Automated Place and Route Methodologies

For Multi-project Test Chips

by

Christopher Lieb

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2015 by the
Graduate Supervisory Committee:

Lawrence T. Clark, Chair
Keith Holbert
Jae-sun Seo

ARIZONA STATE UNIVERSITY

May 2015

ABSTRACT

This work describes the development of automated flows to generate pad rings, mixed signal power grids, and mega cells in a multi-project test chip. There were three major design flows that were created to create the test chip. The first was the pad ring which was used as the staring block for creating the test chip. This flow put all of the signals for the chip in the order that was wanted along the outside of the die along with creation of the power ring that is used to supply the chip with a robust power source.

The second flow that was created was used to put together a flash block that is based off of a XILIX XCFXXP. This flow was somewhat similar to how the pad ring flow worked except that optimizations and a clock tree was added into the flow. There was a couple of design redoes due to timing and orientation constraints.

Finally, the last flow that was created was the top level flow which is where all of the components are combined together to create a finished test chip ready for fabrication. The main components that were used were the finished flash block, HERMES, test structures, and a clock instance along with the pad ring flow for the creation of the pad ring and power ring.

Also discussed is some work that was done on a previous multi-project test chip. The work that was done was the creation of power gaters that were used like switches to turn the power on and off for some flash modules. To control the power gaters the functionality change of some pad drivers was done so that they output a higher voltage than what is seen in the core of the chip.

ACKNOWLEDGMENTS

First of all I would like to thank my parents for all of the dedication and support that they have given me throughout my college career. They have made it possible for me to further my education and career.

I would like to thank my advisor Dr. Lawrence Clark for the thesis topic, help in creating the HERMES block, and for the guidance and expertise knowledge he gave me throughout the pursuit my master's degree. I also thank Dan Patterson for his work on the testing and creation of the Verilog used for the design flows.

Finally I would also like to thank Microchip for the use of their flash module as well as allowing us to tapeout a chip with them, and my colleagues Chandarasekaran Ramamurthy for designing part of the flow for HERMES, Vinay Vashishtha for creating part of the caches used by HERMES, and Srivatsan Chellappa for help with verification of the test chip and the creation of the test structures.

TABLE OF CONTENTS

Page

LIST OF FIGURES v

LIST OF TABLES vii

LIST OF TERMS viii

CHAPTER

1. INTRODUCTION 1

    1.1. Design Flow 1

    1.2. Outline 4

2. PAD RING FLOW 5

    2.1. Pad Ring Usage 5

    2.2. Configuration and Pad Arrangement 5

    2.3. Pad Placement and Power Ring 10

    2.4. Routing and Final Steps 12

    2.5. Verification 13

3. FLASH BLOCK FLOW 15

    3.1. Flash Block Function 15

    3.2. Configuration 16

    3.3. Floor Planning and Initial Placement 18

    3.4. Power 20

CHAPTER Page

3.5. Design Placement and Optimization 22

3.6. CTS and Routing 23

3.7. Verification 28

4. TOP LEVEL FLOW 30

4.1. Configuration and Input Files 30

4.2. Floor Planning and Initial Placement 31

4.3. Power 33

4.4. Design Placement and Optimization 36

4.5. Verification 41

4.6. After Verification 45

4.7. Design Handoff 46

5. PREVIOUS TEST CHIP WORK...49

5.1. Power Gaters 49

5.2. Modified Pad Driver 53

6. CONCLUSIONS 55

REFERENCES 56

LIST OF FIGURES

Figure Page

1.1– ASIC Flow Diagram [1] 1

2.1– IO Rows for the Pad Ring 6

2.2– Example of Input File Syntax 8

2.3– Example IO Output Files 9

2.4 – Power Ring Metal Connections 11

2.5– Finished Pad Ring 14

3.1– Flash Block Diagram 15

3.2– Global Configuration File 17

3.3– Floorplan for Orientation of Flash Block (a) Horizontal (b) Vertical 19

3.4– Power and Ground Grid for the Flash Modules 21

3.5– Timing Output from preCTS Setup Optimization 23

3.6– Setup and Hold Timings for postCTS 25

3.7– Setup and Hold Timings for postRoute 27

3.8– Final Timing Check 29

4.1– Placements of Blocks in tc24 32

4.2– (a) “-flip f” Option (b) “-flip s” Option 34

4.3– Flash Block Power Domain Connections 35

4.4– DRVs Left 37

Figure Page

4.5– Custom SDC File for tc24 Clock Tree 38

4.6– (a) Gridded Routes with Errors (b) Jogged Routes 40

4.7– Signoff Timing Summary 41

4.8– Empty Areas Under HERMES and Test Structures 45

4.9– Flash Block Pins 47

4.10– Finished Design 48

5.1 – Power Gater Circuits 50

5.2 – (a) Transistor and Metal 1 (b) Metal 1 and 2 (c) Metal 2 and 3 51

5.3 – (a) Metal 2 and 3 Connections (b) Guard Ring Added 52

5.4 – PMOS and NMOS Power Gaters 53

5.5 – Pad Driver Voltage Conversion 54

LIST OF TABLES

Table Page

2.1 – Types of Cells Used ... 7

5.1 – Transistor Sizes ... 50

5.2 – NMOS Power Gater Inverters Transistor Sizes ... 50

5.3 – PMOS Power Gater Inverters Transistor Sizes ... 50

LIST OF TERMS

AP	- Aluminum Layer
APR	- Auto Place and Route
ASIC	- Application-specific Integrated Circuit
CMP	- Chemical-mechanical Planarization
CTS	- Clock Tree Synthesis
DECAP	- Decoupling Capacitor
DRC	- Design Rule Check
DRV	- Design Rule Violation
ECC	- Error Correcting Code
ECO	- Engineering Change Order
ESD	- Electrostatic Discharge
FPGA	- Field-programmable Gate Array
GDSII	- Graphic Database System
HDL	- Hardware Description Language
IC	- Integrated Circuit
IO	- Input Output
IP	- Intellectual Property
JTAG	- Joint Test Action Group
LEF	- Library Exchange Format
LIB	- Liberty
LVS	- Layout vs. Schematic
NMOS	- N-type Metal-oxide Semiconductor

PMOS	- P-type Metal-oxide Semiconductor
POC	- Power on Clear
PROM	- Program Read Only Memory
RC	- RTL Compiler
RTL	- Register-Transfer Level
SDC	- Synopsys Design Constraint
SoC	- System on a Chip
TAP	- Test Access Port

CHAPTER 1. INTRODUCTION

1.1. Design Flow

In integrated circuit (IC) design, there are many steps in the standard design cycle to produce a functioning chip. The design cycle is split up into three main sections. They are the front-end design which uses hardware description languages (HDL); verification, which is used to check that the design operates as intended; and finally, the back-end or physical design where the design is synthesized using standard and custom cells. Once the design is finished, the next steps are fabrication which turns the design into a physical device, putting the finished design in a package and the testing step which makes sure that the device is functional.

Physical design is further broken down. Here is used and describe an application-specific integrated circuit (ASIC) physical design flow as seen in Figure 1.1. When followed, this

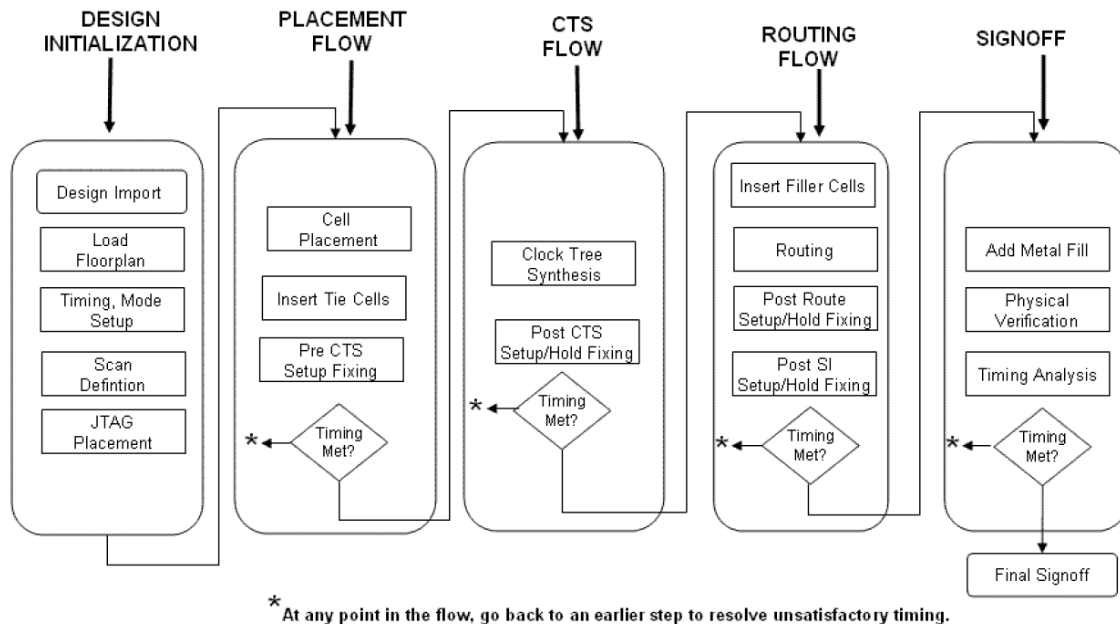


Figure 1.1– ASIC Flow Diagram [1]

ASIC flow produces the design with the desired specifications after the appropriate iterations. The first step of the physical design is getting the design netlist, which is synthesized from the front-end design HDL files. This netlist has a list of all the standard and custom cells that are used and their connections that were specified by the HDL files. This netlist is then passed to the physical design tool which is Cadence Encounter.

With the netlist, the next physical design steps are floor planning and placement. The floor planning and placement is one of the most critical sections when setting up an ASIC design. This is where the size and shape of the block is chosen and where all the main cells are placed within the block. Choosing the right floor plan and placement can take some trial and error. A bad floor plan and placement can result in problems, such as insufficient space for cells that are used for fixing timing problems or routing congestion problems which will degrade signal quality. In the worst case, some cells aren't able to have their pins routed due to routing congestion or cell placement. The floor planning and placement that works best is one that has ample space for timing fixing cells and as well as routing which maintains the metal orientations.

When the floor plan is chosen and the design is placed, the clock that exists in the design is still an ideal clock. Thus a clock tree network must be synthesized. The clock tree is also important to the design since it synchronizes the data transfers between the different functioning elements. When the clock tree is synthesized, buffers are added or removed and gates are either up sized or down sized to balance the clock. This is crucial to avoid setup and hold problems between the functioning elements.

After clock tree synthesis (CTS) has been completed, all the functioning elements need to be wired together. This is where the design is routed and all the pins are

connected as specified in the netlist. The design tool has information on where all the pins in the design are located based on the library exchange format (LEF) files. These files tell what layers the cells have, what type of placement sites they correspond too, and where the locations metal routes and vias are so that these routes and vias act as obstructions for other routes that are created later [2] . The routes that are created are on a grid with different vertical and horizontal metals layers.

At this point the timings for the design need to be checked and fixed to make sure that the design will work at the desired frequency. Both setup and hold timing checks are performed. If the timing checks are not done properly, then the data between two functional elements may not be captured correctly. This could be based on a hold problem where the data has not been held long enough or on a setup problem where the data arrived too late to be obtained. The way that Encounter fixes these problems depends on the nature of the problem. For setup problems, it adds or changes buffers to reduce the signals delay and for hold problems, Encounter adds cells that delay the signal. The latter is obviously easier.

The final steps for the physical design are to add decoupling capacitors (DECAP) and filler cells, and then to run final verification checks. The DECAP and filler cells are placed to fill in empty areas to help with power integrity and increase the density for these areas to meet the density required. After these cells are placed, the final timing for the design is checked again to make sure that the hold and setup timing for the circuit are still met. This final timing is to make sure that the added cells or any other changes don't skew the hold and setup timing to negatively impact the design. After the timing check, the final checks are geometry and connection checks to make sure that there are no cells

that are out of place as well as no shorted or unconnected wires. Once all the checks have passed and the design criteria have been met, the final step is to create a graphic database system (GDSII) of the finished design which is used to get the design fabricated.

1.2. Outline

The first chapter provides the basic flow for a system on a chip (SoC) design to create a working design from register-transfer level (RTL). Chapter 2 provides the flow and its breakdown on how the pad ring for the entire chip was created. Chapter 3 shows how the flash block was synthesized as well as the problems that were encountered during its creation. Chapter 4 shows how everything is put together at the top level when the chip is synthesized with all the finished components. Chapter 5 talks about work that were done on a previous test chip and the conclusions are discussed in Chapter 6.

CHAPTER 2. PAD RING FLOW

This chapter details the steps and data that are needed to create the basic flow for the pad ring which will later be used in Chapter 4 at the top level.

2.1. Pad Ring Usage

The pad ring is critical to any chip since it provides the means to send digital signals in and out of the chip. This is done using a pad driver that converts the digital signals from a usually higher external voltage to a low internal core voltage used by the chip while also providing electrostatic discharge (ESD) protection. These pad drivers connect up to the pads to get signals in and out of the chip, there are a couple of different types that can be used. The only type of bond pads that were available for the design were the wire-bond type. These wire-bond pads use wires for connections along the perimeter of the chip to connect to a package where the signals are brought in and out. The type of metal used for the wires are composed of gold. The reason gold is used is because it has a high resistance to corrosion and it has a high conductivity of electricity. These characteristics making it the most suitable for electrical connectors.

2.2. Configuration and Pad Arrangement

The first step in the creation of the pad ring starts with defining the size of the die, the number of metal layers from the design kit, and an input file that contains the signal ordering for the chip. The size of the die is chosen based on the process used and cost. There are then four input output (IO) rows dimensions that need to be calculated that fit along the outside edge of the die. These four IO rows are used for the pads, drivers, corners, and power connect cells as seen in Figure 2.1 on every edge of the die.

