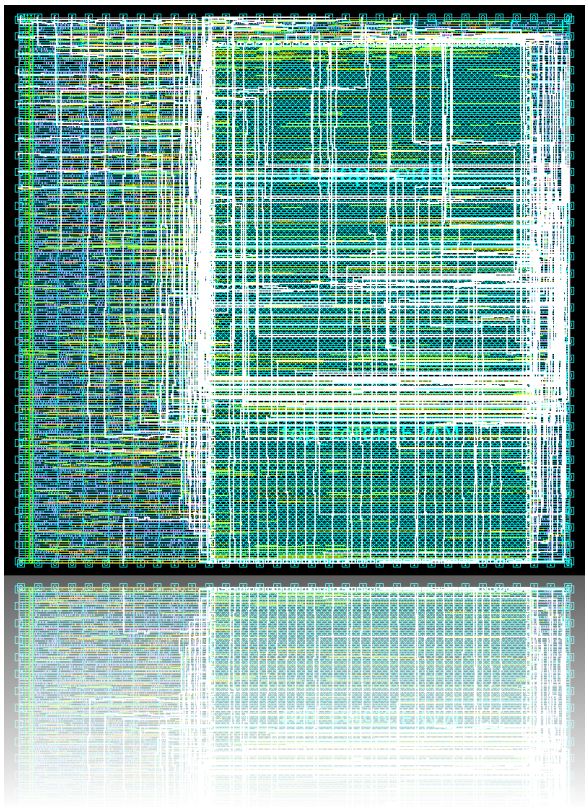
CHALMERS



Advanced Synthesis ECOs with Gate Array Fillers

Masters of Science Thesis in Embedded Electronic System Design

CHIRAYU SHAH

Chalmers University of Technology
Departement of Conmputer Science and Engineering
Gothenburg, Sweden August 2014
Master's Thesis 2014:1

Advanced Synthesis ECOs with Gate Array Fillers

Chirayu Shah

Examiner:Professor Per Larsson-Edefors

VLSI Research Group
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone:+46 (0)31-772 1700

Department of Computer Science and Engineering
Göteborg, Sweden August 2014.

Abstract

Engineering Change Orders (ECOs) are commonly used in the Application Specific Integrated Circuits (ASIC) industry to either fix design bugs or to add new features to the design after first tape out. Due to rapid increase in the design complexity the metal-mask ECOs have become inevitable. Generally, redundant standard cells, known as spare cells, are used to realize such type of ECOs. However, these cells suffer from a major drawback of having predefined functionality and location [1]. As a result, their use becomes limited. To overcome this inflexibility, gate array type spare cells are used.

As the gate array spare cell is configurable, it opens up new possibilities of doing big ECOs. On the other hand, most ECO algorithms offered by the commercial tools are not smart enough to handle such type of ECOs. In order to overcome this limitation, designers prefer using conventional ASIC flow rather than realizing such big changes as an ECO.

In this thesis, a methodology to implement large scale ECOs is presented. This methodology aims to overcome the existing limitations of using the ECO algorithms by incorporating conventional ASIC flow algorithms to perform an ECO. The methodology has been implemented using gate array type cells. Simulation results show that for a medium sized design (12k gates) the implementation consumed 60% more dynamic power and occupied 75% more area as compared to the using regular standard cells. However, if a proper gate array library containing all the required cells is used for mapping, this number would be reduced to 30% and 35% respectively. On the other hand, due to advantages like faster time to market and small manufacturing costs [2] the area and power overhead incurred get compensated.

Keywords: ECO, metal-configurable gate array spare cells, ASIC flow, metal-masks ECOs, spare cells.

Acknowledgements

I am thankful to following people for their contribution:

- My supervisor at Nordic Semiconductor ASA, Dr. Johnny Pihl (Senior R&D Engineer). Without him this master thesis project wouldn't be possible. He helped me regarding the practical arrangements as well as technical matters. He is the "producer".
- My second supervisor at Nordic Semiconductor ASA, Dr. Oyvind Granheim (Senior R&D Engineer). He is "the" guy who has experience with ECOs. He always showed me the right direction whenever needed and supported me through out the project. He is the "director".
- My examiner at Chalmers University of Technology, Professor Per Larsson-Edefors for his constant support throughout the thesis.
- The System Integration group at Nordic Semiconductor for their technical support and friendly behavior towards me.
- My parents for providing the opportunity, support and help through my Master's Studies.

Contents

List of Figures

List of Tables

List of Abbreviations

ASIC	Application Specific Integrated Circuits
CTS	Clock Tree Synthesis
DRC	Design Rule Checks
ECO	Engineering Change Order
FPGA	Field Programmable Gate Array
GA	Gate Array
GDS	Graphic Data System
HVT	High Threshold Voltage
ICC	Synopsys IC Compiler
I/O	Input/Output
NMOS	N-type Enhancement mode Metal Oxide Semiconductor
PMOS	P-type Enhancement mode Metal Oxide Semiconductor
RTL	Register Transfer Level
TCL	Tool Command Language

1

Introduction

ECO commonly referred to as Engineering Change Order is the process of making modifications to the design without running the entire ASIC design flow again. These modifications can either be in the form of a bug fix or adding a new feature. Furthermore, these could be needed either before the chip/design has been taped out (**Pre-Mask**) or after the chip has been taped out (**Post-Mask**).

In the pre-mask stage the ECOs are usually performed to save design time [6]. In this stage the design is already placed and routed but not taped out, so new cells can still be placed. Performing the entire flow again for a small change is time consuming. Thus the need for an ECO. The ECOs can be further divided into two types in this stage.

- **Functional ECO**: This type of ECOs deals with making changes in the logic functionality of the design.
- **Non-functional ECO (timing ECO)**: This type of ECOs deals with changes that involve timing and signal integrity improvement.

In the post-mask stage the ECOs are usually performed to save design cost [6]. In this stage, the chip has been taped out, which means that the transistor masks have already been made. As the masks have been made, only the metal layers could be altered to implement the changes to the design. This is due to the fact that the cost of making metal-masks is much less compared to making all the masks from scratch [7]. Such process in which only metal layers are used to perform the change is known as **metal-only ECO** [1]. Such type of ECO is performed using **Spare Cells** [8][9]. These cells are spread over the design before the chip is taped out.

The focus of this thesis is on carrying out functional ECOs in the post-mask stage

using **gate array** type spare cells. This chapter will discuss the motivation behind, focusing on post-mask stage, using gate array cells as spare cells and finding new ECO methods.In addition to this, the chapter will describe the problem, the objectives of this thesis, and the outline of the method used to fulfill these objectives.

1.1 Motivation

With the increase in the design complexity, more bugs escape the pre-silicon validation and are found post-silicon [7]. As a result post-mask ECOs are increasing. Also, to fix these late failures in a short time and with a low mask cost [10], metal-only ECOs are essential. In order to perform these ECOs, redundant cells, also known as spare cells are required. When a new functionality is to be added or a failure is detected post tape out, the inputs and outputs of these spare cells are rewired according to the intended functionality in order to use them.

In the conventional flow, these spare cells are of type regular standard cells. As a result they have predefined logic functionality such as AND,OR etc. Furthermore, once added, the amount of spare cells as well as the location of these in the design become fixed [1]. This inflexibility could result in an unfixed design. For example, under an ECO specification (see section 2.2), if the desired gates needed to fix the bug are not available in the spare cells, that desired functionality must be deleted or the process of design must be done from scratch. In addition, this type of spare cells always draw leakage current as their inputs must be tied to ground to avoid gate floating [1]. Due to such disadvantages only limited amount of spare cells are inserted in the design.

In order to address this issue and overcome the disadvantages of conventional spare cells, a new type of spare cells called **metal-configurable gate array spare cells** [11][1] is used. The major advantage of using this type of cells as spare cells is that the desired functionality can be made just by re-configuring the metal layers of these cells. Furthermore, as a gate array type cell is composed of unconnected transistor gates [2], these cells consume no leakage current. On the downside, their configurable nature calls for increased area and power overhead [2]. But, as this overhead gets compensated by faster time to market and smaller costs, this type of cells is used in this thesis to realize an ECO.

Usually the amount of changes to be made as an ECO is not big [12]. This is due to the fact that it is not possible to have all the required functionality spare cells in the first place to implement very big changes. Secondly, it becomes impractical and time consuming task to manually modify netlist having large number of changes. Due to these reasons most incremental placement algorithms are not made smart enough to figure out such big ECOs [12]. For example, the Synopsys IC Compiler(ICC) tool would just crash if the ECO changes exceeded a certain threshold. Because programmable gate array spare cells overcome the disadvantages of conventional spare cells, there arises a

need to devise a new ECO methodology that would work irrespective of size of an ECO.

1.2 Problem Description

Using metal-configurable gate array cells as spare cells opens up whole new possibilities like adding new features to the design as well as fixing bugs after the chip has been taped out. Moreover, using only metal layers reduces the mask cost and as a result also the design time drops. In a response to this new technology, Nordic Semiconductors proposed this thesis topic in order to explore the possibilities of metal-configurable gate array cells.

The problem can be understood if we consider Figure 1.1. The left side of the Figure shows the original taped out chip that contains a spare area. The right side of the Figure shows the synthesized netlist representing the new functionality to be added. The problem can be described as follows. Given a new functionality in the form of synthesized netlist and a taped out chip having a spare area filled with gate array type cells, investigate new methods to add this new functionality onto the spare area as an ECO. More importantly, the method should be such that the ICC tool should be able to carryout a post-mask ECO using gate array type cells irrespective of the functionality size.

It is assumed that there exists enough space on the taped out chip to accommodate the new functionality as an ECO. This space can either be created by swapping (see section 4.5.1) or having a dedicated spare area (as in Figure 1.1) inside the chip.

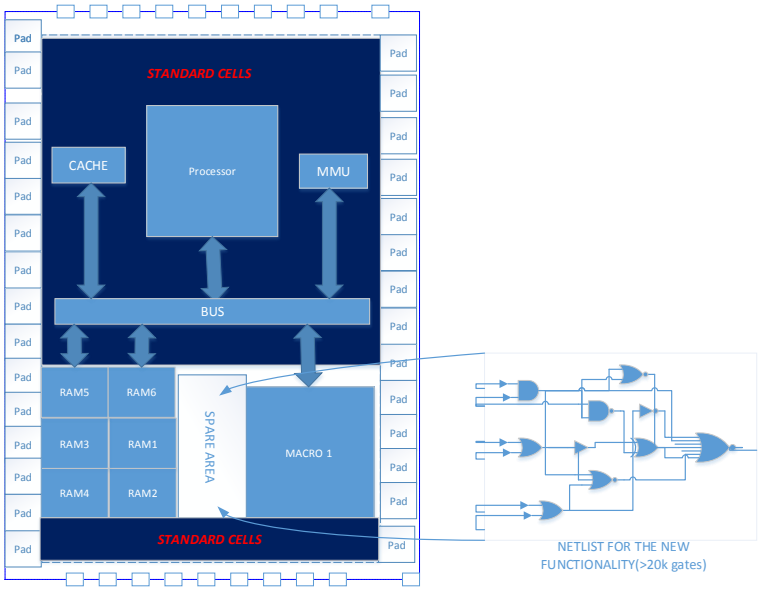


Figure 1.1: Figure representing the ECO Problem

1.3 Objectives

The following are the objectives of this master thesis.

1. Based on ICC's spare cell methodology [13], develop a methodology to insert gate array type fillers inside the chip and carry out a small ECO in the post tape out phase.

2. Using the above methodology as base, propose and explain a new ECO methodology that works on big ECOs and uses gate array type cells.

3. Evaluate the use of gate array type cells in the proposed methodology and compare it against regular standard cells in terms of:

   (a) Area consumption and gate count.

   (b) Power consumption (Dynamic and Leakage).

   (c) Timing requirements.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

Chapter-2 Gives the background information of the important thesis terms. It includes explanation of conventional ICC ECO flows, basic concepts of gate array type

cells and using them as spare cells and advantages of using gate array spare cells over the conventional spare cells.

Chapter-3 Presents methodology to insert gate array filler cells and perform a simple post-mask ECO using them.

Chapter-4 Proposes the new ECO methodology named Advanced Synthesis ECO methodology and explains the usage of gate array type cells to implement the new functionality and the usage of gate array type spare cells to integrate the functionality as a post-mask ECO.

Chapter-5 Introduces the Structured ASIC design methodology and compares it's implementation flow with the newly proposed methodology.

Chapter-6 Presents the evaluation results by using gate array type cells in the methodology and compares them against the regular standard cells.

Chapter-7 Provides conclusions of the work presented and gives suggestions for the future work.

2

Background

This chapter will provide background information of the keywords present in the title of this project. This includes keywords like ”ECO”,”gate array (GA)”,”fillers”. To start with, section 2.1 will describe the need for an ECO at several ASIC design stages. In section 2.2, the ICC's ECO flows are compared. section 2.3 explains the pre-mask ECO flow in detail. The subsequent section discusses the post-mask stage using conventional spare cells. Section 2.5 introduces the GA cells and their configuration process and it is followed by a section that points out the advantages of using GA filler cells as spare cells. At the end of the chapter, the summary is presented.

The flows and the corresponding commands mentioned in this thesis are based on the Synopsys IC Compiler. Only the basic information regarding the commands is mentioned here. For detailed descriptions of these commands, the reader is encouraged to refer to the tool user manual [14]. In case of different tool, the concepts should remain the same but the corresponding commands would differ.

2.1 ECO in the context of ASIC design

Figure 2.1 shows the conventional ASIC design flow on the top, and the various stages where there could be a need for an ECO, at the bottom. After every stage except after tape out, one can perform pre-mask ECO to avoid re-running the previous stages; thus saving time. The post-mask ECO block is shown in green as this stage is the focus of this thesis.

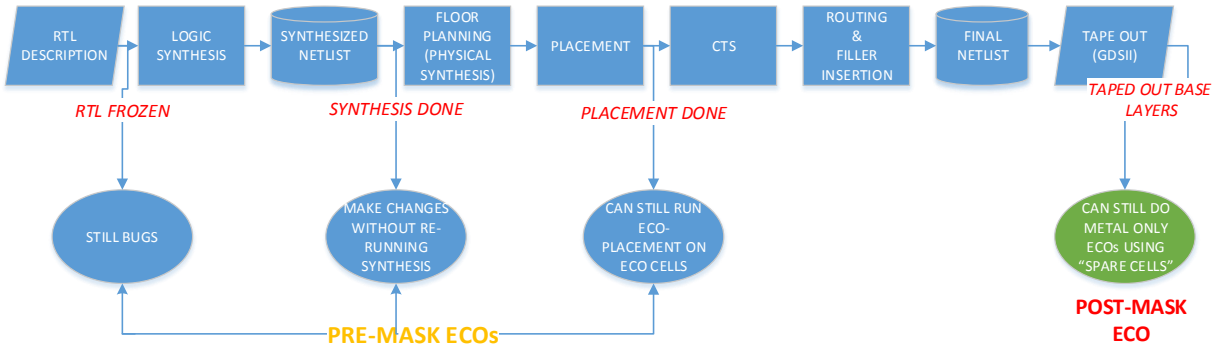


Figure 2.1: Purpose of an ECO in the context of the ASIC design flow

It can be seen from the Figure 2.1 that, once the Register Transfer Level (RTL) code is frozen, in case of a bug found in the code, the ECO could be performed to fix that bug. Moreover, even if the synthesis has been carried out, by modifying the netlist manually the necessity of re-running the synthesis can be avoided. Furthermore, if the standard cells have been placed, the new cells can still be added by performing the ECO placement. Finally, after the tape out by doing metal-only ECOs, the necessity to change the base mask layers could be avoided. Hence, saving cost and time.

2.2 IC Compiler ECO flows comparison

Figure 2.2 compares the ICC's ECO flows. If the ECO has to be performed in pre-mask stage then the corresponding flow in ICC is known as **Unconstrained ECO** flow else, known as **Freeze Silicon ECO** flow for the post-mask stage.

In the unconstrained ECO flow, the placement of the design is not fixed, so new cells can be added and existing cells can be moved or deleted. As a result, spare cells are not needed. After deciding the ECO changes to be made, the original netlist is manually edited to implement those changes. Subsequently, the power and ground connections should be updated if the new cells have been added. Furthermore, the new cells to be added as an ECO are placed using the incremental placement known as ECO placement, followed by incremental Clock Tree Synthesis (CTS) and the ECO route. One additional step of fixing the routing Design Rule Checks (DRC) is performed if the ECO route was not able to remove all the DRC. The amount of iterations the tool has to perform in order to minimize the DRC is known as routing closure [14].

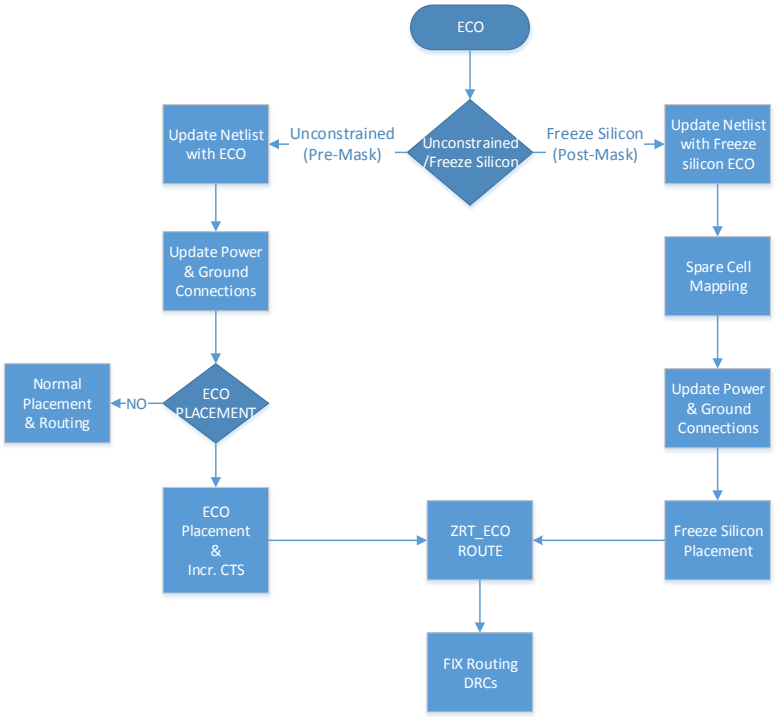


Figure 2.2: Comparison between ICC's ECO flows

In the freeze silicon ECO flow, the placement of the design is fixed so no new cells can be added. As a result, the spare cells that were added prior to running ECO process have to be used to perform the desired change. The design netlist is updated according to the freeze silicon ECO requirements first. Then the spare cell mapping is carried out which in conventional sense would be to find appropriate functionality cells in the design. This mapping could either be performed automatically by the tool or manually by hand. Subsequently, the power and ground connections of the spare cells used should be updated. The existing cells which are no longer required would not get deleted, rather would be made as spare cells. The ECO routing step and the DRC fixing step is similar to the unconstrained flow.

Methods to represent an ECO

Whenever a bug is encountered in the design , a new specification is defined to remove that bug. This specification can be called as an ECO. In ICC, irrespective of the type of ECO, the ECO changes can be represented using two methods.

1. **By using Verilog Netlist.** : The new netlist which is formed by editing the original netlist is called **ECO netlist**. Example 2-1 shows that ECO netlist.

Example 2-1: ECO changes described using Verilog Netlist

```
//Original Netlist.
......
//ECO changes to be made
MYOR2_D1BWP7THVT OR2_eco (.Z ( cnt_0_or_z ) , .A1 ( n18_inv ) ,
                          .A2 ( cnt[0] ) ) ;
MYINV_D1BWP7THVT INV1_eco (.I ( n18 ) , .ZN ( n18_inv ) ) ;
MYNR3_D2BWP7THVT NOR3_eco (.A1 ( cnt[2] ) , .ZN ( cnt_1_nor ) ,
                          .A3 ( cnt[1] ) , .A2 ( cnt[0] ) ) ;
```

2. **By using Tool Command Language (TCL) commands** to describe the changes.
   Example 2-2 shows the ECO change described using TCL command.

