The Thesis Committee for Aseem Sayal
Certifies that this is the approved version of the following thesis:

EDA Design for Microscale Modular Assembled ASIC (M2A2) Circuits

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: S.V. Sreenivasan

Co-Supervisor: Mark McDermott

EDA Design for Microscale Modular Assembled ASIC (M2A2) Circuits

by

Aseem Sayal, B.Tech

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2017

Acknowledgements

I express my heartfelt gratitude towards Prof. S.V. Sreenivasan, Professor, Department of Mechanical Engineering and Electrical and Computer Engineering, The University of Texas at Austin for providing me this wonderful opportunity to work on this project. I would like to thank him for his guidance, motivation, financial support and consistent supervision throughout the project. I would like to convey my gratefulness to Prof. Mark McDermott, Entrepreneur in Residence and Adjunct Faculty member, The University of Texas at Austin for his guidance, motivation, novel ideas and his help in resolving design, tool, and license issues. The results achieved would not have been possible without his expert mentorship.

I would also like to thank Vipul Goyal, M.S.E. Student, Department of Electrical Engineering, for his contribution to other EDA segments of this project, namely Physical Design and Formal Verification, and for helping me with few commands in DEF generation process. I would also like to thank Paras Ajay, PhD. Student, Department of Mechanical Engineering, The University of Texas at Austin for his contributions in mechanical/assembly segment of project, and helping me understand the fabrication and mechanical challenges of this project. I would like to convey my gratefulness to Nandini Chintala, Junaid Alim, and Prahant Joshi from Cadence Design Systems Inc. for their help on providing support in Cadence tools and design setup. I would also like to thank staff members of NASCENT Center, UT Austin for their help in completion of this project. In particular, I would like to thank Dr. Larry Dunn, Dr. Shrawan Singhal and Dr. Ovadia Abed for their motivation and support.

I would like to express my gratitude towards my parents and friends for their kind co-operation and encouragement which helped me in completion of this project. And above all, to the Almighty God, who never ceases in loving us and for the continued guidance and protection.

Abstract

EDA Design for Microscale Modular Assembled ASIC (M2A2) Circuits

Aseem Sayal, M.S.E.

The University of Texas at Austin, 2017

Supervisors: S.V. Sreenivasan, Mark McDermott

As the semiconductor industry has driven down the minimum feature size to well below 50nm, the mask cost to make devices has skyrocketed. The cost for a full set of masks is estimated to be about $1.5M for 90nm node devices and exceeding $2M for 65nm lithography nodes. According to some estimates, mask writing time goes up as a power of five as feature sizes are decreased below 50nm. In addition, higher complexity of large designs increases the number of design re-spins. The above two factors lead to considerable increase in the nonrecurring engineering cost (NRE) for standard cell ASICs, which has become prohibitively expensive for low to mid volume applications. Field programmable gate array (FPGAs) offer an acceptable solution for fast prototyping and ultra-low volume applications, but are generally not seen as a replacement for ASICs because of their highly inefficient space utilization, lower performance/speed and high power consumption. This is particularly the case as mobility has driven expectations for small form factor and low power consumption.

In this work, a new type of ASICs named as Microscale Modular Assembled ASIC (M2A2) is proposed. This technology is a novel application of the high-speed, precision assembly technique for fabrication of ASICs using a limited number of mass-produced feedstock logic circuits. The idea is to share the mask cost for sub-100nm feature sizes

across a large number of ASIC designs, decreasing the NRE for individual designs. The concept of constructing ASICs using repeating logic elements is based on previous works where it has been shown that ASICs made of via/metal configured structured elements can achieve space utilization and performance comparable to cell based ASICs. However, in the proposed technique, we provide significantly more choice in the transistor layer, in terms of feedstock types and their configuration.

This thesis document deals with the electronic design automation (EDA) design for microscale modular assembled ASIC based circuits. The document discusses the design of feedstock cells, generation of feedstock preplaced design, generation of design collaterals to support M2A2 EDA flow, and front end M2A2 synthesis flow to meet the required functionality of design and achieve optimal quality of results (QoR) metrics in terms of circuit performance/speed, power and area.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

This chapter is divided into 6 sections. Section 1.1 presents the background of this research work. The motivation for this research work is discussed in section 1.2. Section 1.3 deals with the existing ASICs, FPGAs and structured ASICs technologies. An overview of M2A2 fabrication technology is presented in section 1.4. Section 1.5 focuses on the EDA design scope and objectives for M2A2 circuits. The specifications and details of ASIC designs implemented using M2A2 technology are described in section 1.6.

1.1 Background

Cutting-edge consumer and scientific applications are driving the need for compact yet powerful devices with a wide variety of functional elements (electronics, photonics, heat sinks, etc.) integrated in a limited amount of space. The considerable increase in the nonrecurring engineering cost (NRE) for standard cell ASICs has made this technology prohibitively expensive for low to mid volume applications such as custom chips for wearables, scientific and medical applications [1]. Field programmable gate array (FPGAs) offer an acceptable solution but are generally not seen as a replacement for ASICs because of their lower performance/speed and higher area and power consumption, particularly in mobile applications. Thus, arises the need to develop low cost solutions for low volume ASICs production.

Heterogeneous integration of varied components such as electronics, photonic and energy storage devices can revolutionize future generations of compact consumer, medical and scientific devices. The current semiconductor fabrication techniques are not suited for heterogeneous integration because of the sheer variety of possibly incompatible fabrication techniques needed for each functional element. Pick-and-place is a cost-effective method for assembling heterogeneous devices from their constituent parts in short time scales. Although such techniques have been demonstrated earlier for micron sized components [2,3,4,5,6,7], none have the combined features of highly parallel pick-and-place, arbitrary constituent distribution, and nanometer-precise placement. The proposed M2A2 technology combines highly parallel pick-and-place, and nanometer-precise placement to explore cost effective approaches for fabrication of ASICs.

This document deals with electronic automation design (EDA) design for proposed Microscale Modular Assembled ASIC (M2A2) circuits. The proposed solution addresses the high NRE cost issue of ASICs production and provides a cost effective solution for low volume production, and at the same time achieving power, area and speed performance approaching standard cell based ASIC design. This technology approaches ASICs in terms of power and performance, and approaches FPGAs in terms of NRE and turnaround time (TAT). Thus, it provides a solution to manufacture high performance system on chip (SoCs) with performance and cost metrics that lie somewhere between standard cell based ASIC designs and FPGAs.

1.2 Motivation

Due to feature shrink below 50nm, the masks cost to make the application specific integrated circuits (ASIC) based system on chip (SoC) has skyrocketed. In addition to the increase in the nonrecurring cost (NRE), the turnaround time (TAT) to manufacture these devices has also increased due to increase in the number of transistors which can be packed within a given area, thereby increasing the design complexity and verification effort. This considerable increase in the nonrecurring engineering cost and turnaround time for ASICs has made this technology prohibitively expensive for low to mid volume devices. On the other hand, field programmable gate array (FPGAs) offer an acceptable cost effective and turnaround time solution. But these devices have low performance/speed, inefficient area utilization and high power consumption. These characteristics make FPGAs infeasible for high performance applications. Thus, FPGAs cannot replace ASICs. The cost vs. performance tradeoff of ASICs and our proposed M2A2 solution is shown in Figure 1 [8]. This is generated for 22nm Intel MPU design using "IC Knowledge Strategic Cost model".

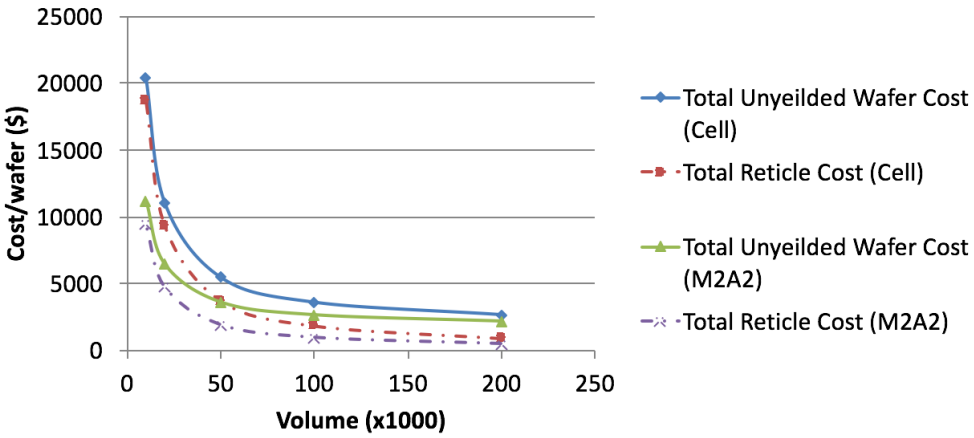


Figure 1: Cost vs. Volume Tradeoff for ASIC and M2A2 [8].

This research work aims to address this problem by providing a cost effective solution which approaches the performance of ASICs and cost/turnaround time of FPGAs technology. The proposed M2A2 technology is based on novel fabrication and EDA design approach, discussed in sections 1.4 and 1.5 respectively.

1.3 Overview of ASICs, FPGAs and Structured ASICs

This section deals with the overview, merits, demerits and comparison of ASICs, FPGAs and Structured ASICs technologies.

1.3.1 ASICs

Application Specific Integrated Circuits (ASICs) are used to implement both analog and digital functionality circuits in high volume [9]. These devices have high performance in terms of circuit speed/timing, low power dissipation and are compact in size with high area utilization. ASICs are fully customized devices optimized for specific end applications and are made up of billions of transistors. It requires high development and NRE costs in order to design and implement. The major components of NRE are masks cost, EDA tools license cost and engineering cost due to high turn-around time. Further, due to high complexity, design re-spins are common which incurs NRE cost every time, since these devices are not reprogrammable. Figure 2 shows the architecture of ASICs standard cells and gate array structures. It can be observed that gate array ASICs have a fully customized transistor layer, whereas standard cell ASICs have standard cells as the basic building blocks, which are formed by a combination of transistors.

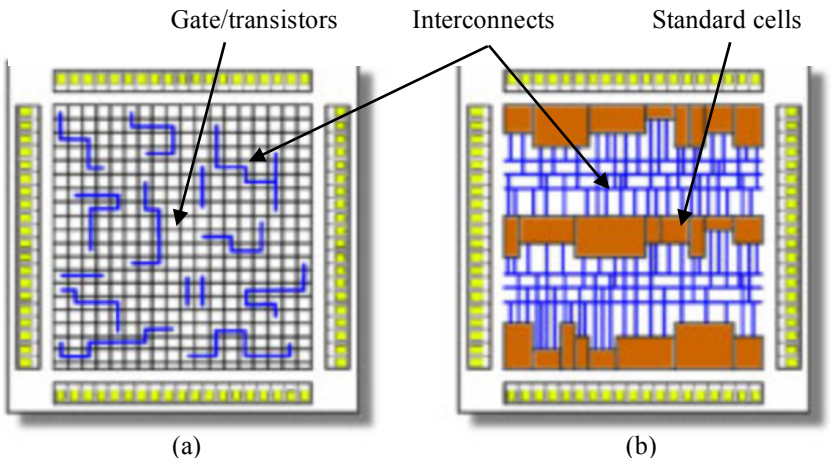


Figure 2: (a) Gate Arrays and (b) Standard Cells ASICs architectures (adapted from [9]).

1.3.2 FPGAs

Field Programmable Gate Arrays (FPGAs) are cost effective reprogrammable devices which are easy to use [9]. FPGAs are known for their flexibility and their ability to be reprogrammed in the field. There is no need to have a full blown design flow and tooling, therefore the NRE investment is low and as a consequence, time to market is less. The FPGA architecture makes inefficient use of silicon, but this is the only way to make a large, fully programmable SoC. This leads to a large device size. Due to limited routing resources, it degrades the system performance by making it run slower and consumes more power. The cost per device is high, but there is no NRE cost.

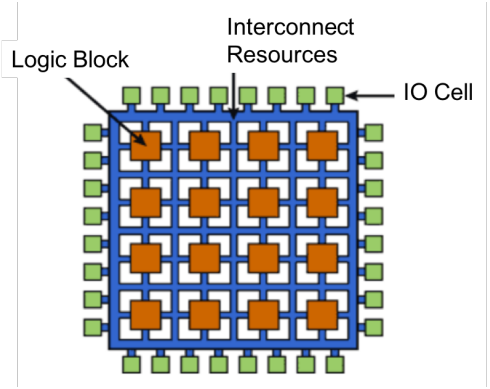


Figure 3: FPGA architecture [9].

1.3.3 Structured ASICs

Structured ASIC is an intermediate technology between ASIC and FPGA, offering high performance, similar to ASICs, and low NRE cost, similar to FPGAs [9,10,11,12]. This technology has less turnaround time to market, due to lower design complexity. It has basically the same structure as an FPGA, but is mask-programmable instead of field-programmable. It configures one or several via layers between metal layers. However, structured ASICs have limitations (discussed later) and have not been widely adopted. They do not appear to provide a feasible intermediate solution between FPGAs and ASICs.

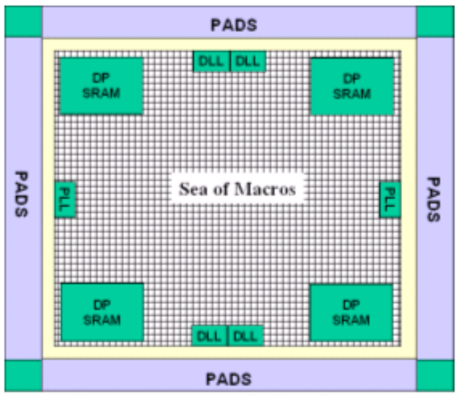


Figure 4: Structured ASIC architecture [10].