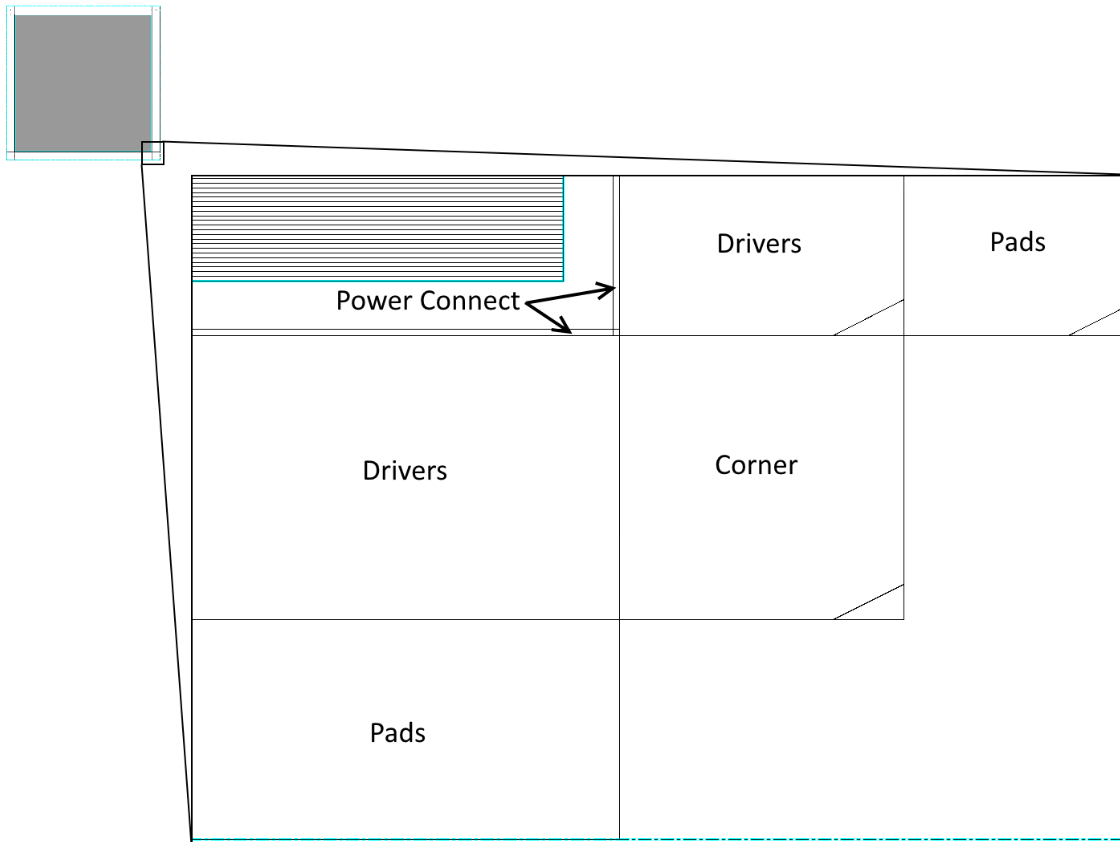


Figure 2.1– IO Rows for the Pad Ring

With the IO rows now placed, the pads, drivers, and all the other cells are then placed in their corresponding locations, guided by the input file that was mentioned earlier. This input file has the signals in the order that is wanted for each of the dies pads on each side using the following syntax "Pad name, Power connect, Analog gap, Pin name and Side".

- "Pad name" is the name corresponding to the pads name from the Verilog.
- "Power connect" is the name from the Verilog corresponding to the power connect cell. However if the pad is not a power pad corresponding to PVDD1CDG_33 or PVSS3CSD_33 then this field is left blank.

- “Analog gap” defines the space in microns between the analog cell and the cell next to it in the same analog domain. This is needed for analog fillers to be placed. This value starts with the first power cut (PRCUT_33) cell but is not used for the last power cut cell if it is next to a digital cell. This is necessary since cells in the digital domain require that digital fillers be used.
- “Pin name” is the signal name from the Verilog needed to create a top level pin.
- “Side” corresponds to the side where the pad is going to be placed using Top, Right, Bottom, and Left for its values.

Some examples of the syntax used for pad ring cells are shown in Figure 2.2 as well as descriptions for the cells used can be seen in Table 2.1.

Cells	Functions
PAD60NA	Bond pads that bring the signals in/out of the chip
PDUW0812CDG_33	Pad drivers convert external to internal voltages
PVDD2PR	Power connect cells connects vdd! to power ring
PVSS2PR	Power connect cells connects vss! to power ring
PRCUTA_33	Separates the analog and digital domain
PDB2A_33	Pass voltage through the analog domain
PVDD2POC_33	Enables the pads when external voltages are applied
PVSS3CDG_33	vss! brought in through this pad
PVDD1CDG_33	vdd! brought in through this pad
PVDD2CDG_33	vddio! brought in through this pad

Table 2.1 – Types of Cells Used

```
PRCUTA_33_PRCUT_1,,35,,Top
PDB2A_33_VPP,,10,VPP_1,Top
PRCUTA_33_PRCUT_2,,,,Top
PVDD1CDG_33_VDD,VDDConnect.PVDD2PR_Inst,,VDD!,Left
PVSS3CDG_33_VSS,VSSConnect.PVSS2PR_Inst,,vss!,Left
PadCell.PDUW0812CDG_33_Inst,,,PKG_PIN,Top
```

Figure 2.2– Example of Input File Syntax

The descending order of the pads in this file will be placed in the pad ring in the arranged order of left to right for the top and bottom sides. For the left and right sides, the pads are arranged in the pad ring from bottom to top. These orders coincide with increasing chip dimensions in the x and y coordinates. The order can be changed to a clockwise or counter clockwise order by changing the “io_order” value from default to clockwise or counterclockwise in the pad IO file [3] . This IO file is generated by running a Perl script from the latest file “generatePadPlacement_v18.pl” which produces two output files. The first file that is generated is the pad placement IO file and the second is the pin placement file that specifies to Encounter where to place the signal pins. Some examples of the files can be seen in Figure 2.3.

The order that is chosen for the arrangements of the pads is attributed to the high speed signals that the HERMES block uses. These high speed paths were chosen to be split evenly between the centers of the four sides. The non-high speed signals were then put on the outside of these high speed paths essentially putting them in the corners. This was done so that power pads could be interleaved between the high speed signals to supply enough power to sustain the high speed transitions of the signals as well as reduce the length of the connection wires. The order for the power to signals ratio is chosen to be

a vss! and vdd! cells next to each other followed by three high speeds cells, then a vddio! power cell followed by three more high speed cells. This was repeated until all of the high speed cells were used.

Signal location file

```
(globals
    version = 3
    io_order = default
)
(iopad
    (row
        (locals row_name="topleft_corner")
        (inst  name="Corner"   offset=0.0000 cell="PCORNER_33" )
    )
    (row
        (locals row_name="top")
        (inst  name="PVDD1CDG_33_VDD"               offset =2   place_status=placed )
        (inst  name="PVSS3CDG_33_VSS"               offset =64  place_status=placed )
        (inst  name="PadCell.PDUW0812CDG_33_Inst"   offset =102 place_status=placed )
        )
    (row
        (locals row_name="pwrConnect_top")
        (inst  name="VDDConnect.PVDD2PR_Inst"       offset=2  cell=PVDD2PR )
        (inst  name="VSSConnect.PVSS2PR_Inst"       offset=64 cell=PVSS2PR )
        )
    (row
        (locals row_name="top_bondpad")
        (inst  name="bp_PVDD1CDG_33_VDD"                offset =2   cell="PAD60NA" )
        (inst  name="bp_PVSS3CDG_33_VSS"                offset =64  cell="PAD60NA" )
        (inst  name="bp_PadCell.PDUW0812CDG_33_Inst"    offset =102 cell="PAD60NA" )
        )
)
```

Pin location file

```
(globals
    version = 3
    io_order = default
)
(iopin
    (up

        (pin  name="VDD"      x=304.49 y=3933.005 layer=AP width=0.5000 depth=0.1400 )
        (pin  name="VSS"      x=364.49 y=3933.005 layer=AP width=0.5000 depth=0.1400 )
        (pin  name="PKG_PIN"  x=424.49 y=3933.005 layer=AP width=0.5000 depth=0.1400 )
         )
)
```

Figure 2.3– Example IO Output Files

The cells that were used for bringing the signals in and out of the design were the PDUW0812CDG_33 pad driver. These pad drivers have many different functions depending on how they are configured by setting the voltages at their pins. They can be configured as an output driver where signals are taken from the core and drive what will be seen on the pads or as an input driver where the signals seen on the pads drive what the core logic will see.

2.3. Pad Placement and Power Ring

The IO pad placement file is used first to place all the pads, drivers, and power connect cells in their corresponding IO rows. At this point, it is a good idea to check the design to make sure that there are no cells corresponding to the pad ring that seem out of place. If there are, it means that these cells weren't used in the input file or the instance name corresponding to these cells has a different name than what was in the input file used to generate the pad placement file. If everything seems fine then filler cells are added between the pad drivers to complete the IO power ring.

Once the power ring is created, the next step is to create the internal power ring that supplies power to the core of the devices. This power ring is created by using two rings around the core for power and ground. These are created on the second and third top most metal layers to allow power signals to pass under it. Once the power ring is created, the core power supply is routed using metal 1 since all the devices get their power from the first metal layer.

With the core metal routed, the next step was to connect the power ring to the power drivers in the pad ring. This is where the power connect cells are used since the

power pad drivers only have their power coming out on metal 1 and metal 2. These power connect cells take the power on these metals and extend them all the way to the top most metals layers to allow easy access for connecting the power from the pads to the power ring. This is done by connecting them to the same metal layer as the pad ring and the metal above to allow the inner ring to be connected as can be seen in Figure 2.4. With the pad ring completed, the well tap cells are placed in the design to prevent latchup from occurring. This is done by tying the wells to their corresponding voltages which prevent them from drifting from their intended voltage.

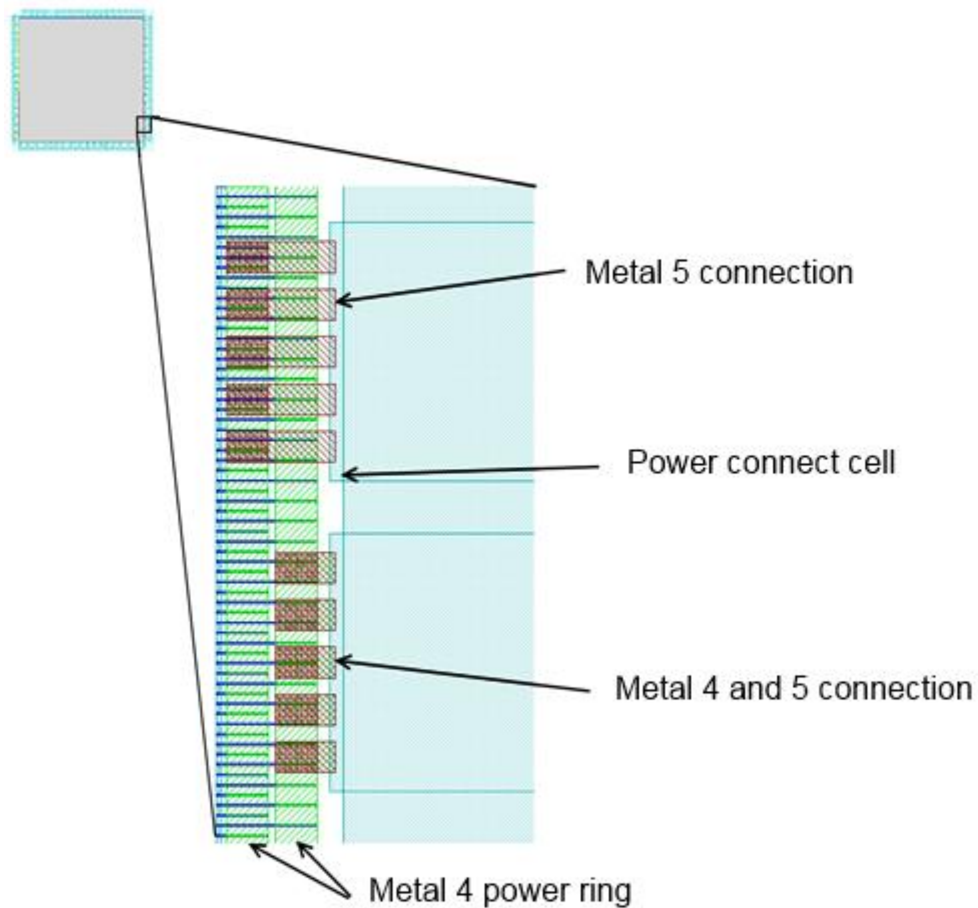


Figure 2.4 – Power Ring Metal Connections

2.4. Routing and Final Steps

With the power in place, tie cells are added to the pad ring. These tie cells are used to tie the pins of the pad drivers to either power or ground to change them to input or output drivers as what was specified in the netlist. With the tie cells placed and everything in the locations that what was wanted, DECAP and filler cells are added to the core of the design. These cells are to fill in the empty areas to the core, improving better power integrity along with increasing the density in these areas, necessary to abide by the density rules for the die.

At this point a trial route of the design is performed. This trial route is used since it is a quick way to make sure that the design can be routed. The trial route does not obey the rules required for metals routes (this is where the detailed route comes in). For the pad ring, there should be no congestion issues, so a failure here indicates a fundamental error. This detailed route removes all the trial routes and routes the design by connecting the pins of the devices as specified in the netlist. The detailed routing step also takes into account the other metal lines and rules required for the design such as metal orientations and grids used by the metal layers.

With the design routed, the top level pins now need to be placed. The file that specifies the pins is used to place the top level pins in their correct location, over the pads on the aluminum (AP) layer. They were placed on the AP layer to make sure that the pads can be connected for packaging while also making sure that everything is hooked up correctly. There is known of at least one university project (outside ASU) that placed these pins on an intermediate layer, passed layout versus schematic (LVS), but had no

pad connections. With the pins connected, the final step is to check the design to make sure that everything is placed in the right spot without geometry or connection problems.