Example 2-2: The ECO changes described using TCL commands

```
create_net cnt[0]_or_z

disconnect_net cnt[0] [get_pins {U98/I}]
create_cell OR2_eco MYOR2_D1BWP7THVT

connect_net cnt[0] [get_pins {OR2_eco/A2}]
connect_net cnt[0]_or_z [get_pins {U98/I}]
connect_net cnt[0]_or_z [get_pins {OR2_eco/Z}]

#connect_net n18 [get_pins {OR2_eco/A1}]
create_cell INV1_eco MYINV_D1BWP7THVT

###For adding an inverter with drive 1 on the n18 net
create_net n18_inv
#disconnect_net n18 [get_pins {OR2_eco/A1}]

connect_net n18 [get_pins {INV1_eco/I}]
connect_net n18_inv [get_pins {OR2_eco/A1}]
connect_net n18_inv [get_pins {INV1_eco/ZN}]

#adding an 3-input nor gate with drive 2 to cnt_reg_1 and U87
create_net cnt[1]_nor

create_cell NOR3_eco MYNR3_D2BWP7THVT

disconnect_net cnt[1] [get_pins {U87/A1}]
connect_net cnt[1] [get_pins {NOR3_eco/A3}]

connect_net cnt[1]_nor [get_pins {NOR3_eco/ZN}]
connect_net cnt[1]_nor [get_pins {U87/A1}]
connect_net cnt[0] [get_pins {NOR3_eco/A2}]
connect_net cnt[2] [get_pins {NOR3_eco/A1}]
```

The cells marked in red in the above examples are the new cells to be added in the design also known as **ECO cells**. In case of a pre-mask ECO, these cells would be added from the cell library, whereas in the case of post-mask ECO, spare cells having similar functionality would be required.

2.3 Pre-mask ECO flow

The aim of this section is provide detailed explanation of all the necessary steps involved in performing a pre-mask ECO. As said earlier, in this type of ECO, as the design has not been taped out, cell placement can be changed.

Figure 2.3 shows the detailed flow for doing an pre-mask ECO. The Figure assumes that the design has already been placed and routed and has spare cells. The part of the flow which is marked in red lines is the ICC's unconstrained ECO flow (as in Figure 2.2).

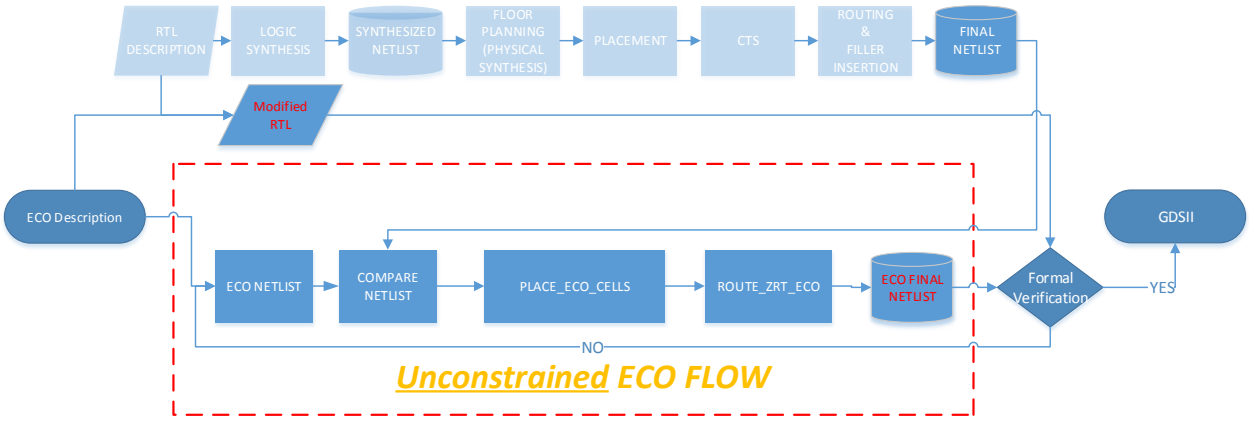


Figure 2.3: Complete pre-mask ECO flow

The basic flow can be divided into 6 main steps.

Step-1 Describe an ECO specification.

Step-2 Capture the specification.

Step-3 Compare the ECO netlist with the design netlist.

Step-4 Update the placement with ECO changes.

Step-5 Perform the ECO routing.

Step-6 Insert filler cells.

Step-1: Describe an ECO specification

An ECO specification is a manually drawn diagram to describe the changes to be made. The specification can be made by exploring the original netlist using the graphical tools such as Design Vision. This tool helps to explore a portion of the schematic to find the nets to be removed and to add the new nets. Figure 2.4 shows an example of manually drawn specification. The blue marked nets and cells are the changes to be made to the existing design. The crossed nets are the nets to be removed.

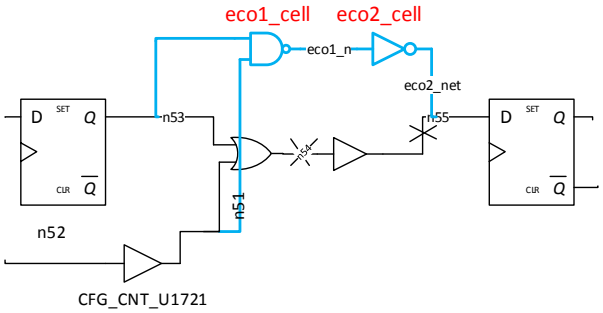


Figure 2.4: An ECO specification

Step-2: Capture the specification

As mentioned in the section 2.2, there are two ways to represent the ECO specification. Example 2-3 shows corresponding ECO netlist of the specification shown in Figure 2.4.

Example 2-3: ECO netlist (ECO by Verilog)

```
//Original Netlist.
MYBUF_D1BWP7THVT CFG_CNT_U1721  (.I ( n52 ) , .ZN ( n51 ) ) ;
 ......
//ECO changes to be made
MYND_D2BWP7THVT eco1_cell (.Z ( eco1_n ) , .A1 ( n51 ) ,
                           .A2 ( n53 ) ) ;
MYINV_D1BWP7THVT eco2_cell (.I ( eco1_n ) , .ZN ( eco2_net ) ) ;
```

Step-3: Compare the netlists

In this step, the tool compares the design netlist (original netlist) with the ECO netlist and displays the ECO changes. From this output, one can also verify whether the tool identified all the intended changes.

If the changes are made by directly editing the original netlist like in example 2-3 then the command used in the ICC tool is:
`eco_netlist -by_verilog_file "verilog file"`

If the changes are described using the TCL file then the command used in the tool is: `eco_netlist -by_tcl_file "TCL file"`.

Step-4: Incremental placement (ECO placement)

If the standard cells are fully placed, the placement of the new cells is performed using incremental placement command. The command `place_eco_cells` is used for this purpose. This command uses the spare area in the design to place the new cells.

Figure 2.5 shows the placed and routed design having spare cells. Figure 2.6 shows the design after incremental placement is performed. It can be seen from the latter that the ICC tool did not use spare cells for placing new cells. Rather, it took new cells from the library and placed them onto the spare area. The two instances that were added according to the example 2-3 are circled.

Figure 2.5: Before the ECO Placement [3]

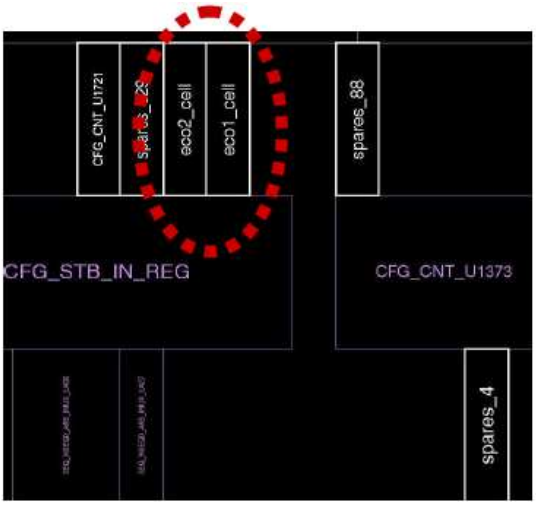


Figure 2.6: After the ECO Placement [3]

Step-5: ECO Routing

ECO routing is performed only if design has already been fully routed. The command used in the ICC to perform this type of routing is `route_zrt_eco`. The command reroutes the modified nets and remove the dangling nets after an ECO. Moreover, it also fixes the remaining routing DRC. Dangling nets are the old nets which are no longer driven by any pins/ports. Such type of nets are created as a result of ECO changes.

Figure 2.7 shows the explanation of what happens during an ECO and after the ECO routing. During the ECO, the tool deletes old wires or adds new wires according to the ECO specification. The dangling wires created due to this can be seen in the top Figure.

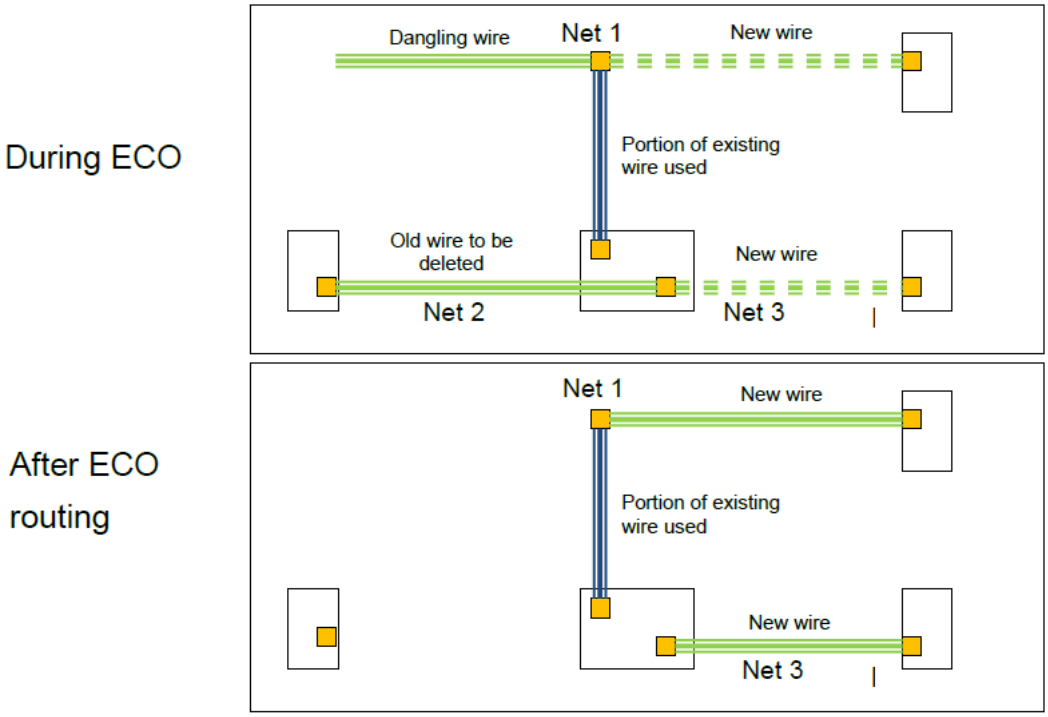


Figure 2.7: The design state during ECO (top) and after ECO routing (bottom) [3]

During the ECO routing, the router touches only those nets which have been modified due to the ECO. Thus the critical nets in the design would be preserved. The bottom Figure shows the state of the design after the ECO routing. The physical connections made by the ECO router can easily be recognized from the Figure.

Step-6: Insert filler cells

After achieving the routing closure in the ECO routing, filler cells are used to fill the remaining gaps in the design. Command `insert_filler_cells` is used in ICC to insert fillers. There are two advantages of filling.

- Ensure the continuity of the power and ground rails in the floorplan.
- Ensure the continuity of the N+/P+ well.

Figures 2.8 and 2.9 show a routed design before and after the filler cell placement. Gaps in the standard cell rows can be seen in the former Figure. After filling the gaps, the resulting design can be seen in latter Figure. Different size filler cells used according to the size of gaps are highlighted in latter.

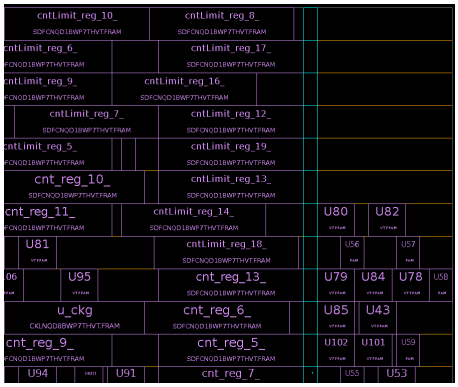


Figure 2.8: Placed and Routed design having gaps in the rows

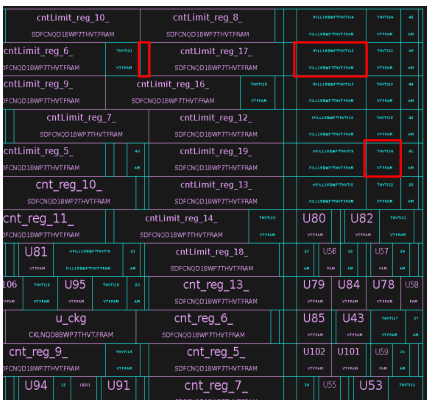


Figure 2.9: Placed and routed design after filling gaps with filler cells

2.3.1 Formal Verification

Formal Verification is performed to detect the differences that might have been introduced into the design during it's implementation. It is used to verify the logical equivalence between the two designs without the use of test vectors.

As shown in Figure 2.10, ECO final netlist (implemented output) that is generated after an ECO has to be verified against the Modified RTL (expected output). They should be equivalent in order for the ECO to be successful. Otherwise the ECO netlist (see Figure 2.3) has to be modified until successful verification is achieved. Tool Synopsys-Formality was used to perform this. This step is explained using an example in section 4.5.1.

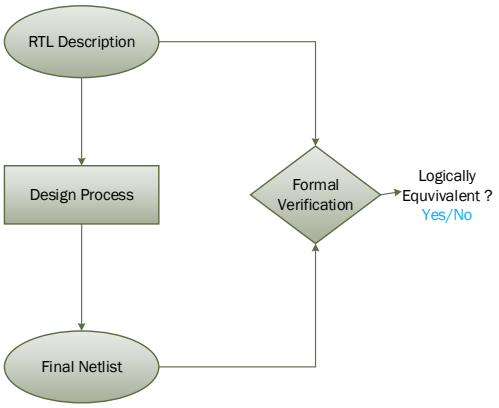


Figure 2.10: Formality[4] verification flow

2.3.2 Complete ECO flow script

Appendix A gives the script for doing the unconstrained ECO flow. It provides necessary commands to perform the basic flow. The script assumes the case where the design is already placed and routed and there is a need for a pre-mask ECO.

2.4 Post-mask ECO using conventional spare cells

In this section, the freeze Silicon ECO flow of ICC is explained in detail. The aim of the section is to show how the conventional spare cells are used to carry out a post-mask ECO.

As said earlier, in the post-mask ECO, the placement of the design is fixed, so new cells cannot be added and old cells cannot be moved or deleted. As a result, the metal-only ECO is essential. In order to perform this, spare cells which in conventional sense are logic gates would be needed. These cells could be added before or after the placement, so that they can be used to accommodate late arriving changes.

The most important design decision in using the conventional spare cells is to decide upon the type of cells to be used as spare cells [8][9][15] and ways to spread them inside the design [16][17][7]. In ICC, there are two ways to insert spare cells in the design. One is by manually instantiating them in the **Verilog netlist** and another is by using `insert_spare_cells` command. The former should be use to insert them before the placement and the latter should be after the placement. Furthermore, once the spare cells have been inserted, they can be spread inside the design either manually, using `spread_spare_cells` command, or automatically by the tool.

Figure 2.11 shows the representation of a design after inserting the spare cells. Each spare cell depict a predefined logic gate that can be activated by connecting it's inputs and outputs according to the ECO specification.

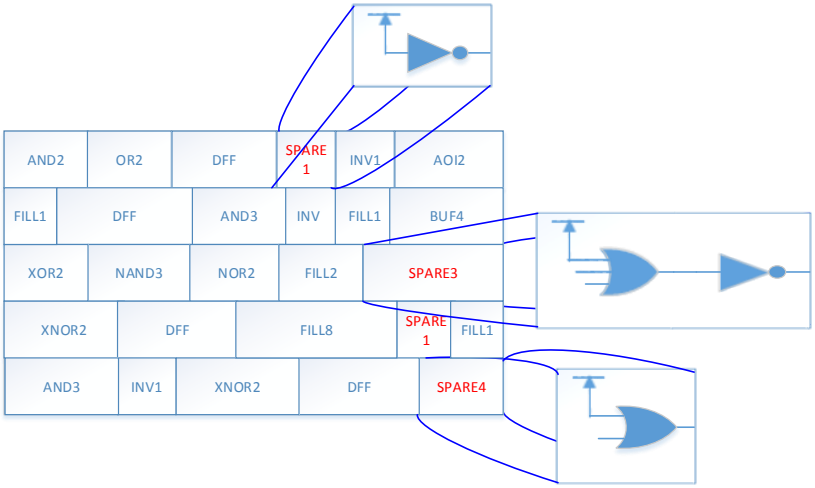


Figure 2.11: Representation of a design having conventional Spare cells

Figure 2.2 shows the basic freeze silicon ECO steps. One additional step of feasibility analysis is performed in this flow before spare cells are mapped. The main purpose of doing feasibility analysis is to find out whether there are enough spare cells in the design to perform the ECO. This step is not required in the pre-mask ECO.

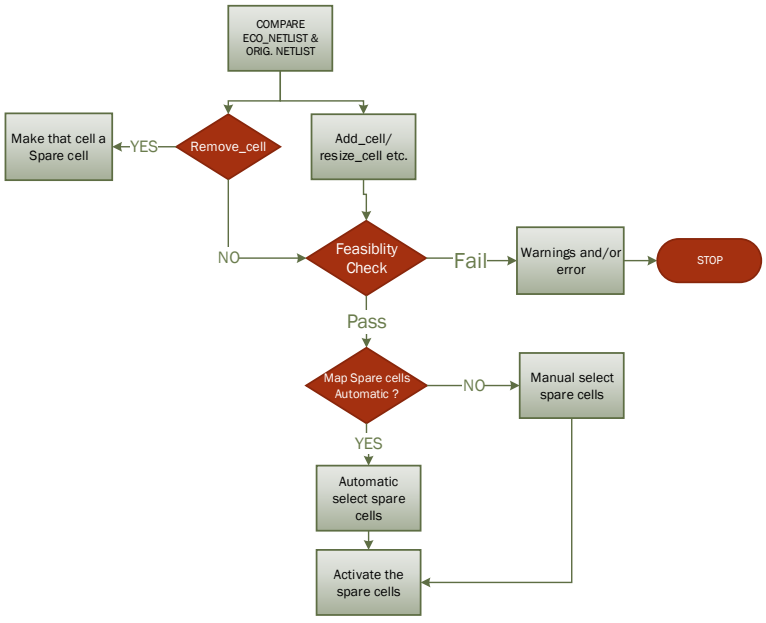


Figure 2.12: Feasibility analysis flow in post-mask ECO

Combining the Figures 2.2, 2.3 and 2.12, the complete flow can be divided into following steps.

Step-1 Describe and Capture the specification.

Step-2 Compare the ECO netlist with the design netlist

Step-3 Check the feasibility of the spare cells

Step-4 Carry out Spare cell Mapping

Step-5 Perform the incremental routing and fix the routing DRC.

Step-6 Perform Formal Verification

Step-1: Describe and Capture the specification

The function of this step is same as the first two steps of sections 2.3. However, this flow is explained using example 2-1 instead of example 2-3.

Step-2: Compare the netlists

Considering example used to explain this flow the corresponding command would be: `eco_netlist -by_verilog_file "verilog file" -freeze_silicon`. In comparison with the command used in unconstrained ECO flow, the additional argument "freeze_silicon" is used in this flow. As a result the functionality of the command changes a bit. Whenever tool detects new cells to be added in this stage, the tool does small feasibility checks for spare cells like valid spacing etc (as shown in Figure 2.12). Furthermore, the removed cells are not entirely deleted, rather made as spare cells for the future ECO.

Step-3: Feasibility checks

Apart from the checks performed by compare netlist command, additional checks like finding matching spare cells inside the design to map to, is performed in this step. Command `check_freeze_silicon` is used in ICC for carrying out feasibility analysis.