1.3.4 Comparison of ASICs and FPGAs

Both ASIC and FPGA are technologies with different benefits, their difference lies in costs, NRE, performance and flexibility. In general, for lower volume designs, FPGA flexibility allows cost savings and quick turnaround; while ASICs chips are more efficient and cost effective for high volume applications. In FPGA, the interconnects and logic blocks are programmable after fabrication, offering higher flexibility of design and ease of debugging in prototyping. However,

the capability of FPGAs to implement large circuits is limited, in both size and speed, due to complexity in programmable routing, and significant space occupied by programming elements, e.g. Static Random Access Memory (SRAMs), Multiplexers (MUXes) [13]. On the other hand, ASIC design flow is expensive. Every new design needs a unique set of masks. Table 1 shows the cost vs. performance tradeoff of ASICs and FPGAs [13].

QoR	FPGA	ASIC
Time to Market	Fast	Slow
NRE	Low	High
Design Flow	Simple	Complex
Unit Cost (for adequate volume)	High	Low
Performance	Medium	High
Power Consumption	High	Low
Unit Size	Medium	Low

Table 1: ASIC vs. FPGA Cost and Performance Comparison (adapted from [13]).

1.3.5 Limitations of Structured ASICs

It is claimed that structured ASICs are basically FPGAs but run faster, are less power hungry, and more economical [9]. The structured ASICs have similar architecture as that of FPGAs. This architecture is required for FPGA because of its routing limitations. However, the structured ASIC has no such limitation. Hence, the FPGA architecture is not only inefficient for structured ASIC, but it is also unnecessary. The structured ASIC has worse utilization than a gate array ASIC solution. The structured ASICs require CAD/EDA tools to address design problems typically faced in ASIC domain, on FPGA based architecture. This results in no or minimal CAD tool support, thus complicating the design implementation process and not reducing the turnaround time to market. It is speculated that it is one of the main bottlenecks in this technology, which prohibited its widespread use. Thus, structured ASIC is not accepted as an intermediate solution between ASICs and FPGAs to address the cost and performance tradeoff [9].

1.4 M2A2 Fabrication Overview

The proposed M2A2 technology makes use of a selective vacuum based pickup mechanism in conjunction with nanometer precise moiré alignment techniques to achieve highly accurate, parallel assembly of feedstock. Unlike current semiconductor fabrication techniques, M2A2

fabrication technique is suited for heterogeneous integration. The M2A2 assembly process is described with the following capabilities:

- Take sub-mm devices & assemble them onto a final substrate.
- Assemble the constituents in parallel, to allow for high throughput.
- Assemble with a placement precision significantly smaller than 100nm, and approaching as small as 10nm (3σ alignment error) or 5nm (3σ alignment error).

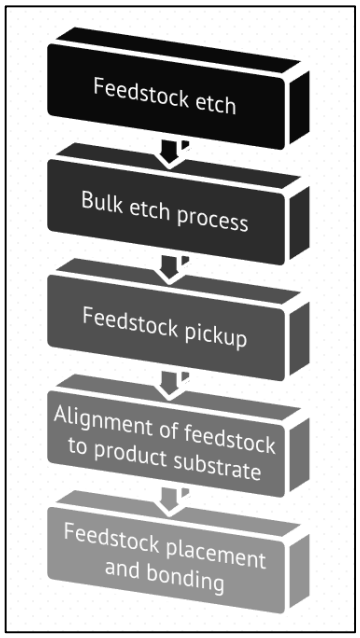


Figure 5: Feedstock fabrication overview.

Figure 5 shows the feedstock fabrication process. The goal is to pick feedstocks available on Silicon-on-Insulator (SOI) wafers and place them to a target substrate with nanometer scale precision. The fabrication of devices on SOI wafers is well established and the buried oxide (BOx) layer allows a way to selectively separate

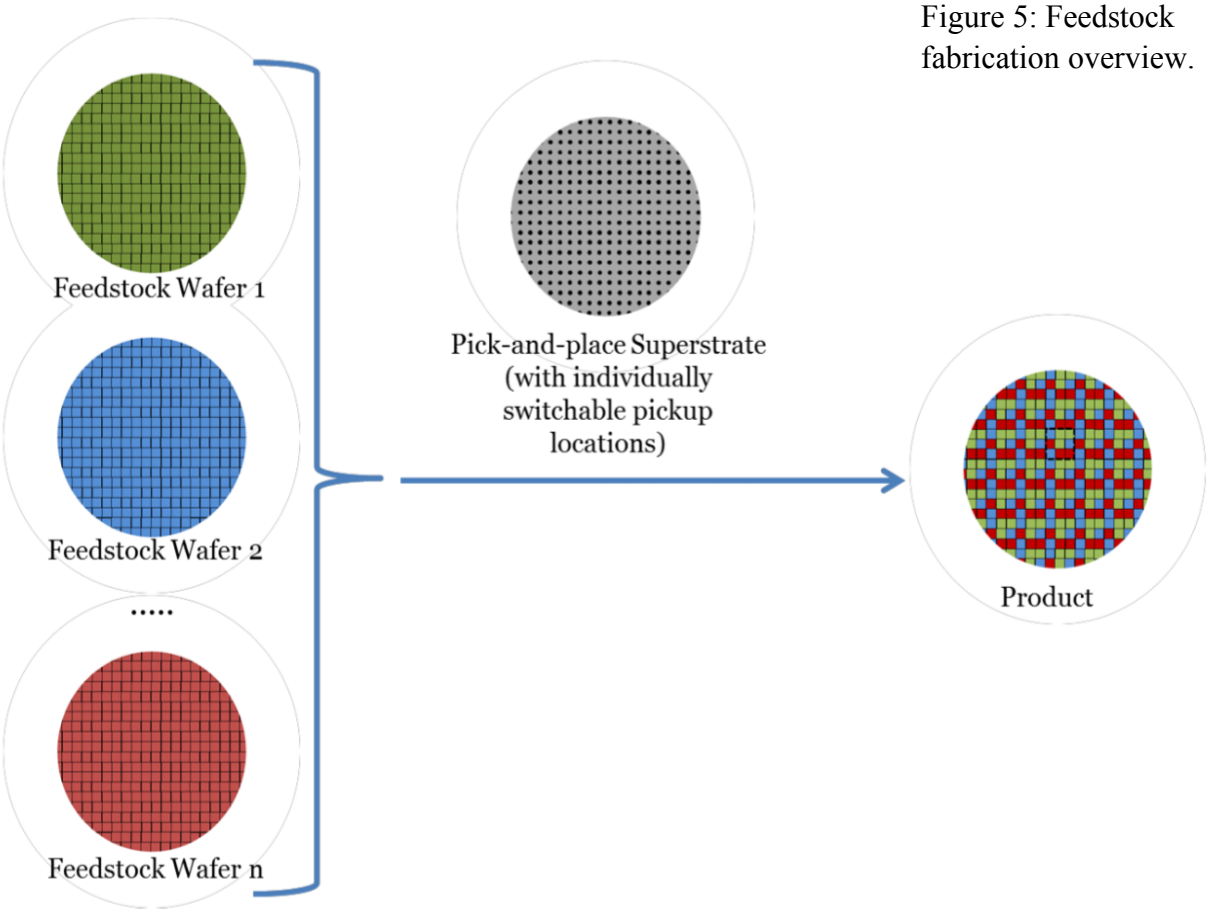


Figure 6: A schematic of the feedstock assembly process wherein the feedstock can include logic and memory elements (macros) [8].

feedstocks from specific locations. A generally applicable approach is outlined in Figure 6. Many different types of feedstock components such as transistors, optical devices and MEMS devices could be used, each available on a separate wafer. The assembly process leverages sub-5nm scale alignment metrology of moiré based schemes [14] to achieve nanometer-precise assembly of disparate microscale feedstocks onto a product substrate.

1.5 M2A2 EDA Design Overview

Electronic Design Automation (EDA) is a combination of software tools required to design integrated circuits (ICs) and boards [15]. All these tools are used in a specific order which makes an EDA design flow. The typical backend ASIC flow is shown in Figure 7.

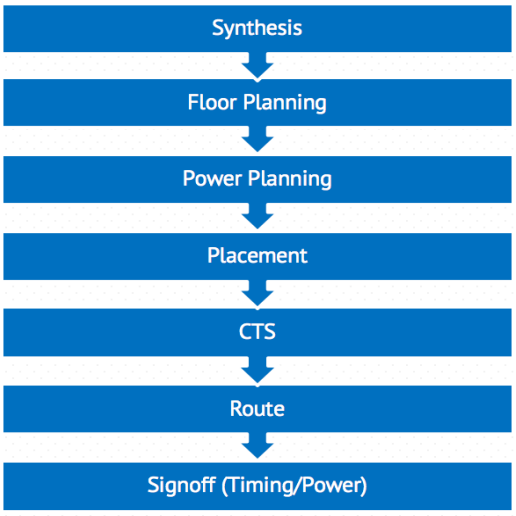


Figure 7: ASIC EDA Design Flow.

It includes synthesizing design which maps the design to standard cell library to meet the functionality of design at optimal performance and power. Next, the physical design of SoC is performed, which includes floor planning (preparing layout of design/chip), power planning (laying out power rails and straps), placement (placing physically the standard cells on the design layout), clock tree synthesis (CTS – routing and building the clock tree) and routing (routing metal interconnects to connect placed cells in design) [16]. Once the design is implemented, timing and power analysis is performed in the design signoff stage. In case of violations at any stage, the flow is run again from the required stage in order to fix it. This change of order is commonly known as engineering change order (ECO). Typically, multiple iterations of this flow are executed to meet

the design specifications. Hence, the turnaround time of ASICs is high. In our work, we have also made use of conventional ASIC flow with few modifications. The M2A2 EDA flow is shown by Figure 8.

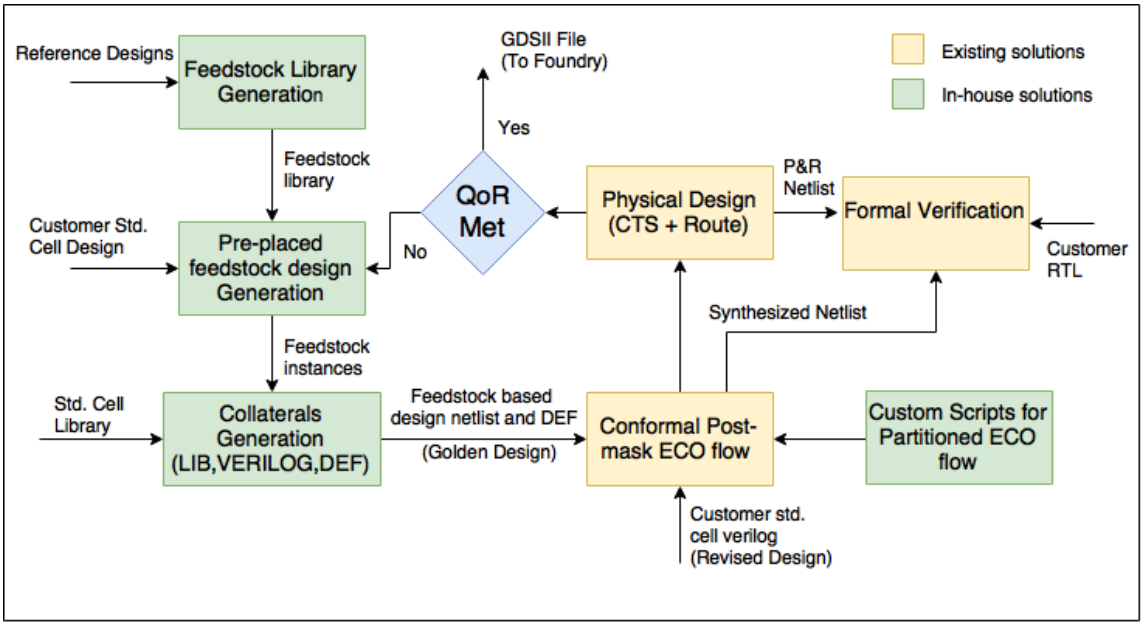


Figure 8: M2A2 EDA flow.

In our approach, we have made use of feedstock cell as the primitive block in our design, unlike ASIC where standard cell is the elementary unit. This is obviously due to the fact that M2A2 fabrication/assembly process is based on microscale pick and place of elementary units. The M2A2 EDA flow first involves feedstock library generation (FLG) based on reference designs, as shown in Figure 8. The M2A2 EDA design implementation steps for customer std. cell design involves the following:

- Feedstock pre-placed design generation (FP2DG)
- Design collaterals generation (DCG)
- Post mask ECO synthesis
- Physical design – CTS and Routing
- Timing/Power Signoff and Design Verification

In the feedstock library generation (FLG) process, the feedstock cells, sized in the range of 50μm *50μm to 100μm *100μm are designed based on standard cell library, which form the basic building blocks of M2A2 circuits. The library generation process is based on learning and heuristic techniques, generated in a way to optimize design performance of reference designs. The feedstock

consists of sea of pre-placed standard cells with no connections among the cells. In other words, the input and output pins of standard cells are floating in a feedstock. The metal 1 (M1) power rail is routed in a feedstock to power the standard cells. Figure 9 shows a $50\mu m$ *$50.86\mu m$ feedstock cell which contains ~800 standard cells. The feedstock has been implemented in this work, using 32nm technology node.

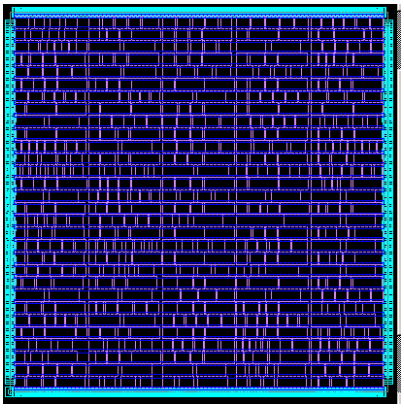


Figure 9: Typical feedstock cell.