2.5. Verification

With everything checked, the design is streamed out as a GDSII file and imported into Virtuoso for LVS and design rule check (DRC). In Virtuoso, the power pins are changed to correspond to power (vdd!) and ground (vss!) along with the addition of a power on clear (POC) pin to match the power pins in the rest of the design. When this is completed, DRC checks were run to make sure that all the layers in the pad ring follow the layout ground rules. These rules are defined by the fabrication plant to make sure that the layer being used will be created properly without any problems arising due to the layout. Once the DRCs were checked and there were no major violations, then LVS was run to make sure that all the devices in the design were connected as specified in the netlist. If the LVS fails then there might be a shorted wire or devices that aren't properly connected. If the LVS and DRC both pass then the pad ring flow is good and ready for use and the finished product can be seen in Figure 2.5.

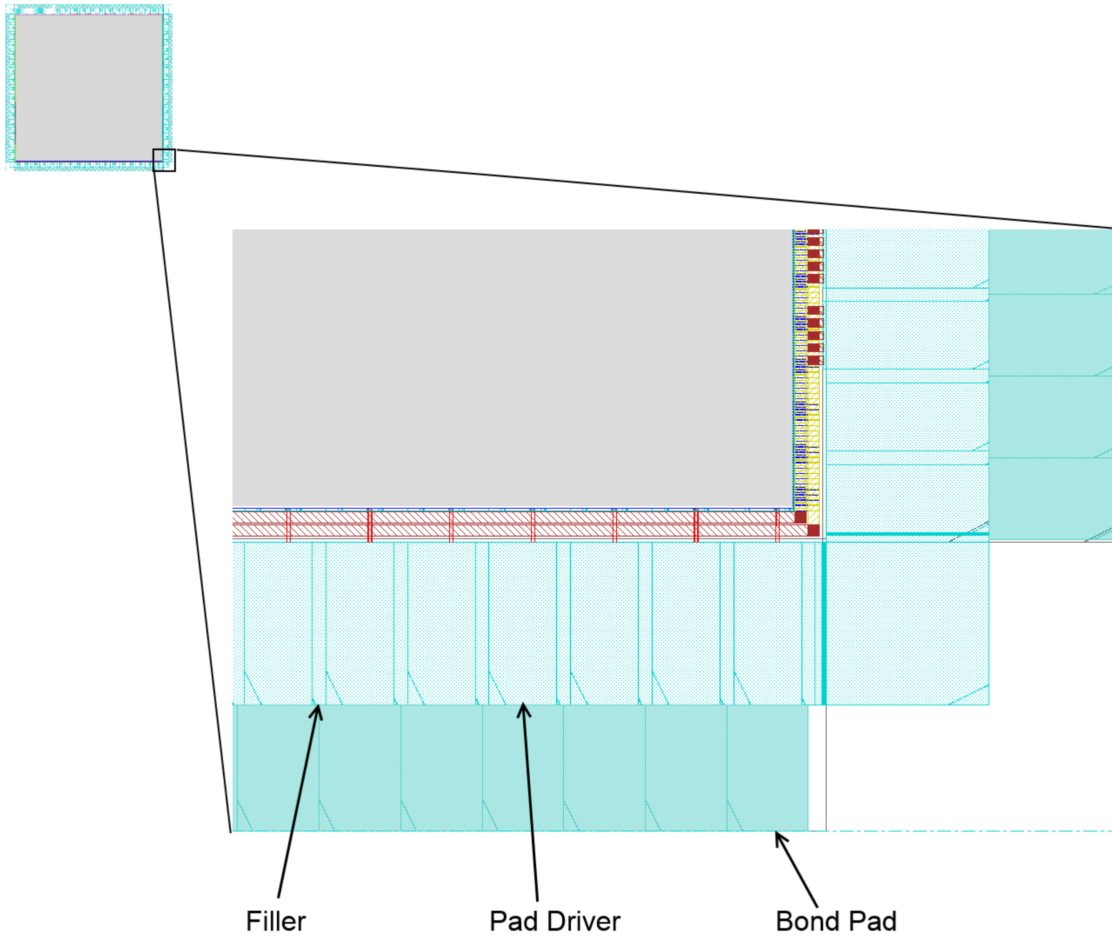


Figure 2.5– Finished Pad Ring

CHAPTER 3. FLASH BLOCK FLOW

This chapter discusses the flow and creation of the flash block that is used in test chip 24 along with the problems that were faced while synthesizing the block.

3.1. Flash Block Function

The design for the flash block is based on the Xilinx XCFXXP series of Platform Flash in-system programmable configuration program read only memory (PROM) that's available to the general public. A high level block diagram of the Xilinx XCFXXP can be seen in Figure 3.1. The embedded flash PROM portion of this block is licensed by Microchip Technology's SST division. The purpose for building this block is to test the key functionality to configure the Xilinx FPGA in radiation while also providing some hardness to this radiation by using temporal flip-flops and error correcting code (ECC).

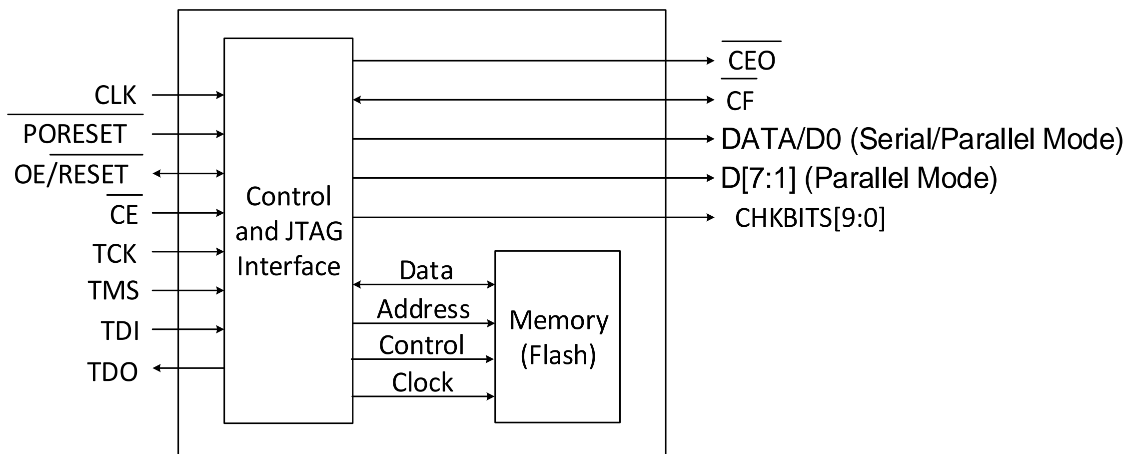


Figure 3.1– Flash Block Diagram

Most of the memory from the flash PROM are used for ECC check bits and for non-volatile control registers. The rest of the memory is used for the field-programmable

gate array (FPGA) configuration data. The control and Joint Test Action Group (JTAG) interface in the design is used to implement the JTAG Test Access Port (TAP), flash memory controller, and all of the other functions that were required to interface with the flash memory and external system.

3.2. Configuration

The first thing that needs to be done is to get the configuration files which are the LEFs, Liberty (LIB) files, and generate a synthesized Verilog structured netlist which will be used for setting up the design. All of these files were obtained from our custom design or the foundry libraries except for one LEF file. This file corresponded to the dummy Microchip flash intellectual property (IP) in which the LEF was needed since without it the block corresponding to the Microchip flash couldn't be placed in Encounter. The LEF we received was incorrect, so a LEF was generated based on the placeholder GDSII file of the dummy flash modules. Since the GDSII had no actual design in it but pin locations, the rest of the area inside of the flash module was put as a blocked layer to prevent metal routes from being routed into.

Once all the files were obtained, they were referenced to in the global configuration file used by Encounter (Figure 3.2). This configuration file also passes the names of the power signals used by the entire block. These power signals are vdd! and vss! for the core power along with PAVrefBank0, PAVrefBank1, VPP_0, VPP_1, IN20U_0, IN20U_1, TM_0[0], TM_0[1], TM_1[0], and TM_1[1] power supplies used by Microchips flash modules. The last file that is referenced in the configuration file is the structural Verilog.

```
global rda_Input
set cwd lpwd
set rda_Input(ui_netlist)     {./FlashBlock_Files/FlashBlock140729.v}
set rda_Input(ui_netlisttype) {Verilog}
set rda_Input(ui_settop)      {1}
set rda_Input(ui_topcell)     {FlashBlock}
set rda_Input(ui_view_definition_file) { \
    ./FlashBlock_Files/FlashBlock_Timing.mode \
    }

# TMR MSFF must be CORE here

set rda_Input(ui_leffile) { \
    ../../lef/tcbn90lpbwp7t_9lmT2RDL.lef \
    ../../lef/FlashBlock/pfm_390t105kx32_v1a0_v7.lef \
    ../../lef/FlashBlock/custom_blocks_lef_09162013.lef \
    }

set init_gnd_net {vss!}
set init_pwr_net { \
                vdd! \
                TM_0[0] \
                TM_0[1] \
                TM_1[0] \
                TM_1[1] \
                IN20U_0 \
                IN20U_1 \
                PAVrefBank0 \
                PAVrefBank1 \
                VPP_0 \
                VPP_1\
              }
```

Figure 3.2– Global Configuration File

This structural Verilog netlist is the output from the RTL Compiler (RC) tool that was used for synthesis. This RC tool takes the front-end Verilog and synthesizes it from a behavioral code based design into a structural gate-level representation which is the hardware equivalent implementation of the design [4] . This design is comprised of cells

that are from the standard library along with our custom cells. While the Verilog is synthesized, the timing for the design is set for a certain speed. With the timing specified, a Synopsys design constraint (SDC) file is generated from RC which is used during timing optimizations and CTS [5] . In fact, the RTL compiler generated netlist has dubious timing, so not much effort should be made there. The auto place and route (APR) optimization tool will do this based on accurate parasitic.

3.3. Floor Planning and Initial Placement

The original floor plan is chosen to be the height of the flash modules and the width of the core area that fit within the pad ring. With the floor plan created, the flash modules were placed on the left and right side of the block with their pins pointed to the inside. This allowed the logic that is connected to the flash modules to be in the center of the block. Having the logic in the center prevents wires from being routed over the flash modules and reduces the wire lengths between the blocks which helped with the timing of the circuit. This floor plan worked until it was later revealed that the process that we would be using was not the assumed 9 metal layers, but actually 7 metal layers. Due to this, the HERMES block had to be redone since it used routes up to the 9th metal layer. For HERMES to be able to be routed on the 7 metal layers, the orientation had to be changed from horizontal to vertical. This overlapped the area that was going to be used for the flash block and thus the flash block orientation was changed from horizontal to vertical as can be seen in Figure 3.3.

Since the flash block orientation changed, both of the flash modules placements were chosen to be on the left side of the resized block. A gap is also put in between them as well as a gap on their right to allow for logic and metal wiring. At this point the

iterative nature of the design process became apparent. This floor plan worked until the design is placed at which point it was discovered that there wasn't enough space for every component. Consequently, the floor plan was sized up by increasing the gap in between the two flash modules. This allowed all the components to be placed, although another problem was encountered later due to this floor plan. The latter problem was that the design couldn't meet its timing specs due to that there was not enough space next to the pins to allow a good amount of the components to be placed next to them. Since these components couldn't be placed by the pins, they were placed far away from them and the result was that it took more time for the signals to propagate to them. So to reduce the time it took for the signals to propagate to the pins, the gap on the right of the flash modules is increased to allow more components to be placed next to the pins as seen in Figure 3.3(b).

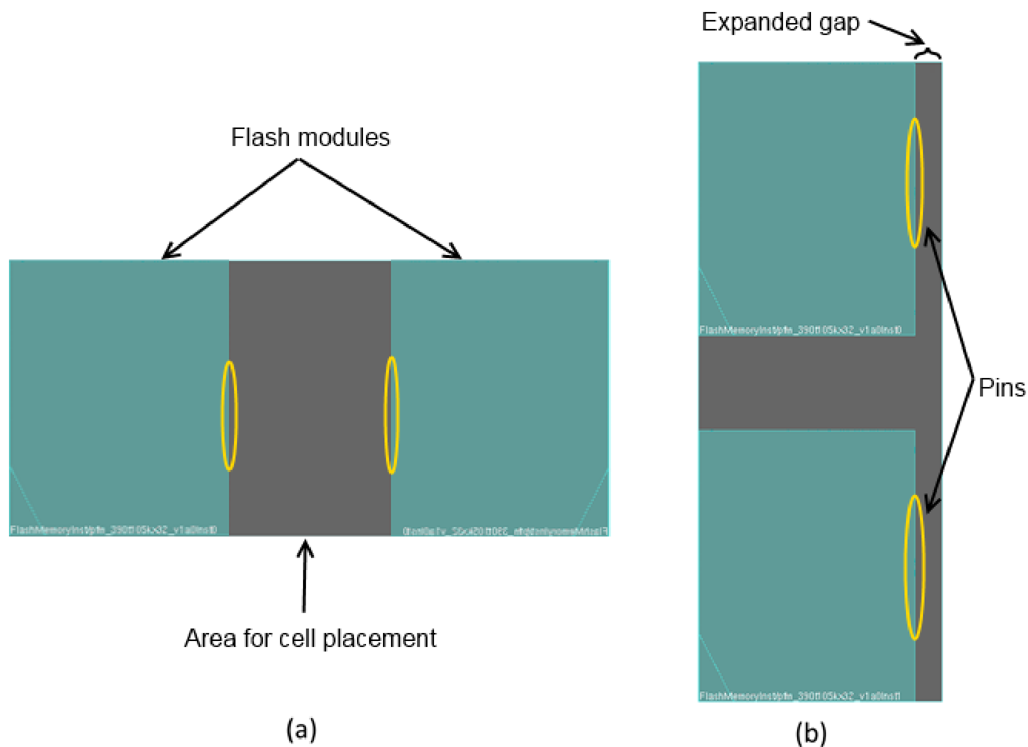


Figure 3.3– Floorplan for Orientation of Flash Block (a) Horizontal (b) Vertical

3.4. Power

Building the power grid for the design followed the flash modules placement. A good power grid is essential, since without it, there can be excessive voltage drops at the devices. These voltage drops can result in timing slowdown or as a worst case, resulting in the device failure. So to prevent these voltage drops, the power grid is made up of a metal grid with low enough resistance to ensure that enough power is supplied to all the devices in the block. The flash block grid is spaced on metal 5 and metal 6 with each section containing a vdd! and vss! as can be seen in Figure 3.4. This grid brings in powers from all sides of the block and is later connected to the power ring. When the power grid is created, if any metals that are located below it are of the same power, a via stack is created down to that metal to connect it to the grid. This gives a good power connect throughout the block at the lower metals where the devices mainly get their power from.

