12-2010

Multi-threshold CMOS Circuit Design Methodology from 2D to 3D

Ross Josiah Thian
University of Arkansas, Fayetteville

MULTI-THRESHOLD CMOS CIRCUIT DESIGN

METHODOLOGY FROM 2D TO 3D

MULTI-THRESHOLD CMOS CIRCUIT DESIGN METHODOLOGY FROM 2D TO 3D

A thesis submitted in partial
fulfillment of the requirements for the degree of
Master of Science in Computer Engineering

By

Ross Thian
Harding University
Bachelor of Science in Computer Engineering, 2008
Harding University
Bachelor of Science in Computer Science, 2008

December 2010
University of Arkansas

ABSTRACT

A new and exciting approach in digital IC design in order to accommodate the Moore's law is 3D chip stacking. Chip stacking offers more transistors per chip, reduced wire lengths, and increased memory access bandwidths. This thesis demonstrates that traditional 2D design flow can be adapted for 3D chip stacking. 3D chip stacking has a serious drawback: heat generation. Die-on-die architecture reduces exposed surface area for heat dissipation. In order to reduce heat generation, a low power technique named Multi-Threshold CMOS (MTCMOS) was incorporated in this work. MTCMOS required designing a power management unit (to control when and which gates are powered), a MTCMOS gate library, and a state saving D-Flip-Flop. This thesis demonstrates converting a traditional 2D chip to a low heat 3D chip design with the use of MTCMOS technology using industry-standard CAD tools.

This thesis is approved for recommendation
to the Graduate Council.

Thesis Director:

_____
Dr. Jia Di

Thesis Committee:

_____
Dr. Dale R. Thompson

_____
Dr. James P. Parkerson

_____
Dr. Scott C. Smith

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed _____
Ross J. Thian

ACKNOWLEDGEMENTS

I thank Dr. Jia Di very much! For his unfailing guidance, direction, encouragement, and patience! Thank you!

Ahmad Al-Zahrani – Previous MTCMOS work

Alicia Donaghey

Brent Hollosi

Cortney Hart

Clay Pankhurst

Daniel Allen

Jason Horst

Joseph Smeal

Jessica Powviriya

Jonathan Harris

Kevin Smith

Mike Hinds

Rob Mize

Stephanie Clark

Steve Comer

Wiwat Leebhaisomboon

My mom who said just put it down.

Thanks to my Father!

HIM!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABELS

1. INTRODUCTION

In very simple terms, 3D chip stacking is very similar to the way a city develops. People move to a new area and commerce begins to explode. As more and more people buy land, the land becomes expensive. Instead of buying more land to house their businesses people develop upwards and skyscrapers emerge. These skyscraper businesses house all of their workers and each part of the business can communicate with other parts in the same building. No longer do employees of the business need to travel from one location to another. Also, all of the business records are accessible from one location. This business becomes a single efficient design.

The effect of 3D chip stacking allows for more transistors per chip, therefore, more processing power. Moore's Law states that the number of transistors per chip will continue to double every eighteen months [1]. Physically this is impossible because transistors cannot get smaller than the atomic level and the current level of manufacturing limits the size of the chip. Also, chips are size limited because fabrication defects increases as the chips get larger. The idea is similar to wiring a house verses a full mall, as complexity increases so does the probability for errors. With chip design one mistake can make for a faulty chip. The alternative is to keep the chips at a producible size and stack them up [2] [3].

To keep Moore's Law after approaching the physical limit of reducing transistor size, IC designers build the chip upwards [4]. One benefit of 3D chips is increased communication bandwidth. 2D chips have a problem with long wire lengths which cause increased power consumption and a reduction in usable chip area for placing logic. On

the contrary, 3D designs allow logic to be stacked and relocated so that connected logic areas can be closer to each other. Skyscrapers, for instance, can house small businesses and rental apartments in one location. Instead of driving from one place to another for meetings or going home to sleep, a person can save time by staying in the skyscraper and riding the elevator.

3D chips improve on-chip communication partially because they are composed of stacked dies. A die is a single-layer logic level of a 3D chip like a floor on a skyscraper. 3D chips communicate to other dies by tunneling metal down to the next die much like an elevator in a sky scraper. 3D increases the number of connections by allowing horizontal and vertical communication. 3D communications use through-chip interconnects placed throughout the die area. Not only is this applied for die-to-die communication but even more importantly, interconnects increase communication to memory even allowing each bit to be accessed directly from the die above. Therefore, memory accessing performance dramatically increases [4] [5]. There is now much more room for memory on chip, because a separate die for memory can be placed on a 3D package [6].

The traditional 2D metric of Moore's Law showed a linear increase in performance based on transistor density alone. It can be expected that 3D power density would increase faster than 2D [7]. High power density generates excessive heat. Heat dissipation is much more of a factor with 3D chips because stacked logic is dense and surrounded by a poor thermal conductor, silicon [8]. Hot spots form at where trapped heat cannot dissipate and where there are dense active regions of logic. Therefore, one disadvantage of 3D IC architecture is heat dispersion from increased power density of

stacked logic [6]. Reducing heat on a 3D chip is the most important issue to making 3D chip design unequivocally better than 2D.

There are a few methods proposed for reducing internal heat on a 3D chip. One method is to use Through Silicon Vias (TSV's) as thermal conductors that tunnel vertically through dies and dissipate heat to the surface layers. Another method in development is to pump liquids through the 3D chip for cooling [3].

Multi-Threshold CMOS (MTCMOS) is implemented in this thesis as a strategy for reducing power generation. The goal is to turn off logic gates when they are inactive to facilitate heat dispersion with TSV's. MTCMOS works by disconnecting the power source from the logic gates when they are not being used to reduce leakage power. The target is to reduce idle power consumption when the chip is inactive. Thermally, hot spots can still occur for low power chips but when combined with thermal via technology, this methodology should allow for improved hot spot reduction. MTCMOS also allows using a low power circuit design methodology without the overhead of adding any additional mechanical structures such as water cooling.

One of the issues associated with 3D MTCMOS synchronous designs is balancing the clocks on multiple dies. With multiple layers of a 3D chip, the clock needs to be synchronized across each layer. A clock can be out of synchronization with another die and cause setup/hold time violations, resulting in circuit malfunction. Clock balancing among multiple dies in a 3D chip is demonstrated in this thesis.

3D chip design also requires logic partitioning. This is to specify where a logic block will be placed among several dies. This is like taking a single floor plan and

specifying what floors certain offices are on. This thesis demonstrates the steps needed to partition a 2D design into 3D while tackling the most important issue, 3D heat dispersion. After mitigating the 3D heat problem it can be expected that not until today's limits with 2D designs are reached on 3D designs (such as routability and memory bandwidth access) that a plateau of increased performance will exist.

2. GOALS

3D chip design has heat issues because heat dissipation only occurs on the outer layers. Some recent research attempts to implement interior heat dissipation using liquid cooling and heat conducting dummy thermal vias up to the heat sink [4]. This thesis' approach is to use the Multi-Threshold CMOS low power technique to reduce heat consumption in 3D chip design [9] [10] [11]. MTCMOS provides a currently attainable heat reduction technique at design time. The goal is to use the MTCMOS technique for lower power, which in turn causes reduced heat generation. First, the MTCMOS gates must be defined. Then, a power management unit needs to be created to control MTCMOS operation. The next step is the comparison between two methods for partitioning logic on each die. By relocating the logic the attempt is to determine how logic partitioning affects hot spot generation. Finally, a methodology is demonstrated to convert a single 2D netlist into 3D.

3. DESIGN PROGRESSION

The organization of this thesis follows steps taken to design the 3D Multi-Threshold CMOS 16-stage pipelined floating point unit (FPU) co-processor. The design started from a 2D Defense Advanced Research Project Agency (DARPA)-sponsored project using a CMRF8SF IBM 130nm PDK during the summer of 2009 for developing and demonstrating low power methodologies [12]. The design evolved into a 3D low heat design during a Tezzaron-sponsored project using a Chartered 130nm Low Power process [13].

The 2D Design started from a floating point co-processor which was initially pipelined in four stages using CMOS logic, as shown in Figure 1. The floating point co-processor implements the IEEE 754 standard without sticky bit implementation [14]. This co-processor was designed using high threshold voltage gates and low threshold voltage gates. The next step was to create a MTCMOS design with a power management unit, as shown in Figure 2. The power management unit turns off gates for saving power when the circuit is idle. These three designs were compared for speed and power characteristics for the DARPA project. The final product for the 3D Tezzaron run demonstrated two different 16-stage MTCMOS pipelined implementations. Figure 3 shows how one of the stages is designed.

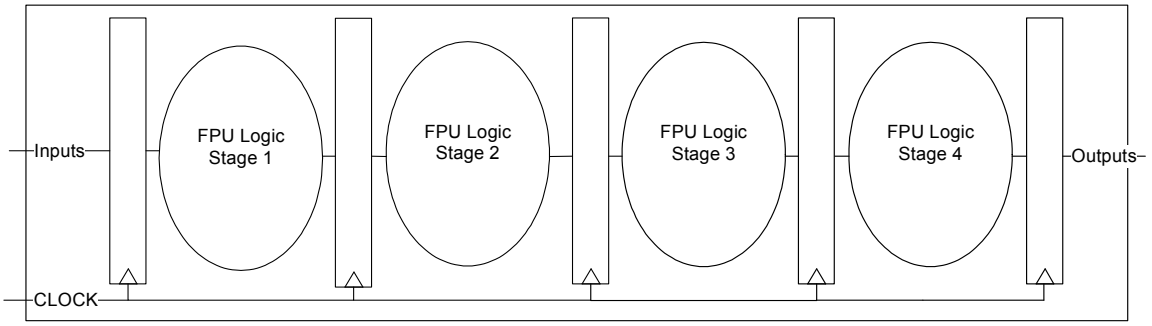

Figure 1. 2D Regular CMOS 4-stage pipeline Floating Point Unit Co-processor

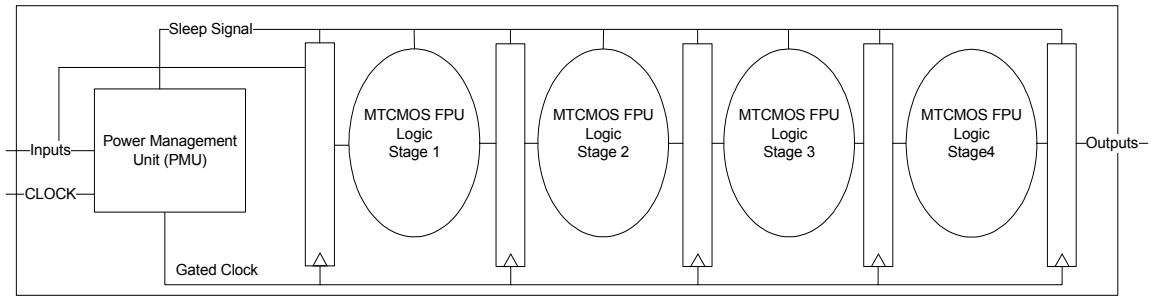

Figure 2. MTCMOS FPU 4-stage pipeline design with Power Management Unit (PMU)

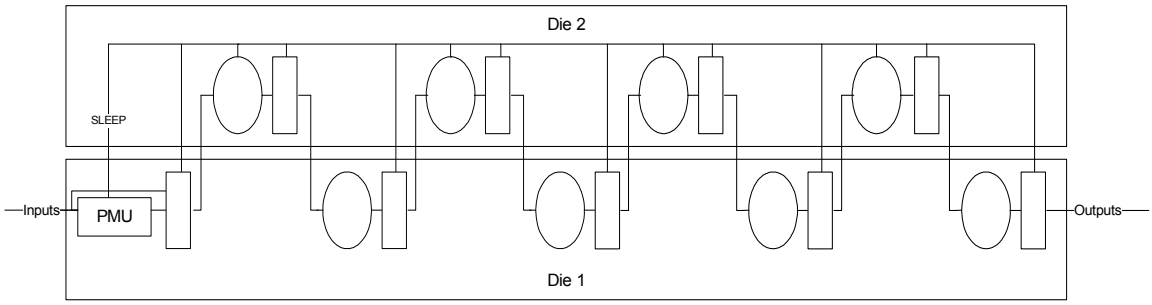

Figure 3. 3D MTCMOS FPU 16-stage pipeline design with Power Management Unit

4. MULTI-THRESHOLD CMOS (MTCMOS)

In chip design, MTCMOS provides a power saving solution. External devices that have a low activity factor can save what was once wasted static power. Inactive gates still draw power in the idle state. This wasted power is called leakage power. MTCMOS saves power by gating the power-ground path in logic gates during idle state.

4.1 Power Saving Philosophy

$$P = C_L V_{DD}^2 f_{0\to1} + t_{sc} V_{DD} I_{peak} f_{0\to1} + V_{DD} I_{leakage}$$

Dynamic Power | Short-Circuit Power | Leakage Power

Figure 4.CMOS circuit power equations [15]

In physics, power is the amount of energy converted over time. Any time energy is converted from one form to another heat is expended. By using less energy, less heat is wasted. Therefore to understand how to use less heat it is necessary to realize how CMOS technology uses power. Figure 4 shows that the CMOS power equation can be broken into three power consuming components. The first component is dynamic power, which consumes the most power because it is based on V_{DD}^2. Frequency represents the switching activity of the circuit. Therefore, as clock speed increases so does dynamic power usage due to switching activity. Short-circuit power is consumed during the time a gate has a direct connection from power to ground. This is dependent on switching activity because short circuiting only occurs during logic transitions. Leakage power

dominates power consumption when there is limited switching activity [16]. MTCMOS provides a way to reduce leakage by gating the path between power and ground.

4.2 Multi-Threshold CMOS Design

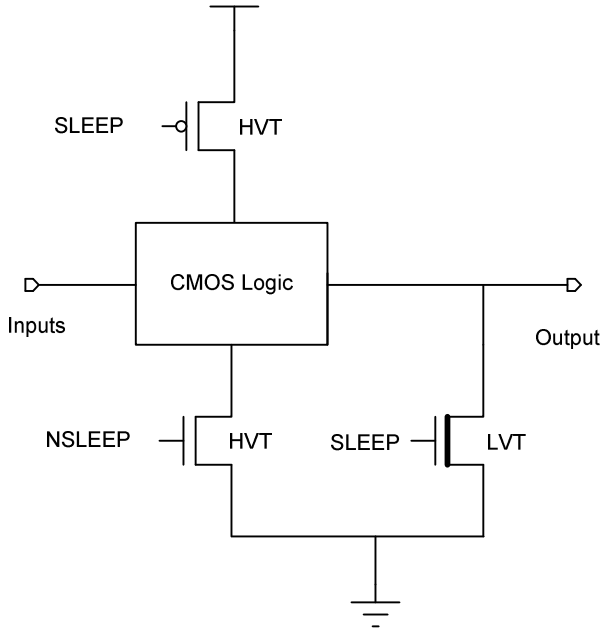

Figure 5. Fine Grained Multi-Threshold CMOS design with HVT and LVT transistors [9] [10]

MTCMOS sacrifices a small reduction in speed for a gain in leakage power savings [17]. Regular CMOS gates consist of single-threshold transistors, i.e., all NMOS/PMOS transistors have the same threshold voltage, respectively. Depending on the threshold voltage used, CMOS circuits may either be fast and leaky or less leaky but slow. On the other hand, Multi-Threshold CMOS uses two kinds of transistors with different threshold voltages. The high voltage threshold (HVT) transistor requires a higher voltage to switch. The higher the threshold voltage the less leakage power is

allowed to pass. Also, the longer time the transistor will take to switch. Conversely, low voltage threshold (LVT) transistors operate much faster, but have larger leakage. In conclusion, Multi-Threshold circuits combine the speed of LVT transistors and the leakage savings of HVT transistors to form a balance between power savings and speed [9] [10].

Figure 5 shows how CMOS gates are modified to MTCMOS. HVT transistors gate the power-ground path in the CMOS logic. A LVT transistor, shown by the heavier transistor symbol, is used to drive the output to logic 0. When sleep mode is activated (logic 1 for SLEEP and logic 0 for NSLEEP) both HVT transistors become open circuits. At this time the LVT output transistor becomes shorted. A disabled sleep mode function resolves to a logic 0 for SLEEP and logic 1 for NSLEEP. The HVT transistors are shorts allowing power connection to ground. The output LVT transistor is open and normal CMOS operation occurs [9] [10].

4.3 DFF_m Design

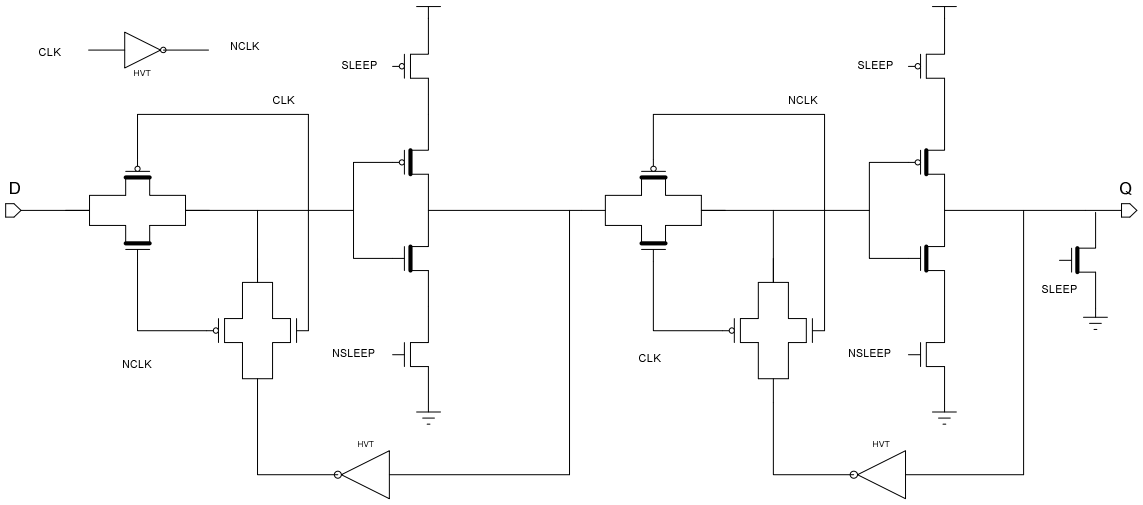

Figure 6. DFF_m D-Flip-Flop with MTCMOS power gating [18]

Figure 6 shows the DFF_m schematic. This is a redesigned D-flip-flop that has the capability to go into sleep mode. The basic operation of this flip-flop is to have two states: the first is to load new data and the second is to save data. When clock is logic 0, the first transmission gate passes the input through the first MTCMOS inverter. Then

clock rises and the input now present after the first MTCMOS inverter comes back around through an inverter and passes through a transmission gate to the MTCMOS inverters input. This saves the state of the D input prior to the clock rising edge. The logic state at the MTCMOS inverter also passes through the next transmission gate. It then it gets inverted by the MTCMOS inverter to be present at Q. Next, the clock falls again and the Q value is inverted and passes through a HVT transmission gate to be saved. The critical path of the logic goes through four gates: transmission gate to gated inverter once again to transmission gate to gated inverter. All of these transistors are LVT design while the other non-critical paths use HVT transistors to reduce leakage power. This includes the three inverters and the two bottom transmission gates. The final output is driven low by a HVT transistor.