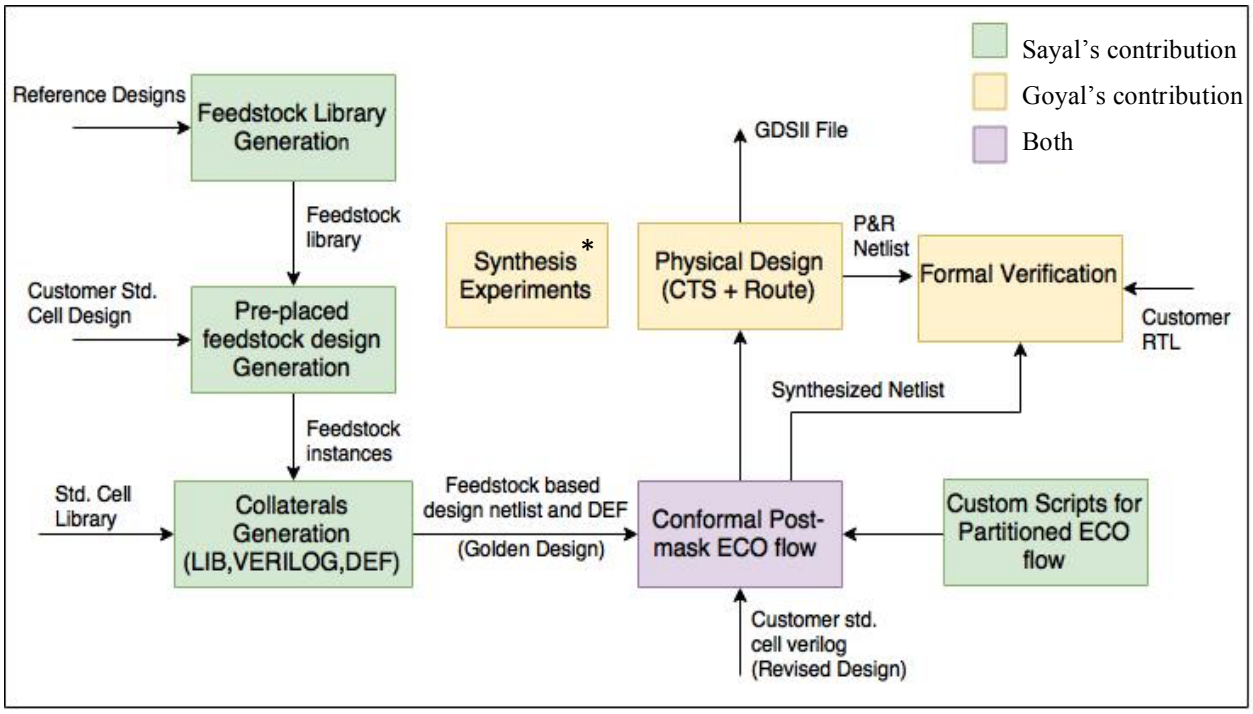
In the feedstock pre-placed design generation (FP2DG) process, the feedstock cells are pre-placed in design in a way to meet the functionality of the design which is implemented in standard cell based ASIC approach. The FP2DG algorithm is implemented based on heuristic optimization techniques and aims to place the desired feedstock cells from feedstock library at specific locations in the design. The objective is to replicate the standard cell based design behavior by meeting the functionality of design, and optimizing performance, area and power consumption.

Once the feedstock pre-placed design based on feedstock library is generated, the ASCII representation of the design and library information is generated in design collaterals generation (DCG) process. The pre-placed design netlist (information on feedstock cell instances in design) and Design Exchange Format (DEF - which captures the physical location of feedstock cells placed in design) files are generated. In the DEF generation process, floor planning, power planning, and placement of standard cells and macro cells is performed. The design liberty files (.lib) are also generated for each feedstock cell based on standard cell library. All these design collaterals are required to run synthesis and physical design flows [17].

Once the feedstock based design collaterals are generated, post mask functional ECO synthesis [18] is performed using Cadence Conformal ECO tool. The design which needs to be implemented using M2A2 technology is fed to the tool as the revised design and feedstock pre-placed design collaterals are fed as the golden design. The post mask ECO flow meets the functionality and constraints of revised design using golden design resources. In other words, it connects the cells placed in the golden design to meet the functionality and design specifications of the revised design. In this way, ASIC standard cell based revised design is implemented using M2A2 feedstock based library.

The synthesis process is followed by conventional ECO physical design processes – clock tree synthesis and routing. The formal verification flow [19] is run to ensure that M2A2 feedstock based implemented design meets the design functionality specifications. In case the design performance specifications such as timing and power are not met, the complete process from FP2DG is repeated to optimize the performance and meet the design constraints.

The complete EDA work has been done in collaboration with one more master's student, Vipul Goyal who is pursuing his M.S.E. in Electrical and Computer Engineering department at The University of Texas at Austin. The EDA work distribution is shown in Figure 10.



**Goyal has also worked on synthesis. He has carried out experiments but has not succeeded in getting working solutions.*

Figure 10: EDA Work Distribution.

This thesis document presents the author's contribution in this project. Chapter 2 deals with the feedstock library generation (FLG) process. Chapter 3 focuses on feedstock pre-placed design generation process (FP2DG). The design collaterals generation flow is described in chapter 4. The proposed synthesis flow methodology making use of post mask ECO approach is discussed in chapter 5. The synthesis results and M2A2 performance comparison with ASIC and FPGA is evaluated in chapter 6. The conclusion and scope of future work is presented in chapter 7.

1.6 Specifications of implemented ASIC designs

The dual amber core processor [20] is taken as the reference design to compare the ASIC standard cell and proposed M2A2 based implementation quality of results (QoR). The power, area and performance is compared for both design implementations. The architecture of dual amber core processor [20] is shown in Figure 11.

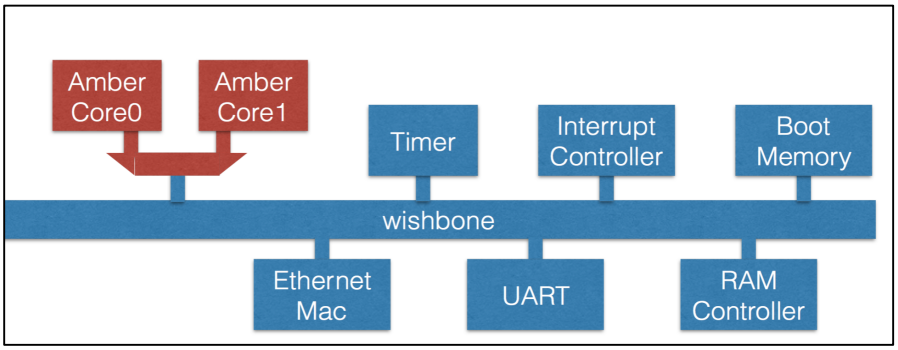


Figure 11: ASIC Design Architecture [20].

The standard cell ASIC design implementation is based on 32nm technology node, i.e. the design uses ARM 32nm standard cell library. The standard cell ASIC implementation is performed on 3 different design corners – high performance, balanced performance, and low power. The specifications of all these corners is shown in Table 2. The standard cell ASIC physical layout for all the 3 corners is shown in Figure 12.

Specifications	High Performance	Balanced Performance	Low Power
Cycle Time (ns)	1.25	2.80	5.00
Total Power (mW)	280	50	25
Die Area (mm^2)	1.05	1.00	0.95
Utilization (%)	65	75	85

Table 2: ASIC Design Specifications.

The standard cell ASIC implementation can be performed using both Synopsys and Cadence synthesis and place and route (P&R) tools. However, the M2A2 design is implemented using Cadence Conformal ECO (synthesis), Cadence RTL Compiler (synthesis), Cadence Encounter/Innovus (P&R), and Cadence Tempus (timing analysis). The comparison of different

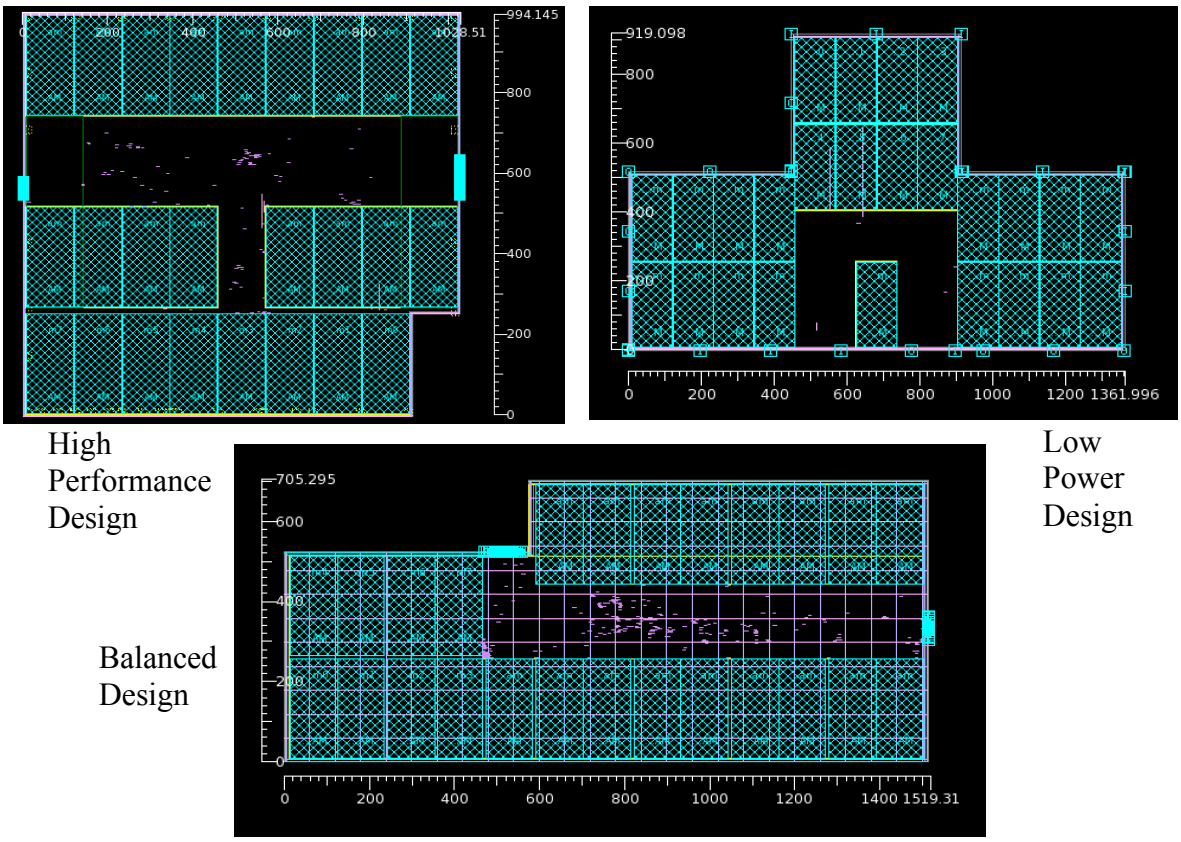


Figure 12: ASIC Design Floorplans.

performance metrics – speed, area, power, etc. is analyzed for both standard cell ASIC and M2A2 implementations. The literature survey on FPGA designs with similar design specifications is performed to obtain performance and cost comparison metrics of M2A2 with the FPGA design.

This document is organized in following chapters. Chapter 2 presents the feedstock library generation (FLG) process. Chapter 3 deals with feedstock pre-placed design generation process (FP2DG). The design collaterals generation flow is discussed in chapter 4. The proposed synthesis flow methodology making use of post mask ECO approach is described in chapter 5. The synthesis results and M2A2 performance comparison with ASIC and FPGA is evaluated in chapter 6. Finally, the conclusion and scope of future work is discussed in chapter 7.

Chapter 2: Generation of Feedstock Library

This chapter deals with the feedstock library generation (FLG) process. The problem statement and algorithm design objectives are presented in section 2.1. The FLG design methodology is illustrated in section 2.2. The literature survey and overview of studied and implemented algorithms is discussed in section 2.3. The design flow of the implemented FLG technique is described in brief in section 2.4. The algorithm is explained in detail with all the optimization techniques in section 2.5. The control knobs and various tradeoffs considered while executing the algorithm are mentioned in section 2.6. The key metrics for analysis quality of outputs and results generated by the algorithm utility are presented in section 2.7. The deliverables, i.e. EDA software developed along with the input dependencies, usage model and output files generated are listed down in section 2.8.

2.1 Problem Statement and Objectives

The feedstock library generation process deals with the designing of feedstock cells which serve as the basic building blocks in M2A2 design. The feedstock cells can be sized in the range of 50μm*50μm to 100μm*100μm, and are designed based on the standard cell library. The FLG algorithm is based on the learning and heuristic techniques, developed in a way to optimize and meet reference design functional requirements and specifications. The feedstock consists of a sea of pre-placed standard cells with no connections among the cells. In other words, the input and output pins of the standard cells are floating. The metal 1 power rail is routed in a feedstock to power the standard cells placed inside a feedstock. It is desired to have limited number of feedstock cells in the library, the lesser the better to reduce the manufacturing and assembly cost. Thus, the FLG problem deals with designing a finite (~10) number of optimum feedstock cells (sized ~50um*50um to 100um*100um) which contains the optimal combination (count, type and placement) of standard cells. These feedstock cells when repeatedly placed in design in a specific order along with the memory elements (macros) and hard IP blocks meet the ASIC design functionality and optimize the design performance.