Figure 2.13 shows the feasibility report that was able to find the required spare cells and Figure 2.14 shows the report in which the missing spare cells are highlighted.

```
icc_shell> check_freeze_silicon

****************************************
  Report : feasibility check
  Design : RTC_norm_spareplaced
  Version: I-2013.12-ICC-SP2
  Date   : Wed Apr  9 16:29:43 2014
****************************************

// Overall summary:
==================================================
Number of eco cells: 3
Number of spare cells: 45
==================================================

// ECO CELLs mapping feasibility:
==================================================
// No references don't have enough spare cells.
// No references don't have enough spare cells.
==================================================

// Eco vs Spare detail report:
================================================================================
Reference       Eco       Spare     Vth           Drive     Power
                Cell      Cell      Group         Strength  Leakage
                                                            (default)
 MYOR2_D1BWP7THVT 1         15        --          0.038494  0.448832
 MYINV_D1BWP7THVT 1         15        --          0.017957  0.119191
 MYNR3_D2BWP7THVT 1         15        --          0.030203  0.620175
Reference       Eco       Spare     Vth           Drive     Power
                Cell      Cell      Group         Strength  Leakage
                                                            (default)
================================================================================
Feasibility check completed.
1
```

Figure 2.13: Spare cell feasibility check "successful"

```
icc_shell> check_freeze_silicon

****************************************
  Report : feasibility check
  Design : RTC_RT
  Version: I-2013.12-ICC-SP2
  Date   : Wed Apr  9 18:19:28 2014
****************************************

// Overall summary:
==================================================
Number of eco cells: 3
Number of spare cells: 0
==================================================

// ECO CELLs mapping feasibility:
==================================================
// These references don't have spare cells:
   MYOR2_D1BWP7THVT
   MYINV_D1BWP7THVT
   MYNR3_D2BWP7THVT
// No references don't have enough spare cells.
==================================================

// Eco vs Spare detail report:
================================================================================
Reference       Eco       Spare     Vth           Drive     Power
                Cell      Cell      Group         Strength  Leakage
                                                            (default)
MYOR2_D1BWP7THVT 1         --        --          0.017957  0.119191
MYINV_D1BWP7THVT 1         --        --          0.038494  0.448832
MYNR3_D2BWP7THVT 1         --        --          0.030203  0.620175
Reference       Eco       Spare     Vth           Drive     Power
                Cell      Cell      Group         Strength  Leakage
                                                            (default)
================================================================================
Feasibility check completed.
```

Figure 2.14: Spare cell feasibility check "failure"

Step-4: Spare cell mapping

After verifying the feasibility of the spare cells, in this step, ECO cells are mapped to spare cells. The spare cells have fixed functionality,so an ECO cell cannot be place at any spare cell location.Thus one can either use `place_freeze_silicon` command to carry-out automatic mapping or `map_freeze_silicon` command to perform manual mapping.

Figure 2.15 represents the original taped out design with spare cells and Figure 2.16 shows the design after the spare cell mapping. The three ECO cells highlighted in example 2-1 can be seen in latter Figure as mapped to appropriate sized spare cells.

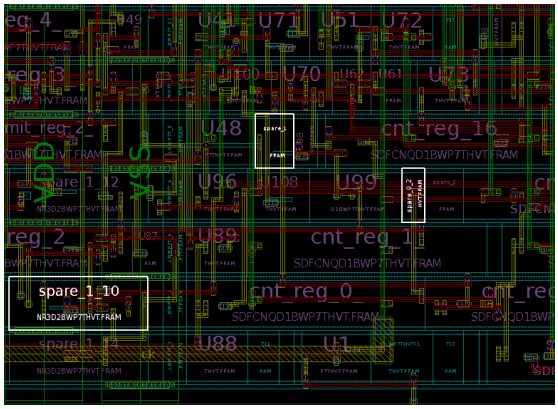


Figure 2.15: Original design having appropriate spare cells

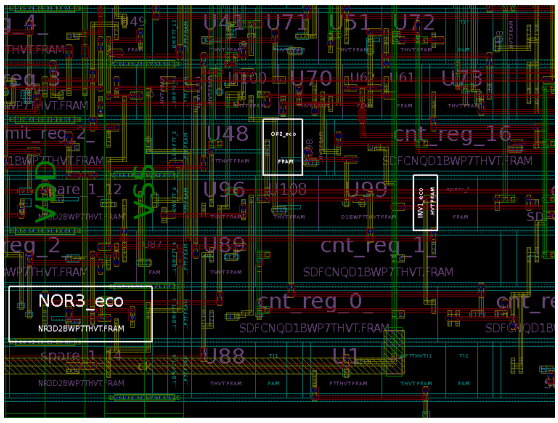


Figure 2.16: Design after spare cell mapping

Step-5: ECO Routing

Once the spare cell mapping is carried out, the connections are made using ECO routing. This step is already explained in section 2.3.

Step-6: Formal Verification

In this step the routed design is formally verified. Refer to section 2.3.1 for more information.

2.4.1 Complete ECO flow script

Appendix B gives the script for doing the freeze silicon ECO flow in ICC.

2.5 Metal-configurable Gate Array spare cells

This section introduces GA type cells and explains the concept of metal-configurable GA spare cells.

2.5.1 Definition

A gate array is an integrated circuit whose internal structure is an array of gates with interconnects initially unspecified [2]. The GA implementation can be divided into two steps.

1. Making transistor masks which is an array of uncommitted transistors gates on the GA chip

2. Metal interconnection between the transistors to configure a logic functionality.

From the above two points it can be said that GA spare cells can be developed by combining array of gates with structured ASICs [18].

2.5.2 Gate Array structure at chip level

Figure 2.17 shows the array of uncommitted transistor gates spread across the chip. It can be seen that the height of every GA cell is fixed and cells are arranged to form rows so that they can share same power and ground signals. This arrangement is similar to the structured ASICs. The space between two rows known as channel is used for inter cell routing.

Figure 2.17 also highlights the width and the height of the smallest GA cell (unit tile). The area occupied by this cell is known as one **gate array tile**. Similarly, the area occupied by smallest size standard cell is known as **standard cell tile**. It can be noticed that, as it is a GA type cell, there is no metal connection between pMOS and nMOS transistors. Thus, it can be said that, each row consists of arrays of GA tiles.

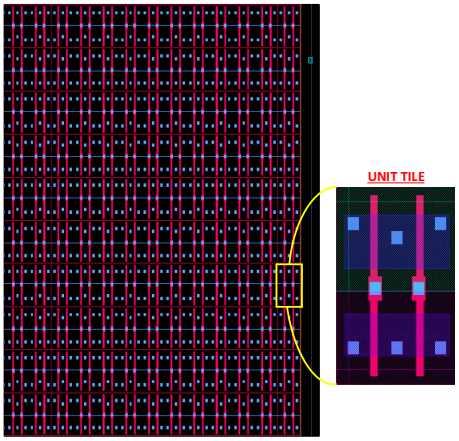


Figure 2.17: Array of unconnected transistors spread to form gate array (GA) structure

In terms of area, the size of a GA tile is more as compared to standard cell tile because the number of transistors and routing space inside a regular standard cell is optimized for fixed functionality. As a result, a significant reduction in width is achieved. One the other hand, in a GA structure, transistors gates are prefabricated, and only metal layers are available for configuring the GA cells. Therefore, there needed to be enough room for intra-cell routing.

Figure 2.18 shows the diagram of prefabricated and programmable layers for a GA cell. It becomes clear from the Figure that the transistor masks are prefabricated and only the metal layers are used to program the GA cells. Furthermore, the number of metal layers used to program the GA cells depends on the library used. By reducing

this number, the metal-masks costs can be reduced.

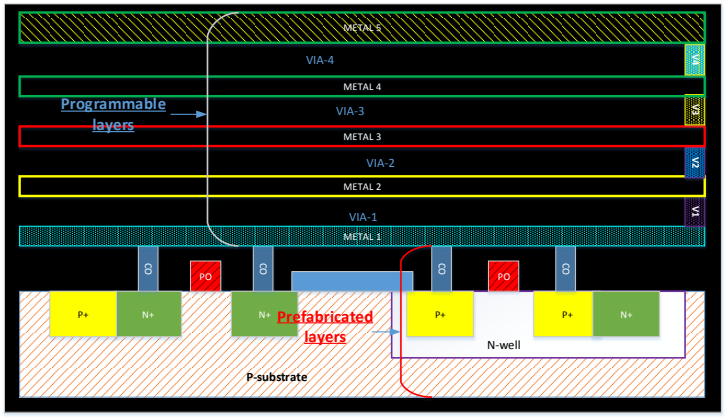


Figure 2.18: Prefabricated Layers and Programmable layers of a GA cell

2.5.3 Programming gate array cells

A GA tile which, when configured as spare cell, can be used to perform metal-only ECO is known as a base cell. The base cell should be such that any desired functionality can be configured upon it using metal layers.

Configuring the new functionality using base cell

Figure 2.19 describes the process carrying out metal-only ECO using base cell. The new functionality to be added is in the form of netlist. As shown in the Figure, this netlist consists of only GA type cells. Every ECO cell in the netlist is created by configuring the metal layers of the base cell.

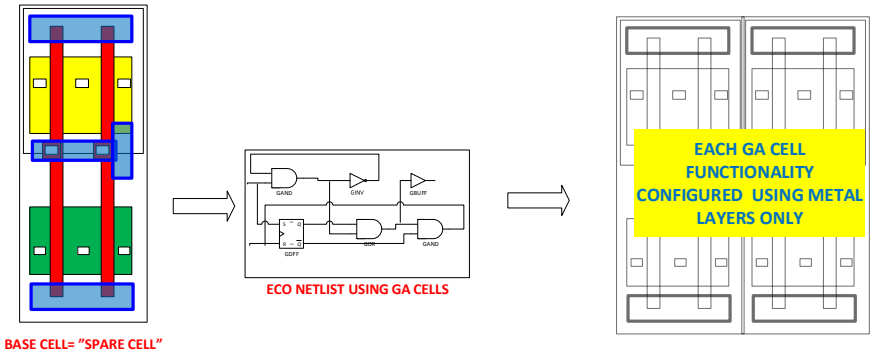


Figure 2.19: Using GA type base cell to realize metal-only ECO

Figure 2.20 shows some examples of configuring the base cell. Combinational logic as well as sequential logic can be realized using a base cell. The combinational logic requires two base cells placed subsequent to each other inside the chip, whereas the sequential logic requires nine base cells to do a metal-ECO. Thus, it can be said that any logic cell can be configured using a base cell if sufficient number of base cells is available.

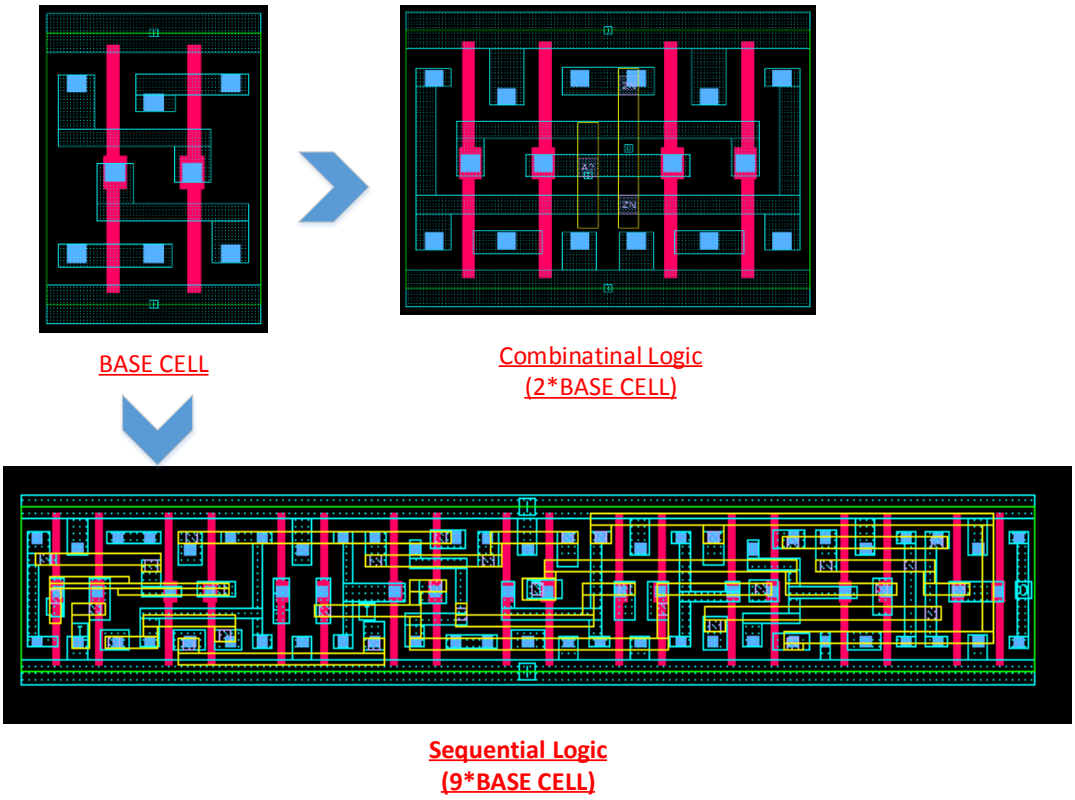


Figure 2.20: Use of base cell to make different types of cells

Base cell in the context of this project

In this project, the smallest sized GA filler cell is used as a base cell (as the title suggest). Figure 2.21 shows different sizes of GA filler cells that are defined in the library. It can be seen that that the GA fillers in this library have gates, source and drains shorted without connections to supply or ground rails. Hence they do not leak.

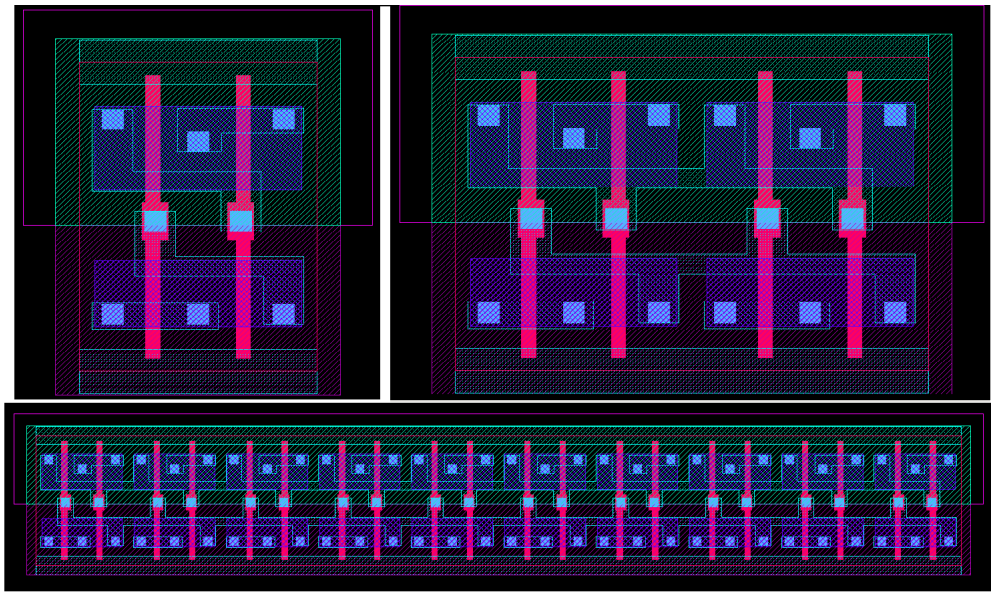


Figure 2.21: Different sized GA filler cells available in the library

2.5.4 Configuring process used in ICC to carryout an ECO

After placing GA fillers inside the design, in case of an ECO, the tool will first search for GA ECO cell in the library and then swap that GA cell with the GA filler. As the transistor mask structure of all the GA type cells is similar, only metal layers would change. Hence, the metal-only ECO.

Figure 2.22 shows example of the ECO swapping process. Lets say the ECO netlist consists of two ECO cells MYGA_NAND and MYGA_MUX4. For an ECO, the ICC would first look for corresponding cells in the GA library. After finding the cells in the library, the tool would look for equal or big sized GA fillers (MYGA_FILL) in the design. Finally it will perform the swapping as shown in the Figure.

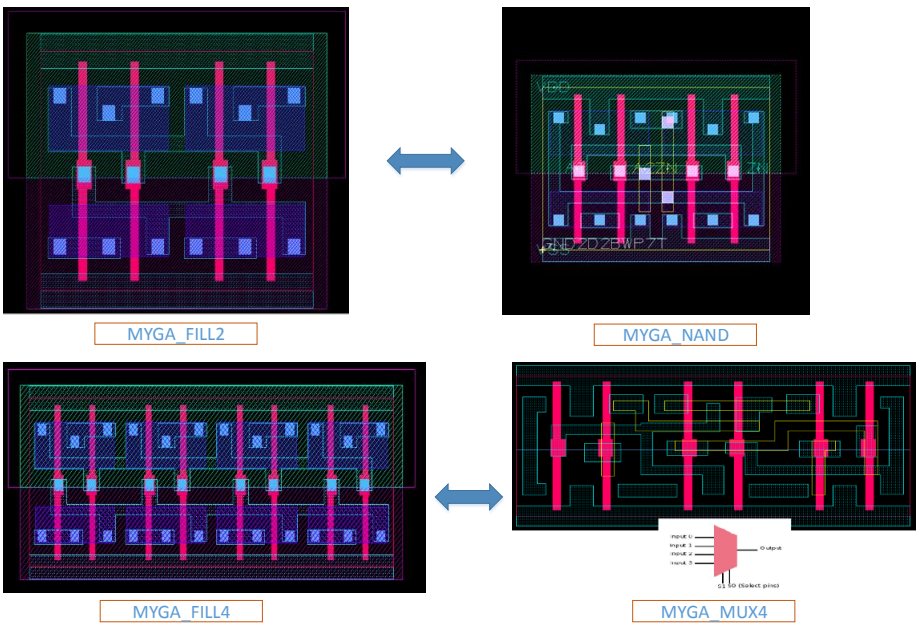


Figure 2.22: ICC's ECO swapping process

2.6 Advantages of using Gate Array cells as spare cells

As compared to conventional spare cells, the following are the advantages of using GA fillers as spare cells

1. As GA fillers are placed in the filler cell area, that area can also be now utilized for spare cell mapping. This is not the case with conventional spare cell. Figure 2.11 shows that spare cells occupy area other than filler cell area. Thus it can be said that 100% spare area of the chip can be utilized using the former one.

2. Unlike conventional spare cells, the inputs of GA fillers are shorted (see Figure 2.21) as a result they do not consume any additional leakage current.

3. GA fillers can be programmed to any functionality. Thus they do not suffer from inflexibility. This is certainly not the case with fixed functionality spare cells.

4. Finding efficient location for spare cell mapping is also a problem in case of conventional spare cells. Failing to find one would affect the routing closure, which ultimately could lead to timing closure failure.

2.7 Summary

This chapter gave the essential background of the important key terms used in the project title, which should assist the reader to understand the detailed flows mentioned in the subsequent chapters. In the beginning, the need for an ECO was explained with respect to various ASIC design stages. This was followed by a detailed explanation of ICC's ECO flows. Finally the chapter concluded by providing the necessary background regarding GA type cells and their usage to realize an ECO. The next chapter will modify the ICC's freeze silicon ECO flow and will present the GA spare cell methodology.

3

Gate Array spare cell methodology

In this chapter, the freeze silicon ECO flow mentioned in the previous chapter will be modified to use gate array (GA) filler cells as spare cells. The steps mentioned are designed to work with ICC tool and are based upon the steps described by Synopsys [13]. The spare cell methodology will be explained in two parts. Section 3.1 will explain necessary steps to to insert GA filler cells as spare cells. Subsequent section will make the use of this spare cells and will present the modified ECO flow. In the last section, the chapter will be summarized.

3.1 Adding Gate Array fillers as spare cells

This section explains the method to add GA fillers inside the design and configure them to use them as spare cells. Appendix C shows the complete script for the same. The script can be explained in five steps.

Step-1 Preparation for adding GA fillers

Step-2 Map the filler cells

Step-3 Report the mapped cells

Step-4 Insert them inside the design

Step-5 Setting them as spare cells

Step-1: Preparation for adding Gate Array fillers

In the most common case, the GA filler cells would not be configured as normal filler cells. This is because standard cell type fillers are usually used for filling the gaps. In

such a case, the GA fillers need to be set as filler cells otherwise, the tool will complain. The script shown below (example 3-1) serves this purpose. example 3-1 highlights the name of the GA filler cells defined in the library.

Example 3-1: Script to set GA filler cells as normal filler cells

```
cmMarkCellType
setFormField mark_cell_type library_name MY_LIB
setFormField mark_cell_type cell_name MYGA_FILL.^*
setFormField mark_cell_type pattern_match 1
setFormField mark_cell_type cell_type "std_filler"
formApply mark_cell_type
formOK mark_cell_type
```

Step-2: Map filler cells

The second command in the script `map_unit_tiles`, is used when there are multiple so called **unit tiles** defined in the library. As the dimensions of the GA cells differs from regular standard cells, there exists a need for differently sized unit tiles, one for GA, and one for regular standard cells.

