering. Fifty years ago, the mere thought of having the computational power of every computer in the world combined could not be fathomed. Computers ran at a maximum of 1 MHz, with little hope of parallel processing. Some computers required massive amounts of power, but that was not near as much of a concern as it is nowadays, as many of today's computers are intended to be battery powered. As technology proceeded to shrink, computers became faster and and required less power to the point of even being portable, or perhaps luggable. With the silicon die shrinking further as transistor sizes decrease, cooling the die becomes quite problematic. In some cases, the power dissipated per unit area exceeds  $1000 \text{ W cm}^{-2}$ ; more power per unit area than a rocket nozzle. [1] For advanced computing to continue at such a pace, other technologies other than CMOS may need to be considered.

Quantum computing, for instance, relies on a probabilistic approach, versus conventional digital computers which can perform one mathematical operation per CPU at a time. This probabilistic nature requires new algorithms, and even though the processing of some quanta of data is near instantaneous, it requires numerous cycles in order to have some level of certainty of the right answer. Quantum computing relies on storing information in a very low energy state, thus requiring superconducting electronics. To date, no room temperature superconductors exist, and the superconductors that work well for such applications require temperatures below 20 K.

Fortunately, computer scientists can get some of the speed benefits without having to understand and develop completely new algorithms for computing with qubits. Technologies such as RQL, RSFQ, and other quantum-based digital logic families offer very high speeds

but still operate digitally, using ordinary bitwise operations. Thus, conventional computers can be built, even if they don't use a single silicon transistor. Such technologies rely on superconducting lines and Josephson junctions, which still require cryogenic environments to operate.

Large cryogenic cooling systems are very expensive, and cannot handle significant thermal loads. For example, a fairly standard cryocooler, the SHI RP-082B2, has a heat capacity of 1 W at 4.2 K. [2] This limits the types of devices or the number of devices in the system. If a superconducting device begins to dissipate too much power, even if it's not too much for the cooling system, the temperature of the die may exceed the critical temperature of the superconducting lines. This would result in a chip that is no longer superconducting. Thus, keeping power dissipation very low is a necessity.

The speed of a cryogenically cooled superconducting computer is limited by the data throughput to and from the computer. High speed coaxial cables add to the thermal load of the system, so these are often kept to a minimum. The pulses used by RQL and RSFQ are extremely small, both in amplitude and time, so level converters must be used. Thus, there is a need for conventional CMOS devices that can operate at relatively low temperatures, perhaps slightly warmer than the rest of the superconducting electronics.

Beyond simple level converters, having other (warmer) CPUs near the superconducting computer would allow for added processing capability by moving more data back and forth between the high speed superconducting computer and the outside world. Superconducting memories have a relatively low density, meaning that the data would need to be swapped fairly often to a slightly slower but larger memory; hence, the need for high density and low power memory structures within the cooling system.

This thesis will cover such test circuits as ring oscillators, a multiply-accumulate unit, and basic SRAM structures. Design information and simulation results are covered. A large appendix provides more information on how these circuits were generated.

## Chapter 2

### Ring Oscillators

#### 2.1 Background

Ring oscillators are a useful and simple devices for use as “canary” circuits, those that can alert the foundry to problems and variations during the run, as well as providing suitable test structures for characterizing new processes. They are able to provide data, depending on construction, for gate capacitance, overlap capacitance, resistance, and other MOSFET C-V characteristics and general parasitics. Since frequency and power measurements are relatively straightforward, ring oscillators are a suitable choice for designers. The following equations describe other common extracted parameters from ROs. [3]

The number of stages is an odd number, and thus, is represented by  $(2\alpha + 1)$ , where  $\alpha$  is an integer.  $\alpha$  should be sufficiently high as to average out noise and variations in the manufacturing process.

$$(2\alpha + 1) = \text{number of stages}$$

The stage delay,  $\tau_p$ , of an RO is calculated from the RO period,  $T_p$ , or the RO frequency,  $f$ . The average of the pull-up ( $\tau_{pu}$ ) and pull-down ( $\tau_{pd}$ ) delay is also equal to the stage delay.

$$\tau_p = \frac{(\tau_{pu} + \tau_{pd})}{2} = \frac{T_p}{2(2\alpha + 1)} = \frac{1}{2(2\alpha + 1)f}$$

The switching capacitance per stage,  $C_{sw}$ , can be calculated from the current consumed by the switching transistors, the number of stages, the voltage, and RO frequency, as seen below.  $I_{DDA}$  represents the measured active current, and  $I_{DDQ}$  represents the measured quiescent current.

$$C_{\text{sw}} = \frac{(\text{IDDA} - \text{IDDQ})}{(2\alpha + 1)V_{\text{DD}}f}$$

From the switching capacitance per stage and stage delay, the switching resistance per stage,  $r_{\text{sw}}$ , can also be calculated.

$$r_{\text{sw}} = \frac{\tau_{\text{p}}}{C_{\text{sw}}}$$

The power per stage,  $P_{\text{sw}}$ , can be calculated from the voltage and switching current.

$$P_{\text{sw}} = V_{\text{DD}}(\text{IDDA} - \text{IDDQ})$$

Another parameter useful for characterizing processes is the power-delay product (PDP), also known as the switching energy, which can be calculated by the product of the switching power per stage and the stage delay.

$$\text{PDP} = \tau_{\text{p}}P_{\text{sw}}$$

These extracted parameters provide a good figure of merit to compare processes.

## 2.2 32 nm Ring Oscillators

Several RO designs have been taped out in a 32 nm process, including a 16-RO design featuring decoding logic to enable exactly one RO, a multiplexer to select the output of the enabled RO, and a divider, to scale down the frequency to allow less expensive test setups to measure the device. Each RO in this configuration has 101 stages.

Twin 6-RO chips have also been taped out, differing only in threshold voltage. Each RO has a separate enable and output pins, and each RO output is divided. Some ROs have 5 stages and others have 199 stages, hoping to yield more insight into reliability testing at the very high frequencies (above 10 GHz) generated by the 5 stage devices.

The 16-RO design will be covered in detail below.



### 2.2.1 Schematic

The initial core layout was developed from a custom-written Verilog netlist, but subsequent designs proved far easier to develop from schematics. Though the use of Encounter Place and Route can save some time during the layout phase, Encounter does not easily place components in the optimal position. The routing is also highly non-deterministic, so from run to run, the design may look drastically different. This is not suitable for test circuits such as ROs that require parasitic elements per stage to be matched as accurately as possible.

For parasitic extraction, a schematic was generated for a single ring oscillator in the core. Figure 2.1 shows the sample schematic with 101 inverting stages.

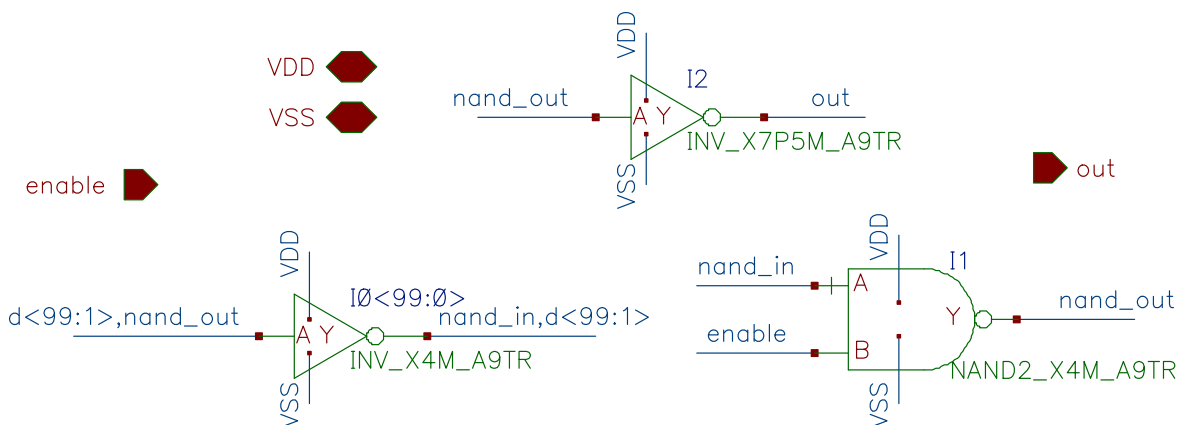


Figure 2.1: One of sixteen ROs in the 32 nm core.

This particular RO is a 101 stage oscillator using 100 4x inverters and a NAND gate for enable, with a 7.5x inverter providing an inverting buffered output. There consists several other devices in the core that have very similar topologies, differing only in threshold voltage and drive strength. The more complicated ring oscillators have load cells between stages to measure other MOSFET parameters, which consist of transistor-based cells with the gates, sources and drains tied in various manners to the power rails, each other, and the input and output of the neighboring inverters.

### 2.2.2 Layout

The core for the 16-RO chip was generated using an Encounter-based flow, covered in the Chapter 3. This is discouraged for new RO designs for the aforementioned reasons, notably the lack of control of parasitic elements between stages. Once the core was completed, it was manually wired up to the pad frame. Figure 2.2 shows the completed chip.

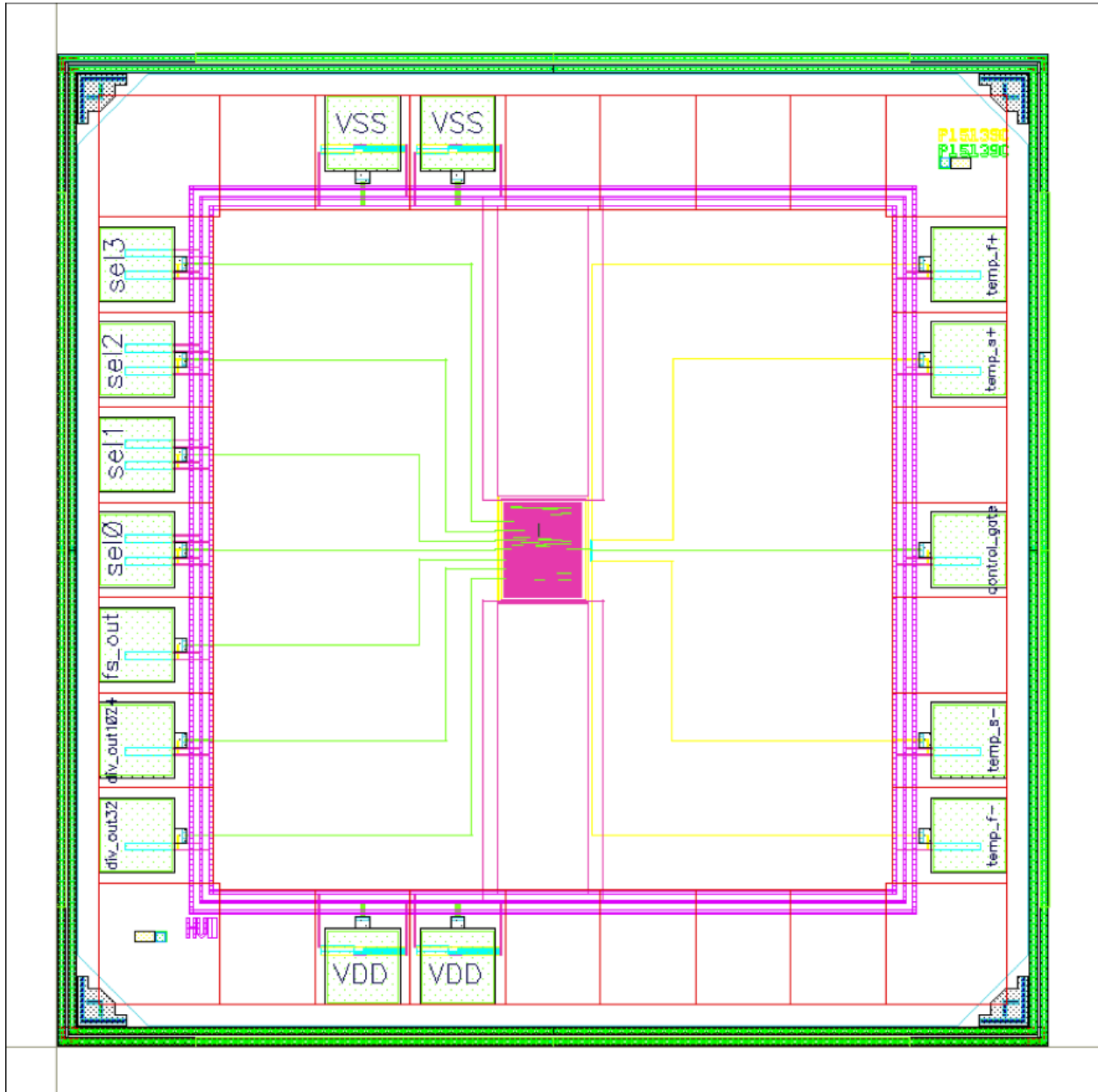


Figure 2.2: The completed 16-RO chip.

The core, expanded in Figure 2.3, is quite small in comparison to the rest of the chip, and though bypass capacitors are distributed within the core, more could be done to improve the power integrity, perhaps something closer to what was later done with the 90 nm RO chip as covered in Section 2.3.

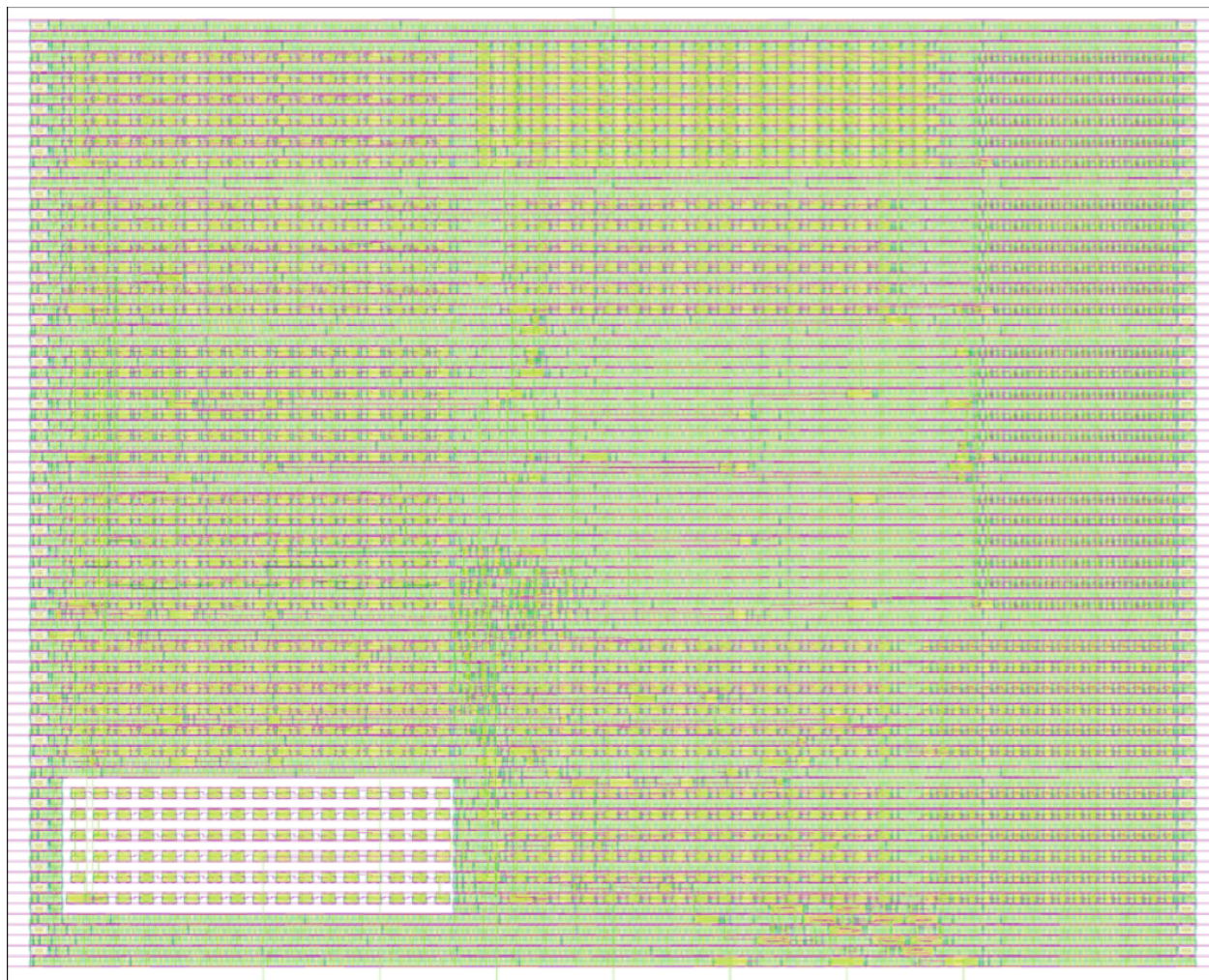


Figure 2.3: The completed 16-RO core.

Within the core, 16 ROs are partially visible, with six at the top, six at the bottom, and two on either side in the middle. The rest of the logic, including a divider, multiplexer, and demultiplexer/decoder, are distributed around the core, but tend to live near the center.

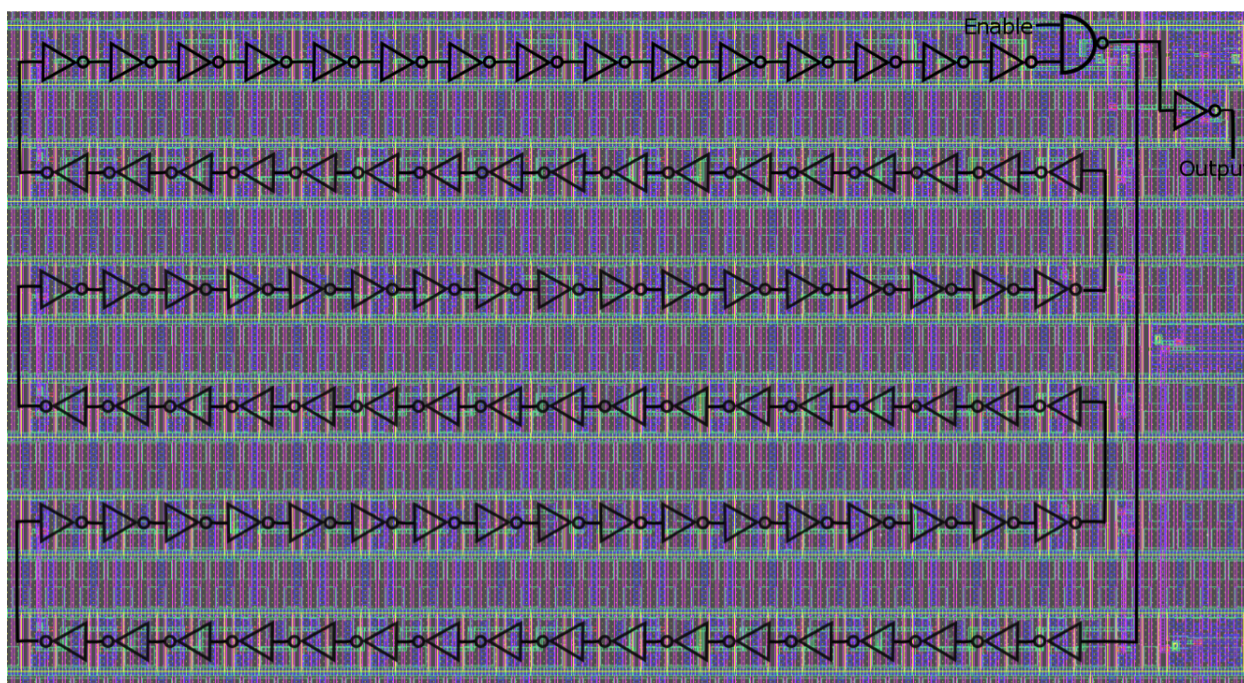


Figure 2.4: A single RO with schematic overlay.

Each RO was constructed using a complex script to create rectangular ROs with a serpentine path. This was in an attempt to reduce parasitics within each RO while still maintaining an Encounter-driven flow. Again, modern RO designs should avoid such a flow to give even more control of such parasitics. Figure 2.4 shows the overlaid schematic on a layout.

### 2.2.3 DRC and LVS

Design rule checking, as covered in Appendix F, passed at the top level, and the chip was successfully taped out. Layout vs. schematic, as covered in Appendix E, was also performed successfully. However, due to lack of simulation of the divider, it was not determined until the chip was back from fabrication that the divider was held in a state of constant reset, due to the use of a TIEHI cell on the active high reset input, instead of a TIELO cell. Simulating would have immediately shown this fault. Thankfully, the chip included a full-speed output, which bypasses the faulty divider entirely, and thus, the chip could still be measured as intended.

It cannot be stressed enough that passing DRC and LVS is not a substitution for a design review, especially without simulation results.

#### 2.2.4 Parasitic Extraction

Parasitic extraction, or PEX, is covered in Appendix G, and was performed after fabrication. A single RO was extracted from the core layout, as running PEX on the entire core proved to be too time consuming and CPU intensive for both PEX and simulation. The single RO took several minutes to extract, and several hours to sweep over a wide voltage range.

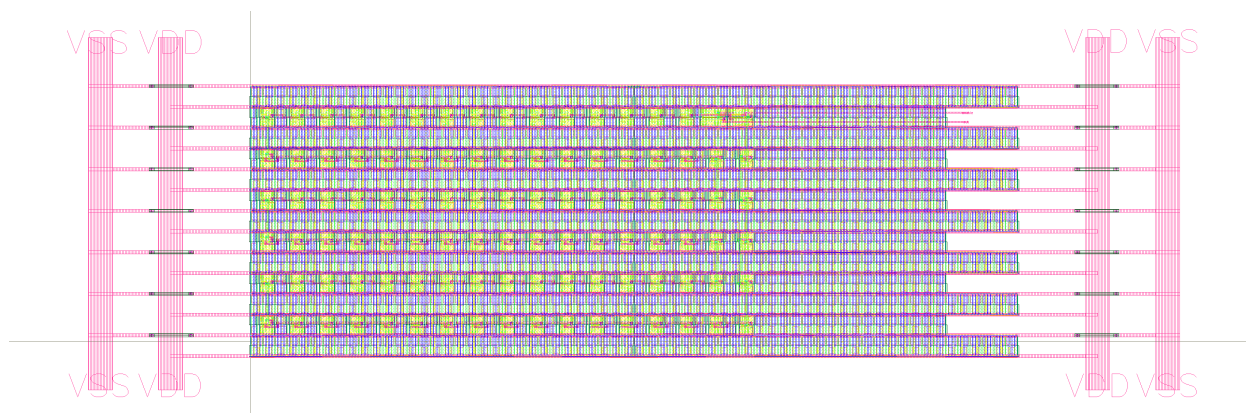


Figure 2.5: A single RO extracted from the core, used for PEX.

As seen in Figure 2.5, the single RO still includes filler cells and partial power rings, in an attempt to capture more of the original construction used within the core. The schematic used for this PEX run can be seen in Figure 2.1. From PEX simulation and actual measurements from the lab, the two were compared.

## 32SOI 101–Stage 4x RVT RO Full Speed Frequency

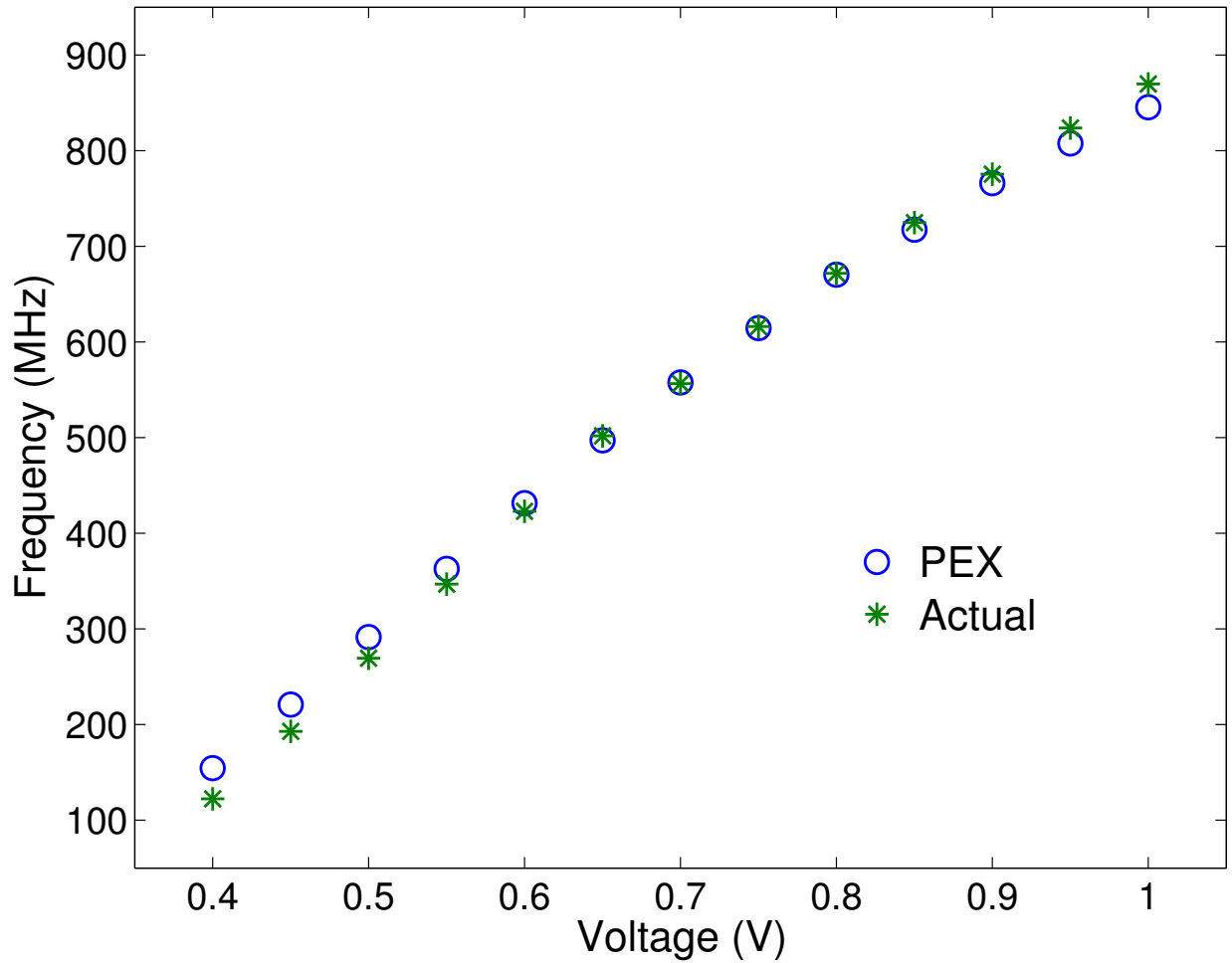


Figure 2.6: The measured RO frequency vs. PEX frequency from simulation at RT.

As seen in Figure 2.6, the two are markedly similar. This shows that simulating with parasitics can be a very good way of determining if the digital circuit will function as expected in real life. Unfortunately, the PEX models do not work at cryogenic temperatures as the models were not designed to operate over such a wide temperature range, but perhaps in the future, such models can be extracted from test results through testing at cryogenic temperatures.

## 2.3 90 nm Ring Oscillators

The 90 nm RO chip was constructed differently than the 32 nm RO chip. Instead of 16 ROs tied together with multiplexers, demultiplexers, and dividers, there are four total ROs on the chip. Each RO is composed of 198 inverters and a single NAND gate for enable, with a drive strength of 1x.

### 2.3.1 Schematic

The RO schematic can be seen in Figure 2.7, which resembles the 32 nm RO with the exception of two inverters feeding the NAND gate, used to create a sharper rise when enabling the device, as well as the lack of an inverting buffer on the full speed output. The output buffer was moved to the divider schematic, but electrically, it is very similar to the previous design.

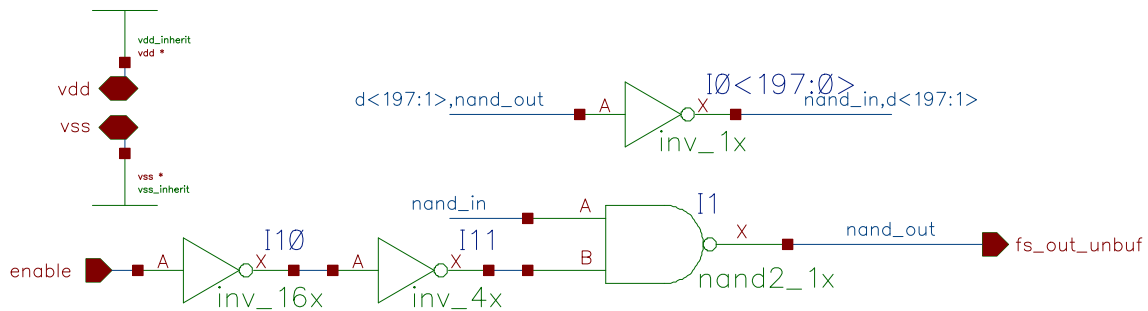


Figure 2.7: The RO schematic as used for the 90 nm chip.

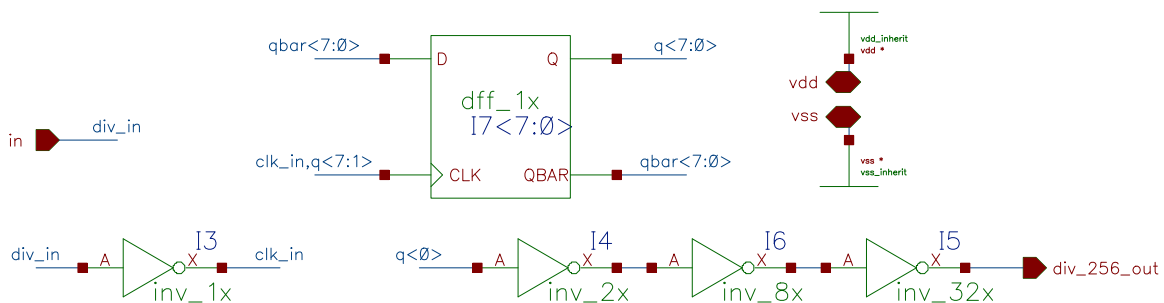


Figure 2.8: The divider schematic as used for the 90 nm chip.

Two ROs feature the divider, as seen in Figure 2.8, but two others are divider-less. The full speed ROs are still buffered through the same inverting buffer arrangement as seen in the bottom of the divider schematic.

The divider consists of eight D flip-flop stages, each dividing the frequency by two, for a total scale factor of 256. The output frequency of each independent RO is delivered to a pad driver to drive the highly-capacitive pad.



### 2.3.2 Layout

The layout differs highly from the previous 32 nm RO. Instead of an Encounter-driven flow, every hierarchical layer was captured in both schematic and layout form, and LVS and DRC were used at every level of the hierarchy.

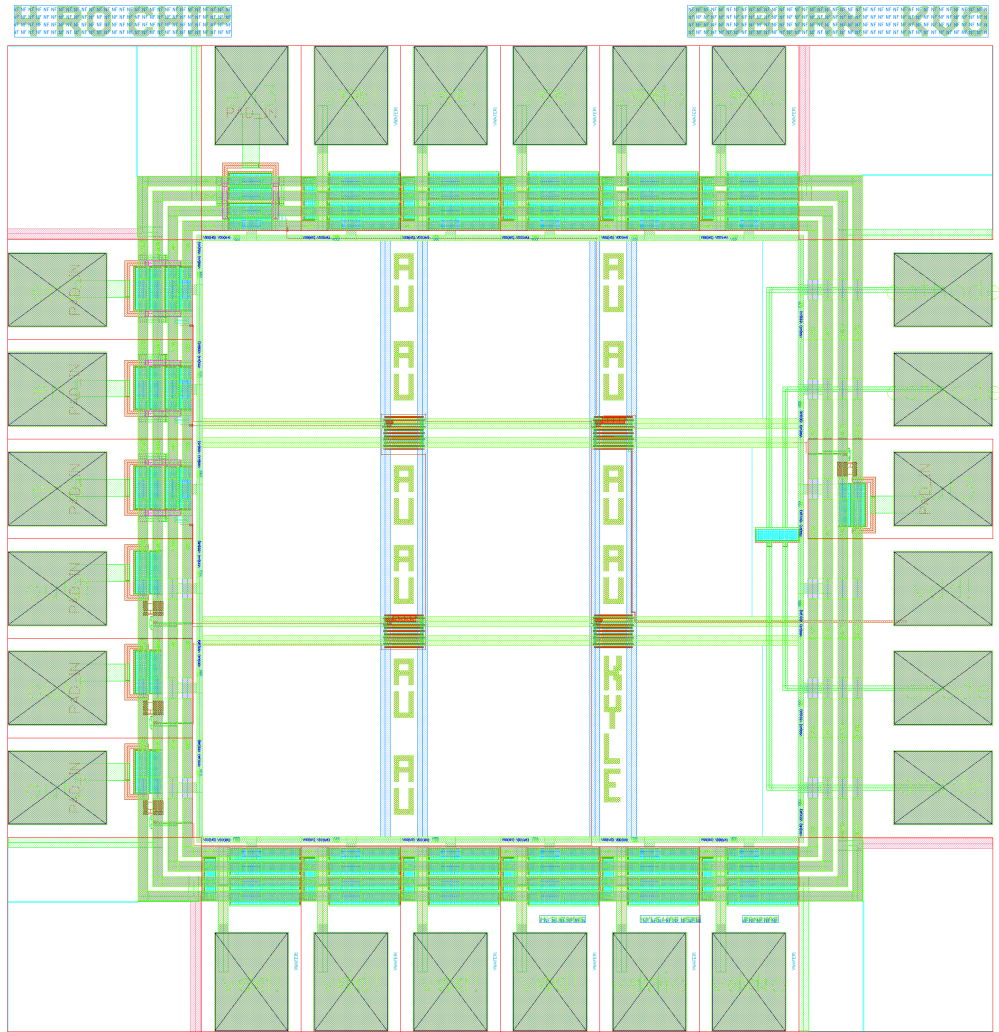


Figure 2.9: The completed 90 nm RO chip.

The completed chip can be seen in Figure 2.9. The four ROs are located near the middle of the chip. They divide the core into 9 separate sections. The blank-appearing places are actually filled with vertical natural capacitors, seen in Figure 2.12. The pad frame uses wider lines for power and ground connections, and thus, the power integrity of this chip would be improved compared to the thinner lines found in the 32 nm version. The separate enable lines allow the chip to be powered but not enabled for static power measurements; this was not possible with the 32 nm chip.

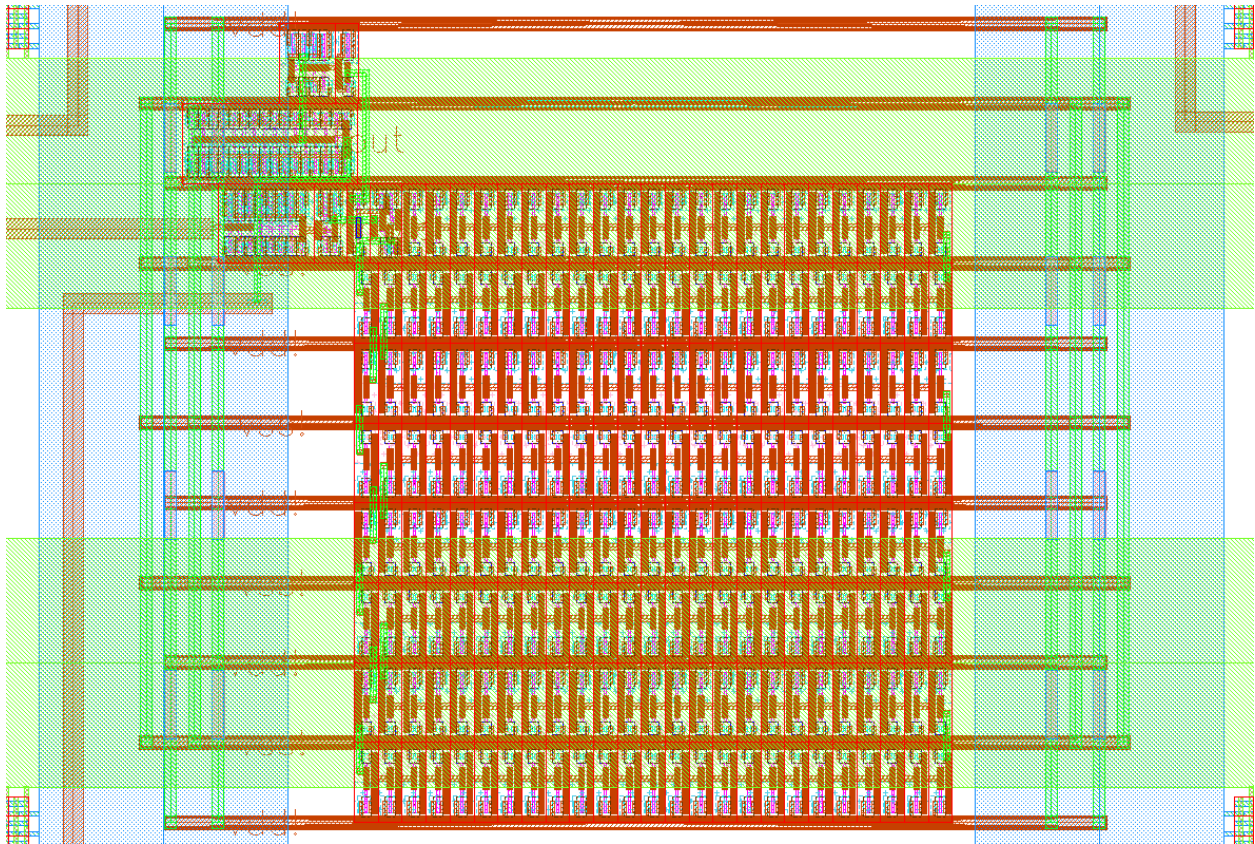


Figure 2.10: One of two full speed ROs on the 90 nm chip.

The layout for the full speed RO, seen in Figure 2.10, shows a power ring for distributing VDD and VSS within the core. Connections are made to the main power rails running horizontally and vertically in green and blue, respectively, through many vias for lower resistance. The buffer at the top left drives a wide line to eventually drive a pad driver.

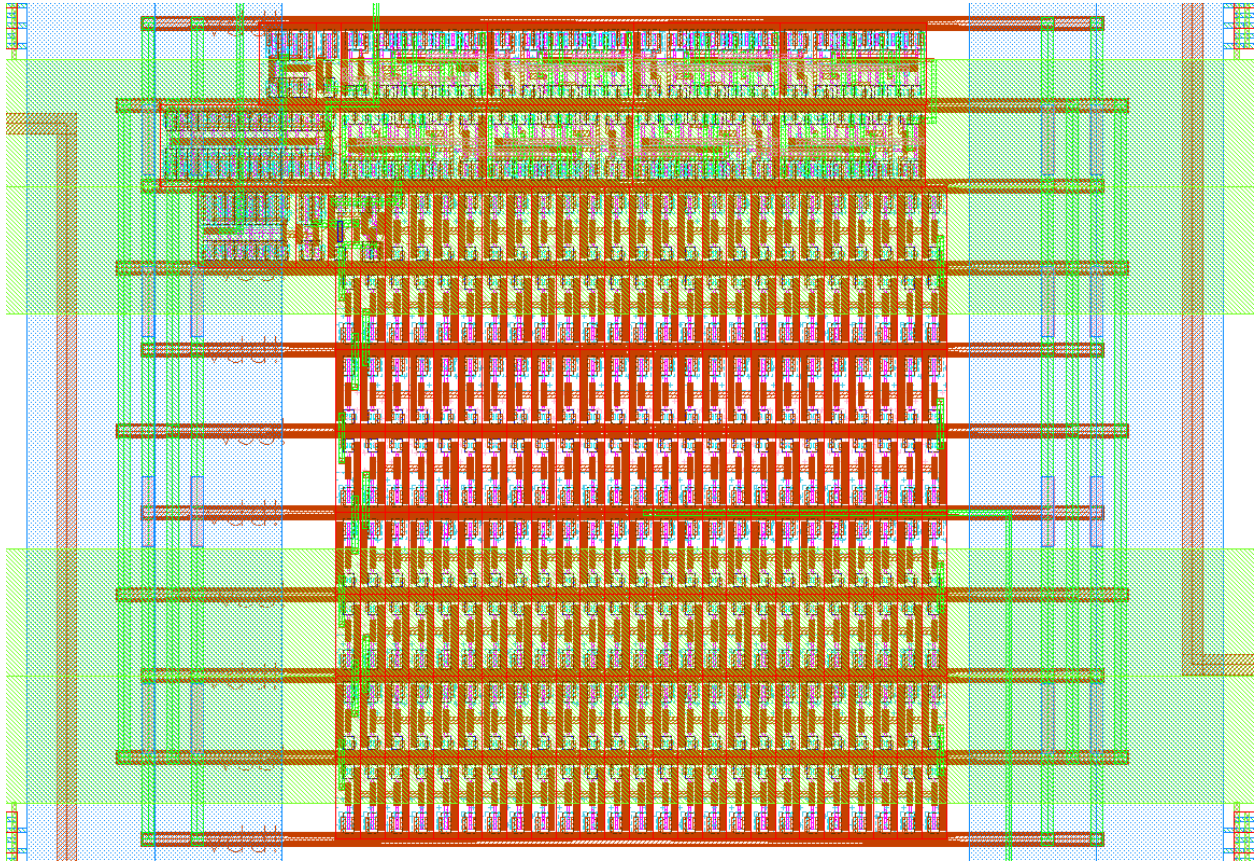


Figure 2.11: One of two divided ROs on the 90 nm chip.

The layout for the divided RO, seen in Figure 2.11, is nearly identical to the full speed version, with the exception of the D flip-flops added on top of the main RO.

One improvement to the power integrity of this chip was the introduction of a “plaid” patterned capacitor.

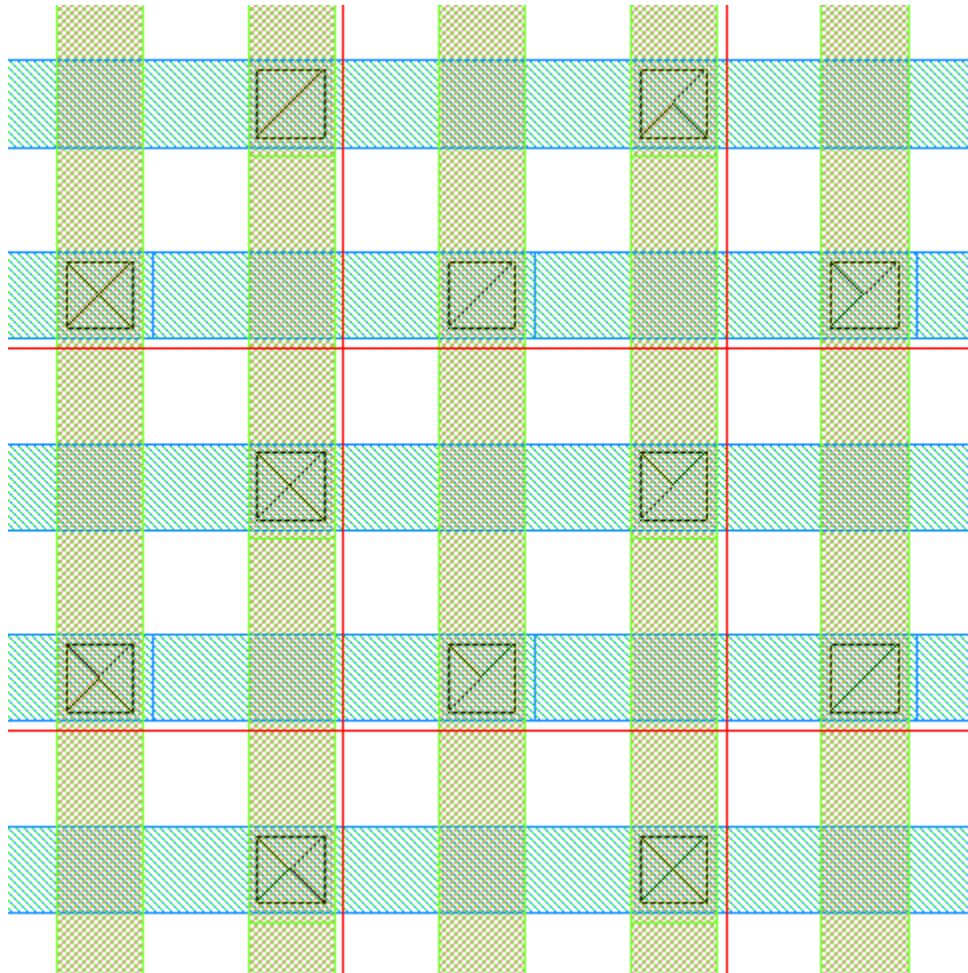


Figure 2.12: “Plaid” capacitor used for bypass across power rails.

This “plaid” capacitor offers capacitance figures comparable to that of the `vncap` PCell found within the technology, but offers lower resistance from end to end. Since it distributes power horizontally and vertically on alternating metal layers, it ties in nicely to the main power ring structure set up by the pad frame. This capacitor is also much more voltage tolerant than a traditional MOSFET gate capacitor as the dielectric between metal layers has a higher breakdown voltage than the thin gate oxide of a MOSFET, which is desirable for stress and reliability testing.

### 2.3.3 DRC and LVS

The design passes DRC with the exception of density rules. As mentioned in the DRC tutorial, failing density rules is acceptable if the manufacturer or aggregator runs their own density fill algorithms, but do not assume this is always the case. The design also passes LVS at the top level.

### 2.3.4 Parasitic Extraction

For the same reasons mentioned in the 32 nm RO PEX section, a subset of the chip was used for parasitic extraction. The layout can be seen in Figure 2.13.

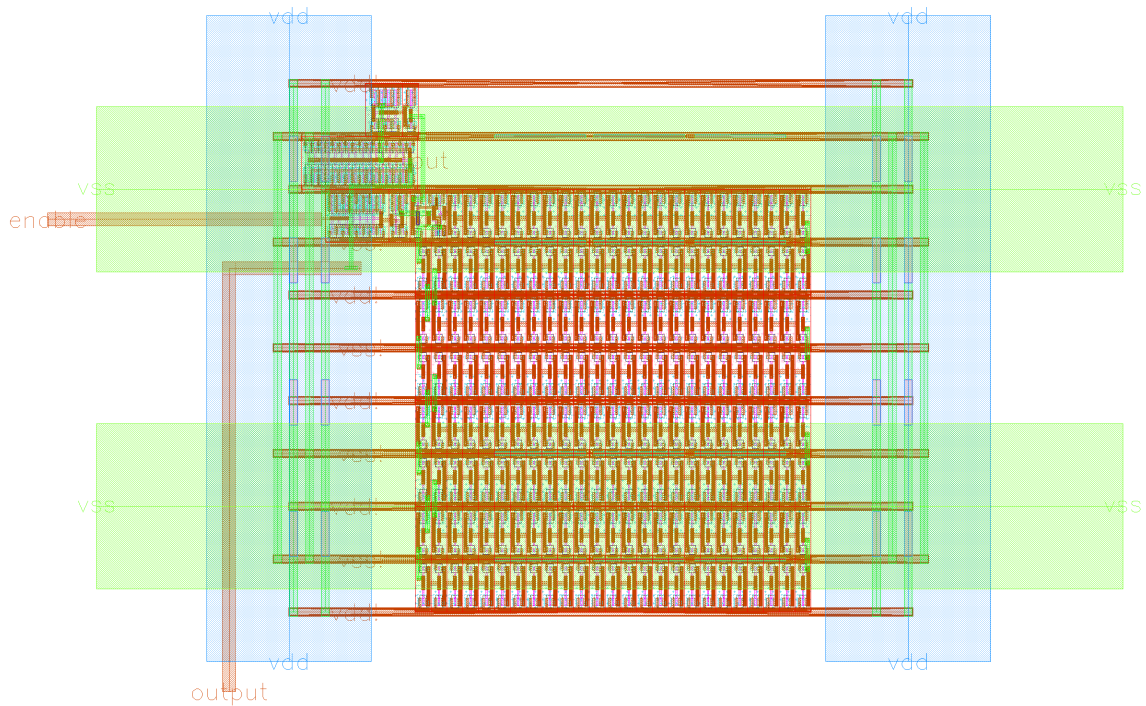


Figure 2.13: The 90 nm layout used for PEX.

The PEX layout again incorporates parts of the power ring structure, but does not include bypass capacitance. Since perfect voltage sources are applied to the RO in simulation, bypass capacitors would have little effect on the simulation results. The PEX results were

simulated, as seen in Figure 2.14, but at the moment, no measurements were taken from actual chips since they are in fabrication.

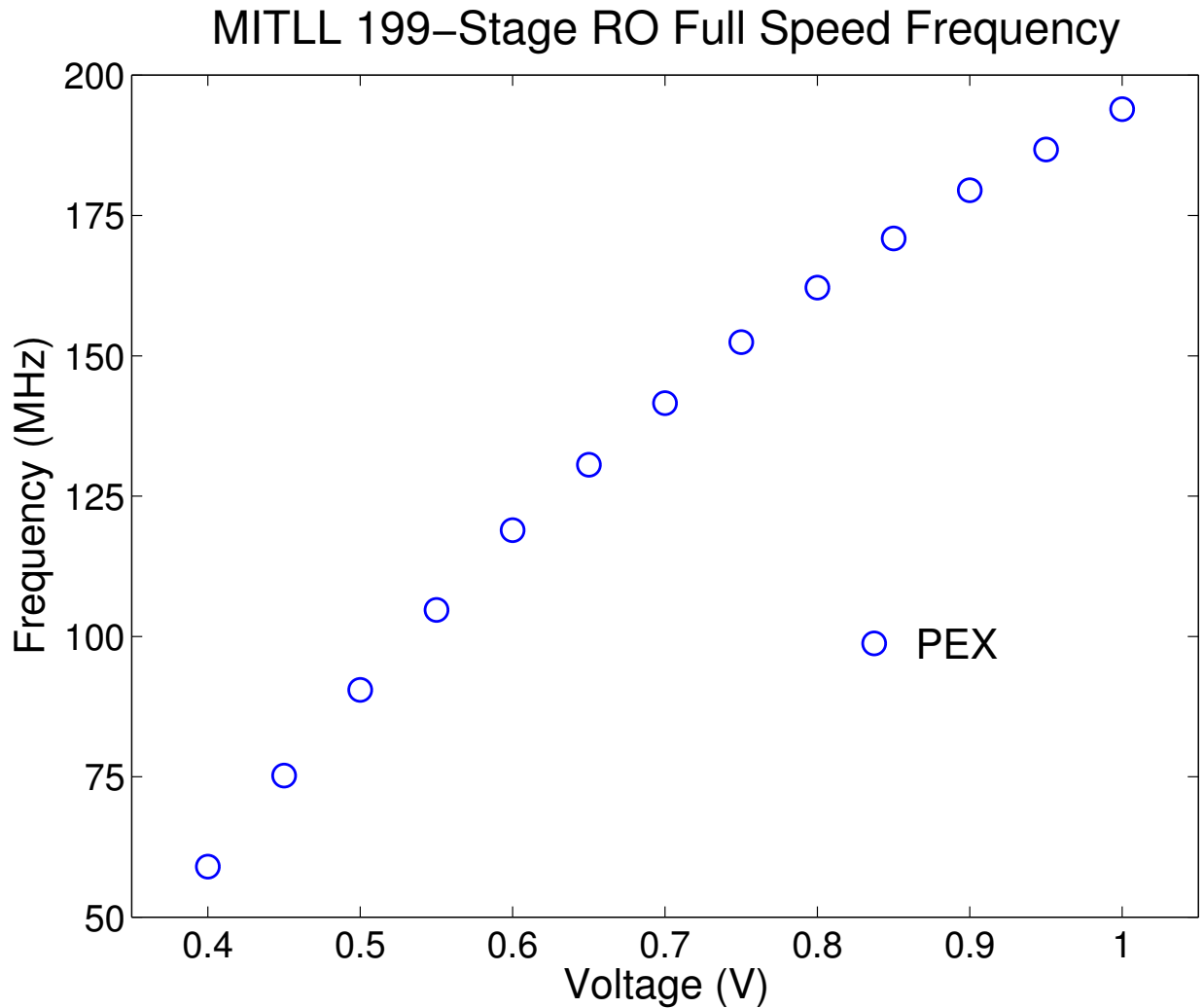


Figure 2.14: Expected frequencies of the 90 nm RO chip at 4.2 K.

The frequencies follow a similar trend as the 32 nm chip, but are a good bit slower due to added gate capacitance from increased transistor size and additional stages.

## Chapter 3

### 32-bit Multiply-Accumulate Unit

#### 3.1 Background

The multiply-accumulate operation is commonly used in digital signal processing. The operation simply takes the product of two numbers and adds it to a third.

$$A = A + (B \times C)$$

The multiply-accumulate operation, though simple, can be quite complex in digital logic form thanks to carry-lookahead adders and parallel multipliers. This particular MAC unit is a two-phase 32-bit MAC and provides a relatively high activity factor for comparison across technologies.

#### 3.2 Hardware Description Language

This particular design flow required the use of an as-provided behavioral Verilog description of the 32-bit MAC. This was then synthesized into a Verilog netlist of standard cells using a rudimentary script executed by the Cadence Encounter RTL Compiler. The netlists were then turned into a digital core using another script to place and route the design using Cadence Encounter Place and Route. Both scripts can be found in the Encounter tutorial.

#### 3.3 Simulation

Simulation is handled by as-provided test benches written for the Cadence Incisive simulator. This is again covered in detail in the Encounter tutorial.

### 3.4 Layout

The rest of the layout was accomplished using Cadence Virtuoso. The core was manually placed within a pad frame and was manually routed to the pads. For future work, the width of the wires should be increased to help with power integrity to the core.

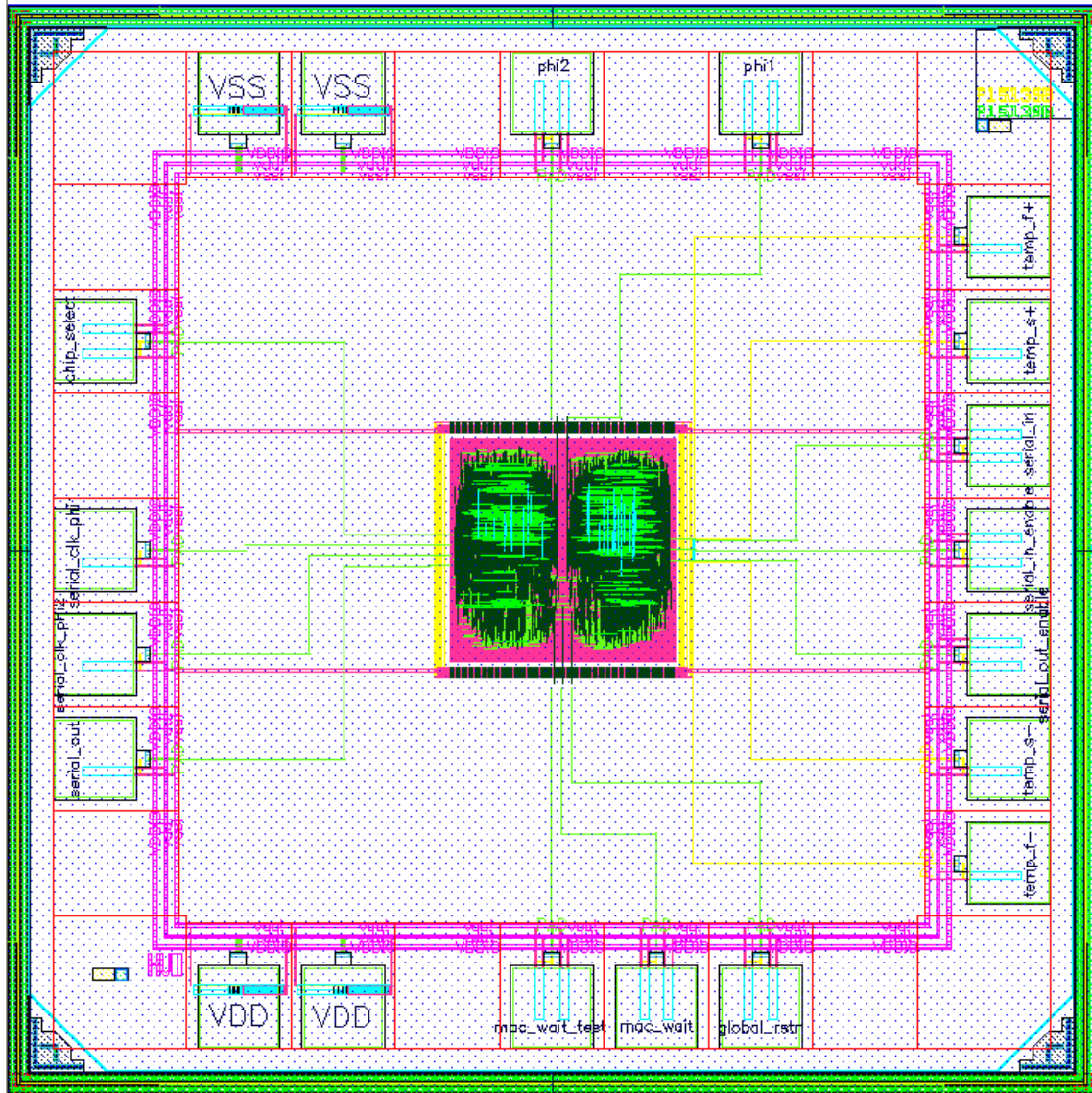


Figure 3.1: The completed 32 nm 32-bit MAC chip.