The main objective in the feedstock library generation process is to design generic feedstocks which can meet the functionality and performance requirements for different designs. This eventually leads to lesser number of feedstock cells in the library, which reduces the manufacturing and assembly cost. At the same time, there should be sufficient number of feedstock

cells to synthesize different set of designs and meet the design functionality and performance constraints such as timing, power and area specifications. The type, count and location for all the standard cells placed inside each feedstock needs to be determined accurately and intelligently in order to synthesize the design with optimal performance. The standard cell utilization should be neither too low which can lead to the wastage of silicon die area, nor too high which can lead to congestion and other design related issues. The accurate placement of standard cells inside each feedstock is of utmost importance, since the interconnect delay depends on it. The locations of these standard cells need to be calculated based on some heuristic techniques. The random placement of cells might lead to very high net length, which can lead to high interconnect/wire delay. In order to provide the optimization knobs and meet other design objectives, spare cells in the form of buffers, tie cells for connecting the pins of spare cells to GND/VDD, and other physical only cells must be placed in a feedstock. Thus, it becomes a very challenging problem to come up with an optimal and limited number of feedstock cells which can optimize the design performance at minimal NRE cost.

2.2 FLG Methodology

The feedstock library generation methodology is discussed in this section. The feedstock library generation is a 2 step process, as shown in Figure 13. In step 1, the design data is prepared. The physical design of reference designs can be performed either in Synopsys IC Compiler tool or in Cadence Encounter/Innovus tool. In order to extract the physical and functional information for these reference designs, a Tcl utility is developed which extracts the standard cell reference and instance names, locations, load cell instance names, pin names, etc. The information obtained from step 1 is fed to the FLG algorithm utility developed in Perl. In step 2, the FLG algorithm utility is executed to generate the feedstock summary, instance and placement reports. The algorithm is implemented based on heuristic and learning analytical design techniques, which are discussed in detail in section 2.5. There is also a feedback loop from

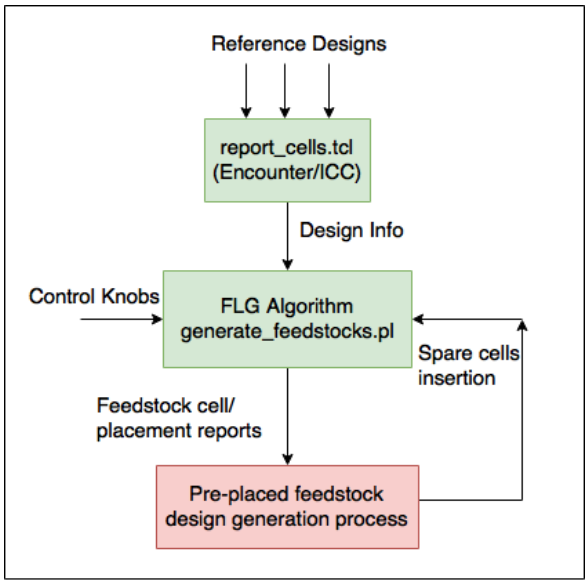


Figure 13: FLG Methodology.

feedstock pre-placed design generation process (FP2DG) to improve the feedstock design. In case, any standard cell needs to be added to a specific feedstock, to meet the design functionality or optimize design performance, there is an option provided to the user in FLG flow. There are various control knobs provided to user to generate a feedstock library as per requirements. The controllability options are described in detail in section 2.6.

2.3 Algorithms Survey

In this section, all the algorithms which were reviewed and implemented to generate generic feedstock cells are discussed. In simple words, the FLG problem can be stated as follows. Assume that there exists a large number of multi-colored points. These points are arranged in a specific order in space. The problem deals with designing a limited number of optimal combinations of these points such that when these combinations are repeatedly placed in space, replicates the original set of points. For one of our ASIC test designs, there exist ~50,000 points of 300 different colors lying in some specific order in space. It is required to make 3 or 5 combination of points which when placed in certain order can cover all the points in original design. It is desired that combinations should occupy a minimum area, and minimum deviation in locations for each point. Thus, this problem deals with significant reduction in variables of equation, i.e. from 300 points to 3 or 5 groups.

The author has reviewed various existing algorithms addressing similar problems. This problem is similar to graph isomorphism problem, which is a NP-complete problem [21]. This problem is further constrained as it involves not only standard cell type matching, but placement of cells is also considered. The objective is to come up with a minimal number of optimal combinations. The author has also reviewed algorithms related to longest common sub-sequence and string matching [22,23]. All these algorithms are restrictive cannot be used to find the limited number of optimal combinations. The author has also reviewed the clustering and other mapping algorithms [24,25].

Based on the literature survey and own ideas, the author has implemented the following algorithms. Some of the algorithms worked well, and are included in the final implementation technique. However, few of them have some limitations and are discarded in final technique. The merits and demerits of each approach are also discussed with each approach.

2.3.1 Greedy clustering

In this approach, the design was divided into different segments of specific size, named as partitions. The number of cells in each partition was calculated. The density of all the partitions was evaluated based on the cell count and area of cells and the partition. The grouping of partitions with similar density values was then performed to form cluster of partitions with similar density values. Each cluster is assigned a unique group name. Then, all the partitions of the same group were compared to form a feedstock, one each for a group. It involved finding the maximum number common cells for each partition, such that it satisfies congestion constraints. This approach was easy to implement and understand. However, there were drawbacks associated with this approach. The grouping of partitions based on density was not an optimal way to generate feedstocks, since the problem was not to find and group exact or same partitions, but to group partitions which had the maximum number of cells in common. It was quite possible that a dense and sparse partition had more cells in common than two sparse or two dense partitions. This technique resulted in the minimal number of cells common for each group. Thus, this technique was not implemented in the final FLG algorithm.

2.3.2 Efficient clustering

This approach was similar to greedy clustering approach, with few modifications. In this approach, once the groups of different partitions were formed based on cell density values, each group was assigned multiple feedstocks, unlike greedy approach where only a single feedstock cell was assigned to each group. The multiple feedstocks were assigned based on a minimum count of common cells across partitions getting mapped to each feedstock. This approach was able to address the issue of low count of common cells across partitions getting mapped to each feedstock However, it resulted in an increase in the number of feedstock cells. The increase in the number of feedstock cells resulted in an increase in the NRE cost, and was not a practical solution. This technique was making use of ~40 feedstocks to implement a high performance design. This approach was also relying on density based clustering, which was not an optimal solution since similar cell density across partitions was not an indicator of common number of cells across them. Because of all these limitations, this approach was also not implemented in final design.

2.3.3 Logic Aware mapping

The overall objective is to synthesize the standard cell based ASIC design using M2A2 feedstock cells. It is possible to have multiple gate level ways to implement the same logic function by performing logic restructuring. For example, an operation F = ~(A.B) can be implemented using either NAND gate or a combination of AND gate and inverter in a design. Thus, the problem of synthesis does not require the existence of all the same standard cells at same locations in feedstock pre-placed design. In this technique, the algorithm was designed to find the logic cones for all the paths that exist in each partition. A logic cone comprises of standard cells which are connected with each other to implement a particular function. The technique made use of a depth first search (DFS) [26] and breadth first search (BFS) [27] algorithms to find the logic cones. Then, the logic cones implementation function was converted to the sum of product (SOP) terms. The SOP expressions for all the partitions were compared to find the similar partitions. However, this approach also did not give good results. The functions across partitions have low similarity, which resulted in a small number of common cells. In order to improve the minimum count of common cells, number of required feedstock cells got increased. The main reason that this approach did not work well was that the problem was further constrained by adding connectivity information. The objective was to generate a feedstock cell which consists of sea of standard cells, with no connections among them. In this algorithm, we were considering the connections among the standard cells as well, assuming that all only the cells which were connected to each other would be placed in the same partition. However, this assumption was incorrect as logic paths are design dependent and can spread across different partitions. Furthermore, there can be multiple paths in one partition, and the cells on these paths might be placed physically close to each other. However, this information across different paths was not captured while making the comparison, since logic cone was evaluated for each path separately. Thus, this technique was also not implemented in final FLG algorithm.

2.3.4 Similarity Driven mapping

This mapping technique does not depend on density or logic of cells inside partitions. In this approach, the design was segmented into multiple partitions. The mapping of partitions in this approach did not consider density and logic constraints. The similarity driven mapping made groups of those partitions which had common cells count greater than a minimum threshold

percentage. The similarity of cells was evaluated in different ways as per the user requirements to optimize the design performance. The most constrained similarity mapping was the one where the cells considered for mapping across different partitions had drive strengths as well as VT classes associated with them. In other words, it was not just the functionality and input output pins of the cells which make them similar, but also the drive strength and VT class. However, this mapping resulted in a low similarity or high number of feedstocks. To resolve this problem, similarity of cells was made on the basis of standard cell functionality and input/output ports. In other words, all different flavors of AND2 gate such as AND2X1_LVT, AND2X4_HVT were considered the same for mapping. The number of variables were reduced to map the cells of same functionality more effectively in design. Since, this technique did not rely on any false assumptions, it was implemented in the final FLG algorithm. However, one of the drawbacks of this technique was that it was resulting in a low mapping coverage. The mapping coverage was the percentage of cells which are taken into similarity consideration while forming feedstocks. Since this technique was dealing with a mapping of cells for static partitions, i.e. mapping process occurred only once and the partitions which had low similarity value were ignored. In order to overcome this problem, dynamic partitions technique was also implemented in final FLG algorithm.

2.3.5 Dynamic Partitions Approach

The dynamic partitions generation algorithm complemented the similarity driven mapping. In this algorithm, new partitions were generated dynamically after each iteration of similarity driven mapping. In order to maximize mapping coverage, maximum number of partitions were required to participate in mapping process. In this approach, the unmapped partitions after similarity driven mapping were regenerated. To regenerate these partitions, the partition was slid in the horizontal direction by few units. This was done to create the partitions with new combination of cells, and still retaining the original unmapped cells. However, this led to the un-scanned regions in design. The cells lying in the unscanned regions were never considered for similarity mapping, and thus coverage can never be made 100%. But this approach significantly improved the coverage in comparison to the similarity driven mapping, and was implemented along with the similarity mapping technique in the final FLG algorithm.

2.3.6 Congestion Aware Placement

This algorithm dealt with the selection and placement of cells inside feedstock. The

similarity of feedstocks were improved in such a way that it did not violate congestion i.e. maximum density constraints. Once the cells were chosen, the placement of cells was performed deterministically based on location of cells in the mapped partitions. However, the congestion constraints had a higher precedence, and cells were placed in such a way that maximum congestion constraints were not violated. Based on the density of cells in each feedstock, the spacing between cells was calculated by considering the congestion factor into an account. This was useful from the scenario that the spreading of cells in feedstock would lead to less routing issues while performing the physical design for M2A2 design. This algorithm was also implemented in final FLG algorithm.

2.3.7 Spare Cells Insertion

The techniques discussed so far optimized the feedstock generation process by employing intelligent techniques based on the design data. However, it is good to provide a control to the user to improve the feedstock design based on experience and other processes information. The optimal design of a feedstock is also dependent on its relative placement with other feedstocks placed in design to implement the standard cell based ASIC design. Hence, there was a need of some feedstock mechanism, to insert some specific cells, or buffers and physical only cells. This was achieved by adding additional spare cells to the design, as listed by user in one of the configuration files. This technique was also implemented in final FLG algorithm, in order to further improve the feedstock configurations generated based on similarity driven
mapping and dynamic partitions approach.

2.4 Overview of implemented technique

In this section, a high level overview of the implemented FLG algorithm techniques is presented. The details for each step in the FLG process is discussed in section 2.5. This algorithm consists of 9 major steps, as shown in Figure 14.

In step 1, the reference designs information was processed and data structures required for downstream processes were generated. The reference designs were then scanned, and divided into partitions. These partitions were analyzed, and partitions which contained similar kind of cells were grouped together. The basis of similarity and grouping is discussed in detail in section 2.5.

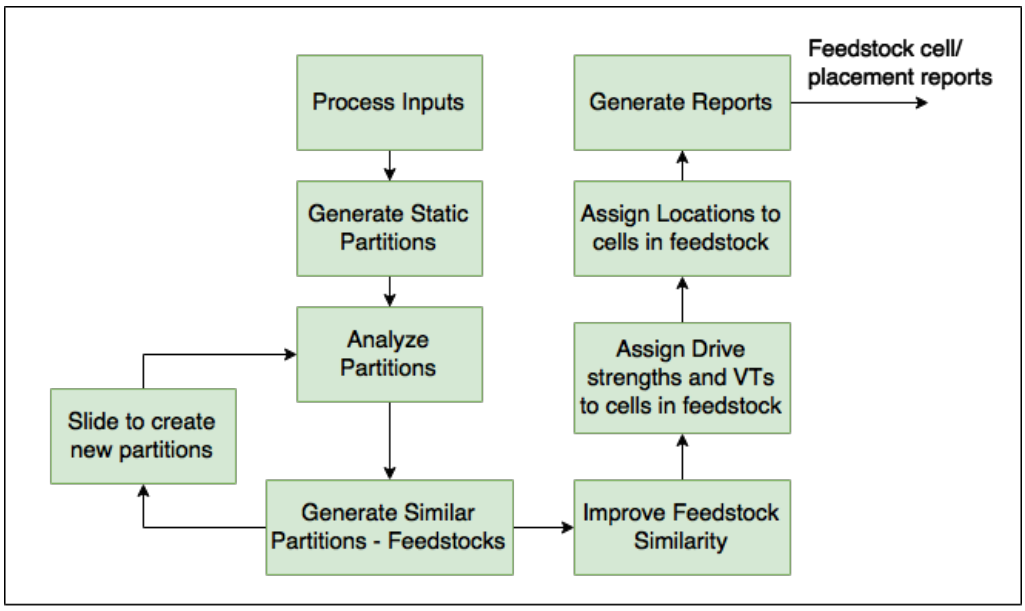


Figure 14: FLG Algorithm Overview.