The width and height of a unit tile should be such that, it equals to the corresponding tile (GA/standard cell) as defined in the section 2.5.2. Before moving into the details of each unit tile it is important to know the purpose of a unit tile irrespective of cell type in the ASIC design.

Concept of a Unit Tile

Unit tile is used when rows are created in the floorplan [19]. Without this information the floorplan will not be created. A unit tile contains following information.

- It defines valid spacing for the standard cells. This information is used by placement command.
- It contains the basic wire pitches according to the technology file. Routing tracks are defined based on this information.
  - Wire pitch= Wire spacing + wire width
- It defines power and ground rails.

Figure 3.1 shows the floorplan in which rows are created with unit tile named (`Tile Name`) "unit". It also highlights three other parameters `site_count, site_space, site_type`. These parameters are explained in the subsequent paragraph.

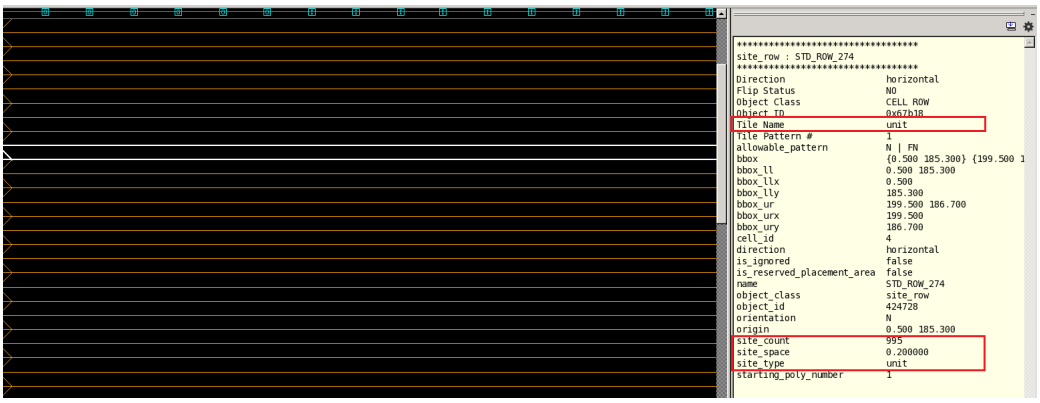


Figure 3.1: A Floorplan in which rows are created with `site_type`=unit

Figure 3.2 explains the two parameters `site_space` and `site_count`. It can be seen from the Figure that a row is divided into sites. The width of each site that is the parameter `site_space`, would be equal to the width of the unit tile and `site_count` would be the number of sites in a row. The third parameter, `site_type` determines which unit tile type to be used to define the rows. Thus, depending on which `site_type` is used, the other two parameters will change accordingly.

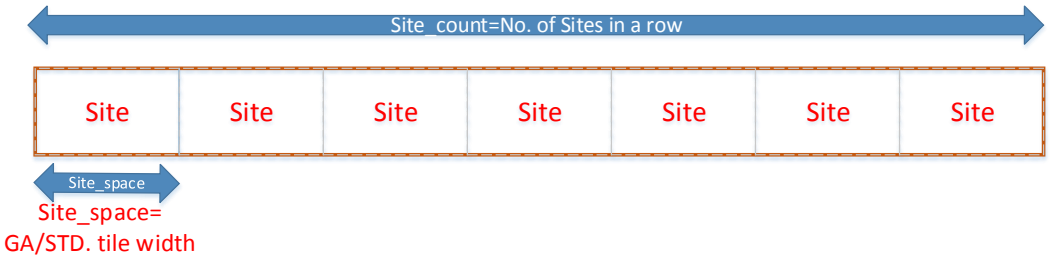


Figure 3.2: Concept of a row with respect to unit tile parameters

Figure 3.3 shows a standard cell unit tile named ”unit” that is defined in the library by default. Whenever the floorplan is created using `site_type= unit`, the tool uses this unit tile to create the rows. This is the default unit tile used by ICC to create rows. The parameter `site_space` which is marked in the Figure with spacing of ’0.2’ is same as the number shown in Figure 3.1. The routing tracks and the power and ground rails can easily be seen in the Figure.

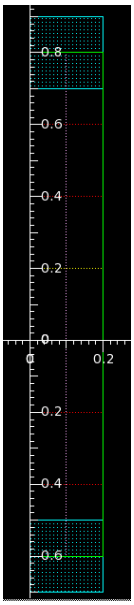


Figure 3.3: Standard cell unit tile ("unit")

Function of this step

The command is used to map multiple GA filler cells to unit tiles in the design. Without this mapping information, the tool will not be able to place the GA filler cells inside the design as GA filler cells have different tile size. The additional argument `unit_tile_name` defines the name of the unit tile to map to.

Step-3: Report Unit tiles

The command `report_unit_tiles` reports the tile mapping created by `map_unit_tiles` command. Figure 3.4 shows the mapping report. It shows the mapping of GA filler cells to unit tile named "unit".

```
icc_shell> map_unit_tiles -lib_cells $fillers1_ga_cells -unit_tile_name unit
1
icc_shell> report_unit_tiles
****************************************
*         GA FILLER UNIT TILE MAPPING       *
****************************************
GA LIBRARY CELL         UNIT TILE NAME
 MYGA_FILL10BWP7THVT          unit
 MYGA_FILL4BWP7THVT           unit
 MYGA_FILL3BWP7THVT           unit
 MYGA_FILL2BWP7THVT           unit
 MYGA_FILLBWP7THVT            unit
```

Figure 3.4: Output of `report_unit_tiles` command showing the mapped filler cells

Step-4: Insert Gate Array fillers

After mapping the GA fillers, the next step is to insert them. The fourth command in the script performs this. The tool will search for rows defined with `site_type=unit` and having gaps to place filler cells.

Figure 3.5 shows the design after GA fillers have been added inside the design. It also highlights the `is_spare_cell` attribute which tells that the cell currently cannot be used as spare cell. This points to the next step.

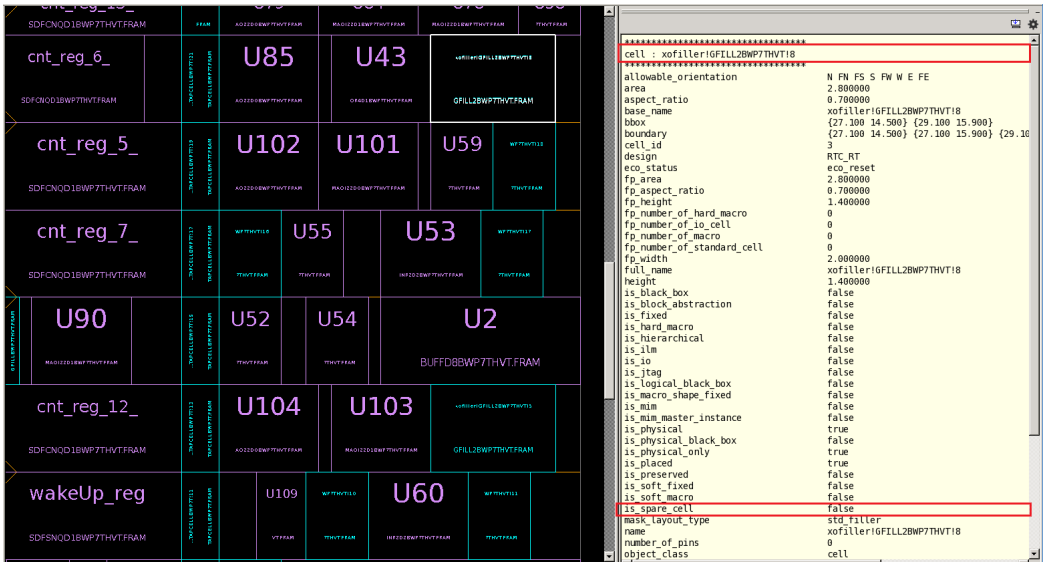


Figure 3.5: Design showing GA fillers inserted

Step-5: Setting the spare cell attribute

The last command in the script is used to set the added GA fillers as spare cells. This is done by setting the `is_spare_cell` attribute. Figure 3.6 shows the final design after GA fillers have been added and marked as spare cells.

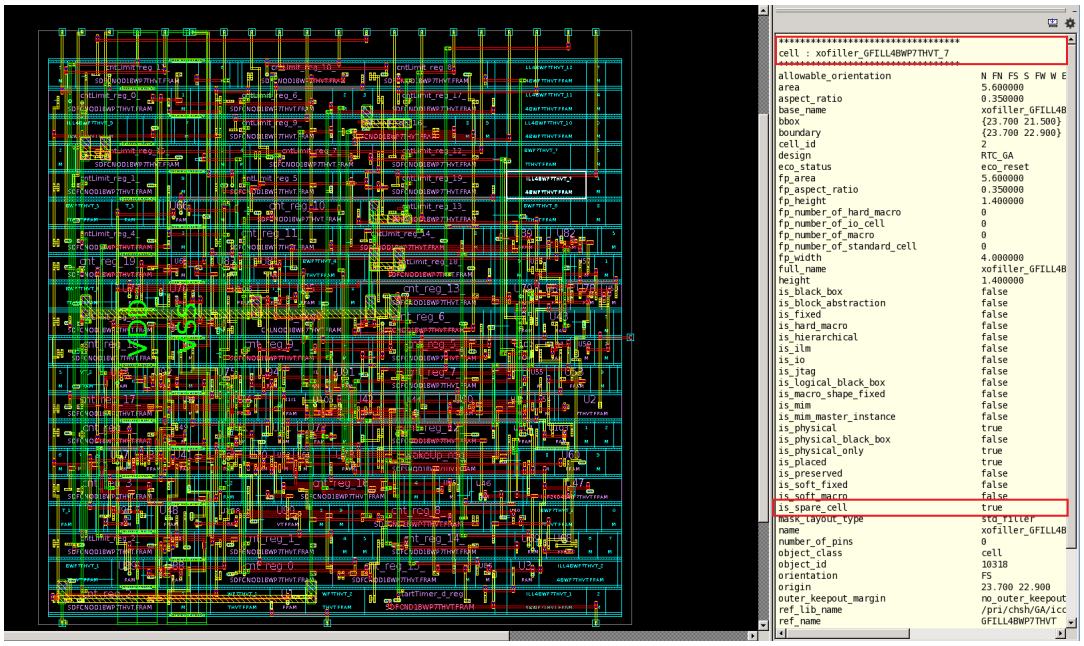


Figure 3.6: Final output design containing GA Fillers and marked as spare cells

3.2 Post-mask ECO using Gate Array fillers

In the previous section, the process of inserting GA fillers and marking them as spare cells was shown. In this section, the method to perform an ECO using this spare cells is presented. Before jumping into the ECO flow, some additional preparation for the spare cell mapping must be performed.

3.2.1 Preparation for the ECO

In order for tool to be able swap every GA ECO cell with appropriately sized GA filler (see section 2.5.4) this step must be performed. Example 3-2 shows the script for the same. The scripts works only on the cells which has a FRAM view [14] defined in the library. By giving all the GA cells the `gate_array_master_type` attribute, any number of GA ECO cells could be swapped with a spare cell (as long as the swapping requirements are matched) [13].

Example 3-2: Script for being able to swap GA ECO cell with GA spare cell

```
set type_1 [list $all_GA_cells_in_library]
open_mw_lib -write_ref MY_LIB
foreach _cell $type_1 {
echo $_cell
open_mw_cel $_cell.FRAM
set_attribute -class mw_cel $_cell.FRAM \
gate_array_master_type 1
```

```
close_mw_cel -save
}
```

3.2.2 Gate Array ECO process

The GA ECO flow can be divided into five steps. Note that, only the necessary changes concerning GA flow (if any) would be mentioned here. Refer to section 2.4 for detailed explanation about the steps.

The following are the steps to perform a freeze silicon ECO using GA cells in ICC .

Step-1 Describe and Capture the specification.

Step-2 Compare the ECO netlist with the design netlist.

Step-3 Check the feasibility of the spare cells.

Step-4 Carry out Spare cell Mapping.

Step-5 Perform the incremental routing and fix the routing DRC.

Step-6 Perform Formal Verification.

Step-7 Verify the metal-only ECO.

Step-1: Describe and Capture the specification

The only difference in this step is that, the ECO specification should use GA type cells in the ECO netlist instead of regular standard cells (see example 2-1 or example 2-2). This means that, OR gate, INV gate and NOR gate should be replaced by their GA equivalents. Example 3-3 shows modified ECO netlist that uses GA type cells.

Example 3-3: ECO netlist of Example 2-1 modified to use in GA ECO flow

```
//Original Netlist.
......
//ECO changes to be made
MYGA_OR2D1BWP7THVT OR2_eco (.Z ( cnt_0_or_z ) , .A1 ( n18_inv ) ,
                            .A2 ( cnt[0] ) ) ;
MYGA_INVD1BWP7THVT INV1_eco (.I ( n18 ) , .ZN ( n18_inv ) ) ;
MYGA_NR3D2BWP7THVT NOR3_eco (.A1 ( cnt[2] ) , .ZN ( cnt_1_nor ) ,
                            .A3 ( cnt[1] ) , .A2 ( cnt[0] ) ) ;
```

Step-2: Compare the ECO netlist with the design netlist

This step is exactly same as the explained in section 2.4.

Step-3: Check the feasibility of the spare cells

The feasibility analysis process is simpler in this case as compared to using conventional spare cells. Instead of searching for matching spare cell, the tool would just have to check for the appropriately sized GA filler. The remaining flow (see Figure 2.12) however, will remains the same.

Step-4: Carry out Spare cell mapping

The mapping process is rather a replacement process. As described earlier, the tool picks the GA ECO cells from the library, finds the spare cells (GA fillers) in the design and swaps the spare cells with GA ECO cells.

Figures 3.7 and 3.8 show the design before and after the mapping. It can be noticed from the Figures that, the "MYGA_OR2"(`OR2_eco`) ECO cell was swapped with two spare cells having a drive strength '1'. Moreover, the "MYGA_NR3"(`NOR3_eco`) was swapped with a spare cell having drive strength, '3' and so on.

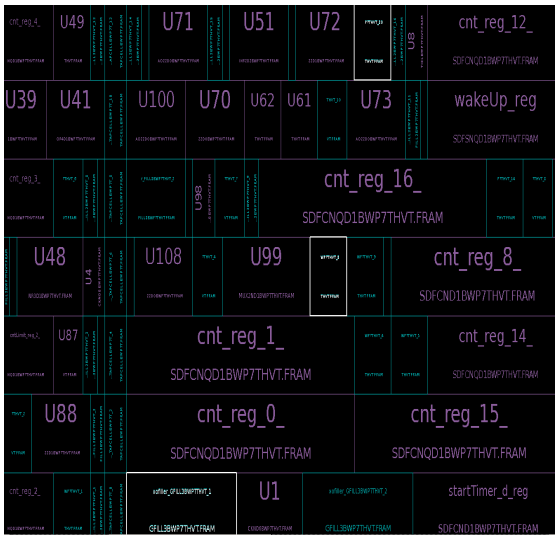


Figure 3.7: Design highlighting the different sized GA spare cells

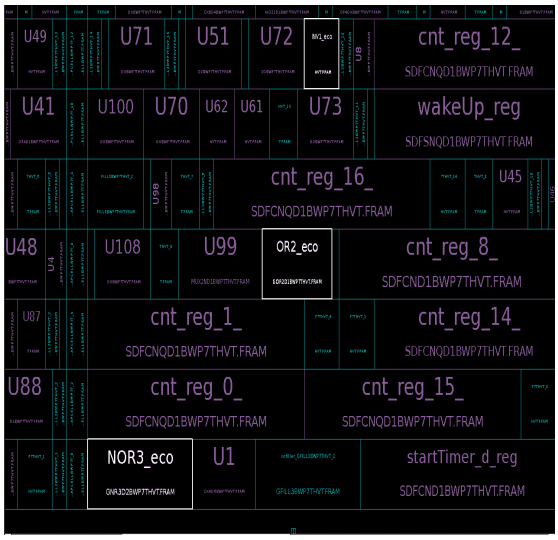


Figure 3.8: Design after carrying out spare cell mapping

Step 5: ECO Routing

The routing process is no different from the one explained in section 2.4. Figure 3.9 shows the transistor gate level view of the design before the ECO. Figure 3.10 shows the resultant design after the complete ECO flow has been performed. It can be seen from the latter that the location of underlying transistor masks after the ECO has not been changed, rather only the internal metal routing has been changed. This validates the metal-only ECO concept.

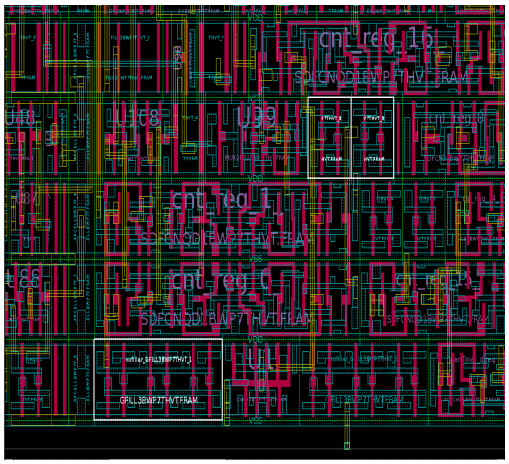


Figure 3.9: Transistor gate level view of the design before the ECO

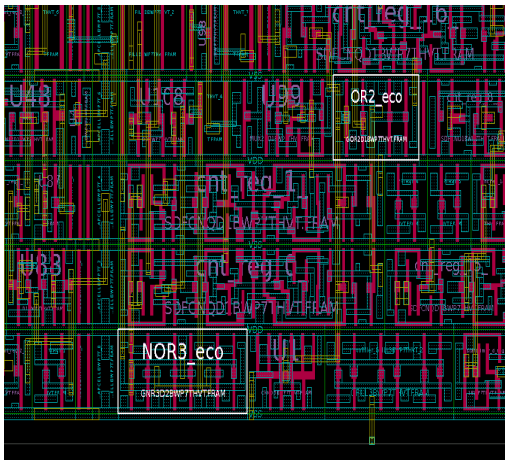


Figure 3.10: Transistor gate level view of the design after the ECO

Step-6:Formal Verification

This step is similar to the one explained in section 2.3.1.

Step-7: Verifying the metal-only ECO

Apart from verifying the ECO functionally, it is also important to verify whether only metal layers were used to carry out an ECO. In this case, the size of the ECO was small and more importantly, the ICC's ECO flow was used. As a result, it is not necessary to verify metal-only ECO, as tool makes sure this thing. However, it will be shown later that, in case of big ECOs, where tool commands are not used, this step has to be verified manually.

3.3 Summary

This chapter discussed the ICC's freeze silicon ECO flow using GA fillers as spare cells. The chapter started by providing steps for adding GA fillers and marking them as spare cells. The subsequent section explained necessary steps to perform a simple ECO using such spare cells with the help of an example. During this process, it was shown that once the ECO is performed one needs to verify the ECO both functionally as well as in terms of layers. The next chapter will make the use of this flow and discuss more advanced methods to carry out an ECO whenever the amount of changes to be made are big.

4

Advanced ECO methodology

Previous chapter explained the ICC's gate array (GA) ECO flow to perform an small ECO like bug fixing. This chapter focuses on using GA ECO flow to perform larger ECOs like adding a new functionality. The limitations of the conventional ICC ECO flow is mentioned in section 4.1. In order to overcome this, section 4.2 will provide overview of the new ECO methodology known as Advanced ECO methodology. This flow is explained using an example mentioned in section 4.3. The flow is divided in two parts and so the subsequent sections serves the purpose of explaining each of them in detail . The chapter will conclude by summarizing this new method.

4.1 Purpose of finding new methods for doing an ECO

The conventional GA ECO flow suffers from some major limitations when it comes to realizing big ECOs. These limitations are as follows:

- Impractical to manually modify netlists having large number of changes.
- Placement algorithms used by the incremental placement command fail to provide optimal results.
- There is no option to perform full CTS. Only incremental CTS can be performed.
- Difficult to solve issues related to routing, timing optimization etc.

Thus, the purpose of finding a new methodology is to overcome the above mentioned limitations by incorporating the full placement and routing algorithms into the ECO methodology.