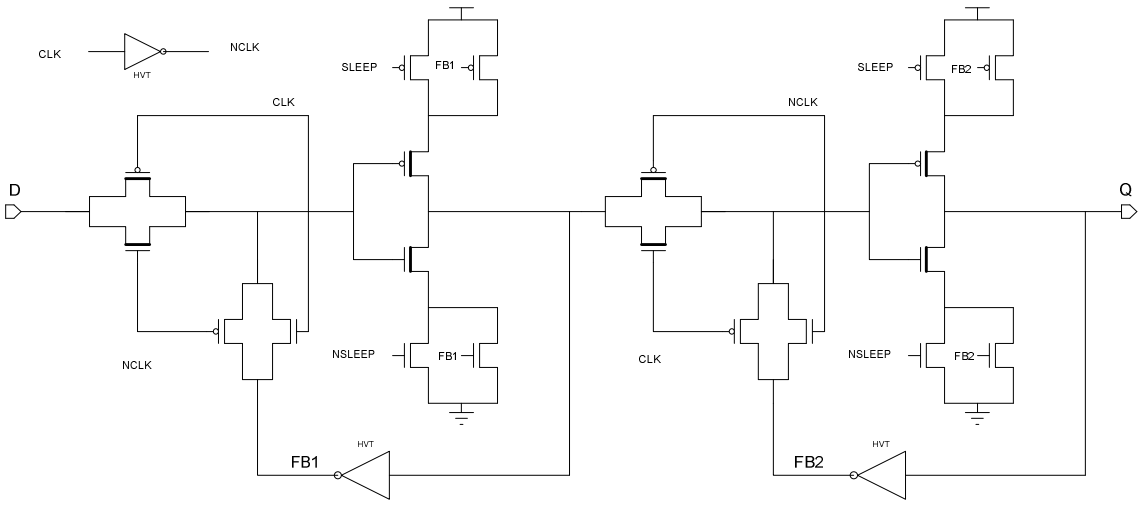

Figure 7. DFF_MTCMOS state saving D-Flip-Flop [18]

The DFF_MTCMOS has a similar design to the DFF_m. The function of the DFF_MTCMOS is the same as DFF_m when sleep is disabled. Additional transistors

were added to retain state when in sleep mode while also saving some leakage power. Figure 7 shows the schematic of the DFF_MTCMOS gate. When sleep mode is activated the MTCMOS inverters go to sleep but the feedback (FB) transistor stays on keeping the current state of the MTCMOS inverter. This works by using the value of an inverted output (FB1 or FB2) to represent the state of the MTCMOS inverter. This state will turn on one of the HVT feedback transistors in the MTCMOS inverter. It will allow the transistor needed to connect the correct logic output to either power or ground [18].

5. PIPELINE STAGES TOP LEVEL DESIGN

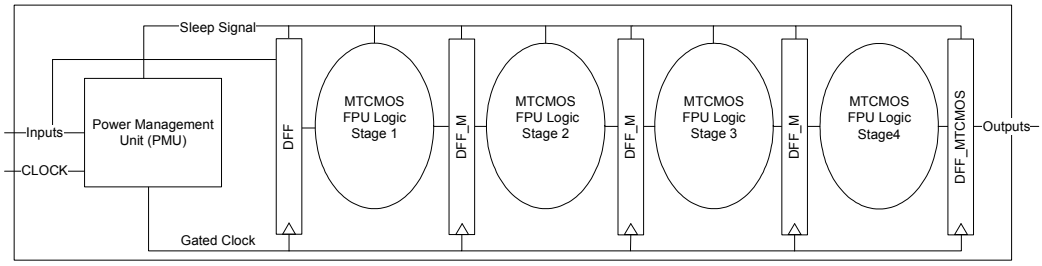

Figure 8. Power management unit and MTCMOS co-processor connection

The pipeline model has different registers for improving leakage savings [15]. Maximizing the number of gates of MTCMOS design saves power. Therefore, the middle DFF_M registers are used to save leakage power by going into sleep mode. In Figure 8, the first register DFF is a normal register latching all new inputs. The final register is a DFF_MTCMOS register that retains their logic state during sleep mode. When sleep is activated the middle stages will lose their data. When sleep is deactivated, the data will have already been latched to the output of the first register stage. Data will then propagate though the combinational logic.

6. POWER MANAGEMENT UNIT

6.1 Overview

The Power Management Unit (PMU) provides clock gating and sleep-control for the MTCMOS co-processor [15] [19]. The PMU functions as a sleep timer counting up to a specified value before turning off the gates. Input signals to the MTCMOS co-processor are monitored. When no new input patterns occur, the data set does not need to be computed and sleep operation can occur. The number of pipeline stages in the MTCMOS design determines the amount of time left before sleep is activated. This wait-for value is determined by an external setting dependant on the time required to flush the pipeline. A timer begins counting up to the wait-for value. Once the counting has been matched a sleep signal is triggered to the MTCMOS co-processor and clock gating occurs. Normal operation of the MTCMOS design resumes when inputs change.

6.2 Overview of components

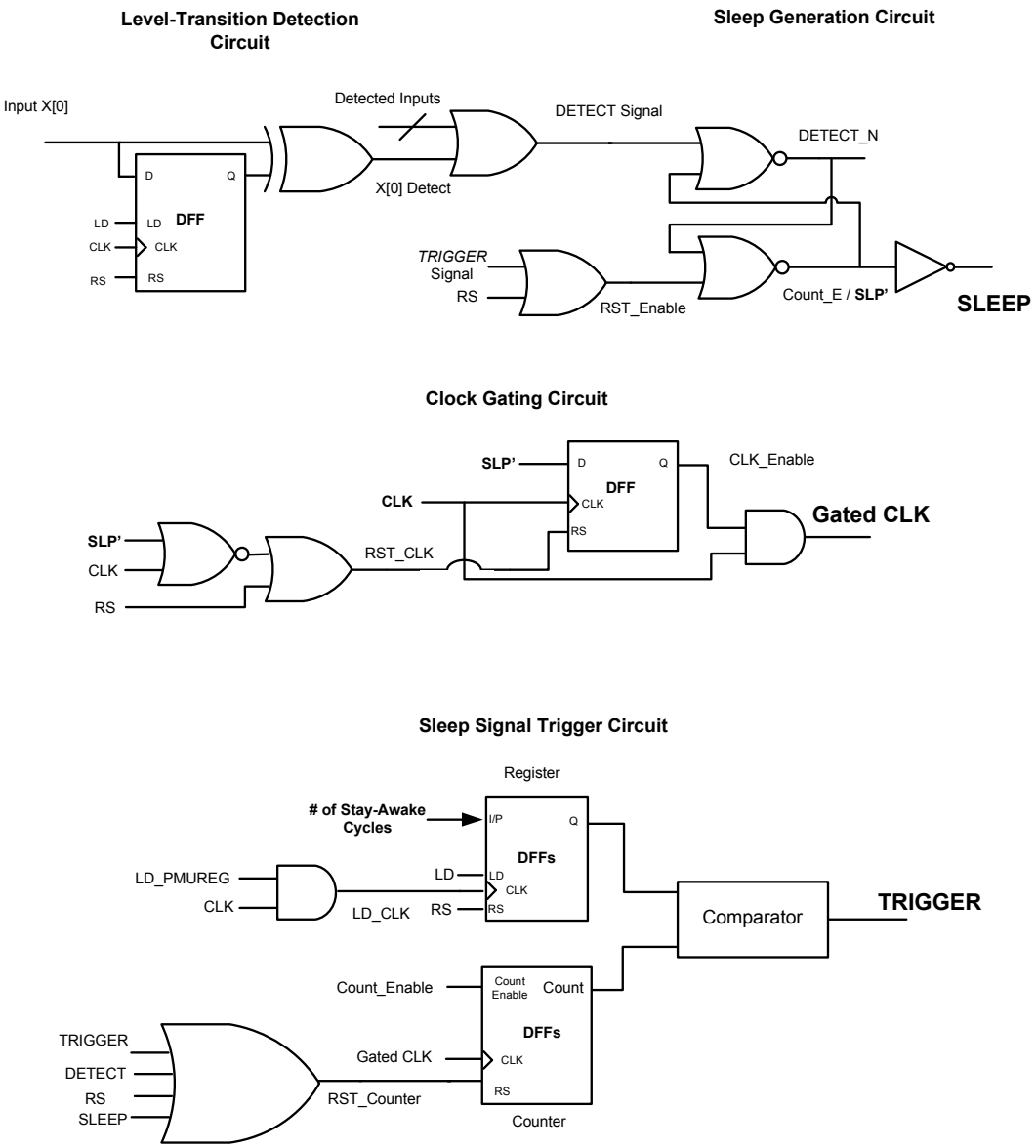

Figure 9. Power Management Unit (PMU) Schematic

Figure 9 shows the full schematic view of the Power Management Unit. The Level-transition detection circuit detects new input combinations and generates a transition detection signal for the Sleep Generation circuit. The Sleep Generation circuit toggles the sleep signal. The Trigger Signal determines when the sleep generator should activate sleep mode by comparing a counter register and a preloaded register value. The counter begins when no new signals have occurred and sleep mode is disabled. Clock gating occurs at the bottom DFF and depends on the current state of Sleep mode. In the following sections the circuit will be analyzed in their parts.

6.2.1 The Level-Transition Detection Circuit

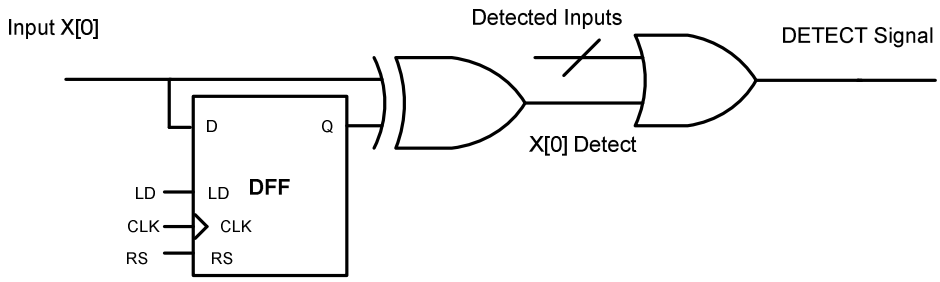

Figure 10. Level-Transition Detection Circuit

The Level-Transition Detection circuit compares sequential inputs and produces a detect signal if the current input is different from the previous value, as shown in Figure 10. A new input arrives at the D input of the DFF and is compared with the value at Q using an XOR gate. All logic inputs pass through the detection circuit and are combined

with OR logic to produce a single detect signal. The CLK input is clock which allows for the detect signal to be a full-pulse-width. This is a redesign that DETECT signal returning to the CLK input of the DFF. It was recommended by the previous designer to make sure that the detect pulse width is long enough to latch new data. In schematic simulation it was found that the gate delays to produce the DETECT signal was not long enough to prevent a setup time issue. Glitches would also occur because the detection of each input did not occur at the same time. The DETECT signal would then be stuck at a logic state even when new input patters arrived. The solution was using a clock to extend the DETECT Signal pulse width to prevent these issues.

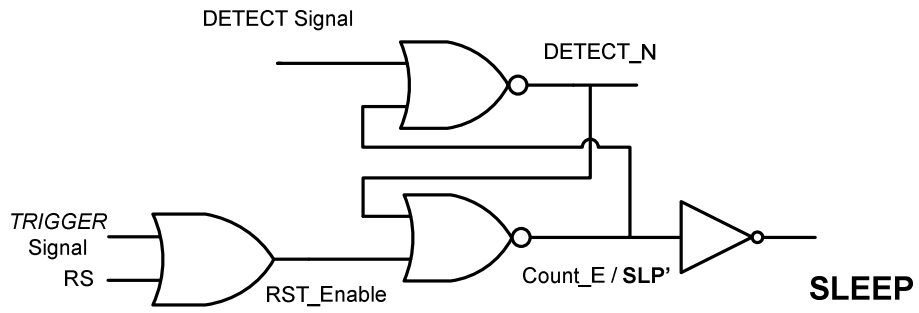

Figure 11. Sleep Generation circuit and Reset Enable

The sleep generation circuit controls sleep signal toggling, as shown in Figure 11. When the DETECT signal is asserted from an input transition and the RST_Enable is logic 0, SLEEP will be deactivated. Sleep will always be activated when RST_Enable is logic 1. RST_Enable is activated when the counter register activates the trigger signal or an external reset signal becomes active. When both the Detect signal and RST_Enable

are logic 0, sleep does not change. This circuitry activates the counter register and clock gating.

6.2.2 **Clock gating**

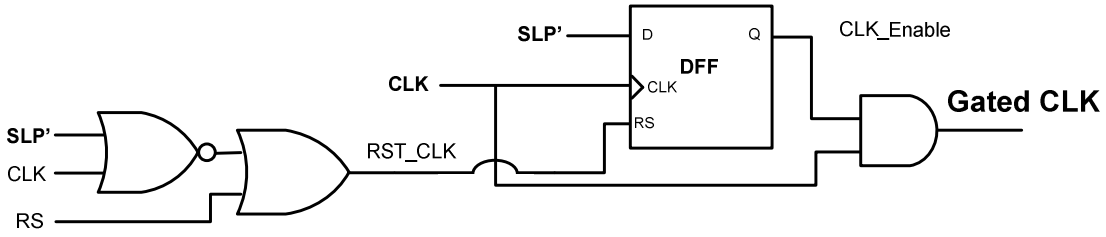

Figure 12. Clock Gating Circuitry [19]

Figure 12 shows the clock gating and reset circuitry. In normal clock gating using the AND gate is all that is necessary. In contrast, our design gates the clock based on a sleep signal that changes on a falling edge. This method of clock gating causes one less clock cycle than desired. The solution is to add a register that delays the inverted sleep signal (SLP') for one half clock period. The clock is now gated to the correct number of clock cycles. The reset logic inverts the clock and resets the gating register after the last desired clock period.

6.2.3 The Sleep Trigger Signal

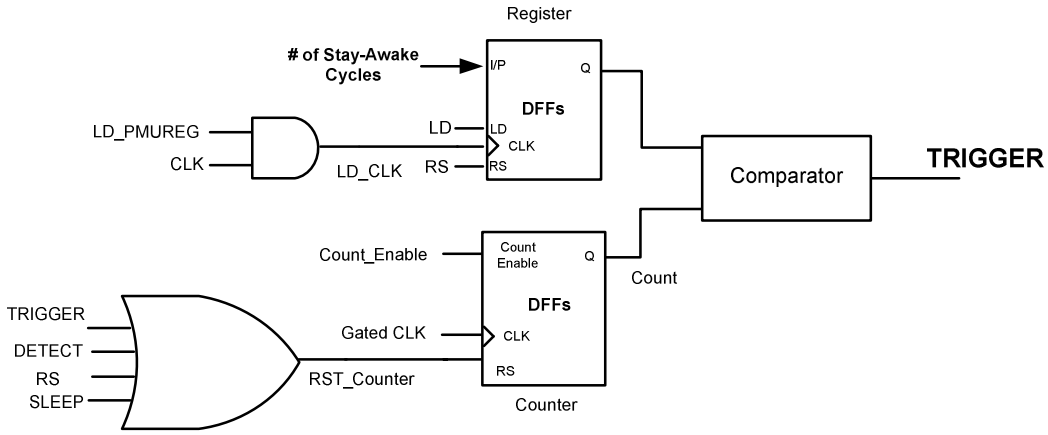

Figure 13. Sleep trigger signal generation circuit [19]

The TRIGGER signal is used to activate sleep mode and disable the clock. The trigger signal is generated from the comparison of two registers: a counter register and a count-to register (see Figure 13). The first is a counter register that counts upwards. This circuit relies on two key inputs: the Count_Enable from the Sleep Generation circuit and a DETECT signal. When Count_Enable is enabled, the counter register begins counting upwards if the RST_Counter is not resetting the register. The DETECT signal passes through the RST_Counter OR gate to the RS input of the counter. While detection still occurring, the counter register is continually being reset to 0. Therefore, counting begins when Count_Enable is logic 1 and detection is no longer resetting the registers values.

The bottom register stores the number of stay-awake cycles dependant on how long the clock and sleep signal should be active after new input detection has ceased.

This register's value is specified externally and is latched only when the LD_PMUREG is enabled. LD_CLK is clock gated by the LD_PMUREG signal. The comparison of this register and the counter register occurs with XOR to OR logic and produces the TRIGGER signal.

7. CLOCK BUFFERING

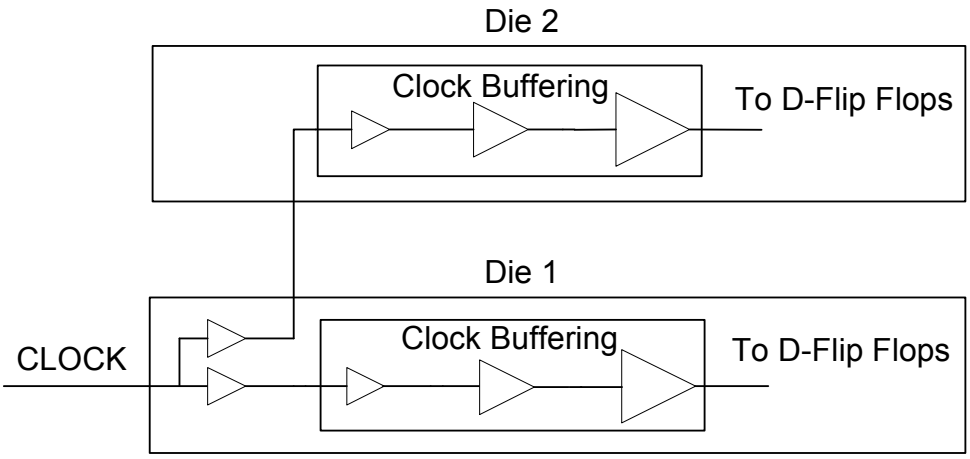

Figure 14. Clock distribution between dies

Clock distribution is essential to synchronous designs. An unbalanced clock tree can cause clock skews and other timing-related errors. In a pipeline design with an unbalanced clock tree, one register stage could receive its clock earlier than another. One example is having a combinational logic block surrounded by registers. If the preceding register is clocked slightly earlier then the output register, the new data could propagate fast enough to be latched by the output register when the previous combinational result should be latched. With a 3D design, which has lots of die-to-die communication, additional measures must be taken to ensure clock balancing [20].

Clock buffering is done by a buffer script written in Python. The buffer script buffers nets depending on their fan-out. This script is used to buffer the clock and sleep signals. First, this script will calculate the fan-out for all signals. Then it will utilize a list of buffer drive strengths to determine which buffers are needed to drive a signal. The buffer script will build up to the correct buffer strength needed by using a series of

inverters. If the fan-out of a net is higher than the maximum buffer drive strength, branching occurs and multiple buffers are used instead. It is necessary to make sure each branch has the same logic depth. If otherwise a skew can cause unbalanced signal arrival for different branches. The buffer script creates the proper branching depth for clock buffering.