The other powers that need to be accounted for were the powers being used by the flash modules which are VPP, VDD, VDD18, and GND as well as other special purpose powers. These powers need their own separate power grid besides GND which is connected to vss! since all the grounds in the design are common. Since these powers are specific to the flash modules, the power grid for these powers were only created over top of the flash modules as can be seen in Figure 3.4. Even though the flash modules were the same, different power supplies were furnished for each flash module.

This is done by putting metal blockage layers on areas where the grid isn't wanted while also leaving room for the metals to extend past the modules to be connected to later. The "addStripe" command is then run with the power nets for one of the flash

modules as well as a set to set distance to allow room for the other modules nets to be added. This command is run for each flash module on metals 5 and 6. After the power grids were created, they were connected to the flash modules by running the tools special route command which connects all the pins from the flash modules to their corresponding wires in the grid. Once the metal routes are created they were checked to make sure that they were connected to flash modules properly.

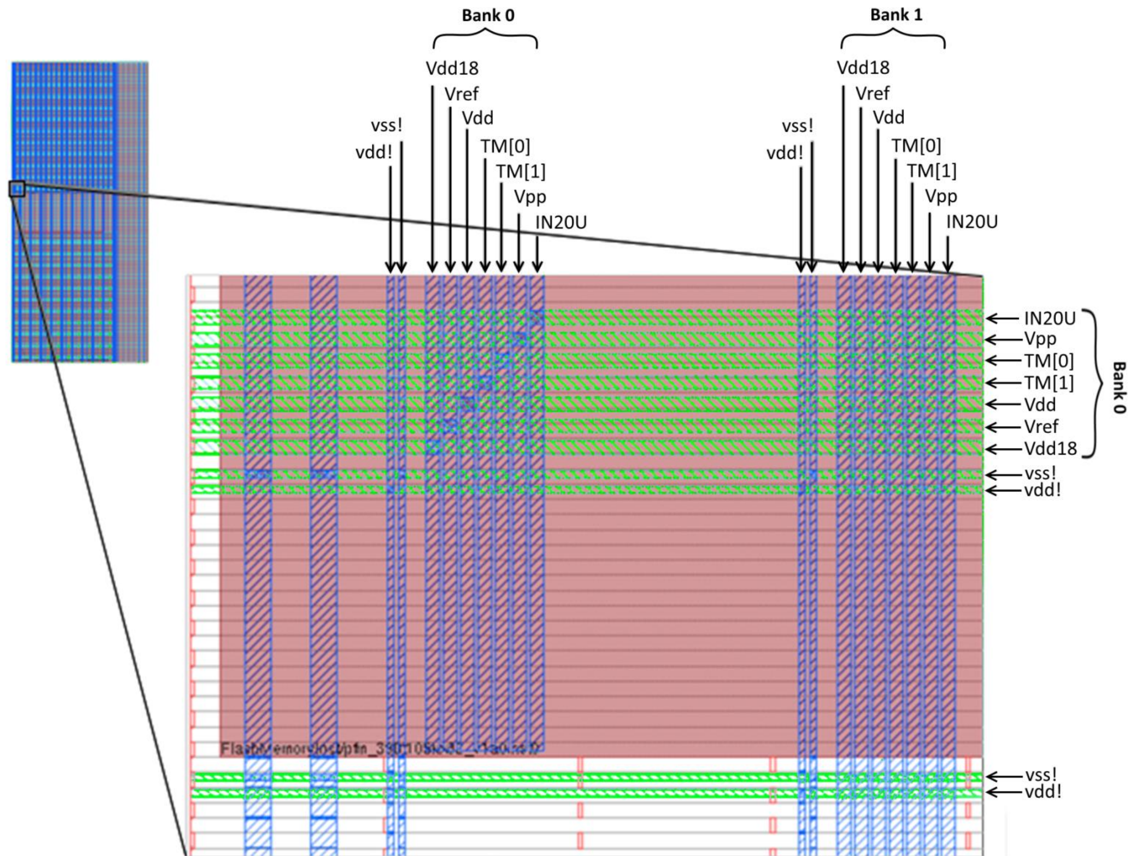


Figure 3.4– Power and Ground Grid for the Flash Modules

3.5. Design Placement and Optimization

Once all the power grids were in place and connected properly, the next step is the standard cell placement. When the tool runs the placement of the components, it looks at all the cells and tries to arrange them to minimize timing and area. This is done by taking cells that communicate with each other and placing them as close together as possible.

With the cells placed, the next step is to optimize the designs current timings with what are now very accurate wire estimates and to fix design rule violations (DRVs). These DRVs are problems in the design that do not meet the designs stand cell specifications such as a wire that has a bad slew rate, generally the result of under driven wires, or excessive fan-out for that driver. All of the DRVs were fixed by the tool for which it splits up the wires and adds buffers for a driver as needed.

After the DRVs were eliminated, an optimization is run to get the initial setup timings. These timings were reported in five different sections, with the smallest or most negative value as the worst in the design as can be seen in Figure 3.5. The first section gives the worst value of all the sections, the second part is the timing between two registers, the third section is the timing from pins into the block to a register, the fourth section is the timing from a register to the block output and the last section in the timing for the signals that comes into the block and goes out without interacting with a register (feedthrough signals). This first timing optimization gives a good indication for how the timing in the design will turn out since the design will not go through vast changes but only ones that tweak the design around.

With the first vertical floor plan that was run, the timing for the design had negative slack which is unfixable. At this point, as mentioned, the width of the floor plan

was increased. This helped the negative slack due to cells being placed too far away from their signal pins which resulted in the design having a positive setup slack. After iteration, if the timing slack was slightly negative, then a couple of steps turned it positive. One of the steps was to run an incremental place design to try and get the cells in a slightly different placement, which helped the timing. The other step run was an optimization that tried to move the cells around to reduce the wire length which helped reduce the timing between the cells.

```
###############################################################
#   Generated by:       Cadence Encounter 12.00-p002_1
#   OS:                 Linux x86_64
#   Generated on:       Sun Aug 24 11:21:21 2014
#   Design:             FlashBlock
#   Command:            optDesign -preCTS -incr -outDir ./Report -prefix preCT...
###############################################################

------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.724  |  0.724  |  2.196  |  2.701  |  7.954  |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |
|     Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |
|           All Paths:|  6849   |  6609   |   251   |   26    |    5    |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Figure 3.5– Timing Output from preCTS Setup Optimization

3.6. CTS and Routing

Once the timing had a positive slack via the placement optimization, the CTS for the design was run. The CTS first removes all the trial routes so that the clock tree can be routed without being constrained by other signal wires. This is necessary since the clock tree route is the most important route and should thus not be constrained by other signal

routes. Once the clock tree is routed, the routes that were created are set to "fixed" so that other steps in the flow don't upset the clock timing. If the route were changed or moved, the clocks timing to the cells downstream from the clock tree will get different times which negatively impact the timing, or worst-case, create a hold violation.

After CTS, the design was reoptimized again using the clock tree defined clock timings. Now hold and setup optimizations can be run as shown in Figure 3.6. Before there was no timing information for the clock, which is required to define the hold timing. If either the hold or setup don't meet the timing, i.e., don't have a positive slack, then they were run again, this time with the incremental flag to drive the timings more positive. While fixing the timing, hold is more critical to fix than setup. This is because you can fix setup timings by slowing down the clock to the logic, while hold is defined by the logic as a race condition and can't be fixed in the finished product. In the former case the design is a bit slow. In the latter case it is nonfunctional.

Once CTS and the timing optimizations were finished, the design was finally able to be routed. This is done by doing a "global detailed route" that routes the design using the process rules. These rules are to keep the routes a certain distance apart from each other along with width restrictions. If these rules were not followed, then problems will occur during the manufacturing process of the chip. Examples of potential problems are wires shorting to each other if they are too close, wires not being connected correctly to each other due to variation during the processing of the chip for which the rules account for, or divots in the chip might occur during manufacturing if the metals are too wide.

Setup timing

```
###############################################################
#  Generated by:      Cadence Encounter 12.00-p002_1
#  OS:                Linux x86_64
#  Generated on:      Mon Aug 25 13:22:30 2014
#  Design:            FlashBlock
#  Command:           optDesign -postCTS -incr -outDir ./Report -prefix post...
###############################################################

------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.114  |  0.114  |  5.282  |  5.576  |  7.631  |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |
|          All Paths:|  6849   |  6609   |   252   |   26    |    5    |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Hold timing

```
###############################################################
#  Generated by:      Cadence Encounter 12.00-p002_1
#  OS:                Linux x86_64
#  Generated on:      Sun Aug 24 13:19:03 2014
#  Design:            FlashBlock
#  Command:           optDesign -postCTS -hold -outDir ./Report -prefix post...
###############################################################

------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.051  |  0.051  |  4.950  |  5.417  |  7.630  |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |
|          All Paths:|  6849   |  6609   |   252   |   26    |    5    |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Figure 3.6– Setup and Hold Timings for postCTS

When the routes were correct and there were no DRCs from the wiring, then the chip design setup and hold times were reoptimized for the final time. Since now that the finalized wires were placed instead of trial routes, the timing of these wires are now precisely known. These finalized wires have set timings because their capacitance and resistance can be extracted. This information is used to find the exact delays whereas the trial routes only had an estimated delay. The differences in the estimated delay to the finalized delay sometimes may make the setup and hold margins negative. If the optimizations can't pull the timings positive then wire spreading is used to reduce the cross coupling capacitance between the lines. This helped by reducing the delay since line to line capacitance predominates and since the cross coupling capacitance is decreased by one over the spacing. This technique was used to decrease the delay and help the setup timing reach a positive slack as can be seen from the timing results seen in Figure 3.7. The timing was taken one step further, in which the smallest slack timings were checked using the tools timing display and the smaller buffers that the tool didn't use were used to help get more positive slack.

At this point, DECAP and filler cells are added to fill in the empty space. If there is not a high enough density throughout the chip then problems can arise in the fabrication process due to lithography and etching loading problems. One such problem can arise during the chemical-mechanical planarization (CMP) polishing processes. When the wafer is polished to create a smooth and even surface, if there are less dense areas on the chip, then these areas may cause non-uniformities in the surface of the die [6] which can affect other processes later down the line resulting in the chip not functioning properly.

Setup timing

```
###############################################################
#  Generated by:      Cadence Encounter 12.00-p002_1
#  OS:                Linux x86_64
#  Generated on:      Sun Aug 24 21:32:33 2014
#  Design:            FlashBlock
#  Command:           optDesign -postRoute -incr -outDir ./Report -prefix po...
###############################################################

------------------------------------------------------------
     optDesign Final SI Timing Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.174  |  0.174  |  5.323  |  5.585  |  7.657  |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |
|          All Paths:|  6849   |  6609   |   252   |   26    |    5    |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Hold timing

```
###############################################################
#  Generated by:      Cadence Encounter 12.00-p002_1
#  OS:                Linux x86_64
#  Generated on:      Sun Aug 24 19:28:21 2014
#  Design:            FlashBlock
#  Command:           optDesign -postRoute -hold -incr -outDir ./Report -pre...
###############################################################

------------------------------------------------------------
     optDesign Final SI Timing Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.207  |  0.207  |  7.700  |  5.844  |  7.639  |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |
|          All Paths:|  6849   |  6609   |   252   |   26    |    5    |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Figure 3.7– Setup and Hold Timings for postRoute

3.7. Verification

Once the routing optimization and fill were done, the chip design was finished and the final timing needs to be checked one last time. This is to make sure that the chip was designed to work at its intended speed and that any other steps after the final optimizations didn't affect the timing in a negative way. If the timing passed the checks for both hold and setup as can be seen in Figure 3.8, the design was then streamed out into a GDSII file and imported into Virtuoso. With the design in Virtuoso, DRC and LVS were run to make sure that there were no layout errors and that the design connections were appropriate.

To ensure the integrity of these checks, no pins were changed or added to the layout at this point except for the power pins. The change that is needed for the power pins is that there was a colon in their name which is removed to follow the schematic. If the colon was not removed, then LVS reported an error since the powers don't match in layout and schematic. Once both DRC and LVS pass, then the flash block is finished and ready for the top level which is discussed in the next section.

Setup timing

```
##################################################################
# Generated by:      Cadence Encounter 12.00-p002_1
# OS:                Linux x86_64
# Generated on:      Thu Aug 28 15:31:45 2014
# Design:            FlashBlock
# Command:           timeDesign -prefix signoff -signoff -reportOnly -outDi...
##################################################################

--------------------------------------------------------------
         timeDesign Summary
--------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.727  |  0.727  |  2.066  |   N/A   |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |   N/A   |   N/A   |   N/A   |
|          All Paths:|  6822   |  6609   |   251   |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Hold timing

```
##################################################################
# Generated by:      Cadence Encounter 12.00-p002_1
# OS:                Linux x86_64
# Generated on:      Thu Aug 28 15:31:15 2014
# Design:            FlashBlock
# Command:           timeDesign -prefix signoff -signoff -reportOnly -hold ...
##################################################################

--------------------------------------------------------------
         timeDesign Summary
--------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.297  |  0.323  |  0.297  |   N/A   |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |   N/A   |   N/A   |   N/A   |
|          All Paths:|  6822   |  6609   |   251   |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+
```

Figure 3.8– Final Timing Check

CHAPTER 4. TOP LEVEL FLOW

This chapter discusses the flow and creation of the top level where all the internal components are connected together to create the finished test chip 24, along with the problems that were faced during synthesization.

4.1. Configuration and Input Files

Once all of the main internal blocks were finished and verified, they needed to have their LEFs generated. These LEF files could be generated in a couple of ways. One way is that every layer used in the block is put in the file. This way Encounter knows where all of the metals are located inside of the block. With the location of all the metal locations known, Encounter can create routes inside these blocks without worrying about shorting routes together. The only problem with this type of configuration for the LEF file is that it took a really long time to create. The other way to create the LEF file is to make every layer in the design as a blocked layer. This way Encounter can't route in these layers and it took a short time to create the LEF file. Since every layer is blocked from being routed to, there's an exception created for the pins. The exception is that cuts were made in the blocked layers where the pins were for Encounter to be able to route to them. The blocked layers type of LEF took a short time to create compared to the other way which took days to finish. A hybrid of these two types were chosen where metal 5 and up were put in as detailed and metal 4 and below were completely blocked. This hybrid type of LEF was chosen since the time for its creation was manageable before the tapeout deadline where the design is sent out to the fabrication plant to be created.