4.2 Overview of the Advanced ECO methodology

Figure 4.1 gives the overview of the Advanced ECO methodology. As shown in Figure, this flow can be divided into two parts **top level flow** and **block level flow**. The block level flow takes RTL of the new functionality as an input and performs conventional ASIC design flow on it (as in Figure 2.1). This flow is modified in order to make it work on GA cells. After performing the ASIC design steps, the FRAM [14] view of the new functionality is created so that, the design can be used as a MACRO in the original chip,

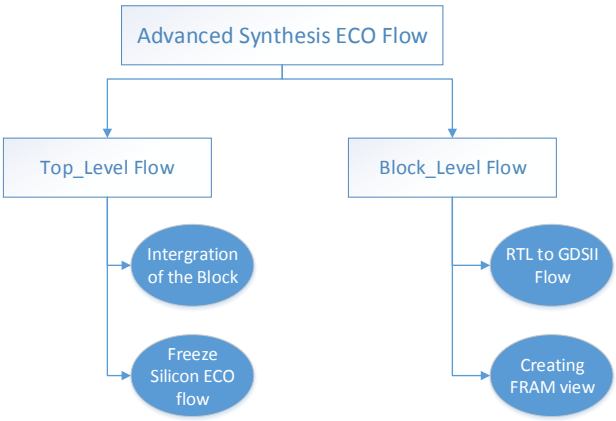


Figure 4.1: Overview of the Advanced ECO methodology

The top level flow takes the MACRO as an input and integrates it into the original taped out chip (top level design). This integration process involves using freeze silicon ECO flow mentioned in the previous chapter.

4.2.1 Overview of the Block Level Flow

Figure 4.2 shows detailed flow chart of the steps to be performed in the block level flow. Starting with the RTL (Block_RTL), followed by synthesis using only GA cells. Subsequently, the gate level netlist is verified in the formal verification step. The physical design steps are then performed.

The physical design steps are also modified to be able to place and route GA type cells using conventional commands. The steps involved in the physical design flow are given in the subsequent subsection. After routing is completed, the final layout netlist is verified against the block_RTL using formal verification. Finally, the FRAM view is created as mentioned earlier. The stream out of a Graphic Data System (GDS) file is optional.

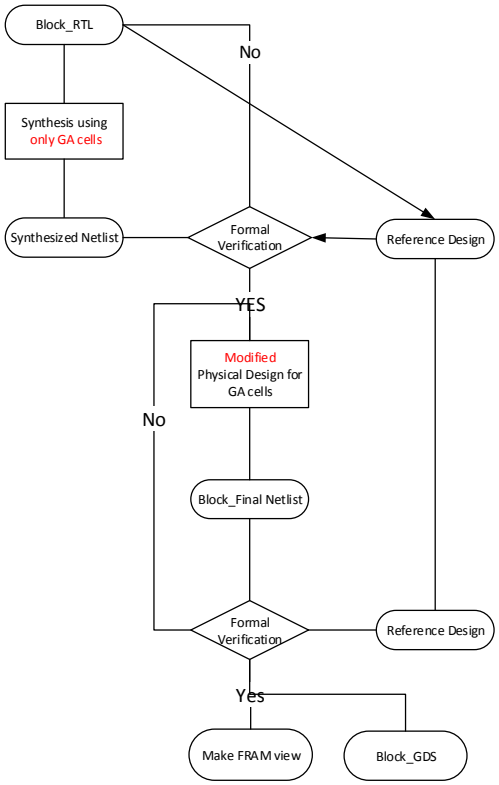


Figure 4.2: Modified ASIC design Flow for using GA type cells

Physical design steps

The following physical design steps are performed, once the synthesized netlist is obtained that used only GA type cells

- Create Floorplan
- Full Placement using conventional placement command `place_opt` instead of incremental placement (`place_freeze_silicon`).
- Full CTS.
- Routing using conventional routing command `route_opt` instead of `route_zrt_eco`
- After achieving the routing closure, GA fillers are inserted in the all the remaining area.
- Create FRAM view using `create_macro_fram` command.

4.2.2 Overview of the Top level flow

As said earlier, the top level flow is used to integrate the MACRO created in the block level flow . The top level flow can be divided into six main steps.

Step-1 Modify the RTL of the top level design

Step-2 Macro Mapping.

Step-3 Write out Verilog Netlist and modify it

Step-4 Perform Gate Array freeze silicon ECO steps

Step-5 Formal Verification.

Step-6 Metal-only ECO verification.

4.3 Example description

The methodology is explained using the example shown in Figure 4.3 . The blocks that are surrounded by the red dashed line is the new functionality to be added (ECO). The blocks that are surrounded by light red line is the top level design (existing chip). The spare area, which will used to integrate the new functionality, is not shown in the top level design.

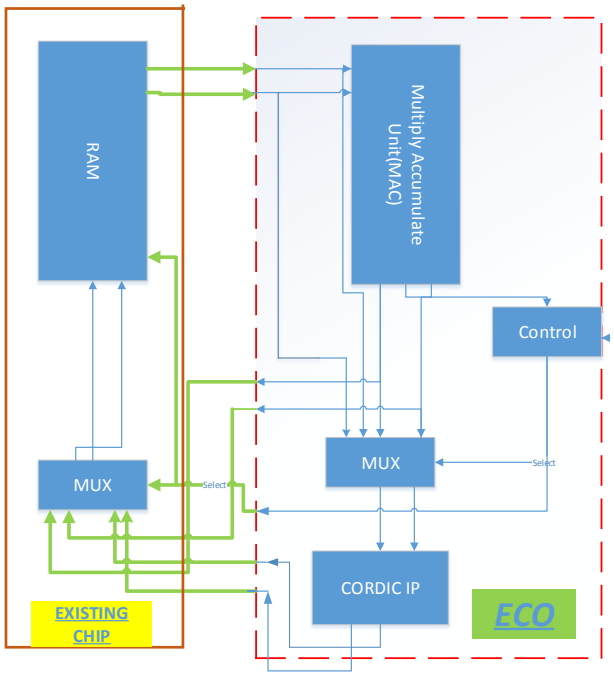


Figure 4.3: Example used for explaining the Advanced ECO methodology

The green arrows, connecting "ECO" block with the "existing chip" block, are the interface arrows. After mapping the MACRO onto the spare area, the inputs and outputs of the MACRO are connected according to these interface arrows. This connection procedure is carried out using the freeze silicon ECO flow.

4.4 Implementation- Block Level Flow

In this section, the block level flow given in Figure 4.2 is explained in detail.

4.4.1 Block_RTL

Example 4-1 shows the TOP level RTL code of the new functionality to be added. The top module name "top_cordic" would be used to represent this functionality in the top level design.

Example 4-1: RTL description of the new functionality

```
`include "mac_updated.v"
`include "control.v"
`include "select_generate.v"
`include "pa_Cordic.sv"
`include "Cordic.sv"


module top_cordic (
  input clk,reset,
  input integer word,
  input pa_Cordic::Mode          mode1,
  input signed [23:0] out1_ram,out2_ram,
  output signed [23:0] out_cordic1,mac_out_real,mac_out_imag,
  output signed [28:0] out_cordic2,
output sel_mux_ram
  );
  wire out_ctrl,out_done,sel_mux;
  wire signed [23:0] cordic_out_x,cordic_out_y;
  wire signed [28:0] cordic_out_z;
 reg signed [23:0] mux_out_real, mux_out_imag;

 mac_v_upd  m1 ( .clk(clk), .reset(reset) , .cnt_word(word),
                .dataa(out1_ram), .datab(out2_ram), .ctrl(out_ctrl),
        .adder_out_real(mac_out_real), .adder_out_imag(mac_out_imag),
                .done(out_done));

control c1( .clk(clk), .done(out_done), .ctrl(out_ctrl) );
select_out sel(.clk(clk), .ctrl(out_ctrl), .sel(sel_mux));

assign mux_out_real = (sel_mux) ? mac_out_real : out1_ram;
assign mux_out_imag = (sel_mux) ? mac_out_imag : out2_ram;
assign sel_mux_ram = sel_mux;
endmodule
```

4.4.2 Synthesis using only Gate Array cells

In this step, the RTL code is synthesized. Figure 4.4 shows the synthesized gate level netlist that contains only GA type cells. Note that, the library should contain all the required GA cells for mapping. Missing some of the cells, could result in a netlist having regular standard cells, which would ultimately lead to GA ECO failure.

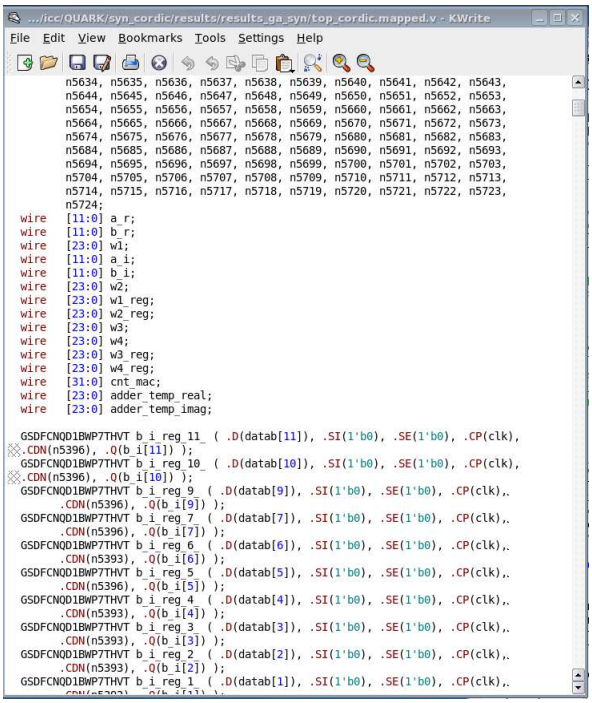


Figure 4.4: Synthesized netlist containing only GA type cells

4.4.3 Floorplanning

After successful formal verification, the physical design steps are followed. The first step in this flow is to create a floorplan.

In the floorplan step, rows are created among other things. [14]. These rows would be used by the placement tool to place the cells. In the context of this methodology, as the netlist contains GA type cells, the rows should be created such that,

- The conventional placement command should be able to place the GA type cells.
- The placed GA cells should follow GA structure (spare area) of the top level design.

Reason for emphasizing on Gate Array cell placement

In ICC, the placement command is designed to work with regular standard cells. This means that, by default the tool looks for regular standard cells in the design netlist. As a result, in some cases the tool would generate error message when trying to place GA type cells (This is shown in section 4.4.5).

Reason for following the Gate Array structure

Figure 4.5 shows the transistor gate level view of the top level design . The area within white dotted lines is the spare area filled with GA filler cells. This spare area is used to map the MACRO.

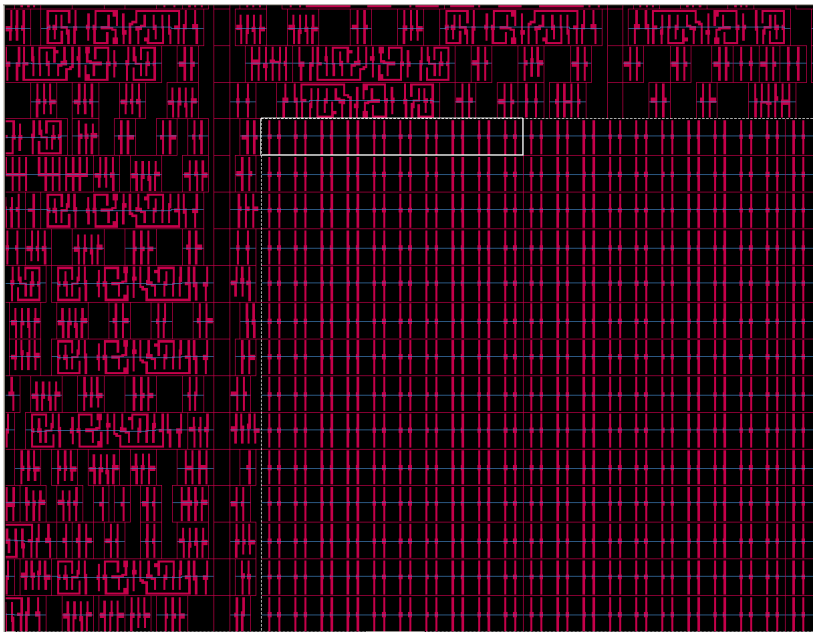


Figure 4.5: Transistor gate (red line) level view of the top level design

It can be seen that, the underlying transistor gates of GA filler cells are aligned below each other and follow a regular GA structure (as in Figure 2.17). Due to such type of arrangement in the top level design, it should be made sure that the placed GA cells follow the same structure. The benefit of doing this is that when the resultant MACRO is to be mapped upon the spare area, the underlying transistor gates of the cells inside the MACRO, would exactly map upon the underlying transistor gates of the GA filler cells (as in Figure 4.18).

After the successful mapping, the GA filler cells are removed from beneath the MACRO. This means that the uncommitted transistors of GA fillers at the top level design would be replaced by the committed GA cells from the MACRO. Furthermore, as the MACRO's underlying transistor gates lie at the same location, the transistor masks would be reused and only the metal-masks would have to be made. Thus, the reason for following the GA structure, is to achieve metal-only ECO irrespective of the ECO size.

4.4.4 Placement of the Gate Array cells

Figure 4.6 shows the transistor gate level view of the placed GA cells whose floorplan was created using `site_type = unit` (by default floorplan). Comparing this Figure with Figure 4.5, it can be said that, if the MACRO was made from this design and mapped onto the spare area, the MACRO mapping will not be successful. Figure 4.17 gives an idea of the unaligned transistor mapping. As a result, the second floorplan objective would have failed.

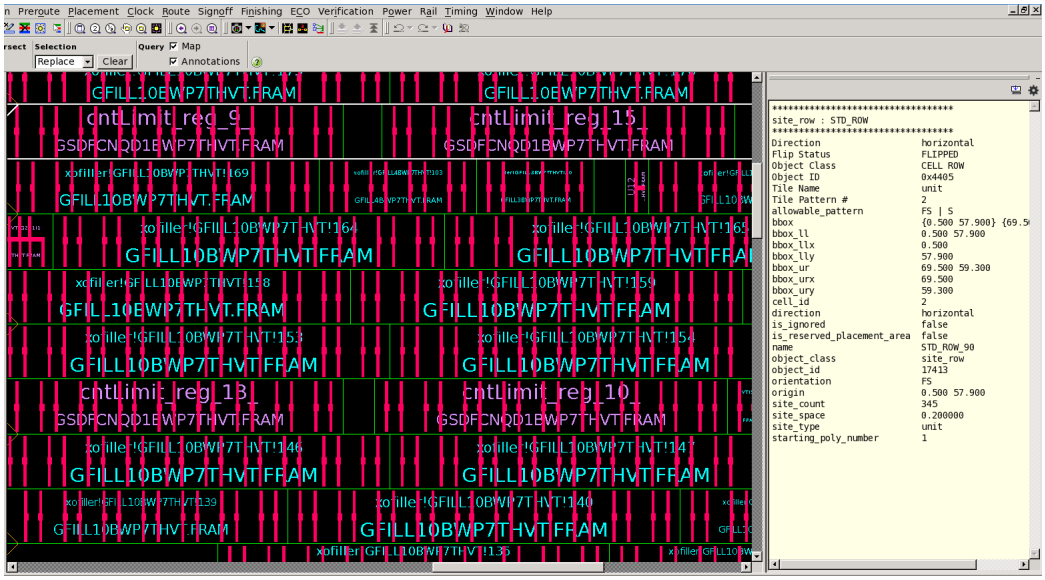


Figure 4.6: GA cells placed on rows created using `site_type = unit`

Thus, it can be concluded that, rows should not be created using standard cell unit tile ("unit"), if GA cells are to be placed.

Need for creating new unit tile for Gate Array cells

The reason for the misaligned placement of the GA cells is that the GA cells are bigger than standard cells. The tool will be able to place the cells wherever it finds the sufficient place in a row, but, the GA structure will not be followed. In order to overcome this, a new unit tile has to be defined in the library such that the width of this unit tile (`site_space`) would be equal to the width of the base cell (GA filler). As a result of creating such unit tile, the distance between the sites in a row, would be equal to the base cell width, which would aid to follow the GA structure.

Gate Array unit tile

In order to follow the GA structure, a new unit tile is defined with `site_type = gaunit`. Generally, this unit tile would be available as predefined in the library, if the library

contained the GA type cells. The reason for not discussing and/or mentioning this unit tile earlier, was to give the reader the purpose for creating one .

Figures 4.7 and 4.8 show the comparison between standard cell unit tile ("unit") and the GA unit tile ("gaunit"). It can be noticed that, the height is same for both but the width is equal to the corresponding base cell width.

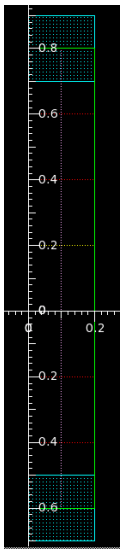


Figure 4.7: Tile used for creating rows with `site_type=unit`

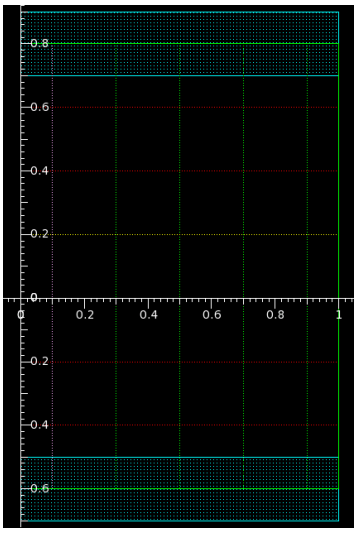


Figure 4.8: Tile used for creating rows with `site_type=gaunit`

4.4.5 New methods for placing Gate Array cells

As the default floorplan did not work for GA type cells, several other methods have been tried out. These methods assume that, both "gaunit" and "unit" tiles are available in the library.

Using `site_type= gaunit` to create rows in a floorplan

In this scenario, all the rows were created using "gaunit" tile. Figure 4.9 shows the corresponding floorplan. The number '1.0' besides the `site_space` parameter indicates the width (in μm) of the "gaunit" tile. Note that, this number is the same as the one shown in Figure 4.8.

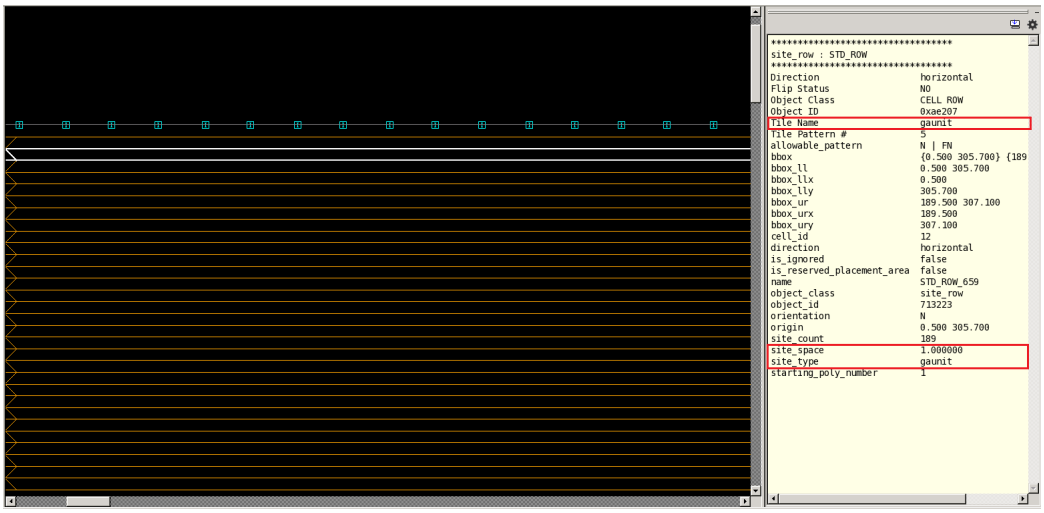


Figure 4.9: Rows created using `site_type = gaunit`

When tried to place the cells, the tool terminated with an error showing that it was not able to find the regular standard cells in the design. The reason for this kind of error was, the tool being not able to find rows created with `site_type= unit`. Thus, it can be concluded that, during placement, the ICC tool by default assumes that the design contains regular standard cells.

Using `site_type= gaunit` and `site_type= unit` to create rows in a floorplan

In order to resolve the errors of previous scenario, in this scenario, some rows were created using standard cell unit tile (”unit”) and some were created using the GA unit tile (”gaunit”). Consequently, the tool was able to place the GA type cells. Figure 4.10 shows that output. Furthermore, it can also be seen that, the second floorplan objective is met i.e. the cells are aligned. Thus, this floorplan choice was chosen.