When buffering a 3D design the clock on each die needs to be balanced. Figure 14 shows the balancing of clock buffering. All external inputs go to Die 1 before going to Die 2. Therefore some additional balancing is for each die to have the same clock depth. The solution is to manually add two buffers as the start of the clock tree. These buffers will be directly connected to the clock and are present on Die 1. One of those buffers will be connected to Die 2; the other passes to the logic on Die 1. Then the buffer script is run on both dies separately. The clock should be balanced for both dies at this point.

8. SLEEP SIGNAL DISTRIBUTION

The sleep signal is distributed to combinational logic and MTCMOS flip-flops. Each gate requires a sleep and inverted sleep signal. The fan-out for sleep is significantly larger than clock but can be buffered using the same strategy. The sleep signal is generated from the PMU on Die 1 and is passed to Die 2. Sleep timing is not as critical as clock but should be balanced as well. The balancing of the sleep signal should be implemented the same way as clock balancing demonstrated before. The buffer script can be run once for both clock and sleep balancing.

9. 3D DIE PARTITIONING

9.1 Overview

In 3D IC design a designer must consider how to partition the logic for each die. A simple example is a floating point unit which has add, subtract, and multiply operations, and the designer may place the multiplier on its own die. Partitioning is important for not only attempting to gain access to a bottom memory layer but also for minimizing hot spots. The organization of logic on die can cause hotpots particularly if high active logic areas are stacked on top of each other. Dense active regions are reduced by using a 16-stage pipeline which segments computational logic into stages Some regions may still have denser circuits than others when a design is automatically placed and routed onto a die. However, this should not be as significant as a whole multiplier stacked on top of another. The goal is to reduce vertically high activity areas by researching different partitioning strategies.

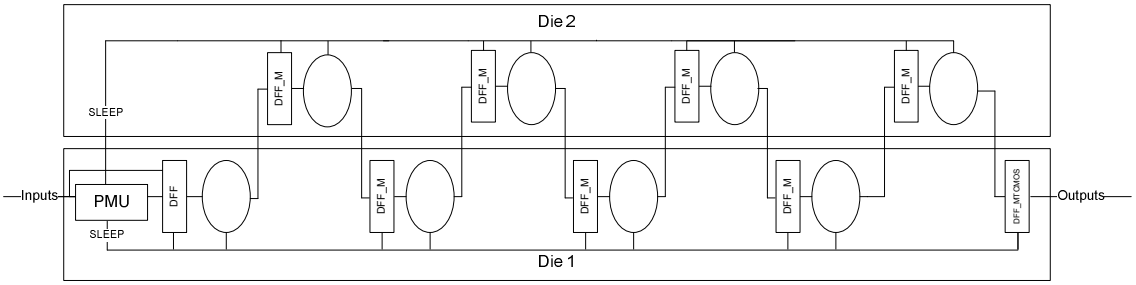

Figure 15. Partitioning Strategy A

Partitioning strategy A distributes the logic by placing odd and even stages on different dies shown in Figure 15. It is expected that other than the power management unit, both dies should have about the same amount of logic.

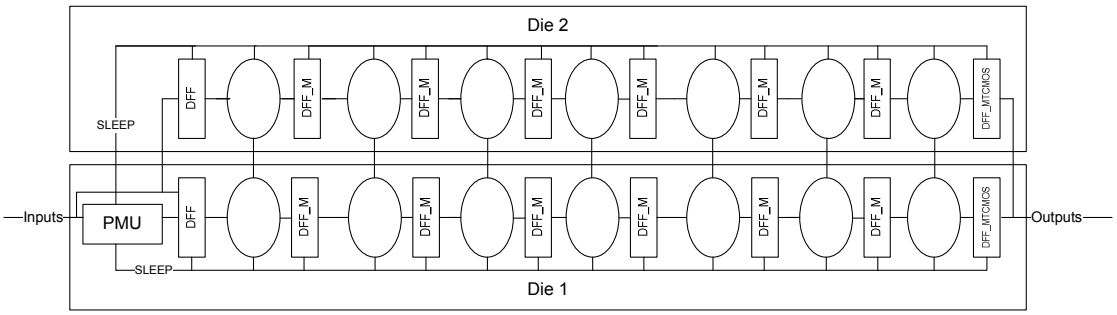

Figure 16. Partitioning Strategy B

Partitioning strategy B has 16 stages but the stages are placed in parallel shown in Figure 16. Half of external inputs are sent to the DFF register on Die 1 and the other half to Die 2. Then each combinational logic block is divided in half between dies. Because of this partitioning strategy, inputs must rejoin through the combinational logic for proper computation to occur. Therefore, at each combinational logic block, there is cross communication with the logic on the opposite die.

PMU placement - Both partitioning strategies place the power management unit on Die 1. This is because inputs arrive to Die 1 and the power management unit must detect new inputs as early as possible to determine if sleep and gated clock should be activated.

10. TEZZARON MODEL

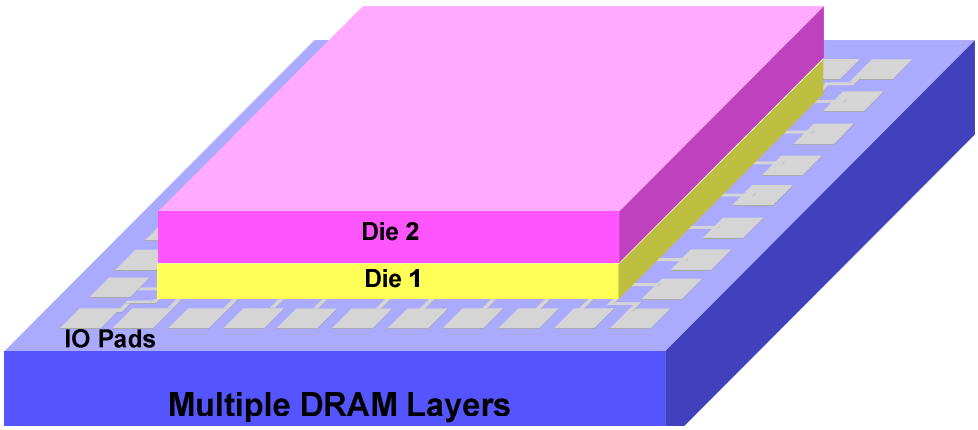

Figure 17. Tezzaron 3D die placement [21]

Tezzaron Semiconductor specializes in 3D chip stacking on memory. While our low-heat design does not use the memory layer, other teams on this fabrication run do use it. We use the Chartered 130 nm low power process [13]. Figure 17 shows how inputs and outputs are connected to Die 1 and how the different dies are stacked. Die 2 is placed on top of Die 1 with the pads connected to Die 1.

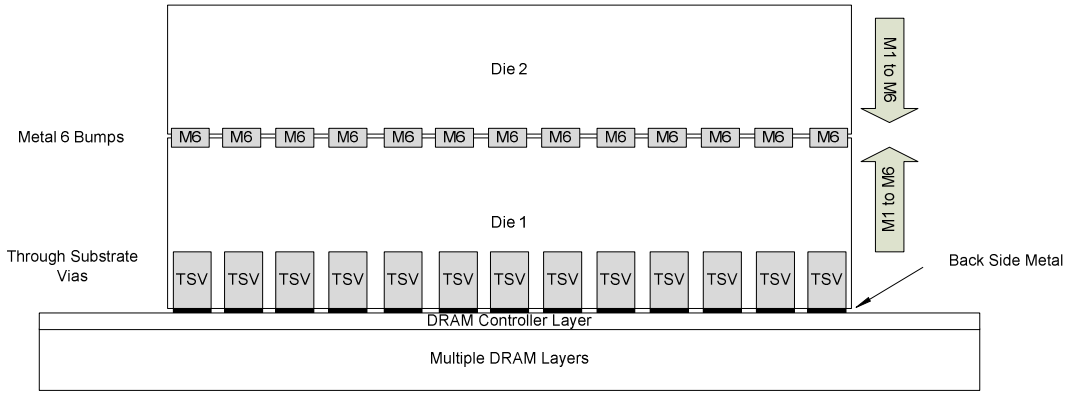

Figure 18. Tezzaron die-to-die connections

The I/O Pads are placed on the surface of the DRAM Controller Layer as shown in Figure 18. The DRAM controller layer connects to through silicon vias (TSVs) with backside metal on Die 1 [5] [21]. The I/O pads and DRAM memory communicate to Die 1 through the TSVs. The TSVs then connect through the bottom silicon to the metal 1 layer on Die 1. Connections from Die 1 to Die 2 are through the metal 6 layer. There are metal 6 bumps that connect the Die 1 to Die 2. The name "TSV" represented a way to connect between dies. Therefore, the name "TSV" is used in the 2D to 3D net list separation section as a connection from Die 1 to Die 2.

11. 3D DIE SEPERATION

In the new area of 3D chip stacking, design tools are not readily available. Designing a 3D chip requires keeping die-to-die communication in mind. Our approach is to take a VHDL net list, convert it to a Verilog net list and then use a series of scripts to convert the design to separate net lists, making them 3D ready.

11.1 Net list from 2D to 3D

The first step is to code the 2D design in VHDL. At this step it is best to take the 2D VHDL design and simulate functionally.

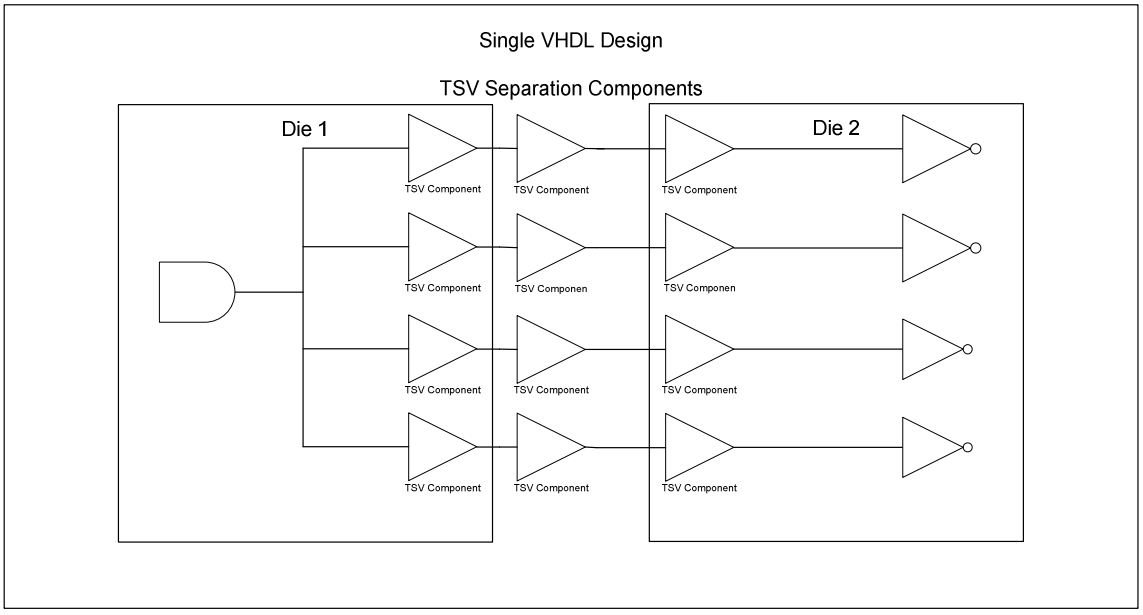

Figure 19. 2D VHDL TSVComponent Placement Locations

11.1.1 TSV Separation using TSVComponents

The second step is to implement the partitioning strategy by manually inserting TSVs (die-to-die connections) in VHDL. Separating the 2D net list into two dies of 3D design (Die 1 and Die 2) requires adding TSVComponents that serve as die segregators. It is necessary to create place holders where we are going to split the net list. The TSVComponents are VHDL buffers that output a single input. Figure 19 shows how a single driver from Die 1, in this case from an AND gate, drives multiple connections in Die 2. The TSVComponents are placed in the sets of three. The first TSVComponent driven by the AND gate represents a pin output connection from Die 1. The middle TSVComponent represents a placeholder for separating Die 1 from Die 2. The right/final TSVComponent is the Die 2 input pin. To connect in the reverse direction from Die 2 to Die 1 the TSVComponents are placed in the opposite driving direction. The name Die 1 TSVComponents applies to the TSVComponents directly connected to Die 1, shown inside the Die 1 box of Figure 19. Likewise Die 2 Components are directly connected to a Die 2 gate. The middle connection is named Die 1 to Die 2 connection. A reversely named Die 2 to Die 1 connection is unnecessary because these placeholders will be removed after splitting the VHDL design into two parts. The final pin names are the names of the wires connected to the middle Die 1 to Die 2 Component. In the Figure 19, this component is outside of the Die 1 and Die 2 regions. Renaming is written about in the next step. Good naming convention is depicted in Figure 20. At the end of this step the 2D single VHDL net list should be partitioned by TSVComponents.

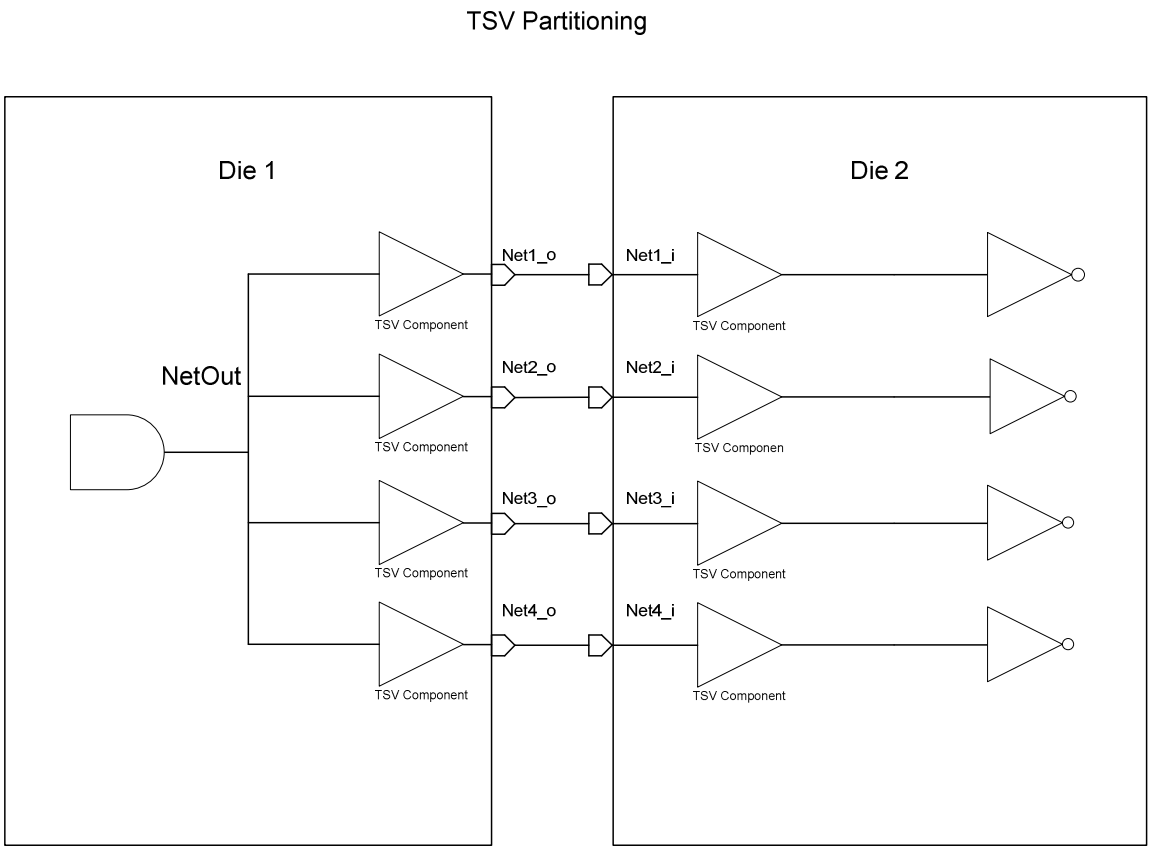

Figure 20. 3D TSV partitioning using Python script removes Die 1 to Die 2 components. Pins created by wire name to Die 1 to Die 2 component.

11.1.2 TSV Partitioning

The TSV partitioning step uses a Python script to separate the net list. In order to do so, first the VHDL 2D partitioned design is converted to a flattened Verilog net list using Synopsys Design Vision. Here the VHDL net list is loaded into the tool and resaved as a flattened Verilog net list. Then the TSV partitioning script uses the middle components to find the die-to-die separation point. The middle components are removed and a net list for each die is created. The wire names that were connected to the middle

components are now replaced by pins. Figure 20 shows the result of partitioning creating a separate Die 1 and a Die 2 net list. Also, a top level net list that connects both dies is necessary. To create the top level net list, along with adding the original ports, pins that are named the same but are on different dies are connected together.

A connection from Die 1 to Die 2 is actually through metal bumps that contact from die-to-die. It is believed that the connections between Die 1 and Die 2 require specific TSV components at the transistor-level schematic design. In turn the TSVComponents are actually metal connections between dies and are not used in the final design. The pins themselves are enough to connect dies.

11.1.3 Post Partitioning TSV Removal

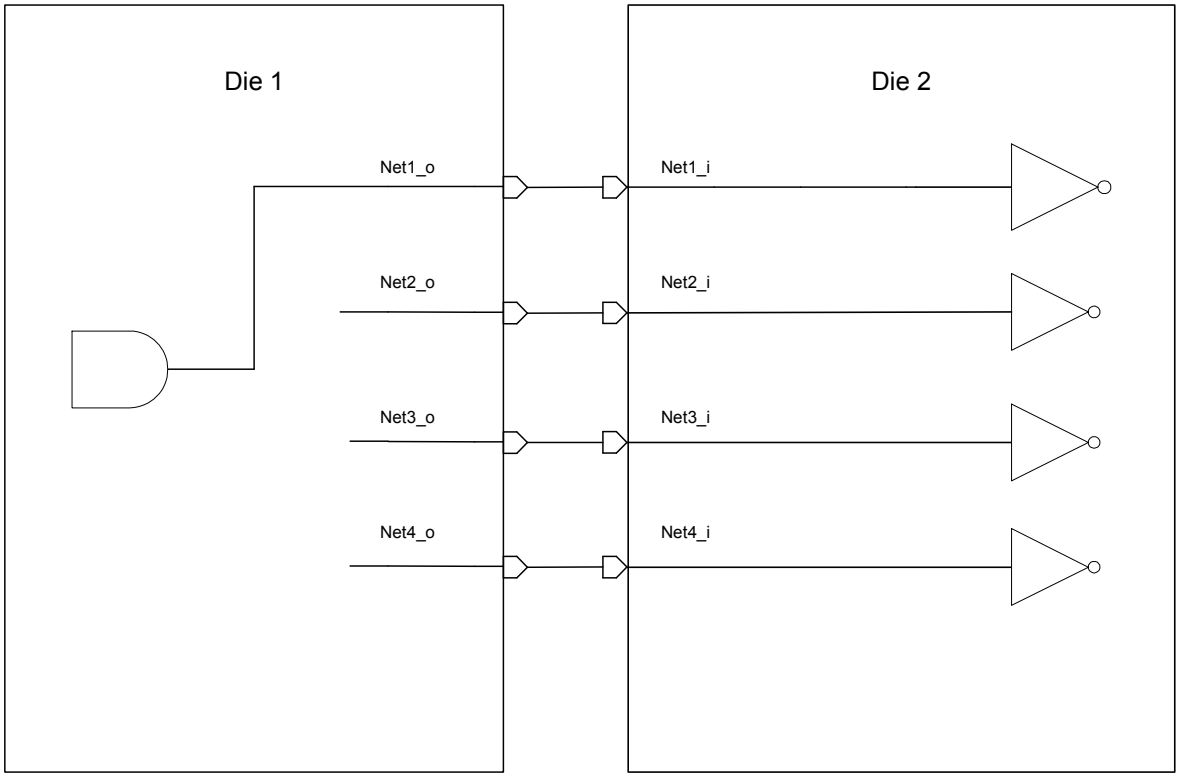

Figure 21 Post-partitioning TSV removal and resulting unconnected nets issue.