With the LEF files generated, they were passed to Encounter along with the other configuration files. These configuration files were generated or obtained the same way as

the flash block, where the Verilog file was generated by passing the RTL through RC as well as the SDC file. The global configuration file is also the same with specifying all the power domain names as well as the LEFs and LIBs needed for the design. The power domains include all of the same powers from the flash block as well as the internal powers vss!, vdd! and the pad ring power (vddio!). For the LEFs and LIBs, the same ones were passed for the standard library files as with the flash block plus some extra files. These extra files corresponded to the LEFs and LIBs of the completed flash, HERMES, clock, and test structure blocks that make up the main components of the internal design.

4.2. Floor Planning and Initial Placement

With the entire configuration files setup and loaded into Encounter and all of the main internal components of the chip completed, they were now combined together to create the full working chip. The first step was to use the pad ring flow described in Chapter 2 to create the pad ring along with the power ring. Once the pad ring was completed, the four major blocks which are flash, HERMIES, clock and test structures blocks were placed in the core area of the design as can be seen in Figure 4.1. The HERMES block was placed on the right since its pins were on the left. This allowed the routing and signal propagation to be around the same for all of HERMES signals to the pad ring. Next, the clock block was placed next to the HERMES pins in order to minimize the clock tree to HERMES. After the clock block was placed, the flash block was placed in the upper left. This was because the powers for the flash block were passed through the analog domain part of the pad ring which was chosen to be in the upper left part of the pad ring. Now that the major blocks were placed, the left over space under the flash block is used for the test structures.

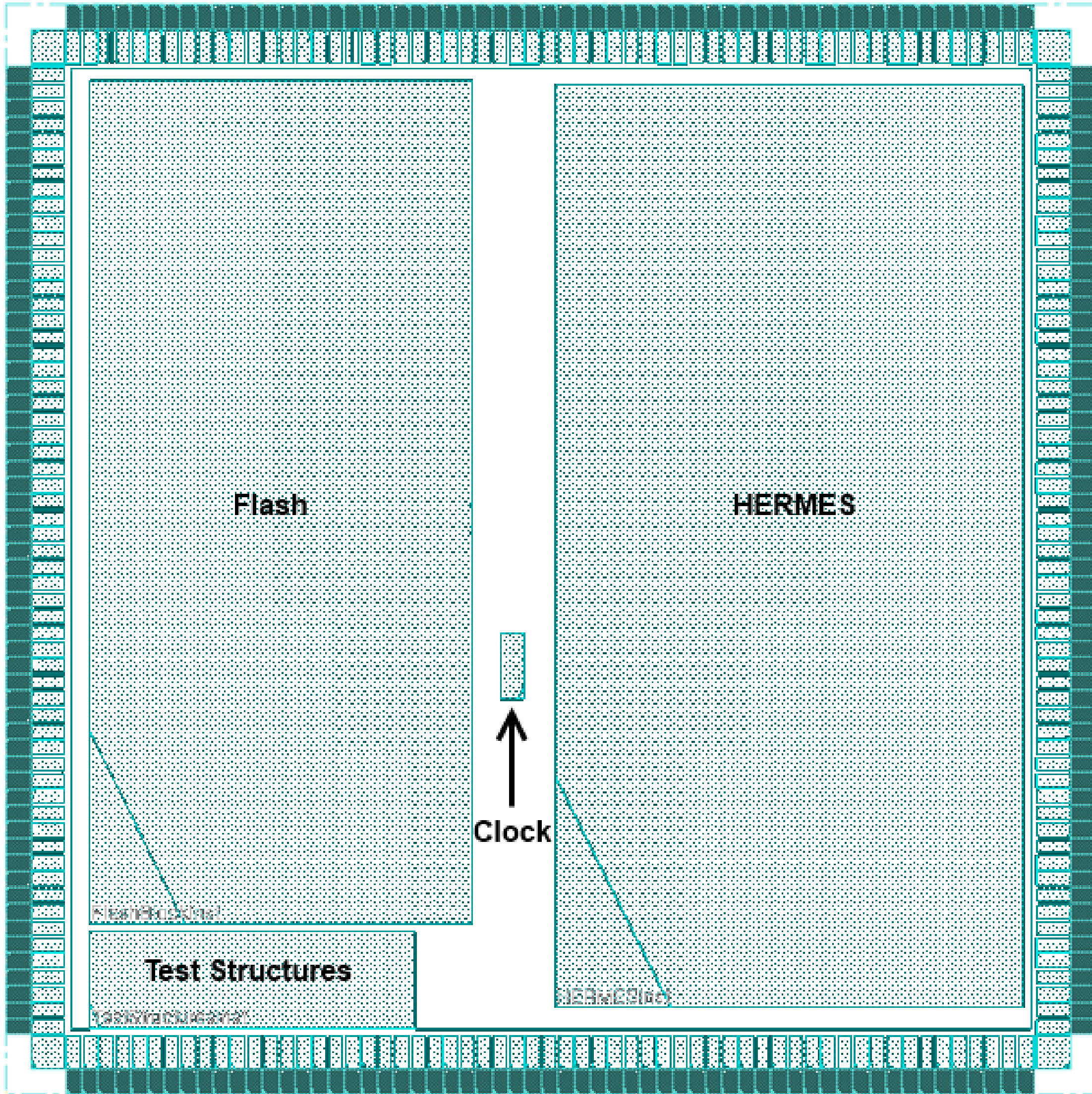


Figure 4.1– Placements of Blocks in tc24

There was one problem that was encountered while placing the HERMES and test structures block. This problem was that the internal power grid corresponding to vdd! and vss! on the metal 1 was offset from the power grid used in the top level, whereas the flash and clock block aligned correctly to the internal power grids. This happened because during the creation of these blocks a different setting was used for the "FloorPlan" command that specified the orientation of the bottom rows in these designs.

This setting is the "-flip" which specifies the orientation of the bottom row in the core area [7] . HERMES and the test devices used the "-flip f" option which flipped the first row from the bottom and every other row as seen in Figure 4.2(a), whereas the flash block and top level used the "-flip s" option that flipped the second row from the bottom and every other row as can be seen in Figure 4.2(b). To fix this problem, the LEF files for HERMES and test structures blocks were modified so that the starting point of the blocks were offset by one row. This resulted in the internal power grid on metal 1 aligning to the power grid used at the top level.

4.3. Power

Once all of the components were placed, the power grid was created to bring in the power to all of the placed components. The first power grid that was created was the metal 1 vss! and vdd! grid which all of the devices use to power the transistors used in the blocks. The metal 1 grid was configured to extend pass the core and connect to the power ring. But before the metal 1 grid was put down, blocking layers were used over top of the flash and test structures blocks. However, the blocking layer over HERMES was missed which caused a problem which will be discussed later in Section 4.5.

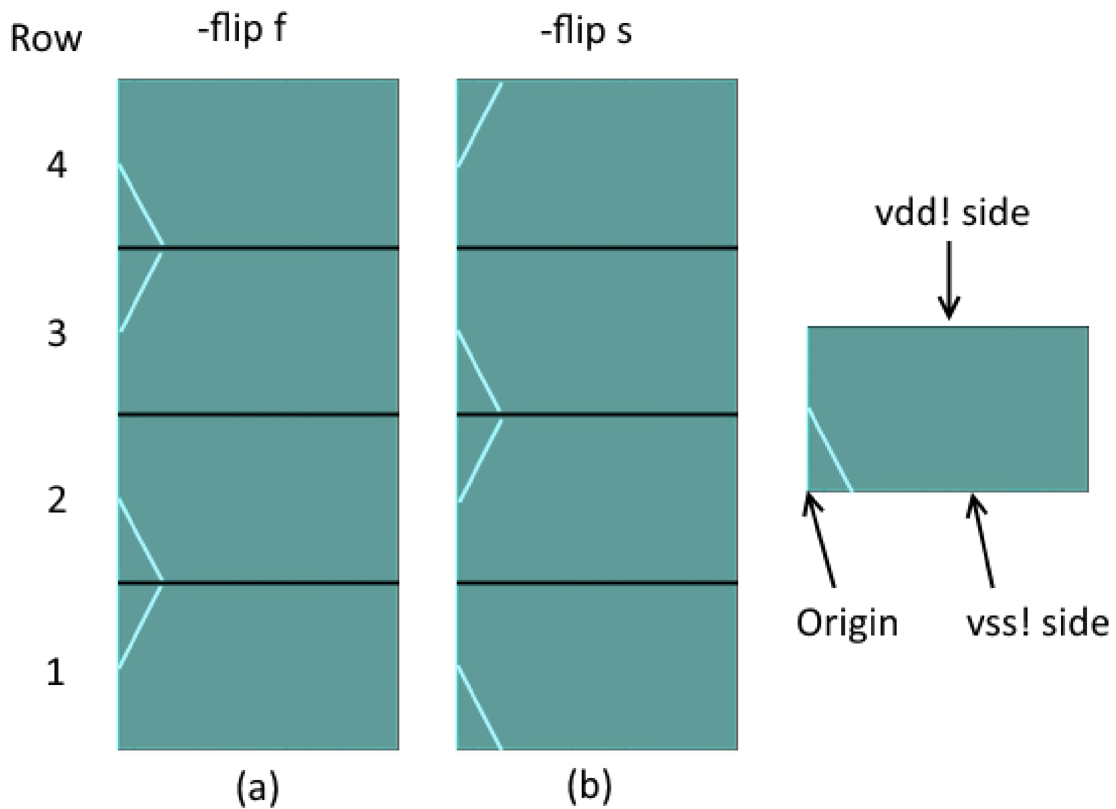


Figure 4.2– (a) "-flip f" Option (b) "-flip s" Option

Next the flash block power grids were created for the flash macros. With the power grid in the flash block already created, the grid needed to be connected up to the analog part of the pad ring where these powers were brought in. Since these powers are passed in above the flash block, routes on metals 2 through 5 are created between the flash block and the different power domains as can be seen in Figure 4.3.

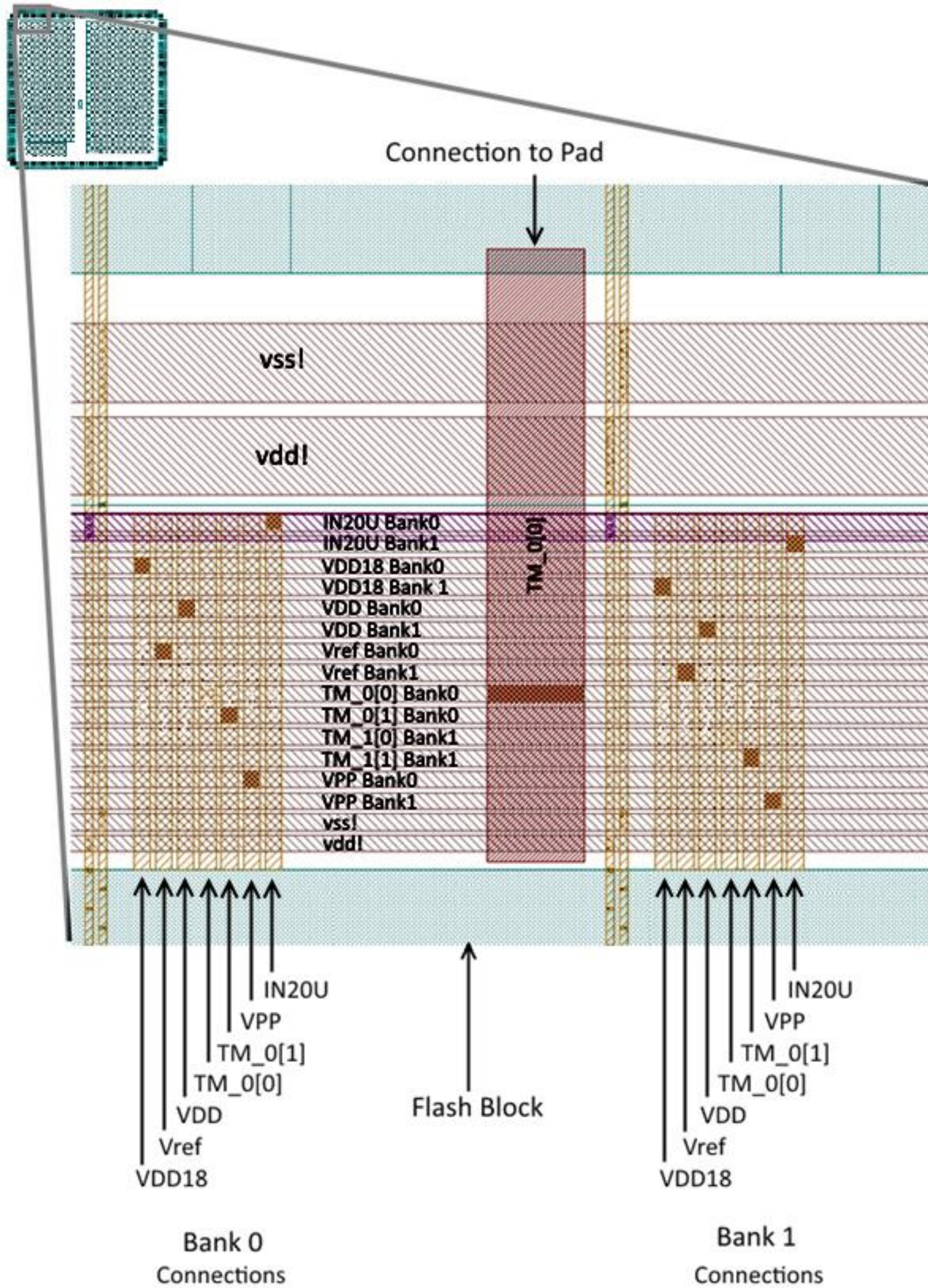


Figure 4.3– Flash Block Power Domain Connections

With the flash block power domains connected properly, the rest of the power grid was created for the vss! and vdd! domains. Since HERMES and the test structures had been shifted up by one row, this offset the horizontal internal metal 5 power grids from the flash block. This couldn't be fixed by shifting the blocks to align the power grids. Since if the blocks were shifted up or down, they would be shifted by two rows to keep the metal 1 power grid aligned as vss! and vdd!. Since HERMES and the test structures were offset by one, the power grids couldn't be aligned. The way around this was to create two sections of the horizontal metal 5 power grids, one over the flash block and the other over HERMES and the test structures. This is fine since when the vertically metal 6 power grid was placed, it connected all of the metal 5 together. Also, it was not a problem for the metal 6 grid in the blocks as they aligned with each other since HERMES and the test structures blocks were offset vertically.