As seen in Figure 3.1, the core is significantly larger than the RO chip. The pad frame itself is rather anemic in that the power rings were not much larger than many of the signal lines and was improved in later 32 nm designs by making the metal lines in the power rings robust.

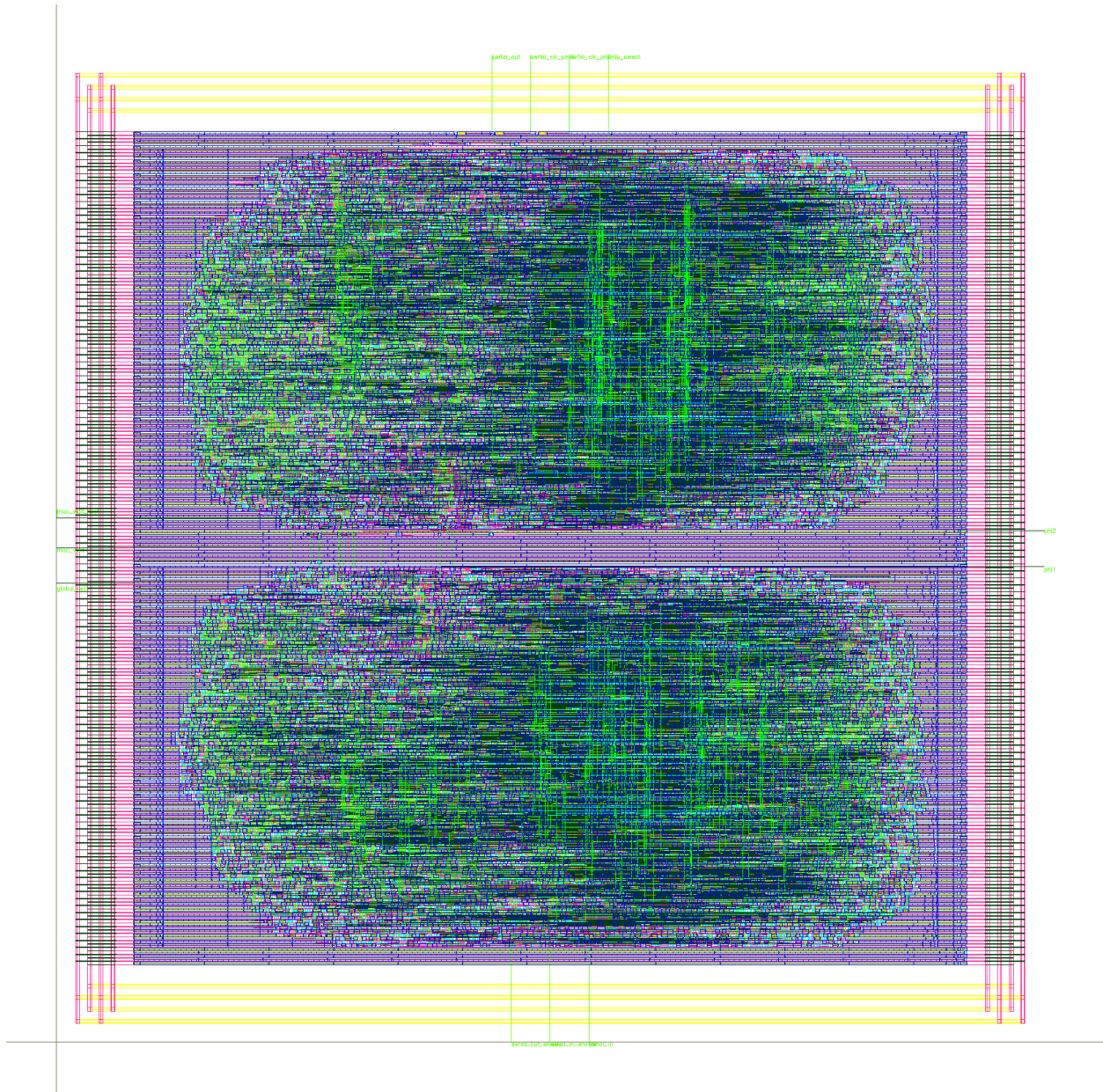


Figure 3.2: The completed 32 nm 32-bit MAC core.

The core was fully constructed using automated scripting techniques within Cadence Encounter Place and Route. The two halves of the chip are identical except for threshold

version; thus, this chip provides two independent MAC units, but only one can be active at a time.

### **3.5 DRC and LVS**

This chip passed all design rule checks required by the foundry, but LVS was not performed. It was assumed that what Encounter generates would be considered accurate. However, this is a poor assumption. In some cases, LVS can catch missing steps in the automated place and route process. Thus, LVS should still be performed on the core and top level with the assistance of the mixed-input mode that Calibre provides, handling both Verilog and SPICE netlists.

## Chapter 4

### Static Random Access Memory

#### 4.1 Background

SRAM is of particular interest in cryoelectronics, as memory technologies in alternative electronic systems can be quite large. Conventional SRAM, if made to operate at cryogenic temperatures, is very desirable. By packing more memory into the dewar with the superconducting supercomputer, information can be processed much faster without seeking an external memory system.

Several bit cells, a sense amplifier, precharge, and a write buffer were designed and simulated in a 14 nm technology, with three bit cells and a sense amplifier actually taped out.

## 4.2 Schematic

One of the first steps in SRAM design is to choose an appropriate inverter and access transistor size for the bit cell. In order to choose, the main figures of merit are size and static noise margin. If the bit cell is too big, the SRAM may not provide the required memory density. However, if the static noise margin of either read or write is too low, the SRAM may simply fail to operate in real-world conditions. Thus, the static noise margin must be analyzed using a butterfly structure. [4]

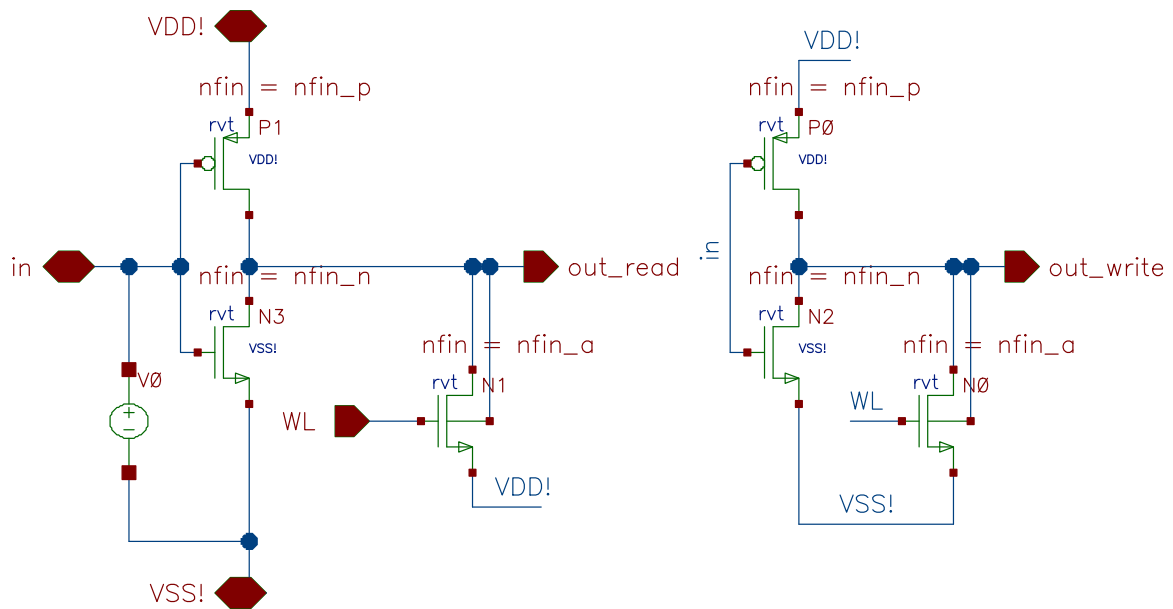


Figure 4.1: The 14 nm butterfly test schematic.

In Figure 4.1, the read and write SNM analyses were performed simultaneously using ADE. Refer to the simulation section for more information.

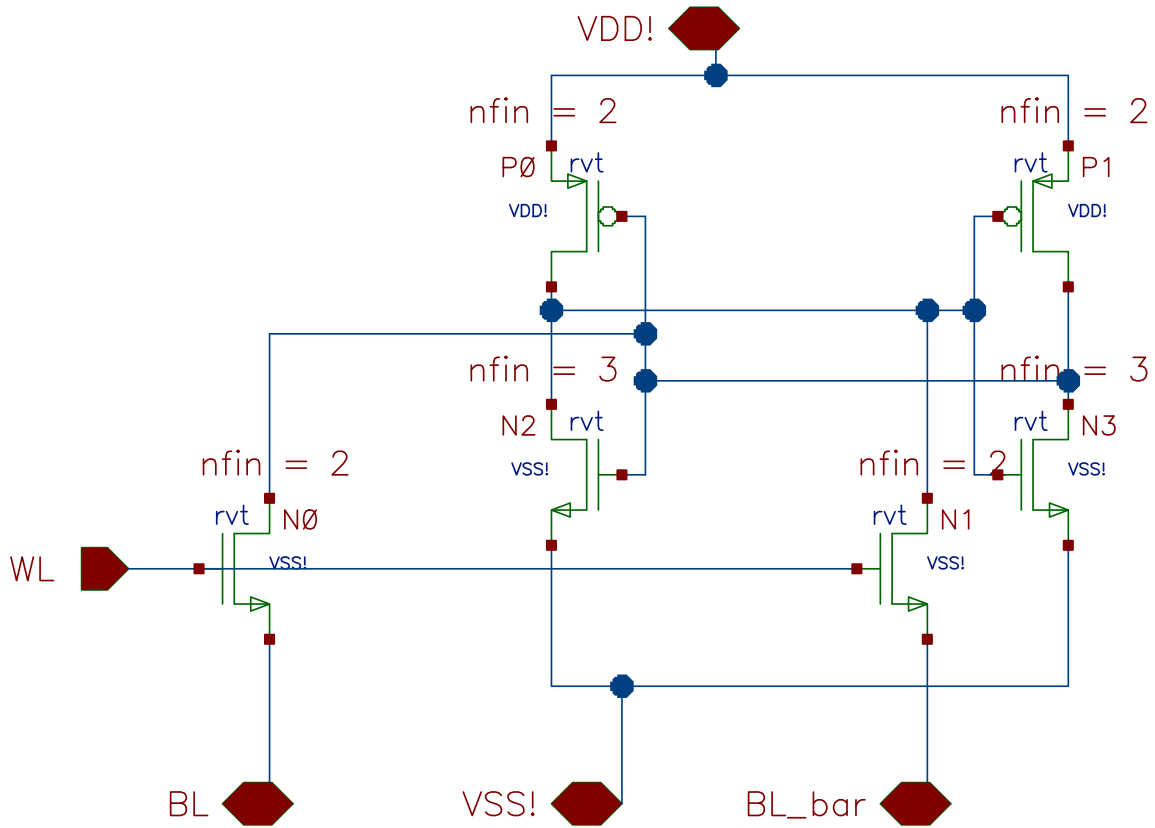


Figure 4.2: The 14 nm 3, 2, 2 bit cell schematic.

Three individual bit cells were designed using the same basic schematic, only changing the sizes of each transistor. The 3, 2, 2 bit cell can be seen in Figure 4.2.

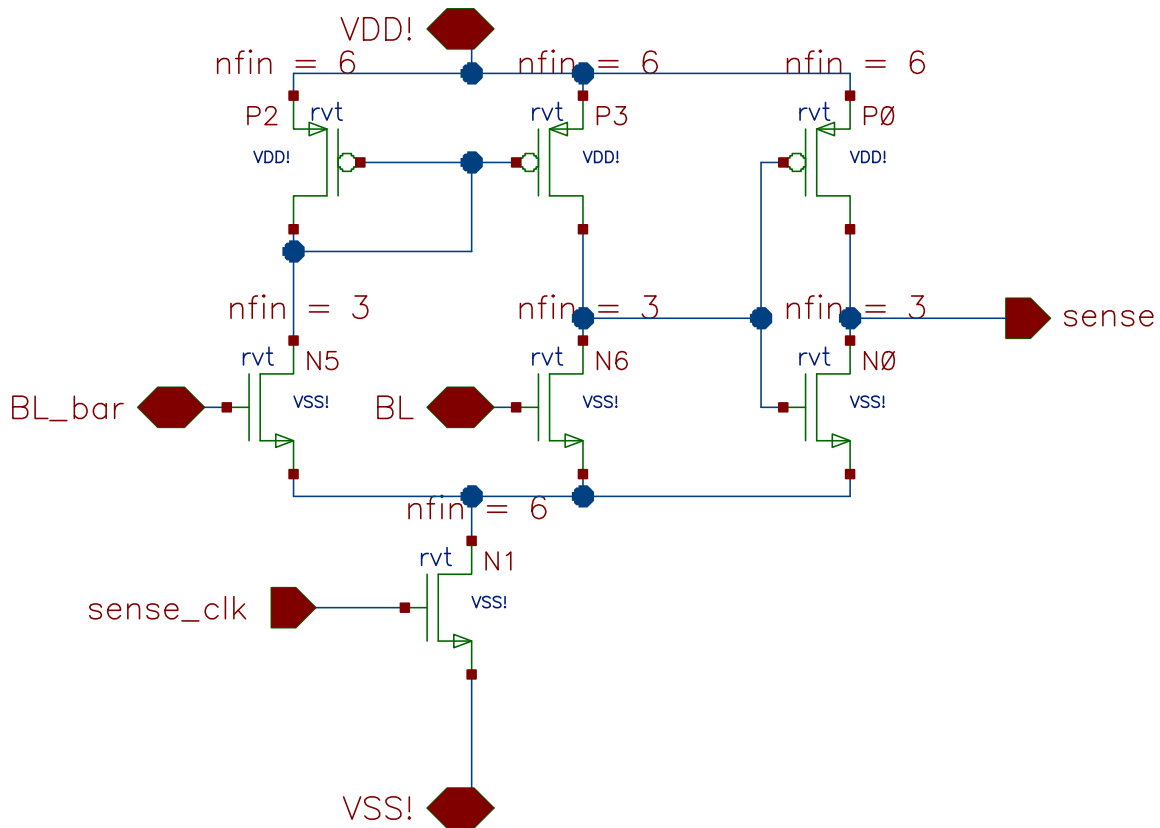


Figure 4.3: The 14 nm sense amplifier schematic.

A sense amplifier, seen in Figure 4.3, was also designed, simulated and taped out with three of the standard 6T bit cells. This sense amplifier is based on a traditional MOSFET differential amplifier, with an n-type MOSFET for enabling the sense amplifier only during a read to minimize static power.

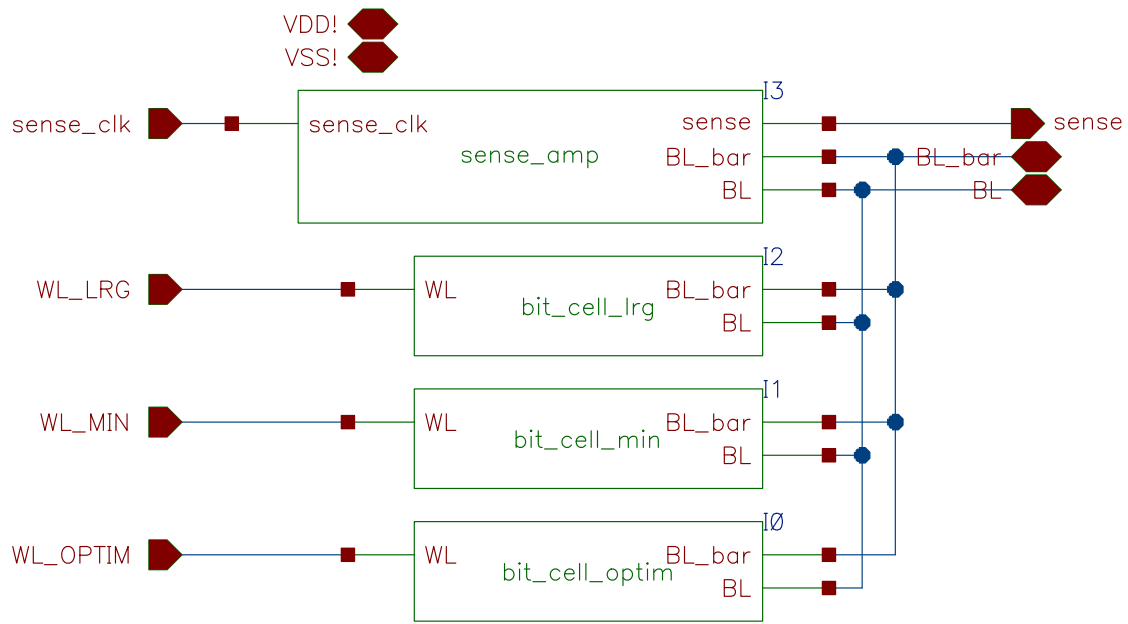


Figure 4.4: The 14 nm top level SRAM circuits schematic.

The three bit cells and sense amplifier were joined together using the bit lines, with separate word lines for enabling each bit cell independently, as seen in Figure 4.4.

### 4.3 Simulation

A simple read and write test was performed to exercise all blocks of a basic SRAM structure, including the bit cell, write buffer, sense amplifier, and precharge circuit.

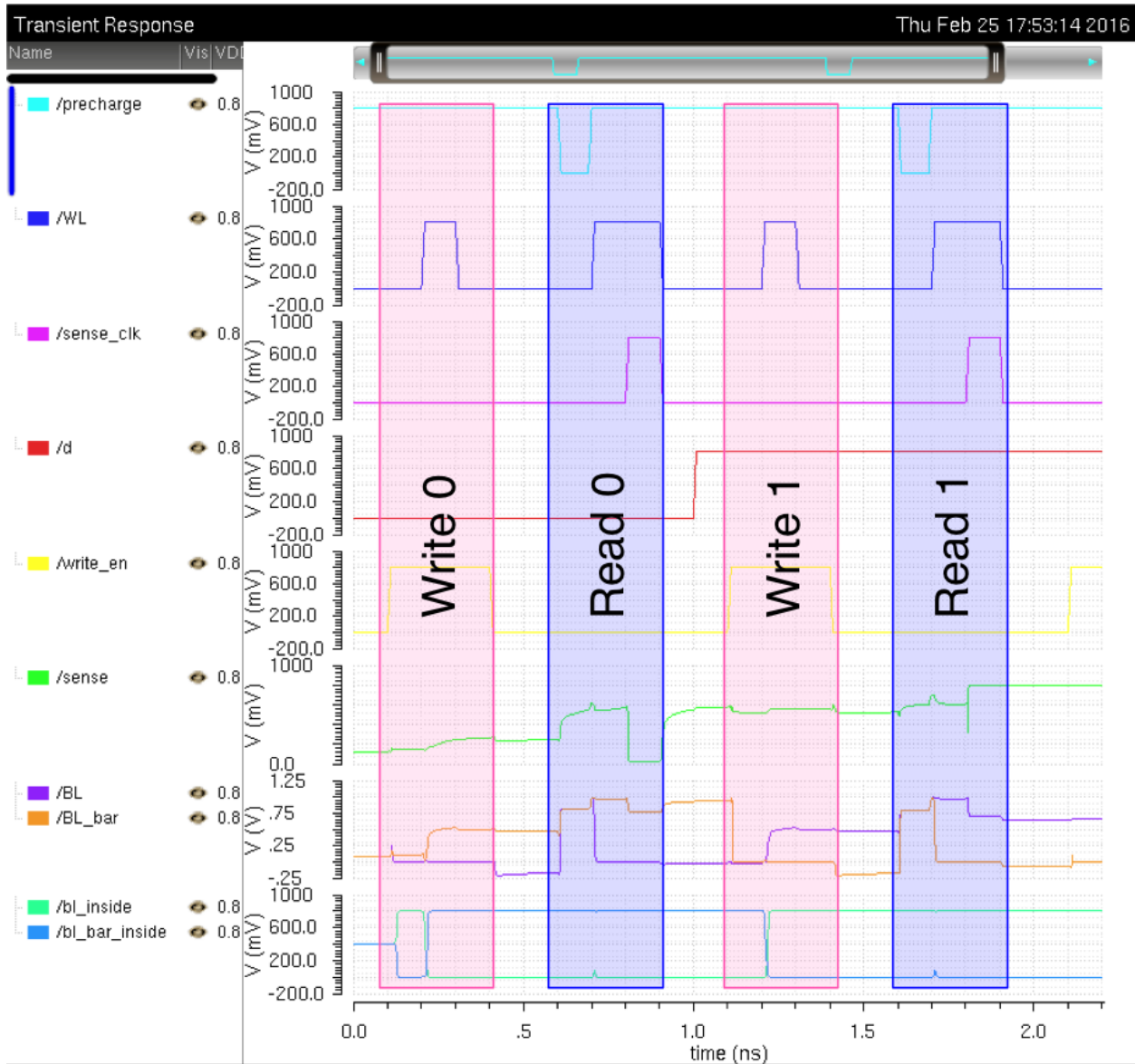


Figure 4.5: The 14 nm SRAM simulation results showing successful reads and writes.

As seen in Figure 4.5, the blocks all perform their intended function, with fast swings provided by the bit cell, sense amplifier, and write buffer.



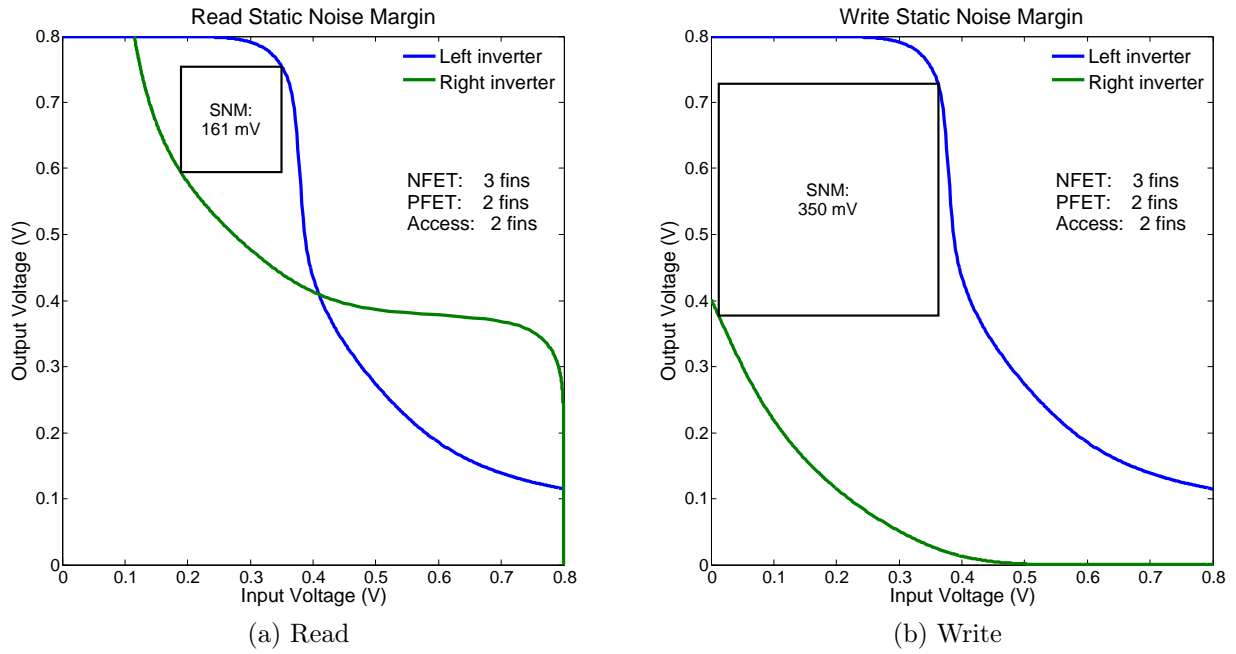


Figure 4.6: Static noise margin for "322" bit cell.

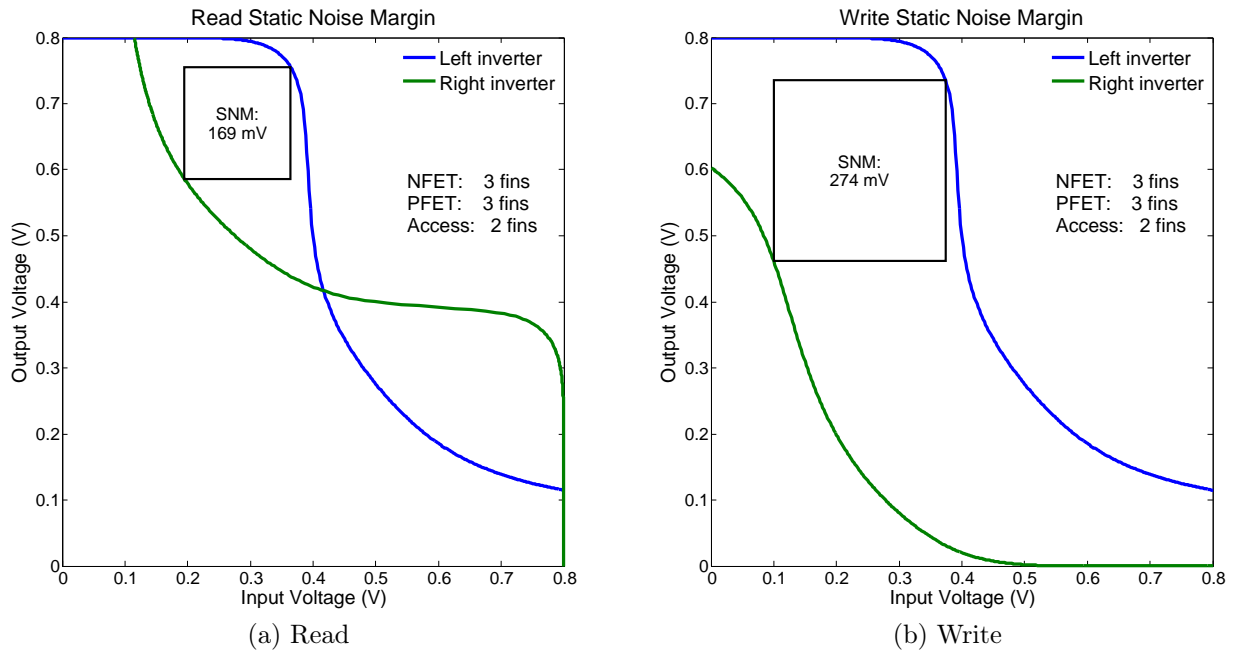


Figure 4.7: Static noise margin for "332" bit cell.

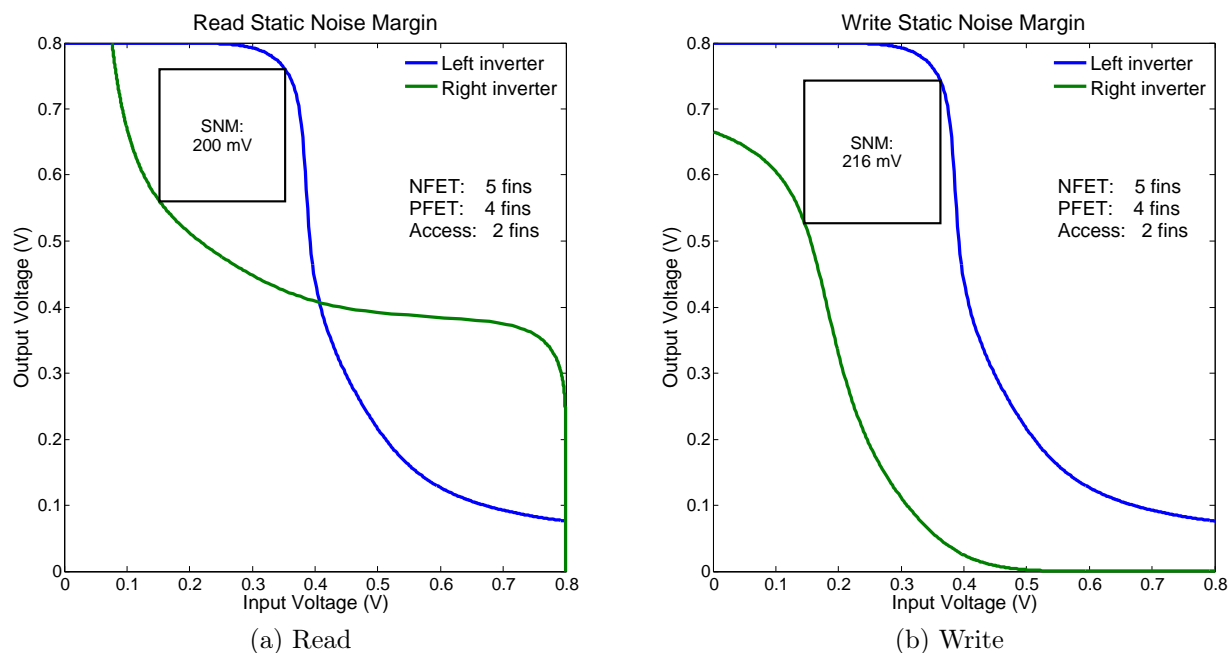


Figure 4.8: Static noise margin for “542” bit cell.

An ADE environment was set up to sweep the three transistors to perform a static noise margin test for both read and write. A MATLAB script was written to read in the exported data and generate static noise margin graphs. See Appendix A for the scripts. Figures 4.7 through 4.8 show increasing read SNM and decreasing write SNM for larger transistors. A summary of static noise margins can be found in Table 4.1.

Table 4.1: Read and Write SNM Summary (0.8 V)

(NFET, PFET, Access)	Read SNM	Write SNM
3, 2, 2	161 mV	350 mV
3, 3, 2	169 mV	274 mV
5, 4, 2	200 mV	216 mV

As seen in Table 4.1, the write static noise margin is higher than the read static noise margin for all sizes of transistors chosen. Increasing the read SNM comes at a great cost to the write SNM. The 5, 4, 2 bit cell was chosen to get the read SNM as close to the write SNM as possible. The 3, 2, 2 and 3, 3, 2 sizes were chosen for their size. In SRAM bit cell design, minimizing the size of the bit cell is critical for high memory density.

#### 4.4 Layout

The top level chip differs not only in size but also in interconnects. The chip measures  $2\text{ mm} \times 1\text{ mm}$ , whereas most other chips are  $1\text{ mm} \times 1\text{ mm}$ . This chip also uses C4 (solder) bumps instead of traditional wire bond pads.



Figure 4.9: The completed 14 nm SRAM test chip.

As seen in Figure 4.9, there is an array of 12 by 5 pads. The left 5 by 5 pads are devoted entirely to the SRAM test structures, whereas the rest of the chip consists of individual transistor test structures.

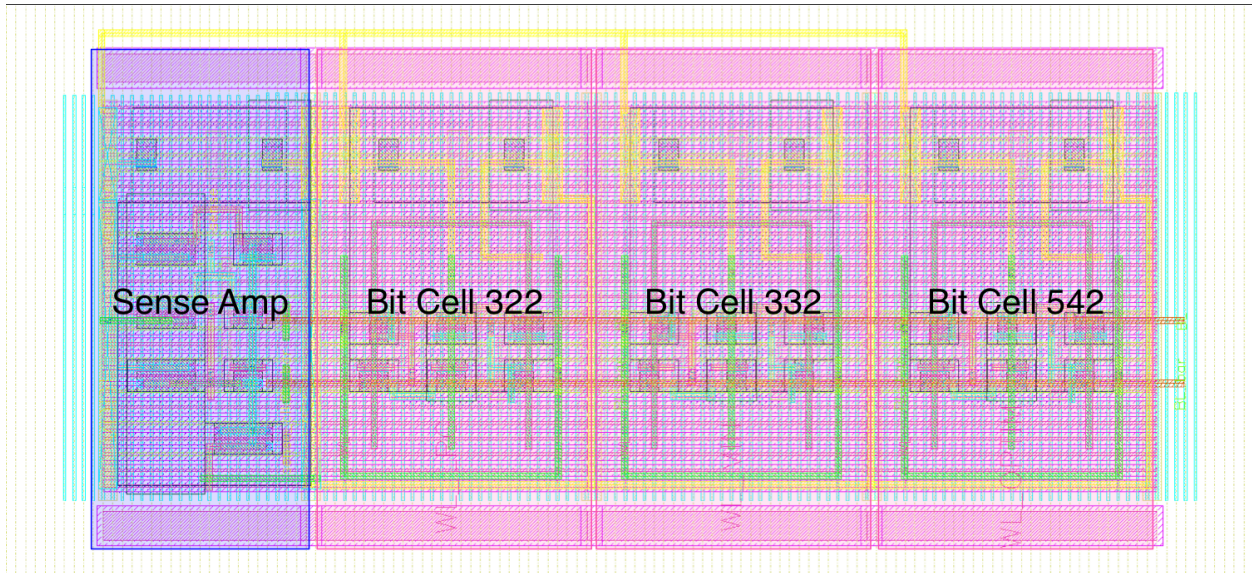


Figure 4.10: The completed 14 nm SRAM test circuits.

The individual structures designed were not optimized for area due to time constraints, seen from the relative density of cells in Figure 4.10. By sharing diffusion regions, the transistors could be much closer together to increase bit cell density.

#### 4.5 DRC and LVS

The 14 nm SRAM test circuits chip passed all design rule checks. Density fill was required by the foundry, resulting in many hours spent making those related checks pass due to a relative lack of documentation. Part of the standard flow for this technology includes an automated density fill step, but this was non-functional in the environment used.

LVS was used throughout the hierarchy and passed at the top level with all circuits, both SRAM and single-transistor portions.

## Chapter 5

### Conclusion and Future Work

#### 5.1 Conclusion

Many areas within the field of cryoelectronics still await exploration. Being able to develop better models for simulating SRAM bit cells and fast logic circuits at cryogenic temperatures would allow designers to build more complicated circuits with confidence that their designs will work when used in superconducting computing environments. As the static power consumption continues to climb, new avenues will need to be pursued. With the use of ring oscillators and other test circuits, it is possible to characterize existing CMOS processes for relevancy in cryoelectronics. The RO chips covered in Chapter 2 will help extract process information such as parasitics and MOSFET C-V characteristics, while the 32-bit MAC unit and SRAM circuits covered in Chapters 3 and 4 will help to show performance of such processes at cryogenic temperatures in both speed and data retention.

#### 5.2 Future Work

The circuits designed here still await testing and analysis, and some are still away at the foundry at the time of writing. A large stockpile of information remains on the servers beyond what can be condensed into this thesis. By building off of the designs constructed to date, better test circuits may be able to yield more accurate information for even more technologies, beyond those used to design the existing test circuits.

## Bibliography

- [1] F. Pollack. (1999) New microarchitecture challenges in the coming generations of CMOS process technologies. Intel. [Online]. Available: <http://research.ac.upc.edu/HPCseminar/SEM9900/Pollack1.pdf>
- [2] (2016) RP-082B2 4K pulse tube cryocooler series. Sumitomo Heavy Industries, Ltd. [Online]. Available: <http://www.shicryogenics.com>
- [3] M. Bhushan and M. Ketchen, *Microelectronic Test Structures for CMOS Technology*. Springer, 2011.
- [4] N. Rahman and B. P. Singh, "Static-noise-margin analysis of conventional 6T SRAM cell at 45nm technology," *International Journal of Computer Applications*, vol. 66, no. 20, 2013.

## Appendices

## Appendix A

### MATLAB Code

#### A.1 SRAM Static Noise Margin Analysis

##### A.1.1 Parser

```
% Cadence .matlab parser for static noise margin analysis
% Kyle Owen - 9 July 2016

nmos = [2:5];    % Finger sizes of NMOS of inverter
pmos = [2:5];    % Finger sizes of PMOS of inverter
access = [2:5]; % Finger sizes of NMOS of access transistor

% number of data points per graph
numPoints = 400;
% total number of sweeps
numRuns = size(nmos,2)*size(pmos,2)*size(access,2);

fid_read = fopen('H:\thesis\14LPP_SRAM\butterfly_read3.matlab');
fid_write = fopen('H:\thesis\14LPP_SRAM\butterfly_write3.matlab');
% ignore first line read to get rid of duplicate titles
top = fgetl(fid_read);
top = fgetl(fid_write);

% parse titles of plots to create 4D matrices later
expression = '\(([^\)]+)\)';    % only grab contents of parenthesis
titles = regexp(top,expression,'match');

% remove all non-numeric characters
for n = 1:numRuns
    titles(n) = titles(n*2);
    titles(n) = strrep(titles(n), '(', '');
    titles(n) = strrep(titles(n), ')', '');
    titles(n) = strrep(titles(n), 'nfin_', '');
    titles(n) = strrep(titles(n), '=', '');
    titles(n) = strrep(titles(n), 'p', '');
    titles(n) = strrep(titles(n), 'n', '');
end
```



```

        titles(n) = strrep(titles(n), 'a', '');
        titles(n) = strrep(titles(n), ',, ', ', ');
    end

    % get rid of extraneous data
    for n = 1:numRuns
        titles(numRuns*2-n+1) = [];
    end

    % parse the rest of the (CSV) files
    data_read = textscan(fid_read, '%f', 'delimiter', ',');
    data_write = textscan(fid_write, '%f', 'delimiter', ',');
    fclose(fid_read);
    fclose(fid_write);

    % reshape matrices into [numPoints, numRuns] size
    data_read = reshape(data_read{1,1}, [numRuns*2, numPoints+1]);
    data_write = reshape(data_write{1,1}, [numRuns*2, numPoints+1]);
    data_read = rot90(data_read);
    data_write = rot90(data_write);
    x_val = data_read(:,1);
    for n = 1:numRuns
        data_read(:,n) = data_read(:,n*2);
        data_write(:,n) = data_write(:,n*2);
    end
    data_read(:, [numRuns + 1 : numRuns*2]) = [];
    data_write(:, [numRuns + 1 : numRuns*2]) = [];

    % organize data into 4D matrices
    for n = 1:numRuns
        index = sscanf(char(titles(n)), '%d');
        master_data_read(index(1), index(2), index(3), :) = data_read(:,n);
        master_data_write(index(1), index(2), index(3), :) = data_write(:,n);
    end

    % data is now parseable with sizes of transistors
    % plot data and read inverse
    for i = pmos
        for j = nmos
            for k = access
                for n = 1:numPoints+1
                    plot_data_read(n,1) = master_data_read(i,j,k,n);
                    plot_data_write(n,1) = master_data_write(i,j,k,n);
                end
                [x, y, w, h] = inscribed_square_write(x_val, plot_data_read, ...

```

```

        plot_data_write);
    snm_write(i,j,k) = w;

    [x, y, w, h] = inscribed_square(x_val, plot_data_read);
    snm_read(i,j,k) = w;
end
end
end

```

## A.1.2 Grapher

```

% Cadence .matlab grapher for static noise margin analysis
% Kyle Owen - 9 July 2016

% list sizes to compare, aligned vertically
nmos = [3 3 5];
pmos = [2 3 4];
access = [2 2 2];

numPoints = 400;    % number of data points per graph
numRuns = 192;     % total number of sweeps

% plot data and read inverse
for n = 1:size(nmos,2)
    i = pmos(n);
    j = nmos(n);
    k = access(n);
    for m = 1:numPoints+1
        plot_data_read(m,1) = master_data_read(i,j,k,m);
        plot_data_write(m,1) = master_data_write(i,j,k,m);
    end
    [x, y, w, h] = ...
        inscribed_square_write(x_val,plot_data_read,plot_data_write);
    figure(1);
    clf;
    plot(x_val(:),plot_data_read(:),x_val(:),plot_data_write(:), ...
        'LineWidth',3);
    set(gca, 'FontSize', 16);
    hold on;
    rectangle('Position', [x,y,w,h], 'EdgeColor','k', 'LineWidth', 2);
    xlabel('Input Voltage (V)', 'FontSize',18, 'FontName', 'Arial');
    ylabel('Output Voltage (V)', 'FontSize',18, 'FontName', 'Arial');
    legend({'Left inverter', 'Right inverter'}, 'FontSize',18, ...
        'FontName', 'Arial');

```

```

legend('boxoff');
title('Write Static Noise Margin','FontSize',20,'FontName','Arial');
text(x+(w/2), y+(h/2), {'SNM:', [num2str(round(w*1000)) ' mV']}, ...
     'FontSize',16,'FontName','Arial','HorizontalAlignment','center');
str = sprintf('NFET: \t\t %d fins \n' ...
             'PFET: \t\t %d fins \nAccess: \t %d fins', ...
             nmos(n), pmos(n), access(n));
text(.55, .55, str, 'FontSize',18,'FontName','Arial');
set(1, 'Position', [100, 100, 800, 800]);
set(1, 'PaperPositionMode', 'auto');
print([num2str(nmos(n)) '_' num2str(pmos(n)) ...
      '_' num2str(access(n)) '_write'], '-depsc', '-fillpage');

[x, y, w, h] = inscribed_square(x_val, plot_data_read);
figure(2);
clf;
plot(x_val(:),plot_data_read(:),plot_data_read(:),x_val(:), ...
     'LineWidth', 3);
set(gca, 'FontSize', 16);
hold on;
rectangle('Position', [x,y,w,h], 'EdgeColor','k', 'LineWidth', 2);
xlabel('Input Voltage (V)','FontSize',18,'FontName','Arial');
ylabel('Output Voltage (V)','FontSize',18,'FontName','Arial');
legend({'Left inverter','Right inverter'],'FontSize',18, ...
      'FontName','Arial');
legend('boxoff');
title('Read Static Noise Margin','FontSize',20,'FontName','Arial');
text(x+(w/2), y+(h/2), {'SNM:', [num2str(round(w*1000)) ' mV']}, ...
     'FontSize',16,'FontName','Arial','HorizontalAlignment','center');
text(.55, .55, str, 'FontSize',18,'FontName','Arial');
set(2, 'Position', [100, 100, 800, 800]);
set(2, 'PaperPositionMode', 'auto');
print([num2str(nmos(n)) '_' num2str(pmos(n)) ...
      '_' num2str(access(n)) '_read'], '-depsc', '-fillpage');
end

```

### A.1.3 Analyzer

```

% Cadence .matlab tool for static noise margin analysis
% Kyle Owen - 9 July 2016

```

```

% for comparing SNM, do you favor read over write, and by how much?
read_percent_over_write = 0;

```

```

nmos = [3];
pmos = [3];
access = [2];

numPoints = 400;    % number of data points per graph
numRuns = 192;     % total number of sweeps

snm_read_normal=snm_read/(max(snm_read(:)));
snm_write_normal=snm_write/(max(snm_write(:)));

for n=access
    figure(3);
    subplot(2,2,n-1);
    surf(snm_read_normal(pmos,nmos,n));
    hold on;
    surf(snm_write_normal(pmos,nmos,n));
    title(['Access Transistor with ' num2str(n) ' Fins']);
    zlabel('Normalized SNM');
    xlabel('PMOS Fins');
    ylabel('NMOS Fins');
end

for i = pmos
    for j = nmos
        for k = access
            diff(i - pmos(1) + 1, j - nmos(1) + 1, k - access(1) + 1) = ...
                abs(((1 - read_percent_over_write) * snm_read(i,j,k)) - ...
                    snm_write(i,j,k));
        end
    end
end

[M,I] = min(diff(:));
[x,y,z] = ind2sub(size(diff),I);
disp(['Read percentage over write: ' ...
    num2str(read_percent_over_write*100) '%']);
disp(['PMOS: ' num2str(x+pmos(1) - 1)]);
disp(['NMOS: ' num2str(y+nmos(1) - 1)]);
disp(['Access: ' num2str(z+access(1) - 1)]);

```

#### A.1.4 Inscribed Square - Read

```
function [x, y, w, h] = inscribed_square(dataSet1,dataSet2)
```

```

if ~isequal(size(dataSet1), size(dataSet2))
    return
end

w = 0;
h = 0;
x = 0;
y = 0;

for idx_1 = 2:size(dataSet1,1)
    for idx_2 = size(dataSet1,1):-1:2
        a = dataSet2(idx_1);
        b = dataSet1(idx_1);
        c = dataSet2(idx_2);
        d = dataSet1(idx_2);
        if(a > d)
            if(c < b)
                temp_l = b - c;
                temp_h = a - d;
                q = abs(temp_l - temp_h);
                if(q < .001)
                    if(temp_h > h)
                        w = temp_l;
                        h = temp_h;
                        x = c;
                        y = d;
                    end
                end
            end
        end
    end
end
end
end
end
end

```

### A.1.5 Inscribed Square - Write

```

function [x, y, w, h] = inscribed_square_write(dataSet1,dataSet2,dataSet3)

if ~isequal(size(dataSet1), size(dataSet2), size(dataSet3))
    return
end

w = 0;

```

```

h = 0;
x = 0;
y = 0;

for idx_1 = 2:size(dataSet1,1)
    for idx_2 = 1:size(dataSet1,1)
        a = dataSet2(idx_1); %voltage of high
        b = dataSet3(idx_2); %voltage of low
        c = dataSet1(idx_1); %x of high
        d = dataSet1(idx_2); %x of low
        if(a > b)
            if(c > d)
                temp_l = c - d;
                temp_h = a - b;
                q = abs(temp_l - temp_h);
                if(q < .001)
                    if(temp_h > h)
                        w = temp_l;
                        h = temp_h;
                        x = d;
                        y = b;
                    end
                end
            end
        end
    end
end
end
end
end

```

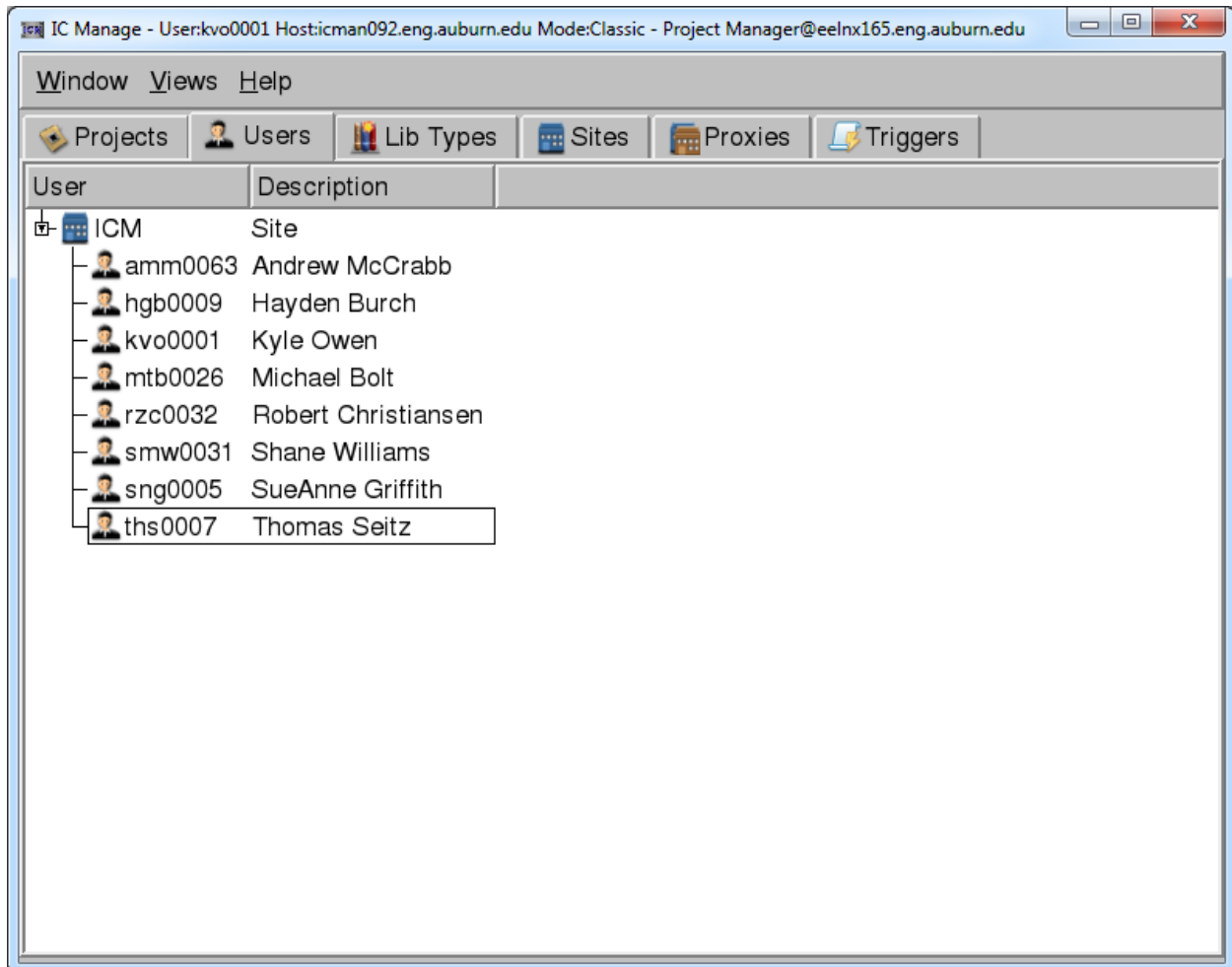
## Appendix B

### IC Manage Tutorial

IC Manage provides version control and configuration management with direct support for Cadence. With IC Manage installed and properly configured, the Cadence-provided library manager is replaced with a different one, allowing access to version control options that Cadence does not provide by default. Once IC Manage is configured, using it is fairly straightforward.

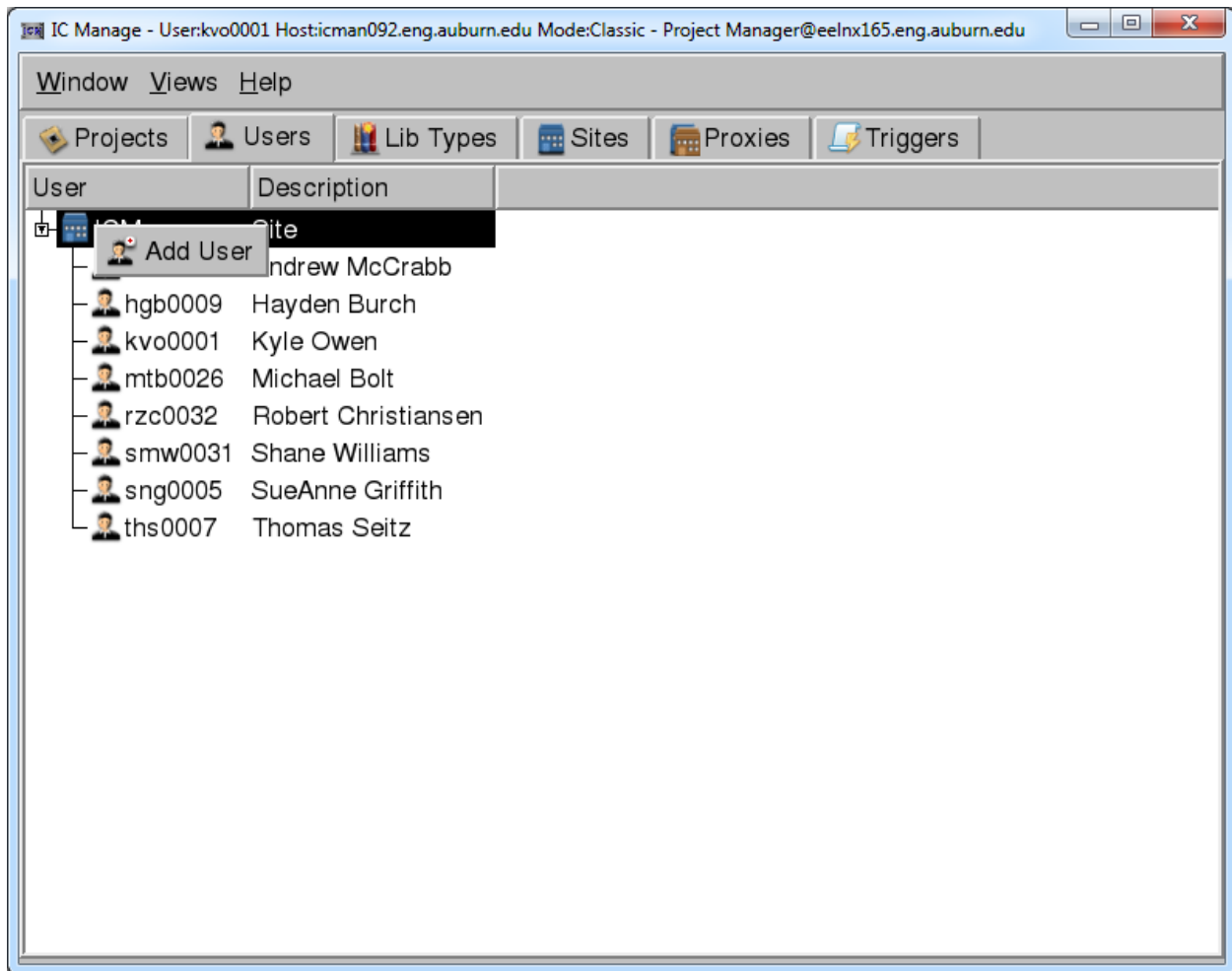
This tutorial will cover the maintenance and general usage of IC Manage how it pertains to the current setup. For more specific inquiries, it is best to consult the documentation provided by IC Manage.

## B.1 Adding a new user

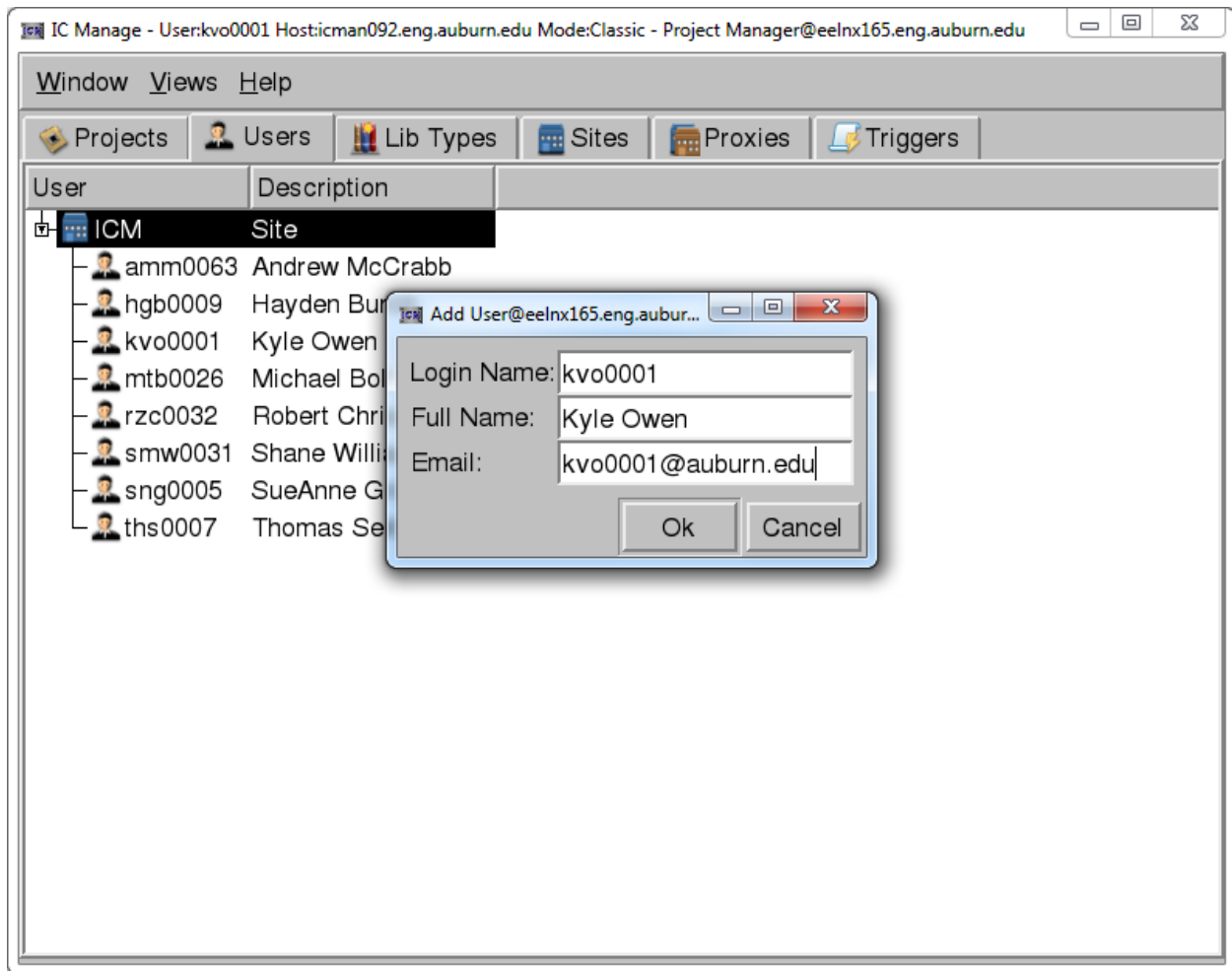


The user list can be found under **Users**.