After partitioning the dies, we removed the TSVComponents as shown in Figure 21. The TSVComponents were no longer necessary because metal connections between dies take the form of pins. Post-partitioning TSV removal script finds and replaces each die's TSVComponents and the pin name replaces the wire name. Comparing Figure 20 and Figure 21 shows that this post-partitioning TSV removal replaces *NetOut* with the name *Net1_o*. A problem arises when TSVComponents are removed. A wire that drives multiple TSVs can only connect to one pin. Our schematic simulator Cadence does not allow multiple pins to be driven by one wire [22]. Each wire that is connected to a pin

takes the name of that pin; therefore a single internal wire cannot be connected to more than one pin. When the next TSVComponent is found, the *NetOut* wire cannot be replaced to *Net2_o* because it has already been replaced to *Net1_o*. To handle this issue we store a list of pins that should be replaced for each wire, in this case a list of {*Net1_o, Net2_o, Net3_o, Net4_o*} for *NetOut*. This Script is run on both dies separately and a list of duplicate nets is generated for each die to be used later. The first wire, *Net1_o*, is the only one connected to the original driver; other nets need to be reconnected.

11.1.4 Duplicate Nets Removal

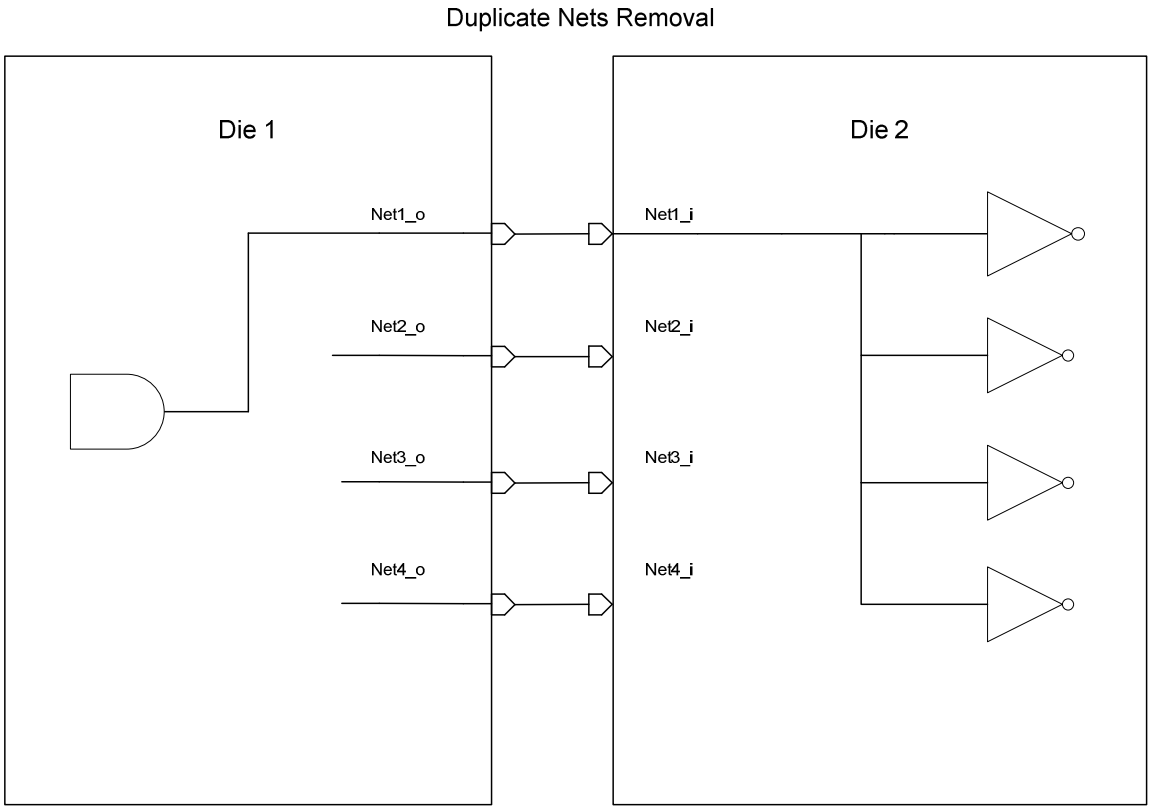

Figure 22 Duplicate nets removal restores unconnected nets.

The previous Figure 21 shows that *Net2_i* to *Net4_i* are not connected to a driver. To reconnect those wires we apply the Duplicate Nets Removal Script. The two Verilog net lists are processed sequentially along with a text file generated from the opposite die that has the lists of the duplicates from the Post Partitioning TSV Removal Script. The first pin always forms a connection to a net. Successive pins need to be connected to the driver. The unconnected pins on Die 1 correspond to connected pins on Die 2. Die 2 does not know that these are unconnected pins. Thus we use the duplicate text file from the opposite die to indicate which set of pins is actually connected to a driver. These pins, which are also wire connections to components, are replaced with the first pin that is in the list. The result of this Duplicate Nets Removal script is shown in Figure 22.

11.1.5 Unused Pins Removal

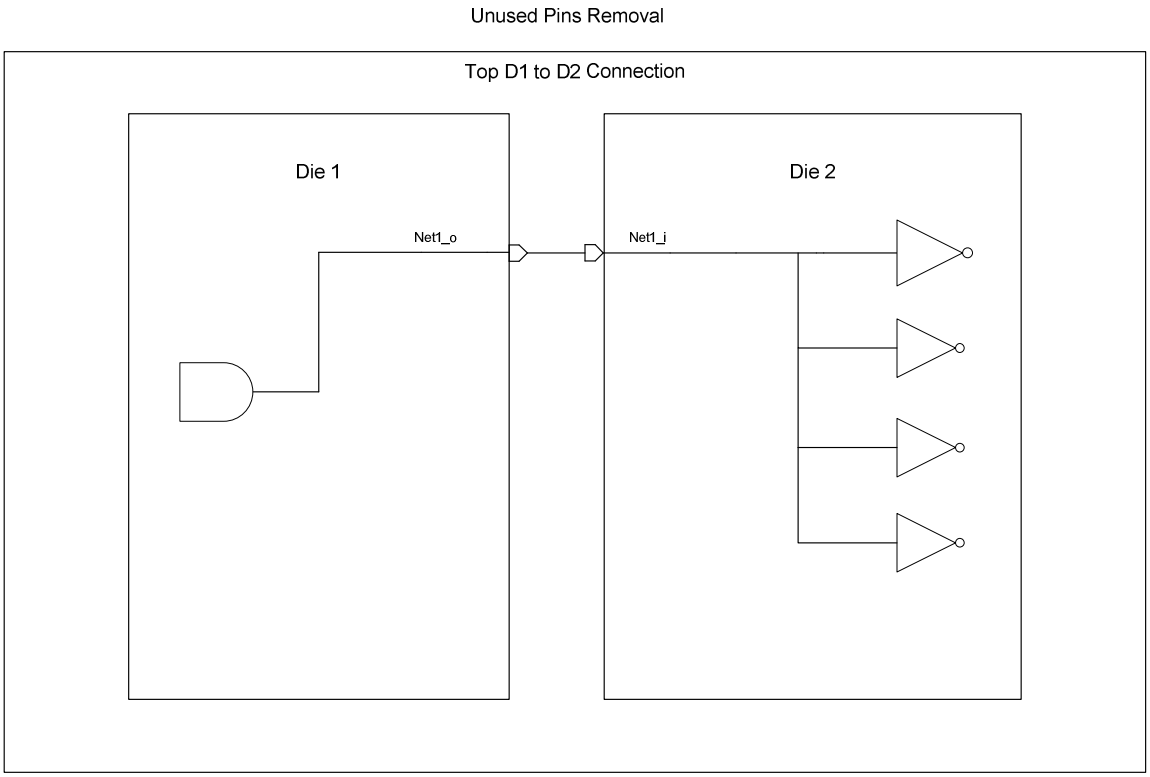

Figure 23. Unused Pin Removal Script removes unused pins from top level net list of Die 1 and Die 2.

Post-duplicate net removal leaves the unused pins. To remove these, we run an Unused Pins Removal Script, Figure 23. This script will find all unconnected pins in Die 1 and Die 2 net lists and remove them. Once the unused pins are found, they are also removed from the top level Die 1 to Die 2 connection net list. The final results are three pin-removed net lists that are ready for functional simulation. This concludes the steps needed to partition a 2D net list into a 3D net list.

12. SIMULATION RESULTS

The MTCMOS 16-stage pipeline floating point co-processor design was simulated in Cadence using the Ultrasim simulator [24]. Figure 24 shows the simulation waveform output. This waveform shows the proper output when after 17 inactive clock cycles sleep is activated. The value of the PMU register is set to 11 HEX, which is 17 clock cycles. The MTCMOS design has 16 pipeline stages, which require two registers around each stage. The reason seventeen clock cycles are used is because there are a total of 17 registers, and each register requires one clock cycle for data to propagate through it. Data persistence is demonstrated at sleep activation. Notice that at sleep activation the data output that was present at Z persists as long as sleep is active. Also, during sleep the clock ceases to toggle.

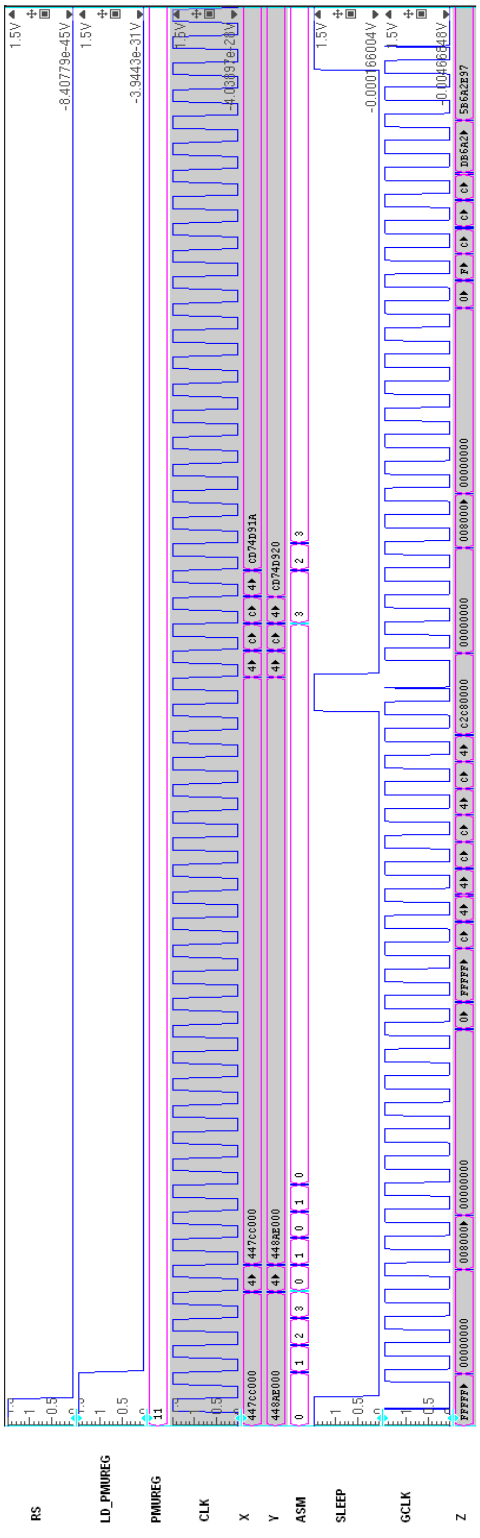

Figure 24. Final Simulation Results – 3D 16-stage floating point co-processor

12.1 Power Analysis

Comparison of Leakage Power between CMOS and MTCMOS Co-Processors			
Speed (MHz)	Design	Average Sleep Mode Power (mW)	Percent of power saved by MTCMOS in sleep mode
25	CMOS MTCMOS	2.7693 0.1293	95.33%
50	CMOS MTCMOS	5.4736 0.0901	98.35%
75	CMOS MTCMOS	8.5594 0.2059	97.59%
100	CMOS MTCMOS	11.2422 0.7532	93.30%
125	CMOS MTCMOS	14.9665 0.2527	98.31%
150	CMOS MTCMOS	16.9857 0.1387	99.18%
175	CMOS MTCMOS	19.924 0.2519	98.74%
200	CMOS MTCMOS	22.7884 0.2925	98.72%
225	CMOS MTCMOS	26.4538 0.8161	96.91%
250	CMOS MTCMOS	29.0793 0.473	98.37%
Average Power Savings:			97.48%

Table 1. Comparison of MTCMOS verses CMOS power consumption in sleep mode.

The full MTCMOS core design with PMU was simulated for power against the same design using CMOS gates without PMU. Power simulation was conducted with Cadence Virtuoso UltraSim because it provides a fast simulation solution to large designs [24]. To simulate this design in a reasonable time frame, UltraSim simulator settings were set to Digital Fast and the aggressive timing mode. These results can be trusted because simulation in a more accurate simulation mode for UltraSim demonstrated better

MTCMOS power savings than that with the fast simulation. Normal operation of the MTCMOS design will have very long periods of inactivity and the intention is to save heat and power when sleeping. Therefore, simulation power results were taken from the average power consumed during sleep mode. Initially there is a surge in power consumption due to the added switching activity of all MTCMOS gates going into sleep mode and sleep signal tree switching activity. Since our simulation ran for ten clock cycles the initial peaks in power would elevate the real average power consumption. At 150 MHz the total initialization sleep mode power was 50 mW that would be around 30 times the average sleep mode power consumption of the CMOS design. The sleep signal tree accounted for ¼ of the total power consumed when transitioning into sleep mode. Additionally, the second clock cycle after sleep initialization was not averaged because the gated clock makes a quick spike to logic 1 before resolving to zero, shown in Figure 24. This causes extra switching activity among the D-Flip-Flops and a slight increase in power consumption over the mean case. Therefore, power results were taken after the first two clock cycles of sleep activation until final sleep deactivation to avoid averaging these power results. Both MTCMOS and CMOS designs were simulated while operating from 250 MHz down to 25 MHz, with each step as 25 MHz, as shown in Table 1. The results show that regardless of speed, MTCMOS provides a consistent leakage power saving over CMOS counterpart. In conclusion, the heat generation of MTCMOS designs when inactive will be greatly reduced from CMOS designs because MTCMOS saves 97.48% of the power of CMOS.

13. FINAL DESIGN

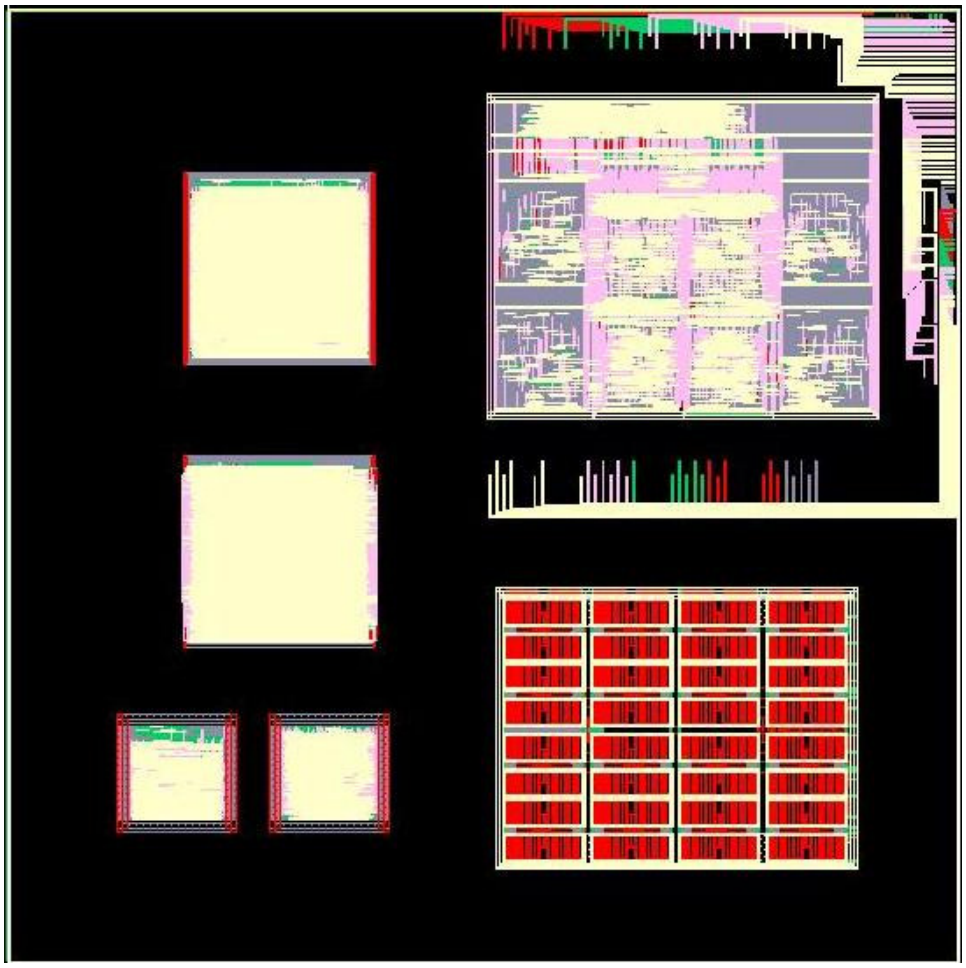

Figure 25. Die 2 Top

Figure 25 shows the final layout of Die 2, the top die. The two designs at bottom left are the MTCMOS 16-stage floating point co-processors, partitioned using the two strategies discussed in Chapter 9. This die is placed on top of Die 1 shown in Figure 26. The Die 1 MTCMOS designs are located at bottom right. Die 1 shows the connection to the pads with the extra wiring.