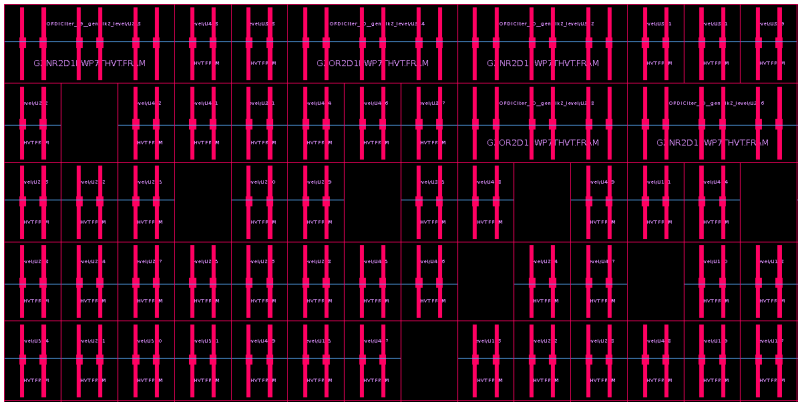


Figure 4.10: Figure showing GA cells placed as well as aligned

On the down side, it was found that, the tool only placed the cells on the rows which were created using "gaunit" tile. This happened because GA cells are attached to "gaunit" tile in the library. As a consequence, number of rows that were created using "unit" tile remained unused. In order to minimize this number and find out the exact number of rows needed with `site_type=unit`, several iterations of floor planing were performed. In the end, it was found that only **one row** should exist in the design with `site_type = unit`.

4.4.6 Insert Gate Array fillers in all the remaining area

After placing GA cells according to the requirements and the design is routed, GA type filler cells are inserted to fill the remaining gaps. Figure 4.10 shows the gaps to be filled and Figure 4.11 shows the final design after filling all such gaps. In the latter, every part of the design is filled with some kind of GA type cell. It is important to make sure this is the case, otherwise the overall methodology would fail. The reason for not having gaps is explained in the MACRO mapping step of the top level flow (see section 4.5.1).

Figure 4.11: Final block level design filled with GA type cells everywhere.

4.4.7 Create MACRO of the final placed and routed design

In this step the MACRO is created. It is used to represent the complete design as a block in the top level design. The FRAM view which is required for this representation is created using `create_macro_fram` command in ICC.

4.4.8 Results-Block Level Flow

The output of this flow is shown in Figure 4.12. It shows the MACRO instantiated in the top level design but not mapped. The command `create_cell` is used to instantiate the design as a MACRO from the FRAM view.

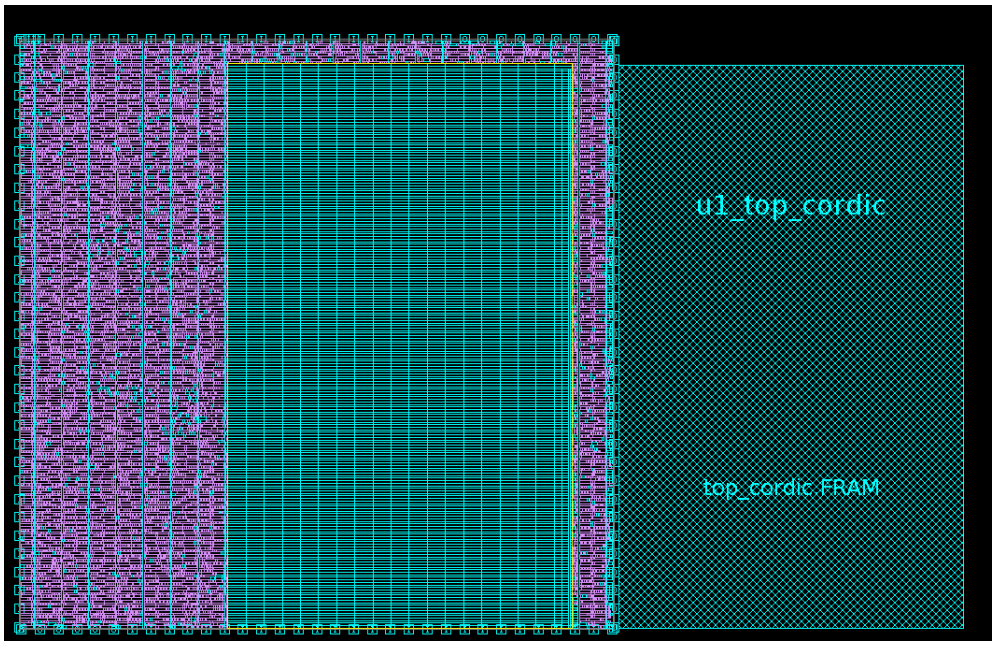


Figure 4.12: New functionality instantiated as an MACRO in the top level design

4.5 Implementation - Top level flow

In this section, the steps of the top level flow as mentioned in the section 4.2.2 would be explained in detail. As mentioned earlier, the purpose of this flow is to integrate the MACRO representing the new functionality into the original taped out chip.

4.5.1 Top level Design Assumptions

In order to integrate the MACRO as an ECO in the top level design, several assumptions have been made regarding the top level design.

1. The top level design contains a spare area filled with GA type fillers.
2. Some functionality exists in the top level design that is implemented using GA type cells.

The first assumption is used to explain this flow. It says that, in order to integrate the MACRO into the top level design, there should exist a dedicated spare area inside the chip. Moreover, this spare area should be filled with GA filler cells.

The second assumption is optional but it is more feasible as compared to the first one. It can be explained using Figure 4.13. The left side of the Figure shows the original taped out chip having some blocks (in green) implemented using GA cells. The new functionality ("New_func_GA") to be added as an ECO is shown in orange.

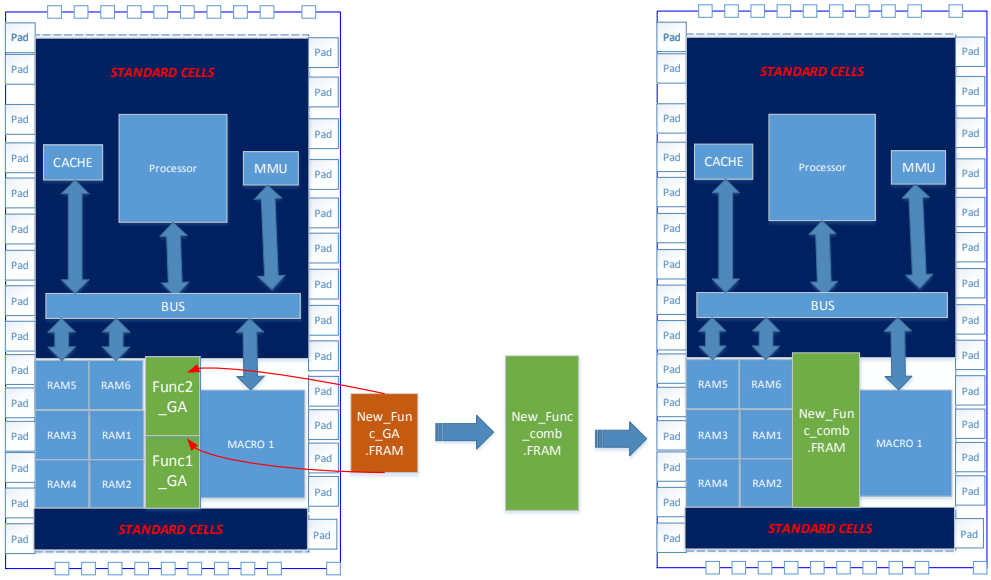


Figure 4.13: ECO flow for top level design containing blocks implemented using GA type cells

Assuming that the new functionality is implemented using GA type cells, it is then possible to swap the new functionality with one of the blocks in green. The process is explained as follows. The RTL of the new functionality is combined with the RTL of the remaining green block. This combined RTL becomes the input to the block level flow. Subsequently, the block level flow is performed on this input. Lets say the output is "New_func_comb.FRAM". In order to insert this combined MACRO as an ECO, both the green blocks are deleted from the original taped out chip and in that same location, the combined MACRO is mapped using top level flow. Thus ensuring the metal-only ECO.

Step-1: Modify the RTL of the top level design

The RTL of the top level design is modified to include the RTL of the MACRO (the Block_RTL). This combined RTL is called the modified top level RTL. This RTL will be used as a reference for formal verification, later in the flow.

Figure 4.14 shows the top level RTL of the original taped out chip and Figure 4.15

shows the modified top level RTL. The modifications highlighted in the later, were made according to the connections defined in the example (green lines in Figure 4.3).

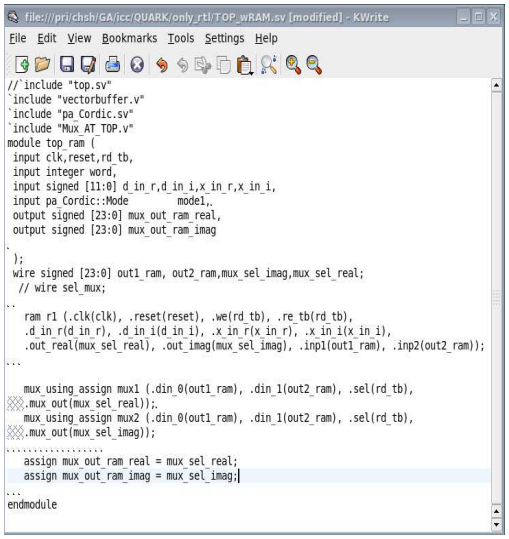


Figure 4.14: Original RTL of the top level design

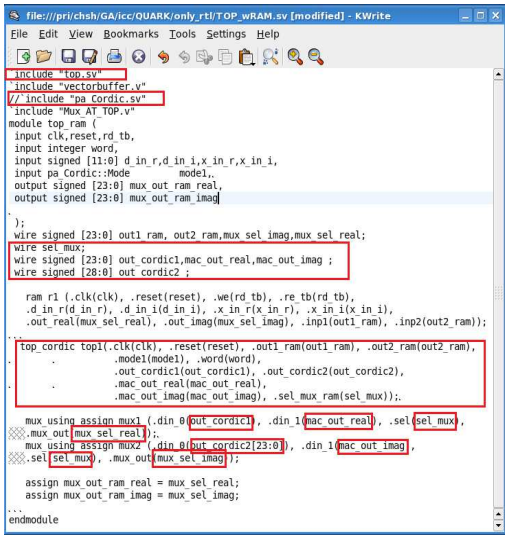


Figure 4.15: Modified RTL of the top level design

Step-2: MACRO mapping

The next step is to map the instantiated MACRO onto the spare area. Figure 4.16 shows the MACRO, mapped upon the filler cell area. During the mapping, one has to make sure that the transistors gates of the cells inside the MACRO, exactly map upon the transistors gates of the filler cells, as explained in section 4.4.3.

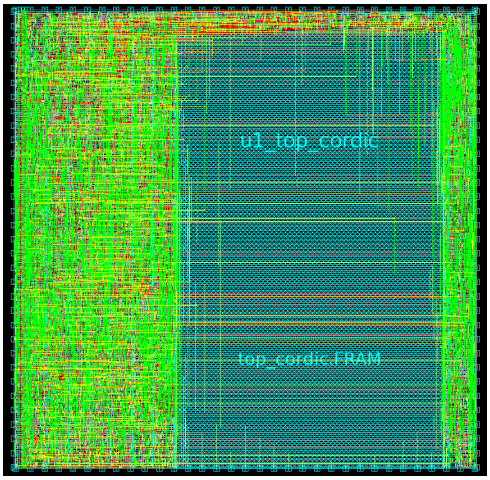


Figure 4.16: Top level design after MACRO mapping

Figures 4.17 and 4.18 show the difference between unaligned transistor mapping and aligned transistor mapping. Comparing the width of the transistor gates in both the Figures, it can be seen that, the transistor gates are thicker in the former. This shows that the mapping is not perfectly achieved.

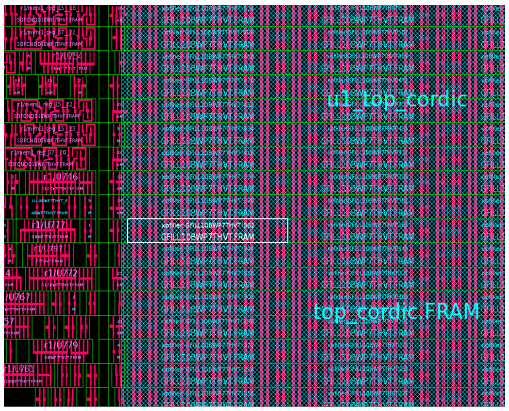


Figure 4.17: Transistor level view of the unaligned transistor mapping

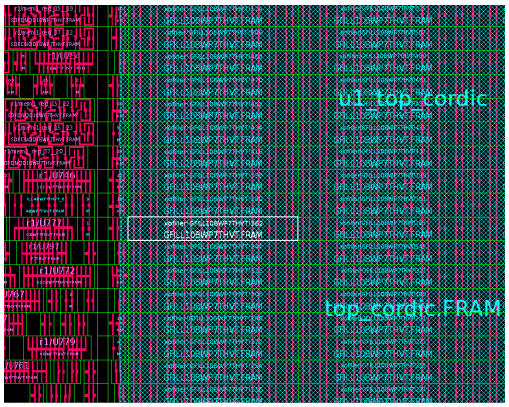


Figure 4.18: Transistor level view of the aligned transistor mapping

Reason for not having gaps in the MACRO

After successful mapping is achieved, the GA fillers are removed from beneath the MACRO. If the MACRO contained some gaps, those areas would not have any transistor gates (as in Figure 4.19). On the other hand, there were no gaps in the spare area of top level design. This means that in the original chip, the transistor masks layers would change. As a result, the design will fail the metal-only ECO verification.

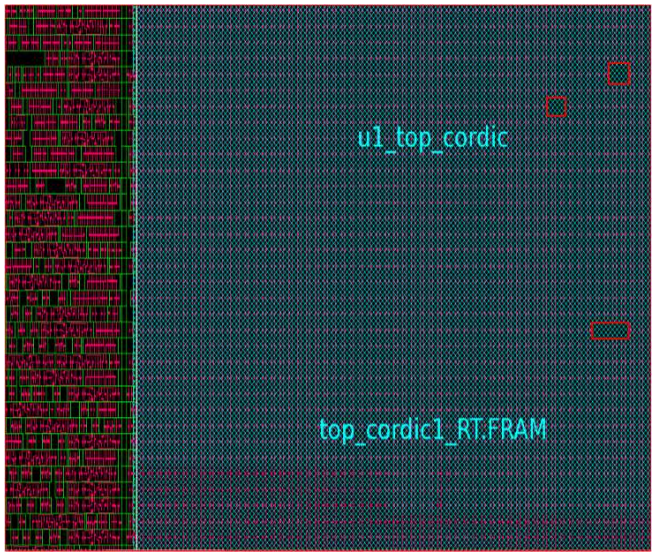


Figure 4.19: Figure highlights the gaps within the MACRO

Step-3: Write out a verilog netlist and modify it

Before making connections between the mapped MACRO and the top level design, a verilog netlist is written out. Figure 4.20 shows that netlist. The top level module name of the block RTL can be seen in the netlist. This means that the tool has found the instantiated MACRO. The I/Os of the MACRO are not connected to any of the pins at the top level design, hence the netlist shows it as "UNCONNECTED".

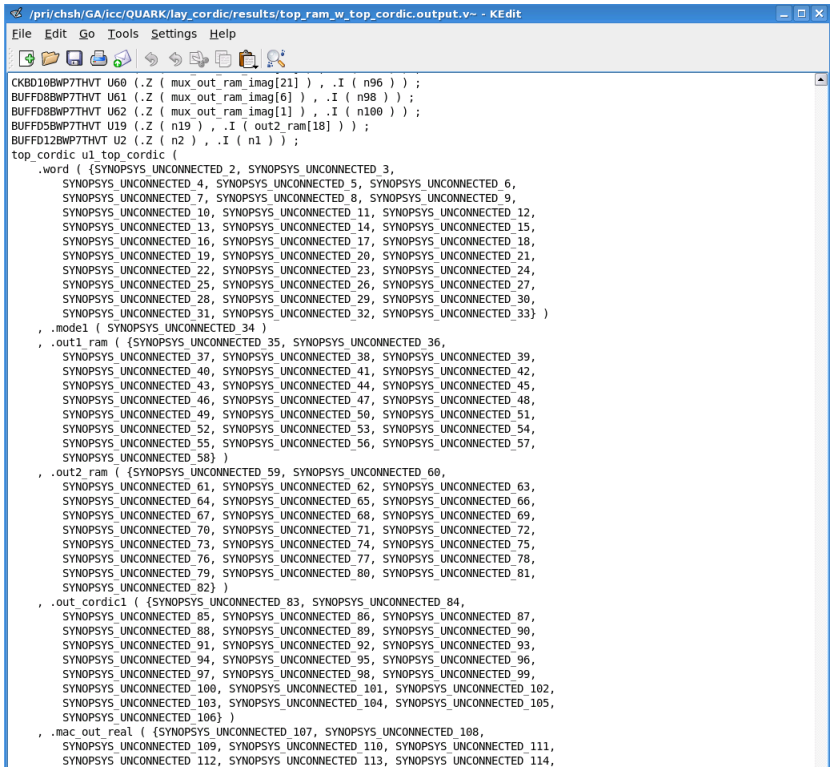


Figure 4.20: Top Level Netlist showing the MACRO (top_cordic)

The written out netlist is then modified according to the connections defined in the example. This netlist now becomes the ECO netlist (as in section 3.2.2). Figure 4.21 shows that modified netlist. The changes made in the netlist are highlighted in red.

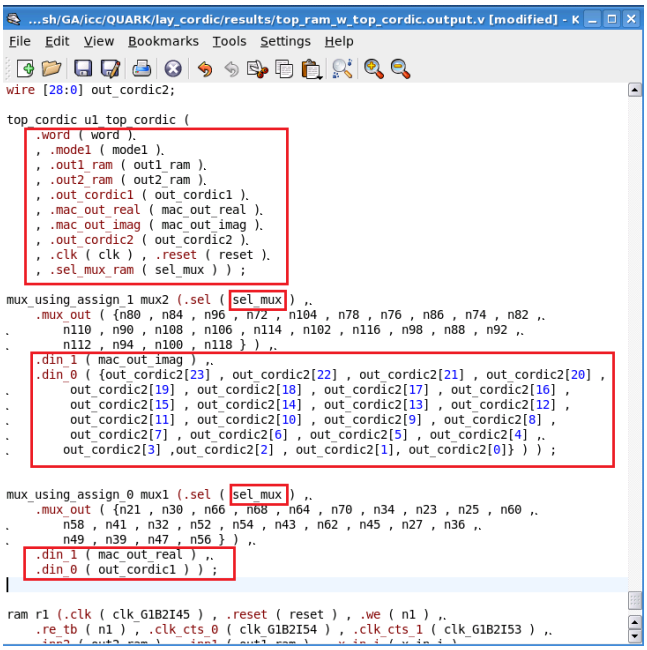


Figure 4.21: Top Level Netlist showing the I/Os of the MACRO connected (ECO-Netlist)

Step-4: Perform Gate Array ECO process

After creating the ECO netlist, the remaining freeze silicon ECO flow (only till step-5) is then performed. Figure 4.22 shows the ECO routed design. It highlights the connections made between the MACRO and the top level design.

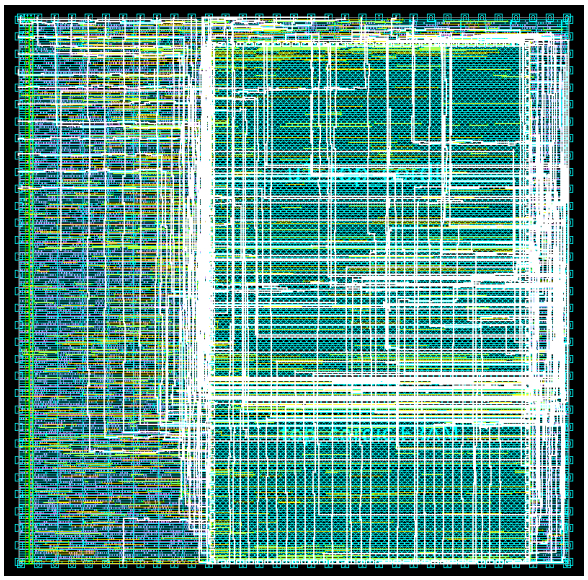


Figure 4.22: Final top level design after the MACRO integration

Step-5: Formal Verification

The final layout netlist which is written out after the ECO routing is then formally verified. That is checking whether the layout is functionally equivalent with the expected outcome (modified top level RTL). This is the most important and time consuming step in the flow.