4.4. Design Placement and Optimization

Now that the power grid was created and connected properly, the design needed to be placed and optimized similarly to how the flash block is in Section 3.5. The first step was to run the command for place design so that the logic separately from HERMES, flash, test structures, and clock instances were put in the design. There was no need to worry about shortage of space for the placed components since they had all of the extra room between the 4 major instances that were manually placed earlier. With the design placed, the DRVs for the design were fixed before optimizing the setup timing. Initially there was a fair amount of DRVs which could have hurt the designs performance, but once the optimization is run to fix the DRVs, all but a couple were fixed by the tool as can be seen in Figure 4.4. The ones remaining were the ones that the tool couldn't fix so

these DRVs were examined and the timing impact that was reported for these were very little. Therefore they were left alone since the small timing had a small impact on the design. After all of the DRVs were fixed, the initial setup timing for the design was optimized.

DRVs	Real		Total
	Nr nets(terms)	Worst Vio	Nr nets(terms)
max_cap	0 (0)	0.000	0 (0)
max_tran	46 (84)	-0.075	153 (303)
max_fanout	0 (0)	0	0 (0)
max_length	0 (0)	0	0 (0)

Figure 4.4– DRVs Left

Once the setup for the design was optimized, the next step was to create the clock tree. The clock tree was run slightly differently from what was run in Section 3.6 for the flash block. The difference was that a custom SDC file was created and passed to Encounter as can be seen in Figure 4.5. This custom SDC file has a list of pins where the origin of the clock started, which is at the pins from the pads ring. This took some trial and error to figure out that the pins needed to start at the origin of the clock. Initially the pins were chosen to be wherever the clock needed to be connected; this is the pins at the start and end of the clock. Once it was figured out that the pins were at the start of the clock tree, the pins corresponding to HERMES were grouped into one clock and the pins for the flash block into another.

The SDC file also specified false paths in the design. If these paths were not set as false paths, then Encounter would use these paths to check the timings of the signals. This would give false timing since the paths are static. The reason that these paths were

false is because they were signals to choose between which blocks the pads are going to be used for getting signals in and out of the chip. These signal paths were going to be set before the blocks were going to be used and are not going to be changed when the blocks were in use.

```
# ####################################################################

#  Created by Encounter(R) RTL Compiler v12.10-p006_1 on Wed Sep 03 16:09:18 -0700 2014

# ####################################################################

set sdc_version 1.7

set_units -capacitance 1000.0fF
set_units -time 1000.0ps

# Set the current design
current_design tc24

create_clock -name "FlashGClk" -add -period 10.0 [list \
        [get_pins PadRingSSInst/PAFromPadCellCH[104]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[105]] \
        ]

create_clock -name "ClockSourceCGClk" -add -period 3.0 [list \
        [get_pins PadRingSSInst/PAFromPadCellCH[132]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[133]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[134]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[135]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[136]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[137]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[138]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[139]] \
        [get_pins PadRingSSInst/PAFromPadCellCH[0]] \
        ]


set_false_path -from [get_ports  BLK_SEL[0] ]
set_false_path -from [get_ports  BLK_SEL[1] ]

set_clock_gating_check -setup 0.0
set_max_fanout 32.000 [current_design]
set_max_transition 0.125 [current_design]
set_max_capacitance 0.0 [current_design]
set_wire_load_mode "segmented"
set_wire_load_selection_group "WireAreaForZero" -library "tcbn90lpbwp7tlvttc"
```

Figure 4.5– Custom SDC File for tc24 Clock Tree

Once the clock tree is routed, the design was optimized for setup and hold timings. There was a problem that was encountered while running the optimization for the hold timings, in which the tool froze and wouldn't respond. It seemed like the tool couldn't complete or start the optimization. Since the tool wasn't able to complete the optimization for hold, this part of the optimization is skipped after it was determined that it is not necessary. This was due to the fact that the hold optimization is for the register cells at the top level and there were no register cells placed in the top level.

After the optimizations, a check place command was run to check the placement of all the cells. This command is used to make sure that there are no overlapping cells and that everything in the design is in their correct place. If this command is skipped and there were cells that overlapped, this would have caused a large problem later which would be caught with the layout verification. The design would then have to be brought back to the check placement step to easily fix the problem, or else almost everything around the overlapping cells would have to be moved or shifted by hand.

Now that the design was checked, it was routed to connect all of the devices together. This was needed since the CTS step removed them to put down the clock tree and also because the wires in the design are trial routes. After the design was routed, there were some errors that were reported. These errors were around the HERMES and the test structures blocks. The cause was that the wires being routed to the pins of these devices were being routed too close to the blockage layers. The problem was discovered to be that the pins from these blocks were off grid and that the tool tried to route the metal routes on grid as can be seen in Figure 4.6(a). Since the tool tried to route on grid it had to route in the areas that caused the error as seen by the red boxes. So to fix this

problem, the routing options were looked at. An option was found that sets routing on grid only as false. This fixed the problem since the tool was now allowed to route off grid to connect the off grid pins as can be seen in Figure 4.6(b). Once the routing was done, if any other steps that were taken to fix a problem that arose later, an engineering change order (ECO) was used. When an ECO was used, the design would be rerouted, but routing would only be run for wires that have problems or have been disconnected by the user or tool. This type of routing is what is wanted since the whole design doesn't have to be routed but only the problem areas.

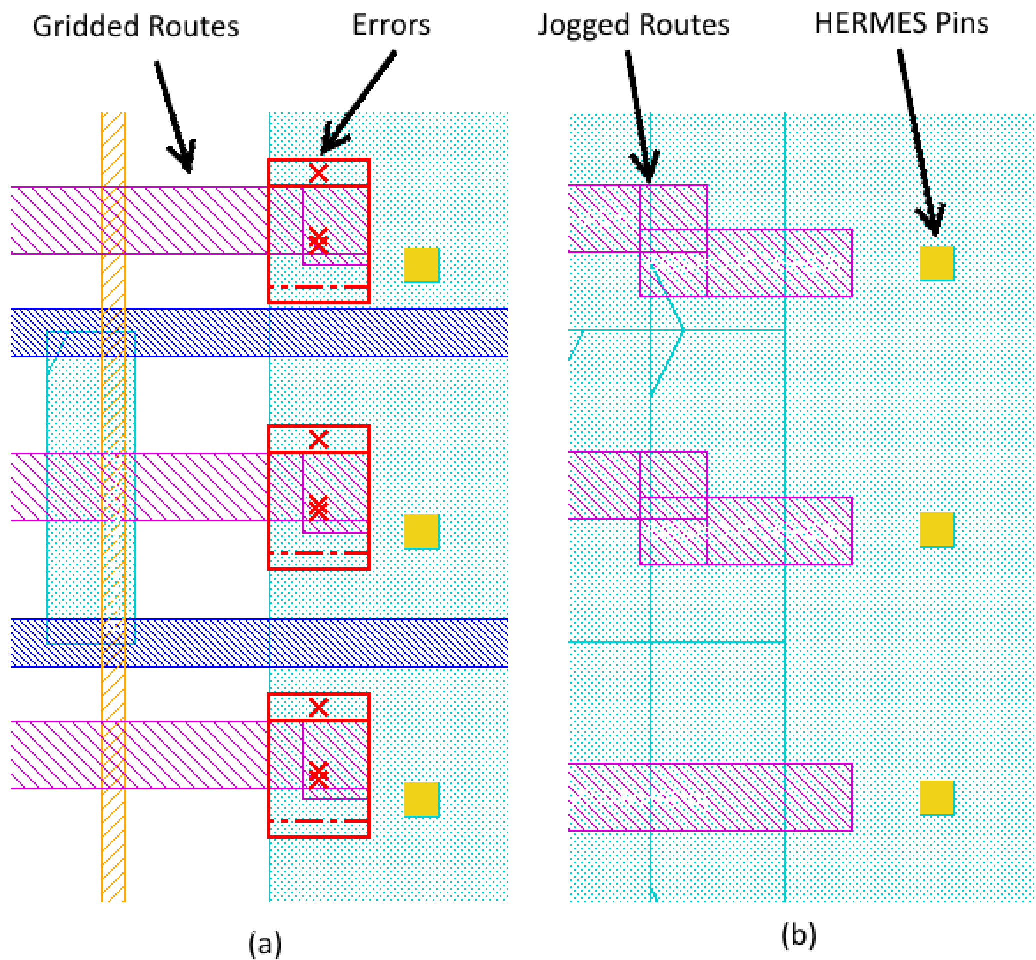


Figure 4.6– (a) Gridded Routes with Errors (b) Jogged Routes

With the design routed, the next step was to run more optimizations. Since the tool was having problems with the optimization, the decision was made to skip this step as being unnecessary since the timing for the design was still good and the tool would have frozen. So the next step was to add the tie cells to the design whose purpose is to tie the inputs of specific cells to a constant voltage. Once the tie cells were placed, DECAP and filler cells were added to fill in the remaining area between the cells in the design. Since the tie cells were placed in after the routing step, ECO routing is run to connect them to their corresponding blocks.

4.5. Verification

Once everything was filled and routed in the design, the entire design needed to be check for errors. Thus a verify connectivity command was run to make sure that all of the devices were connected to as to what is specified in the Verilog netlist passed at the beginning of the design. If the verifying of the connectivity didn't pass, an ECO is run to fix the connection issues. The next command that was run was to verify geometry. This command makes sure that all of the metals don't have DRC errors and that all of the blocks were placed correctly. After the command was run and there were no errors, the designs timing was checked one last time to make sure that there was no negative timing. The report from the final timing showed a positive slack as can be seen in Figure 4.7.

Setup mode	all	reg2reg	in2reg	reg2out	in2out	clkgate
WNS (ns):	1.993	N/A	1.993	N/A	N/A	N/A
TNS (ns):	0.000	N/A	0.000	N/A	N/A	N/A
Violating Paths:	0	N/A	0	N/A	N/A	N/A
All Paths:	153	N/A	153	N/A	N/A	N/A

Figure 4.7– Signoff Timing Summary

Since the final timing is positive, the design was streamed out into a GDSII file and a Verilog file was also created. This Verilog file is used for final verification through simulation. The simulation checks were used to make sure that all of the added devices such as buffers, inverters or any other logic that were added into the design by Encounter didn't change functionality. When the design was being verified through Verilog, an error was indicated saying that some of the cells weren't connected. The problem was chased down to the tie cells. The code still had hard ties that set the logic to a zero or a one. So the verification software saw this and interpreted this as the tie cells in the design as not being connected. The Verilog code was looked at and it was found that the logic to where the tie cells were supposed to be connected was actually connected to the tie cells. So the design was still good once the coded hard ties were removed. This could have been avoided if the tie cells were set up in the RC tool instead of Encounter.

The design was also checked by streaming the GDSII file into Virtuoso and checking the DRC's and LVS to make sure that the design didn't have problems that Encounter missed. While running DRC, a various amount of errors were reported. There was one major area that was showing errors as well as other errors in some random spots. The problem area was at the clock block, in which the power connections from the grid were too close to other metals in the block. Almost all of the connections from the power grid had to be removed to fix these errors. Since a good amount of the connections were removed, this could have hurt the power connectivity to the block. So the block was manually edited to move some of the wires to allow connections to the power grid. The other major DRCs that were reported were density errors which were fixed using fill after everything was verified.

Once all of the DRCs were fixed, LVS was run on the design using the Verilog file that was created from Encounter. When LVS was run it didn't finish since it threw an error that different power supplies were shorted together. With power nets shorted together, this became a big problem since this could take a long time to debug. This is because the power nets are huge since they spread out over the whole chip. It would have taken an extremely long time to try and track down where the short was, since the short could be anywhere in the sea of metal routes.

Luckily the tool had a short isolator to help find where the shorts were. It worked similarly to how the tool found the short by starting at the location of the power pin and following the metal route that it is put on. If another pin is encountered, it is compared to the first pin and if it doesn't match then a short is detected. So to find the short, one of the pins is used as the starting spot. The metal touching the pin was then highlighted by the tool. The tool was then stepped thru to follow the different metal layers that were changed, like the transition from metal 1 to metal 2. While the different metal layers were stepped thru, a spot was eventually found where some metal wires were overlapping.

This spot was where the custom cell brought power from the pad ring on the lower levels to the upper metals to be connected to the power ring. The reason that this short happened at this location was that the LEF that was generated was missing some blockage layers in certain areas. With these blockage layers not in the file, Encounter routed into the cell when the power grid was created essential shorting the routes together. So the metals that caused the short were cut out. Although when the tool was run again, the same type of short was encountered. This time all of the custom cells were looked at and any metals that were not supposed to be connected to them were removed.

LVS was then run again to see if all of the metal shorts are fixed. Unfortunately another short was encountered. This short was tracked down to the metal 1 that is in the HERMES block. This short was due to a power metal running through a custom cell that jogged the power metal slightly to allow connection to the devices in the cramped cell. The short could have been avoided when the power metals were put down in HERMES, in which a blockage layer should have been used like it was for the test structures and flash block. To fix the short the metal 1's that were running through HERMES at the top level were cut out so that only the metal 1's in HERMES were there. This was fine since HERMES had metal 1's in these areas along with the custom jog needed for the custom cell. With the short in the HERMES block fixed, no additional shorts were found when LVS was run again. Now that LVS was able to run, it was able to check the layout to see if the design was connected the way that it was intended. The results showed that this was the case and that there were no problems with the connectivity in the layout.