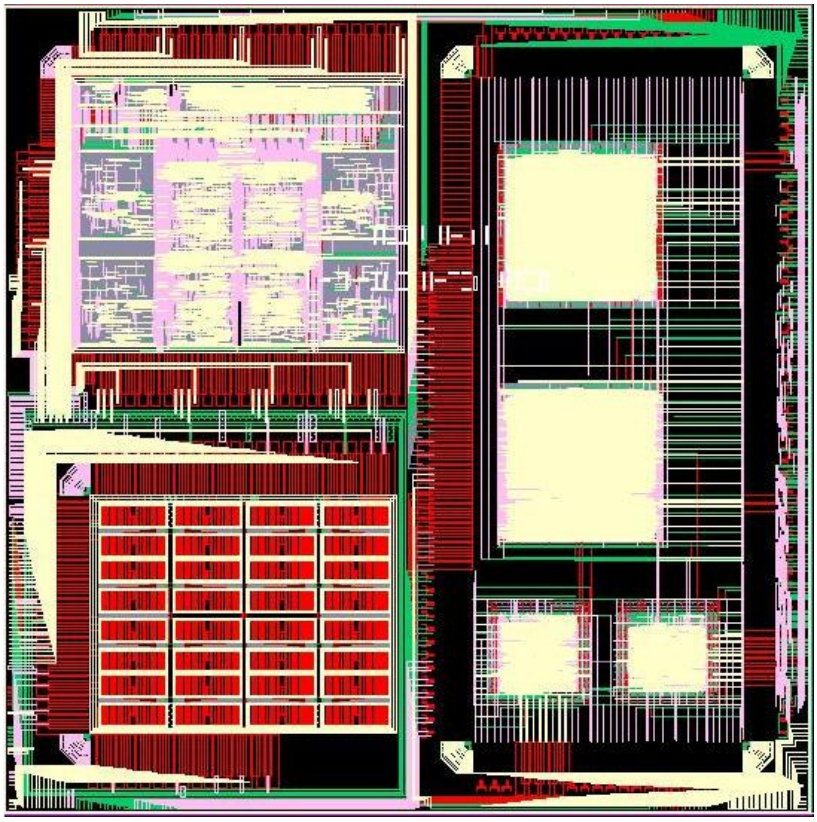

Figure 26. Die 1 Bottom

14. CONCLUSION

In conclusion, this thesis demonstrates a design methodology to implement a 2D CMOS circuit in 3D Multi-Threshold CMOS circuit for low heat characteristics. The designs discussed were two partitioning strategies applied to a 16-stage MTCMOS floating point co-processor. The design proceeded in steps starting with 2D CMOS HDL code developing into the final 3D design. The low power technique MTCMOS was used to reduce heat consumption when in sleep mode. A power management unit was used to control when MTCMOS gates enter sleep mode and activate clock gating. Distributing the clock and sleep signals require balancing for skew between dies. Next, partitioning of the logic was done to compare hot spot generation. Additionally, using a series of scripts, a 2D VHDL design with a 3D partitioning strategy is split into multiple Verilog net lists, which are 3D ready. The final design was simulated for functionality and power demonstrating that this 3D partitioning methodology works and MTCMOS produces lower heat than CMOS designs.

REFERENCES

[1] N. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. Hu, M.Irwin, M. Kandemir, V. Narayanan, "Leakage Current: Moore's Law Meets Static Power," Computer, vol. 36, no. 12, pp. 68-75, Dec. 2003

[2] J. A. Cunningham, "The use and evaluation of yield models in integrated circuit manufacturing," *Semiconductor Manufacturing, IEEE Transactions on*, vol.3, no.2, pp.60-71, May 1990

[3] P. Mercier, S. R. Singh, K. Iniewski, B. Moore, P. O'Shea, "Yield and Cost Modeling for 3D Chip Stack Technologies," *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, vol., no., pp.357-360, 10-13 Sept. 2006

[4] T. Stephen, *"A Survey of 3D Circuit Integration"* March 14, 2008

[5] K.Takahashi, M.Sekiguchi, *"Through Silicon Via and 3-D Wafer/Chip Stacking Technology,"* Symposium on VLSI Circuits Digest of Technical Papers, pp.114-117, June 2006.

[6] B. Black, M. Annavaram, E. Brekelbaum, J. DeVale, L. Jiang, G.Loh, D. McCauley, P. Morrow, D. Nelson, D. Pantuso, P.Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. *"Die Stacking (3D) Microarchitecture*," In Proceedings of MICRO-39, December 2006.

[7] J.M. Rabaey, *Low Power Design Essentials*. New York: Springer, 2009. Print.

[8] IBM, "15 Moore's Years," *IBM Research - Zurich*. Web. September 07 2010. <http://www.zurich.ibm.com/news/10/moore.html>.

[9] S. Mutoh et. al., "*1-V Power Supply High-Speed Digital Circuit Technology with Multithreshold-Voltage CMOS*," IEEE Journal of Solid-State Circuits, Aug 1995, pp. 847-854.

[10] J. Kao and A. P. Chandrakasan, "Dual-Threshold Voltage Techniques for Low-Power Digital Circuits," IEEE Journal of Solid-State Circuits, Vol. 35, NO. 7, July 2000

[11] H. Seongmoo, B. Kenneth, and K. Asanović. "*Reducing power density through activity migration,"* Proceedings of the 2003 international symposium on Low power electronics and design, August 25-27, 2003, Seoul, Korea

[12] IBM CMOS8RF (CMRF8SF) Design Manual.

[13] Chartered Semiconductor Manufacturing Ltd. “0.13um CMOS Logic/Mixed Signal/Rf Technology Design Rules” Version 1R February. 2009

[14] IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic.

[15] J. S. Yuan and J. Di, “Teaching Low-Power Electronics Design in Electrical and Computer Engineering”, IEEE Trans. Education, Vol. 48, no. 1, February. 2005, pp 169-181.

[16] Z. Chen, M. Johnson, L.Wei, and K. Roy, “Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks,” In *Proceedings of the 1998 international Symposium on Low Power Electronics and Design* 1998.

[17] B.H Calhoun, F. A. Honore, and A. Chandrakasan, “Design methodology for fine-grained leakage control in mtcmos” In Proceedings of International Symposium on Low Power Electronics and Design, 2003

[18] J. Kao and A. Chandrakasan, “MTCMOS Sequential Circuits,” Proc. ESSCIRC 2001, 2001.

[19] A.AL-Zahrani, *Design and Analysis of Mulit-Threshold CMOS (MTCMOS) Techniques In Synchronous and Asynchronous Digital Designs.* MS thesis. University of Arkansas, Fayetteville 2009

[20] Jackson, Srinivasan, Kuh, *"Clock routing for high-performance ICs,"* Design Automation Conference, pp. 573-579, 27th ACM/IEEE Design Automation Conference (DAC '90), 1990

[21] MPW 100109 Design Guide Rev 8

[22] http://cadence.com Cadence Design Systems

[24] Virtuoso UltraSim Simulator User Guide November 2006

APPENDIX A. STEPS FOR CONVERTING 2D NETILST TO 3D NETLIST

1. VHDL to Verliog.
    - Rename all components starting with d1_ on die 1 or d2_ on die 2.
    - Then add Add - Seperation Components
        i. Name: "TSVd1buff" if the TSV is on die1 and with label ending with "_die2"
        ii. Name: "buffremove" in inbetween with ending label "_remove"
        iii. Name: "TSVd2buff" if the TSV on die2 and with label ending with"_die1"
    - New VHDL Modules need to be created: TSVd1buff, buffremove, and TSVd2buff. They should pass the same input logic to output.

    - Label Input Signals from the other die should have ending label "_a_i"
    - Label Output Signals from the other die should have ending label "_a_o"

2. Flatten VHDL design to Verilog using synthesis tool such as DesignVision, Leonardo or etc.

3. Separate TSVs into die1 and die2 files
    - Run split.py
    - It will ask for a file name give it your verilog netlist.
        i. All labels ending with "_die1" are TSVs belonging to die1. The corresponding component would be TSVd1buff
        ii. All labels ending with "_die2" are TSVs belonging to die2. The corresponding component would be TSVd2buff
        iii. All labels ending with "_remove" are buffremove components that should be deleted.
    - Once die1 and die2 files are obtained this way, Verify the inputs and outputs for each file:
        i. In die1 file, all signals ending with "_a_o" are to be made primary outputs. These signals connect to die2
        ii. In die2 file, all signals ending with "_a_o" are to be made primary outputs. These signals connect to die1
        iii. All "_a_o" signals are outputs of TSVs
        iv. In die1 file, all signals ending with "_a_i" are to be made primary inputs. These signals connect from die2
        v. In die2 file, all signals ending with "_a_i" are to be made primary inputs. These signals connect from die1
        vi. All "_a_i" signals are inputs of TSVs

4. Simulate functionally using die1 and die2 netlist.
    - Form a connecting top level netlist in VHDL. By connecting complementary tsv signals together.

- Simulate the toplevel netlist with the die1 and die2 together.
- Flatten the connecting netlist to verilog if not already done. This will be used later for cadence simulation.

5. The following steps are ran individually for each die
6. Step1:d1
   - Run ByeRandomLines.py this python code formats the verilog text
     i. [Post Split Netlist.v]
   - Replace all logic 0 and logic 1 (1'b1) signals to vdd and gnd signals in verilog netlist. Remove signals from module list and port list.
   - Run BPI.py This code expands busses and buffers external inputs.
     i. [Post Split Netlist_lines.v]
7. Step2:d1
   - Run PSTR.py > duplicates.txt Pipes out a list of duplicates.
   - No prompt will be shown, type in your netlist name.
8. Repeat Step1:d1 and Step2:d1 for die 2.
9. Step3:d1
   - Run DR.py
   - [other die dupliactes.txt] *Step 2*
   - [netlist name]
     i. When a duplicate net is removed from Die 1 All instances of that net are combined to form one net.
     ii. Die 2 still has all the now unused nets. These must be combined to the same net as well.
     iii. d1 duplicates.txt has a list of the one used and all the ones replaced
10. Step4:d1
    - Run fan.py
      i. Remove vdd and gnd from the fanout file generated.
      ii. A clean file and fanout is generated.
    - Run buffer.py
      i. [filename_clean.v]
      ii. [If new cells are used with new input add to regular expression. Exclude output i.e: .a|.A|.b|B]
11. Step5:d1
    - Remove Sleep
      i. die1: Find all " dff d1_reg0" remove sleep and nsleep ports.
      ii. die2 : repeat for " dff d2_reg0"
    - Run ReplaceSigs.py replaces verilog generated gate names for ones used in schematic simulation.
      i. Edit Sigs.txt : ['old','new']
12. Step6:d1
    - Run AddPowandGnd.py
      i. OutfileName: d1_copPG.v
      ii. Check for extra vdd and gnd at module list and portlist

- Run CadenceReady.py verilog

13. Step7:d2
    - Repeat Step3:d1 to Step6:d1 for die2.
14. Step8: Toplevel netlist: Remove logic1 and 0 from all d1 and d2 netlist.
    i. find ".expi1_0(1'b1), .qmulbit_0(1'b0), "
15. Step9: Top
    - Run TBES on top level verilog
16. Step10:
    - AddPnGTop.py – Same as ADDPOWANDGND.PY with vdd and VDD switched. Likewise gnd and GND are switched.
        i. Adds the VDD and GND connections from top to dies.
17. Step 11:
    - Run CadenceClean.py
18. Step12
    - Run upr.v unused pin remove
        i. Pins left over from tsv removal are wiped.
        ii. Check if logic pins are still there
19. Step13
    - Import to cadence d1 d2 and top rmport files.

APPENDIX B. SPLIT.PY

```
# Author : Mike Hinds
import re

#specify Filename
path = raw_input("File name (TopLvlNetlist.v):")

#clean the input file
def handlewire(item):
      print "wire found"
      wires = item.split(',')
      wirecount = 0
      outline = "wire "
      wires[0] = wires[0].lstrip()
      wires[0] = wires[0].lstrip("wire")
      print wires[0]
      for wire in wires:
            outline = outline + wire+', '
            wirecount = wirecount + 1
            if wirecount == 50:
                  outline = outline.rstrip(', ')+';'
                  fout.write(outline+'\n\n')
                  wirecount = 0
                  outline = "wire "
      outline = outline.rstrip(', ')+';'
      fout.write(outline+"\n\n")

findwire  = re.compile("\s+wire\s+")
fin = open(path, 'r')
fout = open("clean_bool", 'w')
lines = ''
while fin:
      line = fin.readline()
      if line == '':
            break
      #take care of comments
      if   (line.find("//")   !=   -1   or   line.find("*/")   !=   -1)   or
line.lower().find("endmodule") != -1:
            lines = lines+line.rstrip() + "\n"
      else:
            lines = lines+line.rstrip()

listlines = lines.split(';')
for item in listlines:
      if findwire.match(item):
            handlewire(item)
      else:
            fout.write(item+';\n')

fin.close()
fout.close()


#Separate Components
curblock = -1
findprimmoduledec = re.compile("module")

findwirelist = re.compile("\s*wire\s+")
findcompdecs = re.compile("\s*\S+\s+\S+\s*\(.+\)")
```

```
findmoduledecs = re.compile("\s*module\s+(\S+)\s*\(.+\)")
findd1component = re.compile("\s*(\S+)\s+d1_\S+\s*\(.*\)")
findd2component = re.compile("\s*(\S+)\s+d2_\S+\s*\(.*\)")

findinput                                                                    =
re.compile("\.[a|b|c|d|sleep|nsleep|clk]\s*\(\s*(\S*)\s*\)\s*,?")
findoutput = re.compile("\.[z]\s*\(\s*(\S*?)\s*\)")

findprimin = re.compile(".*_a_i")
findprimout = re.compile(".*_a_o")

d1components = []
d2components = []
d1comptypes = []
d2comptypes = []
#make sure I remove duplicates before combining inputs and outputs in
this array
d1sigs = []
d2sigs = []
d1primin = []
d2primin = []
d1primout = []
d2primout = []
d1mods = []
d2mods = []

mod =""
moddie = 0

#open cleaned file
fin = open("clean_bool", 'r')
while 1:
      line = fin.readline()
      if line == '':
            break

      #Determine which block we are in
      if (findprimmoduledec.match(line)) and (curblock == -1):
            curblock = 0
            print "entered primary module declaration..."
            print line

      if (findwirelist.match(line) and curblock == 0):
            curblock = 1
            print "entered wire list..."
            print line

      if (findcompdecs.match(line) and curblock == 1):
            curblock = 2
            print "entered component declarations..."
            print line

      if (findmoduledecs.match(line) and curblock ==2):
            curblock = 3
            print "entered module declarations..."
            print line

      # 0 Primary Module Declaration - determined by the first "module"
declaration
      if curblock == 0:
            a =0
```

```
      # 1 Wire List - determined by the first "wire" declaration
      if curblock == 1:
            a =0

      # 2 Component Declarations - determined by the first component
declaration
      if curblock == 2:

            #get die1 components
            d1comp = findd1component.match(line)
            if (d1comp):
                  d1components.append(line)
                  if not(d1comp.group(1) in d1comptypes):
                        d1comptypes.append(d1comp.group(1))

                  #get input and output signals for the component
                  inps = findinput.finditer(line)
                  #print line
                  for inp in inps:
                        if     not(inp.group(1)     in     d1sigs)     and
not(findprimin.match(inp.group(1))):
                              d1sigs.append(inp.group(1))
                              #Check for primary inputs

                        if       findprimin.match(inp.group(1))       and
not(inp.group(1) in d1primin):
                              d1primin.append(inp.group(1))
                              #print inp.group(1)

                  outps = findoutput.finditer(line)
                  for outp in outps:
                        if     not(outp.group(1)     in     d1sigs)     and
not(findprimout.match(outp.group(1))):
                              d1sigs.append(outp.group(1))
                              #check for primary outputs
                        if      findprimout.match(outp.group(1))      and
not(outp.group(1) in d1primout):
                              d1primout.append(outp.group(1))
                              #print outp.group(1)

            #get die2 components
            d2comp = findd2component.match(line)
            if (d2comp):
                  d2components.append(line)
                  if not(d2comp.group(1) in d2comptypes):
                        d2comptypes.append(d2comp.group(1))

                  #get input and output signals for the component
                  inps = findinput.finditer(line)
                  for inp in inps:
                        if     not(inp.group(1)     in     d2sigs)     and
not(findprimin.match(inp.group(1))):
                              d2sigs.append(inp.group(1))
                              #Check for primary inputs
                        if       findprimin.match(inp.group(1))       and
not(inp.group(1) in d2primin):
                              d2primin.append(inp.group(1))
                        #print inp.group(1)

                  outps = findoutput.finditer(line)
                  for outp in outps:
                        if not(outp.group(1) in d2sigs):
```

```
                              d2sigs.append(outp.group(1))

                              #print outp.group(1)

            if line.find("TSVd1buff") != -1:
                  d1components.append(line)

                  #get input and output signals for the component
                  inps = findinput.finditer(line)
                  #print line
                  for inp in inps:
                        if     not(inp.group(1)     in     d1sigs)     and
not(findprimin.match(inp.group(1))):
                              d1sigs.append(inp.group(1))
                              #Check for primary inputs
                        if       findprimin.match(inp.group(1))       and
not(inp.group(1) in d1primin):
                              d1primin.append(inp.group(1))
                              #print inp.group(1)

                  outps = findoutput.finditer(line)
                  for outp in outps:
                        if     not(outp.group(1)     in     d1sigs)     and
not(findprimout.match(outp.group(1))):
                              d1sigs.append(outp.group(1))
                              #check for primary outputs
                        if       findprimout.match(outp.group(1))      and
not(inp.group(1) in d1primout):
                              d1primout.append(outp.group(1))
                              #print outp.group(1)

            if line.find("TSVd2buff") != -1:
                  d2components.append(line)

                  #get input and output signals for the component
                  inps = findinput.finditer(line)
                  for inp in inps:
                        if     not(inp.group(1)     in     d2sigs)     and
not(findprimin.match(outp.group(1))):
                              d2sigs.append(inp.group(1))
                              #Check for primary inputs
                        if       findprimin.match(inp.group(1))       and
not(inp.group(1) in d2primout):
                              d2primin.append(inp.group(1))

                  outps = findoutput.finditer(line)
                  for outp in outps:
                        if not(outp.group(1) in d2sigs):
                              d2sigs.append(outp.group(1))
                              #check for primary outputs
                        if findprimout.match(outp.group(1)):
                              d2primout.append(outp.group(1))

      # 3 Module List - determined by the endmodule
      if curblock == 3:
            modmatch = findmoduledecs.match(line)
            if modmatch:
                  if moddie == 1:
                        d1mods.append(mod)
                        mod = ""
                  if moddie == 2:
                        d2mods.append(mod)
```

```
                        mod = ""
                  if moddie == 3:
                        d1mods.append(mod)
                        d2mods.append(mod)
                        mod = ""
                  if     modmatch.group(1)     in     d1comptypes     and
modmatch.group(1) in d2comptypes:
                        moddie = 3
                  elif modmatch.group(1) in d2comptypes:
                        moddie = 2
                  else:
                        moddie = 1
            mod = mod + line

#close input file
fin.close()

fout = open('d1_'+path, 'w')
#write module declaration
fout.write("module cop_chip_d1 ( ")
#write inputs
fout.write("sleepall, x, y, clk, asm, ")
count = 0
for inp in d1primin:
      if (count % 10) == 0:
            fout.write("\n")
      count+=1
      fout.write(inp+', ')
#setup outputs for write
outputs = 'z, '
count = 0
for outp in d1primout:
      if (count % 10) == 0:
            outputs = outputs + '\n'
      count+=1
      outputs = outputs + outp + ', '
#write outputs
outputs = outputs.rstrip(', ')+');\n\n\n'
fout.write(outputs)

#end of module declaration
#These primary i/0's go in the D1 file only
d1primin.append("sleepall")
d1primin.append("[31:0]x")
d1primin.append("[31:0]y")
d1primin.append("clk")
d1primin.append("[1:0]asm")
d1primout.append("[31:0]z")
#write input list
for inp in d1primin:
      fout.write("input "+inp+";\n")
#write output list
for outp in d1primout:
      fout.write("output "+outp+";\n")
#write wire list
fout.write("\n    wire ")
#compose wires
wireout = ''
for wire in d1sigs:
      wireout = wireout + wire + ', '
fout.write(wireout.rstrip(', ')+';\n')
```

```
#place components here
for comp in d1components:
      fout.write(comp + '\n')
fout.write("\nendmodule\n")

#place component declarations here
for mod in d1mods:
      fout.write(mod + '\n')

fout.close()

#Write Die2 File
fout = open('d2_'+path, 'w')
#write module declaration
fout.write("module cop_chip_d2 ( ")
#write inputs
count = 0
for inp in d2primin:
      if (count % 10) == 0:
            fout.write("\n")
      count+=1
      fout.write(inp+', ')
#setup outputs for write
count = 0
for outp in d2primout:
      if (count % 10) == 0:
            outputs = outputs + '\n'
      count+=1
      outputs = outputs + outp + ', '
#write outputs
outputs = outputs.rstrip(', ')+');\n\n\n'
fout.write(outputs)

#end of module declaration
#write input list
for inp in d2primin:
      fout.write("input "+inp+";\n")
#write output list
for outp in d2primout:
      fout.write("output "+outp+";\n")
#write wire list
fout.write("\n    wire ")
#compose wires
wireout = ''
for wire in d2sigs:
      wireout = wireout + wire + ', '
fout.write(wireout.rstrip(', ')+';\n')

#place components here
for comp in d2components:
      fout.write(comp + '\n')
fout.write("\nendmodule\n")

#place component declarations here
for mod in d2mods:
      fout.write(mod + '\n')

fout.close()
```