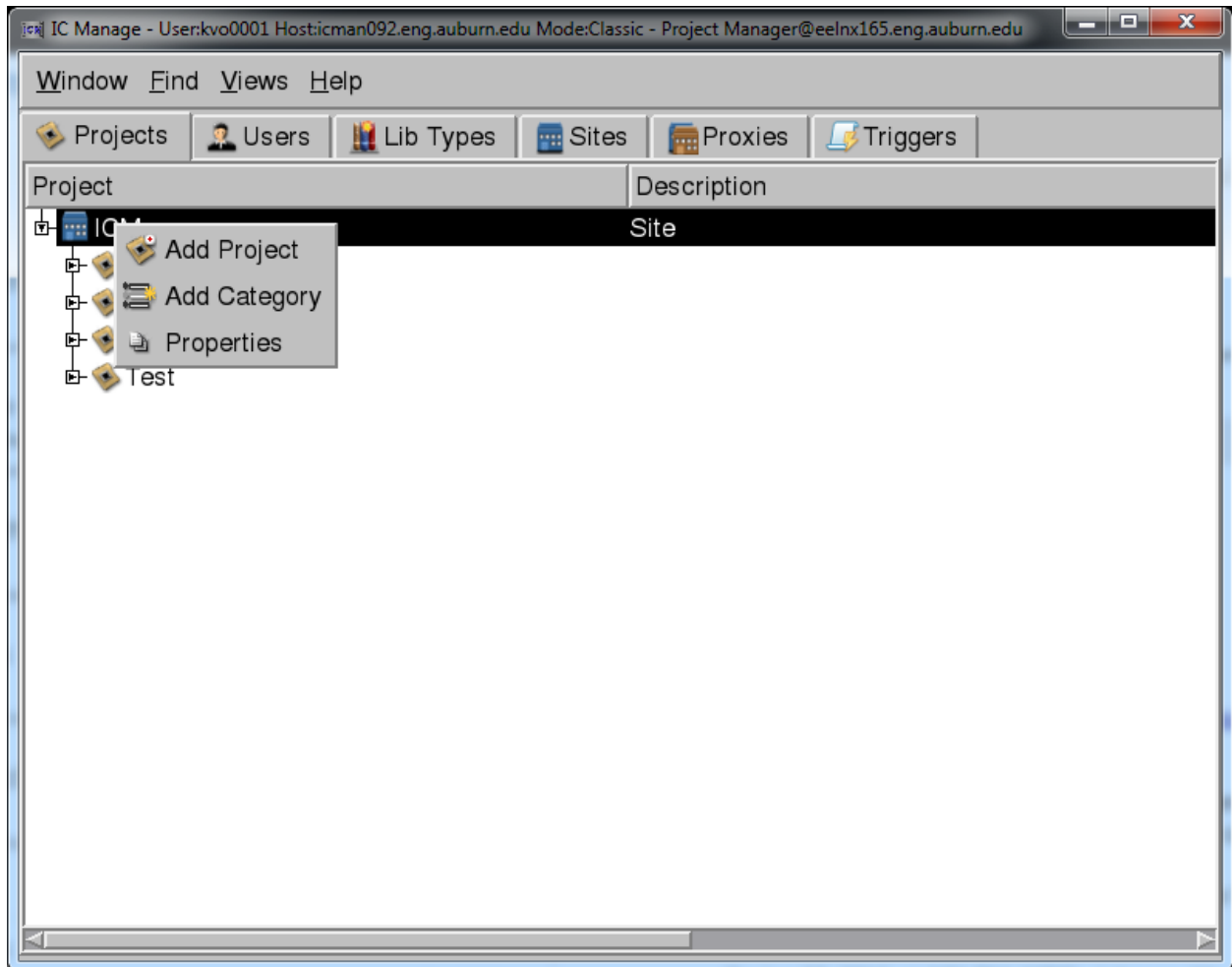


Right click on the site, then click on .

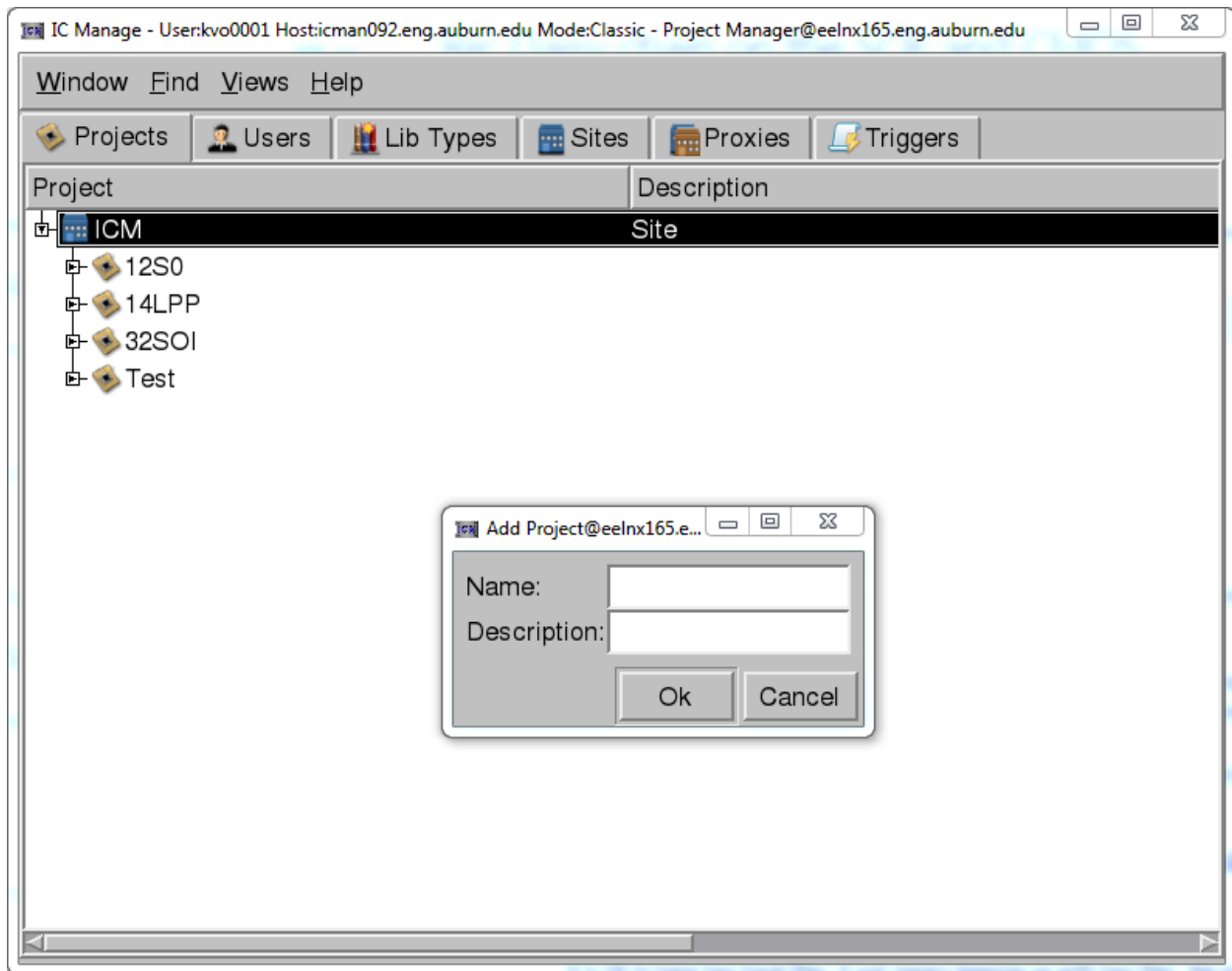


Fill in the required information. The login name must match the user name of the new user exactly. The rest of the information should be as accurate as possible.

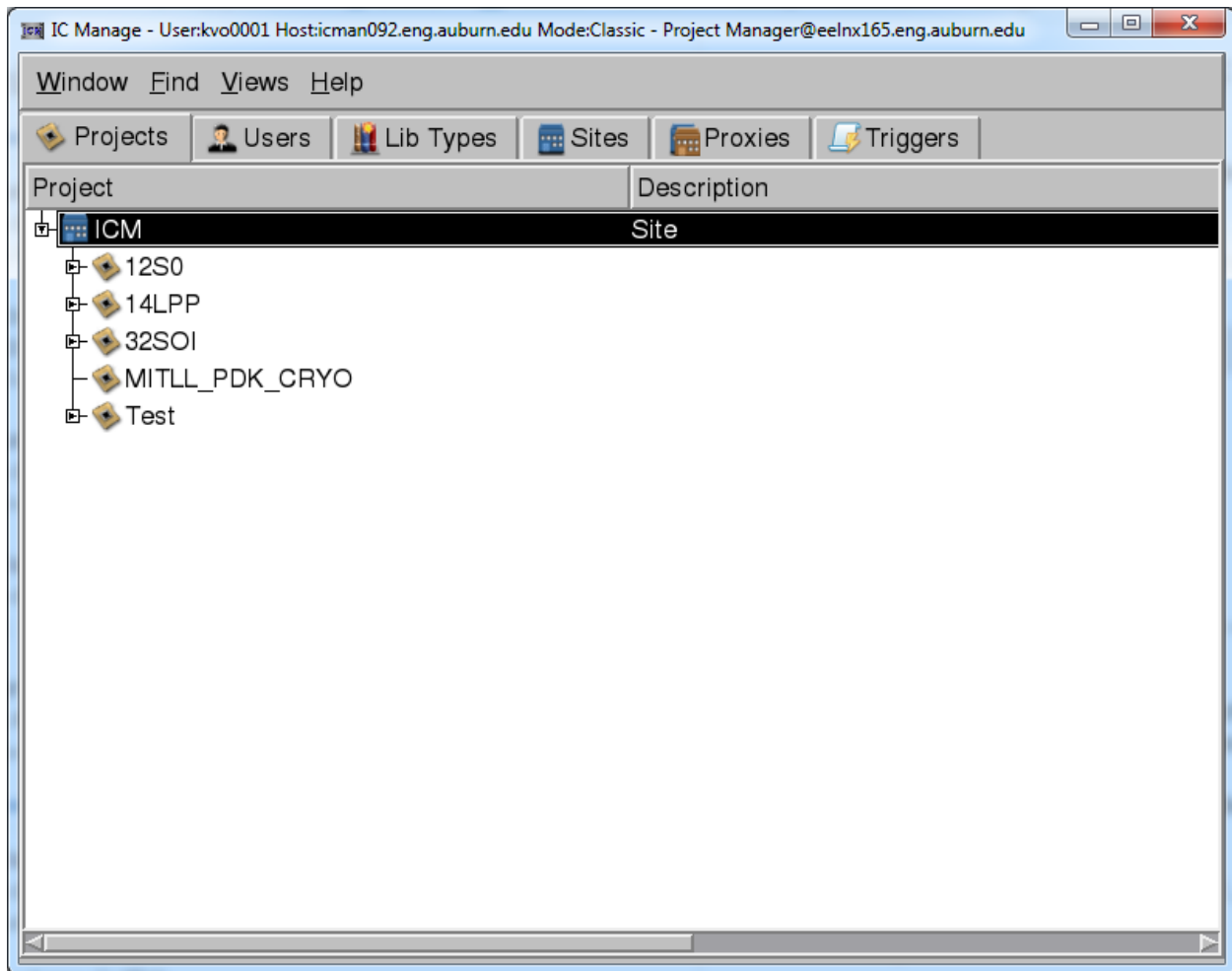
## B.2 Adding a new PDK project



Under the **Projects** tab, right click on the site in which you want to add a new project to. For our case, there is only one site, **ICM**. Then, click on **Add Project**.

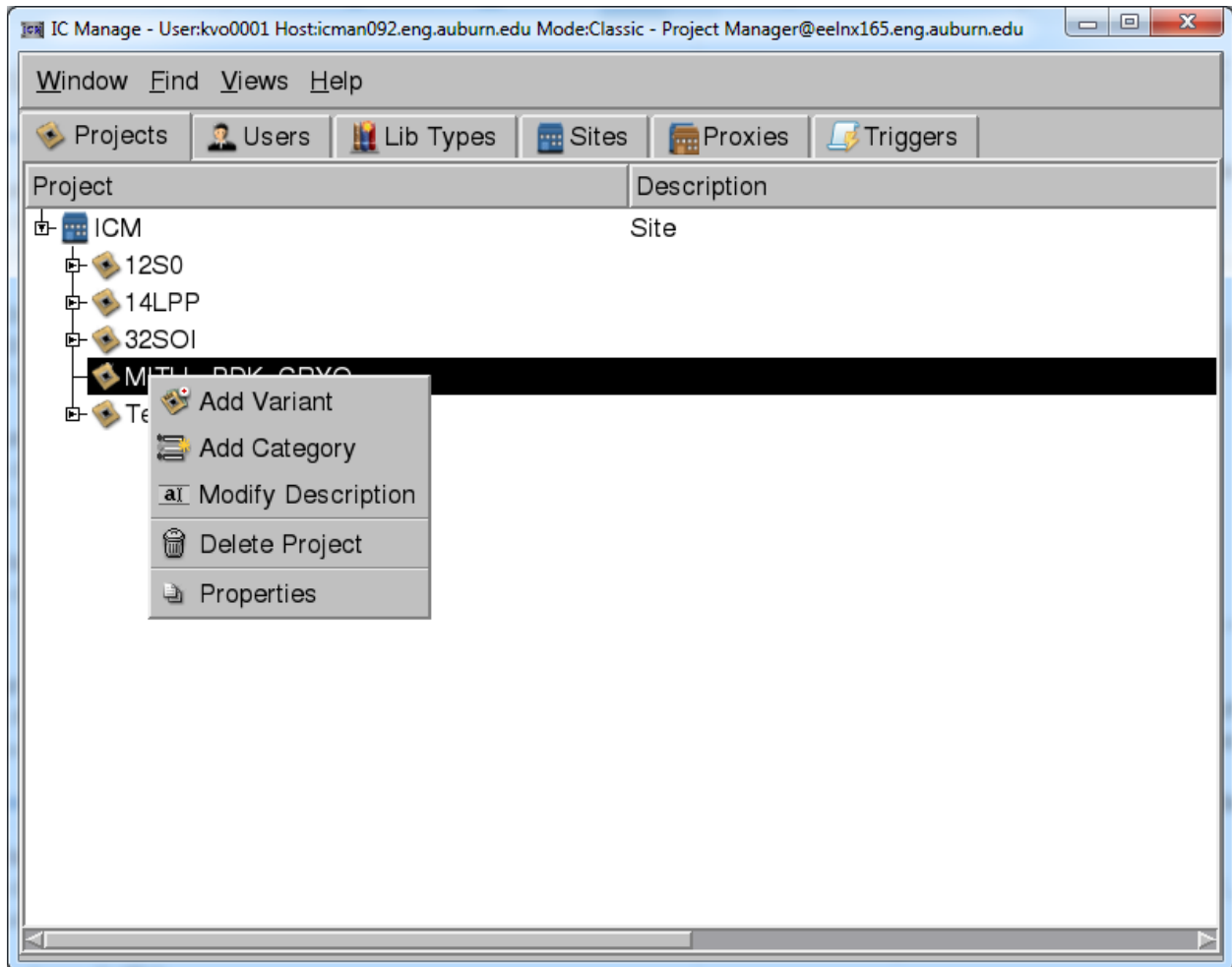


Next, give the project a name. Typically, this will be the name of the PDK you wish to use. For this example, we are adding support for the MITLL\_PDK\_CRYO kit.

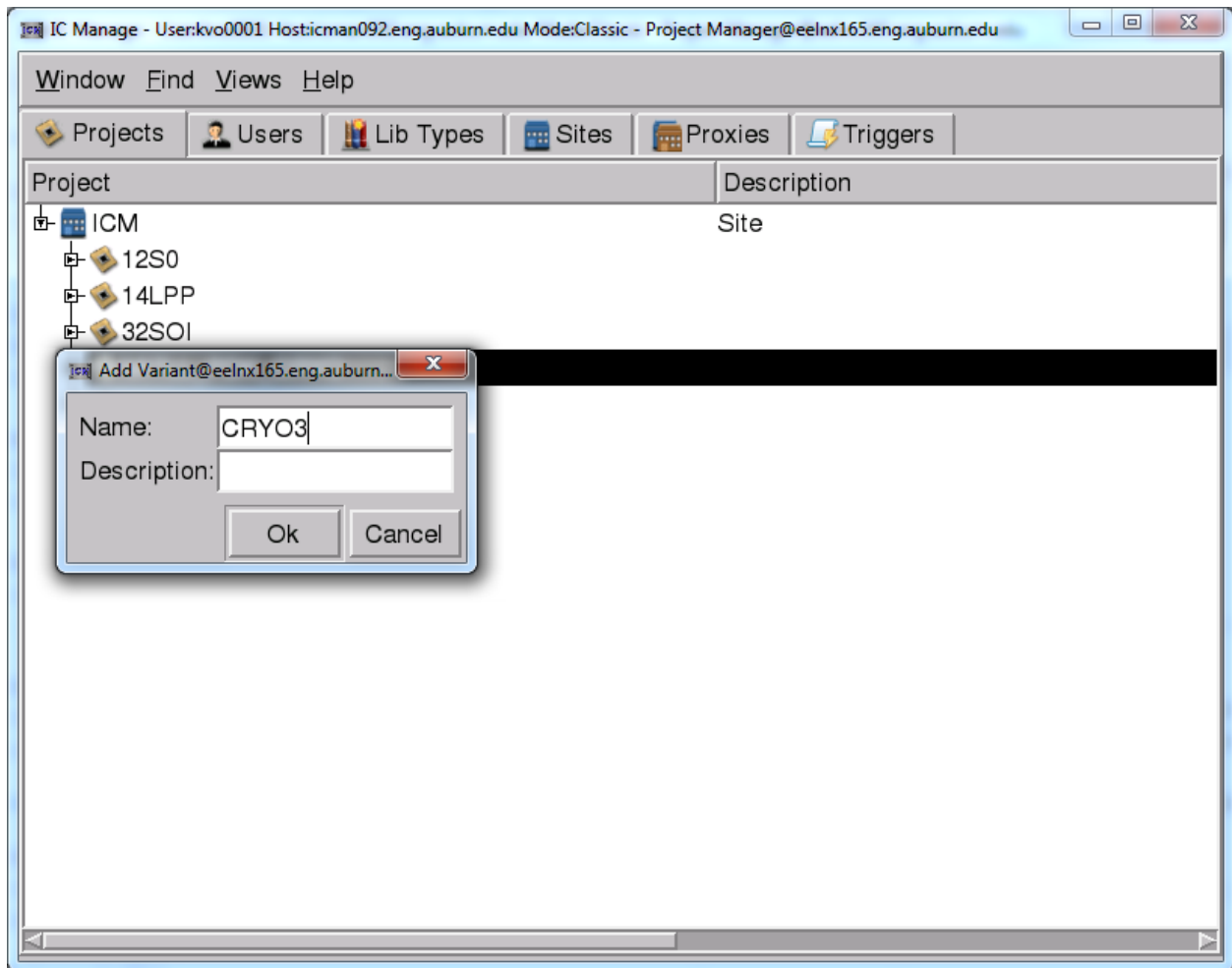


The new project should now appear in the list.

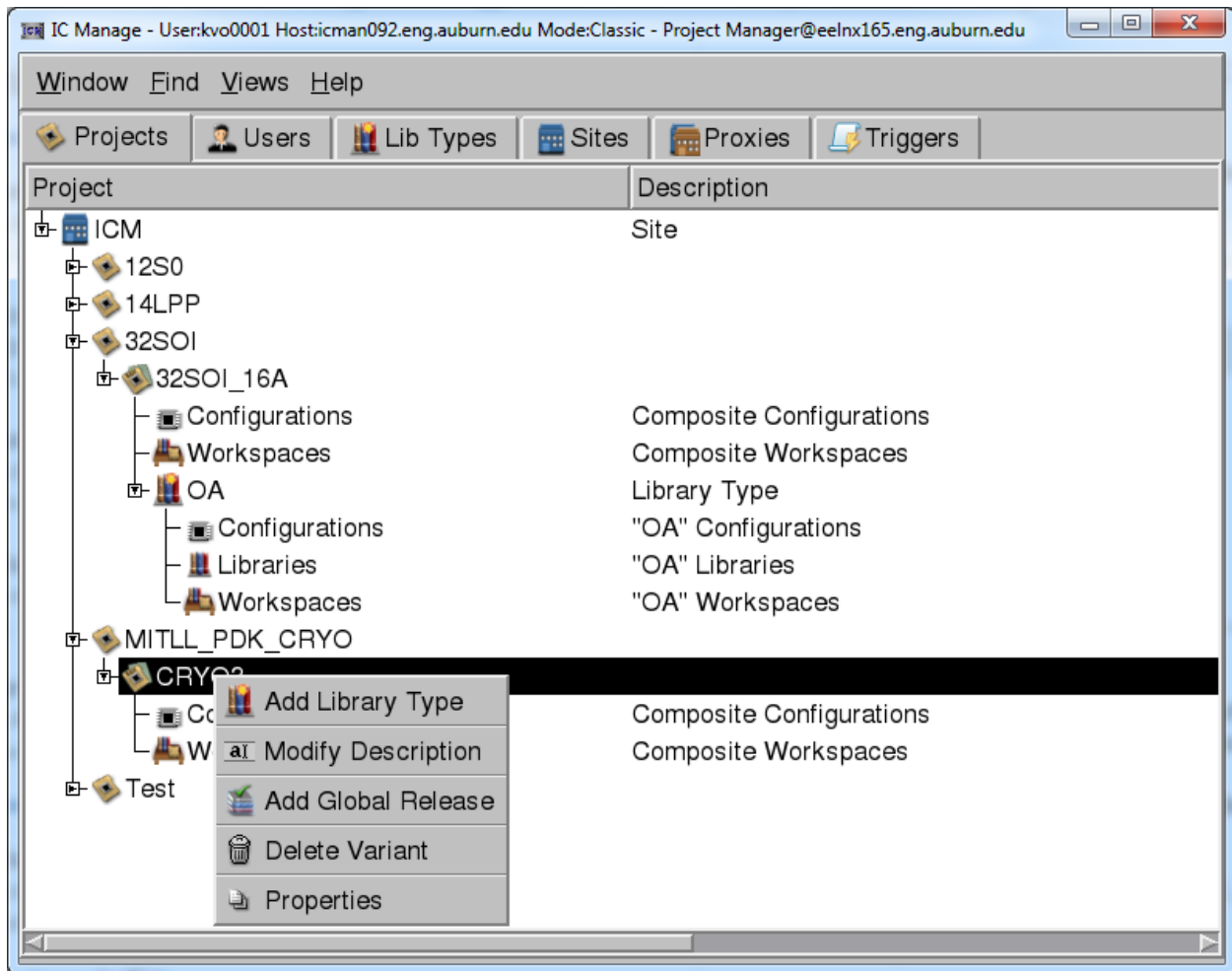
### B.3 Add variant to existing project



Right click on the newly created project to add a variant. This will typically be the name of a particular tapeout. Click on **Add Variant**.

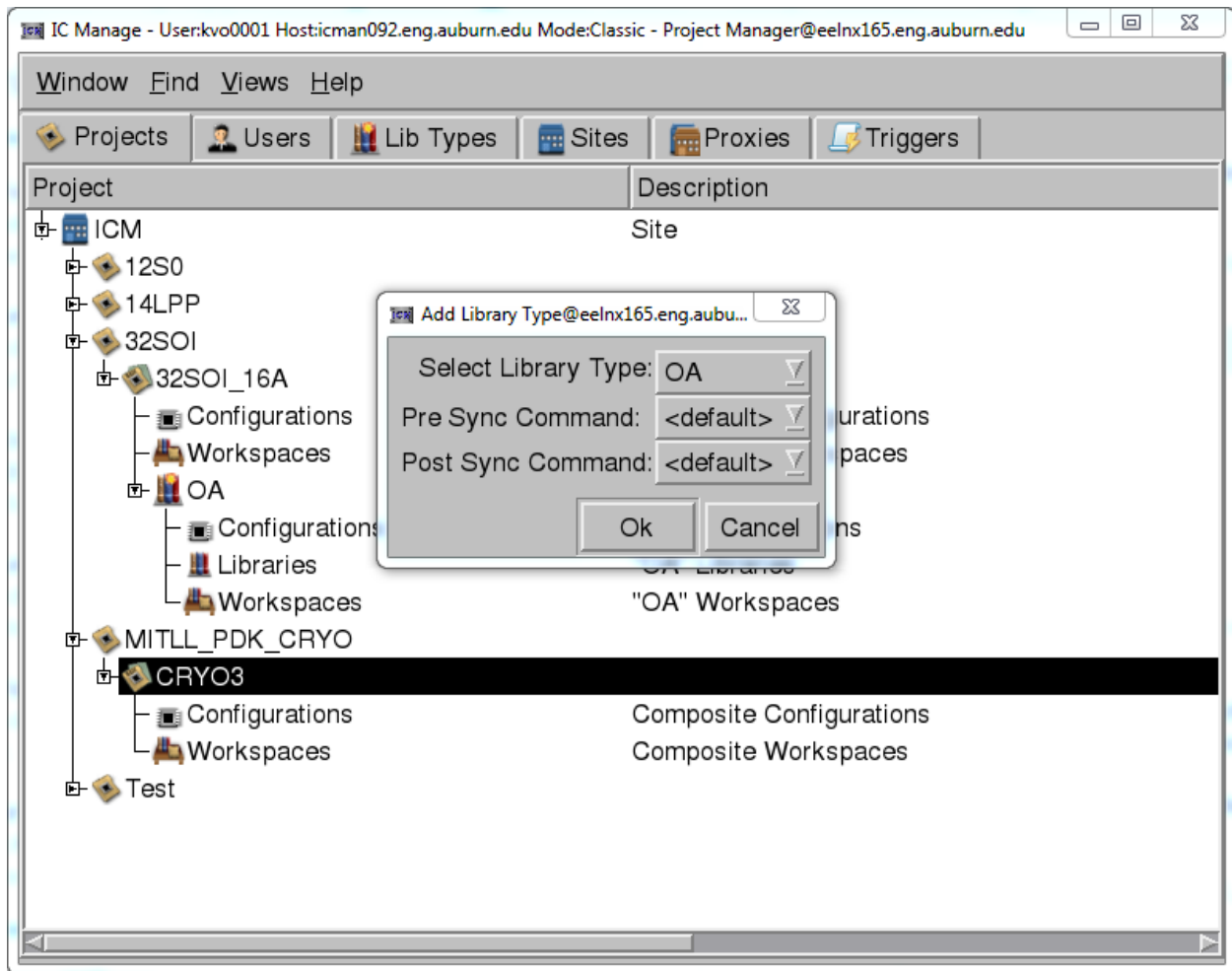


In this example, this is for a tapeout called CRY03, but other options may look like 32SOI\_16A, for example.



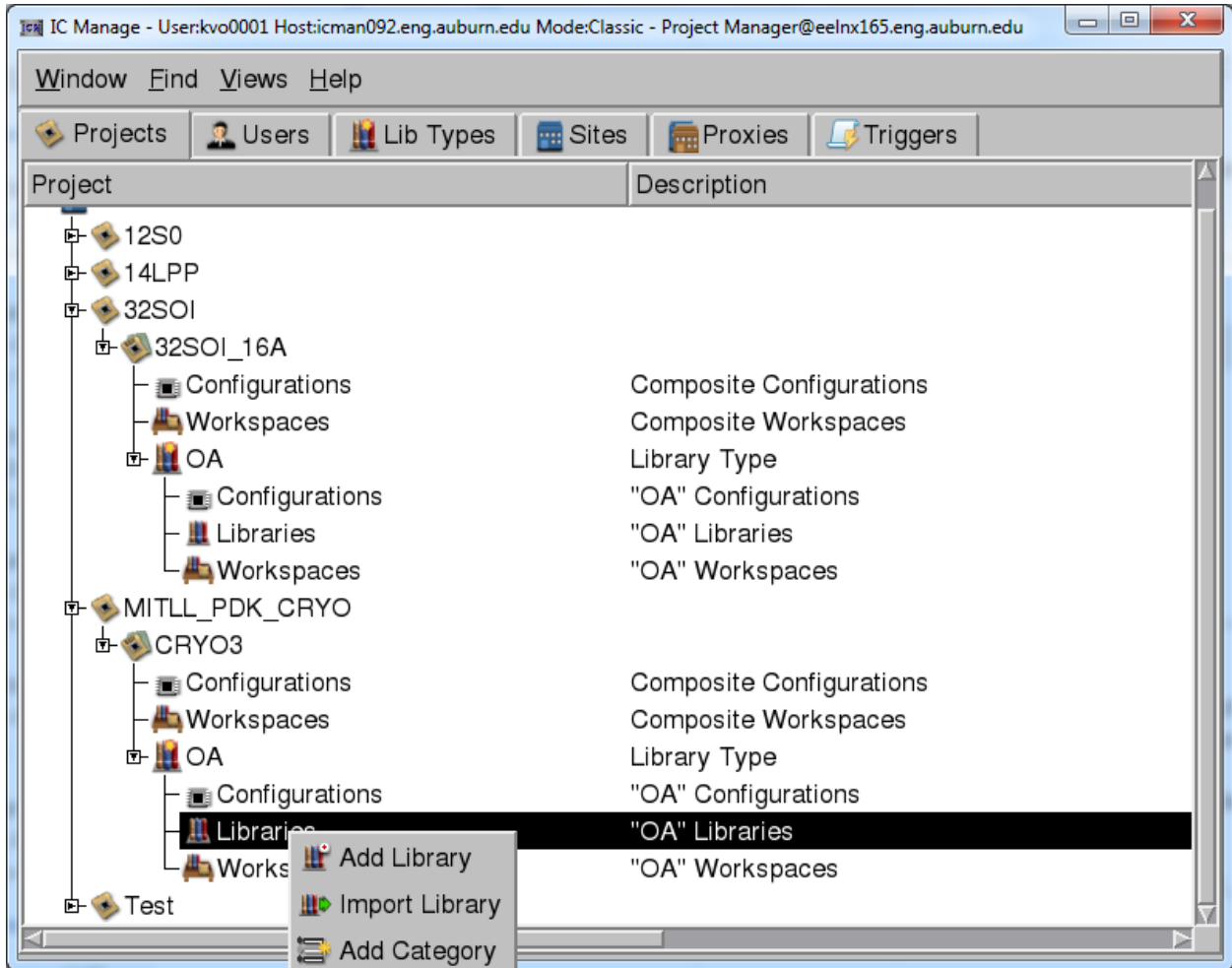
Next, right click on the newly created variant to add a library type. Click on **Add Library Type**. The default library types have already been added.



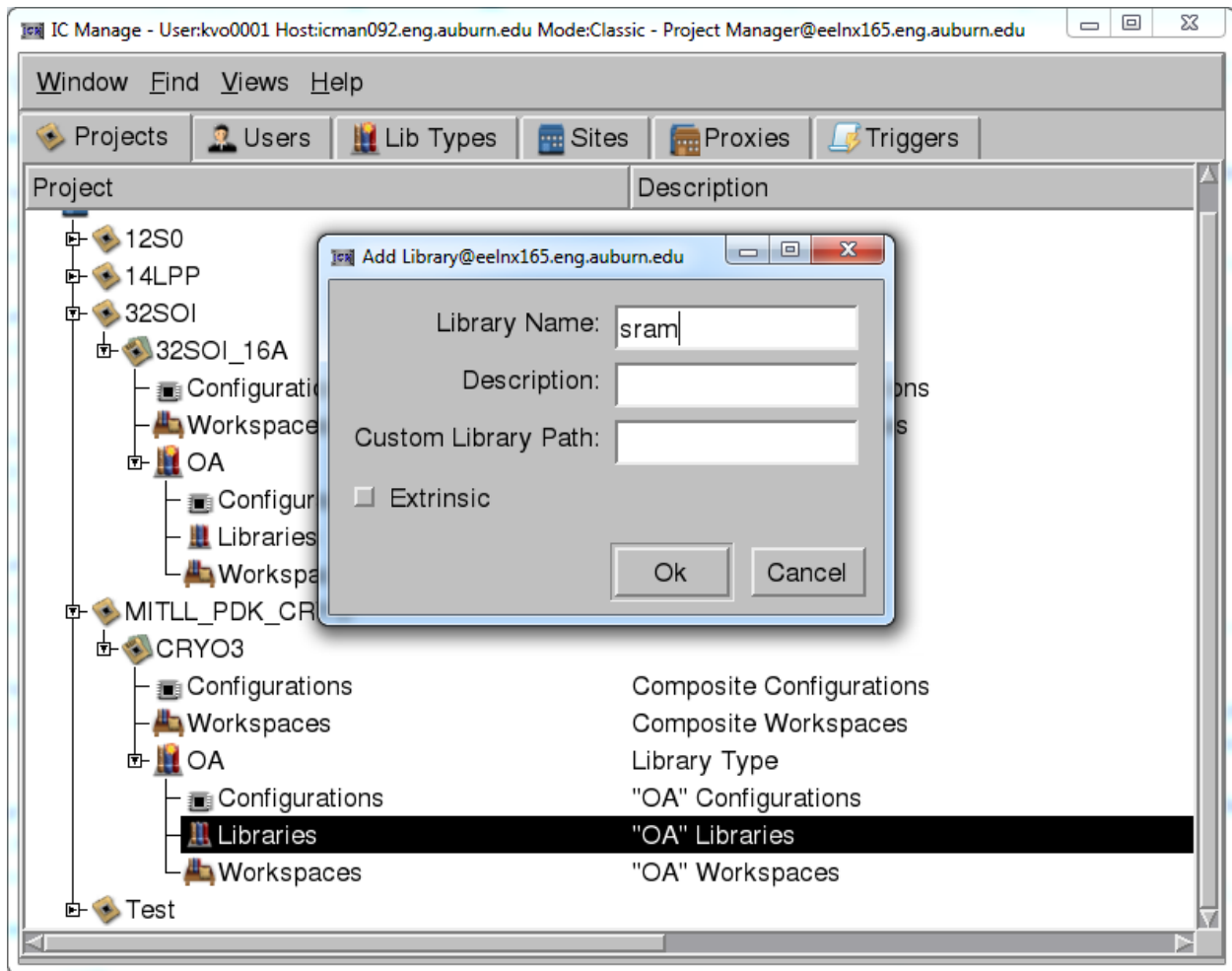


The default library types are `OA` and `OA_adv`. For nodes above 14 nm, use `OA`.

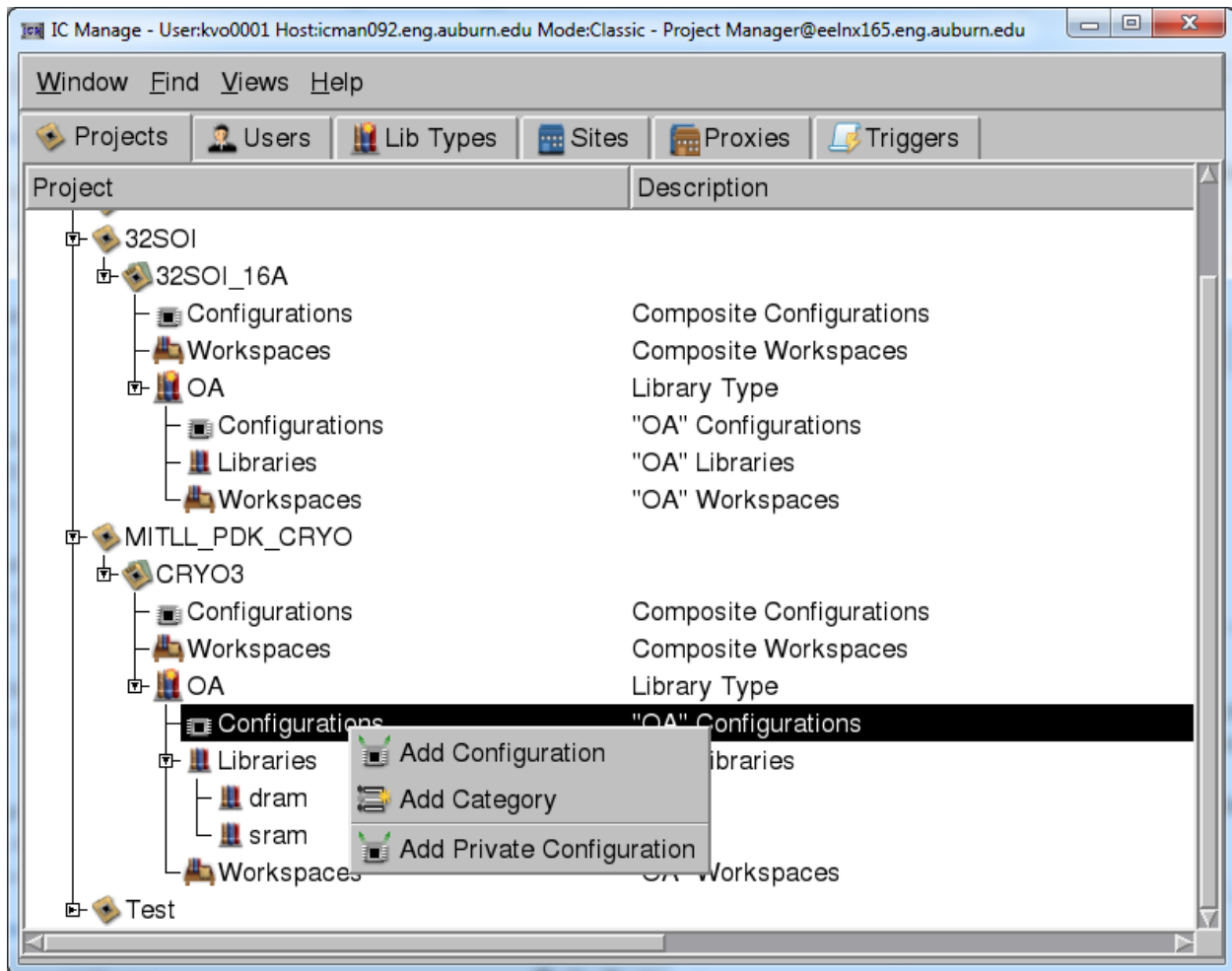
## B.4 Add libraries to existing variant



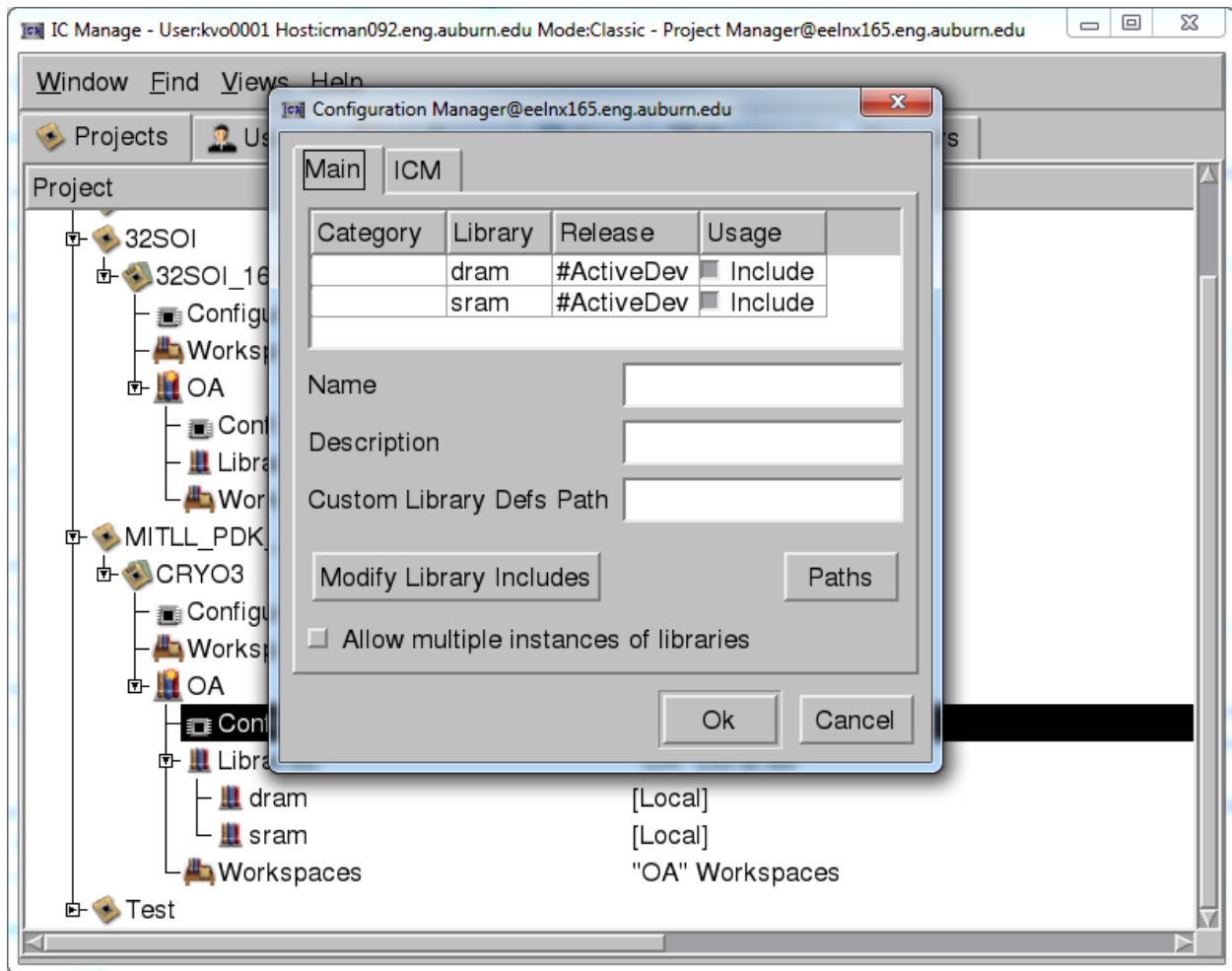
Now that the library type has been added, right click on it and click **Add Library**.



You can add multiple managed libraries at this point. It is recommended to have separate libraries per chip, to keep files separate. Managed libraries should generally only be created for working libraries, such that version control can be used. There is no need to create libraries here for read-only libraries such as vendor-provided standard cell libraries. These will be added later. It is worth noting that you can add more libraries later if needed. Also, names should be descriptive enough so that a description would not be necessary.



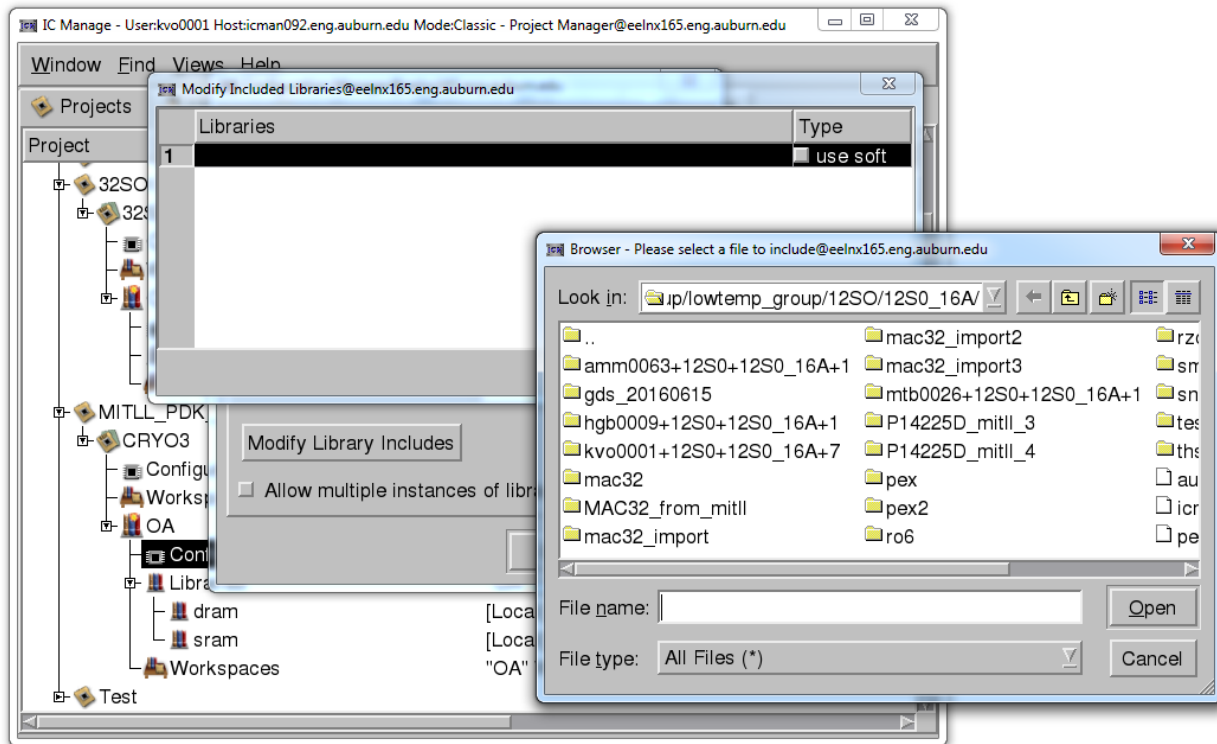
Once you have added the required libraries, you can create a configuration which will allow you to select which libraries will be used. Generally, one configuration should be enough, but if one group is to work on something completely separate from another, two configurations can be created to ensure that one group cannot modify the other files. Click on [Add Configuration](#).



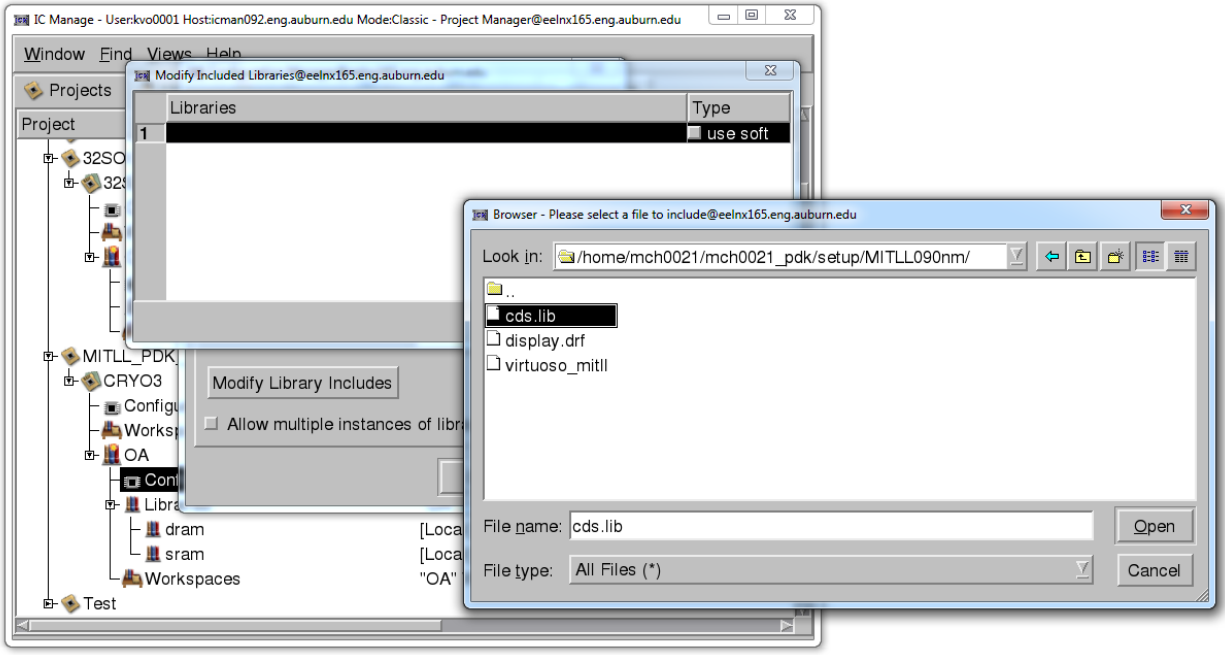
For a basic configuration, click the `Include` box next to all libraries you created. Then, click `Modify Library Includes` to select the default `cds.lib` for a specific technology.

If libraries were added after the configuration was created, these will need to be explicitly added. You must have administrator privileges to remove included libraries from a configuration.

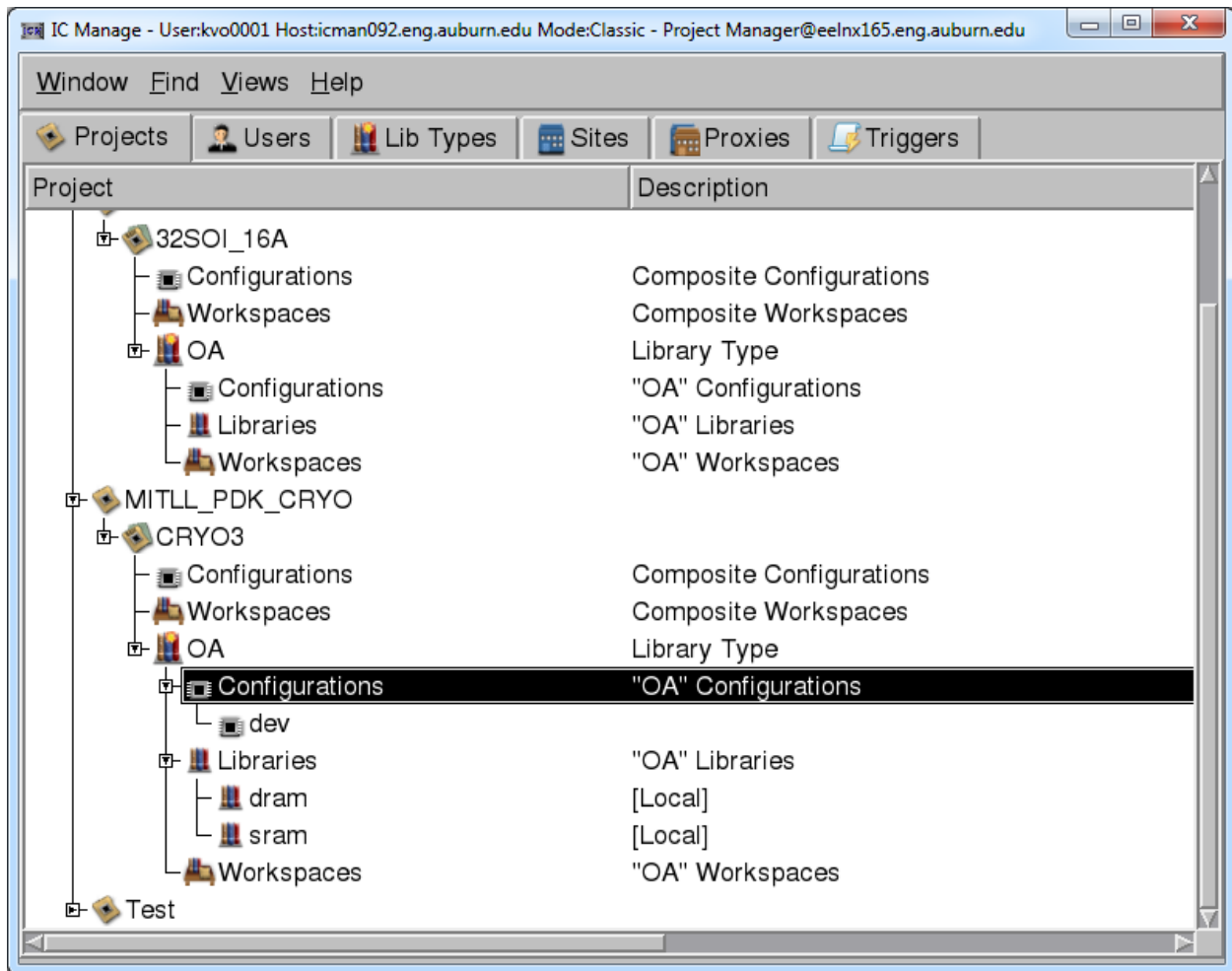
## B.5 Adding unmanaged libraries



The default location is `home > mch0021 > mch0021_pdk > setup > [technology]` for the `cds.lib` of any given technology.



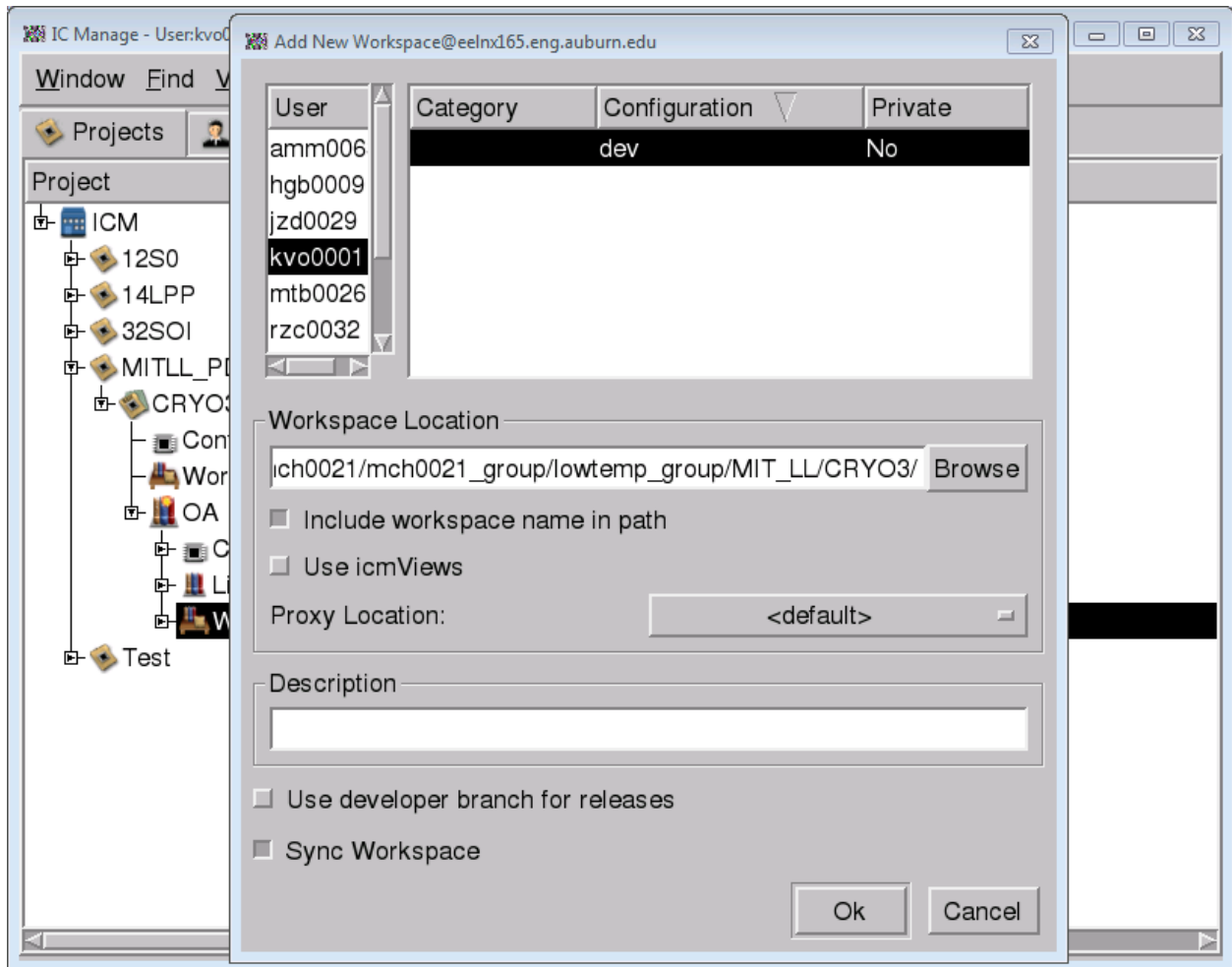
Select the appropriate `cds.lib` for the technology, and click `Open`, then `Ok`, and `Ok` again to get back to the `Project Manager` window.