Now that the design was verified, there were some empty areas found that weren't supposed to be empty. These areas in the design were below HERMES and the test structures as can be seen in Figure 4.8. These gaps were the result of the blocks being offset in the Encounter flow to fix the power grid offset problem. To fix these empty areas, filler and DECAP cells were added by copying the fill and DCAP cells that were already in the design to the empty areas below the HERMES and test structures blocks. Since these filler and DECAP cells were added into the design, this essentially changed the amount of devices in the design. So these filler and DECAP cells were also added into the Verilog file so that when LVS is run again, these devices were matched instead of throwing errors about extra devices being found.

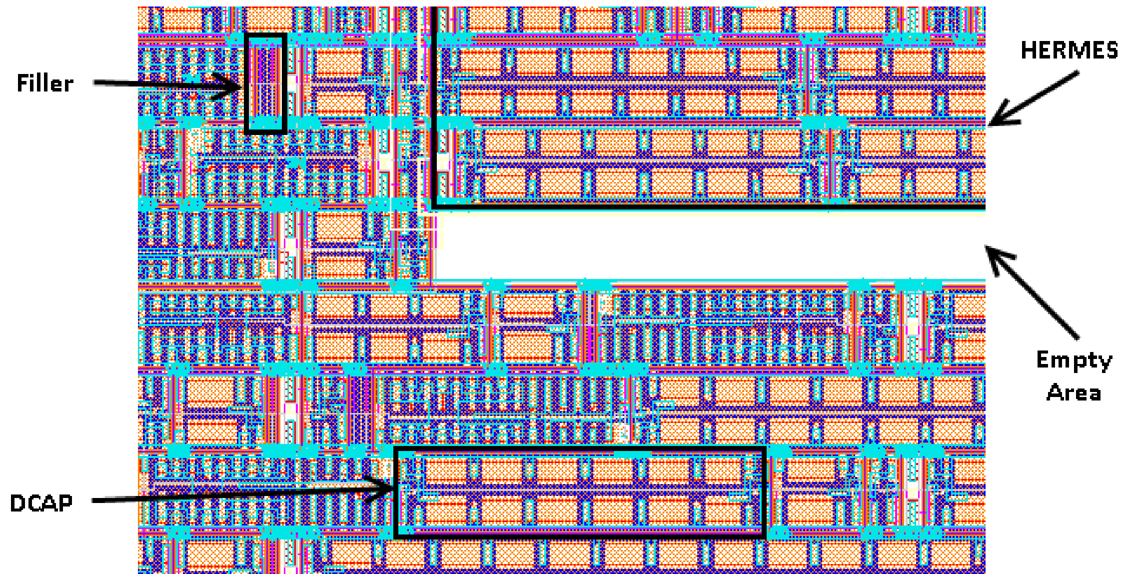


Figure 4.8– Empty Areas Under HERMES and Test Structures

4.6. After Verification

Once, the empty spaces under the HERMES and the test structures were filled in, there were a couple of things that needed to be done before the chip was handed over to Microchip. The first thing that was added to the design was a guard ring. This guard ring is a metal wall around the design which purpose is to protect the chip from contaminants that try to seep in from the sides of the chip. When this ring is obtained from Microchip, it is checked to make sure that it fit. It fit but a problem was found. The via 5's that connect metal layers 5 and 6 together were thick vias instead of small vias. The thick via 5's didn't meet the design specifications so a new guard ring had to be regenerated. This new guard ring that was obtained had the correct vias and was put in the design.

With the guard ring in, the next thing that was done to the design was to add the fill layers. This fill is used to meet the density requirements set by the fabrication plant by

filling the empty area of the design with every layer used. The way that the fill was added was by running the DRC tool with a fill rules file and exporting the results to a GDSII file. This was run with two different rule files, one for the metal layers and one for the other layers used. Once the two GDSII files were created, they were imported into Virtuoso and were put into the design. The DRC tool was run again but with the design rules file to make sure that the density errors seen before were gone. The results from the DRC run showed that no problems were reported with the fill added to the design.

4.7. Design Handoff

Now that everything in the design was finished, it was streamed out of Virtuoso into a GDSII file and sent to Microchip. Once they got the file, they swapped out the dummy flash module with their IP flash module. The design was then checked to make sure that the design was LVS and DRC clean once the real flash module was swapped in. Once LVS was run on the design, the results returned that everything was connected properly. Now that the design was LVS clean, DRC was run to make sure that no errors are reported.

Unfortunately errors were reported, around the outside of the Microchip flash modules that were swapped in. The errors were the result of the wires being routed into part of the flash modules where no blockage layers were used as can be seen in Figure 4.9. This was the case since the flash modules were manually created and these areas were missed from putting in these blockage layers. Also this could have been avoided if during the routing step of the flash block, the flag for allowing the routes to be routed off grid were used. Since the tool keeps the routes on grid, it routed into the flash module. These routes were routed too close to a wide power route inside the flash module.

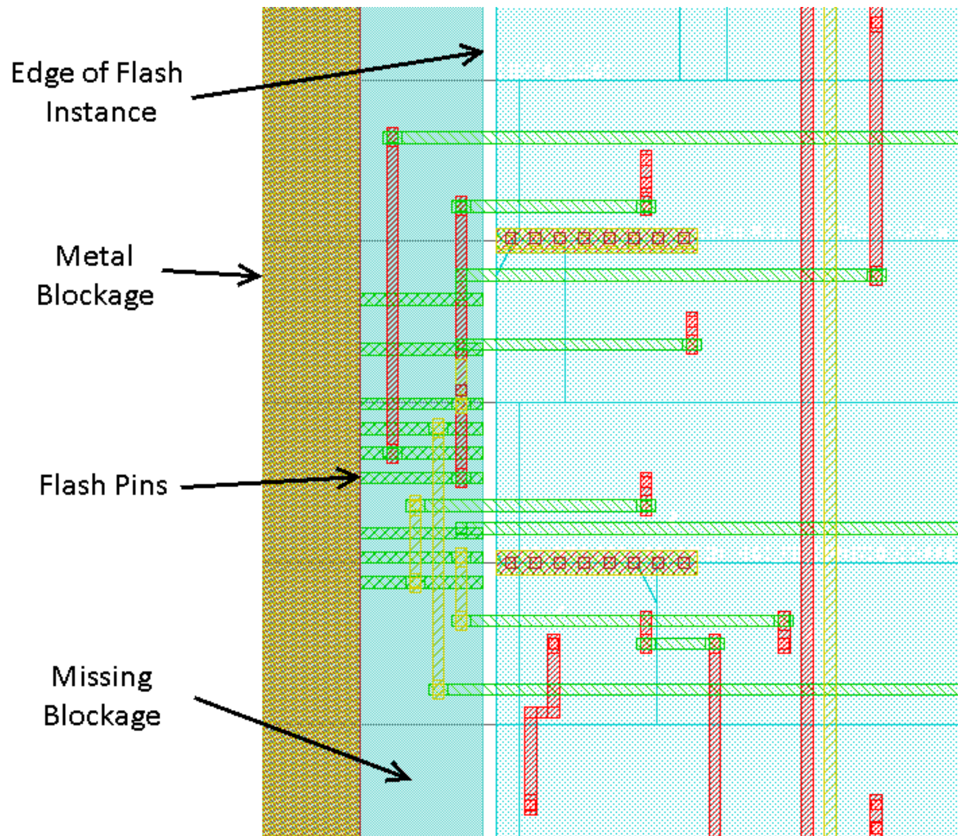


Figure 4.9– Flash Block Pins

Since these error locations were now known to be where the actual flash modules were put in, the routes next to the flash pins that were causing the errors were edited by moving them to fix the DRC errors. Almost all of the errors were fixed except for a couple that would have required a good amount of other routes to be moved. This would have taken a couple of days to correct, but these were waived since the deadline for the finished design was the next day. Once all of the major DRCs were fixed, the finalized design as can be seen in Figure 4.10 was resent to Microchip for them to add their flash modules back in. After Microchip had put the modules back in, they sent the completed design to the fabrication plant to be manufactured. This was the last thing that was done, so now all that was left was to wait for the design to be produced and sent back in a couple of months.

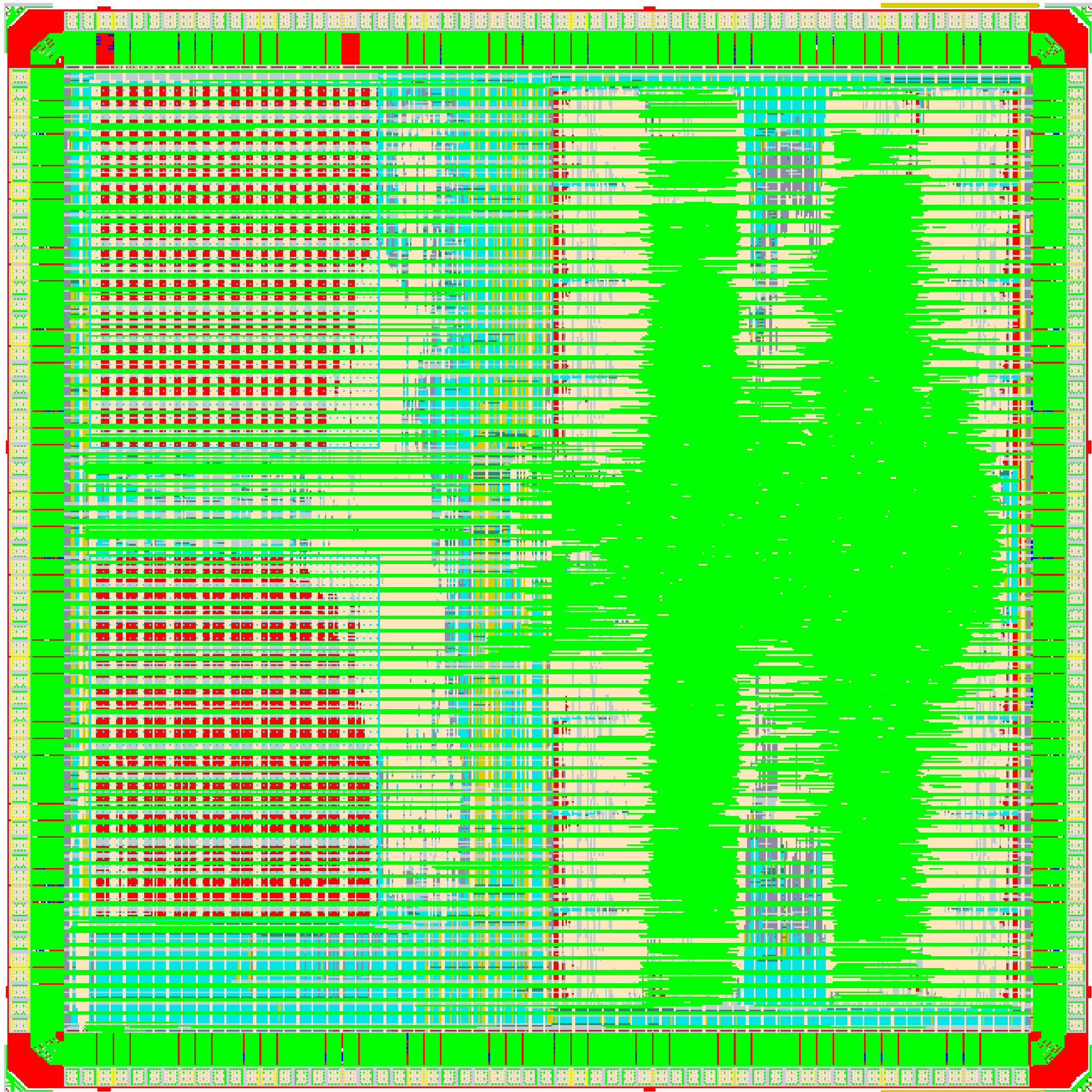


Figure 4.10– Finished Design

CHAPTER 5. PREVIOUS TEST CHIP WORK

This chapter talks about work that was done on a previous test chip which is similar to the one that's discussed in the previous Chapters. The main parts will talk about power gaters and a modified pad driver.

5.1. Power Gaters

The previous test chip was used to see if toggling the power of the flash module could bring it out of latchup that was caused by radiation. So to toggle the power to the flash modules, power gaters were created to toggle the power to these devices. When the power gaters were created, hspice simulations were ran to see what sizes the transistors would need to be so that their resistance would be less than 1Ω when under a 9mA load. This 1Ω or less resistance was needed since the data sheet for the flash modules had this required for the power connections.

The simulation results showed the sizes for a 1Ω resistance through the devices are presented in Table 5.1. The size of the n-type metal-oxide semiconductor (NMOS) with a body voltage of zero volts resulted in almost twice the size of the p-type metal-oxide semiconductor (PMOS). So another simulation was run with the NMOS having a body voltage of 1.2 volts to play with the body biasing. This change resulted in a drop in the size needed to almost the same size of the PMOS. Since the PMOS and NMOS sized were about the same, the smaller size was chosen for design simplicity.

Once the size of the power gater transistors were found, the sizes of the inverters to drive them were calculated and are show in Figure 5.1. The resulting calculations were found to be a 5 times stepup for each inverter for the NMOS power gater as seen in Table 5.2 and 3.5 times stepup for each inverter for the PMOS power gater as seen in Table 5.3.