The two designs to be compared are shown in Figures 4.23 and 4.24. The former is the reference design and the latter is the implemented design. If the formal verification fails, the following steps are looped until successful verification is achieved.

- Modify the final netlist to make it a new ECO netlist.
- Repeat the step-4 of this flow.
- Perform formal verification on the new netlist.

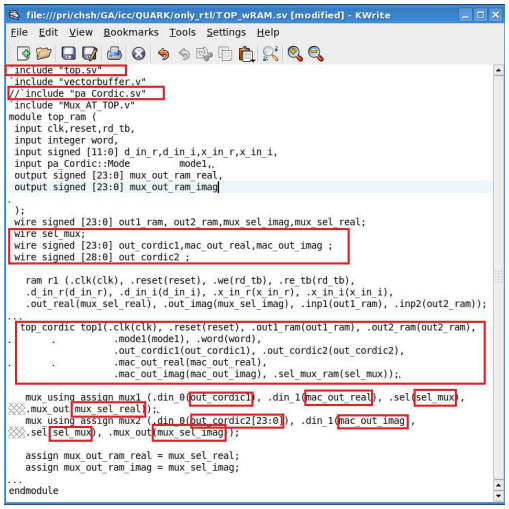


Figure 4.23: Modified RTL of the top level design ("existing chip")

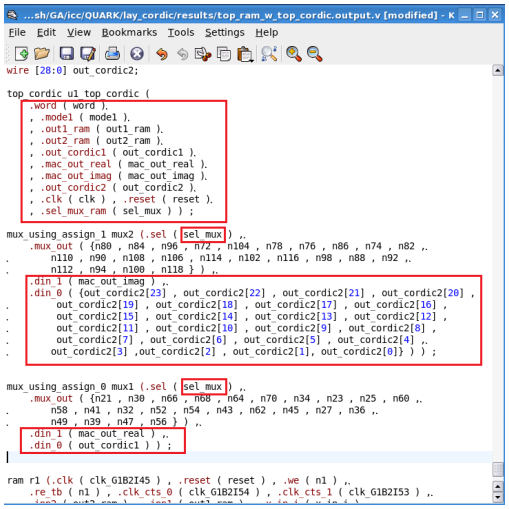


Figure 4.24: The final layout netlist after the complete ECO process

Step-6: Compare GDS using Calibre

Apart from verifying the final layout in terms of functionality, one also has to verify that only metal layers have been used to perform this change. In order to verify that, the GDS has to be streamed out and compared. The two designs whose GDS were compared are shown in Figures 4.25 and 4.26. The original top level design is shown on the left and the final top level design is shown on the right.

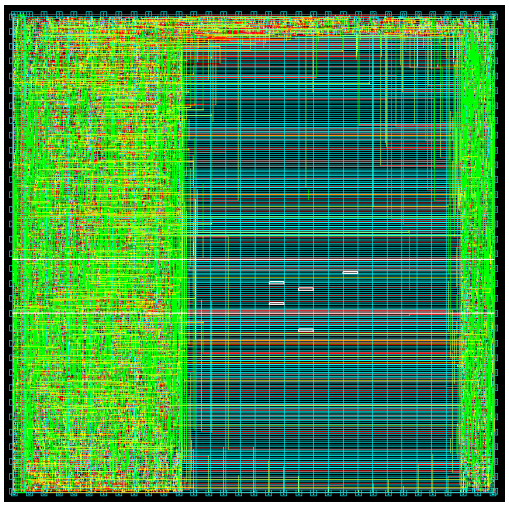


Figure 4.25: Original top Level design with GA filler

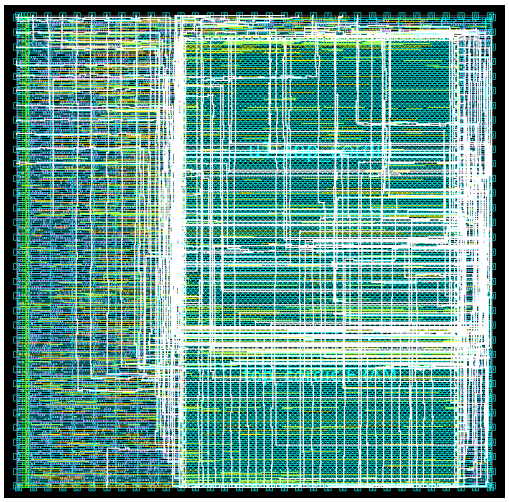


Figure 4.26: Final top level design after the MACRO integration

The stream out of the GDS is done using ICC. But the comparison is done using the Mentor Graphics Calibre tool. The corresponding command in the latter is `compare_gds`. Figure 4.27 shows the output of that command.

```
Session  Edit  View  Bookmarks  Settings  Help
MERGING layer 171 from ./results/top_ram_RT_w_blockage.gds ...  Merged Geometry Cou
READING layer 171 from ./results/top_ram_w_top_cordic.gds ...  Original Geometry Co
MERGING layer 171 from ./results/top_ram_w_top_cordic.gds ...  Merged Geometry Coun
XOR ...  XOR-ed Geometry Count = 1144
259 chsh@bowmore% /n/caduser/tools/calibre/scripts/compare_gds_summary.pl diff1.log
GDS FILE : ./results/top_ram_RT_w_blockage.gds
GDS FILE : ./results/top_ram_w_top_cordic.gds
Layer : 3        XOR-ed Geometry Count =  0
Layer : 6        XOR-ed Geometry Count =  0
Layer : 17       XOR-ed Geometry Count =  0
Layer : 25       XOR-ed Geometry Count =  0
Layer : 26       XOR-ed Geometry Count =  0
Layer : 30       XOR-ed Geometry Count =  0
Layer : 31       XOR-ed Geometry Count =  536685
Layer : 32       XOR-ed Geometry Count =  98726
Layer : 33       XOR-ed Geometry Count =  51244
Layer : 34       XOR-ed Geometry Count =  22849
Layer : 35       XOR-ed Geometry Count =  1047
Layer : 51       XOR-ed Geometry Count =  137907
Layer : 52       XOR-ed Geometry Count =  103955
Layer : 53       XOR-ed Geometry Count =  39031
Layer : 54       XOR-ed Geometry Count =  1615
Layer : 67       XOR-ed Geometry Count =  0
Layer : 68       XOR-ed Geometry Count =  0
Layer : 108      XOR-ed Geometry Count =  0
Layer : 119      XOR-ed Geometry Count =  0
Layer : 171      XOR-ed Geometry Count =  1144
```

Figure 4.27: Compare GDS Results

The keywords mentioned in the compare GDS output can be explained as follows.

Layer Shows the different layer numbers. For example,

17 Polysilicon layer

30 Contact layer

31-35 Metal layers

51-54 Via layers

117 Text layer

remaining different transistor masks layers

Geometry Count Shows the amount of changes the tool found in the final top level design (implementation design), when compared to original top level design (reference design).

The command compares both the designs and points out the layer differences. It gives a number showing the amount of differences found. For example, Layer 17 shows Geometry Count of '0'. This means that no differences have been found with respect to the polysilicon layer. Thus it can be said that the transistor poly-masks have been reused. Layer 31 shows Geometry Count of '536685'. This means that tool has found this many changes with respect to the that metal layer. This tells that the metal layers in the implementation design differs from the referenced design. Such type of output conforms the metal-only ECO.

4.6 Summary

In this chapter, the advanced ECO methodology was proposed and explained in detail. The major advantage of this methodology is that, it works irrespective of the size of an ECO. The chapter began by giving the block diagram which divided the methodology into two parts. Subsequent sections explained individual parts in detail with the help of an example. The chapter concluded by verifying the output functionally as well as verifying that only metal layers were used to implement the change. As this methodology uses GA type cells to realize an ECO, their implementation style is quite similar to the structured ASICs Therefore, the next chapter will be dedicated to discuss and compare the advanced ECO methodology with the structured ASIC methodology.

5

Structural ASICs vs. Advanced ECO methodology

The concept used by advanced ECO methodology to perform metal-only ECO is very similar to the structured ASIC design methodology that prevailed few years back. Metal-configurable gate array (GA) spare cells are used in the former. Because these cells are developed by combining GA with the structured ASICs (see section 2.5.1), it is interesting to find out differences and similarities between structured ASICs and the advanced ECO methodology. This chapter will start by providing the relevant background on structured ASICs. Section 5.2 will compare the GA design style and the structured ASICs design style. In the subsequent section the advanced ECO methodology which uses the GA concept is compared with the structured ASIC methodology. The last section is dedicated to discuss both the methodologies from application point of view. At the end of this chapter the summary is presented. Note that, it is assumed that the reader knows the basic concepts of Field Programmable Gate Array (FPGA) and the standard cell based ASIC.

5.1 Relevant background on Structured ASICs

As the FPGA volumes and transistor densities continue to increase,FPGA sales started to reduce the standard cell based ASIC sales. Thus ASIC vendors came up with a structure similar to FPGA to lure the FPGA customers back to ASIC. This so called structure is known as Structured ASIC [20].

5.1.1 Architecture of Structured ASIC

The architecture of structured ASIC falls between standard cell based ASIC and FPGA [5]. It is made up of two parts.

Structured Element Can be combinational and/or sequential logic blocks

Array of Structured Elements The array can be uniformly spread or non-uniformly spread.

The architecture of the structured ASIC is shown in Figure 5.1 . Structured elements such as Embedded RAM, Prefabricated I/Os are partially formed elements, which means that these elements are pre-configured. Whereas sea of tiles is an array of a structured element called tile, where each tile can be either be partially formed or uncommitted (like gate arrays). The new functionality is configured using such elements.

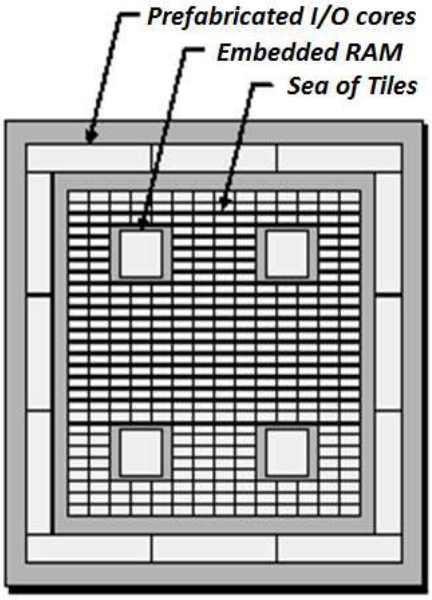


Figure 5.1: Architecture of Structured ASICs [5]

5.1.2 Implementation flow

In order to implement a new functionality using the structured ASICs architecture, following design flow is followed.

Logic synthesis : Mapping the RTL of the new functionality into the structured elements. For example, if the structured element on the chip is of type combinational logic then only such kind of cells are used for mapping the RTL.

Placement : Mapping of each structured element in the synthesized netlist onto array of structured elements on the chip [5]. Once the basic mapping is done, in order to improve the timing, placement optimization's are performed.

Clock tree Synthesis : Distributing the clock network among the structured elements with a goal to minimize clock skew and delay.

Routing : Full routing is performed, that is connecting different structured elements via wires.

5.1.3 Comparison

Table 5.1: Comparison between different types of design styles

	cell based ASIC	**Structured ASICs**	**FPGA**
Design Complexity	Hard to Design	Easy to design	Easy to design
Development time	Long	Short	Short
NRE costs	Big	Small	None
Design Size	Supports Large	Supports Medium	Small or Limited
Complexity support	Large	Medium	Limited
Performance	High	Medium	Limited
Man power	> 5	$> 2 < 5$	1 or 2
Price per chip	Cheapest	Medium	Expensive
Unit Cost(High Volume)	Low($11)	Medium($21)	High($40)
Power Consumption	Low	Medium	High

It is clear from the table above that structured ASICs combine the best of cell based ASIC and FPGA.

5.2 Structured ASICs and the Gate Arrays

Comparing the structured ASIC architecture with the GA structure (as in Figure 2.17), it can be noticed that, the basic concept is similar. While the latter is made up of array of unconnected MOS transistor unit tile, the former is made up of array of structured element that could be in the form of tiles. Had the internal structure of a tile been like unconnected MOS transistor gate, the structured ASIC architecture would become similar to GA.

For implementation, both the technologies make the use of the prefabricated elements to implement the new functionality. Thus, the base layers remain the same and only metal layers are used to configure the functionality. As a result, the time to market and costs are reduced in both the design styles. From the above similarities it can be said that the structured ASIC enjoy all the advantages of GA design style. Hence, GA can

be considered as forefathers of the structured ASIC technology.

A major difference between both the technologies is that, the structured ASIC also offers an array of partially formed elements like the ones shown in the architecture. Whereas in a GA structure, it's always the array of uncommitted MOS transistors. This advantage makes a structured element also like a FPGA element but without the additional overhead for field programmability [18].

5.3 Advanced ECO methodology and the structured ASICs

The advanced ECO methodology uses GA type cells to carryout an ECO and thus it makes the methodology very similar to the structure ASICs. In this section, both the concepts would be compared to find out similarities and differences. The comparison will be made with respect to the assumptions of the top level design. These assumptions are mentioned in section 4.5.1.

5.3.1 Comparison based on Assumption 1

In this assumption, it is assumed that the top level design contains the spare area filled with GA fillers. Comparing both the methodologies, it can be said that, in case of the advanced ECO methodology, the GA fillers are used as base cells, whereas, on the other hand, these would be used as a structured element. Furthermore, in the former case, the new functionality is configured by swapping the corresponding MACRO with the base cells. Whereas in the latter case, the structured ASIC implementation flow would be performed upon the structured element. Because the MACRO is implemented using GA type cells, the result would still be similar in both the cases i.e. reuse of prefabricated layers.

Thus from the above comparison it can be concluded that, advanced ECO methodology uses the concept of structured ASICs for doing metal-only ECO, but the way of configuring the new functionality is different. The former uses the MACRO mapping, whereas the latter uses structured ASIC implementation flow.

5.3.2 Comparison based on Assumption 2

In this assumption it is assumed that the top level design already contains some functionality implemented using GA cells. Adding a new functionality in case of structure ASICs, would be re-configuring the structured elements by following the structured ASIC implementation flow. However, in the advanced ECO process, a new MACRO is created and swapped with the old one.

Note that the size of the new functionality cannot exceed the size of the configurable area. This is true for both the methodologies. In the structured ASICs, as the complete

chip is configurable, the size cannot exceed the chip area, while in the advanced ECO methodology the size cannot exceed the spare area.

5.3.3 Summary of the comparison

In summary, regardless of top level assumption, if the architecture of structure ASIC is made up of GA fillers , the whole chip would be filled with these type of cells. But in the case of advanced ECO methodology, only a small portion of the chip is dedicated to such type of cells. As the remaining portion is implemented using regular standard cells, the advanced ECO methodology can be said to be using cell based ASIC flow for designing major portion of the chip and structured ASIC flow for performing the ECO in the prefabricated area (spare area).

5.4 Applications Domain

This section will describe the manner in which the semiconductor industry has approached the concept of structured ASICs in the past. In addition, several ways of using advanced ECO methodology will also be presented here.

5.4.1 Structured ASIC applications

Few years back, companies like ALTERA, eASIC introduced the structured ASIC technology to the commercial market with products like Altera's HardCopyII and eASIC's Nextreme. These products offer configuration of several VIA and metal layers to make the functionality. The following paragraph demonstrates the use of Altera's Hardcopy structured ASIC.

The design to be produced was first tested on the FPGA platform to conform the design features. After getting the approval from the customers, the design was migrated to structured ASIC platform for production. The customers were able to change the design on the fly by customizing the chip. Altera provided complete platform for Prototyping using an FPGA and then production using structured ASIC. For example, Infineon technologies, prototyped and tested their designs using the FPGAs and then used the Altera's Hardcopy structured ASIC to make several versions of the customized designs according to the customer requirements [21].

5.4.2 Advanced ECO methodology applications

The advanced ECO methodology could be used in following ways.

- Whenever the size of an ECO is quite big, typically 12k gates, then this flow comes very handy as it makes the use of conventional ASIC flow commands.

- For feature swapping. As this flow uses GA type cells, it allows to swap a portion of the chip implemented using GA type cells with a new feature also implemented using GA type cells.

Thus one can think of designing a chip containing some features already implemented using GA cells. So later on, in case of adding a new feature, that portion of the chip can be swapped out within minimum time and with minimum cost.

5.5 Summary

This chapter presented the structured ASICs technology and discussed the similarities and differences with the advanced ECO methodology. First section provided the basic concept of the structured ASICs; Section 5.2 showed that the structured element in the structured ASICs is similar to a GA tile as well as a FPGA element. Subsequently, the advanced ECO methodology was compared with the structured ASIC implementation flow. In the comparison summary, it was concluded that the advanced ECO methodology is a mixture of cell based ASIC and structured ASIC. The next chapter will present area and power comparison results for the advanced ECO methodology followed by discussion of those results.

6

Results and Evaluation

Chapter 4 explained the block level flow and the top level flow of the advanced ECO methodology in detail. The block level flow uses gate array (GA) type cells with conventional ASIC flow in order to create a MACRO of a new functionality. As a result, this chapter performs the evaluation of using such type of cells in block level flow and compares it against using regular standard cells. These two implementations are evaluated with respect to area, power and gate count and the results are presented. In the first section, the setup information and the procedure used to obtain these results is presented. Section 6.2 will compare the regular standard cells with GA cells in terms of cell area and gate count. The subsequent section compares them in terms of power. In the last section the results are summarized.

6.1 Evaluation setup and procedure

In this section, the method used to obtain the area and power results is presented.

6.1.1 Setup Information

Information regarding type of library used to obtain the results is mentioned in this subsection. The following describes the comparison setup.

Technology Used 65nm

Operating Condition Best Case

Library Type High Threshold Voltage (HVT)

Design Size $\approx$ 12k gates

The cells used in the library are characterized at 65nm process technology. In terms of operating condition, `Best Case` was chosen in order to use best combination of temperature and voltage. Other options are `Worst Case` and `Typical Case`. The reason for choosing best case was to be able to obtain results at higher clock frequencies. Otherwise both the designs fail to meet stricter timing constraints for other cases . The third parameter is the type of threshold voltage used. This parameter determines the delay and the power consumed by the gates. These values are defined by the library vendor. For detail information regarding these setup parameters the reader is encourage to refer to relevant Integrated Circuit Design book (like CMOS VLSI Design by Neil Weste et al. [22]) .

6.1.2 Procedure

The tool ICC was used to obtain the area and the power results. The procedure for obtaining these results is outlined as follows.

Cell area and gate count

The procedure used to obtain these parameters is as follows. The RTL for the new functionality was synthesized using GA type cells, then the synthesized design was placed and routed. Once the design was free from routing DRC, the command `report_physical_design` was used to obtain area and gate count . The gate count gives information about number of gates found in the design and the area number tells the area occupied by GA type cells or regular standard cells (Combinational as well as Sequential) in the design . The report generated by the command includes both these numbers. Similar procedure was used to obtain these parameters for regular standard cell based implementation.

In order to note down the effect of clock frequency on both the designs, the design constraint file was modified. For example to run the simulations for 32MHz clock, the clock period in the constraints file was first modified. Then, the complete block level flow was repeated starting from synthesis, for both, regular standard cells as well as GA type cells. This whole process was repeated to obtain results at 32MHz, 64MHz,100MHz and 120MHz.

Power numbers

Dynamic power and leakage power numbers were obtained for both the implementations. The procedure used to obtain these numbers was similar to the one described for obtaining area and gate count. However, the command used in this case was `report_power`. The numbers obtained at this part of the flow are more accurate as compared to obtaining after the synthesis.

For getting power results at different clock frequencies, the process of changing the

clock frequency and running the entire flow was used. Again this results were obtained at four different clock frequencies.

6.2 Cell area and gate count comparison

Figure 6.1 compares the gate count of GA type synthesized design with regular standard cell type. The x-axis shows the different clock frequencies at which these number were obtained. The y-axis shows the number of gates in the design. The blue bar represents the GA type cells and the red bar represents the regular standard cells.

It can be noticed from Figure 6.1 that the count for regular standard cells is between 10-12k, which is the same as the design size. On the other hand, the design which is synthesized using GA type cells has approximately 53% more gate count. The reason behind this high count is the cell library. The GA library used to synthesize the design did not contain all the required GA type cells. In fact, there were only 16 GA cells defined in the library as opposed to 116 regular standard cells. Consequently, the gate count of the latter design is more. For instance, if the GXNOR (GA type XNOR) gate is not available as predefined in the library, the tool would use several GNAND (GA type NAND) gates to make it. Consequently, the gate count of the design will increase.