The partitions which were not similar and were not grouped together, were again analyzed. The un-scanned regions were partitioned again, but with new location coordinates. This was done by shifting the original locations of partitions by some finite distance in horizontal direction. In this way, the new partitions formed contain new combination of cells. These new partitions were again analyzed and similarity driven grouping was performed to either map the partitions to existing groups, or to generate new groups if required. This process was repeated until all the partitions were scanned and covered for grouping analysis. Then, the groups containing all the similar partitions were analyzed to obtain the reference partition, which we call feedstock cell. The similarity of each feedstock was improved by adding cells which were repeated in most of the mapped partitions, and removing cells which would not affect the similarity to a large extent. The cells in the optimized feedstock cells were then assigned drive strengths and threshold voltage (VT). The drive strength and VT assignment was done based on the distribution of drive strengths and VT of cells in the mapped partitions. At this stage, the feedstock configurations and standard cells type and count for each feedstock was finalized. The only task left at this stage was to assign location to all the standard cells in the feedstock. This was done in the next step, where deterministic placement based on cells distribution in the reference designs was performed to assign locations to most of the standard cells in feedstock accurately. The placement was done in such a way that there was no legalization issue, and maximum number of cells were placed based on cells physical location in original design. In the final step, output report files for analysis and

collaterals generation process were generated. The details of all these steps are explained in detail in the next section.

2.5 FLG Algorithm details

In this section, the implementation details of FLG algorithm including all the optimization techniques and tricks is dicussed. The detailed explanation on all the 9 steps of the FLG algorithm is provided in the following subsections.

2.5.1 Inputs Processing

The reference designs information extracted in step 1 of FLG process is fed to the algorithm utility. The design information consists of locations, pins and reference names for all the standard cells used in the reference designs. This information is processed and data structures are made which are used in downstream steps of the FLG process. All the sanity checks in terms of file existence and required collaterals existence are performed in this stage. In case, the user is specifying spare cells, the required data structures are also generated in this step.

2.5.2 Static Partitions Generation

This step deals with the segmentation of a design. Once all the cells information is processed, the whole design is segmented into multiple small regions, named as partitions. The size of each partition is same as that of a feedstock, defined by user. In this step, the whole design is scanned first. The location for cells which lie on all the 4 edges of the design i.e. top, bottom, left and right is taken, and the complete region formed by the bounding box covering the edge cells is segmented into partitions of size equivalent to that of a feedstock.

2.5.3 Partitions Analysis

In this step, all the cells are assigned to the static partitions generated in the previous step. Based on the physical location of cells, each cell is assigned to the partition in which it lies. The relative locations of all the cells lying inside each partition is also evaluated in this step. This information is used while deterministically placing the cells in a feedstock. In the process of assigning cells to each partition, two types of assignment is done. The first assignment is the assignment of the standard cells which include functionality, number of input and output ports,

drive strengths and VT class. In the second type of assignment, only functionality and input output ports is considered. For example in first type of assignment, AND2X1_HVT, AND2X2_HVT, AND2X1_RVT are considered as separate cells since they have either different drive strengths X1, X2 or different VT classes HVT, RVT. In the second type of assignment, all these cells are considered the same since they all belong to AND2 category. The second type of assignment is mainly implemented in FLG process, since the similarity for this type of assignment is higher in comparison to the first one.

2.5.4 Grouping of Similar Partitions to generate Feedstocks

In this step, the partitions generated in previous step are grouped to form feedstocks. After analyzing the partitions, the similarity index of each partition with respect to each other is evaluated. First, the reference partition is assigned for each group. The cell count of all the partitions, evaluated in previous step are compared and the partition with maximum cell count is assigned as a reference partition. Now, all the partitions are compared with this reference partition and similarity index of each partition with this reference partition is evaluated. The similarity index of partition is ratio of common cells found in both the partition and the reference partition to the total number of cells in the partition. If the number of common cells across partition and reference partition exceeds the similarity threshold value which is specified by user, the partition would be grouped with the reference partition. This process is repeated for all partitions, and the partitions which exceed the threshold value are grouped together. The reference partition is assigned as a feedstock, if the number of similar partitions in a group exceeds the minimum threshold of mapped partitions. The threshold value is calculated dynamically based on the maximum number of feedstocks which can exist in the design library, number of existing feedstocks, and unmapped partitions in the design till the previous iteration. The iteration process will be discussed in the next section. The reference partitions, which are assigned as feedstock cells will be optimized further, and would be explained in later steps. After this step, some of the partitions are grouped and some of them are left ungrouped. In order to generate a high quality feedstock library, all the cells/partitions in the reference designs should be considered in the similarity process to generate feedstocks. We term this metric as the mapping coverage, which is percentage of total cells in reference designs which participate in mapping and similarity grouping process. Higher the value of similarity and mapping coverage, better is the quality of feedstock library being generated. In

order to improve the mapping coverage, there is a need to devise some mechanism which can consider ungrouped partitions after this. This process is explained in the next subsection.

2.5.5 Dynamic Partitions Generation

In order to improve the mapping coverage, there is a need to consider ungrouped partitions again for the similarity mapping. The unmapped partitions are generated again, which some delta shift in horizontal direction. The new locations of these partitions are fed to the step 3 of FLG algorithm which is "Analyze Partitions". The cells placed in these regions are assigned to the new partitions. The steps 3 and 4 are performed again, for these partitions, to form a new feedstock based on the grouping of the partitions. In this iterative manner, steps 3, 4, 5 are performed until all the partitions in a design are considered for grouping with some reference partition. It might be possible that grouping of partitions does not lead to a feedstock formation, since the count of partitions grouped is less than feedstock formation threshold as explained earlier. But, the iterative process ensures that the coverage is maximized, while maintaining the high similarity value and limited number of feedstock cells in the library.

2.5.6 Feedstock Similarity Optimization

Once all the partitions are considered for grouping, and reference partitions are assigned as feedstocks, the similarity for each reference partition which is assigned as a feedstock is improved. In this step, the reference partition is compared with its respective mapped partitions. The cell count for each type of cell in reference partition is calculated. The improvement and penalty costs for each type of cell is evaluated then. The improvement cost is the average improvement in similarity value if the cell count of a particular cell in the reference partition is increased by 1. Similarly, the penalty cost is the average degradation in similarity value if the cell count of particular type of cell in the reference partition is decreased by 1. The improvement and penalty costs for all types of cell in the reference partition are sorted in decreasing and increasing order respectively. The cell with the maximum improvement value is added to the reference partition, and cell with the minimum penalty value is removed, to maintain the standard cell utilization. This process is repeated until there is a significant improvement in the similarity value for each feedstock cell. At the end of this step, the count and type of standard cells in each feedstock is frozen. Next, the drive strengths and VT to all the cells in the feedstock are assigned.

2.5.7 Drive strength and VT Assignment

In order to improve the mapping similarity, the drive strength and VT class attributes were removed from cell in step 3 of FLG algorithm. This was done to reduce the number of variables in a similarity mapping equation. Once the cells are frozen in the final feedstocks, the drive strengths and VT are assigned to each cell. This assignment is done based on the distribution of standard cells in the mapped partitions of each feedstock. For each feedstock, the cell count for each type of cell is calculated. The mapped partitions which have the cell count greater than threshold value are taken into consideration. The threshold value is determined by multiplying the user defined critical cells factor with cell count in each feedstock. In case the mapped partitions have a cell count greater than this value, the library cells with this functionality are added to the target library. For each cell, target library is evaluated. The drive strengths and VT is assigned based on distribution of VT and drive strengths in the target library of each cell. This assignment is based on a probabilistic model, i.e. the drive strengths and VT of cells present in the reference design govern the distribution of library cells in the feedstock.

2.5.8 Legalized and Deterministic Cells Placement in Feedstock

This step deals with the placement of the standard cells in each feedstock. Once the standard cells are generated based on the distribution of VT and drive strengths in design, the location to all the cells is assigned in each feedstock. This step is similar to the drive strength and VT assignment step. The locations of cells in the mapped partitions are considered to find the probable location of cells in the feedstock. The mapped partitions in which the standard cell count exceeds the exceeds the threshold value are considered for deriving the cells locations. The threshold value is determined by multiplying the cell count in feedstock with the user defined threshold factor. The relative locations of all the cells in the mapped partitions are evaluated. Then, the critical cells of each feedstock are determined. The locations to all these critical cells are assigned deterministically. All the cells in a feedstock which lie within top critical cell percentage, specified by the user are assigned as critical cells. The location database is maintained for all the critical cells in a feedstock. The other non-critical cells are assigned randomly, in order to ensure the legalized placement of cells.

In order to ensure the legalized placement for all the cells, the landing zones are defined in the feedstocks. The feedstock cell comprises of finite number of standard cell rows; each row width

is same as that of a feedstock. The rows in a feedstock are segmented into multiple zones, which are called “landing zones”. The landing zone height is same as that of a standard cell row and landing zone width is feedstock width/number of zones in a single row. The relative locations of cells in the mapped partitions are analyzed, and probabilistic distribution is done to assign location of each standard cell in feedstock to a specific landing zone. The remaining width value is maintained for each landing zone, and standard cell is added to the landing zone, only if there is sufficient space to place it. This not only ensures legalized placement, but also the deterministic placement since the landing zones are localized regions in a feedstock. The congestion factor, which is defined by the user also controls the placement of cells in the landing zones. The critical cells are first assigned to landing zones. Based on the value of congestion factor, the location of cells in each landing zone is evaluated. This assigns the location to all the cells in a feedstock. In case there is any legalization error due to high standard cell utilization in specific zone, issue is reported. This can be fixed by changing the congestion and deterministic placement control knobs.

2.5.9 Reports Generation

This is the final step in the FLG algorithm process. This step generates all the report files required for the analysis, and in FP2DG and DCG processes which are discussed in chapter 3 and chapter 4 respectively. The summary file captures the FLG metrics (discussed in section 2.7) to analyze the quality of a feedstock library generated. The feedstock cell reports are also generated, which lists down the type, count and location of standard cells placed inside each feedstock. The output files are explained in detail in section 2.8.2.1.

2.6 Control Knobs and Tradeoffs

In this section, the controllability of FLG algorithm perl utility is discussed along with the consideration of design tradeoffs. In order to develop a generic solution which can be used for different corner cases, multiple customized variables are provided to user. Following are the main control knobs provided to user while executing the algorithm utility.

2.6.1 Feedstock Size

The feedstock size, i.e. width and height of a feedstock is set by the user. The feedstock size needs to be provided by user while executing the perl utility in <WIDTH>_<HEIGHT>

format. The unit of width and height dimension is micrometer (μm). The feedstock library gets generated based on this feedstock size.

2.6.2 VT and DS switch for mapping consideration

In the algorithm, there is an option whether to include drive strengths (DS) and threshold voltage (VT) of standard cells into consideration while analyzing the similarity. By default, only the functionality of standard cell is considered as a similarity criterion. It means that all the flavors of AND2 standard cell i.e. AND2X1_HVT, AND2X2_LVT, etc. are considered the same. In case, user wants to consider cells different based on drive strength or VT class values, a control knob is provided to the user.

2.6.3 Horizontal shift in iterative partitions formation

The algorithm iteratively scans the design to cover the maximum number of partitions in similarity analysis, in order to ensure high mapping coverage. In the iterative process, the new partitions are formed by scanning the design differently each time. The coordinates of unmapped partitions are updated for the next iteration in order to map it to existing or new feedstocks. The location of each partition is updated by sliding it in horizontal direction by a fixed value. This value is controlled by user. It allows a new combination of cells in the unmapped partitions in each iteration. The mapping coverage might be less if this variable value is high, since the area of un-scanned regions gets larger.

2.6.4 Threshold similarity

In the design scanning and mapped partitions formation process, the similarity between two partitions is defined as the count of common standard cells present in both the cells. In order to define if the two partitions are similar and can be mapped to the same feedstock or not, a threshold similarity percentage value is provided by the user. In case, the similarity percentage between two partitions is greater than or equal to the threshold similarity value, the partitions get are mapped together. The mapped partitions either get mapped to the existing feedstocks, generated in previous iterations or to a new feedstock. The partitions get mapped to a new feedstock if the partitions count exceed the threshold value set by user. This is done to ensure that generic feedstocks are getting generated which can be used at multiple locations in the design.

2.6.5 Threshold feedstock count

In order to ensure that feedstock library does not consist of infinite or large number of feedstock cells, the threshold count of feedstock cells is provided by the user. This variable internally calculates the minimum number of similar partitions required to generate a new feedstock cell in a library, in each iteration of mapped partitions. Thus, the similar mapped partitions threshold count is calculated dynamically to ensure that maximum mapping coverage is obtained with limited number of feedstock.

2.6.6 Similarity optimization level

Once the feedstock cells are generated, each feedstock is optimized by adding and removing specific standard cells, in such a way that similarity of feedstock with its mapped partitions is improved. This process is explained in detail in section 2.5.6. This process of optimization can be repeated for a finite number of times, and is controlled by the user.

2.6.7 Landing zone width

The location assignment process assigns the accurate locations to all the cells present inside each feedstock. The feedstock is divided into various segments, which are called landing zones. The height of each landing zone is same as that of a standard cell, whereas landing zone width is controlled by the user. In the deterministic placement of cells, each critical cell location is determined and assigned to a particular landing zone. The width of the landing zone should be set carefully. A higher value of width can lead to randomness or error in the placement of cells, whereas a smaller value might result in low standard cell utilization, since the legalized placement of cells might not be possible for a smaller width.

2.6.8 Deterministic cell placement threshold

There is an option to control the minimum percentage of cells which need to be placed deterministically. In case the threshold value is low, it increases the randomness in the placement of remaining cells. However, if the threshold value is high, it might lead to the legalization issues, since high number of cells might get assigned to the same landing zone. Thus, this value should be set carefully by analyzing the placement of cells in the reference design, and by executing and observing the results of FLG algorithm utility.