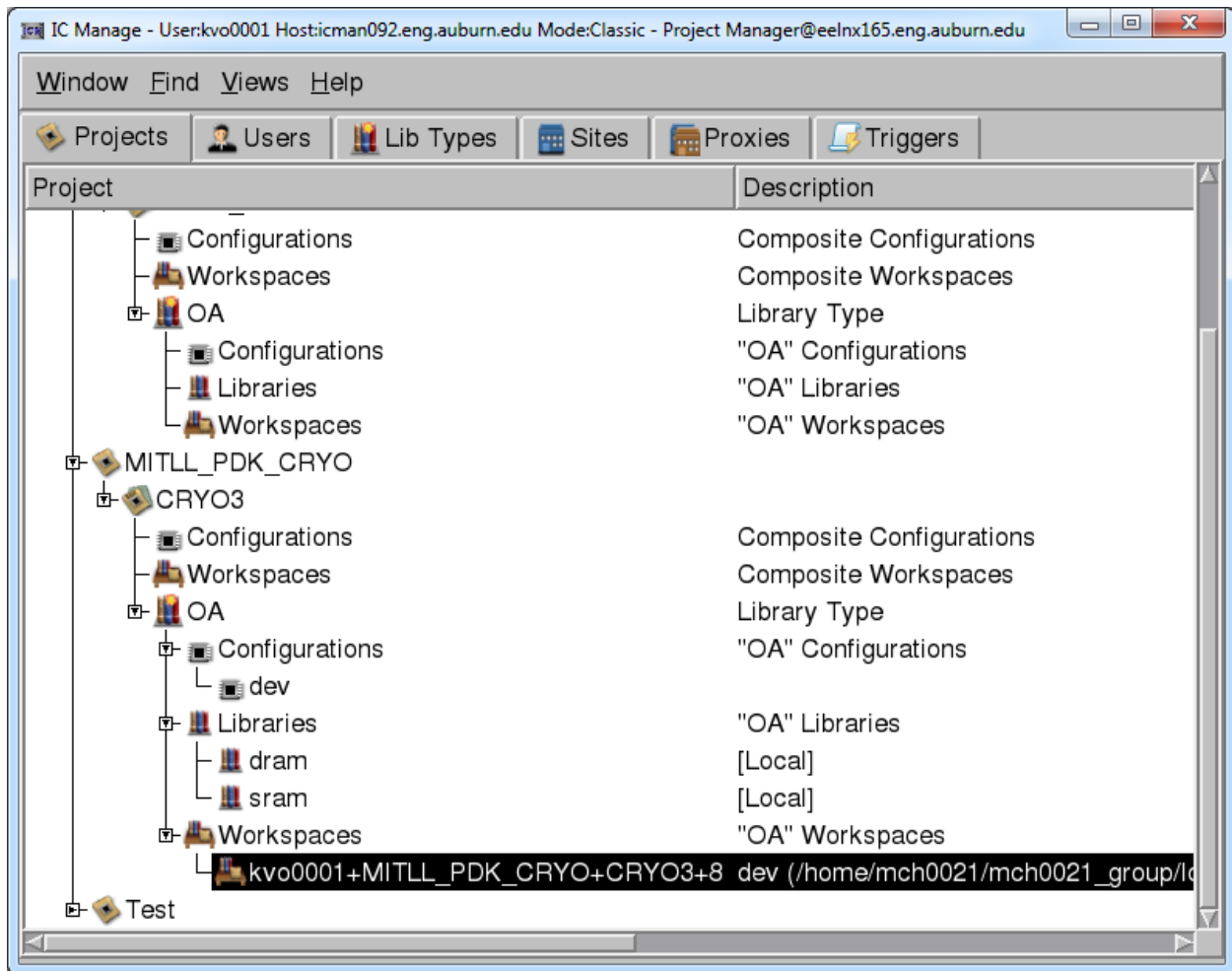
The Project Manager window should now look similar to this. Right click on the Workspaces under OA and click Add Workspace.



## B.6 Adding a new workspace



Ensure the **Workspace Location** is set to the appropriate directory. There should be a tapeout directory under the technology directory under `home ▶ mch0021 ▶ mch0021_group ▶ lowtemp_group`. It is generally a good idea to keep like workspaces in the same place so that it is easy to tell who all is working on a particular project; however, this does not impact the functionality whatsoever. Ensure that **Sync Workspace** is checked and click **Ok**.



The newly created workspace should now appear under Workspaces.

```

kvo0001@eelnx165:/home/mch0021/mch0021_group/lowtemp_group/MIT_LL/CRY03/kvo0001+MITLL_PDK_CRY0+CRY03+8$ ls -al
total 4
drwxr-s----+ 4 kvo0001 lowtemp_group  6 Jun 24 11:36 .
drwxr-s----+ 3 kvo0001 lowtemp_group  3 Jun 24 11:36 ..
-r--r-----+ 1 kvo0001 lowtemp_group 265 Jun 24 11:36 cds.libicm
drwxr-s----+ 2 kvo0001 lowtemp_group  4 Jun 24 11:36 dram
-rw-r-----+ 1 kvo0001 lowtemp_group  76 Jun 24 11:36 .icmconfig
drwxr-s----+ 2 kvo0001 lowtemp_group  4 Jun 24 11:36 sram
kvo0001@eelnx165:/home/mch0021/mch0021_group/lowtemp_group/MIT_LL/CRY03/kvo0001+MITLL_PDK_CRY0+CRY03+8$ cadence-setup
This script will set up the necessary files to run Virtuoso in the current
directory. The available processes are:

1) 8RF
2) 9HP
3) 10RF
4) 12S0
5) 32S0I
6) 8XP
7) 8HP
8) 14S0I
9) 14S0I_foundation_flow
10) MITLL090nm
11) 14LPP
12) 12S0IBM

```

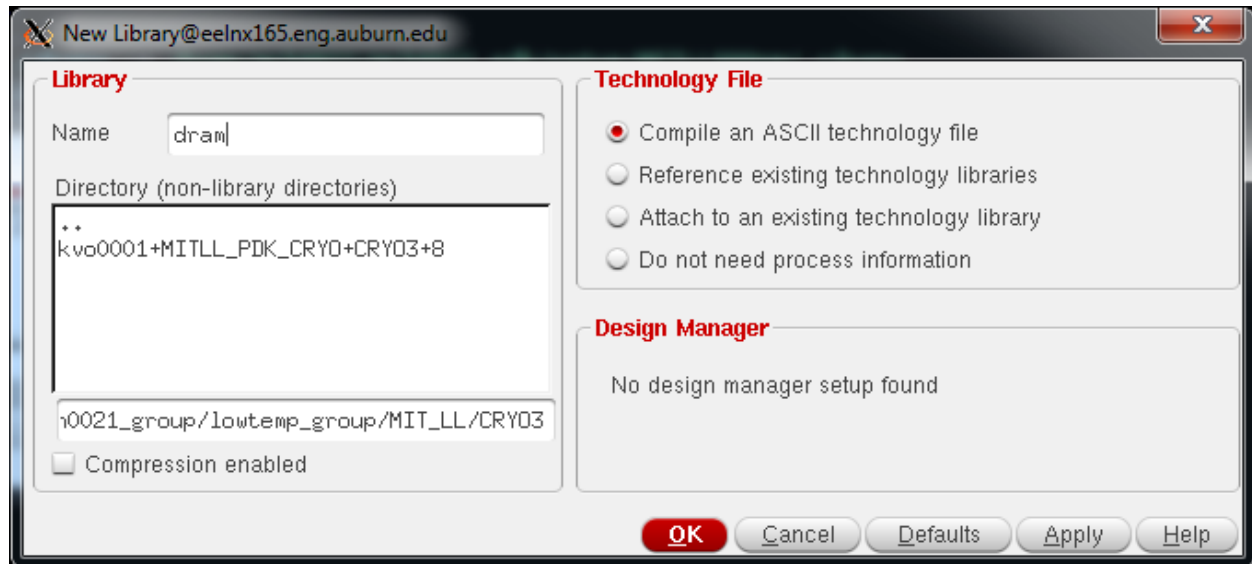
```

Enter the number corresponding to the desired process or q to exit: 10
Do you wish to use IC Manage to manage this project directory? (y/n) y
kvo0001@eelnx165:/home/mch0021/mch0021_group/lowtemp_group/MIT_LL/CRY03/kvo0001+MITLL_PDK_CRY0+CRY03+8$ ls -al
total 8
drwxr-s----+ 4 kvo0001 lowtemp_group  10 Jun 24 11:39 .
drwxr-s----+ 3 kvo0001 lowtemp_group   3 Jun 24 11:36 ..
lrwxrwxrwx. 1 kvo0001 lowtemp_group  50 Jun 24 11:39 .cdsenv -> /home/mch0021/mch0021_pdk/setup/MITLL090nm/.cdsenv
lrwxrwxrwx. 1 kvo0001 lowtemp_group  51 Jun 24 11:39 .cdsinit -> /home/mch0021/mch0021_pdk/setup/MITLL090nm/.cdsinit
-r--r-----+ 1 kvo0001 lowtemp_group 265 Jun 24 11:36 cds.libicm
lrwxrwxrwx. 1 kvo0001 lowtemp_group  54 Jun 24 11:39 display.drf -> /home/mch0021/mch0021_pdk/setup/MITLL090nm/display.drf
drwxr-s----+ 2 kvo0001 lowtemp_group  4 Jun 24 11:36 dram
-rw-r-----+ 1 kvo0001 lowtemp_group  76 Jun 24 11:36 .icmconfig
drwxr-s----+ 2 kvo0001 lowtemp_group  4 Jun 24 11:36 sram
lrwxrwxrwx. 1 kvo0001 lowtemp_group  57 Jun 24 11:39 virtuoso_mitll -> /home/mch0021/mch0021_pdk/setup/MITLL090nm/virtuoso_mitll

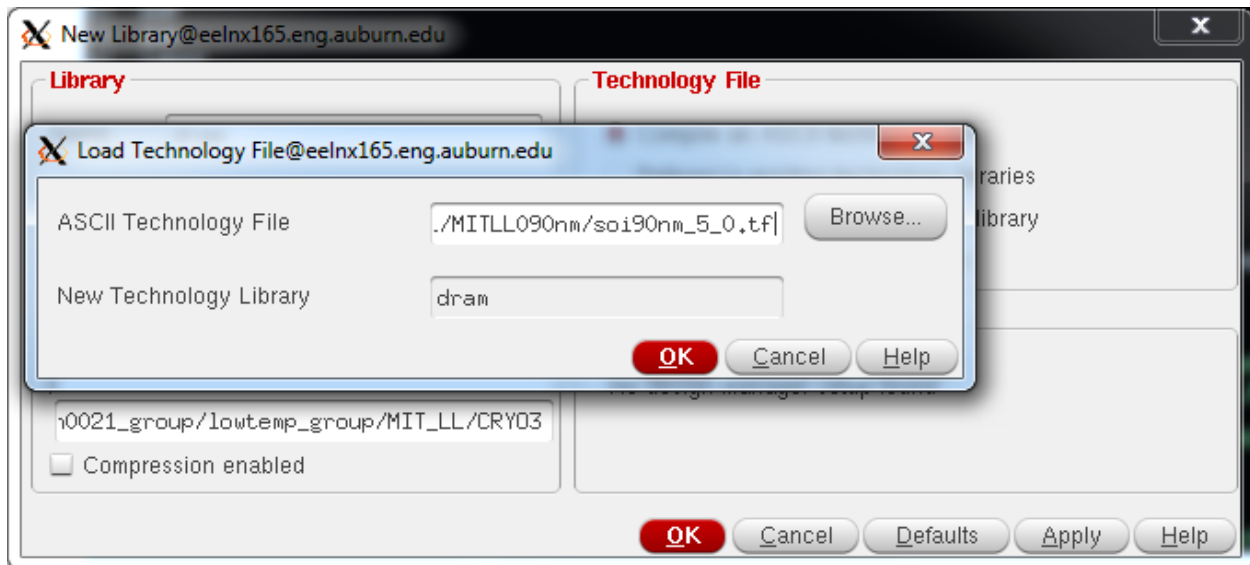
```

The next step is to return to a terminal and run the `cadence-setup` script, which will add the appropriate files to allow Virtuoso and IC Manage to run correctly. Select the desired technology, and type `y` to copy the correct IC Manage-related files. If you examine the files before and after running `cadence-setup`, your output should be fairly similar to the above.

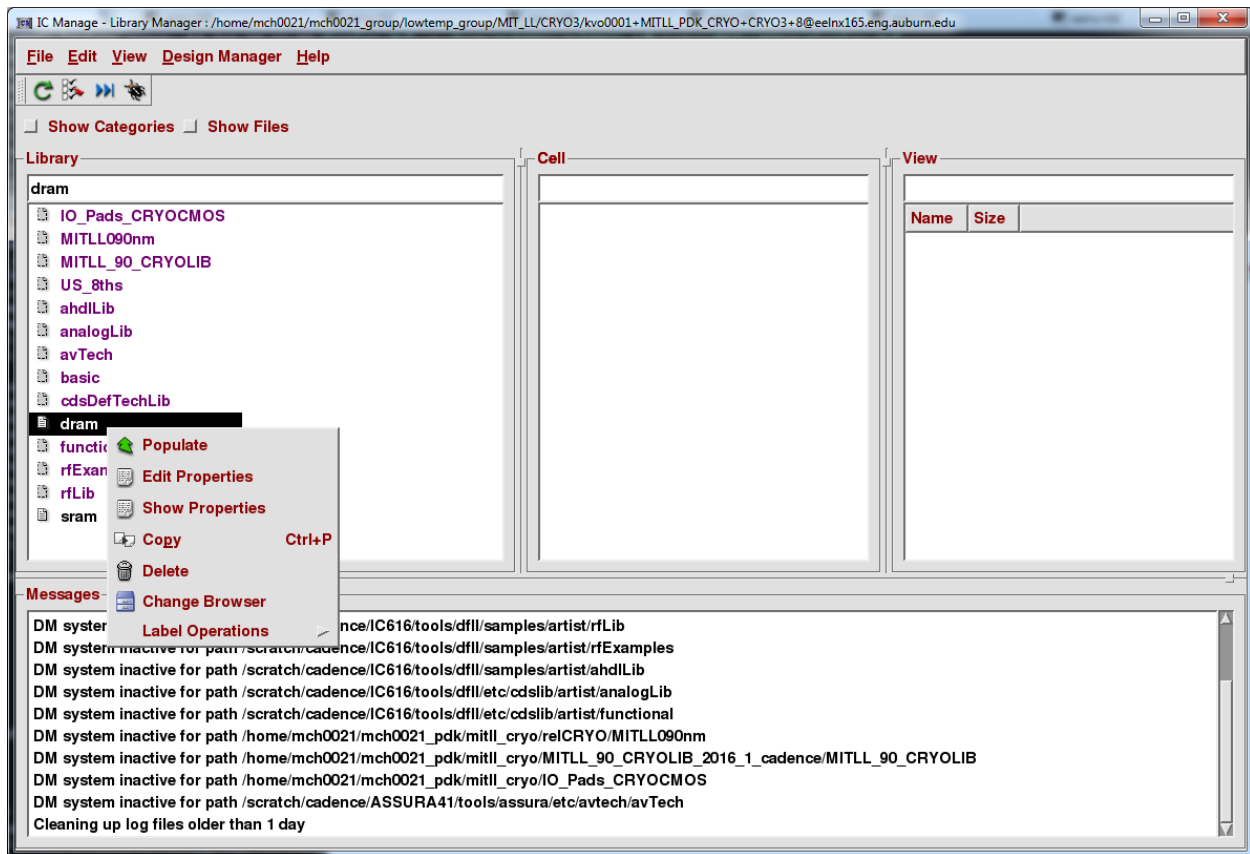
## B.7 Adding library properties



The libraries created by IC Manage only have the required IC Manage information and are otherwise empty as far as Virtuoso is concerned. In order to design in these libraries, they must refer to a technology library. There are several options here. First, if you wish to take an existing unmanaged library and make it managed, it will already have the required information, and you can skip this step. Otherwise, you will need to comment out the library by the same name you wish to add the technology file to in the `cds.lib`, and then create a new library by the same name and compile an ASCII technology file.

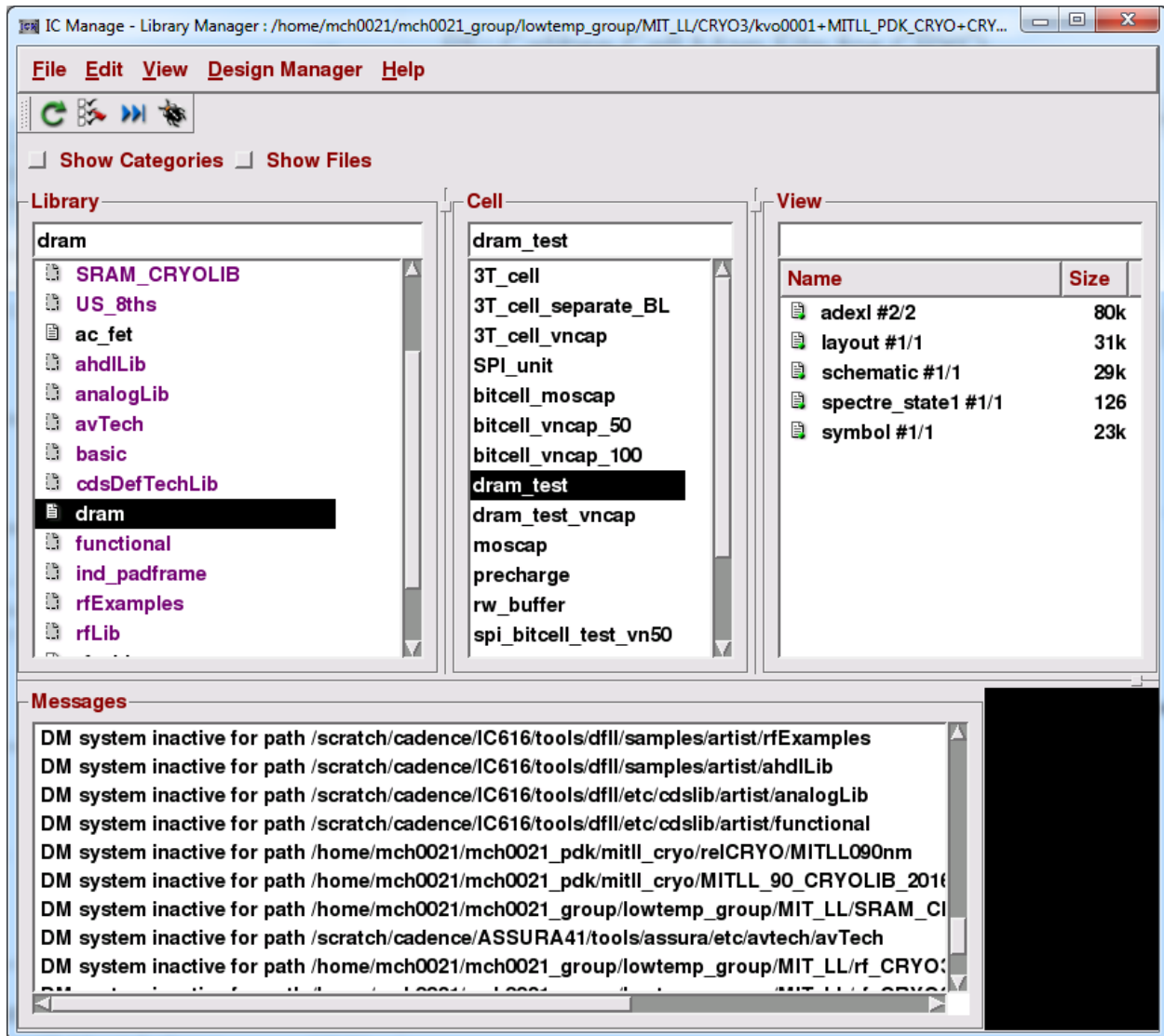


These files will generally end in `.tf` and are located under `home` ▶ `mch00021` ▶ `mch0021_pdk` ▶ `[technology]`. You must be sure to use the same technology file that the fabricator will use. Once this library is created and a technology file is compiled, delete the new library entry in your `cds.lib` so that Cadence has forgotten about it, and uncomment the library created by IC Manage.

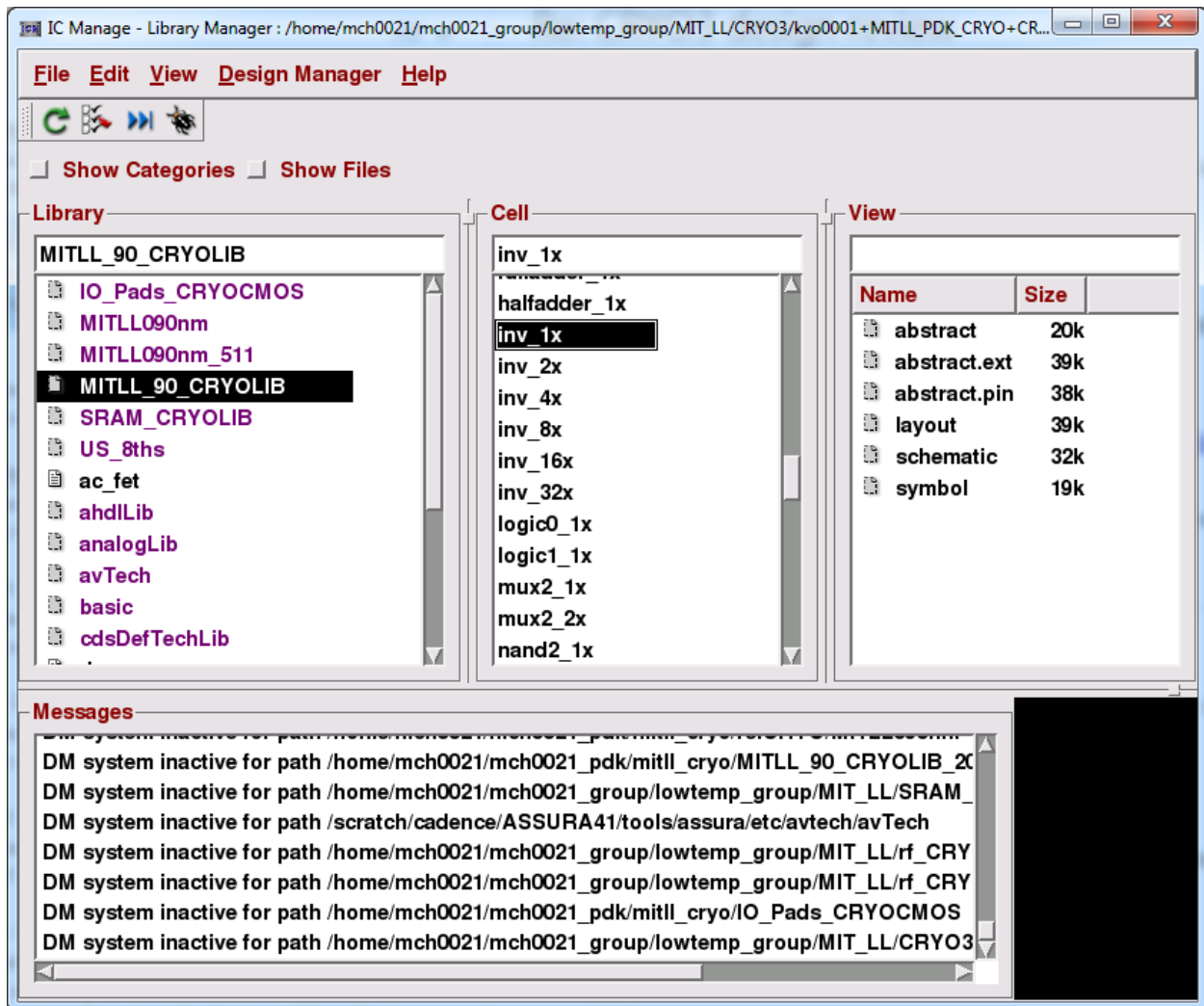


Once you have a library with the technology information loaded, right click on the managed library in the `Project Manager` and click `Populate`. Select the library you created before (and removed from `cds.lib`) to populate the managed library, resident on the IC Manage server, with the technology information. The library should still appear blank, but the technology information should be correctly loaded. If you only see the **background drawing layer** within Virtuoso when editing **layout** views from the library, you have a technology file issue that will need to be resolved.

## B.8 Managed vs. unmanaged libraries



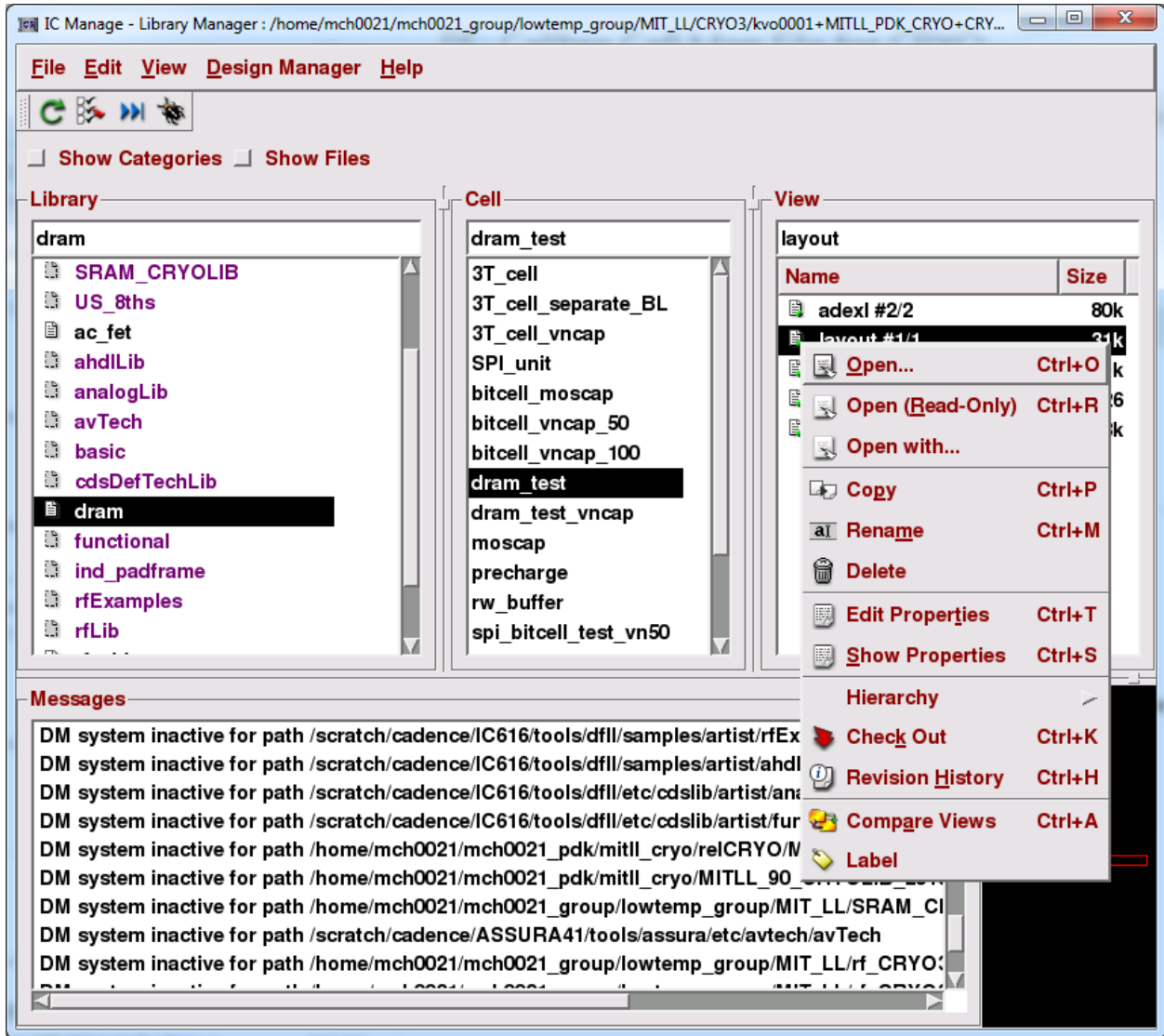
Managed libraries allow for versioning control. Views can be checked in and out. When a cell is checked out, its previous version can be opened, in read mode only, by anyone else. Managed libraries are very useful, as multiple people can work on components of the library at the same time without fear of overwriting others' work. When an error is discovered in a layout, one can revert to a previous version to fix it.



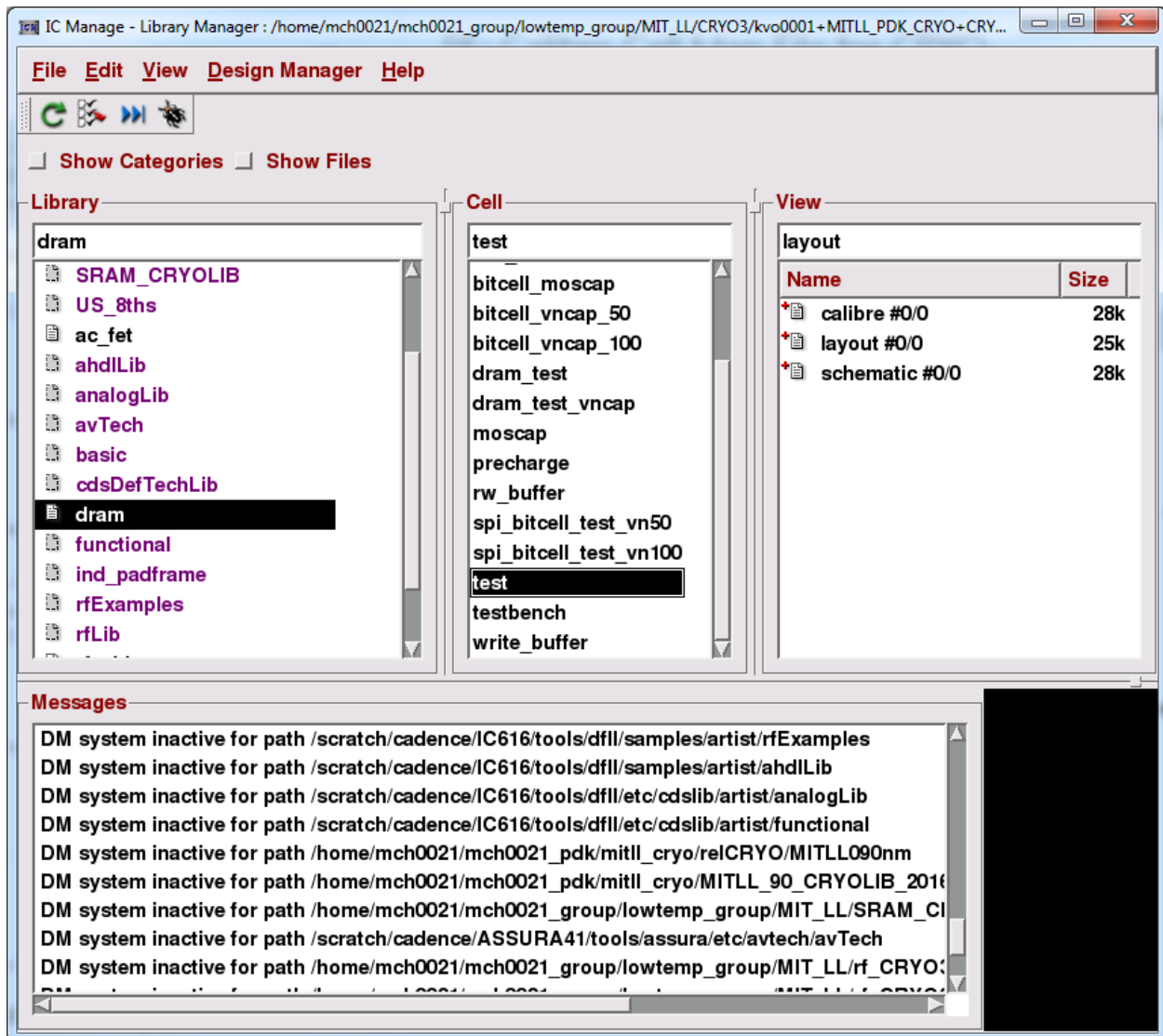
Unmanaged libraries are very useful too, as for libraries that should not change during the course of development, there is no need to take up extra space on the server. Everyone has their own separate versions of managed libraries, creating a lot more space on a network drive. Unmanaged libraries are centrally located, so everyone references the same unmanaged library. Unmanaged libraries do not have to be read-only, but that is typically the case.



## B.9 Editing a view



To open a view for edit, right click on the view you wish to edit and select `Open...`. By default, double-clicking on a view will open for read-only if the view is checked in or checked out by someone else, and will open for write if the view is checked out by you.



Newly-created views will appear with a **+** sign next to the view, and will have a version count of 0.

IC Manage - Checkout Manager@eelnx165.eng.auburn.edu

Checkouts Pending

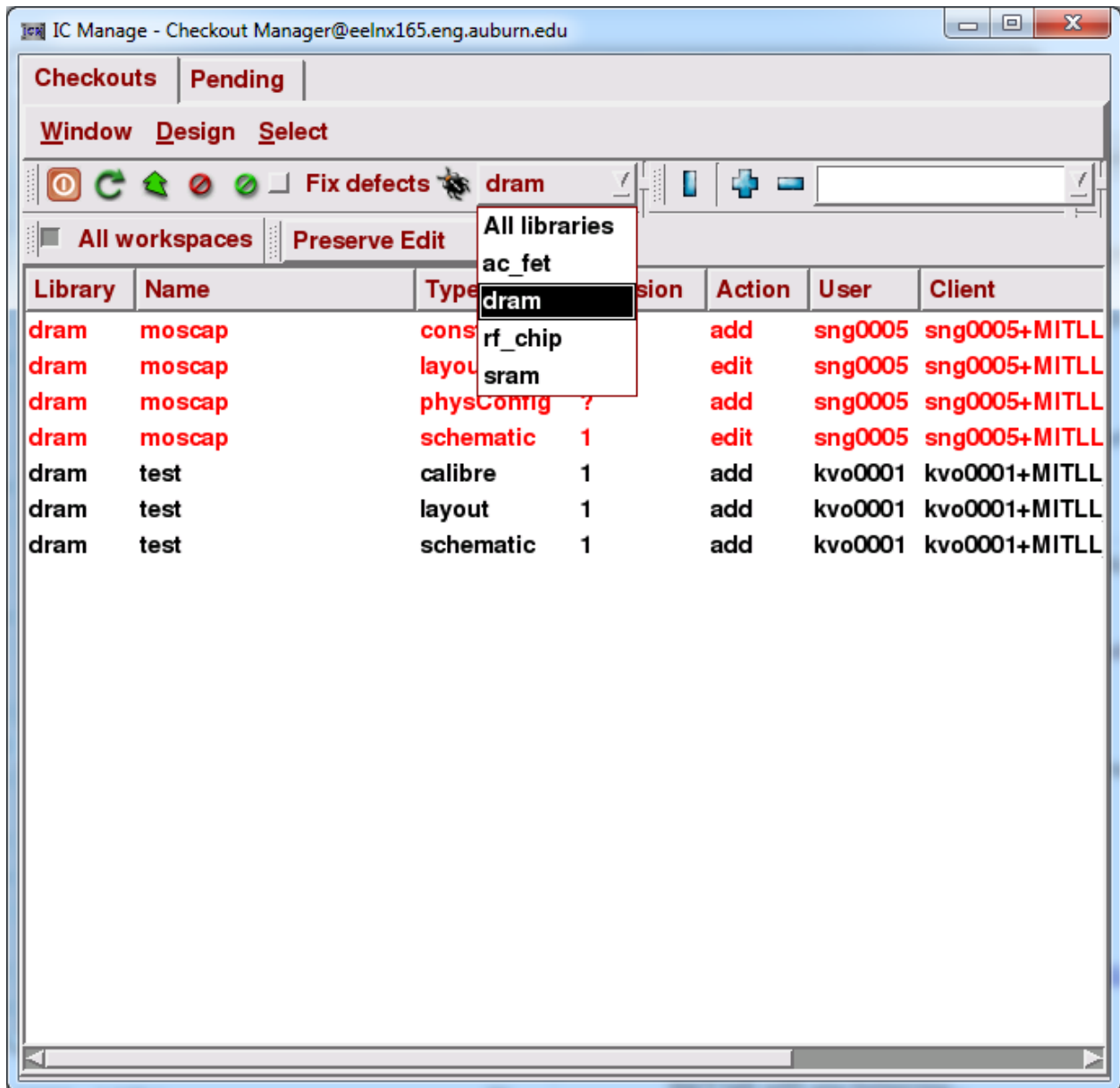
Window Design Select

Fix defects All libraries

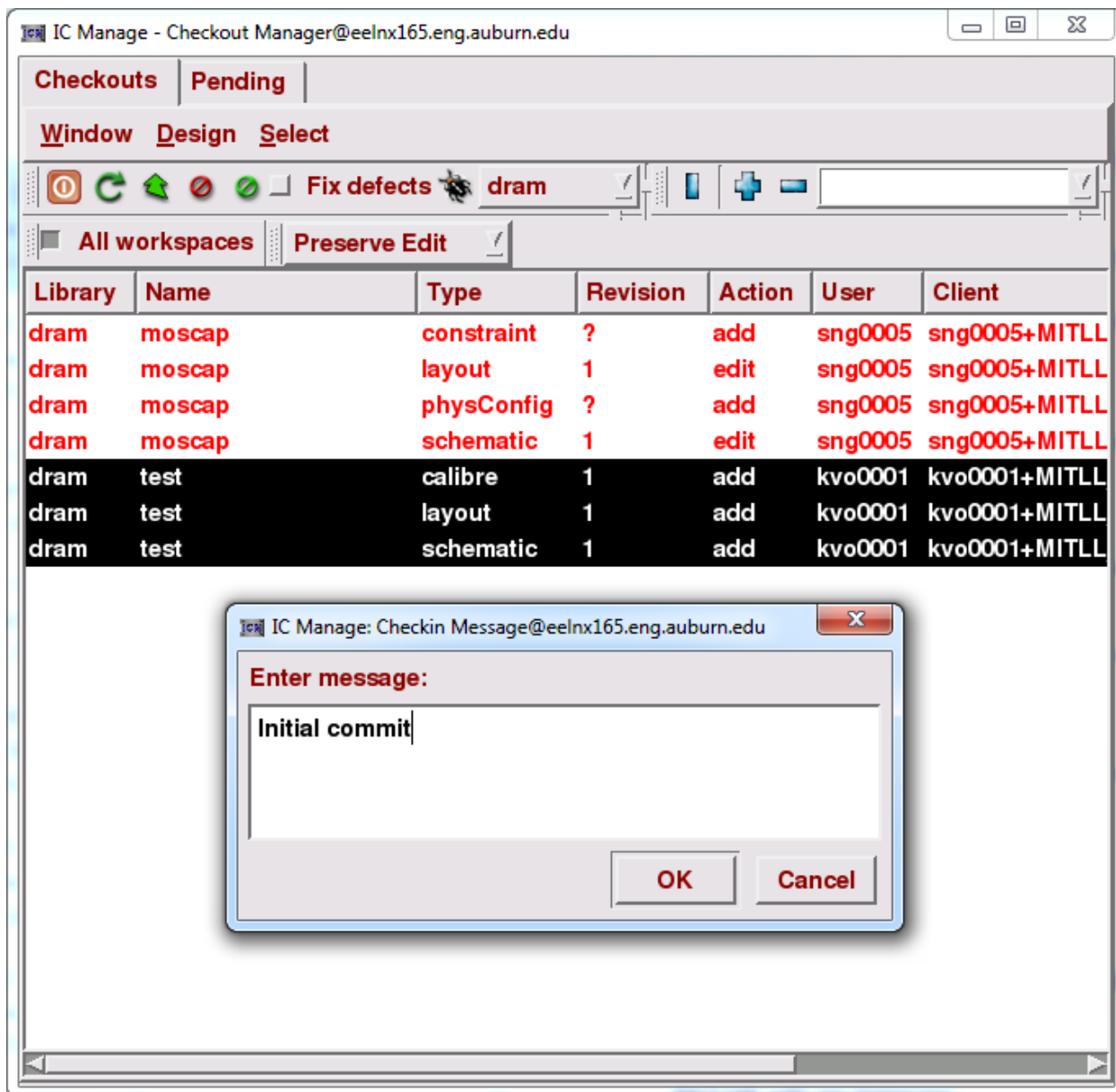
All workspaces Preserve Edit


Library	Name	Type	Revision	Action	User	Client
ac_fet	AC_pads_open_cryo3	layout	1	edit	rzc0032	rzc0032+MITL
ac_fet	ind_TRL	layout	?	add	jzd0029	jzd0029+MITL
ac_fet	pad_cryo3	layout	?	add	rzc0032	rzc0032+MITL
dram	moscap	constraint	?	add	sng0005	sng0005+MIT
dram	moscap	layout	1	edit	sng0005	sng0005+MIT
dram	moscap	physConfig	?	add	sng0005	sng0005+MIT
dram	moscap	schematic	1	edit	sng0005	sng0005+MIT
dram	test	calibre	1	add	kvo0001	kvo0001+MITL
dram	test	layout	1	add	kvo0001	kvo0001+MITL
dram	test	schematic	1	add	kvo0001	kvo0001+MITL
rf_chip	ind_cap_pad	layout	?	add	jzd0029	jzd0029+MITL
rf_chip	ind_padframe	layout	?	add	jzd0029	jzd0029+MITL
sram	bitcell	adexl	2	edit	sng0005	sng0005+MIT
sram	bitcell	schematic	?	add	sng0005	sng0005+MIT
sram	bitcell/data.dm	cell property	1	add	sng0005	sng0005+MIT
sram	butterfly_write	adexl	?	add	ths0007	ths0007+MITL
sram	butterfly_write	adexl_1	?	add	ths0007	ths0007+MITL
sram	butterfly_write	layout	?	add	ths0007	ths0007+MITL
sram	butterfly_write	schematic	?	add	ths0007	ths0007+MITL
sram	butterfly_write/data.dm	cell property	1	add	ths0007	ths0007+MITL

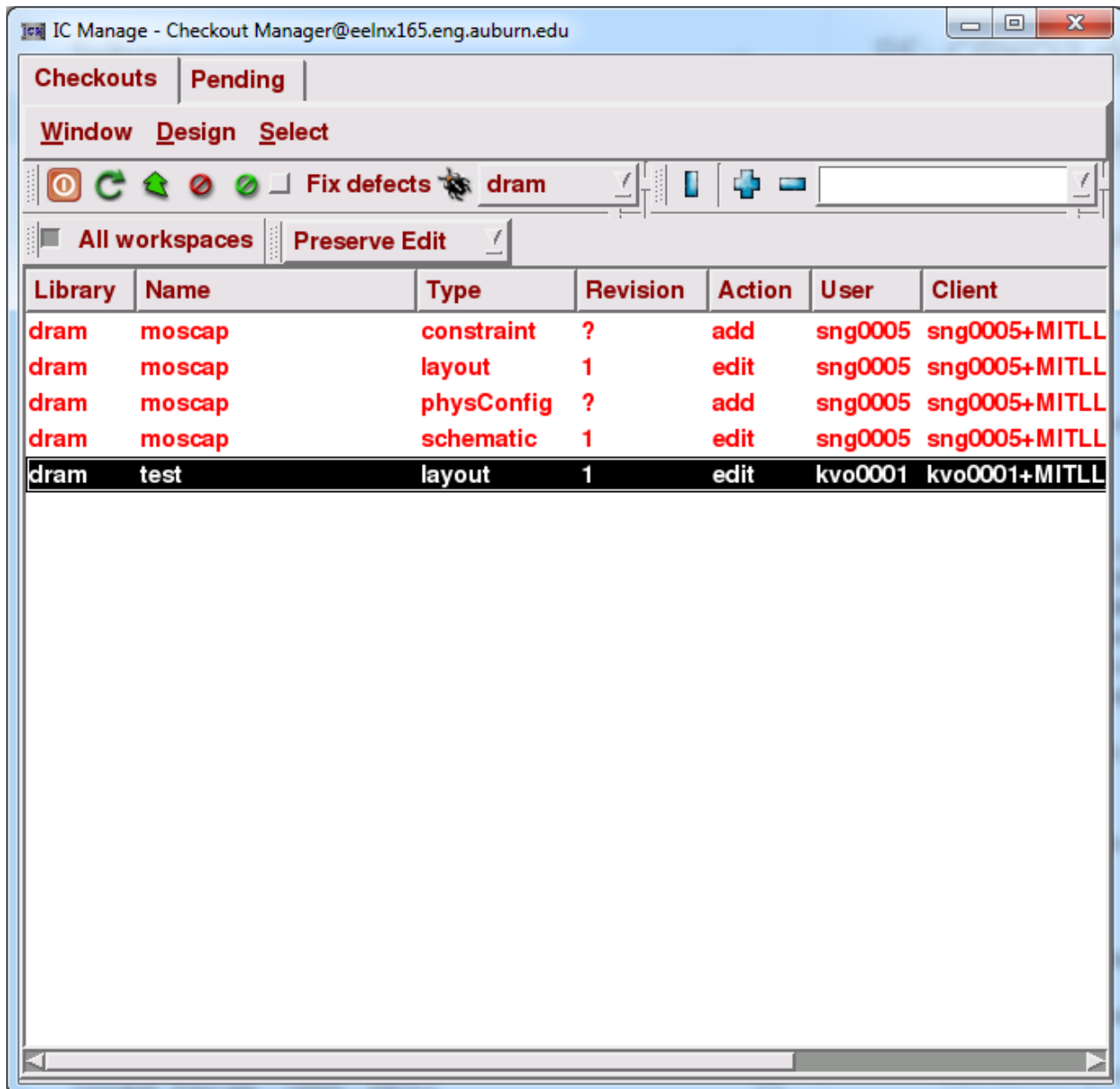
The `Checkout Manager` window allows you to examine which views are checked out across libraries and users. Items in black refer to your user, whereas red items are currently checked out by other users.




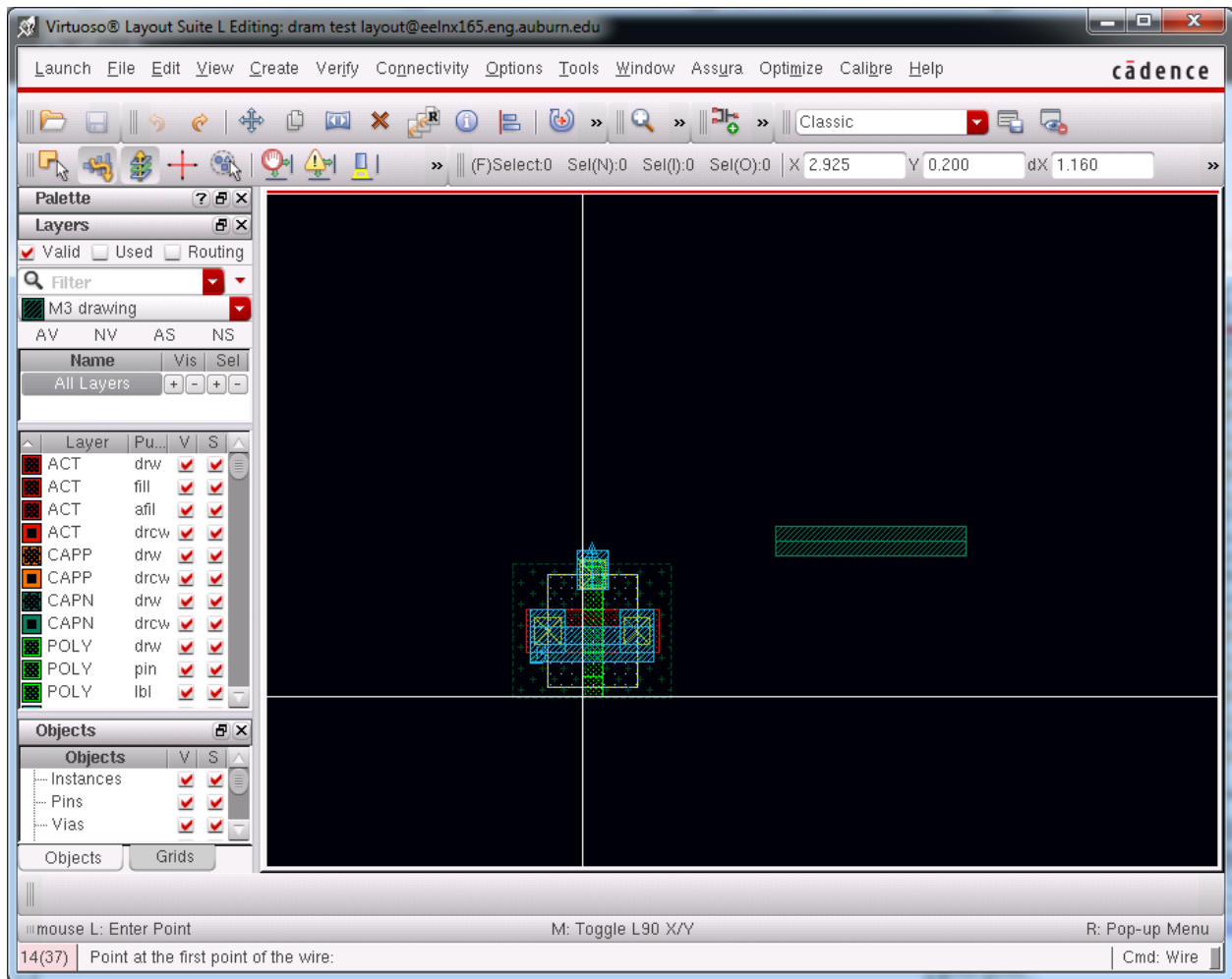
By clicking on the drop down menu, you can view just a single library.



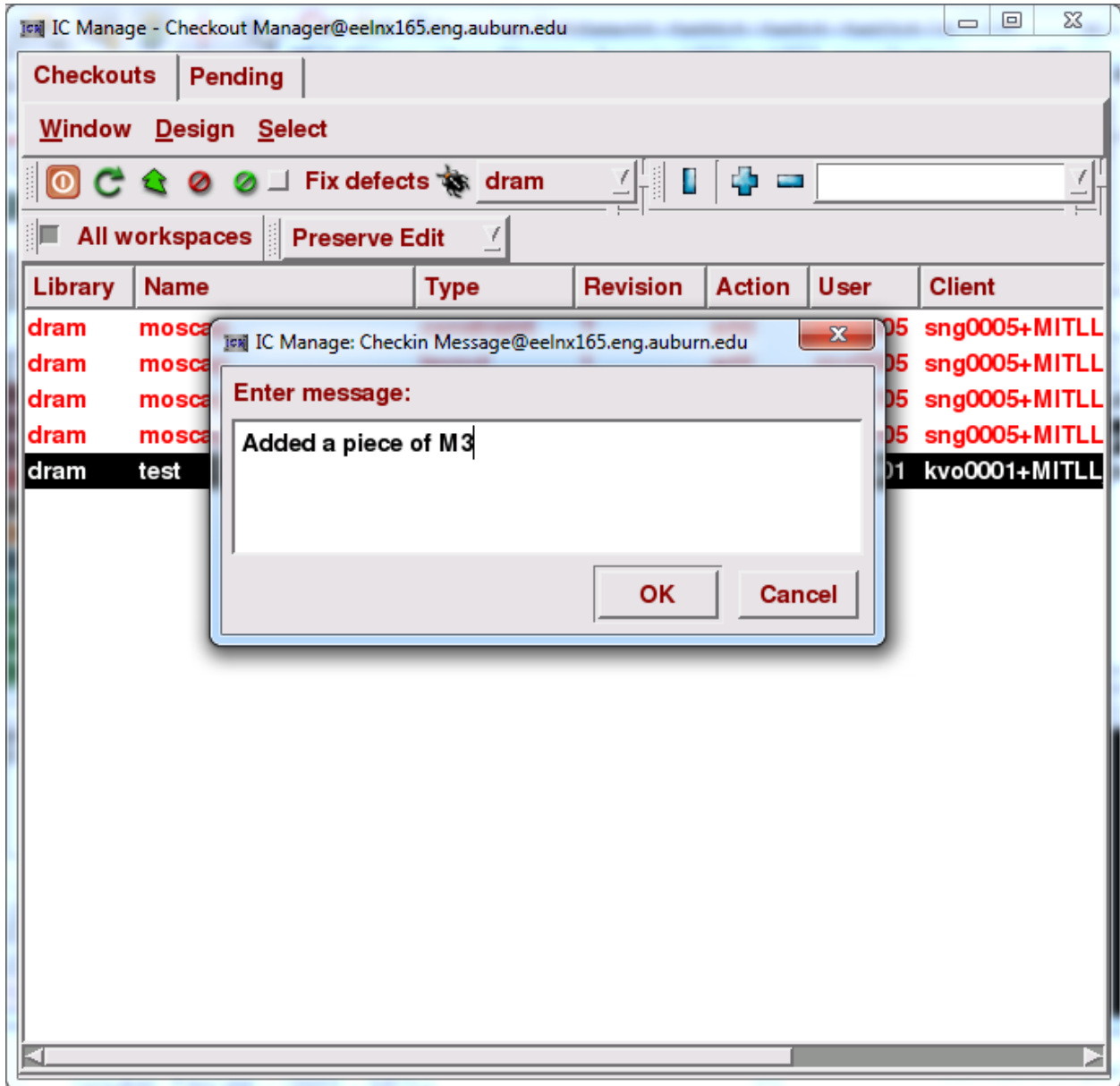
If you wish to add a newly-created view, you can select all related views and click the  button. This will pull up the `Checkin Message` window. This is your opportunity to describe the commit. For initial commits, the message “Initial commit” is probably good enough. Click `OK` when done.



If your view is still open for edit, the `Checkout Manager` will still show the view as checked out, but this time, for edit. If you wish, you can close the view or make it read only. This will not automatically check it in, though. To remedy this, you can either check it in, in which case IC Manage will notice there were no changes and alert you of this, or you can cancel the checkout by pressing the  button.

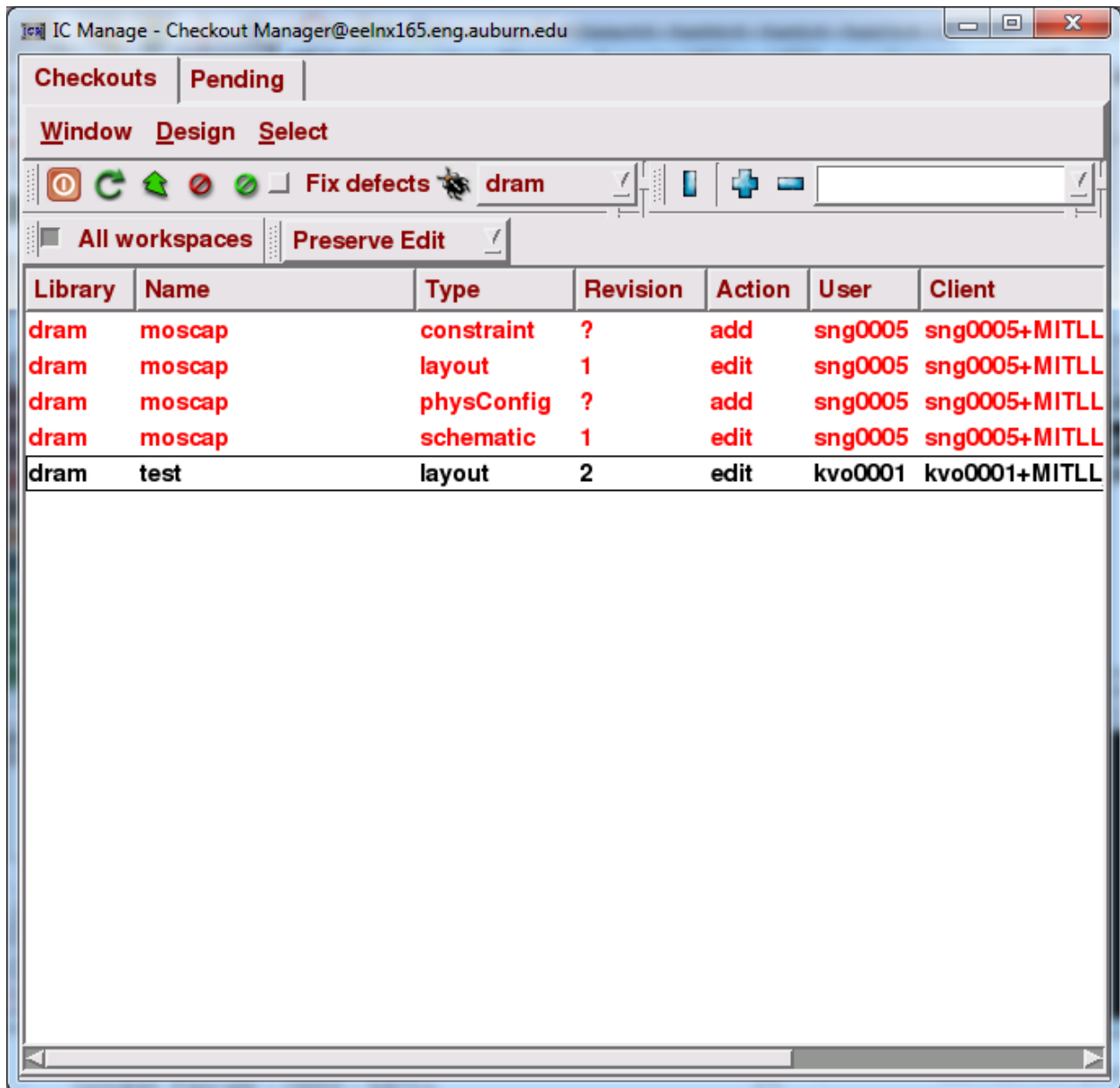


Go to the view you wish to edit and make some changes. Here, a piece of metal was added toward the right. Save the view.