APPENDIX C. BYERANDOMLINES.PY

```
# Author: Ross Thian
import re, array
filea = raw_input('Type the input file name (cop_chip_d1.v):')
cop_chip = open(filea,'r')
Replaced_o = open(filea.rstrip('\.v')+"_lines.v", 'w')
paranthesis = 0
for linecop in cop_chip:
      if (linecop.find('(') != -1):
            paranthesis = paranthesis + 1;
      if (paranthesis > 1):# inside port maping
      #print x, ' \n'#print each set
            linecop = linecop.rstrip()
            #linecop = linecop.lstrip()
            linecop = linecop.replace(';',';\n')
            pass
      linecop = linecop.replace('(',' (')
      linecop = linecop.replace('      ',' ')
      linecop = linecop.replace('     ',' ')
      linecop = linecop.replace('    ',' ')
      linecop = linecop.replace('   ',' ')
      linecop = linecop.replace('  ',' ')
      linecop = linecop.replace('  ',' ')

      Replaced_o.write(linecop) #cop_chip.seek(location,0)
print     '     White     space     cleaned     made,     outfile     =
'+filea.rstrip('\.v')+"_lines.v"
```

APPENDIX D. BPI.PY (BUFFER PRIMARY INPUTS)

```
# Author : Ross Thian
import re, array
# This program finds busses, expands them, then removes brackets from
the netlist.
#    This program also buffers inputs and outputs excluding tsvs
portname_(a_i|a_o)



def flatten(l, ltypes=(list, tuple)):
    ltype = type(l)
    l = list(l)
    i = 0
    while i < len(l):
        while isinstance(l[i], ltypes):
            if not l[i]:
                l.pop(i)
                i -= 1
                break
            else:
                l[i:i + 1] = l[i]
        i += 1
    return ltype(l)

def handleinput(item):
      NTSVInputs = []
      inputs = item.split(',')

      inputcount = 0
      #outline = "input "
      inputs[0] = inputs[0].lstrip()
      inputs[0] = inputs[0].replace("input ",'')

      for input in inputs:
            input = input.strip().rstrip(';')

            # if append non tsvs to list
            if(input.find("a_o")!= -1 or input.find("a_i")!= -1):
                  pass
            else:
                  if(input !='' ):
                        NTSVInputs.append(input)

      return NTSVInputs

def handleoutput(item):
      NTSVOutputs = []
      outputs = item.split(',')

      inputcount = 0

      outputs[0] = outputs[0].lstrip()
      outputs[0] = outputs[0].replace("output ",'')

      for output in outputs:
            output = output.rstrip().rstrip(';')

            # if append non tsvs to list
```

```
            if(output.find("a_o")!= -1 or output.find("a_i")!= -1):
                  pass
            else:
                  if(output !='' ):
                        NTSVOutputs.append(output)

      return NTSVOutputs


findmodule  = re.compile("(\s*module\s+\w+\s*\([\w\s,\n]*)")
findbusport                                                    =
re.compile("\s*(input|output|inout|tri|wire)\s+\[(\d+):(\d+)\]\s*([\\\_\
*\$/A-Za-z0-9-]+)\s*;")


findinput  = re.compile("\s*input\s+")
findoutput  = re.compile("\s*output\s+")

#findmodule = re.compile("(\s*module\s+\w+\s*\()")
a = raw_input("Netlist to Expand Busses (chip.v):")

infile = open(a,'r')


# replaces the first item with the second through the entire file


outdoc =""
modport =""
moddef = ""
inModule = "0";flatten
FileStart = "0";

inputList = []
outputList = []
#print 'infile Lines = ', len(infile) ,'\n'

for line in infile:
      #print "hi"
      if findmodule.match(line):
            FileStart ='1'
            mod = findmodule.match(line)
            print mod.group(0)
            print mod.group(1)
            moddef = moddef + line
            inModule = '1'
      elif inModule == '1' and FileStart == '1':
            if line.find(';') != -1:
                  inModule = '0'
            moddef = moddef + line

      elif findbusport.match(line) and FileStart == '1':
            m = findbusport.match(line)
            direction = m.group(1)
            start = int(m.group(3))
            end = int(m.group(2))
            port= m.group(4)
            #print  m.group(0)  ,m.group(1)  ,  m.group(2),  m.group(3),
m.group(4)
                #print "found"
            #print direction, start, end, port
```

```
            #print port


            if start < end:
                  end += 1;
                  #print 'gt'
            else:
                  temp = start
                  start = end

                  end = temp + 1;
                  #print 'lte'

            for x in range(start,end):
                  outdoc = outdoc + direction + "    " +port+'_'+str(x)
+';\n'
                  modport = modport + " "+port +'_'+ str(x) +','
                  if (direction.find('input') != -1):
                        inputList.append(port +'_'+ str(x));
                  elif (direction.find('output') != -1 ):
                        outputList.append(port +'_'+ str(x));
                  else:
                        pass
            if       (direction.find('input')       !=       -1       or
direction.find('output') != -1 or direction.find('inout') != -1):
                  print "replace: " + port
                  moddef = moddef.replace(port +',',modport)
                  moddef = moddef.replace(port +' ',modport)
                  moddef = moddef.replace(port +')',modport)

            modport = ""
            #outdoc = outdoc.replace('[','_')
            #outdoc = outdoc.replace(']','')
            inputList = flatten(inputList)
            outputList = flatten(outputList)
      elif findinput.match(line):
            inputList.append(handleinput(line))
            outdoc = outdoc + line
            while (line.find(';') == -1):
                  line = infile.next()
                  inputList.append(handleinput(line))
                  outdoc = outdoc + line
            inputList = flatten(inputList)
      elif findoutput.match(line):
            outputList.append(handleoutput(line))           #0
            outdoc = outdoc + line
            while (line.find(';') == -1):
                  line = infile.next()
                  outputList.append(handleoutput(line))
                  outdoc = outdoc + line
            outputList = flatten(outputList)
      elif FileStart == '1':
            line = line.replace(' [','_')
            line = line.replace('[','_')
            line = line.replace(']','')
            for port in inputList:
                  line = line.replace('( '+port+')','('+port+'b)')
                  line = line.replace('('+port+' )','('+port+'b)')
                  line = line.replace('( '+port+' )','('+port+'b)')
                  line = line.replace('('+port+')','('+port+'b)')
            for port in outputList:
                  line = line.replace('( '+port+')','('+port+'b)')
```

```
                  line = line.replace('('+port+' )','('+port+'b)')
                  line = line.replace('( '+port+' )','('+port+'b)')
                  line = line.replace('('+port+')','('+port+'b)')
            outdoc = outdoc + line
      else:
            pass


      #print line
# create bufferd ports

bufflist = ""


print inputList
print outputList
for port in inputList:
      bufflist = bufflist + " BUFX3 InputBuff" + port + " ( .A (" + port
+"), .Y (" +port + "b));\n"

for port in outputList:
      bufflist = bufflist + " BUFX3 OutputBuff" + port + " ( .A (" +
port +"b), .Y (" +port + "));\n"
print bufflist
# add new wires
wires = "wire "
wirecount = 0
for port in inputList:
      wires = wires  + port +"b, "
      if(wirecount >= 25):
            wirecount = 0
            wires = wires.rstrip(", ") +";\n wire "
      wirecount = wirecount + 1
for port in outputList:
      wires = wires  + port +"b, "
      if(wirecount >= 25):
            wirecount = 0
            wires = wires.rstrip(", ") +";\n wire "
      wirecount = wirecount + 1
wires = wires.rstrip("wire ")
wires = wires.rstrip(";\n wire ")
wires = wires.rstrip(", ") +";\n"
outfile = open(a.rstrip('\.v')+'_BE.v','w')

# find first wire to replace
outdoc = outdoc.replace(" wire ", wires + " wire ", 1)

# remove ending comma in module list
moddef = moddef.rstrip();
moddef = moddef.rstrip(';').rstrip(')').rstrip().rstrip(',') + ");\n"

# add bufferlist
outdoc = outdoc.rstrip().rstrip("endmodule") + bufflist + "endmodule\n"

outfile.write(moddef)
outfile.write(outdoc)
outfile.close()
print "Remember if any constant logic values are defined correct them
now:"

print "BUFX3 OutputBufflogic_0 ( .A (gnd), .Y (1'b0));"
```

```
print "BUFX3 InputBuffexpi1_0 ( .A (expi1_0), .Y (expi1_0b));"
print "change expi1_0 to vdd"
print "BUFX3 OutputBufflogic_1 ( .A (vdd), .Y (expi1_0b));"
print "change qmulbit_0 to gnd"
print " BUFX3 InputBuffqmulbit_0 ( .A (qmulbit_0), .Y (qmulbit_0b));"
# do we need to change the input here or before?
# here add the vdd and gnd
# remove the expi1_0 and qmulbit_0 from the module and ports list.

# if a verilog inserted signal is present (1'b1..) replace it with
Logic_1 or Logic_0
# The add the port "input Logic_1, Logic_0"
# Run this script
# Then replace the Logic_0 with gnd
# Before BUFX3 InputBuffLogic_0 ( .A (Logic_0), .Y (0Logic_0b));
# After  BUFX3 InputBuffLogic_0 ( .A (gnd), .Y (Logic_0b));
# do same for vdd. NOTE: AddPowandGnd.py will add vdd and gnd to
modulelist and input list
# remove the port from the ports list and module if necessary since vdd
and gnd should be defined

print ' Busses Expanded, outfile = chip_BPI.v '
#print modport
#print wires

#print moddef
```

APPENDIX E. PSTR.PY (POST SPLIT TSV REMOVE)

```
# Author : Mike Hinds
import re

#specify Filename
path = raw_input("File name (d#_cop3b_lines.v):")
# path = "cop_chip_d1.v"

#clean the input file
def handlewire(item):
      print "wire found"
      wires = item.split(',')
      wirecount = 0
      outline = "wire "
      wires[0] = wires[0].lstrip()
      wires[0] = wires[0].lstrip("wire")
      print wires[0]
      for wire in wires:
            outline = outline + wire+', '
            wirecount = wirecount + 1
            if wirecount == 50:
                  outline = outline.rstrip(', ')+';'
                  fout.write(outline+'\n\n')
                  wirecount = 0
                  outline = "wire "
      outline = outline.rstrip(', ')+';'
      fout.write(outline+"\n\n")

#perform a global replacement of the given signal
def compdecreplace(compdecs, oldsig, replacementsig):
      compdecs            =            compdecs.replace("("+oldsig+")",
"("+replacementsig+")")
      compdecs = compdecs.replace(" "+oldsig+" ", " "+replacementsig+"
")
      compdecs        =        compdecs.replace("(        "+oldsig+")",
"("+replacementsig+")")
      compdecs        =        compdecs.replace("("+oldsig+"        )",
"("+replacementsig+")")
      compdecs    =    compdecs.replace("(        "+oldsig+"        )",
"("+replacementsig+")")
      compdecs      =      compdecs.replace("(      "+oldsig+"      )",
"("+replacementsig+")")
#perform a single replacement of the given signal (no order determinacy)
def singlecompdecreplace(compdecs, oldsig, replacementsig):
      compdecs            =            compdecs.replace("("+oldsig+")",
"("+replacementsig+")",1)
      compdecs = compdecs.replace(" "+oldsig+" ", " "+replacementsig+"
",1)
      compdecs        =        compdecs.replace("(        "+oldsig+")",
"("+replacementsig+")",1)
      compdecs        =        compdecs.replace("("+oldsig+"        )",
"("+replacementsig+")",1)
      compdecs    =    compdecs.replace("(        "+oldsig+"        )",
"("+replacementsig+")",1)
      compdecs      =      compdecs.replace("(      "+oldsig+"      )",
"("+replacementsig+")",1)

findwire  = re.compile("\s+wire\s+")
fin = open(path, 'r')
```

```
fout = open("clean_bool", 'w')
lines = ''
while fin:
      line = fin.readline()
      if line == '':
            break
      #take care of comments
      if   (line.find("//")   !=   -1   or   line.find("*/")   !=   -1)   or
line.lower().find("endmodule") != -1:
            lines = lines+line.rstrip() + "\n"
      else:
            lines = lines+line.rstrip()

listlines = lines.split(';')
for item in listlines:
      if findwire.match(item):
            handlewire(item)
      else:
            fout.write(item+';\n')

fin.close()
fout.close()


#Separate Components
curblock = -1
findprimmoduledec = re.compile("module")

findwirelist = re.compile("\s*wire\s+")
findcompdecs = re.compile("\s*\S+\s+\S+\s*\(.+\)")
findcompdefs = re.compile("\s*module\s+(\S+)\s*\(.+\)")
findd1component = re.compile("\s*(\S+)\s+d1_\S+\s*\(.*\)")
findd2component = re.compile("\s*(\S+)\s+d2_\S+\s*\(.*\)")

findinput                                                                    =
re.compile("\.[a|b|c|d|sleep|nsleep|clk]\s*\(\s*(\S*)\s*\)\s*,?")
findoutput = re.compile("\.[z]\s*\(\s*(\S*?)\s*\)")

#TSV Code
findTSV = re.compile("\s*TSVd\dbuff\s+\S+\s*\((.+\))")
findTSVsigin = re.compile("\.[a|A]\s*\(\s*(\S*?)\s*\)")
findTSVsigout = re.compile("\.[y|Y|z|Z]\s*\(\s*(\S*?)\s*\)")

findprimin = re.compile(".*_a_i")
findprimout = re.compile(".*_a_o")

moduledec = ""
wirelist = ""
compdecs = ""
compdefs = ""

oldsigs = []
replacementsigs = []
duplicatesigs = []
duplicatereplacementsigs = []

#open cleaned file
fin = open("clean_bool", 'r')
while 1:
      line = fin.readline()
      if line == '':
            break
```

```
	#Determine which block we are in
	if (findprimmoduledec.match(line)) and (curblock == -1):
		curblock = 0
		print "entered primary module declaration..."
		#print line

	if (findwirelist.match(line) and curblock == 0):
		curblock = 1
		print "entered wire list..."
		#print line

	if (findcompdecs.match(line) and curblock == 1):
		curblock = 2
		print "entered component declarations..."
		#print line

	if (findcompdefs.match(line) and curblock ==2):
		curblock = 3
		print "entered component definitions..."
		#print line

	# 0 Primary Module Declaration - determined by the first "module"
declaration
	if curblock == 0:
		moduledec = moduledec + line

	# 1 Wire List - determined by the first "wire" declaration
	if curblock == 1:
		wirelist = wirelist + line

	# 2 Component Declarations - determined by the first component
declaration
	if curblock == 2:
		TSV = findTSV.match(line)
		if TSV:
			#grab input and output signals
			sigin = findTSVsigin.search(TSV.group(1))
			sigout = findTSVsigout.search(TSV.group(1))

			#determine replacement and old signals
			if findprimin.match(sigin.group(1)):
				oldsig = sigout.group(1)
				replacementsig = sigin.group(1)
			else:
				oldsig = sigin.group(1)
				replacementsig = sigout.group(1)

			#check for oldsig duplicates
			if oldsig in oldsigs:
				if oldsig in duplicatesigs:
					#we have a repeat duplicate, so add the
corresponding replacementsig to
					#the appropriate duplicatereplacementsigs
array

	duplicatereplacementsigs[duplicatesigs.index(oldsig)].append(repla
cementsig)
				else:
					#add oldsig to duplicate signal list
					duplicatesigs.append(oldsig)
```

```
                              #get oldsigs index for duplicate value
                              x = oldsigs.index(oldsig)

                              #add the two corresponding replacement
sigs to duplicatereplacementsigs
                              tempdr = []
                              tempdr.append(replacementsigs[x])
                              tempdr.append(replacementsig)
                              duplicatereplacementsigs.append(tempdr)


                        #append the signals to the signal list
                        oldsigs.append(oldsig)
                        replacementsigs.append(replacementsig)
                  else:
                        compdecs = compdecs + line
            # 3 Module List - determined by the endmodule
            if curblock == 3:
                  compdefs = compdefs + line

#close input file
fin.close()


print duplicatesigs
print duplicatereplacementsigs
print oldsigs
#process the signals and components
print len(oldsigs)
i = 0
for oldsig in oldsigs:
      print str(i) + " of " + str(len(oldsigs))
      replacementsig = replacementsigs[i]

      #remove old signals from the wirelist
      wirelist = wirelist.replace(" "+oldsig+",", "")
      wirelist = wirelist.replace(","+oldsig+";", ";")
      wirelist = wirelist.replace(", "+oldsig+";", ";")

      if oldsig == "buffwire0_0enc1_2":
            print oldsig + " found!!!"
      #replace old signals
      compdecs             =            compdecs.replace("("+oldsig+")",
"("+replacementsig+")")
      compdecs = compdecs.replace(" "+oldsig+" ", " "+replacementsig+"
")
      compdecs        =        compdecs.replace("(        "+oldsig+")",
"("+replacementsig+")")
      compdecs        =        compdecs.replace("("+oldsig+"        )",
"("+replacementsig+")")
      compdecs    =    compdecs.replace("(        "+oldsig+"        )",
"("+replacementsig+")")
      compdecs      =      compdecs.replace("(      "+oldsig+"      )",
"("+replacementsig+")")
      #print   "looking   for   "+oldsig+"   and   replacing   it   with
"+replacementsig

      i+=1
wirelist = wirelist.replace("wire ;", "")
wirelist = wirelist.replace("wire;", "")
wirelist = wirelist.replace("wire  ;", "")
wirelist = wirelist.replace("wire   ;", "")
```

```
wirelist = wirelist.replace("wire    ;", "")
wirelist = wirelist.replace("wire     ;", "")

#Write output file
fout = open(path.rstrip(".v")+"_TSVrm.v", 'w')
for line in moduledec:
      fout.write(line)
for line in wirelist:
      fout.write(line)
for line in compdecs:
      fout.write(line)
for line in compdefs:
      fout.write(line)
fout.close()
```