2.6.9 Congestion factor

The spacing between the cells placed in a feedstock is controlled by this variable. In case, no space is required between cells placed in a particular landing zone, this factor should be set to 0. Keeping this factor value to 1 results in even spaces among all the cells placed inside landing zones. This variable, thus controls the congestion and utilization of standard cells in landing zones.

2.6.10 Partitions cell placement participation criteria

In the location assignment process, the mapped partitions of a feedstock govern the final location of cells placed inside each feedstock. In order to participate in the decision making process, the location of cells in mapped partition is considered only if the mapped partition contains the minimum number of cells. This criterion is governed by the cell count, which ensures that particular cell is present significantly in the mapped partition so that its location should be considered to decide the location of cell placed inside a feedstock.

2.7 Key Metrics

In this section, the key metrics generated by algorithm for analysis are discussed. The metrics give an idea on how well the feedstock library is designed. Following subsections list down the main metrics generated by the flow.

2.7.1 Feedstock Average Repetition (FAR)

One of the main objectives of the feedstock library generation process is to design generic feedstock cells which can be used multiple number of times. The FAR metric indicates the number of times each feedstock cell is used in a design. It is generated for each feedstock cell. The overall average feedstock repetition is calculated by taking the average of FAR values for each feedstock. Higher the FAR value, better is the feedstock design.

2.7.2 Mapping Coverage

The feedstock library generation process involves the scanning of the reference design to form partitions which eventually form the feedstocks. Thus, it is very important that maximum number of cells in the reference design are considered while generating feedstocks. In the iterative

process of partitions generation, the shifting of partitions might lead to un-scanned regions in design. The cells present in these un-scanned regions are not taken into consideration during feedstock. This may eventually lead to low quality library generation. Thus, the mapping coverage provides the confidence in the quality of library being generated. The lower mapping coverage may lead to sub-optimal synthesis of the feedstock pre-placed design.

2.7.3 Feedstock Similarity

The feedstock is generated by combining the similar partitions in design. The optimal feedstock is then generated has a high similarity value with its mapped partitions. The similarity value is a measure of a feedstock design quality. A high similarity indicates that high number of cells are common between the mapped partitions and a feedstock. This ensures that the utilization of cells in feedstock pre-placed design would be higher while doing the synthesis of design.

2.7.4 Feedstock area utilization

The area utilization of a feedstock is the percentage of area occupied by the standard cells placed inside a feedstock to the total area of a feedstock. A high utilization means more number of standard cells are placed in a feedstock of same size. If the standard cell area utilization is too low, it leads to the wastage of silicon die area, whereas if it is too high, it might result in congestion and other routing issues. The area utilization for each feedstock is reported. This helps in determining the maximum count of buffer and other spare cells which can be added to the feedstock, and still meeting the congestion and maximum area utilization specification.

2.7.5 Deterministic cell placement for feedstock

The standard cells are placed in a feedstock based on a deterministic placement algorithm. The standard cell placement for mapped partitions is analyzed, and critical standard cells of each feedstock are placed at locations by analyzing the location of the standard cells placed in the mapped partitions of feedstock. The percentage of cells which got placed deterministically, and not randomly is given by this metric. This is an important parameter to know the placement quality of cells placed inside each feedstock. A high percentage value ensures that most of the cells are placed based on placement of cells in the reference design, whereas a low percentage means that higher percentage of cells are placed randomly in a feedstock.

2.8 Deliverables

In this section, the EDA software developed in the form of the perl and Tcl scripts in feedstock library generation process is discussed. The usage model, paths, inputs, outputs and other important information is also captured for the better understanding of the flow to the reader.

2.8.1 Design/Cell Report Generation

The process of a design report generation from the reference designs is described in this sub-section. The reference design can be implemented either in Cadence Innovus/Encounter tool or in Synopsys IC Compiler tool.

2.8.1.1 Script #1: report_cells_encounter.tcl

- ***Function:***

The purpose of this Tcl script is to extract the information from the standard cell based ASIC P&R implementation of the reference designs. It generates the data in a text file. The script is sourced in Cadence Encounter/Innovus shell, where the standard cell P&R design is implemented. The script generates the design cell report which contains standard cell instance and reference names, locations, input, output and inout pin names, load cell instance and reference names, etc. This output file is fed as an input to the FLG algorithm utility to generate feedstocks.

Usage:

source /home/projects/courses/spring_16/ee382m16745/feedstock/aseem_scripts/feedstock_generation/scripts/report_cells_encounter.tcl

- ***Inputs:***

1. The user needs to source this file in Encounter/Innovus session, after the design is loaded. The proper design cell view in Encounter ensures that correct data is getting extracted.

- ***Outputs***

1. cells_report.txt

The output cell report file containing the standard cell instance and reference names, locations, input, output and inout pin names, load cell instance and reference names, etc. is generated in the present working directory.

2.8.1.2 Script #2: report_cells_icc.tcl

- ***Function:***

The purpose of this Tcl script is to extract the information from the standard cell based ASIC P&R implementation of the reference designs. It generates the data in a text file. The script is sourced in Synopsys IC Compiler shell, where the standard cell P&R design is implemented. The script generates the design cell report which contains standard cell instance and reference names, locations, input, output and inout pin names, load cell instance and reference names, etc. This output file is fed as an input to the FLG algorithm utility to generate feedstocks.

Usage:

source /home/projects/courses/spring_16/ee382m16745/feedstock/aseem_scripts/feedstock_generation/scripts/report_cells_icc.tcl

- ***Inputs:***

1. The user needs to source this file in ICC session, after the design is loaded. The proper design cell view in ICC ensures that correct data is getting extracted.

- ***Outputs***

1. cells_report.txt

The output cell report file containing standard cell instance and reference names, locations,

input, output and inout pin names, load cell instance and reference names, etc. is generated in present working directory.

2.8.2 Feedstocks Generation

The working of FLG algorithm script is described in this sub-section.

2.8.2.1 Script #3: generate_feedstocks.pl

- ***Function:***

The purpose of this perl script is to generate the feedstock cells based on the reference designs information. The FLG algorithm is executed to generate generic feedstock cells. The script generates the summary file with key metrics, feedstock cell and placement reports which are used in netlist and DEF generation processes, discussed in detail in chapter 4.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/feedstock_generation/scripts/generate_feedstocks.pl

-file	*<Instance report file in standard format dumped from ICC shell>*	*(REQUIRED)*
-window_size	*<XLENGTH_YLENGTH> in ums (same as feedstock size)*	*(REQUIRED)*
-design_size	*<XLENGTH_YLENGTH of design>*	*(REQUIRED)*
-custom_cellfile	*<Custom cell file path to manually add cells to a specific feedstock>*	*(OPTIONAL)*
-include_ds	*(Include this switch if user wants to compare cells with drive strengths in process of making feedstocks; By default: disabled*	*(OPTIONAL)*
-include_vt	*(Include this switch if user wants to compare cells with VT classed in process of making feedstocks; By default: disabled*	*(OPTIONAL)*
-print_inst	*(To dump intermediate partitions reports)*	*(OPTIONAL)*
-debug	*(To print Debug messages in case of error)*	*(OPTIONAL)*

- ***Inputs:***

1. *-file <Design/Cell report>*

This file is generated in step 1 of FLG process by sourcing *report_cells_icc.tcl* in ICC

session or *report_cells_encounter.tcl* in Encounter/Innovus session. This file contains standard cell instance and reference names, locations, input, output and inout pin names, load cell instance and reference names for all the reference designs.

2. –window_size <WIDTH_HEIGHT>

The user needs to specify the size of the feedstock cell which needs to be generated. The design is scanned and segmented into various partitions, and then similar partitions are grouped to form feedstocks. The partition and feedstock cell size is defined by this variable. The unit of height and width dimension is micrometer (μm).

3. –design_size <WIDTH_HEIGHT>

This switch is used to provide the width and height of reference designs. The width and height values of all the reference designs are added to get this value. The approximate values of the reference design also work, as this is required to estimate the distribution of cells in the design.

4. –custom_cellfile <Custom cells report>

This switch is an optional argument. In case, the user wants to add the spare cells to any feedstock. This might be required based on the feedback from feedstock pre-placed design generation process, or buffers/tie-cells needs to be inserted in design. The user can specify the name and count for all the cells which needs to be added for each feedstock in this file.

5. –include_ds

This switch is an optional argument. In case, the user wants to consider the drive strengths of standard cells into consideration along with the functionality of cell in the similarity analysis of the partitions, this switch can be used. By default, it is disabled.

6. –include_vt

This switch is an optional argument. In case, the user wants to consider the threshold voltage (VT) of standard cells into consideration along with the functionality of cell in the similarity analysis of the partitions, this switch can be used. By default, it is disabled.

7. –print_inst

This switch is an optional argument. In case, the user wants to dump the intermediate output reports, this switch can be used. The flow generates partition reports for each design scan iteration.

8. –debug

This switch is an optional argument. In case, the user wants to print the help/debug messages from the script, this switch can be used.

- ***Outputs***

1. Feedstock summary report – feedstock_summary.rpt

The summary of all the key metrics for the feedstock library generation process, discussed in section 2.7 is listed in this file. The file gets generated in this directory: ${PWDIR}/REPORTS/ directory, where ${PWDIR} is the present working directory.

2. Feedstock cells summary report – feedstock_cells.rpt

The summary of all the standard cells along with their count in each feedstock and feedstock key metrics is mentioned in this file. The file gets generated in this directory: ${PWDIR}/REPORTS directory, where ${PWDIR} is the present working directory.

3. Feedstock output report – feedstock_output.rpt

This output file captures the information related to all the standard cells, along with their count, input, output and inout pins. The file gets generated in this directory: ${PWDIR}/ REPORTS/ directory, where ${PWDIR} is the present working directory.

4. Feedstock placement report – feedstock_placement.rpt

This output file lists down the physical information for all the standard cells placed inside each feedstock. This file captures the standard cell reference names along with their respective locations placed inside each feedstock. The file gets generated in this directory: ${PWDIR}/ REPORTS directory, where ${PWDIR} is the present working directory.

Chapter 3: Feedstock Pre-placed Design Generation

This chapter deals with the feedstock pre-placed design generation (FP2DG) process. The problem statement and algorithm design objectives are presented in section 3.1. The FP2DG design methodology is illustrated in section 3.2. The design flow of the implemented technique is discussed in brief in section 3.3. The algorithm is explained in detail with all the optimization techniques in section 3.4. The control knobs and various tradeoffs considered while developing the algorithm are described in section 3.5. The key metrics for analyzing the quality of outputs and results generated by the algorithm utility are discussed in section 3.6. The deliverables, i.e. the EDA software developed along with the input dependencies, usage model and output files generated are listed down in section 3.7.

3.1 Problem Statement and Objectives

The feedstock pre-placed design generation process deals with the generation of the feedstock pre-placed design which can be used to synthesize the standard cell based ASIC design. The requirement is to effectively map the standard cells placed in ASIC design to the standard cells placed inside the feedstock pre-placed design. It selectively chooses and places feedstocks and memory elements (macros) optimally in the original floorplan. The FP2DG algorithm is based on the learning and heuristic techniques, developed in a way to optimize and meet the reference design functionality requirements and performance constraints. Thus, the FP2DG problem deals with the mapping of a large number of standard cells placed at different locations in the design to a finite (~10) number of feedstock cells generated in the feedstock library. In order to quantify the complexity of the optimization problem, we can take an example of ASIC high performance design, which is discussed in section 1.6. The types of standard cells instantiated in the design are ~300, and the total number of standard cell instances is ~50000. Thus, the total number of ways to place the 300 types of cells at 50000 locations is 300^{50000}. The FP2DG algorithm aims to find the optimal configuration (1 of 300^{50000}) of feedstock cells placed in the design.

The main objective in the feedstock pre-placed design generation process is to generate a design with the optimal feedstock cells and memory macros placed at optimal locations in the design. The standard cells placed in the feedstock pre-placed design are used to synthesize the ASIC standard cell based design. Hence, the foremost objective of FP2DG process is to generate the feedstock based design which can meet the functionality requirements of ASIC design. In other

words, the functionality of standard cell based ASIC design can be met using feedstock based design. The other objectives of FP2DG algorithm are to minimize the silicon die area, meet the ASIC design performance/circuit speed in a congestion free design to meet the design constraints with a low TAT. An increase in the silicon die area increases the NRE cost. Hence, a lot of emphasis is put to optimize the die area. The algorithm works in an area recovery mode, where the feedstock pre-placed design is generated with a minimal area increase from the standard cell based ASIC design. The feedstock selection and placement process should also consider the critical paths in the design, to operate in timing aware mode. At the same time, it must be ensured that congestion is not too high in feedstock pre-placed design to obviate any congestion and design related issues. Currently, an area recovery mode is supported in the FP2DG algorithm which tries to generate design that meets the ASIC design functionality requirements at an optimal performance.

3.2 FP2DG Methodology

The feedstock preplaced design generation (FP2DG) methodology is discussed in this section. The feedstock pre-placed design generation is a 2 step process, as shown in Figure 15. In step 1, the design data is generated. The standard cell based physical design ASIC implementation is performed either in Synopsys IC Compiler tool or in Cadence Encounter/Innovus tool. In order to extract the physical and functional information of the design, a Tcl utility is developed which extracts standard cell reference and instance names, locations, load cell instance names, cell pin names, etc. The information obtained from step 1 in a form of text file is fed to the algorithm utility developed in Perl. In step 2, the FP2DG algorithm utility is executed to generate the feedstock pre-placed design summary, instance, and placement reports. These reports are fed to design collateral generation flow to generate golden design netlist and DEF files. The algorithm is implemented based on heuristic and learning analytical design techniques, which are discussed in detail in section 3.4. There are various control knobs provided to the user in order to generate the feedstock pre-placed design as per requirements and to optimize design performance. The controllability options supported by this flow are discussed in detail in section 3.5.