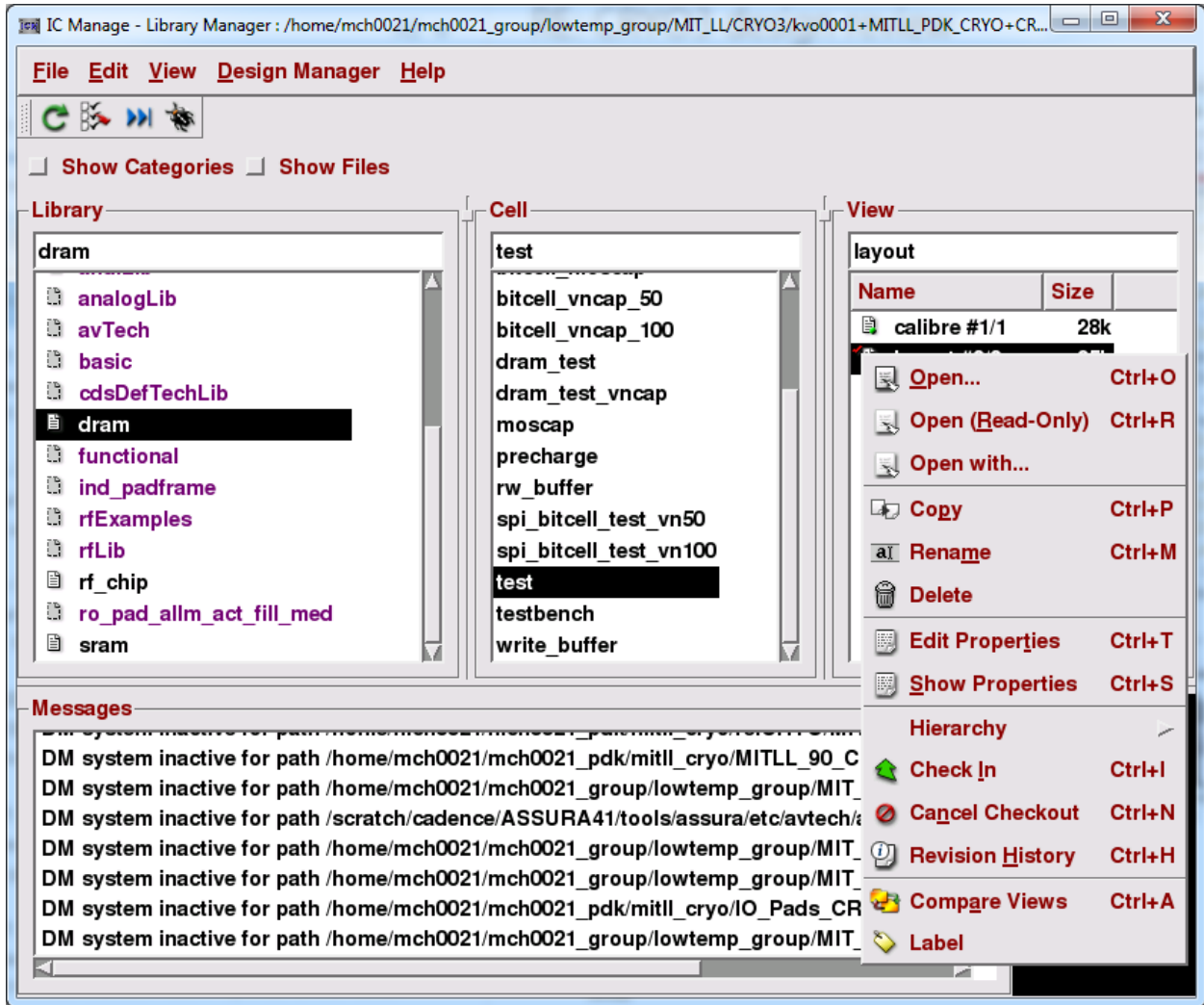
Go back to the `Checkout Manager` and check in the view. This time, your commit message should be descriptive enough for you to know what was changed. Click `OK` when you are done.



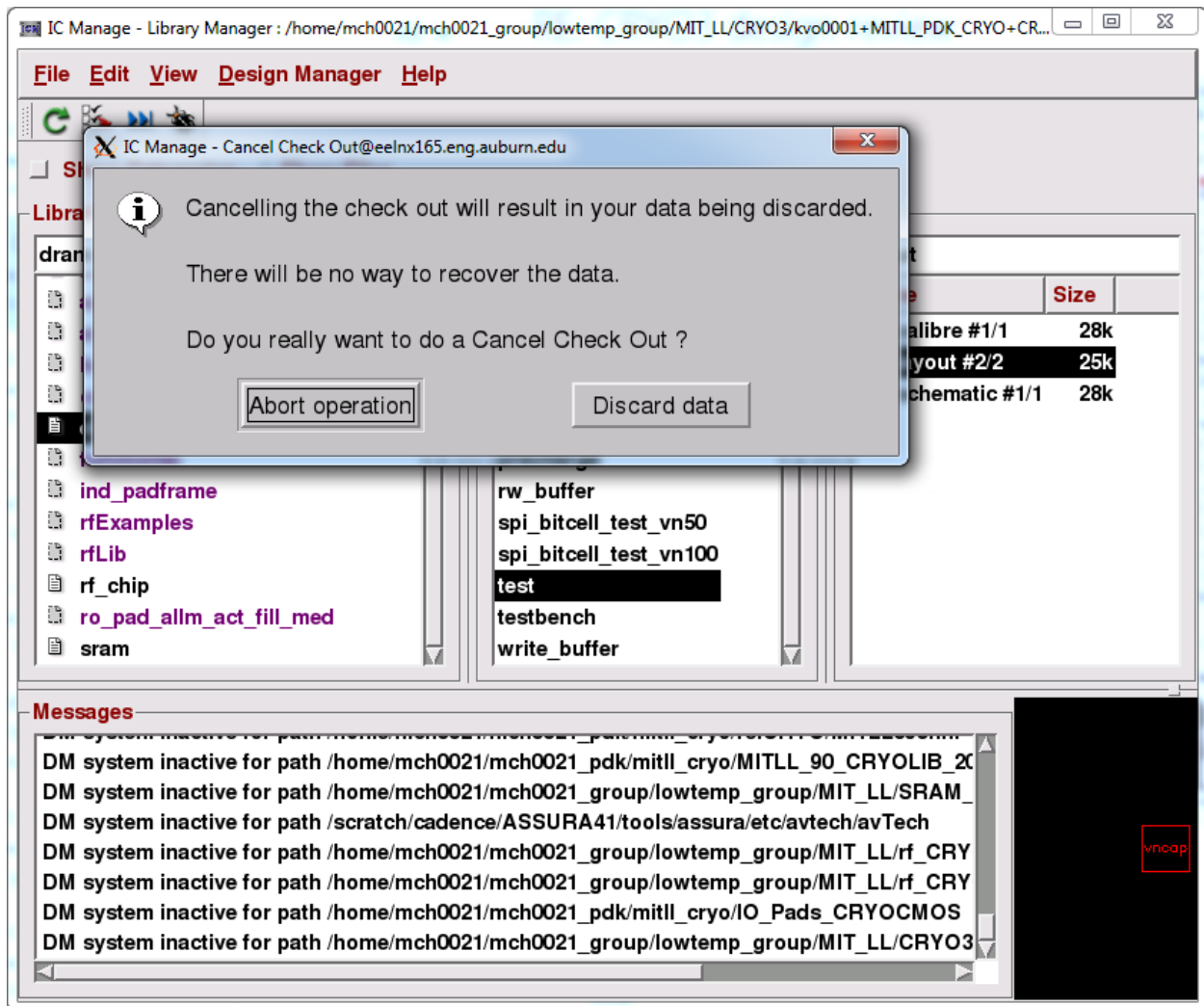


The revision number should now be one greater than it was prior. If the view was not closed or changed from edit mode to read-only mode, it will still be checked out, as seen above.

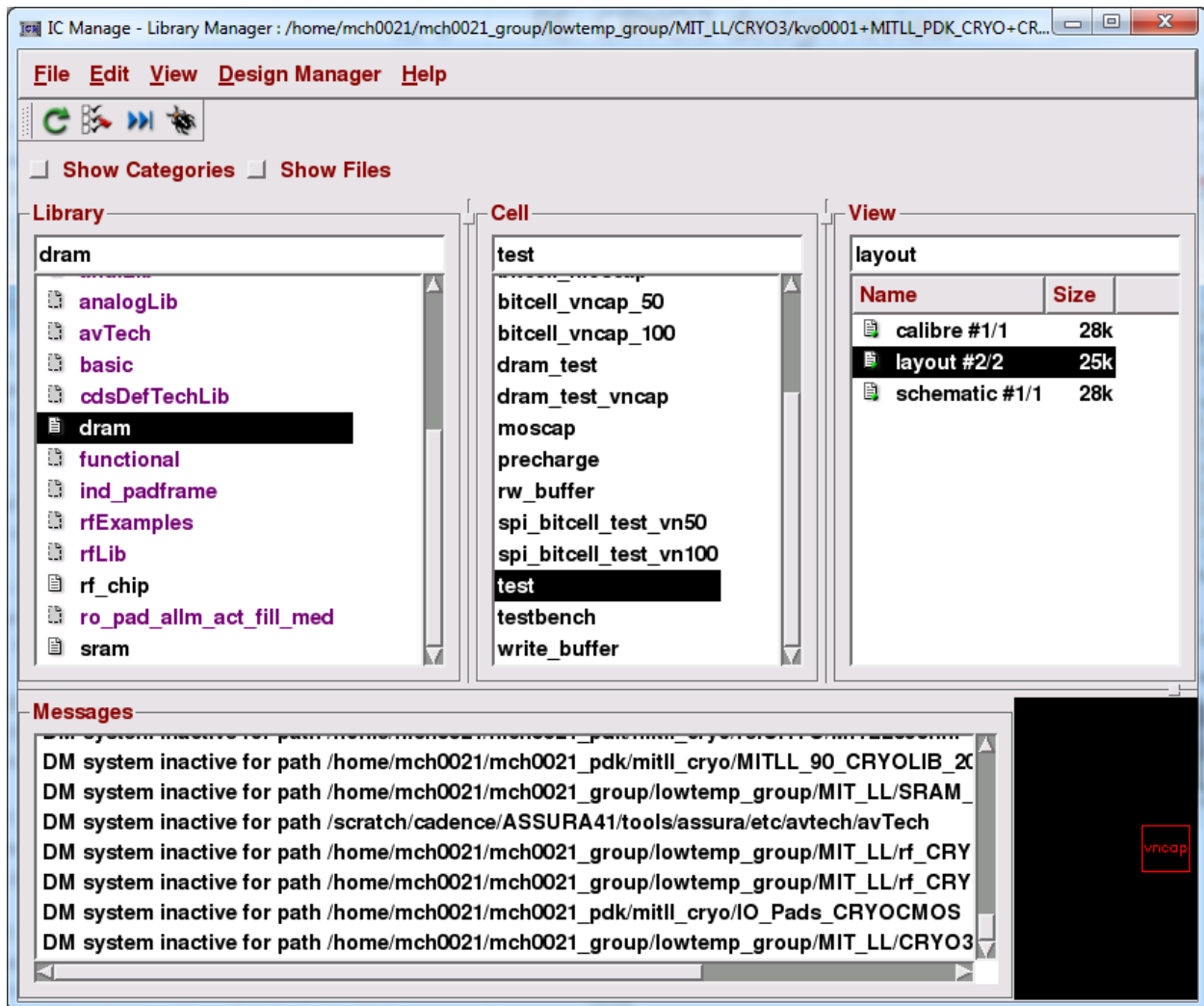
## B.10 Canceling a checkout



If the change you made was undesirable, as the case might be for accidentally drawing a random piece of metal and accidentally saving, you may wish to cancel the checkout. Right click on the view and choose `Cancel Checkout`.

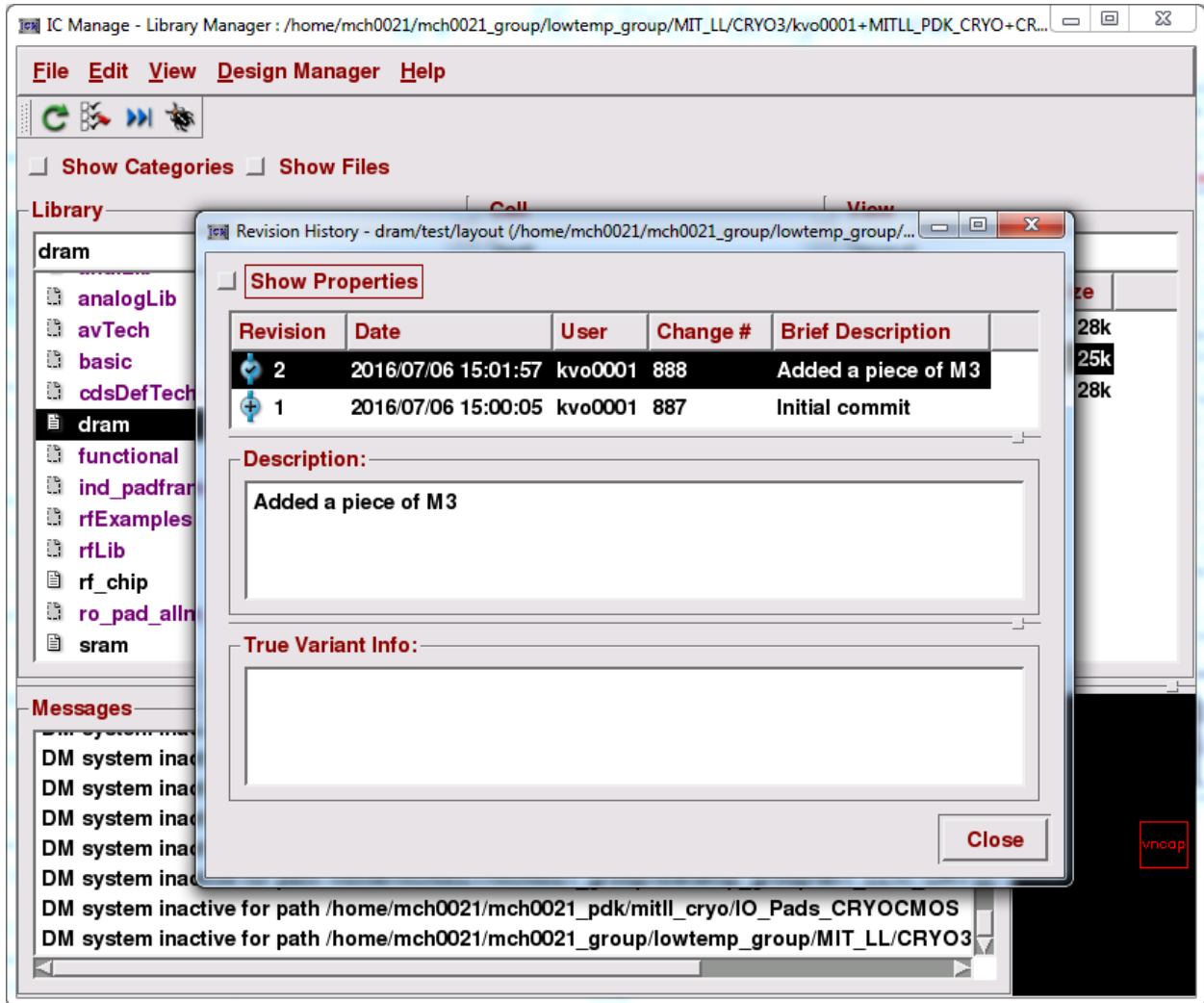


This will cause a loss of data and cannot be undone. You should only do this if you are sure of what you are doing. Click  to continue with the process, or  if you do not wish to lose the data.

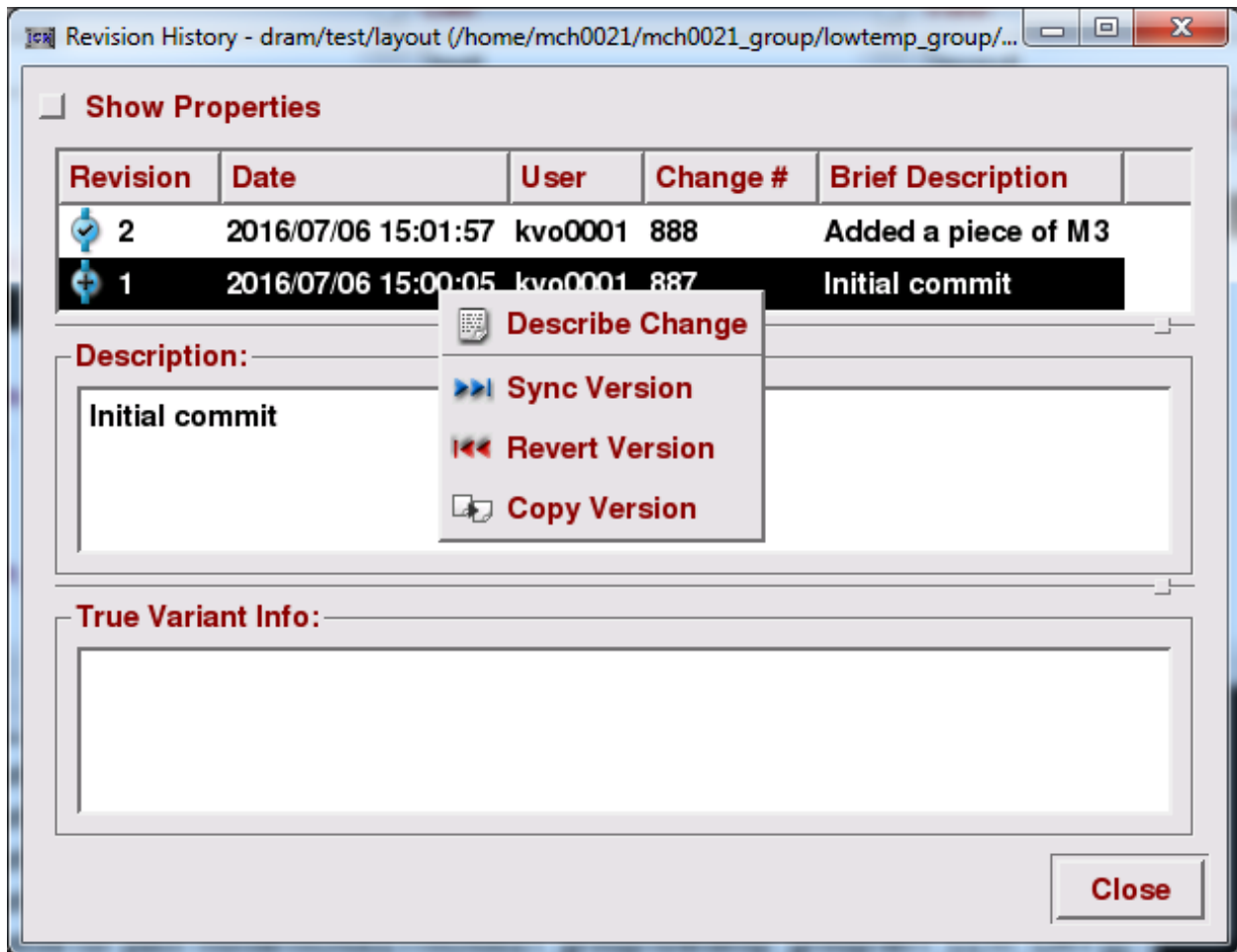


Canceling the checkout does not affect the revision number, as seen above.

## B.11 Reverting to an older version



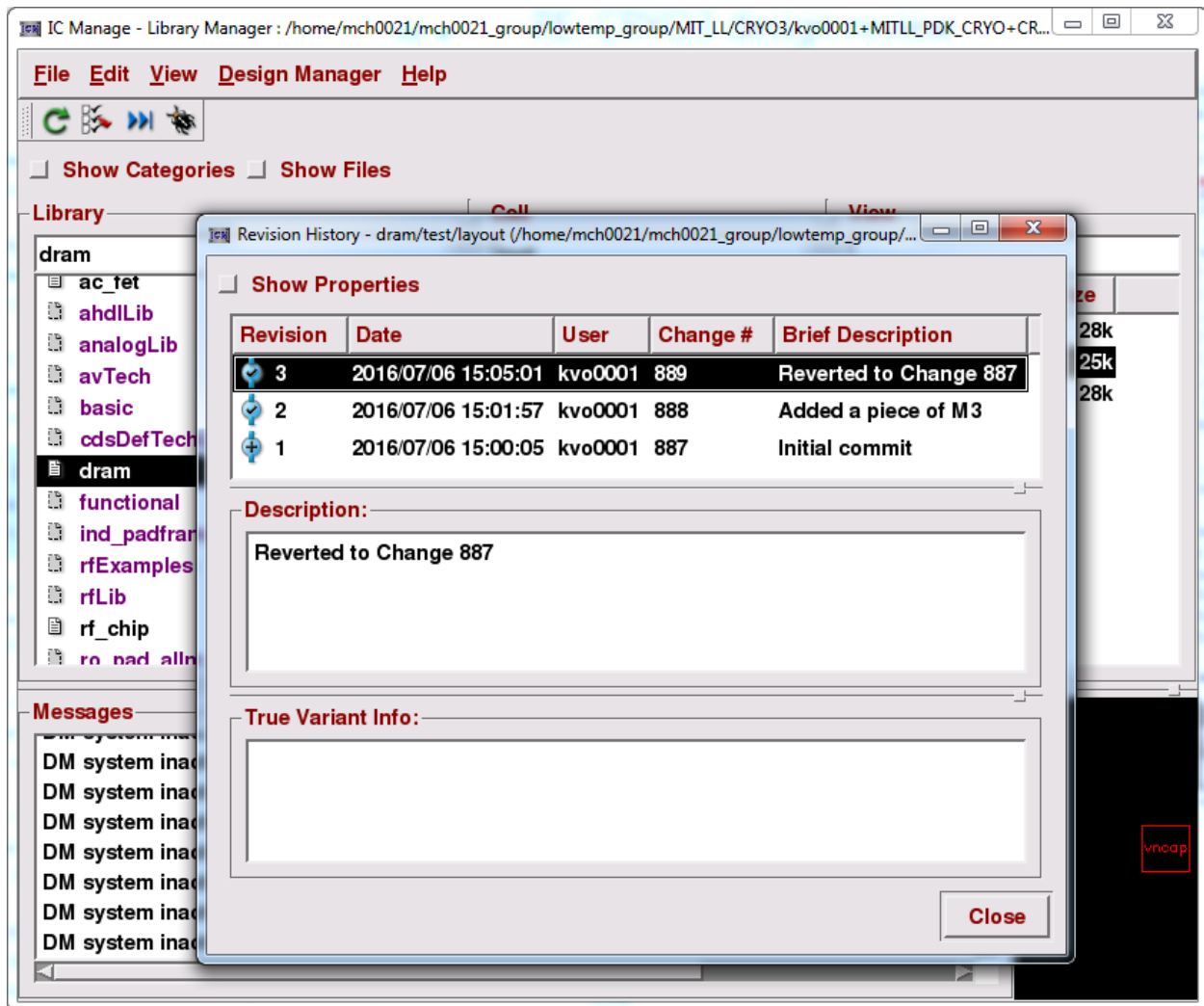
Instead, if you wish to view the revision history to date, right click on the view and select `Revision History`. This will show all revisions and the brief description from the commit.



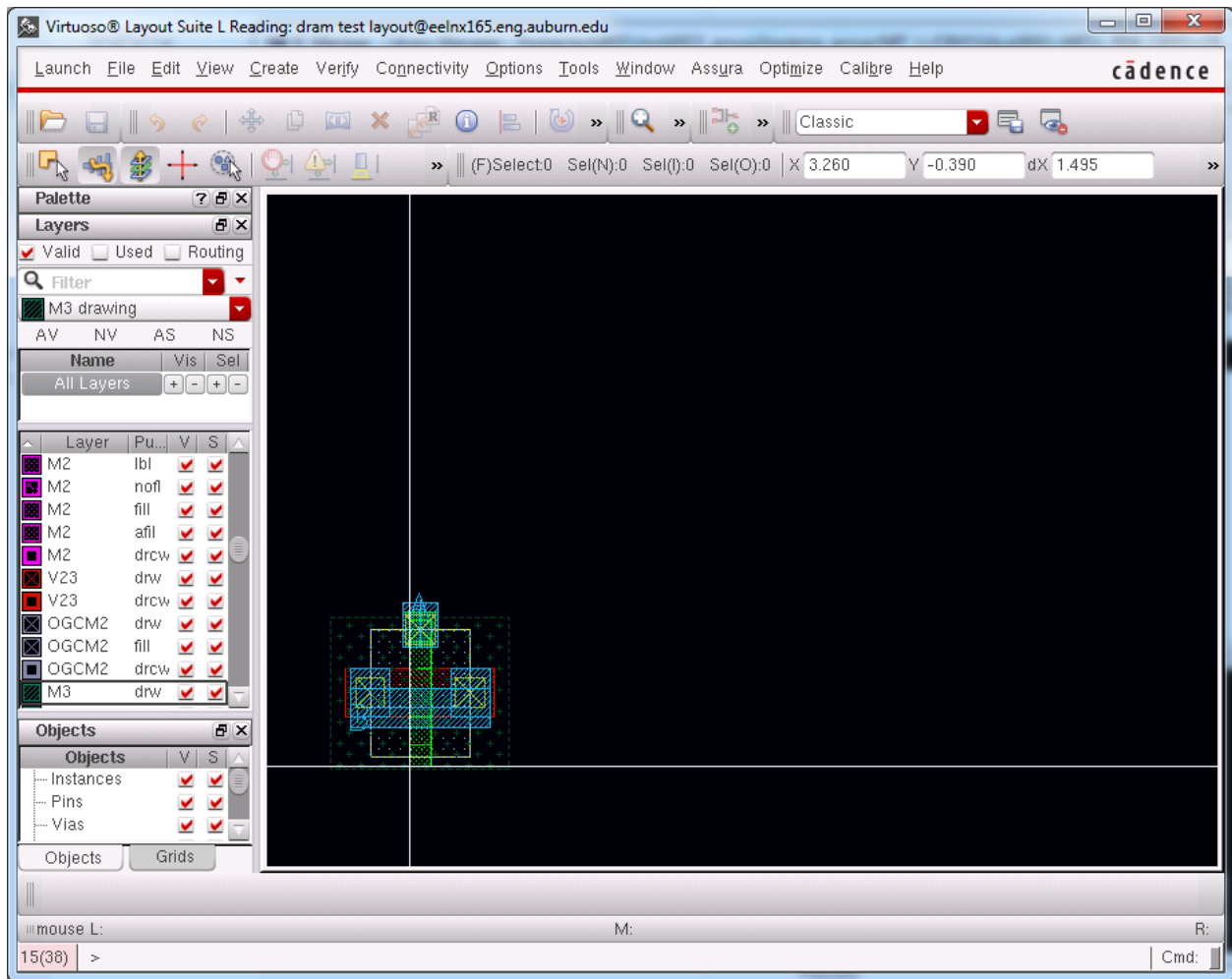
If you wish to revert to an older revision, you can right click on the revision you wish to revert to and select **Revert Version**.



Select **Yes** to revert. This does not cause a loss of data and can be undone.

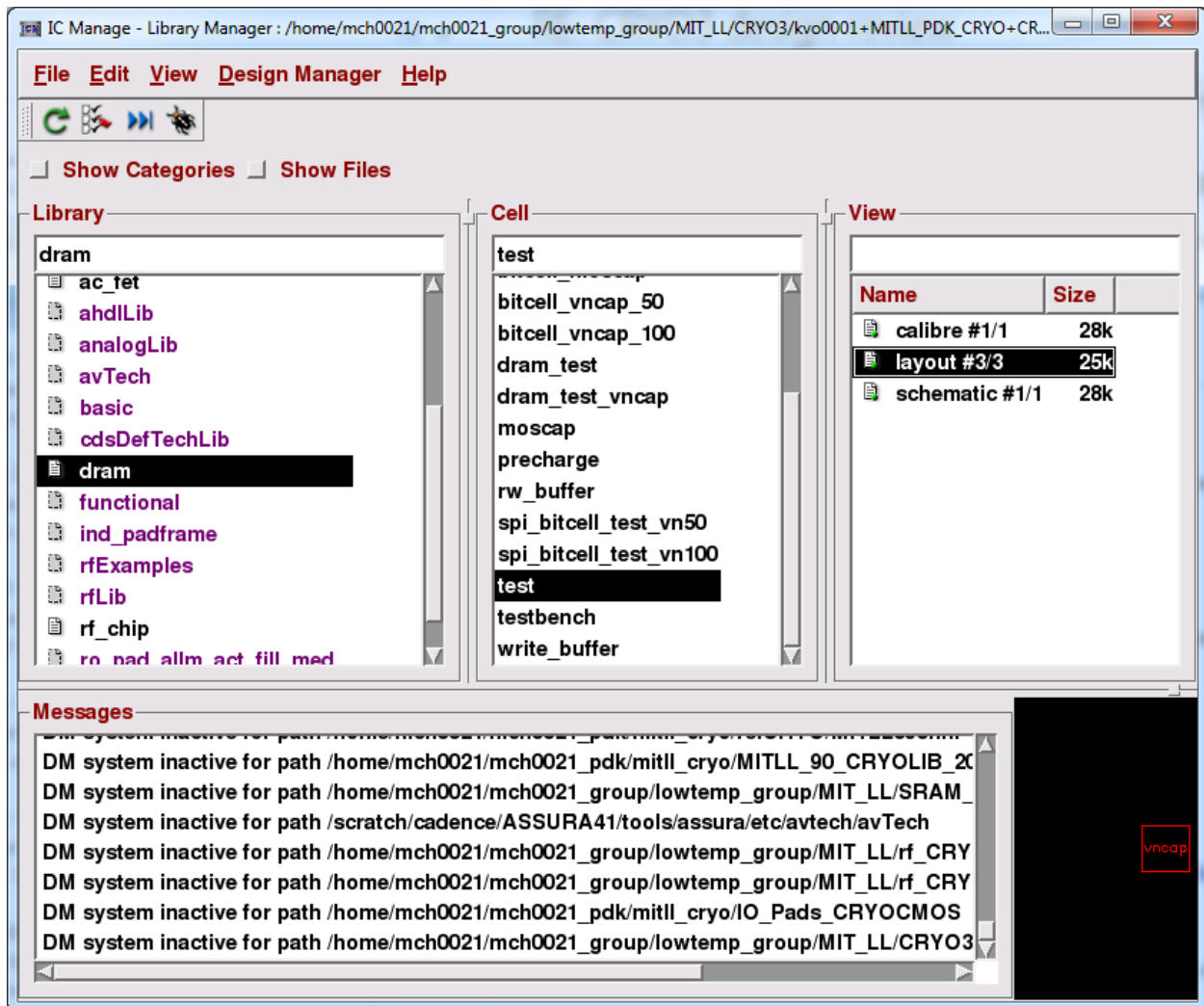


The revision history is automatically updated to show that it was reverted. This causes a new revision in the history, which is identical (in this case) to the first revision.



IC Manage may want to refresh the library and views. Click  to do so. You should see that the layout has been reverted.





Again, the revision count has been incremented after a revert. You can even revert a revert by reverting back to a view prior to the revert.

## Appendix C

### Plotting Schematics Tutorial

This tutorial will show how to create prettier schematics from Cadence. These schematics will be free from aliasing effects, as they are rendered in vectors rather than pixels. First, ensure that you have created a `.cdsplotinit` file in your home directory. This will tell Cadence how to output schematics when plotting.

#### C.1 `.cdsplotinit` Contents

```
EPS|Encapsulated Postscript: \  
    :manufacturer=Adobe: \  
    :type=epsfC: \  
    :maximumPages#1: \  
    :resolution#300: \  
    :paperSize="Unlimited" 72000 72000:
```

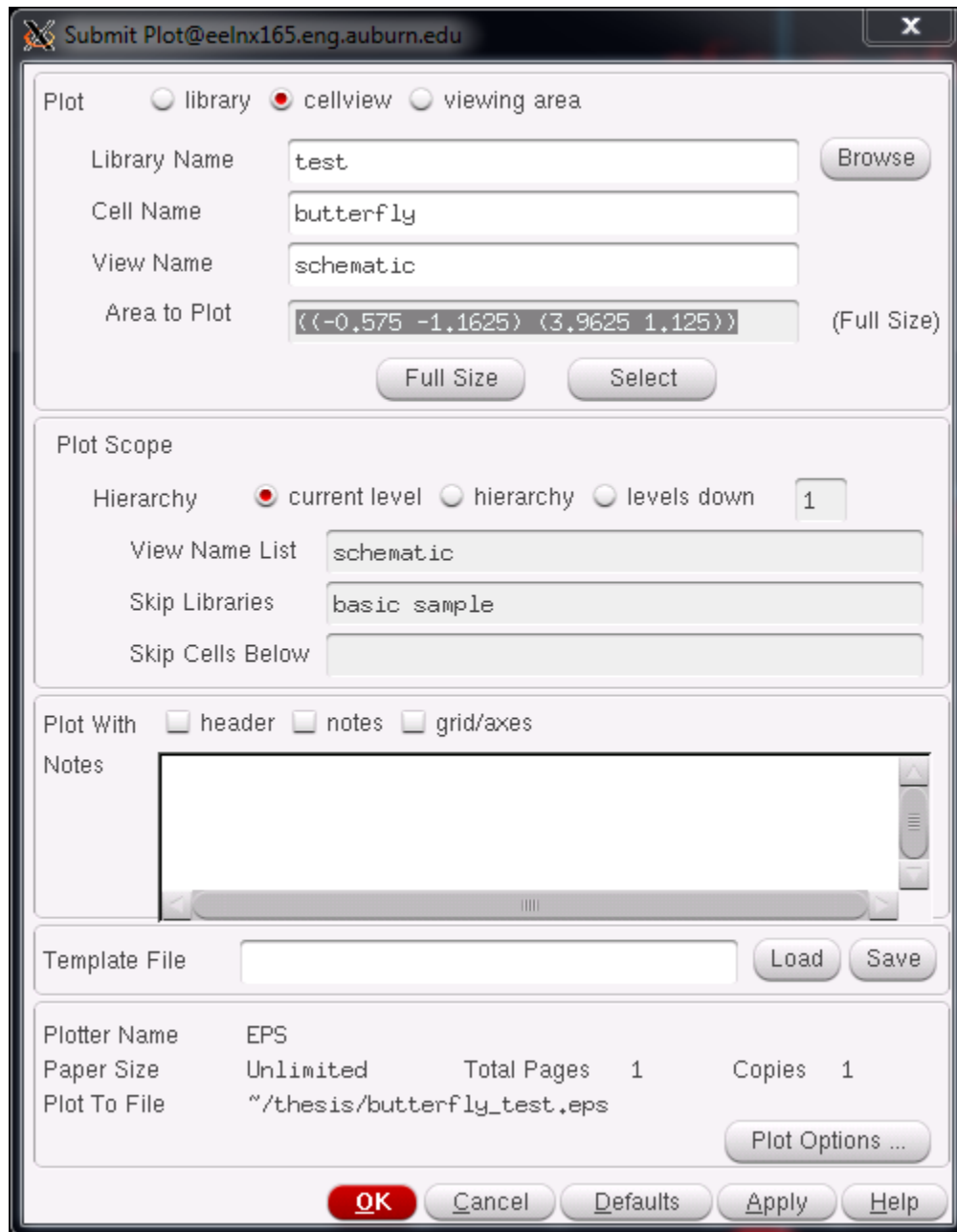
The above `.cdsplotinit` will allow the creation of color Encapsulated PostScript (EPS) files. With a little help from `sed`, a stream editor commonly used for manipulating files, anyone can have nicer looking schematics from Cadence Virtuoso. Save the following `sed` script in a convenient location.

## C.2 schematic\_beautify.sed Contents

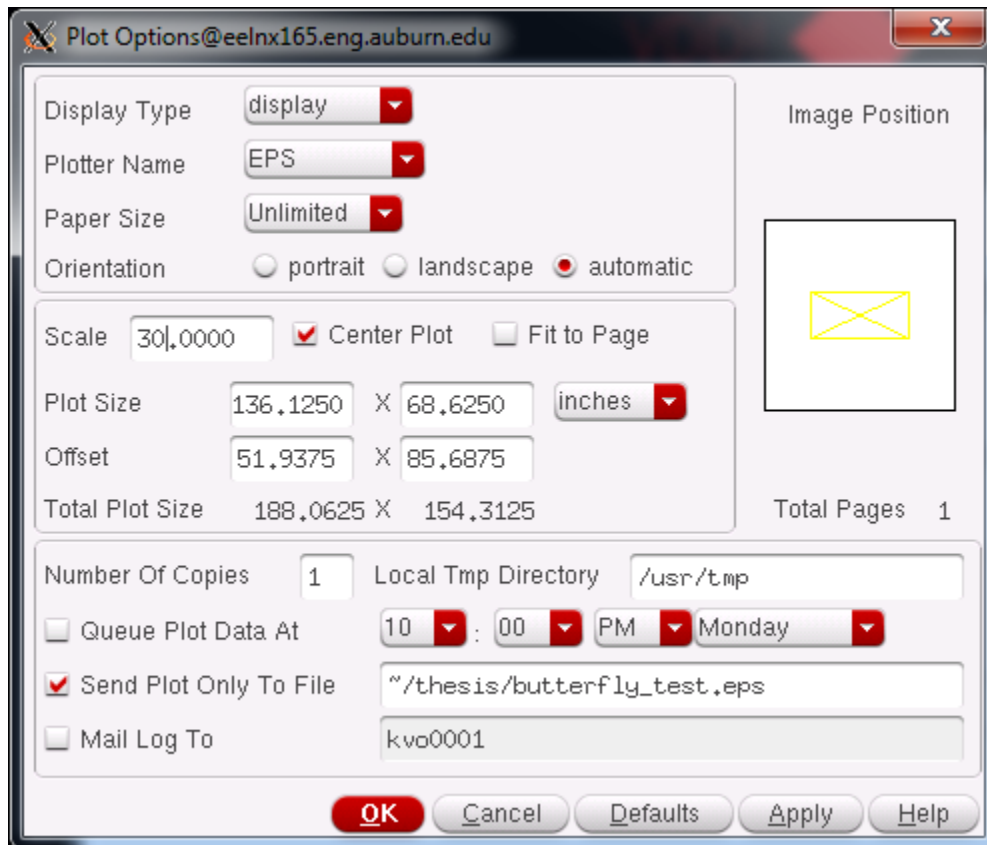
```
/1 setlinewidth/c\  
40 setlinewidth\  
1 setlinejoin\  
1 setlinecap  
s/1000 0 0/500 0 0/g  
s/0 800 400/0 400 0/g  
s/224 749 1000/0 250 500/g  
s/0 1000 1000/0 250 500/g  
s/851 800 0/0 100 500/g
```

This “schematic beautifier” will take in crummy-looking EPS-format schematics and output pretty schematics. It searches for a few key components of the EPS file and replaces them. Namely, it looks for color and line width information and replaces them with values that give the schematic more photogenic qualities.

Now open the schematic you wish to plot. Click on   to pull up the  window.

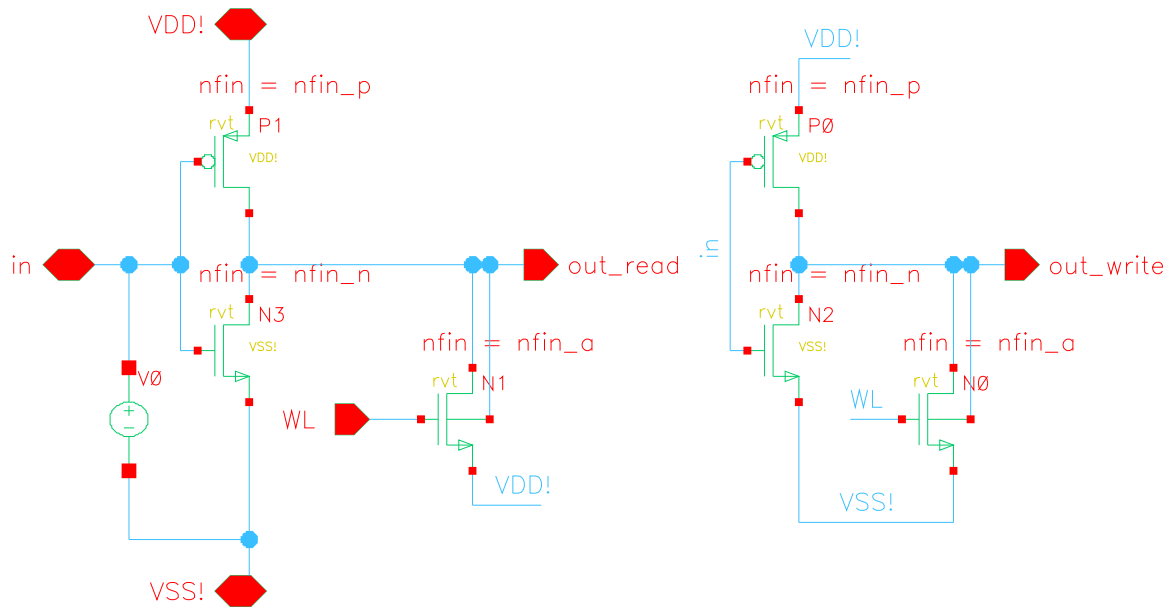


As seen above, the window will, by default, select **Full Size**. Instead, use the **Select** button to draw around the entirety of the schematic you wish to capture. The difference is that the **Full Size** option is not aware of labels, and thus, will cut them off. Also, uncheck the **header** box. Then, click on the **Plot Options...** button to continue.

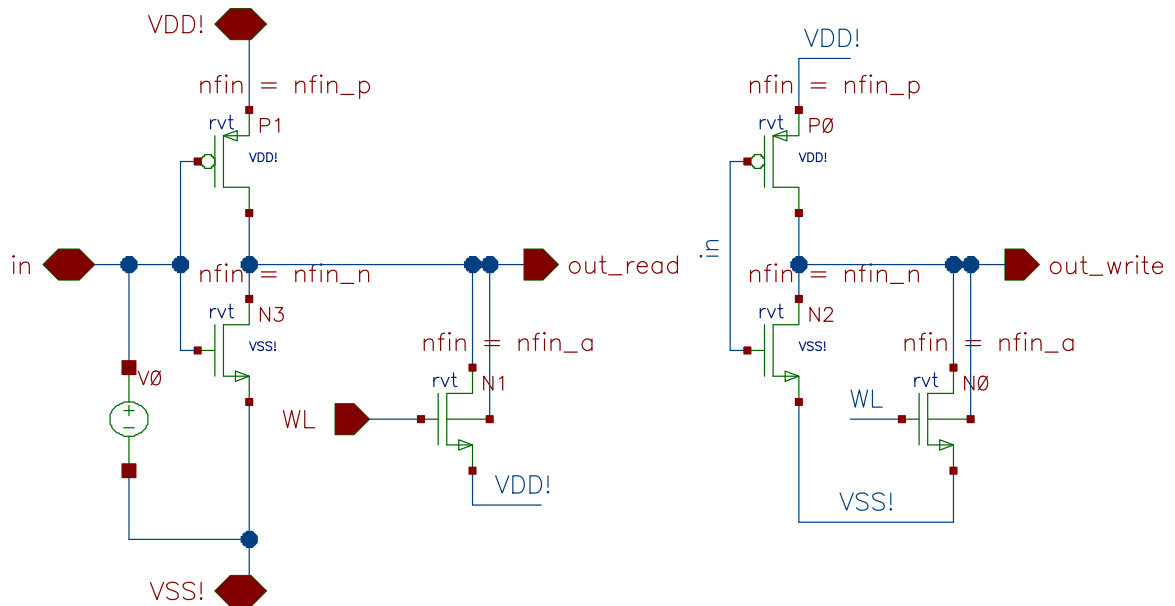


The window should appear. By default, the EPS option should be selected. Change the scale to 30 and select `Center Plot`. You can play around with the scale if your plot appears too small still; decrease the scale number to increase the apparent size. The plotter does not automatically adjust line width to compensate for scale. Uncheck the `Mail Log To` box, check the `Send Plot Only To File` box, and fill in the text box with the file name you wish to give your plot. Be sure to use the .eps extension so that other applications will know what it is.

This is what the schematic will look like by default:



The line width is very narrow and the colors are horrendous against a white background. After running `sed -f schematic_beautify.sed [input EPS] > [output EPS]`, the schematic should look something like this:



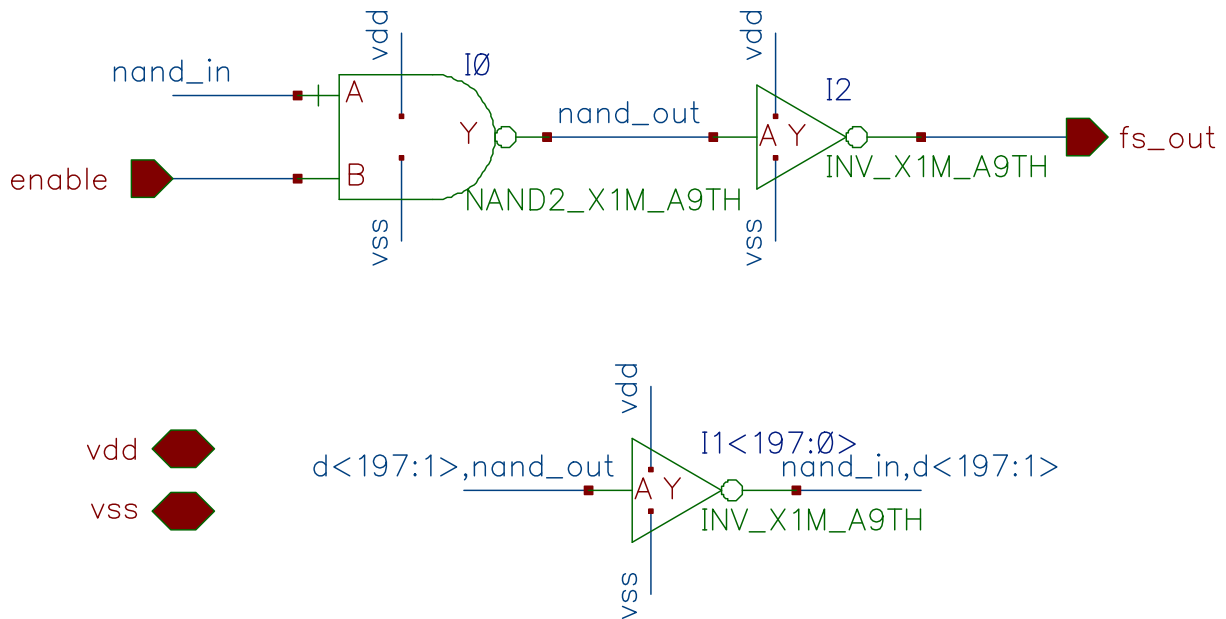
The lines are thicker, and the colors are more pleasing to the eye.

## Appendix D

### ADE Tutorial

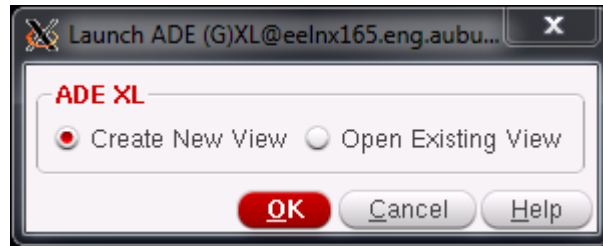
The Analog Design Environment, or ADE, is a powerful simulation environment that allows for easy analog simulation of schematics. ADE can use several different simulators, such as Cadence Spectre and HSPICE, among others. With the proper configuration, ADE can even perform hybrid analog and digital simulations, saving a lot of simulation time. For a large SRAM, for instance, the address decoding logic can be simulated digitally, whereas the individual bit cells remain fully analog.

To begin, start with a schematic that you'd like to simulate. For this example, a 199-stage ring oscillator will be simulated.

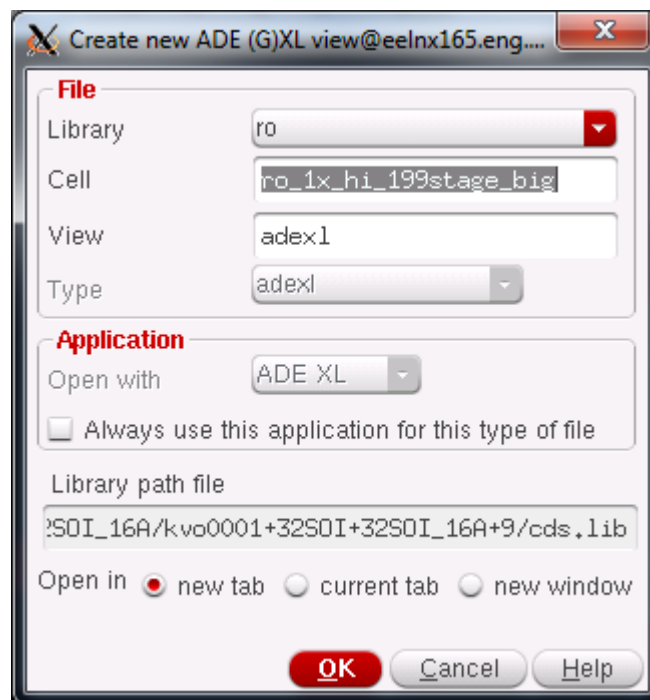


The schematic includes ports for VDD, VSS, and any inputs and outputs. It is not required to have a port for any signals that you wish to examine, but it is highly recommended that for any nets that you do want to look at, they are at least named descriptively.

The next step is to launch ADE from the schematic view. This will allow the creation of an ADE view where configuration information can be stored. Click on **Launch >> ADE XL** to begin the process.

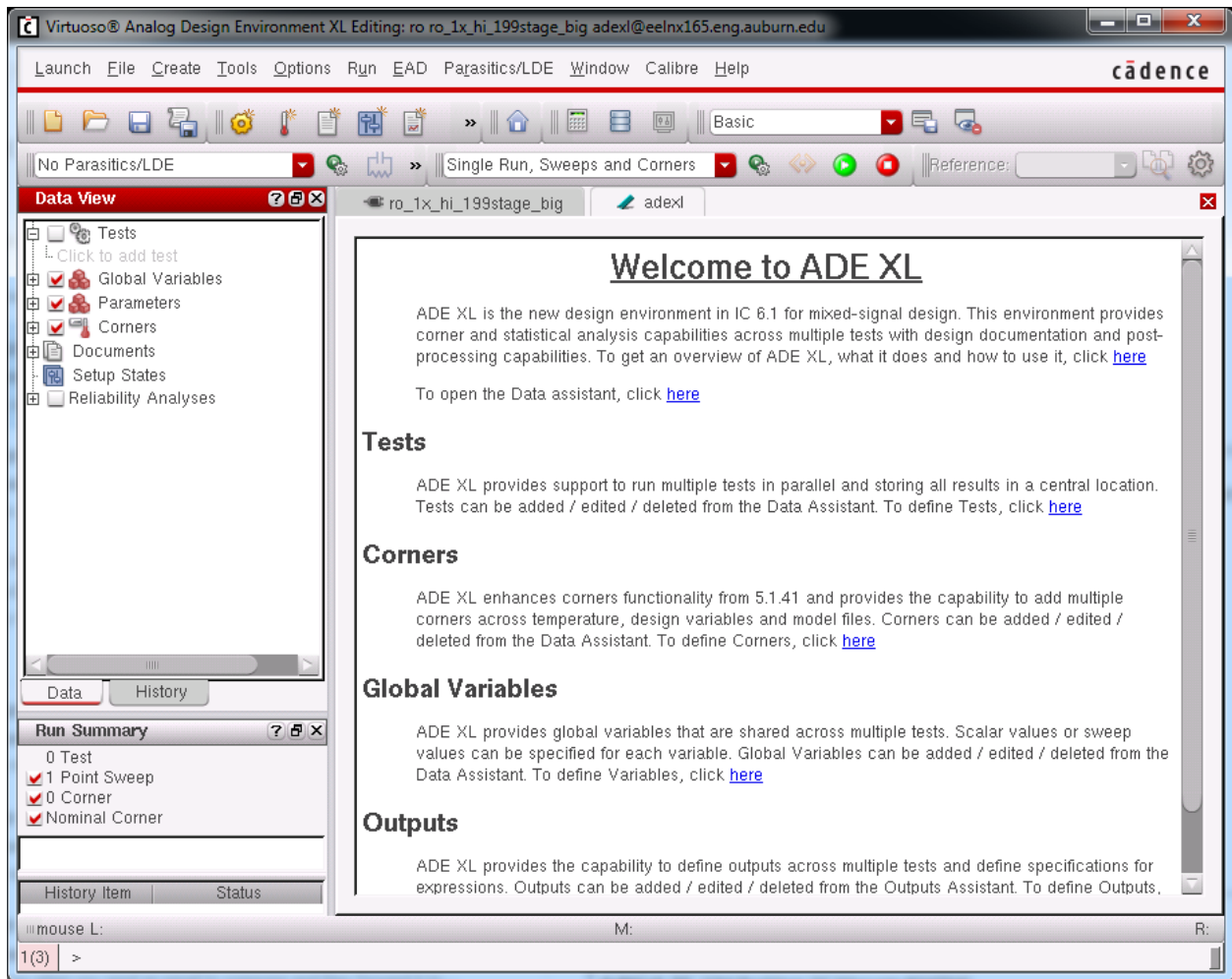


Select the **Create New View** option and click **OK**. A new window should pop up.

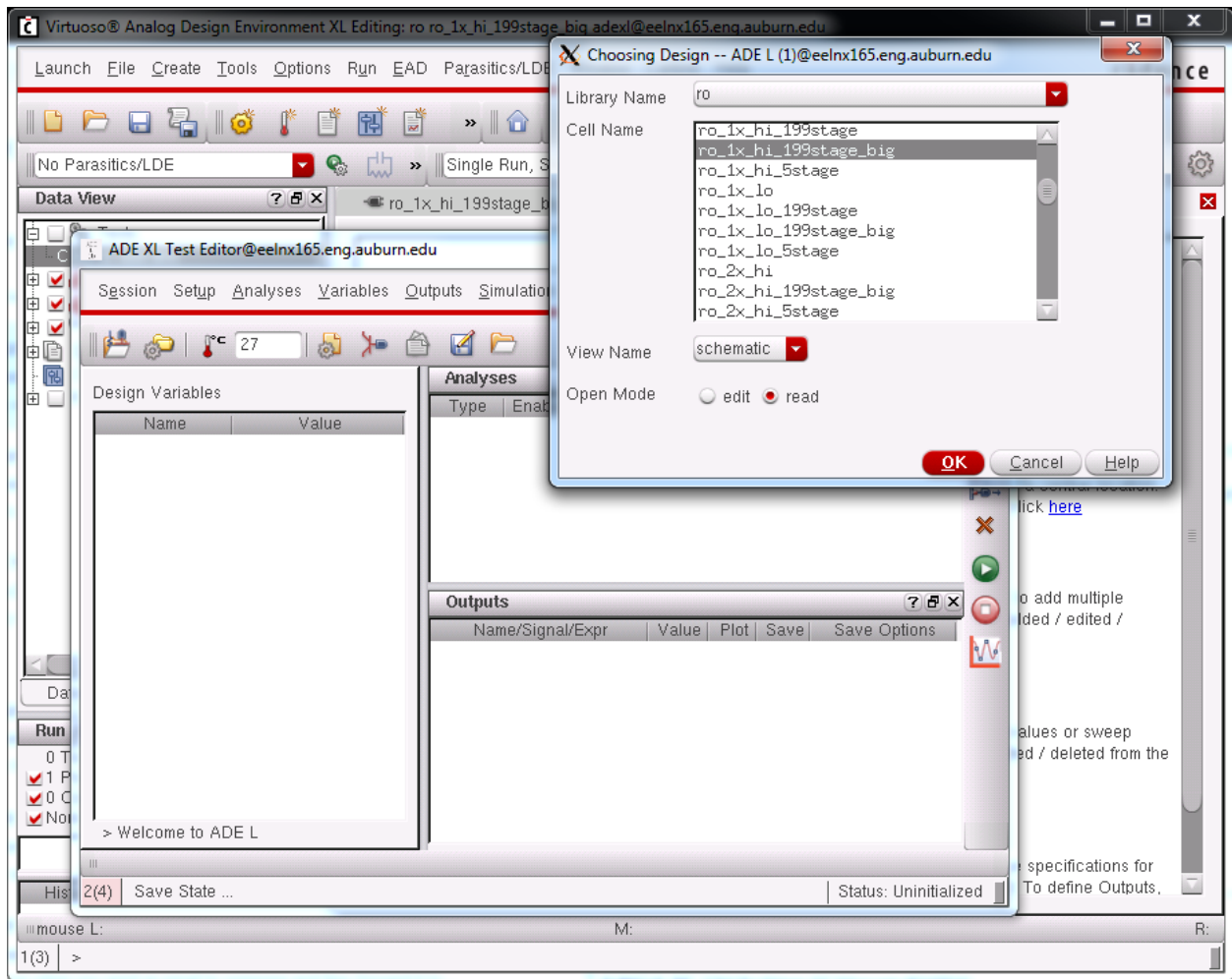


Go ahead and click **OK** and watch as the ADE XL window is generated.