APPENDIX F. DR.PY (DUPLICATES REMOVAL)

```
# Author : Ross Thian
# This program finds and replaces nets that are duplicated after tsv
removal
# When a tsv is removed several connection can actually be connected to
the same net
# We combine these nets into one.



import re, array
a = raw_input("Duplicates File name? (duplicates.txt) : ")
duplicates = open(a,'r')
b = raw_input("Post TSV Remove file name (d1_cop3b_TSVrm) : ")
cop_chip = open(b,'r')
DuplicatesRemoved = open(b.rstrip('\.v')+"_ntsv.v",'w')

line = duplicates.readline()
# gets to valid arrary of duplicates
while (line.find("entered component declarations...") == -1):
      line = duplicates.readline()
line = duplicates.readline()
      ## 5th list of duplicates
line = duplicates.readline()


#for copline in cop_chip # sDuplicateswitch a_o with a_i
line = line.replace('a_o','a_*')
line = line.replace('a_i','a_o')
line = line.replace('a_*','a_i')

#print 'cop_chip Lines = ', len(cop_chip) ,'\n'
replacements = eval(line)
endofwire = -1
for linecop in cop_chip:
      if((endofwire == -1) & (linecop.find('wire ') != -1)):
            for linecop in cop_chip:
                  print linecop
                  if (linecop.find('wire ') == -1):
                        location = cop_chip.tell()
                        print location
                        endofwire = 1
                        break

      if(endofwire == 1):
            for x in replacements:
                  #print x, ' \n'#print each set
                  for j in range(1,len(x)):
                        #print x[j] , ' ' ,x[0] , ' \n'
                        if (linecop.find(x[j]) != -1):
                              linecop = linecop.replace(x[j],x[0])
                              #print x[j] , ' ' ,x[0] , ' \n'
      DuplicatesRemoved.write(linecop) #cop_chip.seek(location,0)
print ' Cleaned duplicates, outfile = cop_chip_d2_ntsv.v '



##line = duplicates.readline()
##print line +'\n'
```

APPENDIX G. REPLACESIGS.PY

```
# Author : Ross Thian
import re, array
a = raw_input('netlist.v :   ')
sigs = open('sigs.txt','r')
cop_chip = open(a,'r')
Replaced_o = open(a.rstrip('\.v') +'r.v','w')

# replaces the first item with the second through the entire file

## 5th list of duplicates
line = sigs.readline()


#print 'cop_chip Lines = ', len(cop_chip) ,'\n'
replacements = eval(line)
for linecop in cop_chip:
      for x in replacements:
            #print x, ' \n'#print each set
            linecop = linecop.replace(x[0],x[1])
                        #print x[j] , ' ' ,x[0] , ' \n'
      Replaced_o.write(linecop) #cop_chip.seek(location,0)
print ' Replacements made, outfile = '+ a.rstrip('\.v') +'r.v'
```

APPENDIX H. EXAMPLE SIGS.TXT

```
[[' dff d1_reg0_',' DFFX4 d1_reg0_'],[' dff d2_reg0_',' DFFX4
d2_reg0_'],[' dff d1_reg16_',' DFF_MTCMOS d1_reg16_'],[' dff ',' DFF_Em
'],[' and2 ',' AND2_Bm '],[' and3 ',' AND3_Bm '],[' and4 ',' AND4_Bm
'],[' nand2 ',' NAND2_Am '],[' nand3 ',' NAND3_Am '],[' nor2 ',' NOR2_Am
'],[' or2 ',' OR2_Bm '],[' or3 ',' OR3_Bm '],[' or4 ',' OR4_Bm '],['
xnor2 ',' XNOR2_Am '],[' nxor2 ',' XNOR2_Am '],[' xor2 ',' XOR2_Am '],['
invs ',' INVX1 '],[' inv ',' INVERT_Bm
'],['.a','.A'],['.b','.B'],['.c','.C'],['.d','.D'],['.sleep','.SLEEP'],[
'.nsleep','.NSLEEP'],['.q','.Q'],['.nq','.QN'],['.qn','.QN'],['.Qn','.QN
'],['.z','.Y'],['.y','.Y'],['.Z','.Y'],['.clk','.CK'],['.Clk','.CK']]
```

APPENDIX I. ADDPOWANDGND.PY

```
# Author Brent Hollosi Modified by Ross Thian
def AddPorts(VerIn, VerOut):
        import re
      count=0
      add = 0
      first = 0
      p = 0
      Fin = open(VerIn, 'r')
      Fout= open(VerOut, 'w')
      FirstInput=False
      InModule=False
      while 1:
            i,l=0,0

            line = Fin.readline()
            if not line:
                  break
            l=len(line)
            if(line[i:i+6]=="module"):
                  add = 0
                  first = 0
            while i < l:
                  if(line[i] == '('):
                        if(p == 0 and first == 1):
                              line=line[:i+1]+" .VDD(vdd), .VSS(gnd),
"+line[i+1:]
                              l = l+23
                        if(p == 0 and first == 0):
                              line = line[:i+1]+" vdd, gnd,"+line[i+1:]
                              l = l + 10
                              first = 1
                        p = p + 1
                  if(line[i] == ')'):
                        p = p - 1
                  if(line[i:i+5]=="input" and add != 1):
                        Fout.write("  inout vdd, gnd;")
                        add = 1
                  i = i + 1


            count=count+1
            Fout.write(line)
      Fin.close()
      Fout.close()


a=raw_input("Input File: ")
b=raw_input("Outpt File: ")
AddPorts(a, b)
```

APPENDIX J. CADENCEREADY.PY

```
# Author Ross Thian
import re
def handleinout(item):

      inouts = item.split(',')
      #inoutcount = 0
      outline = ""
      inouts[0] = inouts[0].lstrip()
      inouts[0] = inouts[0].lstrip("inout")

      for inout in inouts:
       #outline = outline + inout+', '
       outline = outline +"inout "+inout+';\n'
       #inoutcount = inoutcount + 1
       #if inoutcount == 50:
       #     outline = outline.rstrip(', ')+';'
       #     fout.write(outline+'\n\n')
       #     inoutcount = 0
       #     outline = "inout "
      #outline = outline.rstrip(', ')+';'
      fout.write(outline+"\n")
def handlemodule(item):

      modules = item.split(',')
      modulecount = 0
      outline = "module"
      modules[0] = modules[0].lstrip()
      modules[0] = modules[0].lstrip("module")

      for module in modules:
       outline = outline + module+', \n'
       #modulecount = modulecount + 1
       #if modulecount == 50:
       #     outline = outline.rstrip(', ')+';'
       #     fout.write(outline+'\n\n')
       #     modulecount = 0
       #     outline = "module "
      outline = outline.rstrip(', \n')+';'
      fout.write(outline+"\n\n")
def handleoutput(item):

      outputs = item.split(',')
      #outputcount = 0
      outline = ""
      outputs[0] = outputs[0].lstrip()
      outputs[0] = outputs[0].lstrip("output")

      for output in outputs:
       #outline = outline + output+', '
       outline = outline + "output " +output+';\n'
       #outputcount = outputcount + 1
       #if outputcount == 50:
       #     outline = outline.rstrip(', ')+';'
       #     fout.write(outline+'\n\n')
       #     outputcount = 0
       #     outline = "output "
      #outline = outline.rstrip(', ')+';'
      fout.write(outline+"\n")
```

```
def handleinput(item):

      inputs = item.split(',')
      #inputcount = 0
      outline = ""
      inputs[0] = inputs[0].lstrip()
      inputs[0] = inputs[0].lstrip("input")

      for input in inputs:
            #outline = outline + input+', '
            outline = outline +"input "+ input+';\n'
            #inputcount = inputcount + 1
            #if inputcount == 50:
            #     outline = outline.rstrip(', ')+';'
            #     fout.write(outline+'\n\n')
            #     inputcount = 0
            #     outline = "input "
      #outline = outline.rstrip(', ')+';'
      fout.write(outline+"\n")
def handlewire(item):

      wires = item.split(',')
      wirecount = 0
      outline = "wire "
      wires[0] = wires[0].lstrip()
      wires[0] = wires[0].lstrip("wire")

      for wire in wires:
            outline = outline + wire+', '
            wirecount = wirecount + 1
            if wirecount == 50:
                  outline = outline.rstrip(', ')+';'
                  fout.write(outline+'\n\n')
                  wirecount = 0
                  outline = "wire "
      outline = outline.rstrip(', ')+';'
      fout.write(outline+"\n\n")
      #fout.write("//endofwire\n\n")

fanoutlist = []

a=raw_input("Create Cadence Clean File from (abcd.v): ")
findwire  = re.compile("\s+wire\s+")
findinout  = re.compile("\s*inout\s+")
findinput  = re.compile("\s*input\s+")
findoutput = re.compile("\s*output\s+")
findmodule = re.compile("module\s+")
findendmodule = re.compile("endmodule")
#findinput =
print "Cadence clean netlist in "+a.rstrip('\.v')+"_Cln.v"
fin = open(a, 'r')

fout = open(a.rstrip('\.v')+'_Cln.v', 'w')

lines = ''
while fin:
      line = fin.readline()
      if line == '':
            break
      lines = lines+line.rstrip()

listlines = lines.split(';')
```

```
for item in listlines:
      item = item.replace("//endofwire",'')
      if findwire.match(item):
            handlewire(item)
      elif findinout.match(item):
            handleinout(item)
      elif findinput.match(item):
            handleinput(item)
      elif findoutput.match(item):
            handleoutput(item)
      elif findmodule.match(item):
            handlemodule(item)
      elif findendmodule.match(item):
            fout.write("endmodule")
      else:
            fout.write(item+';\n')

fin.close()
fout.close()
```

APPENDIX K. TBES.PY (TOP BUS EXPANSION SCRIPT)

```
# Author Ross Thian
# This program works on the toplevel netlist.
# It expands busses and replaces them in the module def and component
def.

import re, array

findport  =
re.compile("\s*(input|output|inout|tri|wire)\s+\[(\d+):(\d+)\]\s*([\\\_\
*\$/A-Za-z0-9-]+)\s*;")
findmodule  = re.compile("(\s*module\s+\w+\s*\([\w\s,\n]*)")
#findmodule = re.compile("(\s*module\s+\w+\s*\()")
a = raw_input("Netlist to Expand Busses (chip.v):")

infile = open(a,'r')


# replaces the first item with the second through the entire file


outdoc =""
modport =""
moddef = ""
compports = []
inModule = "0";
FileStart = "0";
#print 'infile Lines = ', len(infile) ,'\n'

for line in infile:
      #print "hi"
      if findmodule.match(line):
            FileStart ='1'
            mod = findmodule.match(line)
            #print mod.group(0)
            #print mod.group(1)
            moddef = moddef + line
            inModule = '1'
            if line.find(';') != -1:
                  inModule = '0'
      elif inModule == '1' and FileStart == '1':
            if line.find(';') != -1:
                  inModule = '0'
            moddef = moddef + line

      elif findport.match(line) and FileStart == '1':
            m = findport.match(line)
            direction = m.group(1)
            start = int(m.group(3))
            end = int(m.group(2))
            port= m.group(4)
            #print m.group(0) ,m.group(1) , m.group(2), m.group(3),
m.group(4)
                #print "found"
            #print direction, start, end, port
            #print port


            if start < end:
```

```
                  end += 1;
                  #print 'gt'
            else:
                  temp = start
                  start = end

                  end = temp + 1;
                  #print 'lte'
            compport = ""
            portreplace = []
            for x in range(start,end):
                  outdoc = outdoc +"  " +direction + "   "
+port+'_'+str(x) +';\n'
                  modport = modport + " "+port +'_'+ str(x) +','
                  compport = compport + '.'+ port +'_' +str(x)+ '(' +
port +'_' +str(x)+ '), '
            if (direction.find('input') != -1 or
direction.find('output') != -1 or direction.find('inout') != -1):
                  print "replace: " + port
                  moddef = moddef.replace(port +',',modport)
                  moddef = moddef.replace(port +' ',modport)
                  moddef = moddef.replace(port +')',modport)

            modport = ""
            portreplace.append('.' + port +'('+port+'),')
            portreplace.append(compport)
            compports.append(portreplace)
            #print portreplace
            #outdoc = outdoc.replace('[','_')
            #outdoc = outdoc.replace(']','')
      elif FileStart == '1':
            line = line.replace(' [','_')
            line = line.replace('[','_')
            line = line.replace(']','')
            outdoc = outdoc + line
      else:
            pass
      #print line
outfile = open(a.rstrip('\.v')+'_BE.v','w')
#print moddef;
#moddef = moddef.rstrip(';').rstrip(')').rstrip(',') + ");\n"
moddef = moddef.rstrip();
moddef = moddef.rstrip(';').rstrip(')').rstrip().rstrip(',') + ");\n"
outfile.write(moddef)

for port in compports:
      outdoc = outdoc.replace(port[0],port[1])
      #print port[0]
      #print port[1]
outfile.write(outdoc)
outfile.close()
print ' Busses Expanded, outfile = ' + a.rstrip('\.v')+'_BE.v'
#print modport
#print moddef
```