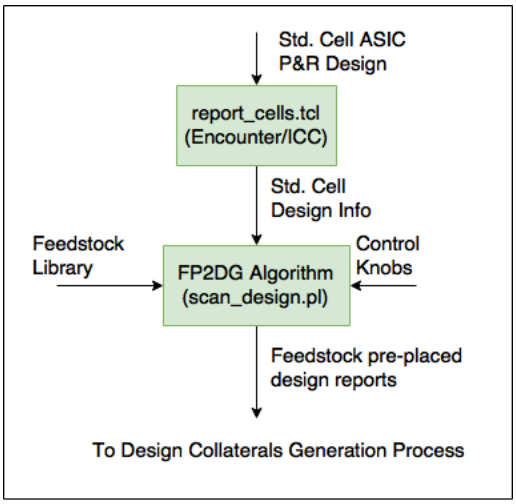


Figure 15: FP2DG Methodology.

3.3 Overview of implemented technique

In this section, a high level overview of the implemented FP2DG algorithm is presented. The details for each of the step in FP2DG process is discussed in detail in section 3.4. The FP2DG algorithm consists of 11 major steps, as shown in Figure 16.

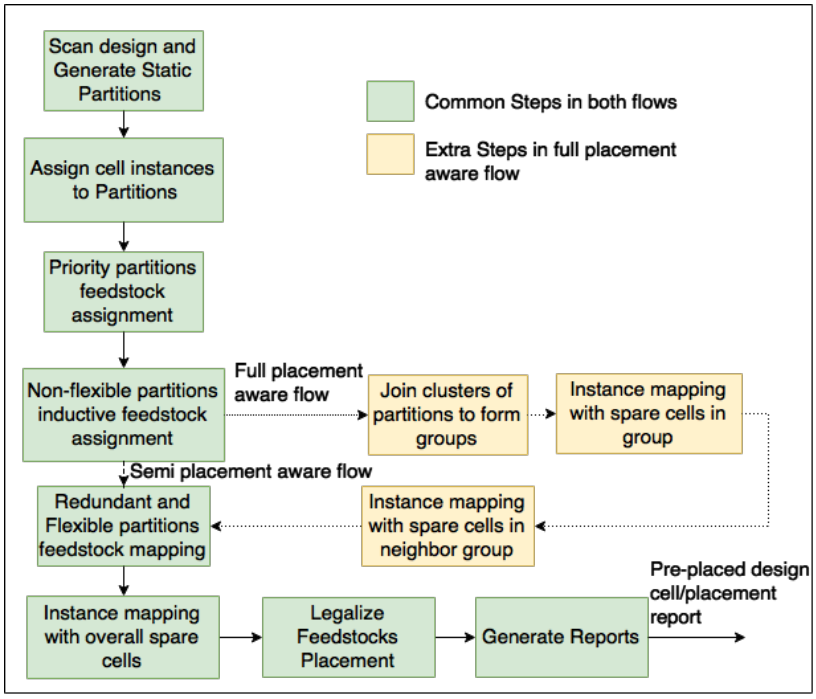


Figure 16: FP2DG Algorithm Overview.

The standard cell based ASIC design information in a form of text file was taken as an input. This information was processed to form the required data structures. Based on this information, the design was scanned and segmented into different partitions of size same as that of a feedstock cell. The mapping of cells in the original ASIC design to the feedstock based design was performed in next steps. The mapping started in the most localized manner and ended with the least localized way. The cells in the ASIC design were mapped to each partition based on their locations. The cells lying within the partition got mapped to the same partition. In step 3, the feedstocks were assigned to the priority partitions. Each feedstock cell was assigned to the partition to find out the most optimal feedstock cell for a particular partition. In case, the difference in the count of similar cells between the best and second best mapped feedstock to a partition was high, and exceeded the threshold value defined by user, the partition was declared as a priority partition. These were the most constrained partitions in the design, and the optimal feedstocks got mapped

to these partitions. In step 4, the neighboring partitions of these priority partitions were assigned a feedstock cell, if those were not defined as the flexible partitions. The criterion to define any partition as a flexible partition in the design depends on the standard cell area utilization in the partition. In case, the partition contained less number of cells, and the standard cell area utilization for the partition was below the user defined threshold value, it was declared as a flexible partition. This inductive feedstock assignment occured to assign feedstocks to the neighboring sites. The mapping occured in such a way that feedstocks assigned to the non-flexible partitions map the cells of their original partitions as well as map the unmapped cells of the priority partition. This process was repeated again and again until all the priority and non-flexible partitions were assigned feedstocks.

The steps 5, 6 and 7 in this 11 step process occured only for the full-placement aware flow. In the full placement aware flow, the clusters of partitions lying close to each other in design were joined to form groups. The number of partitions in a horizontal and vertical dircetion to be grouped was controlled by user. In step 5, groups were formed and partitions were mapped to each group. In step 6, the unmapped cells after step 4 were mapped with the spare cells present within the same group. The spare cells were the cells which were placed in the feedstock pre-placed design but were not required in ASIC standard cell based design. The unmapped cells were mapped with the spare cells lying within the same group. In step 7, the mapping of the unmapped cells occured with the spare cells lying within the neighbor group of the desired feedstock site.

The steps 8, 9, 10 and 11 were common for both the semi-aware and full-aware placement flows. The flexible partitions feedstock assignment was performed in this step. As defined earlier, these partitions had low standard cell area utilization. The feedstocks were assigned to these flexible partitions in such a way that nearby unmapped cells were also mapped to this feedstock. At this stage, all the required feedstock cells were placed in the design. In case there was any valid site in pre-placed design, where feedstock cell was not placed. This could happen in the case when no cells were originally present in the standard cell based design. There was an option controlled by user to place feedstock cells in these valid sites. These feedstocks cells were termed as redundant feedstocks, and were placed in design which might improve the design performance.

Once all the feedstock cells were placed, the mapping statistics were calculated. In case there were unmapped cells, the flow tried to map it with the spare cells lying anywhere in the

design. This was the final mapping step which utilized all the spare cells placed anywhere in design to map the unmapped cells. Then, the feedstock locations were legalized to ensure that no feedstock cell was going out of the valid area, and was not overlapping with any macro cell. Once the legalization of all the feedstock cells was performed, the output reports were generated. The flow generated the feedstock instance map and placement report files which were fed to the design collateral generation process (discussed in chapter 4) to generate the netlist and DEF files of the feedstock pre-placed design. The summary file listing down all the key metrics, discussed in detail in section 3.6 was also generated. The group statistics and standard cell ASIC design report was generated, which was used in the partitioned synthesis flow.

3.4 FP2DG Algorithm Details

In this section, the implementation details of the FP2DG algorithm including all the optimization techniques are mentioned. The detailed explanation of all the 11 steps of the FP2DG algorithm is provided in subsections below.

3.4.1 Static Partitions Generation

This step deals with the segmentation of the design. Once the standard cells information from the reference ASIC design is processed, the whole design is segmented into various small regions, named as partitions. The size of each partition is same as that of a feedstock. In this step, the whole design is scanned first. The location of the cells which lie on the 4 edges i.e. top, bottom, left and right is taken, and the complete region formed by bounding box covering the cells placed on the edges is segmented into partitions of size equivalent to that of a feedstock.

3.4.2 Cells mapping to Partitions

In this step, the standard cells of the ASIC design are assigned to the static partitions generated in the previous step. Based on the physical location of the cells, each cell is assigned to the partition in which it lies. In the process of assigning the cells to each partition, only functionality and input output ports is considered. For example, AND2X1_HVT, AND2X2_HVT, AND2X1_RVT cells have either different drive strength values i.e. X1, X2 or different VT classes i.e. HVT, RVT. All these cells are considered the same since they all belong to the AND2 gate category. In addition to assigning cells to the partitions, the flexible and neighbor partitions are also determined. The coordinates of partitions are calculated first. Then, based on the location of

partition and its size, its neighboring partitions i.e. left, right, up, and down partitions are identified. These neighboring partitions are used in downstream mapping steps of FP2DG process, and will be explained in further sub-sections. Some of the partitions in the design are declared as flexible partitions. The flexible partitions are those partitions in the design which have minimal number of cells inside them. The area utilization of each feedstock, which is the ratio of the area occupied by cells placed inside a partition to the total area of a partition is evaluated. If the utilization value lies below the flexible partition threshold value which is specified by the user, the partition is declared as a flexible partition. The reason for declaring flexible partitions in the design is to assign feedstocks to these partitions at the last, since these partitions have lesser mapping constraints than other partitions placed in design. The feedstock assignment can be done in the end to ensure the optimal feedstock is assigned to these partitions which can map the cells of the flexible partitions as well as map the unmapped cells of the neighboring partitions.

3.4.3 Priority Partitions Feedstock Assignment

This is the first step where the feedstock assignment to the partitions is performed in FP2DG process. In this process, each partition is taken one by one. All the different feedstock cells which exist in the feedstock library are attempted to map with each partition one by one. The similarity analysis is performed for each feedstock assignment to a partition. The similarity analysis involves finding out the common number of cells in the feedstock and a partition. The similarity values for the optimal and the second best feedstock are compared. The best feedstock is the one which has the maximum value of similarity. Similarly, the second best feedstock is determined. In case, there is a significant difference in the similarity values of the best/optimal and the second best feedstock assigned to a partition, which exceeds the priority feedstock criterion specified by the user, the partition is declared as a priority partition. The best feedstock is assigned to it. In this way, the most constrained partitions which have higher standard cell utilization and for which an optimal feedstock can be defined easily without considering the neighboring partitions are assigned the optimal feedstock. In this step, each priority partition is assigned an optimal feedstock. It is possible that some of the cells in these partitions remain unmapped. The mapping of these unmapped cells are taken into consideration in the next steps of the algorithm.

3.4.4 Non-flexible Partitions Feedstock Assignment

Once the priority partitions are assigned feedstocks, feedstock assignment to non-flexible

partitions are performed. In this step, the neighboring partitions of each priority partition, is taken one by one. These neighboring partitions which are neither declared as a priority partition, nor as a flexible partition in the previous steps are further analyzed. The partitions which are neither declared as a flexible nor as a priority are declared as a non-flexible partitions. For each non-flexible neighboring partition, its optimal feedstock is determined which has the highest similarity value. The similarity value in this type of mapping is evaluated by finding the number of common cells between the neighboring partition and feedstock, and the common cells between unmapped cells of already placed priority partition and feedstock. The optimal feedstock cell for each non-flexible neighboring partition is determined. Then, the non-flexible neighboring partition which has the best similarity value when mapped with its optimal feedstock is assigned the feedstock. In this way, this process is repeated till all the non-flexible neighboring partitions of the priority partitions are assigned the feedstocks. This mapping is based on an inductive effect, where the feedstock assignment is based on the supervised learning mechanism from the existing placed feedstocks in the design. It might be possible that there lie some non-flexible partitions in the design which do not have priority partitions as their neighbors. This mapping process ensures assignment of feedstocks to these kind of partitions, and this type of mapping is not terminated until all the non-flexible partitions are assigned the feedstocks.

3.4.5 Groups Formation

The FP2DG algorithm supports two different optimization modes, namely the full-placement aware and the semi-placement aware. The full placement aware flow does a better job in the localized mapping of the cells in comparison to the semi-placement aware flow. The steps discussed in sections 3.4.5, 3.4.6, 3.4.7 are performed only in the case of a full placement aware flow. The semi-aware process skips all these steps and proceeds directly to the step discussed in 3.4.8. In this step, the partitions are clustered to form the groups. The partitions which are lying physically close to each other are clustered together to form groups. The purpose of generating groups is to perform the mapping of unmapped cells of the partitions with the spare cells of the partitions placed within the same group. In this step, neighbor groups, i.e. groups lying in the left, right, up, down directions for each group are also determined. This information is used in the process of cross group mapping, which is explained in sub-section 3.4.7. The size of each group is controlled by the user. The user needs to provide the number of partitions in both horizontal and vertical direction which should be clustered together to form a group.

3.4.6 Group Cells Mapping

As mentioned earlier, this step takes place only in the full placement aware flow. In this step, the unmapped cells in the partitions which belong to the same group are mapped with the spare cells of the partitions which belong to the same group. In this way, mapping of the unmapped cells in the partitions takes place. This kind of assignment is more localized in comparison to mapping of cells with spare cells placed anywhere in the design. The unmapped cells for each partition is evaluated again, once the spare cells get mapped with unmapped cells across different partitions in the same group.

3.4.7 Cross Group Cells Mapping

This process of mapping is similar to the group cells mapping which is discussed above. The only difference in the cross group mapping is that the spare cells get mapped to the unmapped cells of partitions which lie in the neighboring group. The neighbor group for each group is already determined while generating the group metrics, as discussed in section 3.4.5. In this step, the unmapped cells after group cells mapping process are mapped with the spare cells of the partitions which belong to the neighbor group. The next steps are common for the full and semi- placement aware flows.

3.4.8 Redundant and Flexible Partitions Feedstock Assignment

Once the feedstocks are assigned to the priority and non-flexible partitions, and different localized mapping techniques are performed to generate the optimal feedstock pre-placed design, the feedstocks are assigned to the flexible partitions. In this step, the feedstock with the maximum similarity value is assigned to the flexible partition. The similarity value is the sum of the number of common cells between the feedstock and the flexible partition, and number of common cells between the feedstock and the unmapped cells of neighboring partitions. This process of mapping is similar to feedstock assignment of non-flexible partitions. However, the feedstock assignment process for the flexible partitions takes place at the end since they have more flexibility in choosing the feedstock. This ensures that feedstocks placed at these sites map the remaining unmapped cells of neighboring sites. In this step, there is an option to insert redundant feedstocks to the valid sites, if specified by the user. There might exist a scenario, where no cells are placed in the some of the partitions. It can happen, if the original ASIC design has empty regions. The feedstock pre-placed

design can utilize this area to place feedstock cells, since it won't cost any extra area. In case user wants to enable insertion of redundant feedstocks to such sites, the redundant feedstocks can be added to the design. This might help in improving the performance of design while doing synthesis or physical design by providing more cells to the EDA tools.