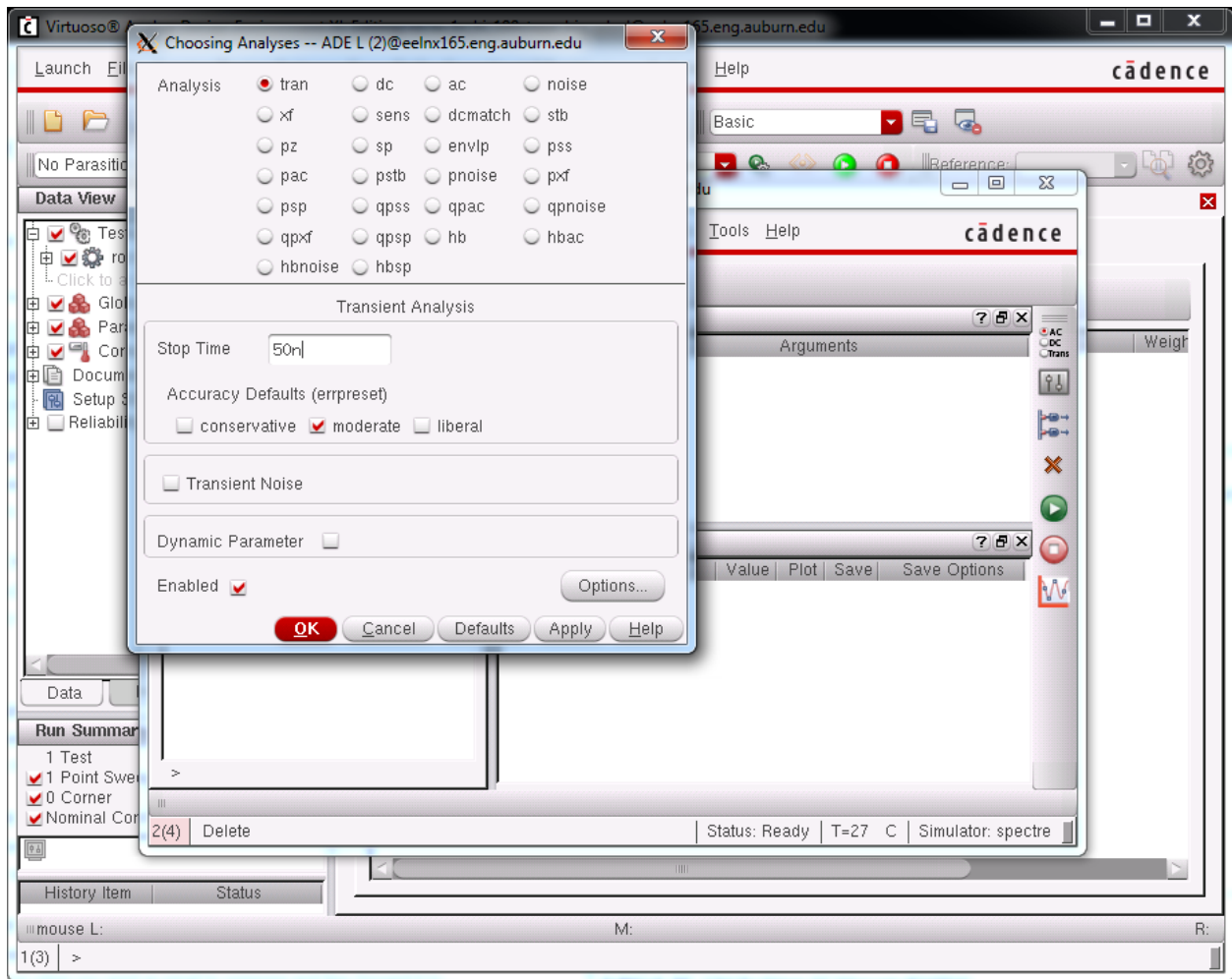


The ADE XL window should pop up, as seen above. Then, click once (do not double-click) on **Click to add test** below **Tests** on the left side of the window. Two new windows should appear.

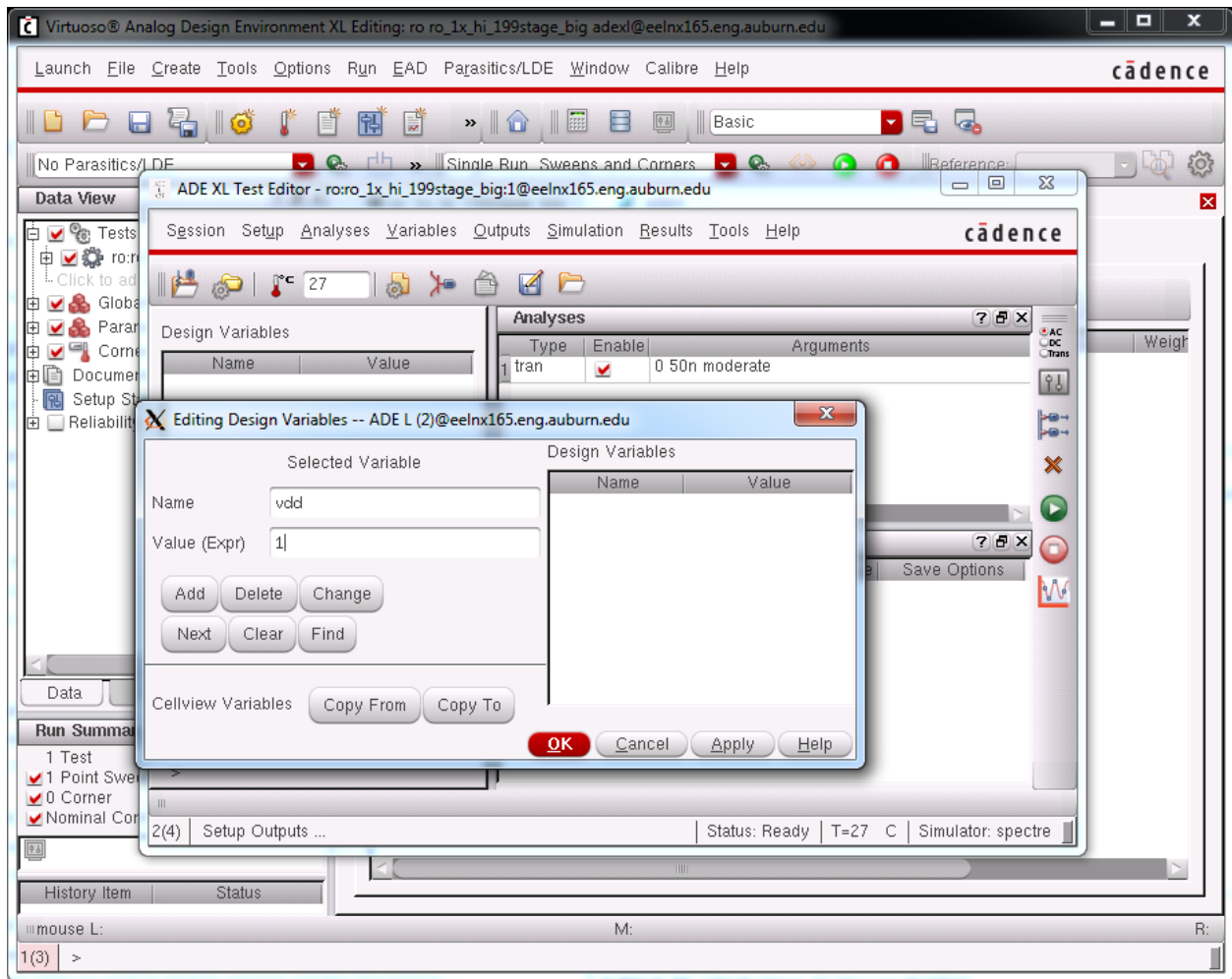


The **Choosing Design** window confirms what schematic you will simulate. You can open for read or edit, depending on if you intend to make further changes during the course of simulating. Click **OK** after you've made your selection.

The other window that appeared is the **ADE XL Test Editor** window, which allows for the manipulation of a particular simulation. Quite a bit of setup is involved here in this window.

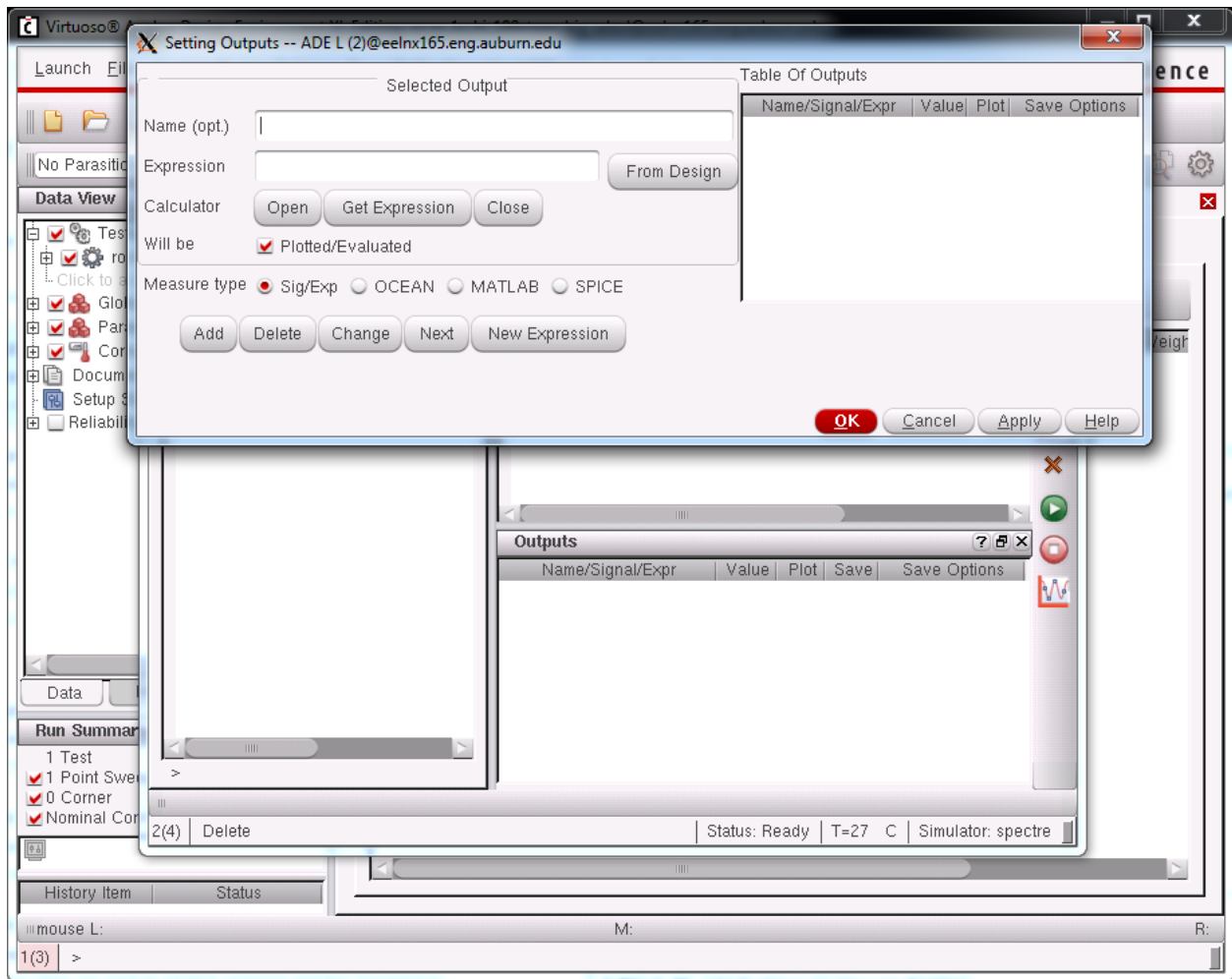


First, click on the **Choosing Analyses** button, top right in the **ADE XL Test Editor** window. For a RO, we will look at voltage over time, so performing a transient analysis makes the most sense. For a RO, you will want to examine several periods, so pick a simulation stop time that makes sense for the expected frequency. 50 ns might be a good starting point. As far as accuracy is concerned, **moderate** seems to be a good starting point as well. By default, this analysis will be enabled. Click **OK** once you're done.

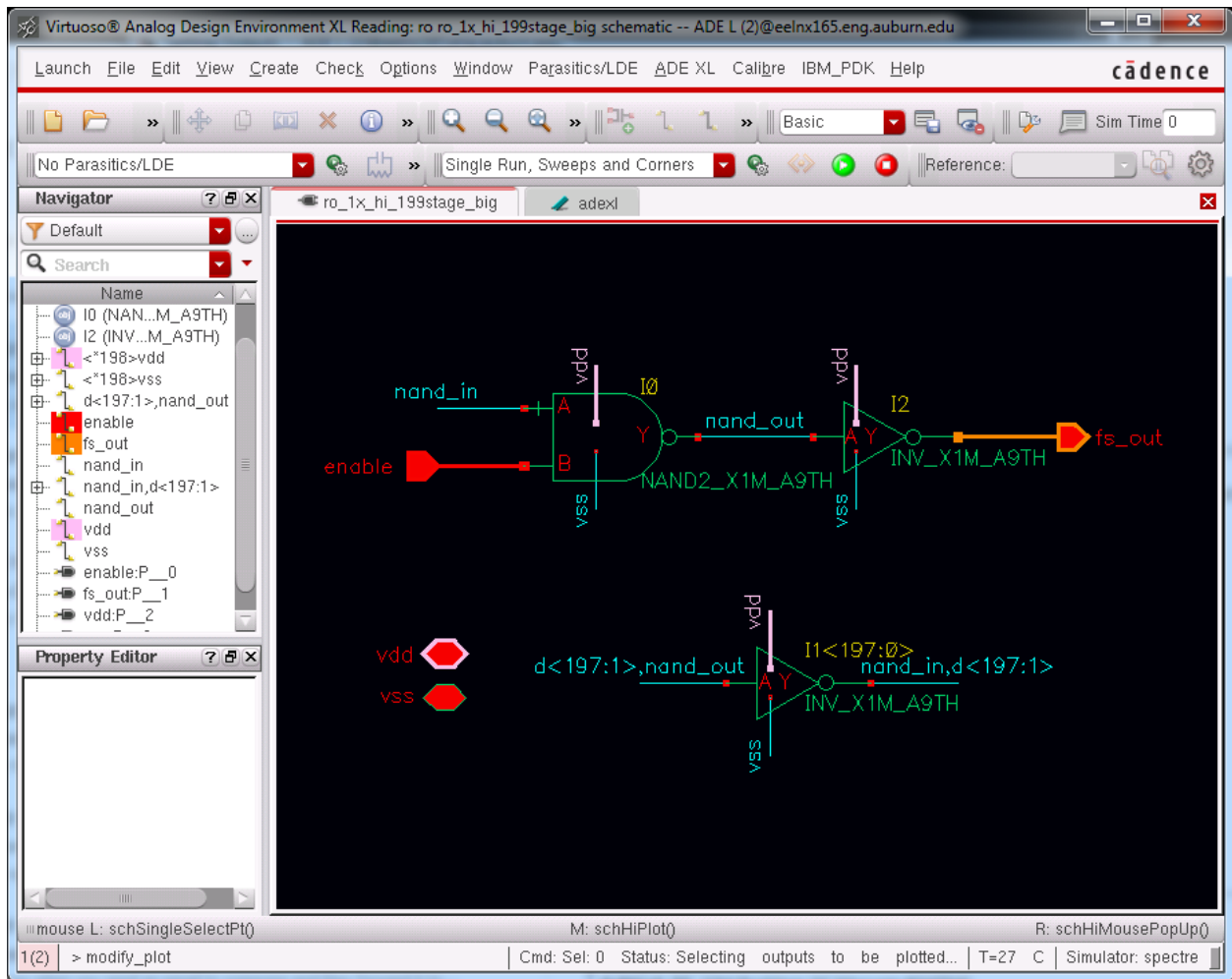


Next, click on the **Editing Design Variables** button, right below the **Choosing Analyses** button. For this RO, there is one voltage source, and we may want to easily sweep this source later for comparing frequencies versus voltage. Fill in the variable name and value as shown. If you want to sweep a variable, you can use the notation  $0.4:0.05:1.0$  to sweep from 0.4 V to 1.0 V in 0.05 V increments, or a comma-separated list of specific voltages, such as  $0.4, 1.0$ , to only simulate at 0.4 V and 1.0 V.

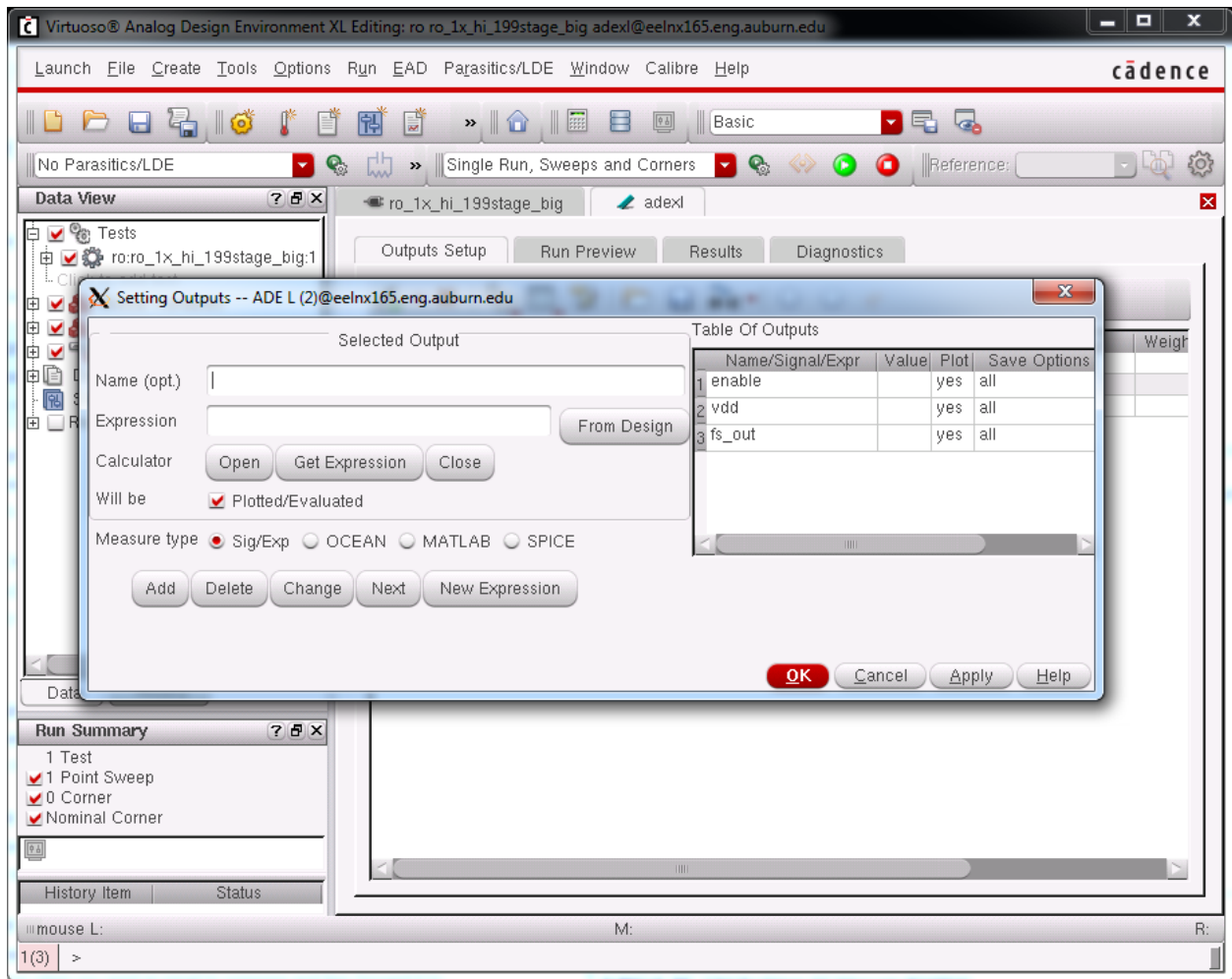
If you wish to make changes later to the design variable, it is recommended to edit the **Global Variables** list in the main ADE XL window, as these values can override design variables.



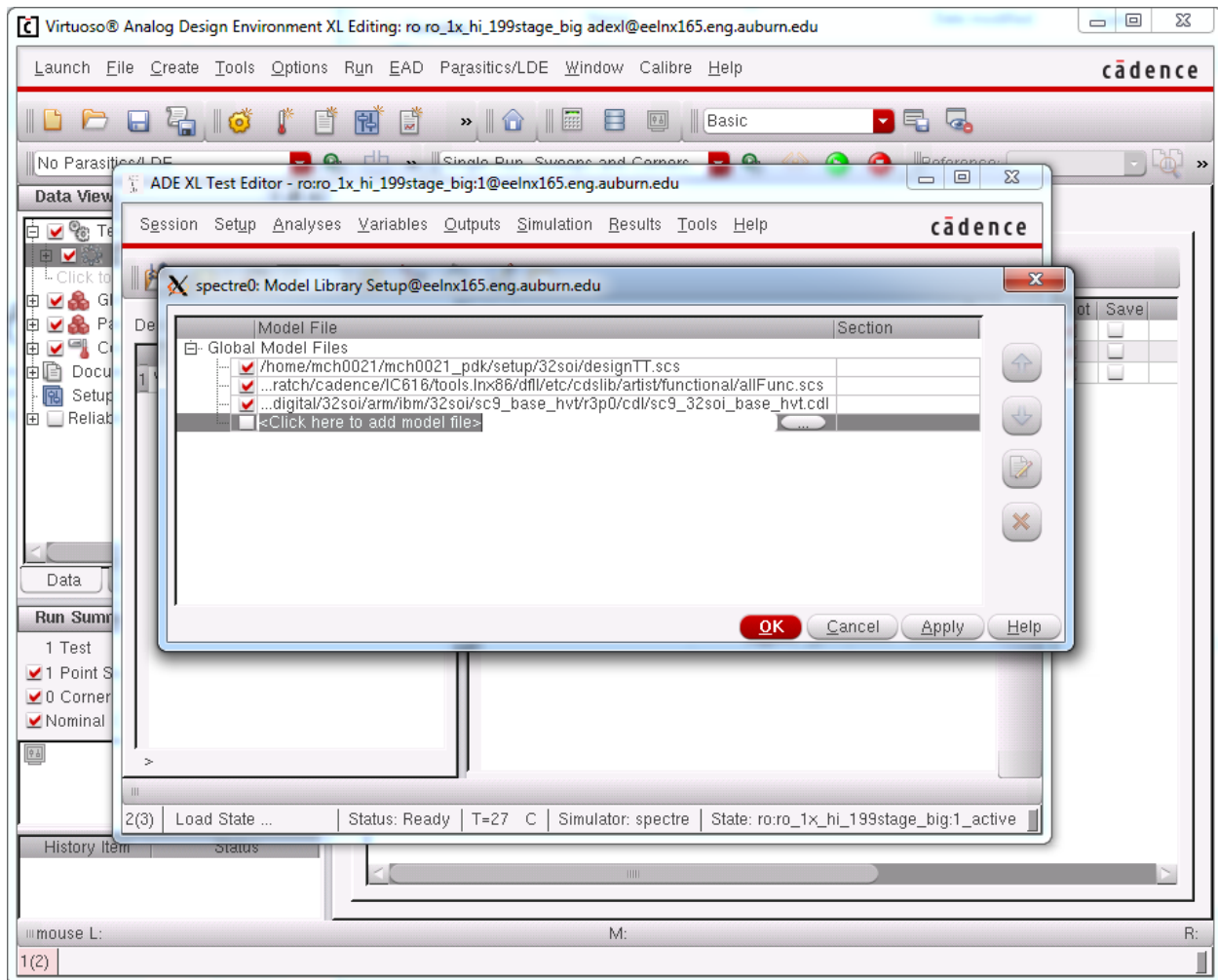
You can now select which outputs to plot by clicking the **Setting Outputs** button, below the **Editing Design Variables** button. Then click **From Design**. This will cause ADE to pull up your schematic and allow you to click on specific nets you wish to plot.



It may be useful for debugging purposes to plot `vdd` and `enable` as well as the desired `fs_out`. You must press `Esc` immediately after you are done selecting nets in the schematic window, else you may inadvertently add or remove outputs at a later time in the schematic window.

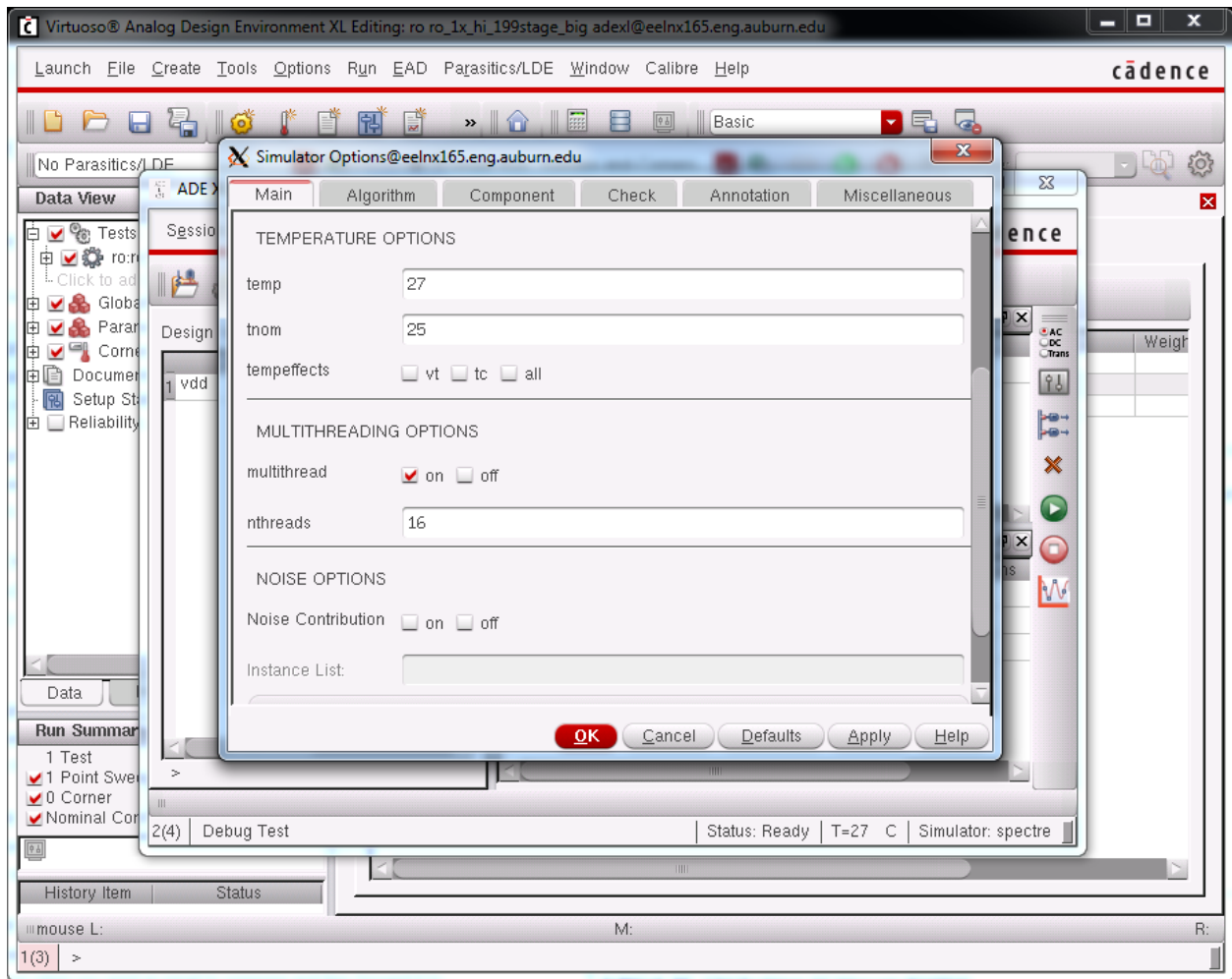


Once you're done adding outputs, click **OK** to proceed.

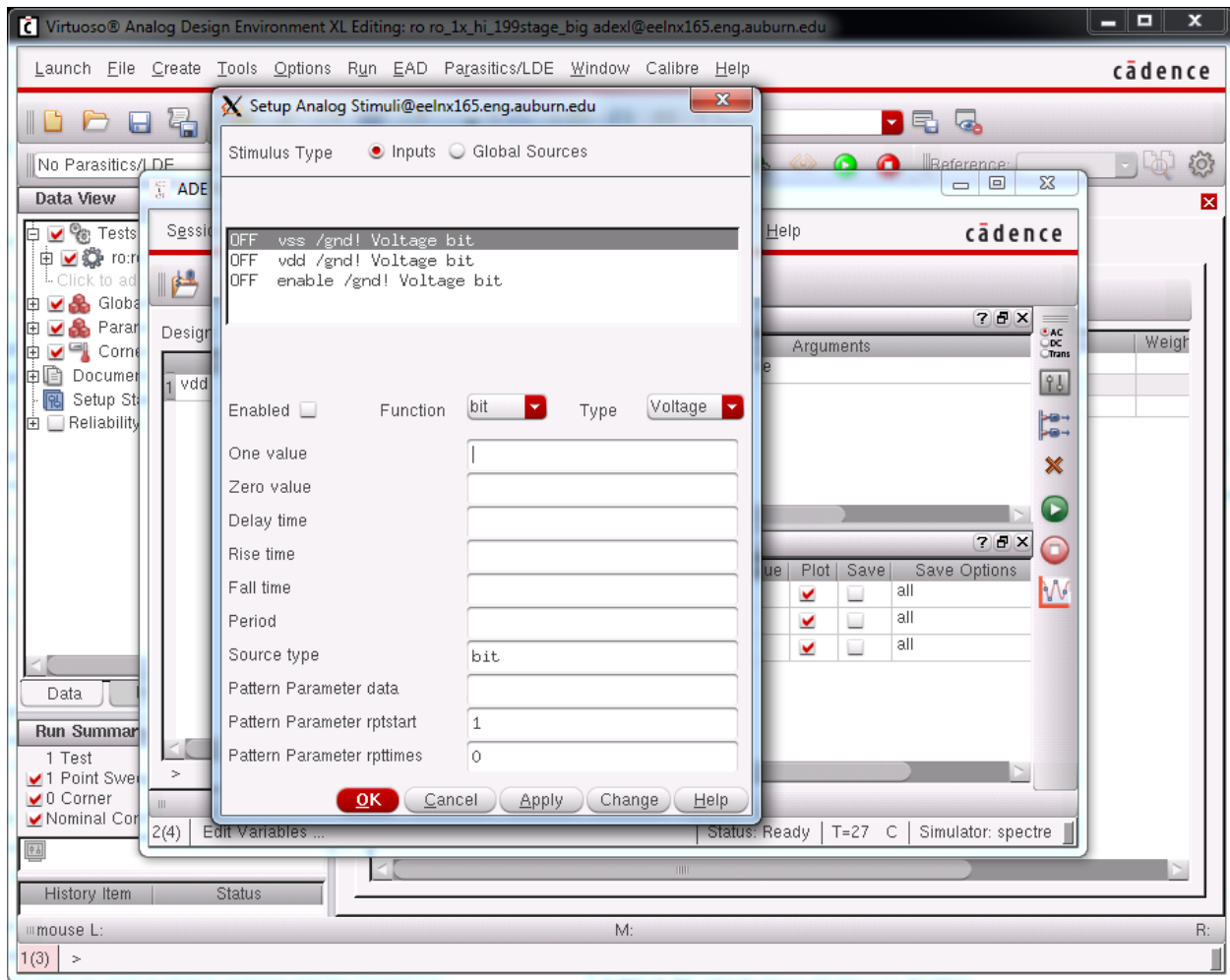


In the **ADE XL Test Editor** window, click **Setup** > **Model Libraries...** to select which model libraries the simulator references while running. If you are simulating anything with standard cell libraries, you should ensure that you have an auCd1 view alongside the symbol and layout views, as well as include the CDL netlist as provided by the standard cell library vendor. The auCd1 view is simply a copy of the symbol view, renamed as auCd1, which provides the netlist with the required pin information for netlist extraction.

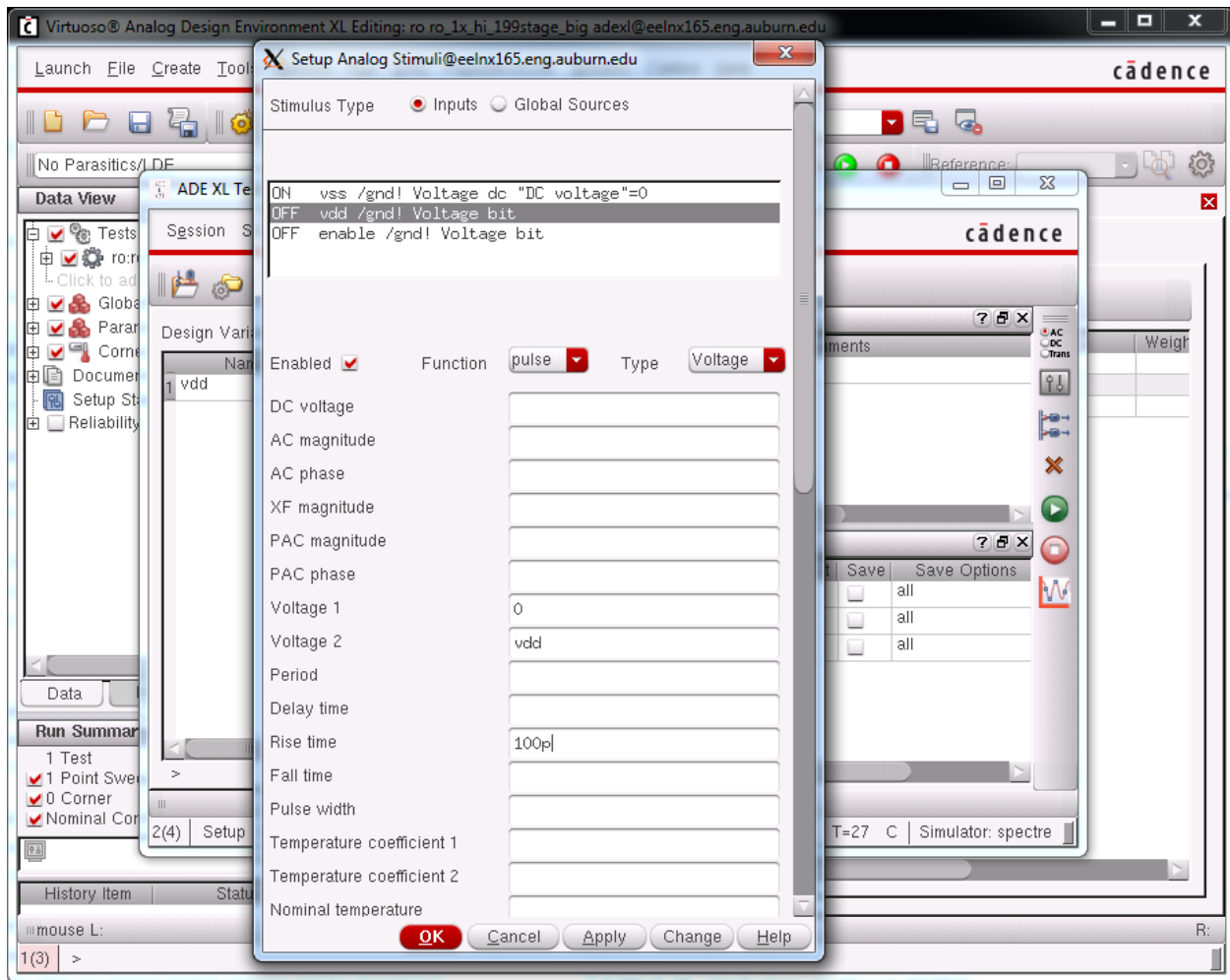




Another problem with the default ADE XL setup is the multithreading options. Click on **Simulation** **>** **Options** **>** **Analog...** and ensure that **multithread** is set to **on** and **nthreads** is set to a reasonable number like 16.

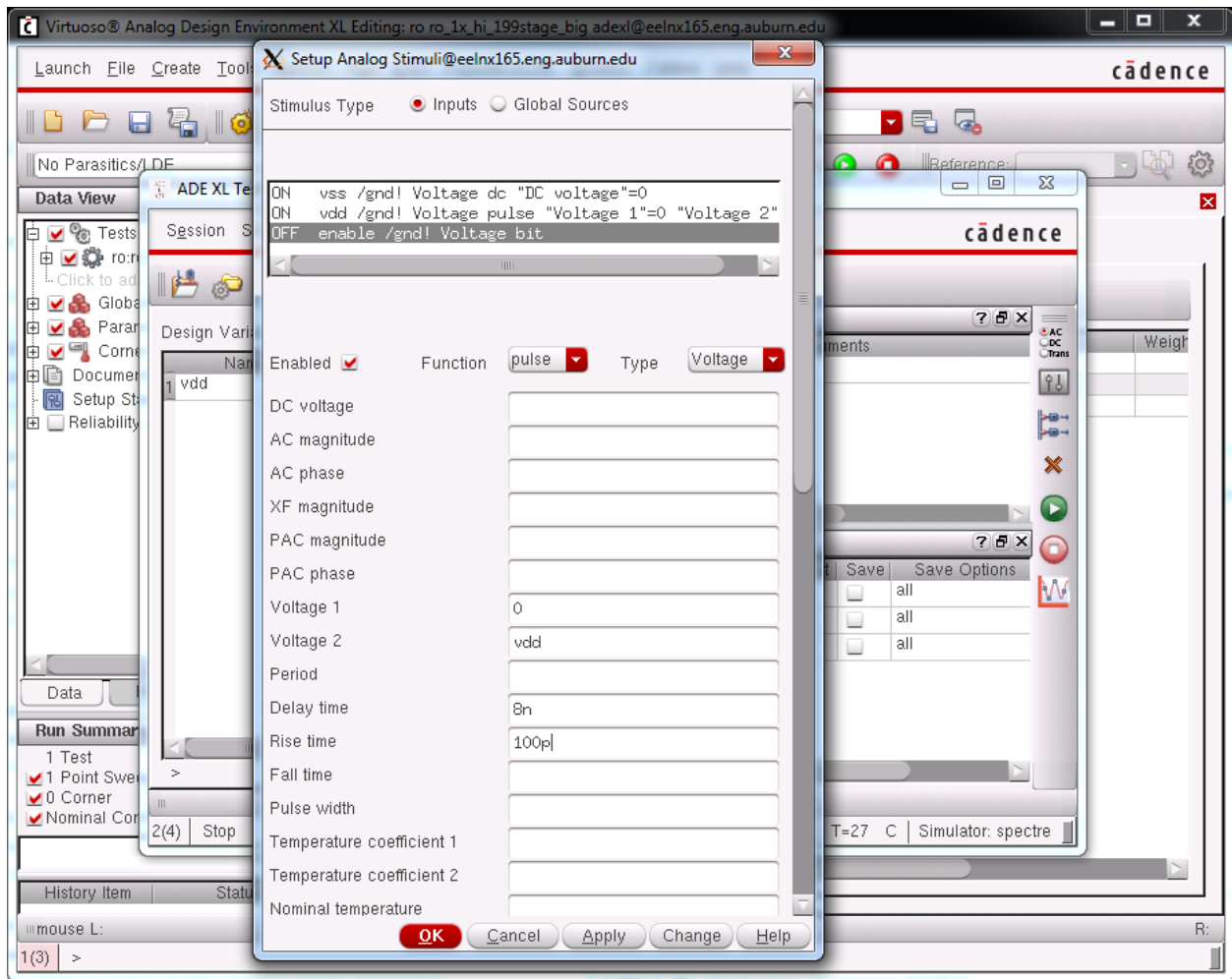


Though we have defined `vdd` to 1 V, the simulator does not know where to apply any voltage sources; though `vdd` shares the same name as the port, we must either set up stimuli or add voltage sources within the schematic. Since this schematic is also used for layout and LVS purposes, setting stimuli is appropriate. Click on `Setup` `Stimuli...` to begin.

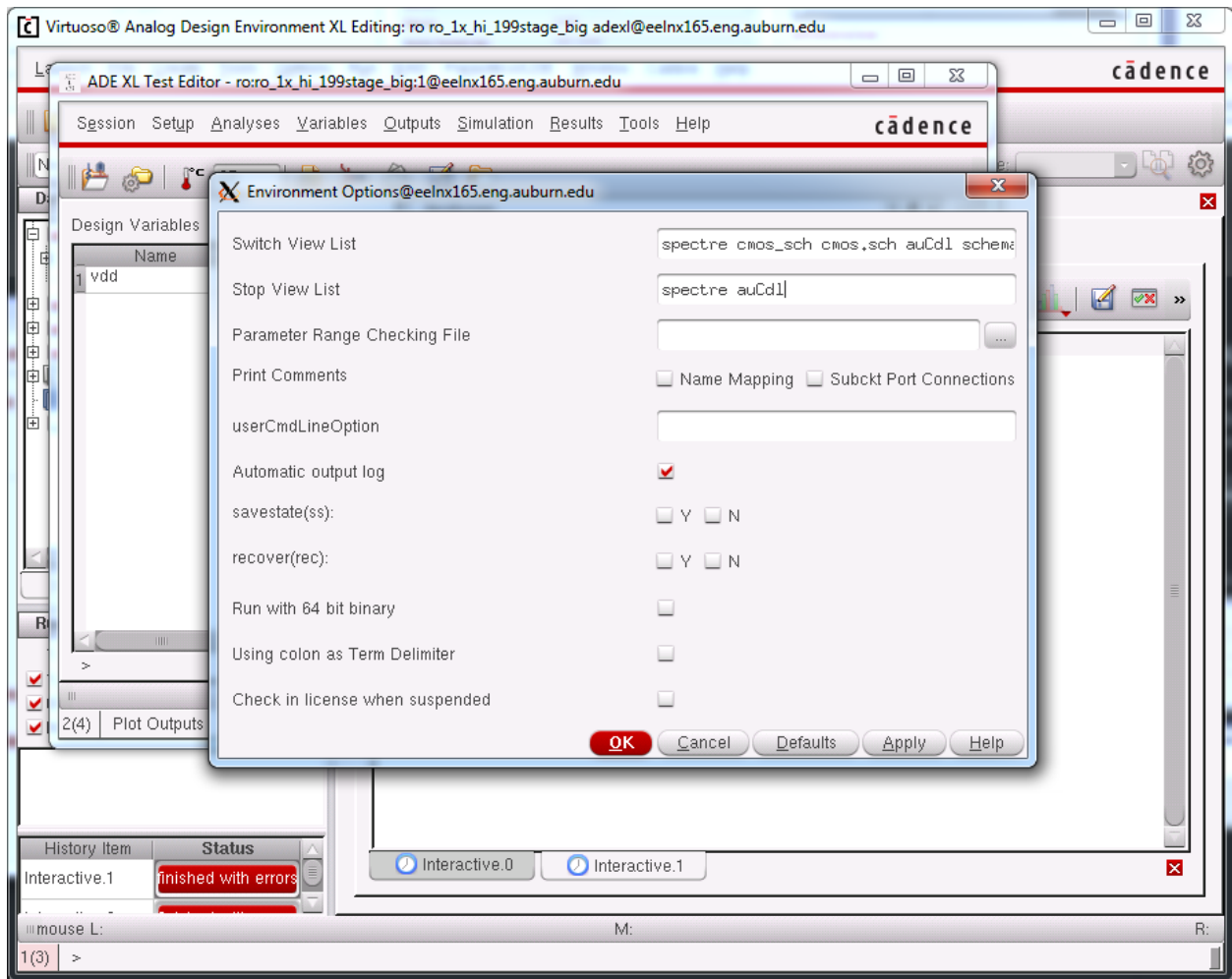


For `vss`, ensure that the stimuli is enabled, and select the function of `dc`. Then, set the `DC voltage` field to 0, which will effectively ground the node through a 0 V voltage source. Be sure to click `Apply` after any changes.

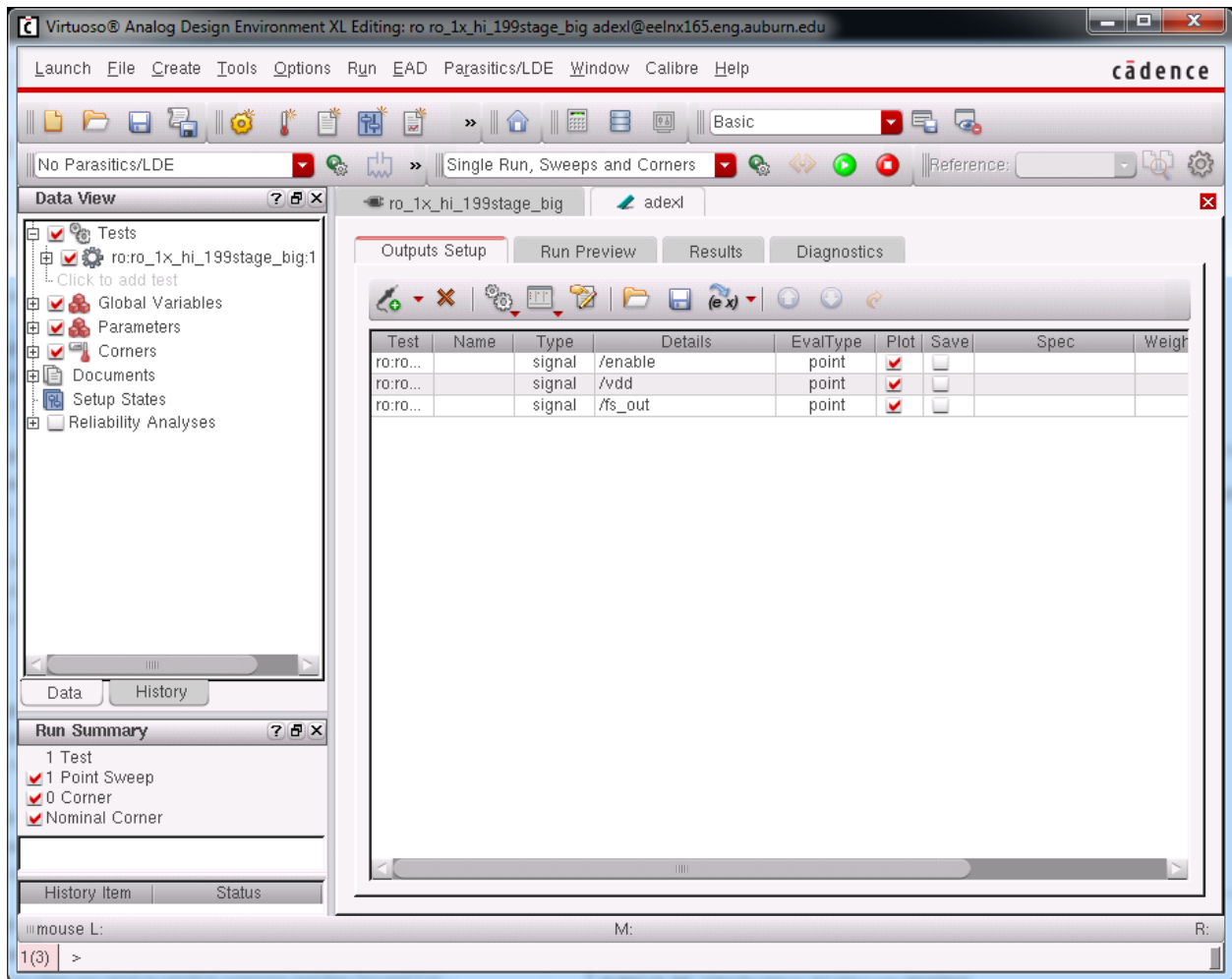
For `vdd`, select the `pulse` function, which will allow the voltage applied to turn on gradually. This will help initialize the ring oscillator in a manner that will produce oscillations. Set the rise time to 100p for 100 ps. Also, set `Voltage 2` to `vdd`, which references the global variable we set earlier. This will get replaced with the numeric value at the start of the simulation automatically.




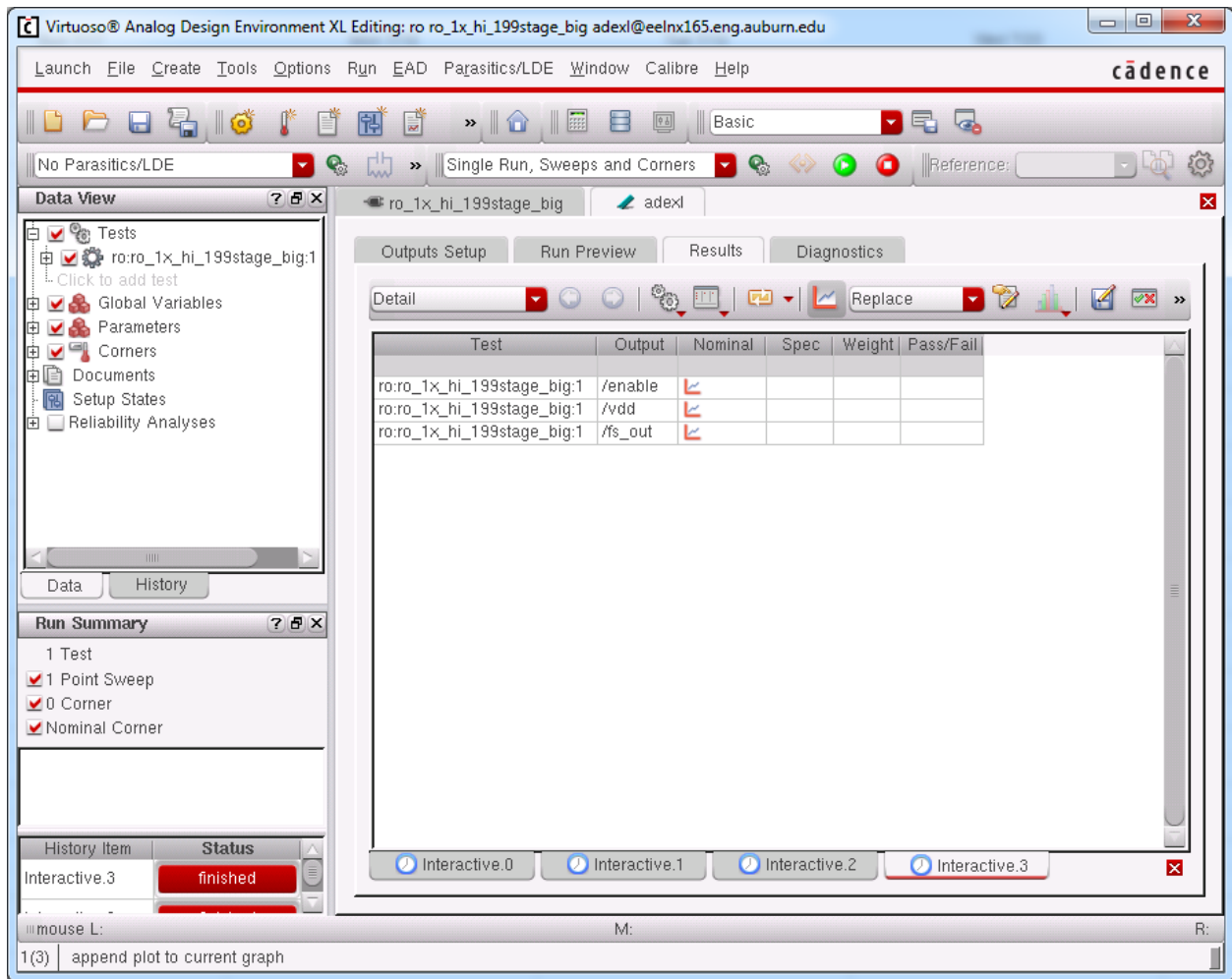
Lastly, for `enable`, we will again use the pulse function with the same parameters, but will add 8 ns of delay. This will allow the RO to become initialized to its static disabled state prior to enabling it. If this value is too short, some inverters will be in their inverted state; due to the fast nature of the enable pulse, those states will encircle the ring continuously, forming an undesirable output waveform. Click `OK` when you are done.



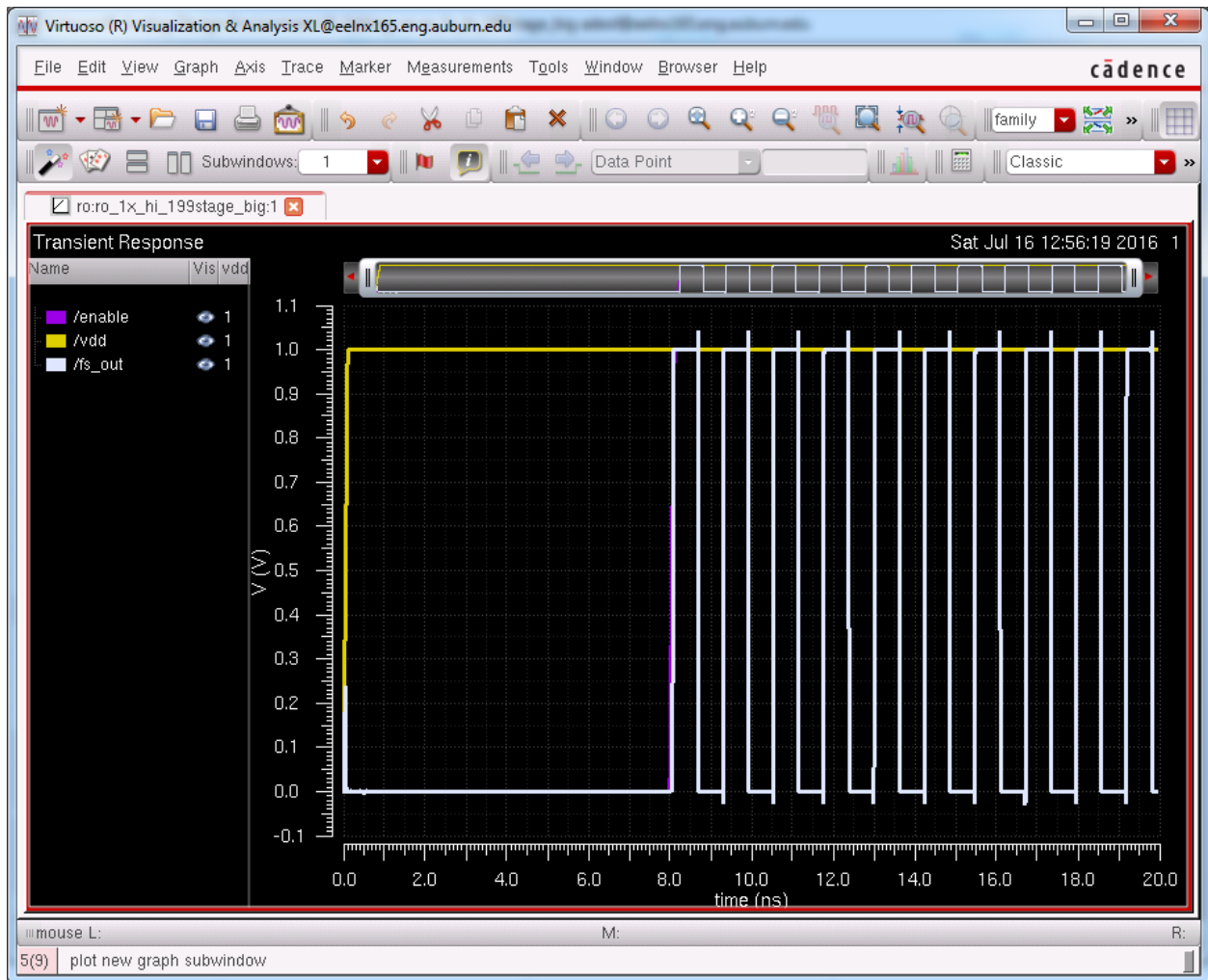
The last setup portion requires telling the netlister which views to use during netlisting. Click on **Setup** > **Environment...** to bring up the **Environment Options** window. Add **auCd1** before **schematic** in the **Switch View List** and after **spectre** in the **Stop View List**. Click **OK** when you are done.



Go back to the main ADE XL window and click the  button. This is not the same as the similar-looking button in the test editor window. The simulation has now started and may take a few seconds to many hours to complete depending on the complexity of the circuit. For a RO with no parasitics and 199 stages, it will likely take several minutes to generate 50 ns of data.