APPENDIX L.UPR.PY (UNUSED PIN REMOVAL)

```
# Author Ross Thian
import re

AllInout = []
AllInput = []
AllOutput = []
AllWire = []
d1compdecs = ""
d1wirelist = ""
d1modulelist = ""
d1portlist = ""
d2compdecs = ""
d2wirelist = ""
d2modulelist = ""
d2portlist = ""

def toTSVonly(item):
      onlyTSV = []
      for port in item:
            if port.find("_a_") != -1:
                  onlyTSV.append(port)
      return onlyTSV

def flatten(l, ltypes=(list, tuple)):
    ltype = type(l)
    l = list(l)
    i = 0
    while i < len(l):
        while isinstance(l[i], ltypes):
            if not l[i]:
                l.pop(i)
                i -= 1
                break
            else:
                l[i:i + 1] = l[i]
        i += 1
    return ltype(l)

def handleoutput(item):

      outputs = item.split(',')
      #AllOutput.append(outputs)
      outputcount = 0
      outline = "output "
      outputs[0] = outputs[0].lstrip()
      outputs[0] = outputs[0].replace("output ",'')

      for output in outputs:
            output = output.strip().rstrip(';')
            AllOutput.append(output)
            outline = outline + output+', '
            outputcount = outputcount + 1
            if outputcount == 50:
                  outline = outline.rstrip(', ')+';'
                  outline = outline+'\n\n'
                  outputcount = 0
                  outline = outline +"output "
      outline = outline.rstrip(', ')+';'
```

```
	outline = outline.replace('output;','')
	return outline + "\n"
def handleinout(item):

	inouts = item.split(',')
	#AllInout.append(inouts)

	inoutcount = 0
	outline = "inout "
	inouts[0] = inouts[0].lstrip()
	inouts[0] = inouts[0].replace("inout ",'')
	#print inouts[0]

	for inout in inouts:
		inout = inout.strip().rstrip(';')
		AllInout.append(inout)
		outline = outline + inout+', '
		inoutcount = inoutcount + 1
		if inoutcount == 50:
			outline = outline.rstrip(', ')+';'
			outline = outline+'\n\n'
			inoutcount = 0
			outline = outline +"inout "
	outline = outline.rstrip(', ')+';'
	outline = outline.replace('inout;','')
	return outline + "\n"
def handlemodule(item):

	modules = item.split(',')
	modulecount = 0
	outline = "module"
	modules[0] = modules[0].lstrip()
	modules[0] = modules[0].replace("module",'')

	for module in modules:
		outline = outline + module+', \n'
	outline = outline.rstrip(', \n')+';'
	return outline + "\n"

def handleinput(item):

	inputs = item.split(',')
	#AllInput.append(inputs)
	inputcount = 0
	outline = "input "
	inputs[0] = inputs[0].lstrip()
	inputs[0] = inputs[0].replace("input ",'')

	for input in inputs:
		input = input.strip().rstrip(';')

		AllInput.append(input)
		outline = outline + input+', '
		inputcount = inputcount + 1
		if inputcount == 50:
			outline = outline.rstrip(', ')+';'
			outline = outline+'\n\n'
			inputcount = 0
			outline = outline + "input "
	outline = outline.rstrip(', ')+';'
	outline = outline.replace('input;','')
```

```
      return outline + "\n"
def handlewire(item):

      wires = item.split(',')
      #AllWire.append(wires)
      wirecount = 0
      outline = "wire "
      wires[0] = wires[0].lstrip()
      wires[0] = wires[0].replace("wire ",'')

      for wire in wires:
            wire = wire.strip().rstrip(';')
            AllWire.append(wire)
            outline = outline + wire+', '
            wirecount = wirecount + 1
            if wirecount == 50:
                  outline = outline.rstrip(', ')+';'
                  outline = outline+'\n\n'
                  wirecount = 0
                  outline = outline + "wire "

      outline = outline.rstrip(', ')+';'
      outline = outline.replace("wire;",'')
      return outline + "\n\n"
      #fout.write("//endofwire\n\n")

def handlewirelist(wires):
      wirecount = 0
      #AllWire.append(wires)
      outline = "wire "

      for wire in wires:
            wire = wire.lstrip().rstrip()
            AllWire.append(wire)
            outline = outline + wire+', '
            wirecount = wirecount + 1
            if wirecount == 50:
                  outline = outline.rstrip(', ')+';'
                  outline = outline+'\n\n'
                  wirecount = 0
                  outline = outline + "wire "
#     if wirecount != 0:
      outline = outline.rstrip(', ')+';'
#     else:
      outline = outline.replace("wire;",'')
      return outline + "\n\n"
      #fout.write("//endofwire\n\n")

d1=raw_input("Removed Unused Ports from (cop_chip_d1.v): ")
d2=raw_input("Removed Unused Ports from (cop_chip_d2.v): ")
top=raw_input("Removed Unused Porprint d1portlistts from top Level
(cop_chip_top.v): ")
findwire  = re.compile("\s+wire\s+")
findinout  = re.compile("\s*inout\s+")
findinput  = re.compile("\s*input\s+")
findoutput = re.compile("\s*output\s+")
findmodule = re.compile("\s*module\s+")
findendmodule = re.compile("endmodule")

#findinput =
```

```
D1in = open(d1,'r')
D2in = open(d2,'r')



#fin = open(a, 'r')
#fout = open(a.rstrip('\.v')+'_rmport.v', 'w')
#f2out = open(a.rstrip('\.v')+'_prt2.v', 'w')

#D1
lines = ''
while D1in:
      line = D1in.readline()
      if line == '':
            break
      lines = lines+' '+(line.rstrip()+' ').lstrip()
      lines = lines.replace('( ','(')


listlines = lines.split(';')
D1in.close()


CreateReplaceList = -1

PrimaryPorts = []
PrimaryInputs = []
PrimaryOutputs = []
for item in listlines:
      item = item.replace("//endofwire",'')
      if findwire.match(item):
            d1wirelist = d1wirelist + handlewire(item)
      elif findinout.match(item):
            d1portlist = d1portlist + handleinout(item)
      elif findinput.match(item):
            d1portlist = d1portlist + handleinput(item)
      elif findoutput.match(item):
            d1portlist = d1portlist + handleoutput(item)
      elif findmodule.match(item):
            d1modulelist = d1modulelist + handlemodule(item)
      elif findendmodule.match(item):
            print "endmodule"
      else:
            d1compdecs = d1compdecs + item+';\n'

pinsoutD1 = open(d1.rstrip('\.v')+'pins.v', 'w')
pinoutD1 = []
d1AllInput = []
d1AllInout = []
d1AllOutput = []


AllInput = flatten(AllInput)
AllOutput = flatten(AllOutput)
AllInout = flatten(AllInout)
AllWire = flatten(AllWire)
#print AllWire
for Input in AllInput:
      if(d1compdecs.find('('+Input.lstrip('input').lstrip().rstrip()+')'
) == -1):
```

```
            Input = Input.lstrip('input').lstrip().rstrip()
            findinput1  = re.compile("\s*"+Input+"\s*,")
            d1portlist = findinput1.sub('',d1portlist)
            findinput2  = re.compile(",\s*"+Input+"\s*;")
            d1portlist = findinput2.sub(';',d1portlist)
            findinput3  = re.compile("input\s*"+Input+"\s*;")
            d1portlist = findinput3.sub('',d1portlist)
            re.compile("\s*"+Input+"\s*,")

            d1modulelist = findinput1.sub('',d1modulelist)
            findinput4 = re.compile(",\s*"+Input+"\s*\)")
            d1modulelist = findinput4.sub(' )',d1modulelist)
            pinoutD1.append(Input.lstrip('input').lstrip().rstrip())
            #pinsout.write(Input.lstrip('input').lstrip().rstrip()+'
Input Removed\n')
      else:
            d1AllInput.append(Input)
#print AllInput
#print d1portlist


for Inout in AllInout:
      if(d1compdecs.find(Inout.lstrip('inout').lstrip())== -1):
            Inout = Inout.lstrip('inout').lstrip().rstrip()
            findinout1  = re.compile("\s*"+Inout+"\s*,")
            d1portlist = findinout1.sub('',d1portlist)
            findinout2  = re.compile(",\s*"+Inout+"\s*;")
            d1portlist = findinout2.sub(';',d1portlist)
            findinout3  = re.compile("inout\s*"+Inout+"\s*;")
            d1portlist = findinout3.sub('',d1portlist)

            d1modulelist = findinout1.sub('',d1modulelist)
            findinout4 = re.compile(",\s*"+Inout+"\s*\)")
            d1modulelist = findinout4.sub(' )',d1modulelist)

            pinoutD1.append(Inout.lstrip('inout').lstrip().rstrip())
            #pinsout.write(Inout.lstrip('inout').lstrip().rstrip()+'
Inout Removed\n')
      else:
            d1AllInout.append(Inout)
#print d1portlist
for Output in AllOutput:
      if(d1compdecs.find('('+Output.lstrip('output').lstrip().rstrip()+'
)')== -1):
            Output = Output.lstrip('output').lstrip().rstrip()
            findoutput1  = re.compile("\s*"+Output+"\s*,")
            d1portlist = findoutput1.sub('',d1portlist)
            findoutput2  = re.compile(",\s*"+Output+"\s*;")
            d1portlist = findoutput2.sub(';',d1portlist)
            findoutput3  = re.compile("output\s*"+Output+"\s*;")
            d1portlist = findoutput3.sub('',d1portlist)

            d1modulelist = findoutput1.sub('',d1modulelist)
            findoutput4 = re.compile(",\s*"+Output+"\s*\)")
            d1modulelist = findoutput4.sub(' )',d1modulelist)
            pinoutD1.append(Output.lstrip('output').lstrip().rstrip())
            #pinsout.write(Output.lstrip('output').lstrip().rstrip()+'
Output Removed\n')
      else:
            d1AllOutput.append(Output)
#print d1portlist
findendmodulerep = re.compile('endmodule ;')
```

```
d1compdecs = findendmodulerep.sub('endmodule',d1compdecs)




AllInout = []
AllInput = []
AllOutput = []
AllWire = []

#D2
lines = ''
while D2in:
      line = D2in.readline()
      if line == '':
            break
      lines = lines+' '+(line.rstrip()+' ').lstrip()
      lines = lines.replace('( ','(')


listlines = lines.split(';')
D2in.close()


CreateReplaceList = -1

PrimaryPorts = []
PrimaryInputs = []
PrimaryOutputs = []
for item in listlines:
      item = item.replace("//endofwire",'')
      if findwire.match(item):
            d2wirelist = d2wirelist + handlewire(item)
      elif findinout.match(item):
            d2portlist = d2portlist + handleinout(item)
      elif findinput.match(item):
            d2portlist = d2portlist + handleinput(item)
      elif findoutput.match(item):
            d2portlist = d2portlist + handleoutput(item)
      elif findmodule.match(item):
            d2modulelist = d2modulelist + handlemodule(item)
      elif findendmodule.match(item):
            print "endmodule"
      else:
            d2compdecs = d2compdecs + item+';\n'

pinsoutD2 = open(d2.rstrip('\.v')+'pins.v', 'w')
pinoutD2 = []

AllInput = flatten(AllInput)
AllOutput = flatten(AllOutput)
AllInout = flatten(AllInout)
AllWire = flatten(AllWire)

d2AllInput = []
d2AllInout = []
d2AllOutput = []
#print AllWire
```

```
for Input in AllInput:
      if(d2compdecs.find('('+Input.lstrip('input').lstrip().rstrip()+')'
) == -1):
            Input = Input.lstrip('input').lstrip().rstrip()
            findinput1  = re.compile("\s*"+Input+"\s*,")
            d2portlist = findinput1.sub('',d2portlist)
            findinput2  = re.compile(",\s*"+Input+"\s*;")
            d2portlist = findinput2.sub(';',d2portlist)
            findinput3  = re.compile("input\s*"+Input+"\s*;")
            d2portlist = findinput3.sub('',d2portlist)
            re.compile("\s*"+Input+"\s*,")

            d2modulelist = findinput1.sub('',d2modulelist)
            findinput4 = re.compile(",\s*"+Input+"\s*\)")
            d2modulelist = findinput4.sub(' )',d2modulelist)
            pinoutD2.append(Input.lstrip('input').lstrip().rstrip())
            #pinsout.write(Input.lstrip('input').lstrip().rstrip()+'
Input Removed\n')
      else:
            d2AllInput.append(Input)
for Inout in AllInout:
      if(d2compdecs.find(Inout.lstrip('inout').lstrip())== -1):
            Inout = Inout.lstrip('inout').lstrip().rstrip()
            findinout1  = re.compile("\s*"+Inout+"\s*,")
            d2portlist = findinout1.sub('',d2portlist)
            findinout2  = re.compile(",\s*"+Inout+"\s*;")
            d2portlist = findinout2.sub(';',d2portlist)
            findinout3  = re.compile("inout\s*"+Inout+"\s*;")
            d2portlist = findinout3.sub('',d2portlist)

            d2modulelist = findinout1.sub('',d2modulelist)
            findinout4 = re.compile(",\s*"+Inout+"\s*\)")
            d2modulelist = findinout4.sub(' )',d2modulelist)
            pintoutD2.append(Inout.lstrip('inout').lstrip().rstrip())
            #pinsout.write(Inout.lstrip('inout').lstrip().rstrip()+'
Inout Removed\n')
      else:
            d2AllInout.append(Inout)
for Output in AllOutput:
      if(d2compdecs.find('('+Output.lstrip('output').lstrip().rstrip()+'
)')== -1):
            Output = Output.lstrip('output').lstrip().rstrip()
            findoutput1  = re.compile("\s*"+Output+"\s*,")
            d2portlist = findoutput1.sub('',d2portlist)
            findoutput2  = re.compile(",\s*"+Output+"\s*;")
            d2portlist = findoutput2.sub(';',d2portlist)
            findoutput3  = re.compile("output\s*"+Output+"\s*;")
            d2portlist = findoutput3.sub('',d2portlist)

            d2modulelist = findoutput1.sub('',d2modulelist)
            findoutput4 = re.compile(",\s*"+Output+"\s*\)")
            d2modulelist = findoutput4.sub(' )',d2modulelist)
            pinoutD2.append(Output.lstrip('output').lstrip().rstrip())
            #pinsout.write(Output.lstrip('output').lstrip().rstrip()+'
Output Removed\n')
      else:
            d2AllOutput.append(Output)
findendmodulerep = re.compile('endmodule ;')
d2compdecs = findendmodulerep.sub('endmodule',d2compdecs)
```

```
D1out  = open(d1.rstrip('\.v')+'_rmport.v', 'w')
D2out = open(d2.rstrip('\.v')+'_rmport.v', 'w')


d1PortsOut = open('d1_tsv\'s','w')
d2PortsOut = open('d2_tsv\'s','w')

#For brents tsv macro
d1OutPorts = []
d1AllInput = toTSVonly(d1AllInput)
#print d1AllInput
d1OutPorts.append(d1AllInput)
#print 'd1OutPorts'
#print d1OutPorts
d1AllOutput = toTSVonly(d1AllOutput)
d1OutPorts.append(d1AllOutput)


d2OutPorts = []
d2AllInput = toTSVonly(d2AllInput)
d2OutPorts.append(d2AllInput)
d2AllOutput = toTSVonly(d2AllOutput)
d2OutPorts.append(d2AllOutput)



ReplaceToWireD1 = []
ReplaceToWireD2 = []

strd2OutPorts = str(flatten(d2OutPorts))
strd2OutPorts = strd2OutPorts.replace('a_o','a_*')
strd2OutPorts = strd2OutPorts.replace('a_i','a_o')
strd2OutPorts = strd2OutPorts.replace('a_*','a_i')

for pin in flatten(d1OutPorts):
      if (strd2OutPorts.find(pin) == -1):
            ReplaceToWireD1.append(pin)
            pinoutD1.append(pin)

strd1OutPorts = str(flatten(d1OutPorts))
strd1OutPorts = strd1OutPorts.replace('a_o','a_*')
strd1OutPorts = strd1OutPorts.replace('a_i','a_o')
strd1OutPorts = strd1OutPorts.replace('a_*','a_i')

for pin in flatten(d2OutPorts):
      if (strd1OutPorts.find(pin) == -1):
                  ReplaceToWireD2.append(pin)
            pinoutD2.append(pin)
d1wirelist = d1wirelist + handlewirelist(ReplaceToWireD1)

# inpu problem
for port in ReplaceToWireD1:
      port = port.lstrip().rstrip()
      findport1  = re.compile("\s+"+port+"\s*,")
      d1portlist = findport1.sub('',d1portlist)
```

```
      findport2  = re.compile(",\s+"+port+"\s*;")
      d1portlist = findport2.sub(';',d1portlist)
      findport3  = re.compile("(input|output|inout)\s+"+port+"\s*;")
      d1portlist = findport3.sub('',d1portlist)
      #re.compile(",\s*"+port+"\s*,")

      d1modulelist = findport1.sub('',d1modulelist)
      findport4 = re.compile(",\s*"+port+"\s*\)")
      d1modulelist = findport4.sub(' )',d1modulelist)
# inpu problem
#print d1portlist
print ReplaceToWireD1
d1FinalPorts = []
for port in d1OutPorts:
      keep = -1
      for wire in ReplaceToWireD1:
            if wire.find(str(port)) != -1:
                  keep == 1
      if keep == -1:
            d1FinalPorts.append(port)



#d1PortsOut.write(str(d1OutPorts))

d2wirelist = d2wirelist + handlewirelist(ReplaceToWireD2)

for port in ReplaceToWireD2:
      port = port.lstrip().rstrip()
      findport1  = re.compile("\s*"+port+"\s*,")
      d2portlist = findport1.sub('',d2portlist)
      findport2  = re.compile(",\s*"+port+"\s*;")
      d2portlist = findport2.sub(';',d2portlist)
      findport3  = re.compile("(input|output|inout)\s+"+port+"\s*;")
      d2portlist = findport3.sub('',d2portlist)
      #re.compile("\s*"+port+"\s*,")

      d2modulelist = findport1.sub('',d2modulelist)
      findport4 = re.compile(",\s*"+port+"\s*\)")
      d2modulelist = findport4.sub(' )',d2modulelist)

d2FinalPorts = []
for port in d2OutPorts:
      keep = -1
      for wire in ReplaceToWireD2:
            if wire.find(str(port)) != -1:
                  keep == 1
      if keep == -1:
            d2FinalPorts.append(port)


pinsoutD1.write(str(pinoutD1))
pinsoutD1.close()
pinsoutD2.write(str(pinoutD2))
pinsoutD2.close()
d2PortsOut.write(str(d2FinalPorts))

D2out.write(d2modulelist)
#print portlist
D2out.write(d2portlist)
#print wirelist
D2out.write(d2wirelist)
```

```
#print compdecs
D2out.write(d2compdecs)



d1PortsOut.write(str(d1FinalPorts))

D1out.write(d1modulelist)
#print portlist
D1out.write(d1portlist)
#print wirelist
D1out.write(d1wirelist)
#print compdecs
D1out.write(d1compdecs)
#possibly All inputs ports on d1 are wire names to d2
#possibly all ports to d2 are opposing pins to d1
#in theory we could remove all pins and wires from d1's list
#remove all pins from d2's list

Topin = open(top,'r')
Topout = open(top.rstrip('\.v')+'_rmport.v', 'w')
TopoutFile = ""

print "Replace Top Level Ports"
for line in Topin:
      TopoutFile = TopoutFile + line

for port1 in pinoutD1:
      strport1 = str(port1)
#     findFinalport1  =
re.compile("\."+strport1+'\('+"\s*"+strport1+"\),")
#     findFinalport2  =
re.compile(",\s*\."+strport1+'\('+"\s*"+strport1+"\)"+"\s*"+"\)")
#     findFinalport3  =
re.compile("\(\s*\."+strport1+'\('+"\s*"+strport1+"\)"+"\s*"+"\)")
#     findFinalport4  = re.compile(strport1+",")
#     TopoutFile = findFinalport1.sub('',TopoutFile)
#     TopoutFile = findFinalport2.sub(' )',TopoutFile)
#     TopoutFile = findFinalport3.sub('()',TopoutFile)
#     TopoutFile = findFinalport4.sub('',TopoutFile)

      findFinalport1  =
re.compile(",\s*\."+strport1+'\('+"\s*"+strport1+"\),")
      TopoutFile = findFinalport1.sub(',',TopoutFile)
      findFinalport2  =
re.compile("\."+strport1+'\('+"\s*"+strport1+"\),")
      TopoutFile = findFinalport2.sub('',TopoutFile)
      findFinalport3  =
re.compile(",\s*\."+strport1+'\('+"\s*"+strport1+"\)"+"\s*"+"\)")
      TopoutFile = findFinalport3.sub(' )',TopoutFile)
      findFinalport4  =
re.compile("\(\s*\."+strport1+'\('+"\s*"+strport1+"\)"+"\s*"+"\)")
      TopoutFile = findFinalport4.sub('()',TopoutFile)

      findFinalwire1  = re.compile(",\s*"+strport1+",")
      TopoutFile = findFinalwire1.sub(',',TopoutFile)
      findFinalwire2  = re.compile("wire\s*"+strport1+",")
      TopoutFile = findFinalwire2.sub('wire ',TopoutFile)
      findFinalwire3  = re.compile(",\s*"+strport1+";")
      TopoutFile = findFinalwire3.sub(';',TopoutFile)

#for port1 in pinoutD1:
```

```
#     print port1
#     #strport1 = str(port1)
#     TopoutFile = TopoutFile.replace(', \n        .'+port1+'(','')
#     TopoutFile = TopoutFile.replace(port1+'),',' \n')
#     TopoutFile = TopoutFile.replace('.'+port1+'(','')
#     TopoutFile = TopoutFile.replace(port1+') )',')')

#           findFinalport1  =
re.compile("\s*\."+strport1+'\('+"\s*"+strport1+"\),")
#           findFinalport2  =
re.compile(",\s*\."+strport1+'\('+"\s*"+strport1+"\)"+"\s*"+"\)")
#           findFinalport3  =
re.compile("\(\s*\."+strport1+'\('+"\s*"+strport1+"\)"+"\s*"+"\)")
#           TopoutFile = findFinalport1.sub('',TopoutFile)
#           TopoutFile = findFinalport2.sub(')',TopoutFile)
#           TopoutFile = findFinalport3.sub('()',TopoutFile)

for port2 in pinoutD2:
      strport2 = str(port2)
      strport3 = strport2
      strport3 = strport3.replace('a_o','a_*')
      strport3 = strport3.replace('a_i','a_o')
      strport3 = strport3.replace('a_*','a_i')

      findFinalport6  =
re.compile(",\s*\."+strport2+'\('+"\s*"+strport3+"\),")
      TopoutFile = findFinalport6.sub(',',TopoutFile)
      findFinalport7  =
re.compile("\."+strport2+'\('+"\s*"+strport3+"\),")
      TopoutFile = findFinalport7.sub('',TopoutFile)
      findFinalport8  =
re.compile(",\s*\."+strport2+'\('+"\s*"+strport3+"\)"+"\s*"+"\)")
      TopoutFile = findFinalport8.sub(' )',TopoutFile)
      findFinalport9  =
re.compile("\(\s*\."+strport2+'\('+"\s*"+strport3+"\)"+"\s*"+"\)")
      TopoutFile = findFinalport9.sub('()',TopoutFile)

      findFinalwire4  = re.compile(",\s*"+strport2+",")
      TopoutFile = findFinalwire4.sub(',',TopoutFile)
      findFinalwire5  = re.compile("wire\s*"+strport2+",")
      TopoutFile = findFinalwire5.sub('wire ',TopoutFile)
      findFinalwire6  = re.compile(",\s*"+strport2+";")
      TopoutFile = findFinalwire6.sub(';',TopoutFile)
      #print "searching for " + strport2 + " "+ strport3


      #print line
Topout.write(TopoutFile)
print "D1 Ports Cleaned in "+d1.rstrip('\.v')+"_rmport.v"
print "D2 Ports Cleaned in "+d2.rstrip('\.v')+"_rmport.v"
print "Top-Level Ports Cleaned in "+top.rstrip('\.v')+"_rmport.v"
```