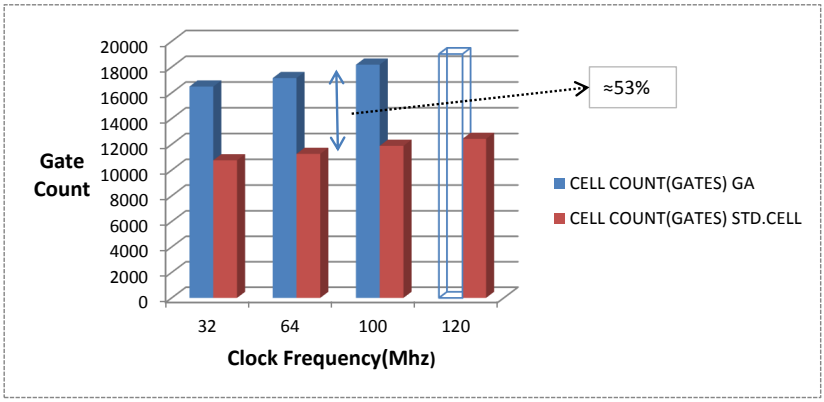


Figure 6.1: Graph showing the gate count of the design synthesized using GA cells vs. regular standard cells

In terms of timing, the GA design was not able to meet the timing constrains at 120MHz clock frequency. This is shown in the graph as empty red column.

Figure 6.2 shows the area comparison between GA cells and regular standard cells. The x-axis represents the clock frequency in MHz, and y-axis represents the cell area in square microns. It can be seen that, GA cells approximately occupy 75% more area (blue) as compared to regular standard cells (red). The reason can be explained as follows. In the previous paragraph, it was said that due to not having a proper GA

library the synthesis tool used 53% more gates. Moreover, a GA cell defined in the library is 25% larger than the regular standard cell. Considering the XNOR example of previous paragraph, if an GXNOR gate is made using several GNAND gates, the total area occupied by such GXNOR would be more than having predefined GXNOR gate in the library. Thus, the area number (75%) could be derived by combining the gate count (53%) with the average gate size (25%).

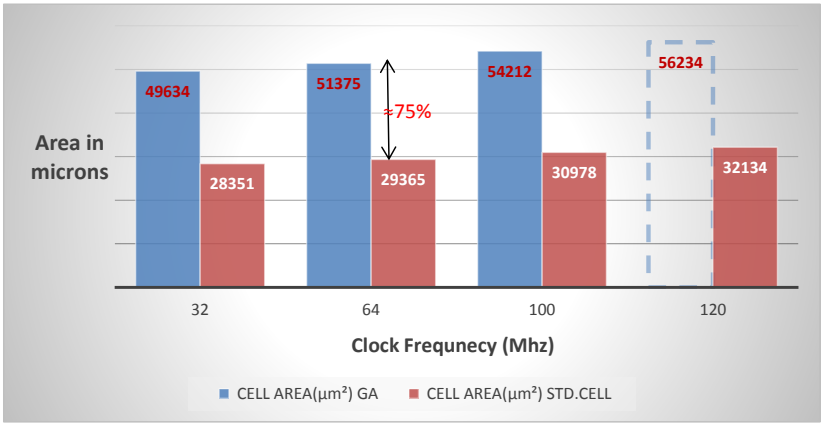


Figure 6.2: Graph showing area occupied GA cells as compared to regular standard cells

Comparing the designs about the clock frequency, it can be said that, as the clock speed increases, the tool tries to meet these stricter timing constraints by using faster gates with higher driver strengths. As a result the area increases. This increase is irrespective of the cell type as shown in Figure 6.2.

6.3 Power comparison

Figure 6.3 displays the graph of dynamic power consumption (in μW). Comparing the power consumed by GA cells (blue) with the regular standard cells (red), it can be observed that, the former consumes approximately 60% more power irrespective of the clock frequency. The reason can be explained as follows. In the previous graph (Figure 6.2) it was shown that the GA cells occupy approximately 75% more area. Consequently, the wire capacitance will increase due to long wires. Furthermore, due to GA cells being wide the diffusion capacitance increases [22]. Since the dynamic power (P_{dyn}) is directly proportional to capacitance (from Equation 6.1), the increase is evident.

The significant increase in dynamic power with respect to clock frequency can also be explained using the Equation 6.1. It shows that the dynamic power is dependent on the clock frequency. Hence, there is a linear increase.

$$P_{dyn} = \alpha * C * V^2 * f \tag{6.1}$$

α= Switching Activity factor, C= capacitance, V= voltage, f= frequency.

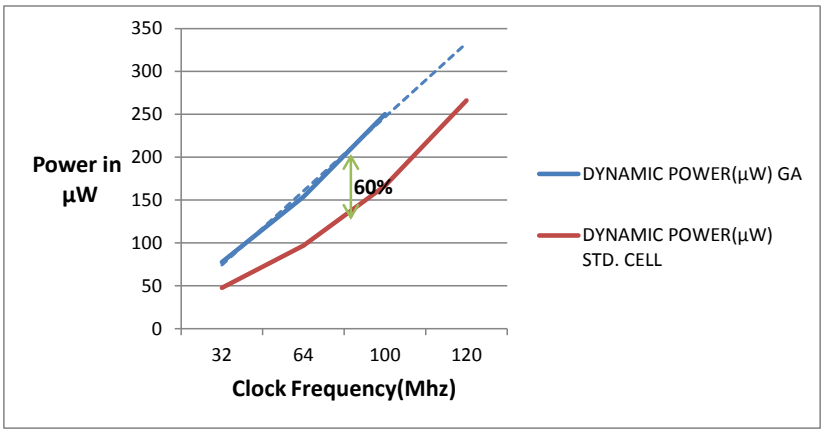


Figure 6.3: Graph comparing dynamic power consumption between GA cells and regular standard cells

The leakage power (in nW) with respect to clock frequency is shown in Figure 6.4. It can be seen that the GA based implementation (in blue) consumes more than twice the leakage power. The reason can be explained as follows. It is apparent from the Equation 6.2 that the leakage power P_{leak} is directly proportional to leakage current I . Furthermore, it is known that this current increases with gate width. Thus, it can be said that since GA cells are wider (see section 4.4.4), they leak more. Apart from being wide, due to higher gate count, the combined cell leakage dominates the leakage power. As a result, the leakage at the chip level becomes double.

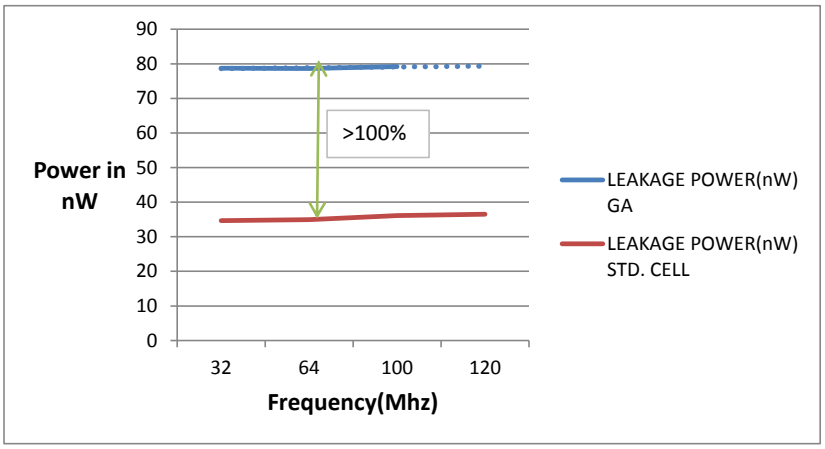


Figure 6.4: Leakage power consumption comparison between GA cells and regular standard cells

Comparing the leakage curve with the dynamic power curve, it can be deduced that, unlike the former one, the increase in dynamic power with respect to clock frequency is linear. This is because of the fact that dynamic power depends upon area as well as clock frequency whereas the leakage power only depends upon area. As only 5-8% increase in the cell area (irrespective of the type of cell) is observed, the increase in the former is marginal.

$$P_{leak} = V * I \tag{6.2}$$

I= Leakage Current, V= Supply Voltage.

6.4 Summary

In this chapter, the block level flow of the advanced ECO methodology was evaluated against the conventional flow that involves using regular standard cells. The first section described the procedure used to obtain the results. In the subsequent section, it was found that, due to not having enough GA type library cells defined, the design used 53% more cells. Consequently, the area increased by 75%. In section 6.3, both the designs were compared for power. The results were again in the favor of regular standard cells. The next chapter will provide conclusions on the overall methodology, considering the advantages and the disadvantages of the block level flow.

7

Conclusion and Future Work

In this thesis, the conventional method of doing ECOs was modified to use gate array (GA) fillers as spare cells. Moreover, using this modified method as base, a new methodology was proposed and explained. This methodology overcomes the disadvantages of the previous methods and works irrespective of the size of an ECO. During this process, many challenges were identified concerning the usage of the tool and solutions were also proposed. After the implementation, the GA cells were evaluated in terms of area and power.

From the evaluation results, the first thing found was that, the GA based design contains 53% more gates due to lack of proper GA library. By using a proper library, containing all the required GA type cells, it would be possible to reduce this high count. On the down side, it is not certain whether one can create all the regular standard cells in a GA structure (ex. Clock Tree buffers etc.). Therefore, even by using a proper library, it is possible that the design might end up having more gates. Due to this reasons, it is assumed that the GA based design could still have 10% more gates. Thereby keeping a margin between regular standard cell and GA implementation.

In terms of cell area, it was found that the GA implementation occupies 75% more area irrespective of clock frequency. But this was the consequence of having higher gate count as well as GA cells being larger. Thus, if it is possible to reduce the gate count to 10%, then from section 6.2 it can be proved that the area could also be expected to get reduced to 35%. On the positive side, it will be shown in the subsequent paragraph that the GA implementation can achieve utilization's as high as 90%.

Chip utilization is a term which represents what percent of the total chip area has been utilized by the cells. In case of regular standard cell based implementation, as these cells are optimized w.r.t area and power it is not possible to exceed this number

beyond 70-75% [14]. Exceeding this limit, could result in congestion because the routing tool requires sufficient area between the cells to route the signals. Whereas in case of GA based implementation, because of the GA structure being wide, it is possible to achieve much higher utilization.

As dynamic power is directly proportional to area ,the GA based implementation also consumes 60% more dynamic power. If the area could be reduced to 35%, it can be said that, there would be a notable reduction in dynamic power too. This reduction can be approximated to 30-35% (from section 6.3. For the leakage power, it was explained that, the higher leakage was because of wider GA cells and higher gate count. Thus if the number of gates decreases, less area would be occupied by those gates. Hence, the total leakage power could also be expected to be reduced accordingly.

When it comes to doing an ECO, it was shown in Chapter 4 that with the help of gate array cells it is possible to perform large ECOs. Moreover, as only metal layers are used for this purpose, the design time required to perform such big ECOs reduces drastically and so as the manufacturing costs.

Thus from the above discussions and arguments, it can be concluded that, even though using GA type cells can cost in terms of area and power, advantages like faster time to market, lower cost and higher chip utilization make them an ideal choice for realizing an ECO.

7.1 Thesis Contributions

The following things have been successfully carried out in this thesis:

- Modified the conventional ICC ECO flow to use programmable GA spare cells.
- Developed a new ECO methodology to overcome the disadvantages of the ICC's conventional ECO flow.
- Evaluated the GA type based implementation in terms of area and power and compared it with regular standard cell based implementation.

7.2 Future Work

In continuation of the work presented in this thesis, the following are the suggestions that could be explored in future.

Improvements in the Advanced ECO methodology

- The Advanced ECO methodology presented uses conventional ASIC design flow as a part of the methodology. However, only basic ASIC design steps have been explained in this flow. Steps like DFT insertion, Static timing analysis, Test and Verification etc. which are important constituents of this flow have not been included. Thus one can perform this steps for achieving more accurate verification coverage.

Improvements in the power results

In order to obtain more accurate power results the following steps are suggested.

- Use proper GA library containing all the required GA type cells.
- Generate VCD (Switching activity) files by simulating the netlist at particular clock frequency.
- Use these VCD files to generate accurate power reports in ICC.
- Repeat the above steps at different clock frequencies.

Bibliography

[1] H.-Y. Chang, I. H.-R. Jiang, and C. Yao-Wen, "ECO Optimization Using Metal-Configurable Gate-Array Spare Cells," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 32, pp. 1722–1733, Nov. 2013.

[2] P. Weil, M. Roy, N. Ashley, and B. Ogada, *CMOS Gate Array*, San Jose State University, May 2003.

[3] Synopsys, "ICC ECO flow Application Note," *Solvnet Doc Id: 025277*, March 2009. [Online]. Available: https://solvnet.synopsys.com/retrieve/025277.html

[4] *Formality® User Guide*, 2013rd ed., Synopsys, March. 2014.

[5] D. Lander, H. Yamamoto, and S. Erickson, *STRUCTURED ASIC's*, UCLA, Spring 2004. [Online]. Available: eda.ee.ucla.edu/EE201A-04Spring/ASICslides.ppt

[6] Wikipedia, "Engineering change order," Oct. 2007. [Online]. Available: http://en.wikipedia.org/wiki/Engineering_change_order

[7] K.-h. Chang, I. L. Markov, and B. Valeria, "Reap What Y ou Sow: Spare Cells for Post-Silicon Metal Fix," in *2008 International Symposium on Physical Design.*, Portland, Oregon, USA, Apr. 2008.

[8] W. Jacques, D. Chiang, and J. Tolentino, "Efficient use of spare gates for post-silicon debug and enhancements," *U.S. Patent 6 255 845*, July 3 2001.

[9] D. Lee, "Method and apparatus for quick and reliable design modification on silicon," *U.S. Patent 5 696 943*, Dec. 9 1997.

[10] A. Balasinski, "Optimization of sub-100-nm designs for mask cost reduction," *SPIE J. Micro/Nanolith. MEMS MOEMS*, vol. 3, pp. 322–331, Apr. 2004.

[11] T. Petit and P. Yves, *ECO Flow Integration Based on Mask-configurable Logic and Conformal ECO*, Cadence, Lund, 2012, at Lund Circuit Design Workshop.

[12] M. K. Dadhich, A. Bandlish, and V. Nimran, "An Innovative Flow To Implement Large Scale Design Changes In The Final Stages Of Physical Implementation ," Oct. 22 2009.

[13] Synopsys, "Gate Array Filler Flow for IC Compiler ECO," *Solvnet Doc Id: 026225*, Jan. 2010. [Online]. Available: https://solvnet.synopsys.com/retrieve/026225.html

[14] *IC Compiler^{TM} Implementation User Guide*, 2013rd ed., Synopsys, Dec. 2013.

[15] C. M. Giles, "Modular Collection of Spare Gates for Use in Hierarchical Integrated Circuit Design Process," *US Patent 6650139 B1*, Nov. 2003.

[16] M. Brazell and A. Essbaum, "Method for Allocating Spare Cells in Auto-Place-Route Blocks," *US Patent 6993738*, Jan. 2006.

[17] P. Chaisemartin, "Structure and Method of Repair of Integrated Circuits," *US Patent 6586961*, Jul. 2003.

[18] *Structured ASIC, Evolution or Revolution?*, Phoenix, Arizona, USA, Apr. 18-21 2004.

[19] Synopsys, "Reason for a unitTile," *Solvnet Doc Id: 009454*, Feb. 25 2004. [Online]. Available: https://solvnet.synopsys.com/retrieve/009454.html

[20] B. Zeidman, "THE DEATH OF THE STRUCTURED ASIC," *Article Id 386*, June 26 2006. [Online]. Available: http://chipdesignmag.com/display.php?articleId=386

[21] *Spend less.Do more.Get there first.*, Altera, January. 2006. [Online]. Available: http://www.altera.com/literature/po/hardcopy-structured-asic.pdf

[22] N. Weste and D. M. Harris, *CMOS VLSI Design :A Circuits and Systems Perspective*, 4th ed. Addison-Wesley, 2011.

[23] N. Kim, T. Austin, and D. Blaauw, "Leakage Current: Moore's Law Meets Static Power," *IEEE Computer Society*, vol. 36, pp. 68 – 75, Dec. 2003.

[24] *Using Tcl With Synopsys® Tools*, 2013rd ed., Synopsys, Dec. 2013.

A

IC Compiler Pre-mask ECO TCL script

```
#-------------------------------------------------------------------------
# File name= Unconstrained_ECO_Flow_after_Pnr.tcl
# Author= Chirayu Shah.
# Description= Contains  basic  ECO commands for the ICC
# in order to perform the "pre-mask ECO flow".
# The scripts assumes that the design setup files already configured
# and variables used here are already set.
#-------------------------------------------------------------------------


source -echo ./icc_setup.tcl
##Open Design
open_mw_lib $MW_DESIGN_LIBRARY
## Make a copy of the original design
copy_mw_cel -from $ICC_ECO_STARTING_CEL -to $ICC_ECO_CEL
#Open the CEL view of the Placed and routed design
open_mw_cel $ICC_ECO_CEL;

### Read ECO file
if {$ICC_ECO_FLOW_TYPE == "verilog"} {
eco_netlist -compare_pg -by_verilog_file $ICC_ECO_FILE
}
if {$ICC_ECO_FLOW_TYPE == "tcl"} {
eco_netlist -by_tcl_file $ICC_ECO_FILE
}
## update  Power & Ground connections
derive_pg_connection -power_net $MW_POWER_NET -power_pin $MW_POWER_PORT
                     -ground_net $MW_GROUND_NET -ground_pin $MW_GROUND_PORT
```

```
## Place ECO cells
# Remove the filler cells if they are added
if {$ADD_FILLER_CELL} {
place_eco_cells -eco_changed_cells
                -remove_filler_references "$FILLER_CELL_METAL␣$FILLER_CELL"
}
## Insert fillers
insert_stdcell_filler -cell_without_metal $FILLER_CELL
        -connect_to_power $MW_POWER_NET -connect_to_ground $MW_GROUND_NET
## ECO route
route_zrt_eco -reroute modified_nets_first_then_others
```

B

IC Compiler Post-mask ECO TCL script

```
#-------------------------------------------------------------------------
# File name= Freeze_ECO_Flow.tcl
# Author= Chirayu Shah.
# Description= Contains  basic  ECO commands for the IC compiler
# in order to perform the "post-mask ECO flow".
# The scripts assumes that the design setup files already configured
# and variables used here are already set.
#-------------------------------------------------------------------------
source -echo ./rm_setup/icc_setup.tcl
##Open Design
open_mw_lib $MW_DESIGN_LIBRARY
## Make a copy of the original design
copy_mw_cel -from $ICC_ECO_STARTING_CEL -to $ICC_ECO_CEL
open_mw_cel $ICC_ECO_CE

## Read ECO file
     if {$ICC_ECO_FLOW_TYPE == "verilog"} {
       ## For functional ECO :
       eco_netlist -compare_pg -by_verilog_file -freeze_silicon $ICC_ECO_FILE
     }

     if {$ICC_ECO_FLOW_TYPE == "tcl"} {
       eco_netlist -by_tcl_file -freeze_silicon $ICC_ECO_FILE
     }
 ## Connect Power & Ground
derive_pg_connection -power_net $MW_POWER_NET -power_pin $MW_POWER_PORT
                     -ground_net $MW_GROUND_NET -ground_pin $MW_GROUND_PORT
```

```
##Do feasibility analysis
check_freeze_silicon
##map ECO cells to spare cells
place_freeze_silicon
##Perform the ECO routing
route_zrt_eco
```

C

Adding Gate Array fillers as spare cells

```
#-------------------------------------------------------------------------
# File name= Freeze_ECO_Flow.tcl
# Author= Chirayu Shah.
# Description= Script for adding gate array fillers inside the chip
# The script assumes that there are multiple unit tiles defined .
# in the library and gate array fillers are already set as std. fillers.
#-------------------------------------------------------------------------
set fillers1_ga_cells {GFILL10BWP7THVT,GFILL4BWP7THVT,GFILL3BWP7THVT..}
map_unit_tiles -lib_cells $filler1_ga_cells -unit_tile_name unit
report_unit_tiles
insert_stdcell_filler -cell_with_metal $fillers1_ga_cells
set_attribute -class cell [ filter_collection \
[ get_cells -all *] "ref_name=~*GFILL*" ] \
is_spare_cell true
```