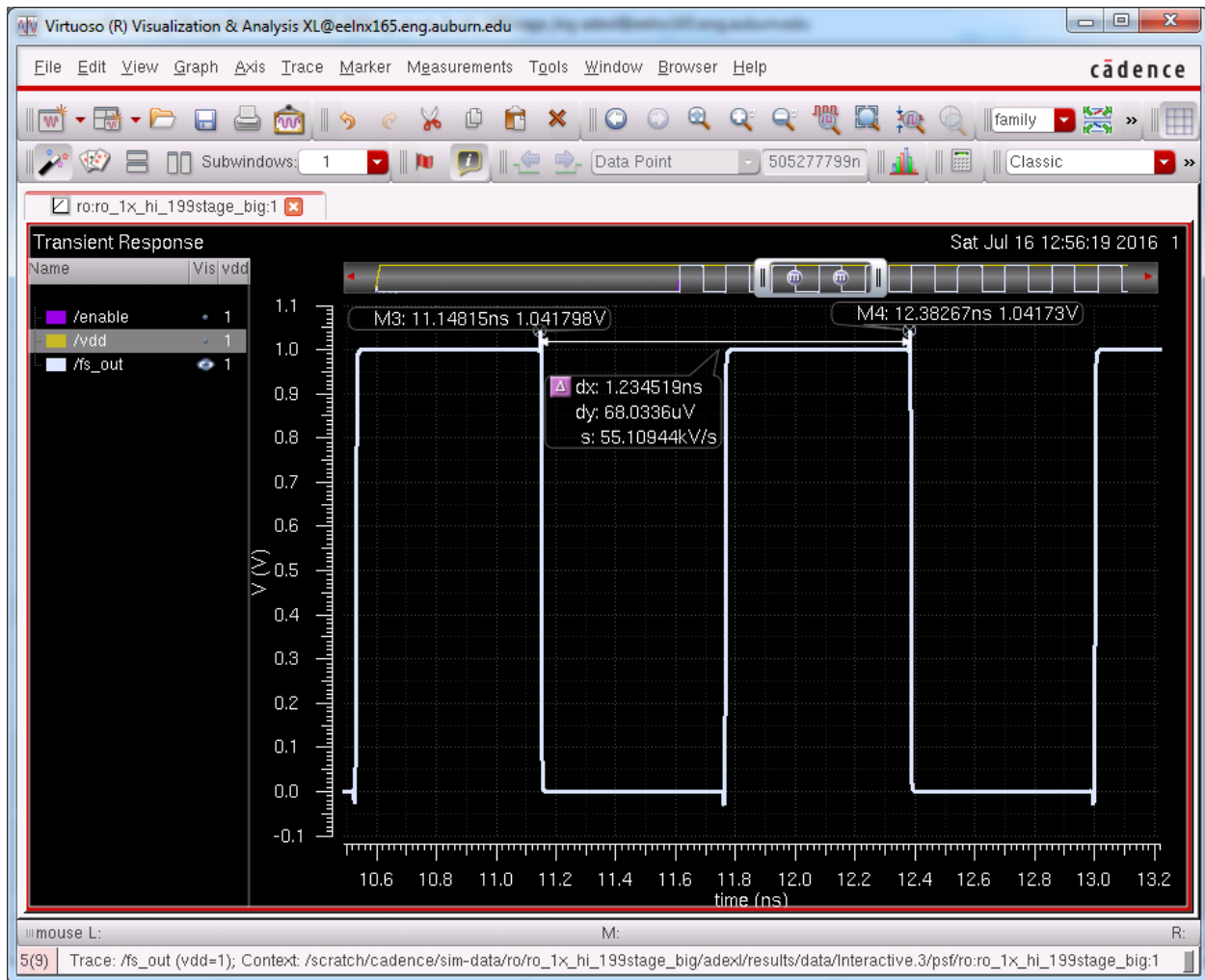


Once the simulation has completed, you should see a graph icon next to each output under the **Results** tab. Click the graph icon button next to the **Replace** drop-down box to bring up the graph window.

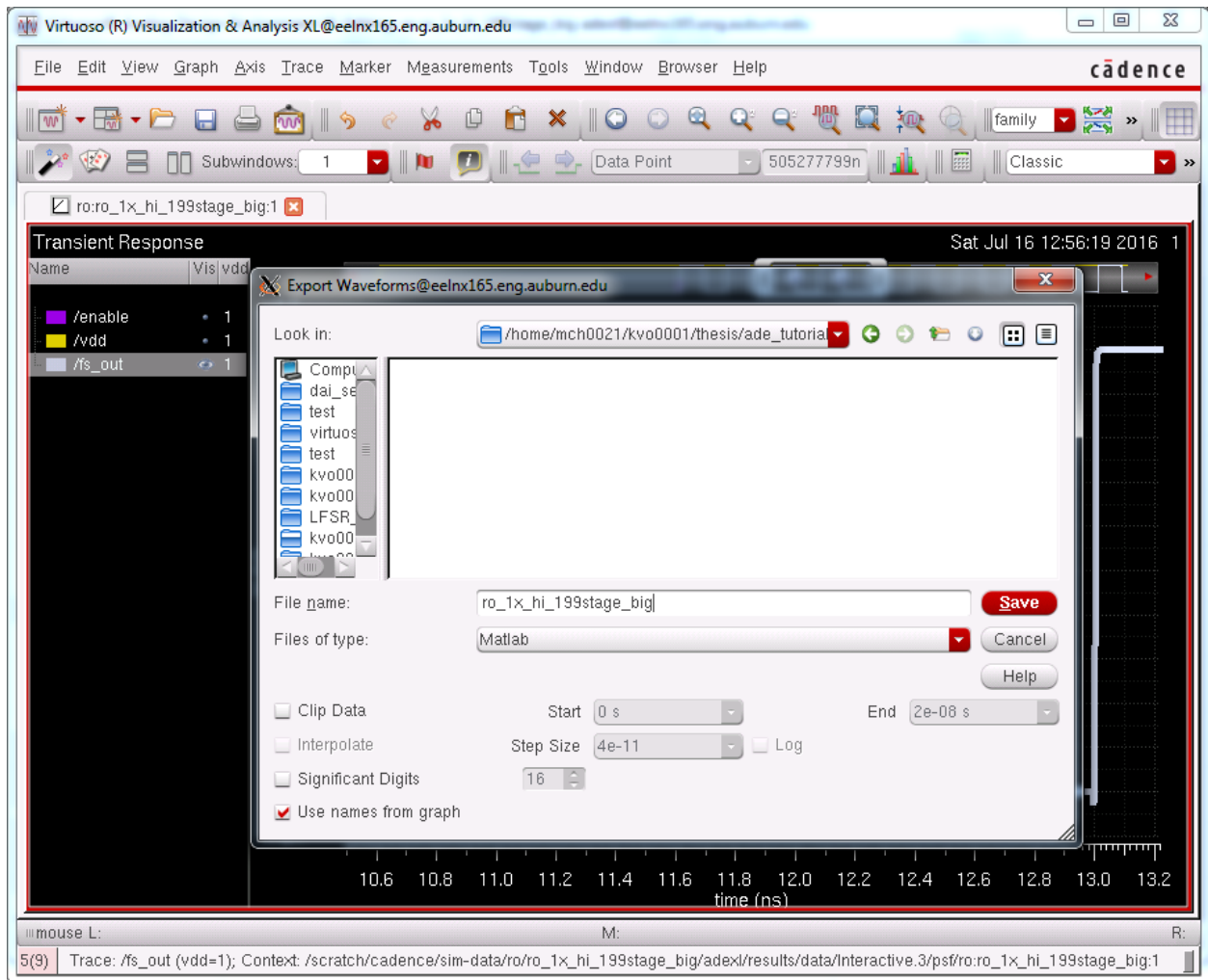


The graph window is now displayed with all signals on top of each other. If you wish, you can right click on each signal and move it to a new strip. You can also toggle their visibility by clicking on the eye symbol.





You can zoom in horizontally by moving the two sides of the view window above the main graph. Now, move your mouse over one of the peaks of the output and press **M**. This will create a marker. Do it again for the next peak, then press **ctrl** while clicking the peaks to make sure they are both selected. Finally, press **shift** + **↑** + **D** to get the delta value between both peaks. As you can see, the period for this RO is approximately 1.23 ns, or about 813 MHz.

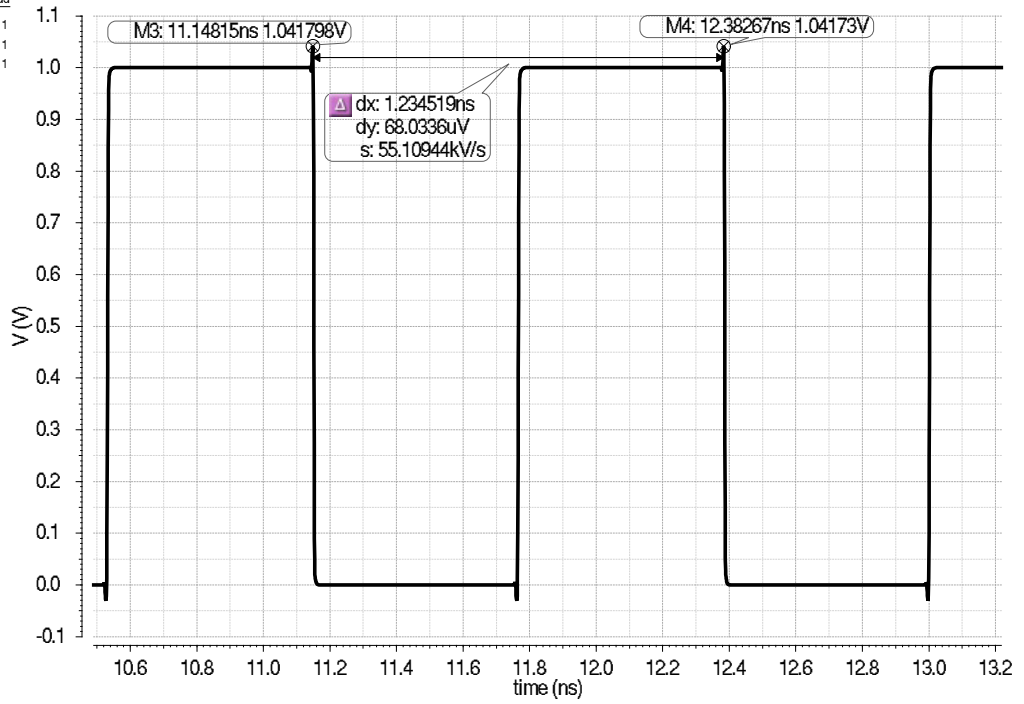


In order to properly save simulation data for use in other applications, right click on the waveform you wish to save and click **Send To** **Export...**. Select the **Matlab** option, which saves in a convenient format for importing into MATLAB.

Transient Response

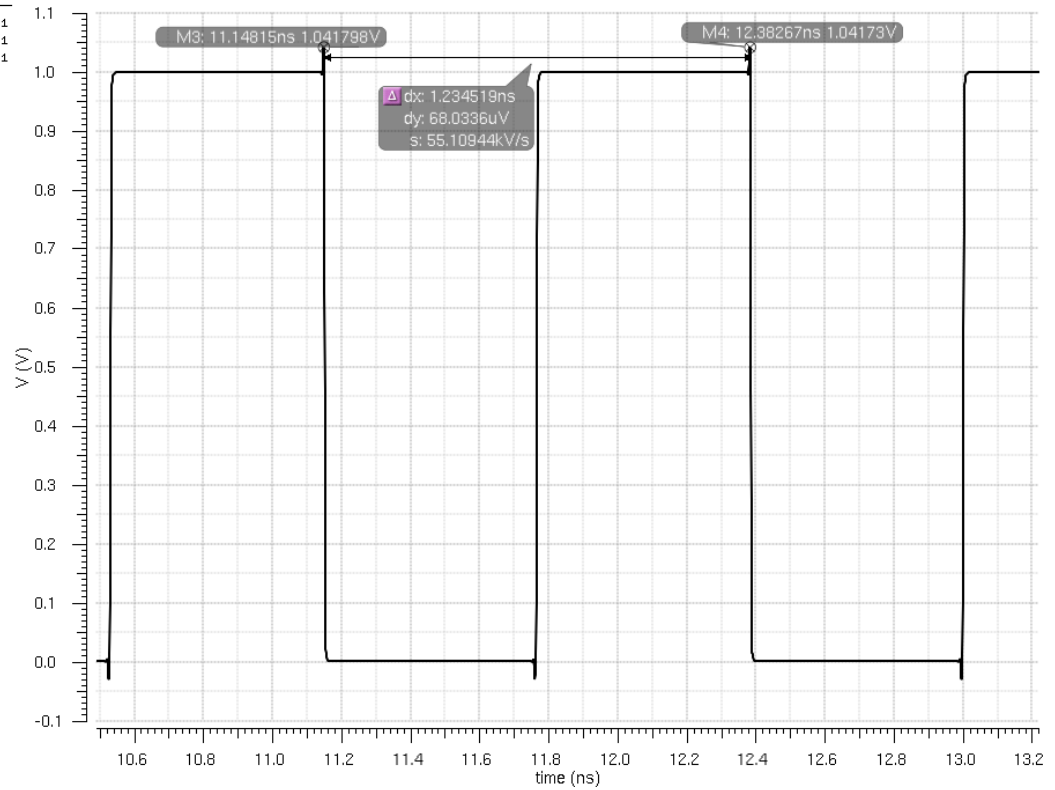
Sat Jul 16 12:56:19 2016 1

Name	Vis	vdd
/enable	•	1
/vdd	•	1
/fs_out	•	1



If you wish to save just the plot, saving as an Encapsulated PostScript is not a very good idea. Though many of the objects in the EPS are vectors, not all of them are, and it just looks bad.

Name	Value
/enable	1
/vdd	1
/fs_out	1



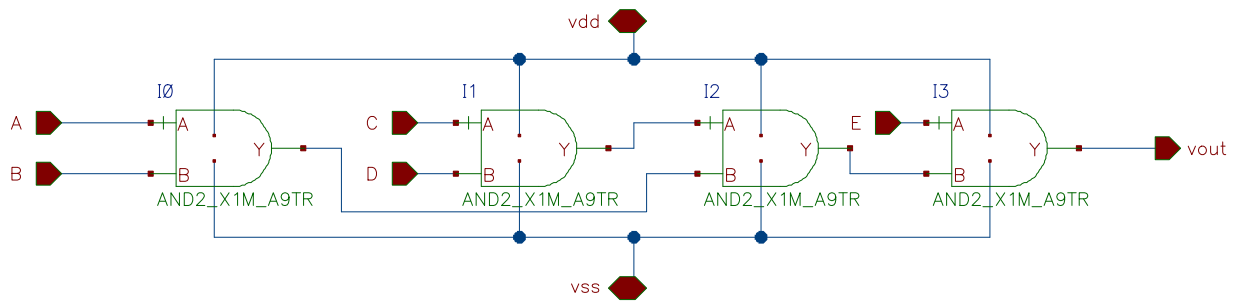
Saving as a Portable Network Graphics file looks somewhat better, but it is not presentable when scaled. The best option is to use the exported MATLAB data and plot it externally.

## Appendix E

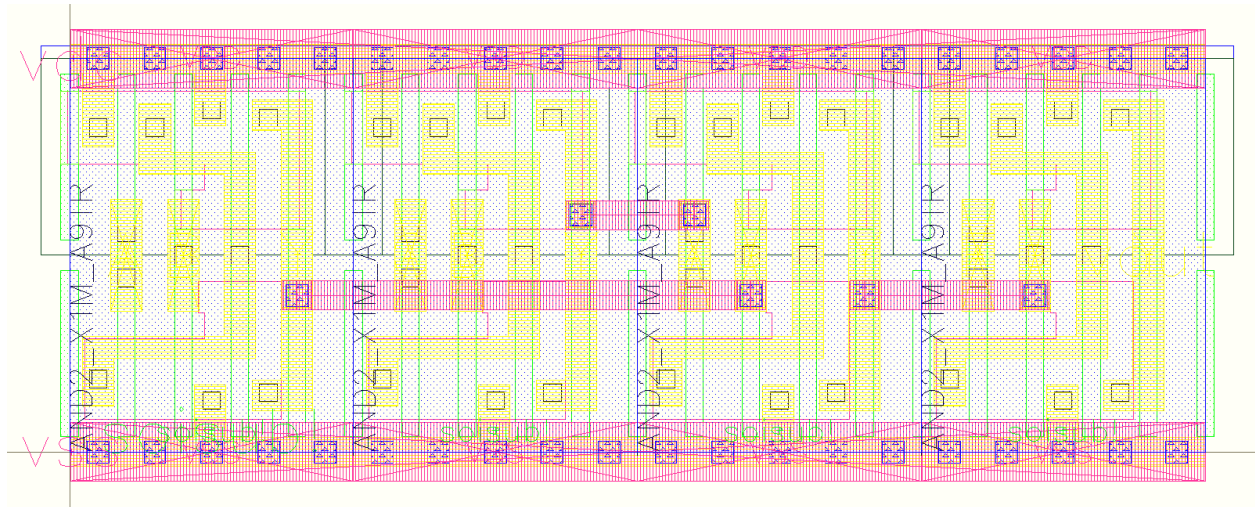
### LVS Tutorial

Layout versus Schematic, or LVS, is a powerful tool for design checking. It compares a schematic, which has hopefully been simulated at this point to check for logical correctness, to an extracted netlist from the layout. LVS takes the layout, flattens all of the shapes across each layer, and constructs a new netlist based on this information. Thus, LVS can find errors in connectivity, such as wires that are not connected as they should, or shorts across nets. LVS is a vital component of the design process, even for “simple” designs. LVS goes hand in hand with design rule checking, or DRC, which will check if the layout is fit for fabrication.

This tutorial will introduce LVS from the beginning, starting with a completed schematic and layout.



The schematic is a simple 5-input AND gate constructed from 2-input AND gates. Though this is a very simple schematic, LVS will happily handle much more complicated designs, but may take a bit more time to process. LVS should be used from the beginning, all of the way through the hierarchy to the top level. Waiting until the top level to try to start running LVS is a recipe for disaster as it can be difficult to pinpoint where exactly the design is failing. Likewise, breaking a big design up into smaller portions can make LVS a much easier process for the same reason.

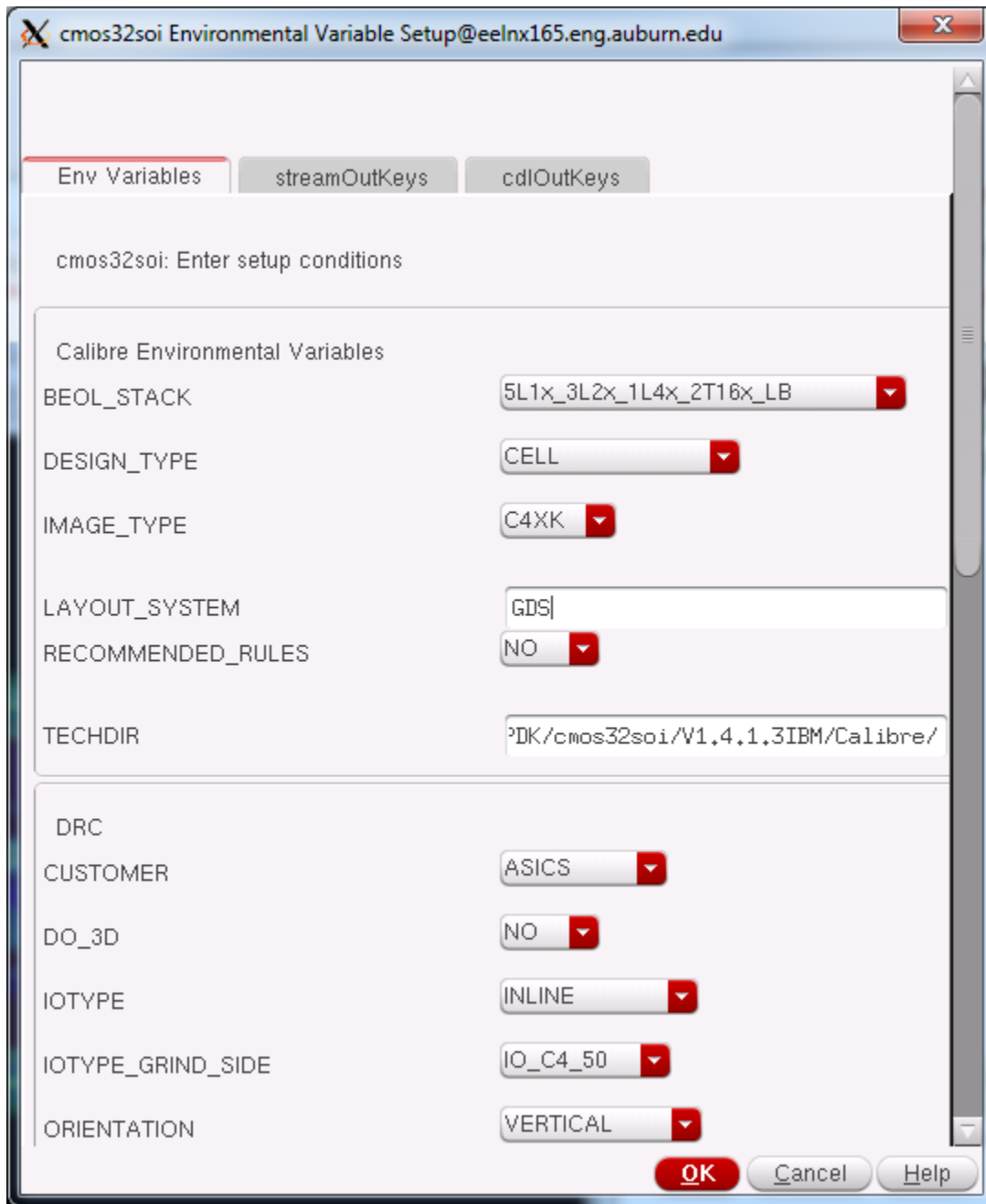


The layout is also relatively simple and is constructed from standard cells. All wiring has been completed. Wiring can be assisted using connectivity information from the schematic if you choose to launch `Schematics XL` and `Layout XL`; launching one will typically automatically launch the other. When routing in the layout, the netlist information is used to show where pins are to be connected, and flashing markers alert you to when it is connected incorrectly. This is still no substitute for LVS, as this only handles partial connectivity information; LVS also checks for device parameters.

Another step during layout is to draw pins. Pins allow for easy routing in the next level of the hierarchy using `Layout XL` as discussed previously. Pins also have an inherent label, which is required for LVS. If a port is used in the schematic, a label must be drawn in the layout. Pins give more information than labels alone and are recommended. If labels are missing, LVS will generate an error saying it found a port in the source (schematic) but not in the layout. Likewise, if extra labels are drawn in the layout, LVS will also generate an error. Remember to draw labels on the same layer as the metal, but use the `label` purpose instead of `drawing`.



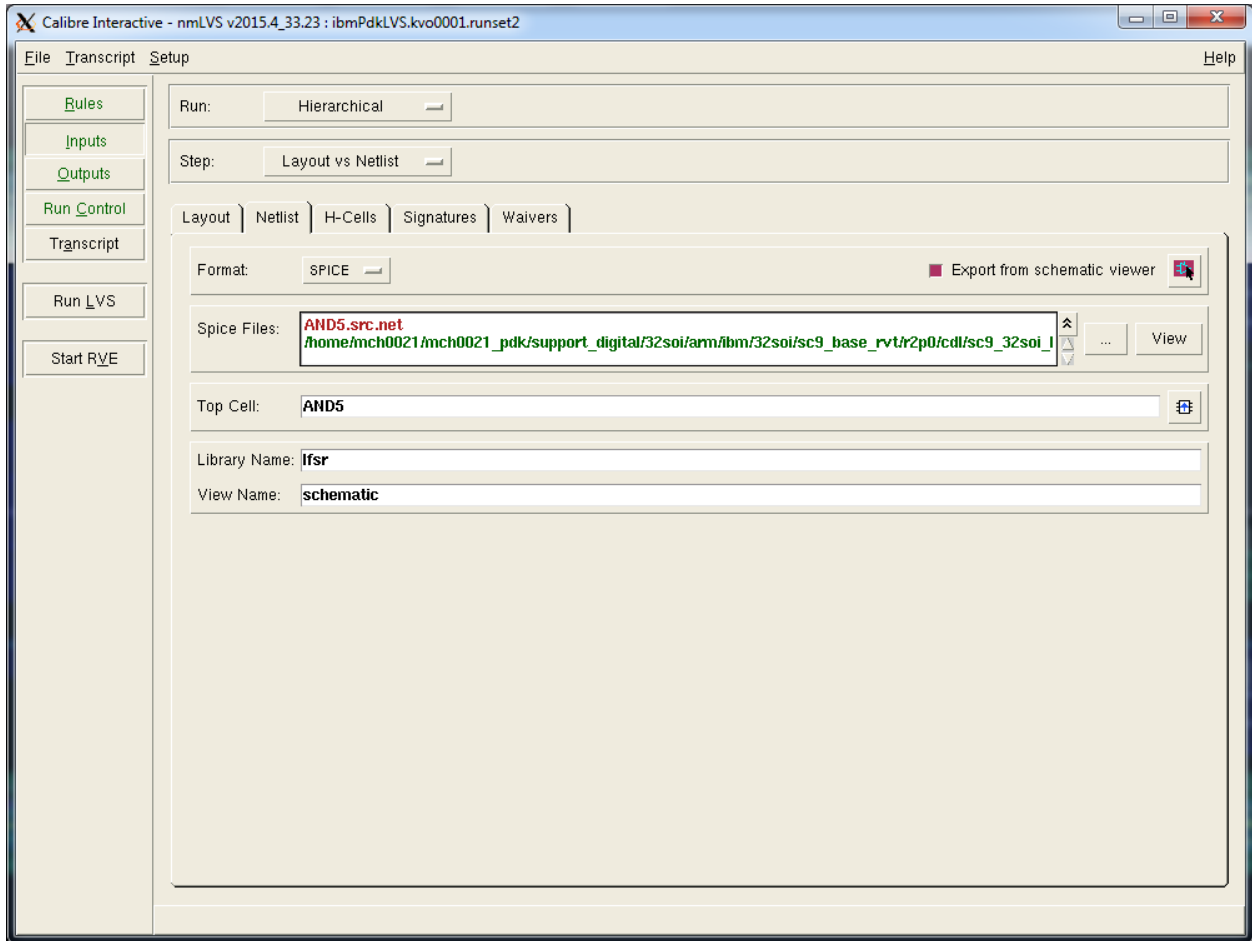
As seen above, pins must be drawn over existing pins if they are deeper in the hierarchy, as is the case with using standard cells. LVS will only read the pins from the top level layout to match against the schematic ports of the top level schematic. The `VDD` and `VSS` pins are not shown above and reside on another layer.



For some kits, there is a checking menu provided by the vendor. If so, launch Calibre LVS from there. Otherwise, launch Calibre LVS from the `Calibre` menu. The `BEOL_STACK` option will need to match the technology file used. The `DESIGN_TYPE` option should be `CELL` for portions of the chip not containing the chip boundary layers, and should be `CHIP` otherwise. `CUSTOMER` should be set to `ASICS`, and `ORIENTATION` should be set to whichever orientation the gates of the transistors are drawn. In many processes, this can only be



**HORIZONTAL** or **VERTICAL**, not both. For most cases, this will take care of all of the initial setup for LVS, but when in doubt, consult the LVS manual provided by the vendor. Click **OK** when done.

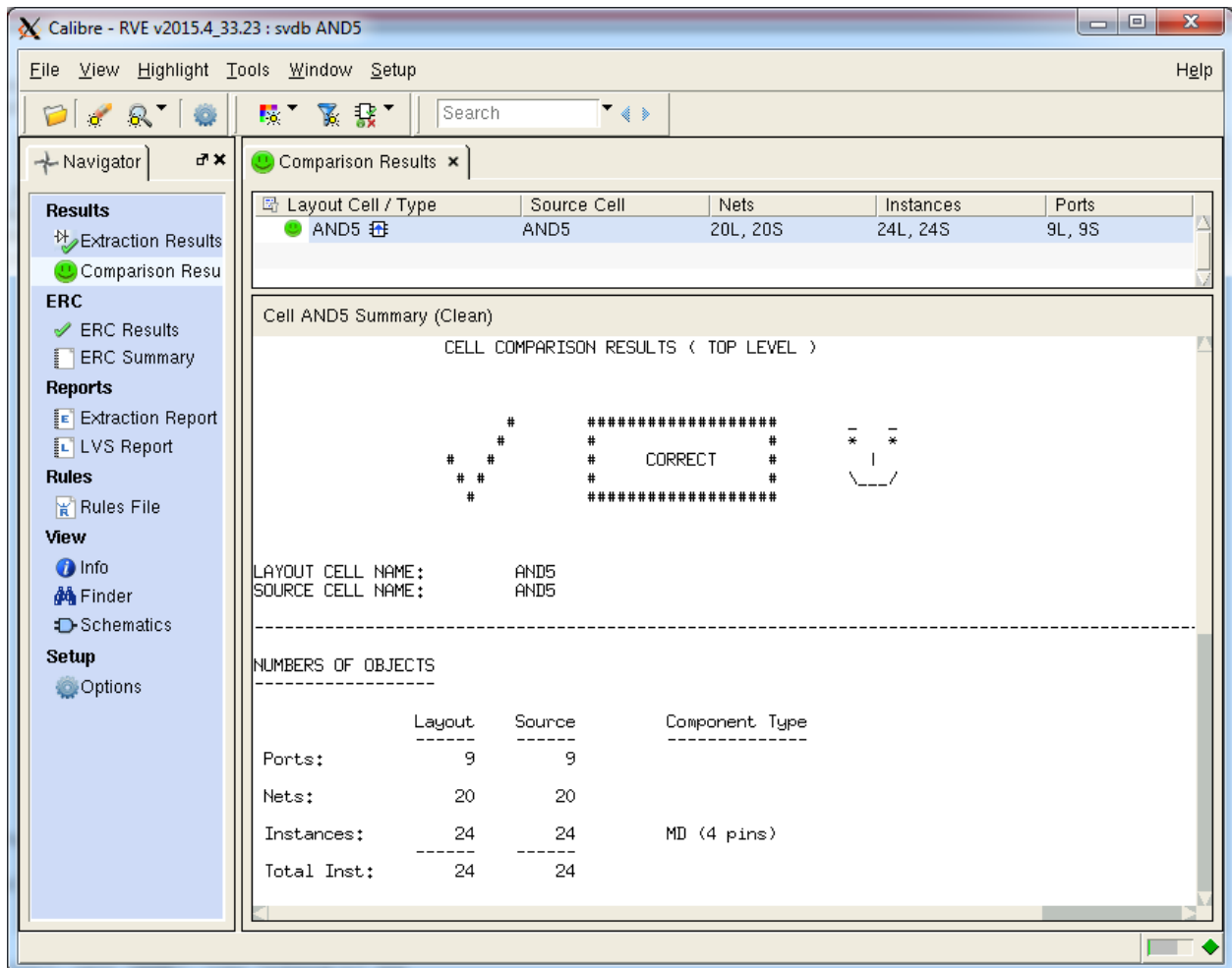


Once Calibre LVS has launched, navigate to the **Netlist** tab under **Inputs**. Ensure that the schematic will be exported. The file may be **red** if it does not exist or is not readable by the current user. The former option is acceptable, as the export step will create it. If it is not writable, ask the owner of the file to change its permission to allow you to write to it. A **green** file means that it already exists, but still ensure that the schematic is exported, as the current one may be out of date if the schematic was updated.

Click in the text box and add a new line. Then, use the **...** button to browse for a new file. You will need to locate the standard cell library's CDL netlist as the standard cell

library does not typically come with schematic views. If this is not the case, you can skip this step.

If you are trying to match a layout view under a different name than its schematic, you can change the `Top Cell` and even `Library Name` here too. There is also support for using a Verilog netlist instead of a SPICE netlist; you can experiment with this with the `Format` menu.



If the LVS run was successful, you will be presented with multiple 😊, including an ASCII version, as well as the `CORRECT` statement. If you did not pass, you will see `INCORRECT` as well as `Xs` in several locations.

Common errors include:

- Forgetting a pin or label in the layout

- Putting too many labels in the layout
- Not connecting all nets of the same name, such as multiple VDD or VSS rails (this generally generates a warning and may be acceptable at low levels of the hierarchy)
- Watching carefully for inherited connections (i.e., those ending in !)
- Incorrect parameters between schematic and layout
- Drawing into a device such that the parameters inadvertently change (i.e., drawing more metal inside a capacitor)
- Shorting two nets together
- Not connecting two portions of a net

One good tip to improve the likelihood of finding errors is to break the schematic into decreasingly smaller pieces until the error is found. Dividing the schematic in half to find which half fails, over and over again, can be useful if the schematic is quite large. Some errors can be cryptic at times, and it may be best to use the results window, called the RVE, to open its schematic representation of both input netlists. Sometimes, further information can be gained by looking at what it thinks is connected. However, the RVE can have a difficult time interpreting multiple transistors as a single gate and may lead to more confusion.

## E.1 lvs.runset Contents

```
1 *lvsRulesFile: /home/mch0021/mch0021_pdk/global_foundries/cmos14lpp/
  ↪ REL_GF/CalibreLVS/LVS/ln14lpp.lvs.cal
2 *lvsRunDir: ./calibre_run
3 *lvsLayoutPaths: test.calibre.db
4 *lvsLayoutPrimary: test
5 *lvsLayoutLibrary: test
6 *lvsLayoutView: layout
7 *lvsLayoutGetFromViewer: 1
8 *lvsSourcePath: test.src.net
9 *lvsSourcePrimary: test
10 *lvsSourceLibrary: test
11 *lvsSourceView: schematic
12 *lvsSourceGetFromViewer: 1
13 *lvsSpiceFile: test.sp
14 *lvsERCDatabase: test.erc.results
15 *lvsERCSummaryFile: test.erc.summary
16 *lvsReportFile: test.lvs.report
17 *cmnWarnLayoutOverwrite: 0
18 *cmnWarnSourceOverwrite: 0
19 *cmnRunMT: 1
20 *cmnRunHyper: 1
21 *cmnSlaveHosts: {use {}} {hostName {}} {cpuCount {}} {a32a64 {}} {rsh
  ↪ {}} {maxMem {}} {workingDir {}} {layerDir {}} {mgcLibPath {}}
  ↪ {launchName {}}
22 *cmnLSFSslaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture {}}
  ↪ {minMemory {}} {resourceOptions {}} {submitOptions {}}
23 *cmnGridSlaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture {}}
  ↪ {minMemory {}} {resourceOptions {}} {submitOptions {}}
24 *cmnFDILayoutLibrary: test
25 *cmnFDILayoutView: layout
26 *cmnFDIDEFLayoutPath: test.def
27 *cmnPromptSaveRunset: 0
28 *cmnSaveRunsetChanges: 0
```

It can be very useful to save a runset file, which gives Calibre enough information to set up the run. Above is a sample runset file from a 14 nm kit. Loading this file at the start of Calibre is preferable to setting all of the options by hand. In the case of other technologies, the vendor's custom menu can set most of these options, but may still require some tweaking

following the custom menu. With a `runset` file, you can load as many options in as you would like, saving time in the long run.

Decomposing this `runset` file line by line, the first line sets where the LVS rules file is located. This should be within the technology's directory. Line 2 sets where Calibre generates all of the run-related files. To simplify the directory structure, it can be useful to make a new directory within the working directory and have Calibre use this instead, as is the case here. Make sure this directory exists prior to running, as it will not create its own. Lines 3 through 5, 8 through 10, 13 through 16, and 24 through 26 will be overridden when starting Calibre LVS through the `Calibre` menu in the `Virtuoso Layout` window. Lines 7 and 12 tell Calibre to export the most recent layout and schematic views from the library instead of using the (possibly) already exported database and netlist files. Lines 17 and 18 indicate that it is fine to overwrite the existing database and netlist files during export to avoid additional pop up windows. Lines 19 through 23 tell Calibre that it is fine to run multi-threaded and hyperscaled, which can tremendously speed up runs with certain technologies. In other technologies, running multi-threaded can prevent Calibre from completing due to a bug. Finally, lines 27 and 28 prevent Calibre from asking if you would like to save any changes to the `runset` file. Only the maintainer of the Cadence-related files should have permission to change such a file, so these prompts are disabled to prevent a user from trying to make changes.

It is recommended that any LVS `runset` files should be modeled after this one, which is fairly complete as far as such files go.

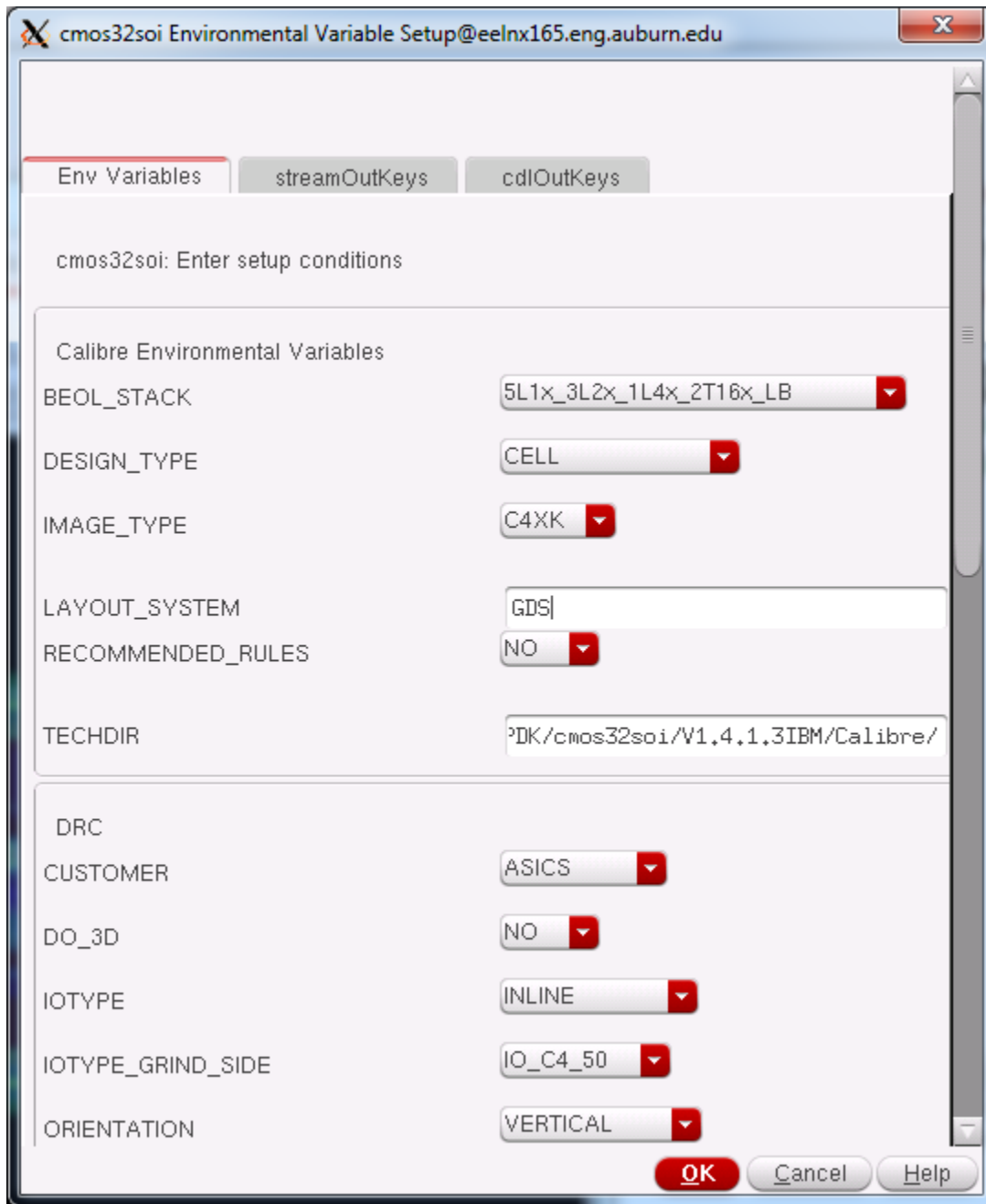
## Appendix F

### DRC Tutorial

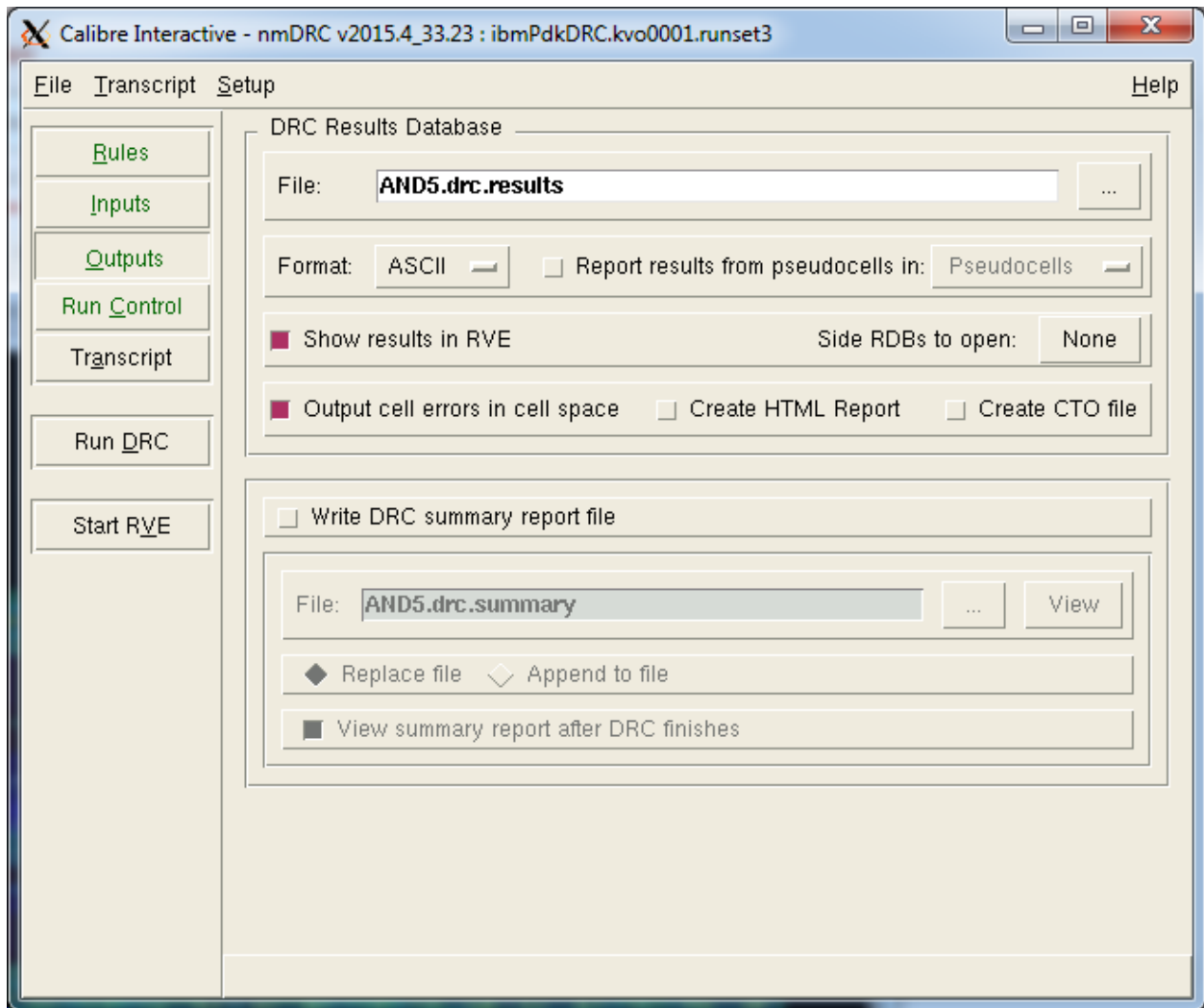
Design rule checking, or DRC, checks the layout for any potential manufacturing problems. With node sizes getting increasingly advanced, the number of checks that the DRC tool must perform is increasing as well. For some technologies, it is not uncommon to see nearly *twenty-thousand* checks. Needless to say, even with parallel processing, the complete DRC process may take several minutes to over an hour, depending on how large and complex a design is.

The rules that the DRC tool checks are varied, but the simplest of rules can check the spacing between objects, the widths of objects, and the enclosure of two objects. Of course, DRC rules may also check for shapes that simply cannot be fabricated, such as a text or label on a drawing layer. There are also minimum area rules, stating that shapes must be above a certain size to be manufactured. Density rules are also important; failures can either be too high or too low. In some cases, density rules can be waived, as the vendor or aggregator may fill the design automatically during creation of the reticle. Antenna violations can also be a concern; when too much metal is connected to a single gate, high voltage can build up during processing and decrease the chances of the IC working. This can be mitigated by putting more gates on the net, decreasing the amount of metal, or adding antenna diodes. Consult the technology's design manual for best practices.

The same layout as in the LVS tutorial will be used during the following steps.

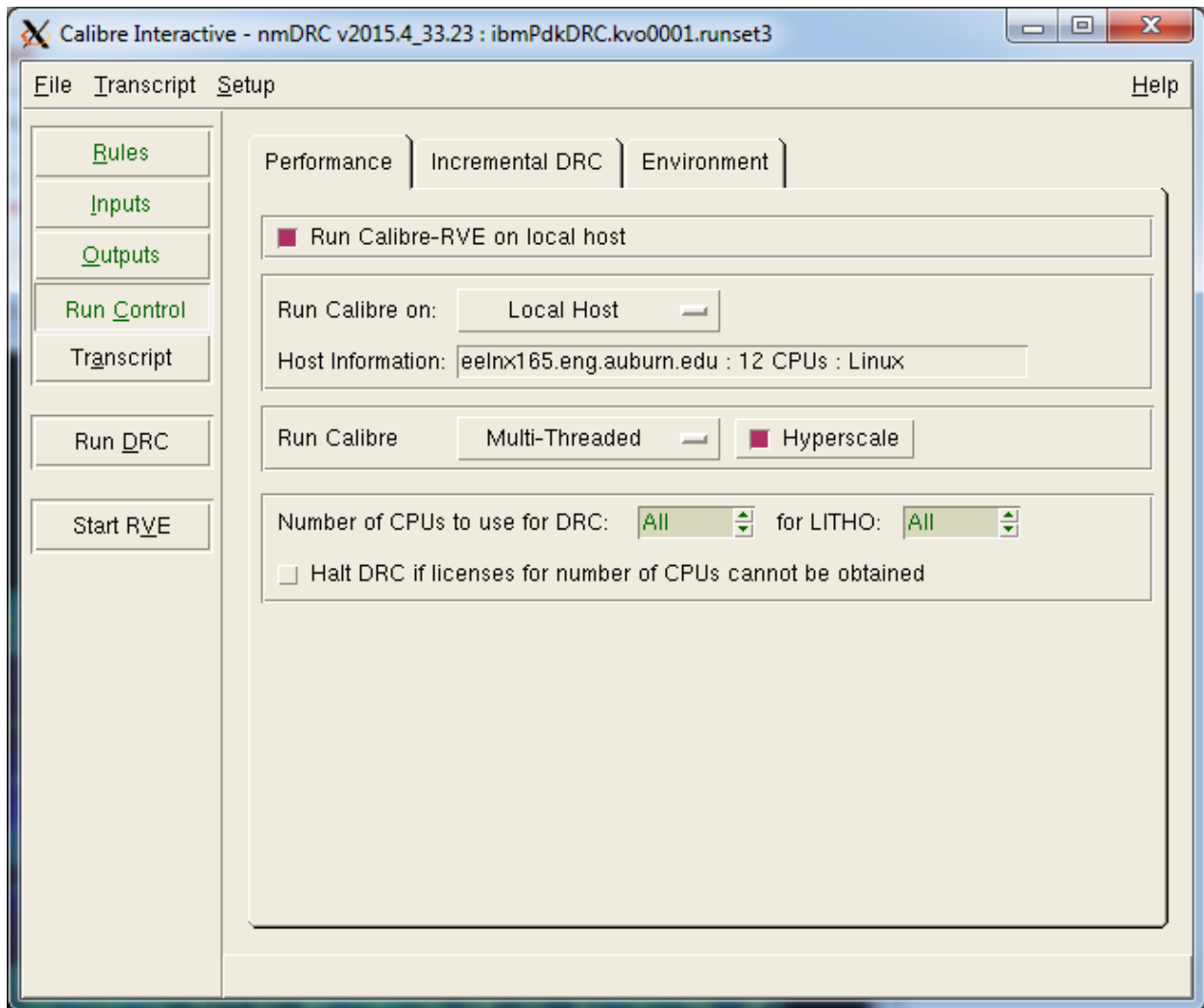


Like the LVS tutorial, options must be set up for certain technologies when a custom menu is provided. The options should be the same, but DRC is pickier about the `IMAGE_TYPE`, `IOTYPE`, `IOTYPE_GRIND_SIDE`, and `ORIENTATION` options. Recall that the orientation must be set to `HORIZONTAL` or `VERTICAL` and that designs must not have both.

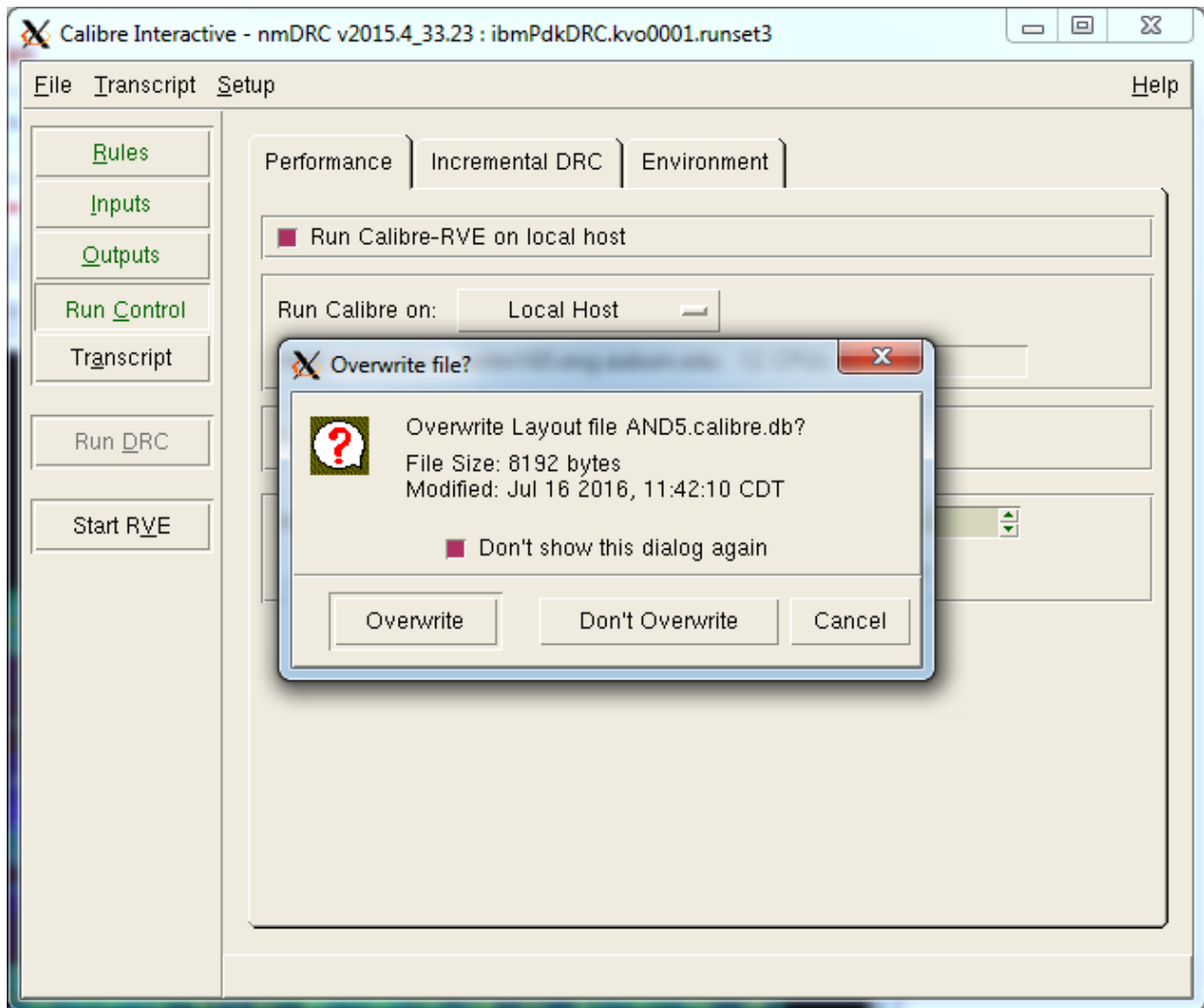


Under the **Outputs** section, you can choose to turn off the **Write DRC summary report file** option, which will cut down on the amount of clutter generated when running DRC. However, the summary report is useful when sharing with other people via email.

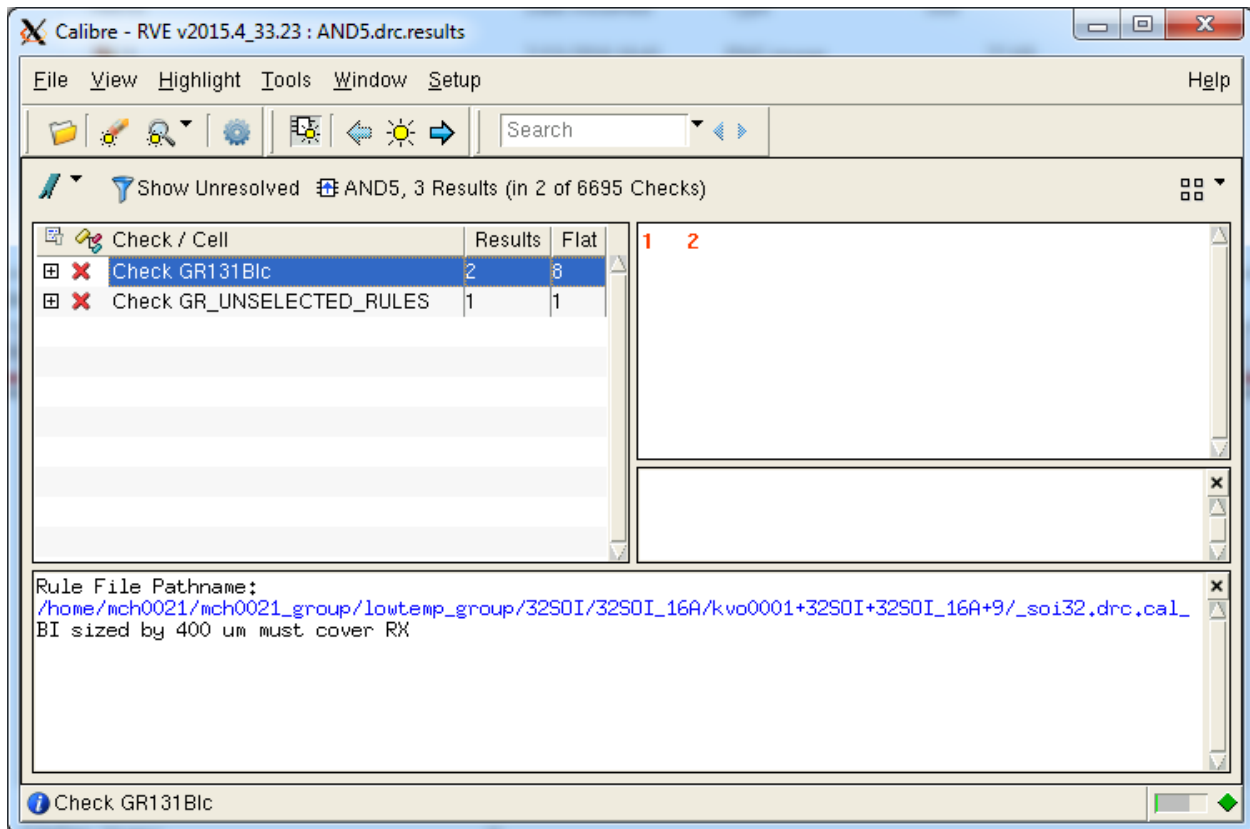




Under the **Run Control** section, set the **Run Calibre** options to **Multi-Threaded** and **Hyperscale**, which will vastly speed up run times.



You can prevent the `Overwrite file?` dialog from being generated a second time by marking the appropriate box. Click `Overwrite` so that the latest layout file is used during DRC.



Depending on the technology, a “DRC-clean” design may contain some errors still. This is because some of the rules can be waived without causing any penalty. For example, some rules pertaining to density can be ignored if the manufacturer has an automated fill procedure. In the case of the above results, the design is considered clean as it is a cell-level layout and does not need to meet the first rule at this level of the hierarchy.

Some common DRC errors include:

- Metal shape too close to metal shape
- Metal shape not overhanging via far enough
- Too little metal per via
- Metal shape not wide enough
- Metal shape not large enough (i.e., too small of area)

- Metal shape forms too small of a notch
- Metal shape is off-grid (e.g., a circle does not lie on-grid)

In many cases, one error in a layout may lead to multiple DRC errors. For instance, a metal shape that is too small may generate both minimum area and minimum width violations. It is best to consult the technology's design manual which contains not only every design rule, but also helpful diagrams that indicate what the rule is talking about. Though the rules are generally just boolean logic, the rules can be cryptic at times, especially depending on the technology.