Transistor Type	**Size**	**Applied Voltage**	**Vbias**
PMOS	4220 µm	2.5 V	2.5 V
NMOS	7040 µm	1.2 V	0 V
NMOS	4320 µm	1.2 V	1.2 V

Table 5.1 – Transistor Sizes

	Inv 1	**Inv 2**
PMOS	132 µ	660 µ
NMOS	54 µ	264 µ

Table 5.2 – NMOS Power Gater Inverters Transistor Sizes

	Inv 1	**Inv 2**	**Inv 3**
PMOS	90 µ	330 µ	1050 µ
NMOS	40 µ	125 µ	450 µ

Table 5.3 – PMOS Power Gater Inverters Transistor Sizes

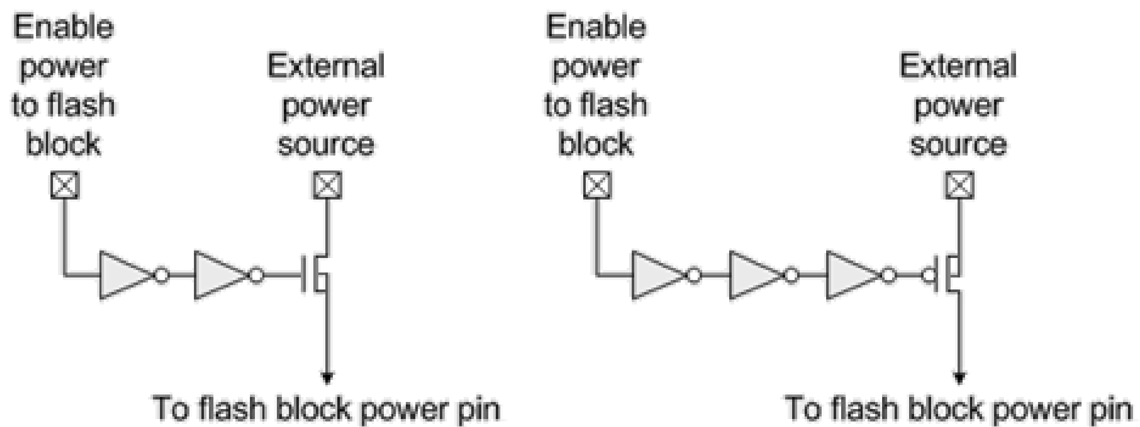


Figure 5.1 – Power Gater Circuits

The layout for the power gater transistor as seen as in Figure 5.2(a) were created so that when they were tiled next to each other, the conections were joined together and were spread throughout the connected devices. The connections across the metals, except for metal 3 are seen in Figure 5.2. Once all the transistors were tiled together and the size of the corresponding transistor created as what was calculated from Table 5.2 and Table 5.3, the alternating connections from metal 2 to metal 3 were added for the power connections as seen in Figure 5.3(a).

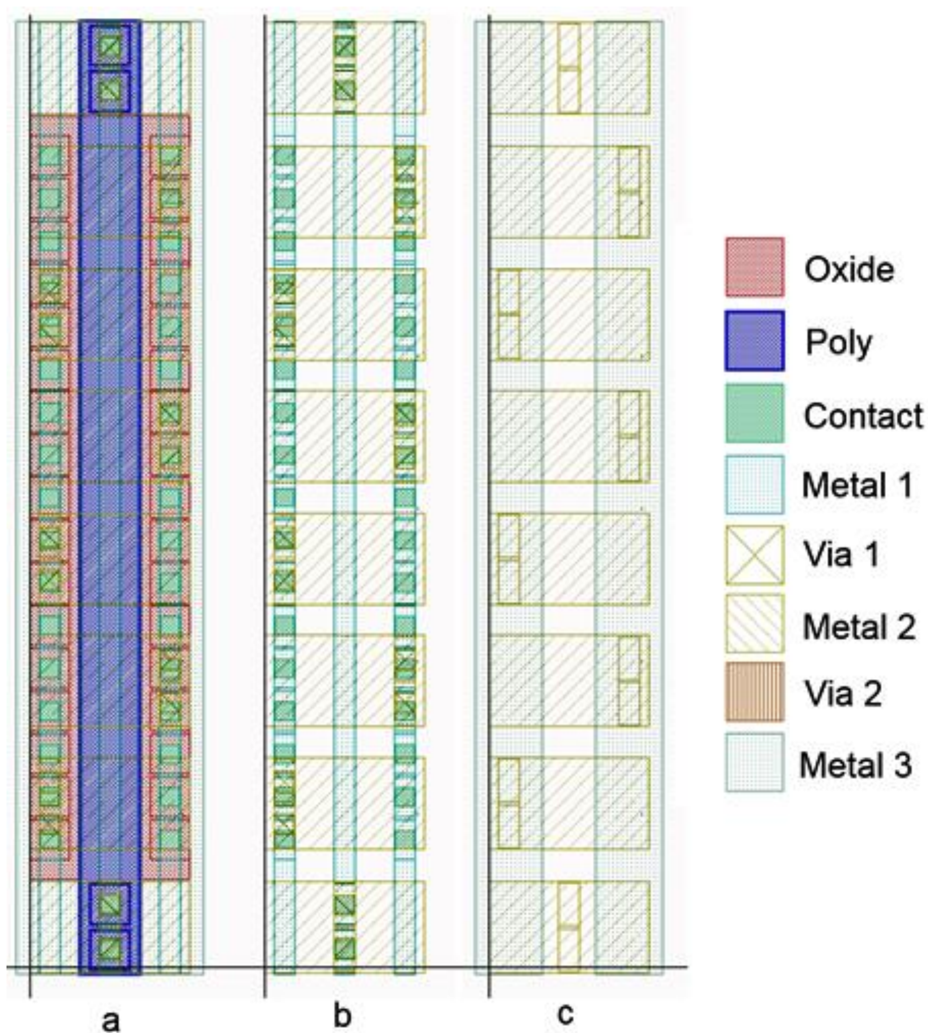


Figure 5.2 – (a) Transistor and Metal 1 (b) Metal 1 and 2 (c) Metal 2 and 3

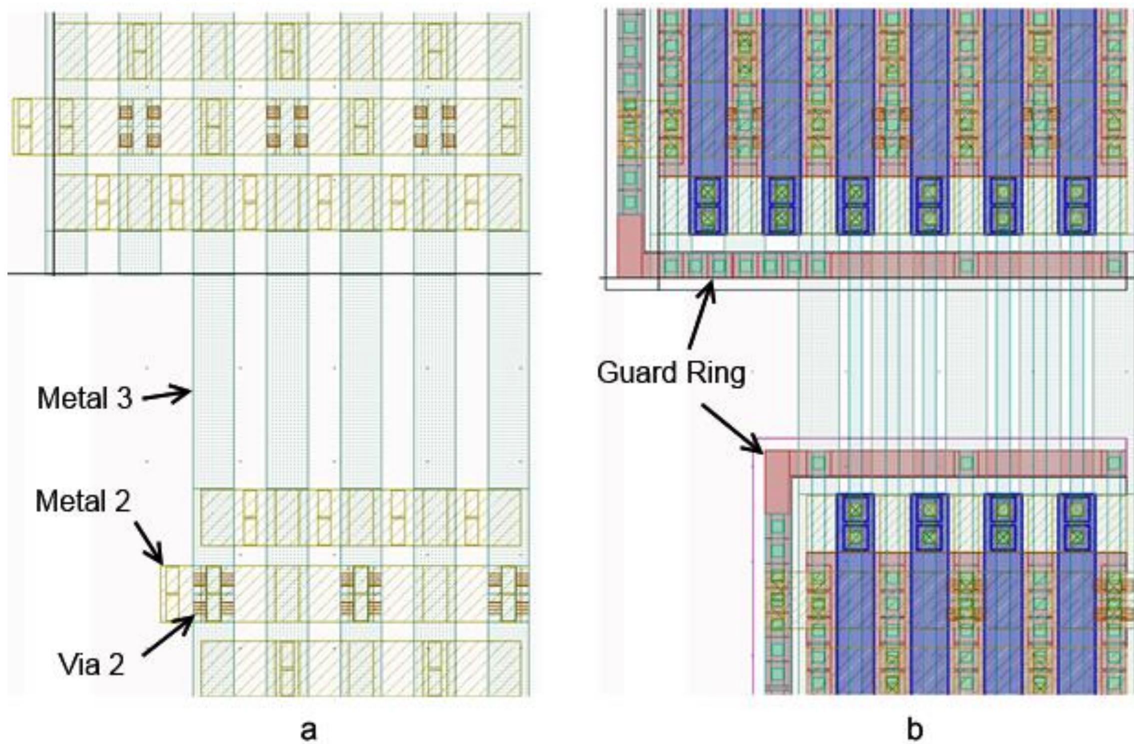
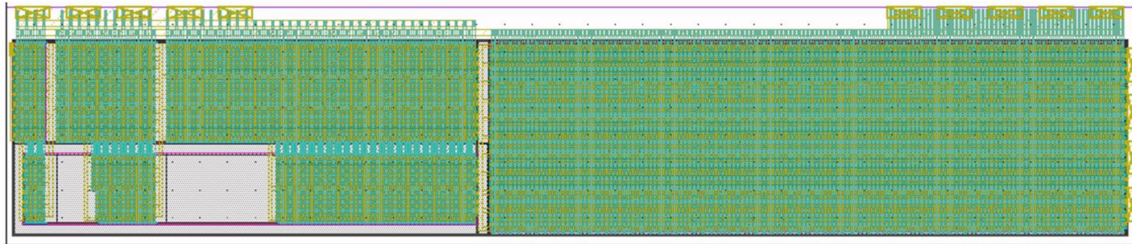


Figure 5.3 – (a) Metal 2 and 3 Connections (b) Guard Ring Added

Once the NMOS and PMOS were connected to create the inverter, a guard ring was created around the transistors as well as the power gaters transistor. This guard ring tied the substrates of the corresponding transistors to their intended voltages through metal 1 and 2 connections as seen in Figure 5.3(b). This was done to separate the transistors from the rest of the chip and to provide protection for the transistors from latching up. The inverters were then connected together to their corresponding power gaters as seen in Figure 5.4. With all of the componects connected together, connections for the power and control were created. The power connections use metal 2 and were made with wide connections to reduce the resistence.



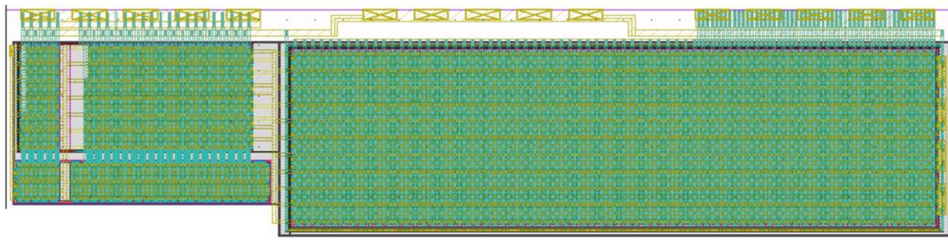


Figure 5.4 – PMOS and NMOS Power Gaters

5.2. Modified Pad Driver

Once the power gaters were completed, there needed to be a way to control these devices. During the simulation of these devices, two different voltages were used. One was the core internal low voltage and the other was the high voltage used in the pad drivers. The one that worked best for controlling the power gaters was the high voltage. The only problem with this approach was that there were no high voltage signals in the core where the power gaters were going to be used. So to get the high voltage as control signals into the core area of the chip, the pad drivers were modified to output the high voltage instead of the low core voltage.

The main uses of the pad drivers are to convert the high external voltages to low core voltages used by the internal devices and vice versa. This is done by using four inverters as seen in Figure 5.5. Inverters 1 and 2 on the left are using the high voltage

which translates the external voltage on the pad to the high voltage that these inverters are using. Inverters 3 and 4 work in the same way as inverters 1 and 2 in which they translate the high voltage at their inputs to the core low voltage that they use. So to get the high voltage out of the pad driver, inverters 3 and 4 were removed and inverter 2 was upsized to help drive the strength. The output of this driver was then connected to the power gaters to control them.

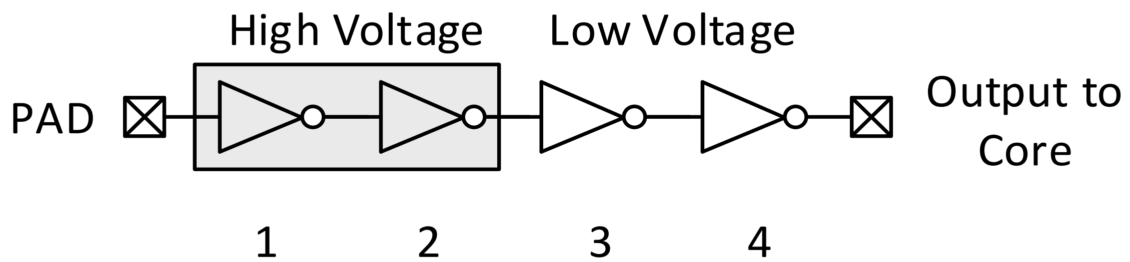


Figure 5.5 – Pad Driver Voltage Conversion

CHAPTER 6. CONCLUSIONS

This thesis has presented a couple of different flows for the creation of a finished test chip. The first flow was one laid the beginning base for the test chip by generating the pad ring which was used to get signals in and out of the chip. It also created the base for the power grid by creating the power ring that supplies power to the core devices. Even though problems were encountered during the flow, it can be improved on by incorporating the tie cells into the RC to when the Verilog was converted.

The second flow was the flash block flow that was generated to create a custom design for testing purposes. The flash block flow showed that a bad placement of the design can result in the design not being created properly for the specifications that are wanted. It also showed the flow for optimizations and workarounds to try and get the timings of the block to a positive slack if they were negative. This flash block flow helped make the top level flow easier since it followed this flow once everything was worked out.

The final flow that was discussed was the top level flow which took all of the major components and put them together to create a working test chip. This flow worked well except for some problems that were faced. These problems were the blocks that were offset from the power grid, wires being shorted together, and routes being too close to blockage layers. These problems could have been avoided if the options used were coherent as well as the blockage layers that were used correctly. Despite all the problems that were encountered, the chip was able to be created and successfully tapeout, as well as all of the pitfalls that were encountered to be used as a learning experience to be avoid while creating another test chip.

REFERENCES

[1] *EDI System Foundation Flow User Guide*. San Jose, CA: Cadence Design Systems, 2012. 12.

[2] Ferla, Giuseppe. *Advanced Topics in Microelectronics and System Design*. Singapore: World Scientific, 2000. 99

[3] *EDI System User Guide*, San Jose: Cadence Design Systems, 2012. 108.

[4] Gangadharan, Sridhar, and Sanjay Churiwala. *Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)*. New York, NY: Springer, 2013. 2.

[5] Brunvand, Erik. *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*, Boston: Addison-Wesley, 2010. 314

[6] B. G. Ko, H. C. Yoo, J. G. Park, and U. G. Paik. *Effects of Pattern Density on CMP Removal Rate and Uniformity*. Journal of the Korean Physical Society 39, 2001.

[7] *Encounter Digital Implementation System Text Command Reference*. San Jose: Cadence Design System, 2012. 616.