## F.1 cell\_noden.runset Contents

```
1 *drcRulesFile: /home/mch0021/mch0021_pdk/global_foundries/cmos14lpp/
  ↳ REL_GF/CalibreDRC/DRC/cmos14lpp_xl.drc.cal
2 *drcRunDir: ./calibre_run
3 *drcLayoutPaths: test.calibre.db
4 *drcLayoutPrimary: test
5 *drcLayoutLibrary: test
6 *drcLayoutView: layout
7 *drcLayoutGetFromViewer: 1
8 *drcResultsFile: test.drc.results
9 *drcEnvVars: { BATCH NO Runset } { TECHDIR /home/mch0021/mch0021_pdk/
  ↳ global_foundries/cmos14lpp/REL_GF/CalibreDRC Runset } {
  ↳ LAYOUT_SYSTEM GDSII Runset } { LAYOUT_OUT GDS Runset } { DP_OFF_ALL
  ↳ NO Runset } { RDB_DIR ./RDB_RESULTS Runset } { DENSITY_RESULTS
  ↳ density.results Runset } { ANTENNA_RESULTS antenna.results Runset }
  ↳ { BEOL_STACK 13M_3Mx_2Cx_4Kx_2Hx_2Gx_LB Runset } { DESIGN_TYPE
  ↳ CELL_NODEN Runset } { IOTYPE 3ON6 Runset } { OUTLINE_CHECK YES
  ↳ Runset } { FCPBGA NO Runset } { C_ORIENTATION VERTICAL Runset } {
  ↳ P_DENSITY_CHECK NO Runset } { CHIP_DIE_COUNT OVER_5CHIP_SHOT Runset
  ↳ } { CELL_BOUNDARY DB_EXTENT Runset } { ERROR_LIMITATION DEFAULT
  ↳ Runset } { RECOMMENDED_RULES NO Runset } { OCD_OVL_RECOMMENDED_RULES
  ↳ NO Runset } { CELL_FINE_SS NO Runset } { CUSTOM_CELL_SS NO Runset }
  ↳ { CELL_SS 10 Runset } { DO_SRULES YES Runset } { REMOVE_ELUP_BEVEL4
  ↳ NO Runset } { MOB_OPTION NWMOB Runset } { BOOLEAN_DEBUG NO Runset }
  ↳ { STB_DEBUG NO Runset } { DP_CHECK_DESIGN_M1 YES Runset } {
  ↳ DP_CHECK_DESIGN_M2 YES Runset } { DP_CHECK_DESIGN_M3 YES Runset } {
  ↳ CA_COLORED NO Runset } { CB_COLORED NO Runset } { DP_LAYOUT_OUT OA
  ↳ Runset } { DP_GENERATION_TOOL_M1 YES Runset } {
  ↳ DP_GENERATION_TOOL_M2 YES Runset } { DP_GENERATION_TOOL_M3 YES
  ↳ Runset } { DP_AUTO_STITCH_CA YES Runset } { DENSITY_FLATTEN_M NO
  ↳ Runset }
10 *drcWriteSummary: 0
11 *drcSummaryFile: test.drc.summary
12 *drcIncrDRCSlaveHosts: {use {}} {hostName {}} {cpuCount {}} {a32a64 {}}
  ↳ {rsh {}} {maxMem {}} {workingDir {}} {layerDir {}} {mgcLibPath {}}
  ↳ {launchName {}}
13 *drcIncrDRCLSFSslaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture
  ↳ {}} {minMemory {}} {resourceOptions {}} {submitOptions {}}
14 *drcIncrDRCGGridSlaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture
  ↳ {}} {minMemory {}} {resourceOptions {}} {submitOptions {}}
15 *cmnWarnLayoutOverwrite: 0
16 *cmnRunMT: 1
17 *cmnRunHyper: 1
```

```

18 *cmnSlaveHosts: {use {}} {hostName {}} {cpuCount {}} {a32a64 {}} {rsh
   → {}} {maxMem {}} {workingDir {}} {layerDir {}} {mgcLibPath {}}
   → {launchName {}}
19 *cmnLSFSlaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture {}}
   → {minMemory {}} {resourceOptions {}} {submitOptions {}}
20 *cmnGridSlaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture {}}
   → {minMemory {}} {resourceOptions {}} {submitOptions {}}
21 *cmnFDILayoutLibrary: test
22 *cmnFDILayoutView: layout
23 *cmnFDIDEFLayoutPath: test.def
24 *cmnPromptSaveRunset: 0
25 *cmnSaveRunsetChanges: 0

```

Like the LVS tutorial, the process of setting up Calibre for running DRC can be minimized by using a `runset` file. Above is a very complete file from a 14 nm technology. Consult the description of the `lvs.runset` file for the rest of the lines. Only the differences will be discussed here.

Line 9 is perhaps the most different between `runset` files. This very long line sets up a large number of environment variables for the DRC rule deck. These are all technology-specific, but the syntax is the same. The only notable options here are the `BEOL_STACK`, which must be set to the proper stack option as used in the layout, as well as the `DESIGN_TYPE` option. Both of these options are typical of DRC rule decks across vendors and technologies. The `DESIGN_TYPE` option here indicates whether or not full chip-level checking is to be run or not, which would include checking such shapes as the chip edge, I/O pads, etc. The option also enables or disables checking density, which as previously discussed, may be safely ignored if the manufacturer takes care of density fill, or if you are running cell-level tests and do not wish to consider density.

## F.2 chip\_den.runset Partial Contents

```
9 *drcEnvVars: { BATCH NO Runset } { TECHDIR /home/mch0021/mch0021_pdk/
  ↪ global_foundries/cmos14lpp/REL_GF/CalibreDRC Runset } {
  ↪ LAYOUT_SYSTEM GDSII Runset } { LAYOUT_OUT GDS Runset } { DP_OFF_ALL
  ↪ NO Runset } { RDB_DIR ./RDB_RESULTS Runset } { DENSITY_RESULTS
  ↪ density.results Runset } { ANTENNA_RESULTS antenna.results Runset }
  ↪ { BEOL_STACK 13M_3Mx_2Cx_4Kx_2Hx_2Gx_LB Runset } { DESIGN_TYPE CHIP
  ↪ Runset } { IOTYPE 3ON6 Runset } { OUTLINE_CHECK YES Runset } {
  ↪ FCPBGA NO Runset } { C_ORIENTATION VERTICAL Runset } {
  ↪ P_DENSITY_CHECK NO Runset } { CHIP_DIE_COUNT OVER_5CHIP_SHOT Runset
  ↪ } { CELL_BOUNDARY DB_EXTENT Runset } { ERROR_LIMITATION DEFAULT
  ↪ Runset } { RECOMMENDED_RULES NO Runset } { OCD_OVL_RECOMMENDED_RULES
  ↪ NO Runset } { CELL_FINE_SS NO Runset } { CUSTOM_CELL_SS NO Runset }
  ↪ { CELL_SS 10 Runset } { DO_SRULES YES Runset } { REMOVE_ELUP_BEVEL4
  ↪ NO Runset } { MOB_OPTION NWMOB Runset } { BOOLEAN_DEBUG NO Runset }
  ↪ { STB_DEBUG NO Runset } { DP_CHECK_DESIGN_M1 YES Runset } {
  ↪ DP_CHECK_DESIGN_M2 YES Runset } { DP_CHECK_DESIGN_M3 YES Runset } {
  ↪ CA_COLORED NO Runset } { CB_COLORED NO Runset } { DP_LAYOUT_OUT GDS
  ↪ Runset } { DP_GENERATION_TOOL_M1 YES Runset } {
  ↪ DP_GENERATION_TOOL_M2 YES Runset } { DP_GENERATION_TOOL_M3 YES
  ↪ Runset } { DP_AUTO_STITCH_CA YES Runset } { DENSITY_FLATTEN_M NO
  ↪ Runset }
```

Line 9 is the only line that differs between `runset` files from the same technology to allow checking of the entire chip with density checks. `DESIGN_TYPE` is the only variable changed.

The `runset` file presented here is the most complete DRC `runset` in terms of options and should be emulated for other `runset` files in other technologies.

## Appendix G

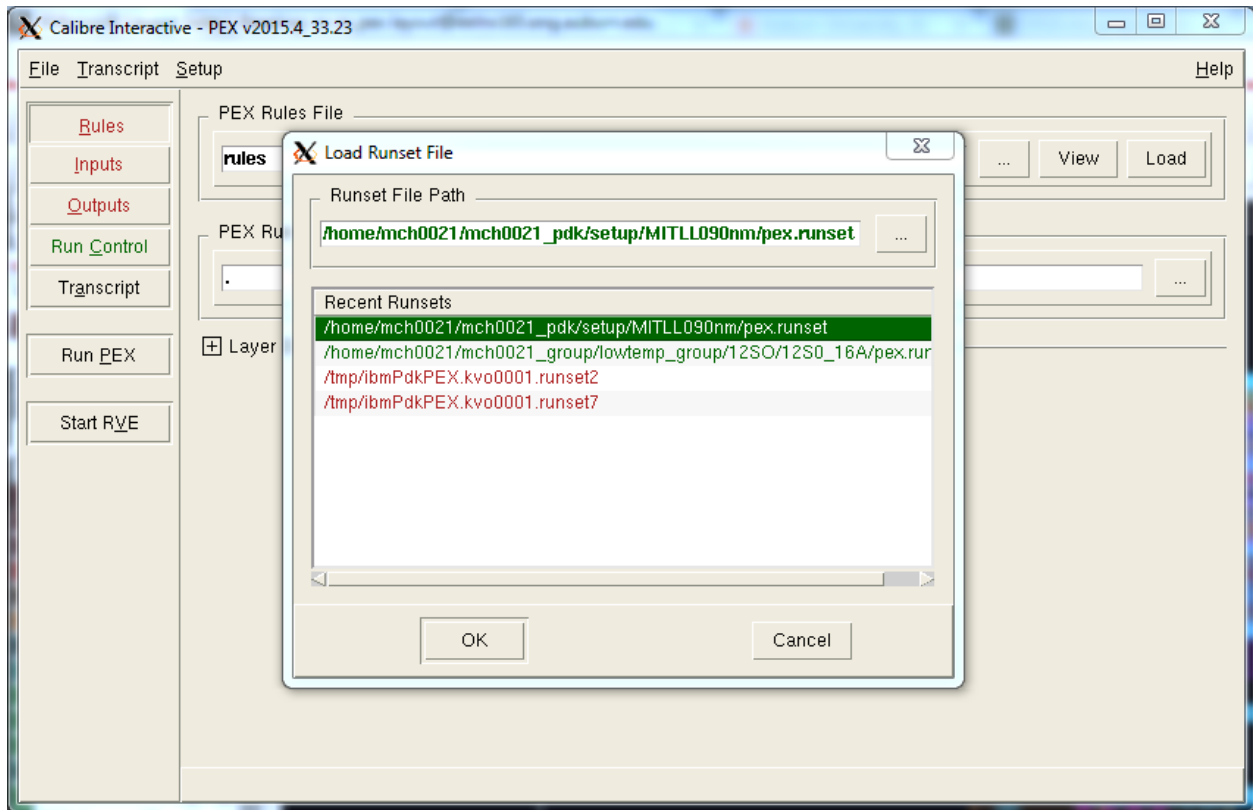
### PEX Tutorial

Parasitic extraction, or PEX for short, is a powerful tool for extracting resistive, capacitive, and inductive parasitic components from a layout. This is very useful to a designer who wants to ensure that the layout will function as well as the schematic has shown it to through simulation. Parasitics may slow the circuit down enough to the point of failing at the desired frequency, for instance. PEX requires both an input layout as well as a schematic, as the first step PEX performs is LVS, which is explained in detail in the LVS tutorial. PEX can output a netlist of parasitics combined with the original schematic, as well as a summary of parasitics. The netlist can also be turned into a separate schematic view called a `calibre` view, named after the Calibre PEX tool that extracts the parasitics. This view can be simulated using ADE, which is covered in the ADE tutorial.

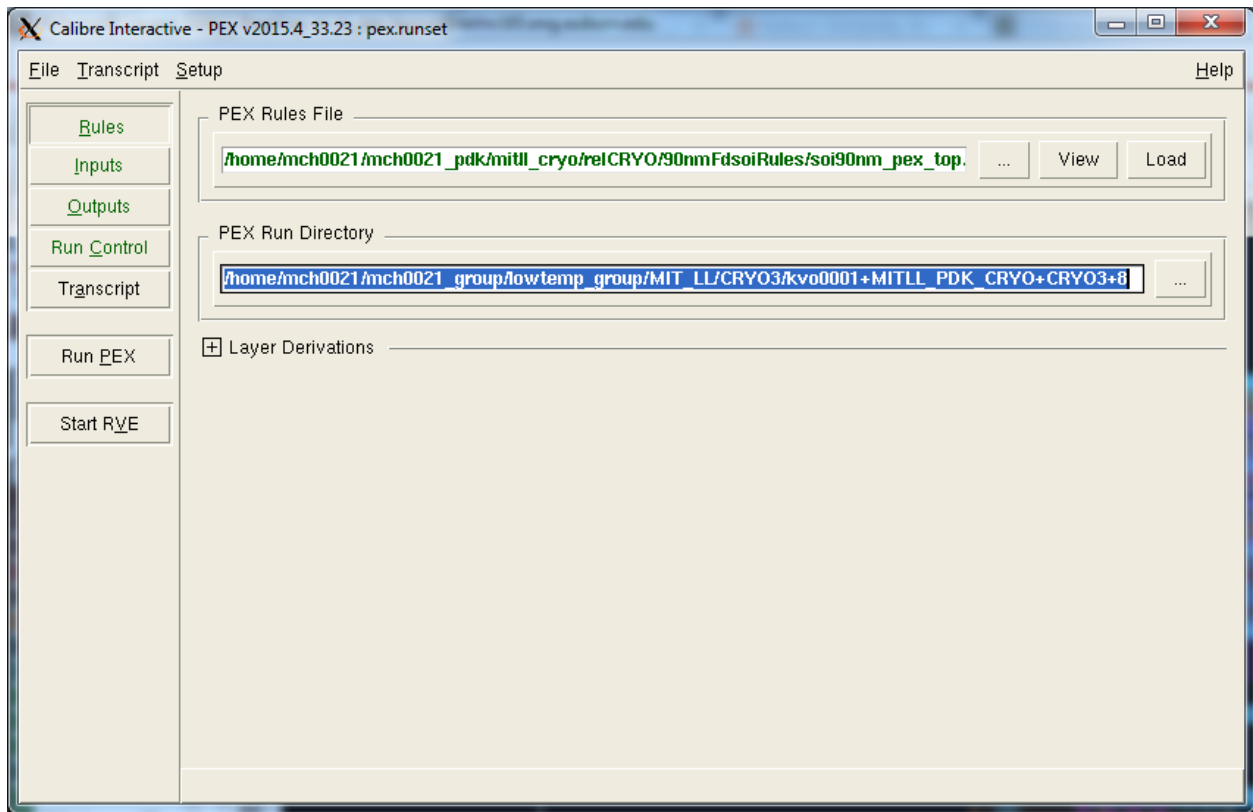
This tutorial covers PEX for both a 90 nm technology and a 32 nm technology, which differ slightly in setup. If you have not run LVS, it is advised to run that first, as PEX will not successfully complete if LVS does not pass.



## G.1 90 nm PEX Setup



To start Calibre PEX, navigate to the **Calibre** menu in the **Virtuoso Layout** window and click on **PEX**. By default, Calibre will ask for a runset file. Load the file from `home ▶ mch0021 ▶ mch0021_pdk ▶ setup ▶ [technology]`.



The `runset` file will pre-load many of the correct options, but you will need to ensure the `PEX Run Directory` is set to a directory that you own. Otherwise, you are clear to click `Run PEX`. Since PEX is dependent on LVS to run, you may confirm that the schematic information is correct under `Inputs` before proceeding.

Calibre View Setup@eelnx165.eng.auburn.edu

CalibreView Setup File:

---

CalibreView Netlist File:

Output Library:

Schematic Library:

---

Cellmap File:

Log File:

---

Calibre View Name:

Calibre View Type:  maskLayout  schematic

Create Terminals:  if matching terminal exists on symbol  Create all terminals

Preserve Device Case

Execute Callbacks

Suppress Notes

Reset Properties:

---

Magnify Instances By:

---

Device Placement:  Layout Location  Arrayed

Parasitic Placement:  Layout Location  Arrayed

Show Parasitic Polygons

---

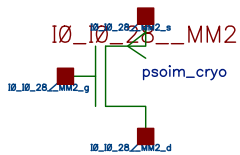
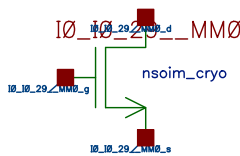
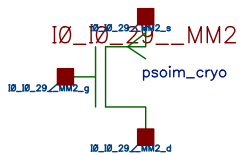
Open Calibre CellView:  Read-mode  Edit-mode  Don't Open

Generate SPECTRE Netlist

---

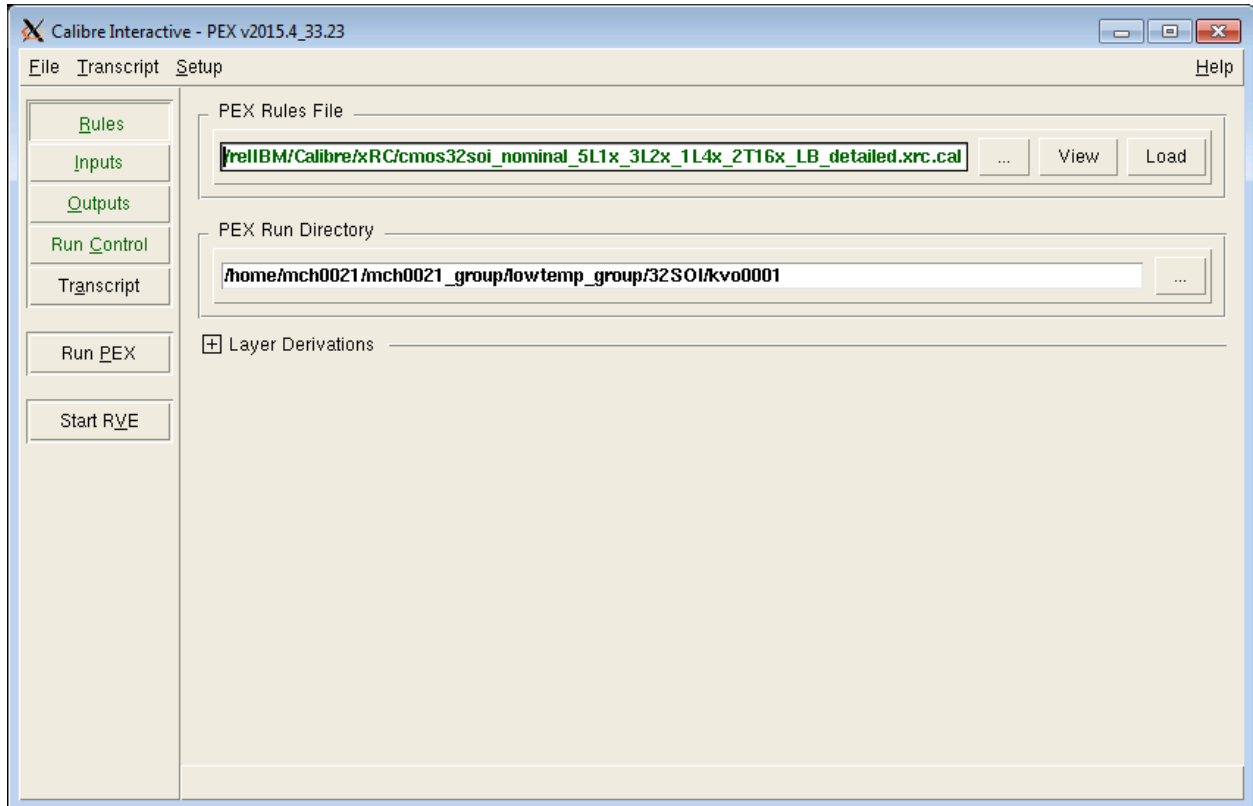
Always Show Dialog

With many PEX runs, the desired output is a `calibre` view. The `Calibre View Setup` window will pop up as soon as PEX has completed. Be sure to use the `Cellmap File` as provided by the vendor of the technology. This is crucial to map the extracted components from the layout back to the schematic view. Change the `Calibre View Type` to `schematic`, and be sure to select `Create all terminals`. You will want to select `Preserve Device Case` as well, as there may be mapping problems if this is not selected. The `Device Placement` option can be set to `Arrayed`, which will force all of the ports at the top, followed by transistors, followed by the dispersed resistive and capacitive parasitics. You will likely want to open the `calibre` view for reading after, so set the `Open Calibre CellView` option to `Read-mode`, then click `OK`. Depending on the number of parasitics, it may take a few seconds to many hours to complete. Due to the nature of simulation, if it takes many hours to complete, the simulation may fail to run altogether due to memory constraints. If this is the case, follow along in the 32 nm tutorial to learn how to reduce the number of parasitics.

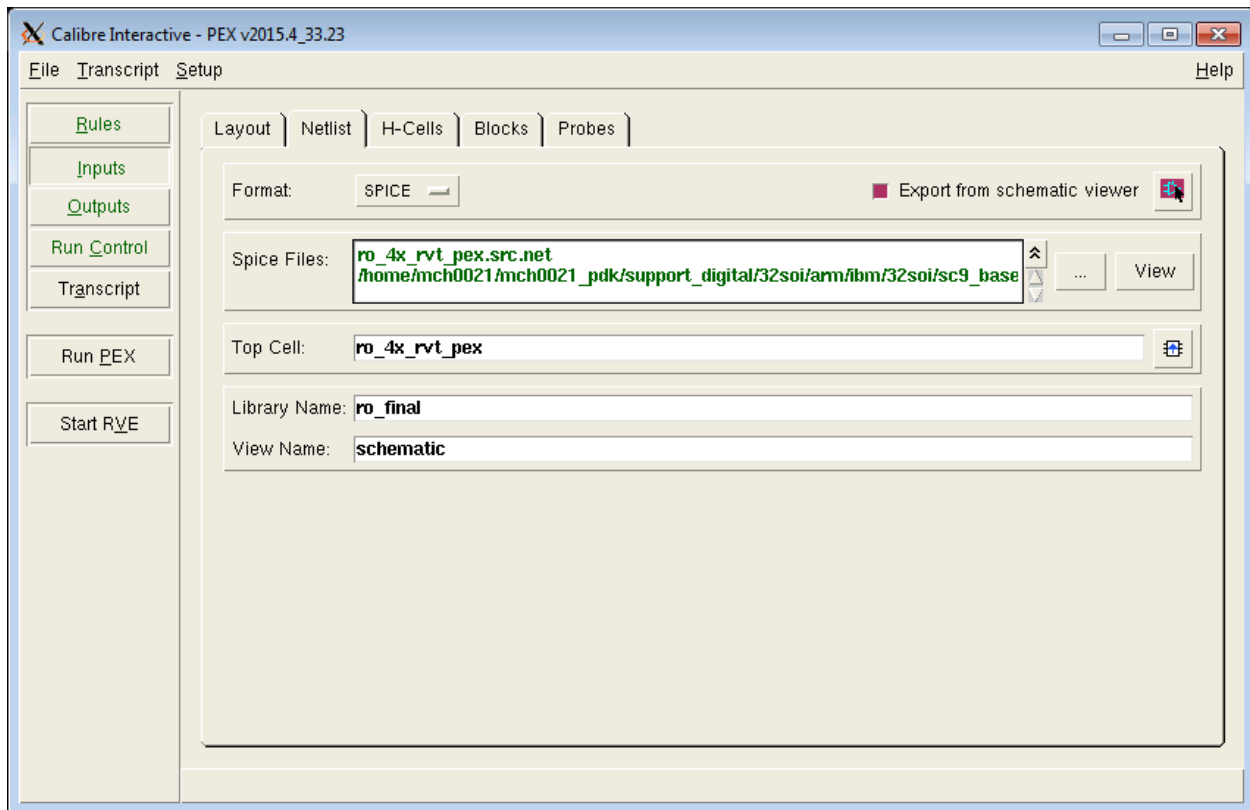


The extracted schematic is not very readable. As mentioned before, the ports are at the top, with the devices underneath. The transistors are labeled by their hierarchical instance names from the initial schematic view, which makes identification possible, but not easy. This can be useful if you wish to examine other nets during simulation that are not brought out to a port.

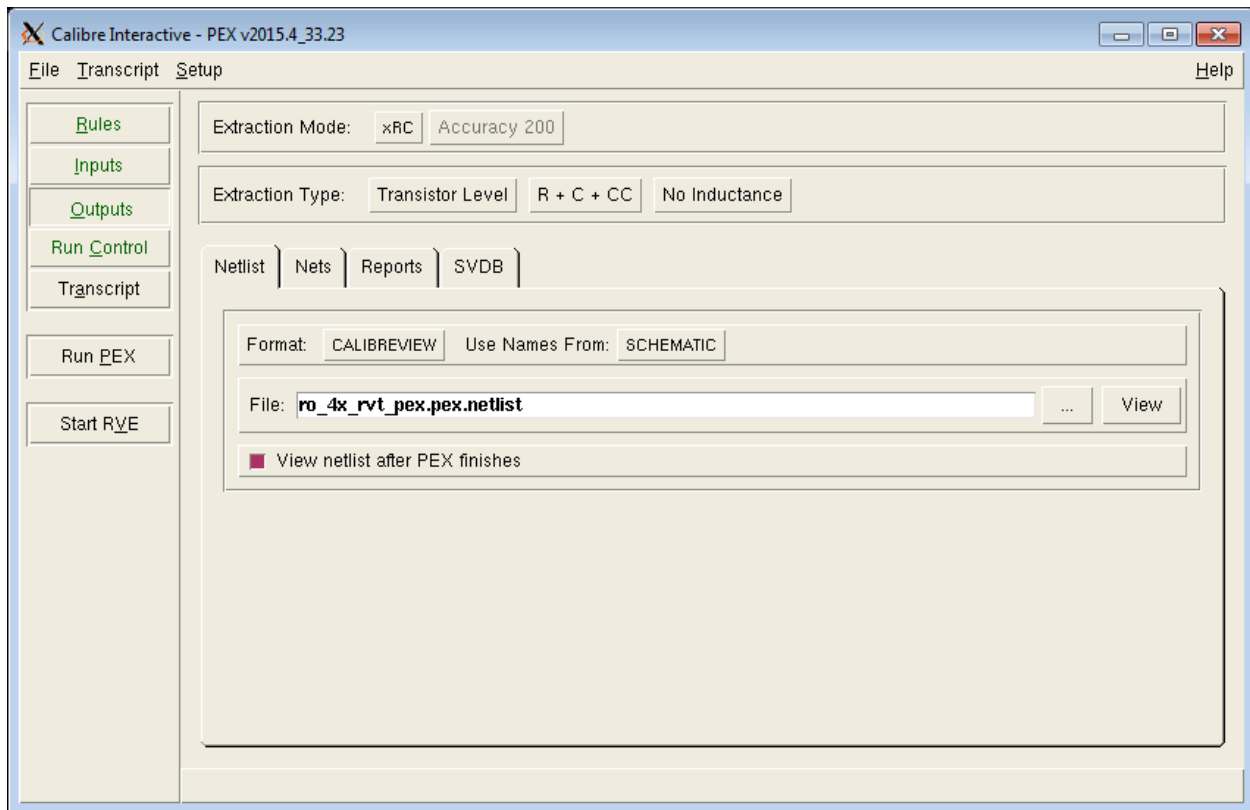
## G.2 32 nm PEX Setup



Running PEX in other vendors' kits can be slightly different. For those vendors that provide a custom Calibre menu, you may be disappointed to see that there is not a PEX option. However, the environment variables that are used when running DRC and LVS are still very pertinent to running PEX. Thus, running LVS immediately prior to running PEX is a good idea, as this will preset the environment variables to their correct values. Once this is done, you can start PEX from the `Calibre` menu and load the correct rules file from `home ▶ mch0021 ▶ mch0021_pdk ▶ [path to technology] ▶ Calibre ▶ xRC`. There may be several rules files to choose from, so be sure to choose the one that corresponds to the correct BEOL stack.

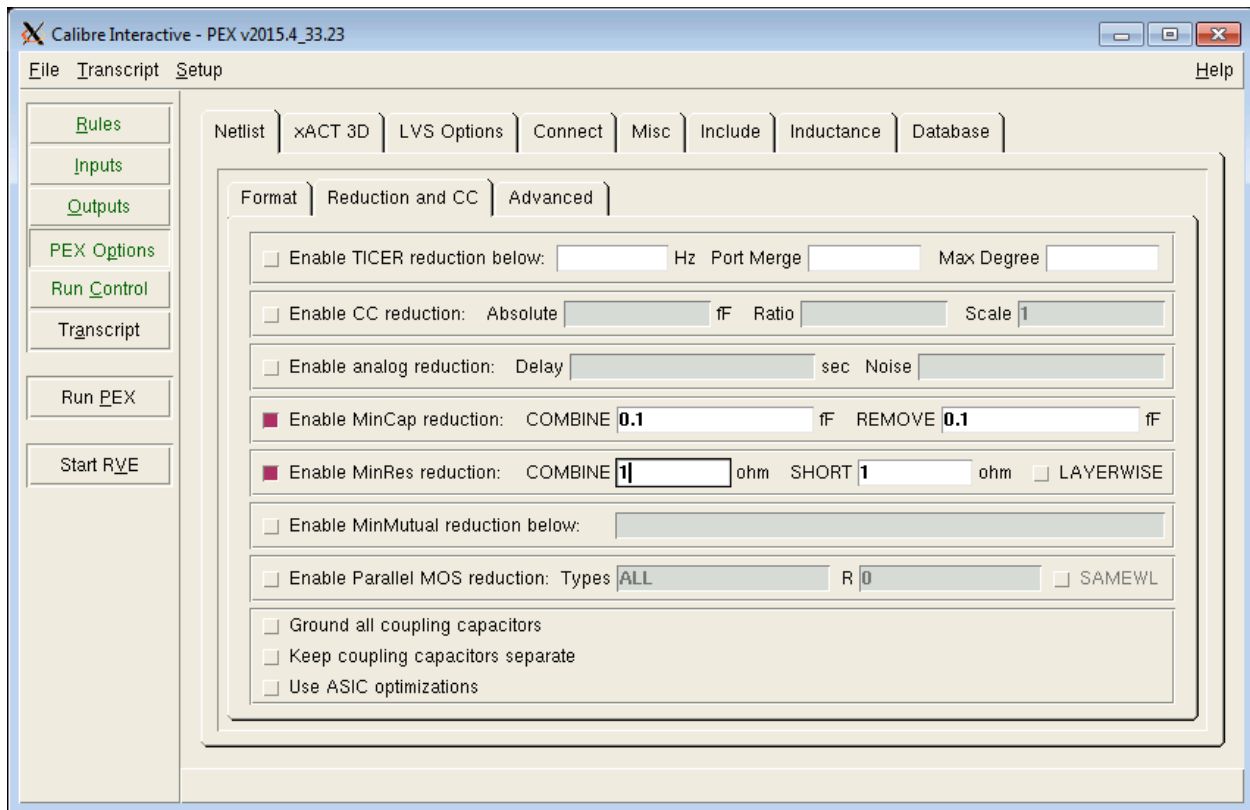


If you are using a standard cell library that does not have schematic views, as is often the case, be sure to include the CDL netlist. Refer to the LVS tutorial for details on how to do this.

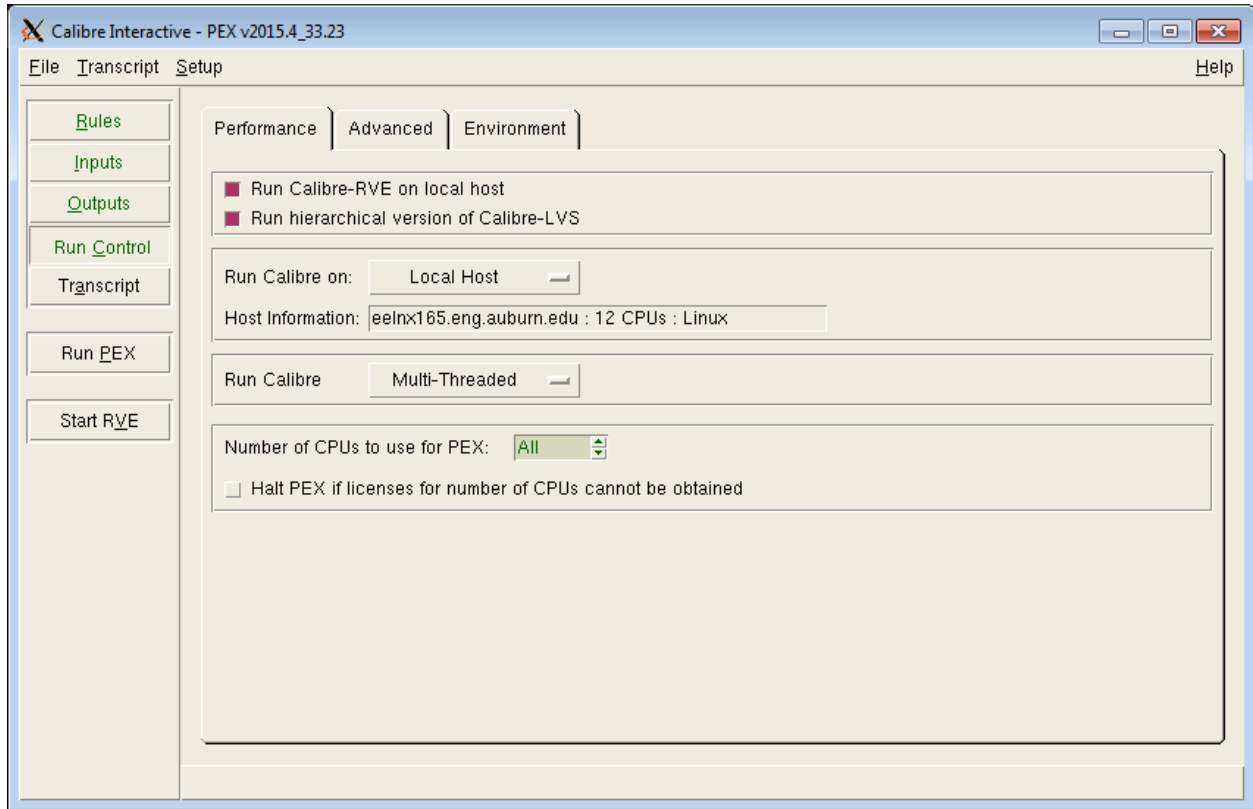


Under **Outputs**, select the **CALIBREVIEW** format. This will allow the creation of the aforementioned calibre view. You may also want to change the extraction type to just extract resistances and lumped capacitances, but no coupling capacitances, for instance. This is to reduce the number of parasitic elements, which otherwise can be too large for simulation. By reducing the number of elements to extract, the simulation may be less accurate, but will simulate much faster. You can also choose to extract inductances as well.

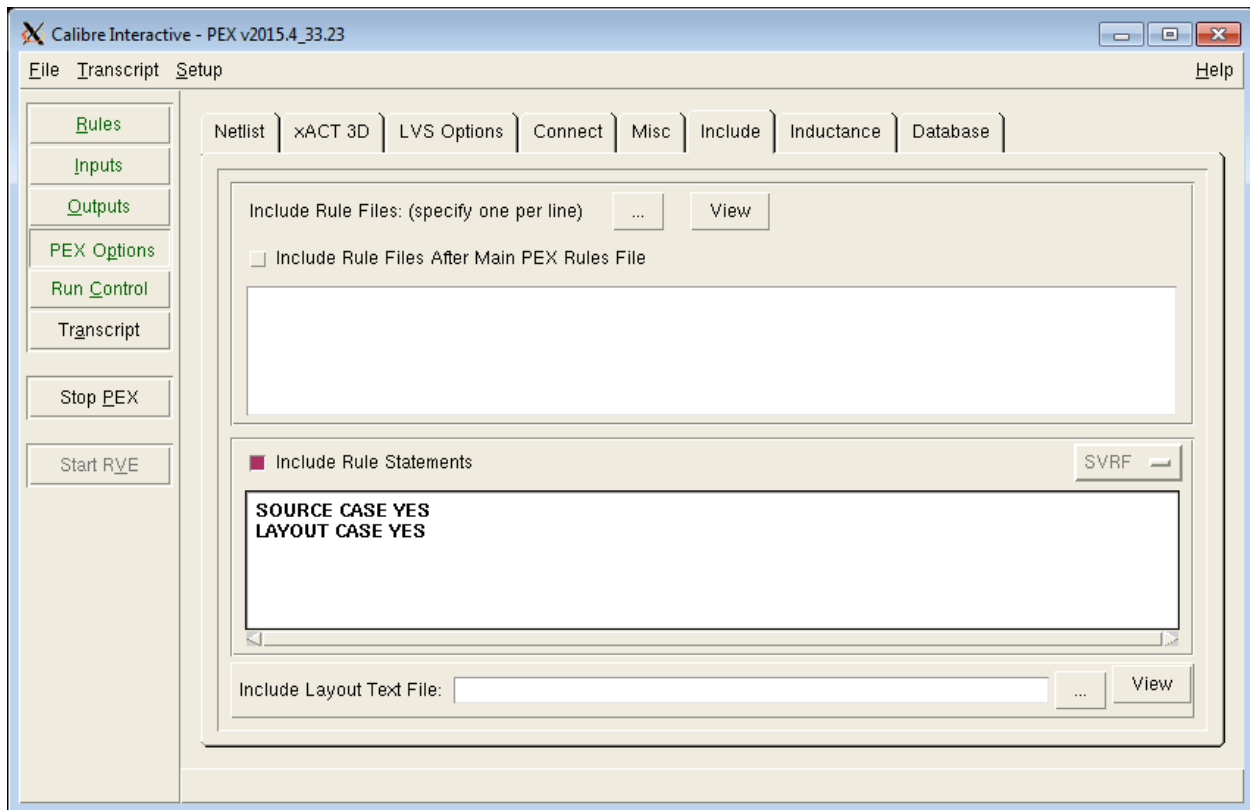




If, when running PEX, you find that the number of parasitics generated is far too high, you can enable `MinCap reduction` and `MinRes reduction`. This will remove or combine capacitances or resistances based on their value. You can minimize the number of parasitics by eliminating those that will not impact simulation much. Over 100,000 instances is probably too many for a reasonable simulation time. If increasing the reduction parameters is not possible, consider breaking the layout into smaller chunks and run PEX separately. For instance, PEX on the entire chip may not be possible, but PEX on the important pieces of the core, plus PEX run separately on the individual pads, might be quite feasible.



In order to cut down on the time spent extracting, be sure to enable the **Multi-Threaded** option.



If you are seeing errors related to instances not found in the map file, there may be an issue with the capitalization of such instance names. To remedy this, you will need to force Calibre to be case sensitive. Click on **Setup** >> **PEX Options** to get the **PEX Options** button. Then, under the **Include** tab, click on the **Include Rule Statements** and type **SOURCE CASE YES** and **LAYOUT CASE YES** to keep the capitalization the same in both the schematic netlist and the layout netlist. Some rules files may include those lines by default, but do not assume that it is the case until you have checked.

### G.3 pex.runset Contents

```
1 *pexRulesFile: /home/mch0021/mch0021_pdk/mitll_cryo/relCRY0/
  ↳ 90nmFdsoiRules/soi90nm_pex_top.rules
2 *pexRulesFileLastLoad: 1467823017
3 *pexRunDir: /home/mch0021/mch0021_group/lowtemp_group/MIT_LL/CRY03/
  ↳ kvo0001+MITLL_PDK_CRY0+CRY03+8
4 *pexLayoutPaths: test.calibre.db
5 *pexLayoutPrimary: test
6 *pexLayoutLibrary: dram
7 *pexLayoutView: layout
8 *pexLayoutGetFromViewer: 1
9 *pexSourcePath: test.src.net
10 *pexSourcePrimary: test
11 *pexSourceLibrary: dram
12 *pexSourceView: schematic
13 *pexSourceGetFromViewer: 1
14 *pexEnvVars: {USE_MTK {} Unset} {TOPMETAL M5 Runset} {PEX_PROCESS
  ↳ CRY0_4K Runset}
15 *pexReportFile: test.lvs.report
16 *pexPexNetlistFile: test.pex.netlist
17 *pexPexNetlistFormat: CALIBREVIEW
18 *pexPexReportFile: test.pex.report
19 *pexStartRVE: 1
20 *cmnWarnLayoutOverwrite: 0
21 *cmnWarnSourceOverwrite: 0
22 *cmnResolution: 5
23 *cmnUseCBforRVE: 0
24 *cmnRunHier: 2
25 *cmnRunMT: 1
26 *cmnSlaveHosts: {use {}} {hostName {}} {cpuCount {}} {a32a64 {}} {rsh
  ↳ {}} {maxMem {}} {workingDir {}} {layerDir {}} {mgcLibPath {}}
  ↳ {launchName {}}
27 *cmnLSFSlaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture {}}
  ↳ {minMemory {}} {resourceOptions {}} {submitOptions {}}
28 *cmnGridSlaveTbl: {use 1} {totalCpus 1} {minCpus 1} {architecture {}}
  ↳ {minMemory {}} {resourceOptions {}} {submitOptions {}}
29 *cmnFDILayoutLibrary: dram
30 *cmnFDILayoutView: layout
```

```

31 *cmnFDIDEFLayoutPath: test.def
32 *cmnTraceProperties: {1 { MP "W" "W" 0 0}} {1 { MP "L" "L" 0 0}} {1 { MN
  → "W" "W" 0 0}} {1 { MN "L" "L" 0 0}} {1 { MP5T "W" "W" 0 0}} {1 {
  → MP5T "L" "L" 0 0}} {1 { MN5T "W" "W" 0 0}} {1 { MN5T "L" "L" 0 0}}
  → {1 { D "A" "A" 0 0}} {1 { R "w" "w" 0 0}} {1 { R "l" "l" 0 0}} {1 {
  → C(vncap) "w" "w" 0 0}} {1 { C(vncap) "l" "l" 0 0}} {1 { C(vncap)
  → "fw" "fw" 0 0}} {1 { C(vncap) "fs" "fs" 0 0}} {1 { C(vncap) "nf"
  → "nf" 0 0}} {1 { C(vncap) "toplev" "toplev" 0 0}} {1 { C(vncap)
  → "botlev" "botlev" 0 0}}

```

Here is a sample `runset` file from the 90nm technology. After examining the LVS `runset` and the DRC `runset` files, this should look very familiar. Improvements could be made to prevent the `runset` from asking if you wish to save it upon closing Calibre, as well as changing the run directory to a local directory within your workspace, or perhaps the workspace itself.

Appendix H  
Encounter Tutorial

## H.1 Behavioral Verilog

### H.1.1 mac32\_dual\_wrapper.v Contents

```
1  //mac32_dual_wrapper.v
2  //Kyle Owen
3  //21 September 2014
4  //Wraps two serial-wrapped MACs (one LVT, one RVT) into a single
   ↪ instance
5  //with a chip select line to pick the active MAC.
6
7  module mac32_dual_wrapper(
8
9             phi1,
10            phi2,
11            global_rstn,
12            mac_wait,
13            mac_wait_test,
14            serial_in,
15            serial_in_enable,
16            serial_clk_phi1,
17            serial_clk_phi2,
18            serial_out,
19            serial_out_enable,
20            chip_select
21 );
22
23 input phi1;
24 input phi2;
25 input global_rstn;
26 input mac_wait;
27 input mac_wait_test;
28 input serial_in;
29 input serial_in_enable;
30 input serial_out_enable;
31 input chip_select;
```

```

31 output serial_clk_phi1;
32 output serial_clk_phi2;
33 output serial_out;
34
35 wire rstn_hvt, rstn_rvt;
36 wire serial_out_hvt, serial_out_rvt;
37 wire serial_clk_phi1_hvt, serial_clk_phi1_rvt;
38 wire serial_clk_phi2_hvt, serial_clk_phi2_rvt;
39
40 assign serial_out = (chip_select) ? serial_out_rvt : serial_out_hvt;
   ↪ //select LVT when chip_select = 0; otherwise RVT
41 assign serial_clk_phi1 = (chip_select) ? serial_clk_phi1_rvt :
   ↪ serial_clk_phi1_hvt;
42 assign serial_clk_phi2 = (chip_select) ? serial_clk_phi2_rvt :
   ↪ serial_clk_phi2_hvt;
43 assign rstn_rvt = global_rstn && chip_select; //reset RVT when
   ↪ chip_select = 0
44 assign rstn_hvt = global_rstn && ~chip_select; //reset LVT when
   ↪ chip_select = 1
45
46 mac32_serial_wrapper_hvt hvt_mac(
47     .phi1(phi1),
48     .phi2(phi2),
49     .rstn(rstn_hvt),
50     .mac_wait(mac_wait),
51     .mac_wait_test(mac_wait_test),
52     .serial_in(serial_in),
53     .serial_in_enable(serial_in_enable),
54     .serial_clk_phi1(serial_clk_phi1_hvt),
55     .serial_clk_phi2(serial_clk_phi2_hvt),
56     .serial_out(serial_out_hvt),
57     .serial_out_enable(serial_out_enable)
58 );
59
60 mac32_serial_wrapper_rvt rvt_mac(
61     .phi1(phi1),
62     .phi2(phi2),
63     .rstn(rstn_rvt),
64     .mac_wait(mac_wait),
65     .mac_wait_test(mac_wait_test),
66     .serial_in(serial_in),
67     .serial_in_enable(serial_in_enable),
68     .serial_clk_phi1(serial_clk_phi1_rvt),
69     .serial_clk_phi2(serial_clk_phi2_rvt),
70     .serial_out(serial_out_rvt),

```

```

71         .serial_out_enable(serial_out_enable)
72     );
73
74     endmodule

```

## H.2 Tcl Scripts

### H.2.1 rtl\_rvt.tcl Contents

```

1  set_attribute lib_search_path /home/mch0021/mch0021_pdk/_
   ↪ support_digital/32soi/arm/ibm/32soi/sc9_base_rvt/r2p0/lib
2
3  set_attribute library
   ↪ {sc9_32soi_base_rvt_tt_nominal_max_0p90v_50c_mxs.lib}
4
5  read_hdl ../hdl/mac32_serial_wrapper_rvt.v
6  read_hdl ../hdl/mac32_two_phase_pipelined_rvt.v
7
8  elaborate mac32_serial_wrapper_rvt
9
10 set_attribute max_fanout 8 designs/*
11
12 set_attribute avoid true [find / -libcell *]
13 set_attribute avoid false {AND2_X1M_A9TR BUFH_X1M_A9TR DFFQ_X1M_A9TR
   ↪ LATNQ_X1M_A9TR INV_X1M_A9TR INV_X2M_A9TR INV_X4M_A9TR INV_X7P5M_A9TR
   ↪ INV_X16M_A9TR MXT2_X1M_A9TR NAND2_X1M_A9TR NOR2_X1M_A9TR
   ↪ OR2_X1M_A9TR XOR2_X1M_A9TR}
14
15 synthesize -to_mapped -effort high
16
17 write_hdl > ../hdl/mac32_serial_wrapper_rvt_synth.v
18
19 report gates

```

### H.2.2 top\_level.tcl Contents

```

1  set_attribute lib_search_path /home/mch0021/mch0021_pdk/_
   ↪ support_digital/32soi/arm/ibm/32soi/sc9_base_rvt/r2p0/lib
2
3  set_attribute library
   ↪ {sc9_32soi_base_rvt_tt_nominal_max_0p90v_50c_mxs.lib}
4

```



```

5 read_hdl ../hdl/mac32_dual_wrapper.v
6
7 elaborate mac32_dual_wrapper
8
9 set_attribute max_fanout 8 designs/*
10
11 set_attribute avoid true [find / -libcell *]
12 set_attribute avoid false {AND2_X1M_A9TR BUFH_X1M_A9TR DFFQ_X1M_A9TR
  ↳ LATNQ_X1M_A9TR INV_X1M_A9TR INV_X2M_A9TR INV_X4M_A9TR INV_X7P5M_A9TR
  ↳ INV_X16M_A9TR MXT2_X1M_A9TR NAND2_X1M_A9TR NOR2_X1M_A9TR
  ↳ OR2_X1M_A9TR XOR2_X1M_A9TR}
13
14 synthesize -to_mapped -effort high
15
16 write_hdl > ../hdl/mac32_dual_wrapper_synth.v
17
18 report gates

```

### H.2.3 pr.tcl Contents

```

1 set basename top_level
2 set structure_name mac32_dual_wrapper
3 set init_gnd_net VSS
4 set init_pwr_net VDD
5 set init_design_uniquify 1
6 set init_lef_file {/home/mch0021/mch0021_pdk/support_digital/32soi/
  ↳ 5L1x_3L2x_1L4x_2T16x_LB/sc9_tech.lef
  ↳ /home/mch0021/mch0021_pdk/support_digital/32soi/arm/ibm/32soi/
  ↳ sc9_base_hvt/r3p0/lef/sc9_32soi_base_hvt.lef
  ↳ /home/mch0021/mch0021_pdk/support_digital/32soi/arm/ibm/32soi/
  ↳ sc9_base_rvt/r2p0/lef/sc9_32soi_base_rvt.lef}
7 set init_design_settop 0
8 set init_verilog ../hdl/${basename}.v
9 set pwr_global VDD
10 set pwr_pin VDD
11 set gnd_global VSS
12 set gnd_pin VSS
13 set pwr_nets [list ${pwr_global} ${gnd_global}]
14
15 set fp_width 214.89
16 set fp_height 215.10
17 set fp_bound_x 20.02
18 set fp_bound_y 20.00

```

```

19 set bb_to_edge_x 7.54
20 set bb_to_edge_y 4.20
21 set welltap_width 1.56
22
23 set welltap_interval [expr {$fp_width - $welltap_width}]
24
25 set bb_width [expr {$fp_width - 2 * $bb_to_edge_x}]
26 set bb_height [expr {$fp_height / 2 - 2 * $bb_to_edge_y}]
27
28 set bb1_bl_x [expr {$fp_bound_x + $bb_to_edge_x}]
29 set bb1_bl_y [expr {$fp_bound_y + $bb_to_edge_y}]
30 set bb1_tr_x [expr {$bb1_bl_x + $bb_width}]
31 set bb1_tr_y [expr {$bb1_bl_y + $bb_height}]
32
33 set bb2_bl_x $bb1_bl_x
34 set bb2_bl_y [expr {$bb1_tr_y + 2 * $bb_to_edge_y}]
35 set bb2_tr_x $bb1_tr_x
36 set bb2_tr_y [expr {$bb2_bl_y + $bb_height}]
37
38 init_design
39 clearGlobalNets
40 globalNetConnect ${pwr_global} -type pgin -pin ${pwr_pin} -inst *
41 globalNetConnect ${gnd_global} -type pgin -pin ${gnd_pin} -inst *
42 setNanoRouteMode -drouteUseMultiCutViaEffort high
43
44 ## M1 avoidance
45 setNanoRouteMode -routeWithViaInPin false
46 setNanoRouteMode -routeWithViaOnlyForStandardCellPin false
47 setNanoRouteMode -routeBottomRoutingLayer 1
48
49 puts "#####"
50 puts "##### Specifying Floorplan #####"
51 puts "#####"
52 floorPlan -site SC9_32S0I -s $fp_width $fp_height $fp_bound_x
   → $fp_bound_y $fp_bound_x $fp_bound_y
53
54 puts "#####"
55 puts "##### Specifying Black Box #####"
56 puts "#####"

```

```

57 specifyBlackBox -cell mac32_serial_wrapper_rvt -size $bb_width
   → $bb_height -coreSpacing 0.0 0.0 0.0 0.0 -minPitchLeft 2
   → -minPitchRight 2 -minPitchTop 2 -minPitchBottom 2 -reservedLayer { 1
   → 2 3 4 5 6 7 8 9 10 11 12} -pinLayerTop { 3 5 7 9 11} -pinLayerLeft {
   → 2 4 6 8 10 12} -pinLayerBottom { 3 5 7 9 11} -pinLayerRight { 2 4 6
   → 8 10 12} -routingHalo 0.0 -routingHaloTopLayer 12
   → -routingHaloBottomLayer 1 -placementHalo 0.0 0.0 0.0 0.0
58 specifyBlackBox -cell mac32_serial_wrapper_hvt -size $bb_width
   → $bb_height -coreSpacing 0.0 0.0 0.0 0.0 -minPitchLeft 2
   → -minPitchRight 2 -minPitchTop 2 -minPitchBottom 2 -reservedLayer { 1
   → 2 3 4 5 6 7 8 9 10 11 12} -pinLayerTop { 3 5 7 9 11} -pinLayerLeft {
   → 2 4 6 8 10 12} -pinLayerBottom { 3 5 7 9 11} -pinLayerRight { 2 4 6
   → 8 10 12} -routingHalo 0.0 -routingHaloTopLayer 12
   → -routingHaloBottomLayer 1 -placementHalo 0.0 0.0 0.0 0.0

59
60 puts "#####"
61 puts "##### Adding Power Rings #####"
62 puts "#####"
63 addRing -skip_via_on_wire_shape Noshape -use_wire_group_bits 2
   → -use_interleaving_wire_group 1 -skip_via_on_pin Standardcell -center
   → 1 -stacked_via_top_layer LB -use_wire_group 1 -type core_rings
   → -jog_distance 4.0 -threshold 4.0 -nets {VDD VSS} -follow core
   → -stacked_via_bottom_layer M1 -layer {bottom M1 top M1 right M2 left
   → M2} -width 1.04 -spacing 2 -offset 4.0

64
65 puts "#####"
66 puts "##### Adding Well Taps #####"
67 puts "#####"
68 addWellTap -cell BITIE_A9TR -cellInterval $welldtap_interval -fixedGap
   → -skipRow 1 -prefix WELLTAP

69
70 puts "#####"
71 puts "##### Placing Pins #####"
72 puts "#####"
73 editPin -pinWidth 0.1 -pinDepth 0.22 -fixedPin 1 -fixOverlap 1 -unit
   → MICRON -spreadDirection clockwise -side Top -layer 3 -spreadType
   → center -spacing 10.0 -pin {serial_out serial_clk_phi2
   → serial_clk_phi1 chip_select}
74 editPin -pinWidth 0.1 -pinDepth 0.22 -fixedPin 1 -fixOverlap 1 -unit
   → MICRON -spreadDirection counterclockwise -side Bottom -layer 3
   → -spreadType center -spacing 10.0 -pin {serial_out_enable
   → serial_in_enable serial_in}

```

```

75 editPin -pinWidth 0.1 -pinDepth 0.22 -fixedPin 1 -fixOverlap 1 -unit
   ↳ MICRON -spreadDirection counterclockwise -side Left -layer 3
   ↳ -spreadType center -spacing 10.0 -pin {mac_wait_test mac_wait
   ↳ global_rstn}
76 editPin -pinWidth 0.1 -pinDepth 0.22 -fixedPin 1 -fixOverlap 1 -unit
   ↳ MICRON -spreadDirection clockwise -side Right -layer 3 -spreadType
   ↳ center -spacing 10.0 -pin {phi2 phi1}
77
78 puts "#####"
79 puts "#####          Placing Cells          #####"
80 puts "#####"
81 setPlaceMode -fp false
82
83 placeDesign
84
85 puts "#####"
86 puts "#####          Moving Black Boxes          #####"
87 puts "#####"
88 setObjFPlanBox Instance rvt_mac $bb1_bl_x $bb1_bl_y $bb1_tr_x $bb1_tr_y
89 setObjFPlanBox Instance hvt_mac $bb2_bl_x $bb2_bl_y $bb2_tr_x $bb2_tr_y
90
91 puts "#####"
92 puts "#####          Trial Routing          #####"
93 puts "#####"
94 trialRoute -maxRouteLayer 6 -floorplanMode
95
96 puts "#####"
97 puts "#####          Loading Netlists          #####"
98 puts "#####"
99 loadBlackBoxNetlist ../hdl/mac32_serial_wrapper_rvt_synth.v
100 loadBlackBoxNetlist ../hdl/mac32_serial_wrapper_hvt_synth.v
101
102 puts "#####"
103 puts "#####          Placing Cells          #####"
104 puts "#####"
105 placeDesign
106
107 puts "#####"
108 puts "#####          Moving Black Boxes          #####"
109 puts "#####"
110 setObjFPlanBox Instance rvt_mac $bb1_bl_x $bb1_bl_y $bb1_tr_x $bb1_tr_y
111 setObjFPlanBox Instance hvt_mac $bb2_bl_x $bb2_bl_y $bb2_tr_x $bb2_tr_y
112
113 puts "#####"
114 puts "#####          Converting to Fence          #####"

```

```

115 puts "#####"
116 convertBlackBoxToFence -cell mac32_serial_wrapper_rvt
117 convertBlackBoxToFence -cell mac32_serial_wrapper_hvt
118
119 puts "#####"
120 puts "#####          Placing Cells          #####"
121 puts "#####"
122 placeDesign -inPlaceOpt
123
124 puts "#####"
125 puts "#####          Adding Filler          #####"
126 puts "#####"
127
128 set fillercap_cells [list FILLSGCAP128_A9TH FILLSGCAP64_A9TH
  ↪ FILLSGCAP32_A9TH FILLSGCAP16_A9TH FILLSGCAP8_A9TH FILLSGCAP4_A9TH]
129 set filler_cells [list FILL128_A9TH FILL64_A9TH FILL32_A9TH FILL16_A9TH
  ↪ FILL8_A9TH FILL4_A9TH FILL2_A9TH FILL1_A9TH]
130
131 foreach fill_cell $filler_cells fillcap_cell $fillercap_cells {
132     set fill_cells [list $fill_cell $fillcap_cell]
133     addFiller -cell ${fill_cells} -prefix FILLER
134 }
135
136 puts "#####"
137 puts "#####          Special Route          #####"
138 puts "#####"
139 sroute
140 puts "#####"
141 puts "#####          Detail Route          #####"
142 puts "#####"
143 globalDetailRoute
144 puts "#####"
145 puts "#####          Verifying Geometry          #####"
146 puts "#####"
147 verifyGeometry
148
149 puts "#####"
150 puts "#####          Write Post-route Verilog          #####"
151 puts "#####"
152
153 saveNetlist ../hdl/postroute/${basename}.v
154
155 puts "#####"
156 puts "#####          Write SDF File          #####"

```

```
157 puts "#####"
158
159 write_sdf ../sdf/${basename}.sdf
160
161 puts "#####"
162 puts "##### Streaming Out GDSII #####"
163 puts "#####"
164 streamOut ../gds/${basename}.gds -mapFile ../tech.map -libName DesignLib
    ↪ -structureName ${structure_name} -stripes 1 -units 2000 -mode ALL
165 #exit
```