3.4.9 Placement un-aware Cells Mapping

This is the final step in which the cells are mapped in the design. In this step, no additional feedstocks are placed in the design. In this step, the remaining unmapped cells are mapped with the spare cells of partitions placed anywhere in the design. In this way, this is the placement un-aware mapping of cells where the locations of spare cells are not considered while mapping the unmapped cells. This is the final attempt by FP2DG flow to map the cells in the original design with the cells of the feedstock pre-placed design. In case there are any unmapped cells still present in the design, those are marked as the unmapped cells, and are reported. It might be possible to synthesize the design with few unmapped cells, since synthesis tool does perform logic restructuring. However, the sequential elements like the registers, flops and latches need to be mapped in the feedstock preplaced design. For example, there might be a scenario where NAND gate is missing in the feedstock pre-placed design, but there are spare AND and INV gates present which can implement the functionality.

3.4.10 Feedstock Placement Legalization

In this step, the location of feedstocks are legalized in the feedstock pre-placed design. The feedstocks are originally assigned the location of partitions to which each one is getting mapped in the mapping process. However, there is a need to legalize the placement of the feedstocks in the design. The width of the regions where feedstocks are placed needs to be the integral multiple of the feedstock width. It is however possible that this criterion is not met for some regions in the design. Due to this, there is a need to move the feedstocks to nearby valid sites, or slide the macro cells to the desired location in order to ensure that no feedstock is overlapping with other feedstock or with the macro cell placed in the design. The legalization is controlled by user configuration file where the user can specify the valid feedstock placement sites. Further, the user can specify if there is any notch or if any macro cell is placed in the valid. Based on the user defined slide direction and magnitude, the desired feedstock cells are shifted in order to ensure the proper placement of feedstocks and macro cells in the floorplan.

3.4.11 Reports Generation

This is the final step in FP2DG algorithm process. This step generates all the report files required for the analysis, and in DCG process which is discussed in chapter 4. The summary file captures all the main FP2DG metrics (discussed in section 3.6) to analyze the quality of the feedstock pre-placed design. The feedstock instance map and placement report files are generated which are fed to design collateral generation process (discussed in chapter 4) to generate the netlist and DEF files of the feedstock pre-placed design. The group statistics and standard cell ASIC design report are also generated, and are used in the partitioned synthesis flow. The output files are explained in detail in section 3.7.1.

3.5 Control Knobs and Tradeoffs

In this section, the controllability of the FP2DG algorithm utility is discussed along with consideration of design tradeoffs. In order to develop a generic solution which can be used for different corner cases, multiple customized variables are provided to the user. Following are the main control knobs provided to user while executing the algorithm utility.

3.5.1 Full/Semi placement aware flow

The FP2DG flow supports two different optimization modes, namely full placement aware and semi-placement aware modes. The user has an option to run any of the optimization modes. The priority and non-flexible partition feedstock assignment is enabled for both the flows. However, the group creation, and group and cross group mapping is enabled only in the full-placement aware mode. The critical path length is expected to be less for the full-placement aware mode.

3.5.2 Redundant feedstock cells insertion

There might exist some valid standard cell placement sites in the design where no cells are originally placed in the ASIC design. There is an option to insert feedstock cells at such sites, if desired by the user. It is benefecial to insert the feedstock cell at such sites, since adding the feedstock cells won’t cost any additional area and might result in improving the design QoR in terms circuit performance/speed.

3.5.3 Priority Partitions Criterion

The FP2DG flow first assigns feedstocks to the priority partitions, and later to non-flexible and flexible partitions. The criterion to define any partition in the design as a priority partition is controlled by the user. In the feedstock assignment process, each feedstock cell is assigned to a partition to find out the most optimal feedstock cell for a particular partition. In case, the difference in count of the similar cells between the best and second best mapped feedstock to a specific partition is high, and exceeds threshold value defined by the user, the partition is declared as a priority partition and best feedstock gets mapped to it. Thus, the user defines the priority partition criterion in the design by specifying the threshold for the similarity difference between best and second best mapped feedstock to a particular partition.

3.5.4 Flexible Partitions Criterion

The flexible partitions are the partitions which are assigned feedstocks at the last, after the feedstock assignment to priority and non-flexible partitions has taken place. The criterion to define any partition as a flexible depends on the standard cell utilization in the partition. In case, the partition contains less number of cells, and the standard cell area utilization for the partition lies below the user defined threshold value, the partition is declared as a flexible. The intent is to place more constrained partitions first, so that when the flexible partitions are placed in the design at later stage, the feedstock assignment to these partitions are going to map the unmapped cells from the constrained neighbor partitions.

3.5.5 Group Size

In the full placement aware mode, the partitions which are lying physically close to each other are clsutered together to form the groups. The number of partitions in the vertical and horizontal direction to be grouped together is controlled by the user. Once the groups are formed, the unmapped cells in each partition for a particular group is mapped with the spare cells of the same group. Thus, if the group consists of more partitions, there is a high probablity of mapping the unmapped cells to the spare cells. However, this might result in large interconnect delay, since the cells belonging to the same logic path might be placed far away in the design. Thus, the size of the group should be set neither too small nor too large.

3.5.6 Waive mechanism

The FP2DG flow supports two placement optimization modes, which again consists of various steps in the mapping of cells in ASIC design to feedstocks. There is a possibility that after the mapping and feedstock assignment process, few standard cells are not mapped to any feedstock in the feedstock pre-placed design. It is not necessary to map all the cells, as the synthesis tool can perform logic restructuring to implement the same functionaliy. Thus, there is an option supported in the flow which can be used by user to generate the output reports even if all the cells are not mapped in the design.

3.6 Key Metrics

In this section, the key metrics generated by the FP2DG algorithm for analysis are described. These metrics give an idea on the quality of the feedstock pre-placed design generated by the FP2DG flow. Following sub-sections lists down the main metrics generated by the flow.

3.6.1 Standard Cell Count Ratio of FP2D to ASIC design

The standard cell count in the feedstock pre-placed design is reported, and its comparison with the standard cell based ASIC design is performed and reported in the output summary file. The standard cell count for the feedstock pre-placed design is calculated by adding the standard cell count for each feedstock instantiated in the design. The ratio of the cells in the feedstock pre-placed design to the ASIC implementation is calculated. Higher the ratio, probably more will be the congestion if both the designs have almost the same area. It is also an indicator of the quality of feedstock library. This ratio being closer to 1 indicates that the cells placed inside feedstock have high similarity with the cells placed in the design, and will result in an optimal performance.

3.6.2 Standard cell Area Ratio of FP2D to ASIC design

Similar to the standard cell count ratio, the standard cell area ratio for the feedstock pre-placed design to ASIC design is calculated. This ratio indicates an approximate increase in the overall area of the M2A2 feedstock design in comparison to the standard cell based ASIC design. It is calculated by taking the ratio of the area occupied by standard cells in the M2A2 design to the area occupied by cells in the ASIC design. It is not the ratio of the die area of both the designs.

However, the die ratio can be calculated easily by evaluating the coordinates of the floorplans. The higher ratio implies more die area which increases the NRE cost.

3.6.3 Spare cells, Unplaced and Placed cells count

In the feedstock pre-placed design, the feedstocks are instantiated in the design to replicate the standard cells placed in ASIC design. Thus, FP2D might contain some cells which are placed in design but are not present in the ASIC standard cell based design. These cells are called spare cells. The placed cells are the cells which are required in the feedstock pre-placed design to map the standard cells placed in the ASIC design. The unplaced cells are the cells which are placed in the standard cell ASIC design and are not present in the feedstock pre-placed design. The count of all these types of cells is reported to analyze the performance of the FP2DG algorithm.

3.6.4 Valid, Placed and Redundant Feedstock statistics

In the feedstock pre-placed design, the feedstock cells can be placed on all the valid sites. The valid sites are the locations in the design where standard cells can be placed. There might exist some valid sites in the design where standard cells are not placed in the original ASIC design. However, there is an option to place feedstock cells at such sites. These feedstocks are defined as redundant feedstocks, since these are not required but placed in the feedstock pre-placed design. The placed feedstocks are those which are placed at the valid sites and are required, since cells are present on these locations in the original ASIC design. The count of all the placed, redundant and valid feedstocks are reported in the summary file.

3.6.5 Feedstock Cell Distribution

The feedstock pre-placed design instantiates different types of feedstock cells to replicate the standard cell based ASIC implementation. This information on instance count of all the feedstock cells in design is captured to get an idea about the most widely used feedstocks in the design. It also provides an insight for those feedstock cells which are not used frequently. The analysis of this information can lead to improvement in the feedstock cell design, which eventually improves the usage of those feedstocks which are underutilized, i.e. not used frequently in design.

3.6.6 Instance Mapping Statistics

The FP2DG process involves the mapping of cells in 5 steps, starts from the maximum localized mapping and ends with the least localized mapping. The metrics for each of the mapping steps are mentioned in the following sub-sections.

3.6.6.1 Instance mapping with desired feedstock site

This is the most localized mapping of cells in the ASIC design to FP2D. The percentage of cells in the ASIC design which gets mapped to the standard cells of the feedstock placed at the same location, where cells are originally present in ASIC design is reported in this mapping metric. The maximum deviation in the location of any cell is the sum of height and width values of a feedstock. It is desired to have a high value for this type mapping, since it leads to low interconnect delay.

3.6.6.2 Instance mapping with neighbor of desired feedstock site

The cells in the ASIC design which are not mapped with the desired feedstock site are attempted to map with the spare cells of the feedstock placed at the neighboring site. The percentage of cells which are mapped with cells of neighboring feedstock site is reported by this metric. It is the second best localized mapping for the cells. The maximum deviation in the location of any cell in both the designs is twice the sum of width and height values of a feedstock. The path length and interconnect for this kind of mapping is expected to be more than that of a mapping within a feedstock, since the deviation in the location of cells is more in this type of mapping.

3.6.6.3 Instance mapping with group of desired feedstock site

In the full placement aware flow, the cells in the ASIC design which do not get mapped to the original or neighbor feedstock sites are attempted to map with the other feedstock sites placed in the group. The group is a cluster of physically connected feedstock sites in the feedstock pre-placed design. The cells in the ASIC design which get mapped to the cells of feedstocks placed within a group of desired feedstock site, other than the desired and the neighbor feedstock, are reported by this metric. The maximum deviation in the location of any cell in both designs for this type of mapping is the sum of width and height values of a group.

3.6.6.4 Instance mapping with neighbor group of desired feedstock site

This mapping is also supported for the full placement aware flow. This mapping is similar to the mapping within a group, but the group in this case is not the group of the desired feedstock site, but the neighboring group. The cells which remains unmapped after the group mapping are attempted to map with spare cells of the neighboring group. The maximum deviation in the location of any cell in both designs is twice the sum of width and height values of a group.

3.6.6.5 Instance mapping with spare cells placed at other locations

The cells which are not mapped till the previous stage, which differs for both semi placement aware and full placement aware flows are mapped to the spare cells in feedstocks placed anywhere in design. This mapping is the least localized and cells can be mapped to the spare cells placed anywhere in the design. The maximum deviation in the location of any cell in both the designs is the sum of width and height values of the design. Thus, this kind of mapping might result in the significant path length and interconnect delay. The percentage of cells which get mapped in this way is reported by this metric.

3.6.6.6 Unmapped Instances

The cells in the ASIC design which do not get mapped to the cells in the feedstock pre-placed design are reported by this metric. The cell types along with their count and overall percentage of unmapped cells are reported in the summary file to get the QoR for FP2DG process.

3.7 Deliverables

In this section, the EDA software information in terms of perl script developed in the feedstock library generation process is mentioned. The usage model, paths, inputs, outputs and other important information is also captured for better understanding of the flow to the reader.

3.7.1 Script #1: scan_design.pl

- ***Function:***

The purpose of this perl script is to generate the feedstock pre-placed design based on the standard cell ASIC design by executing the FP2DG algorithm. It generates the feedstock instance

and placement reports which are used in the design collaterals generation process to generate the golden design collaterals i.e. golden design netlist and DEF files. The golden design collaterals are required in the synthesis flow.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis/scripts/scan_design.pl

-design_file	*<Instance report file dumped from ICC/Encounter shell>*	*(REQUIRED)*
-feedstock_output	*<Feedstock output report file>*	*(REQUIRED)*
-feedstock_summary	*<Feedstock summary file>*	*(REQUIRED)*
-coordinate_file	*<Coordinates file with standard. cell placement area>*	*(OPTIONAL)*
-semi_placement_aware	*(Specify in case group sharing is not considered)*	*(OPTIONAL)*
-no_place_all	*(Specify in case user do not want to place feedstocks on all empty sites)*	*(OPTIONAL)*
-waive	*(Specify this switch to proceed in case cells are not found)*	*(OPTIONAL)*

- ***Inputs:***

1. –design_file <Design/Cell report>

This file is generated by sourcing *report_cells_icc.tcl* or *report_cells_encounter.tcl* in Cadence Encounter/Innovus or Synopsys ICC sessions respectively, after the standard cell based P&R design is loaded. This file lists down the standard cell instance and reference names, locations, input, output and inout pin names, load cell instance and reference names for all the reference designs.

2. –feedstock_output <Feedstock output report>

This file is generated by the feedstock library generation process. This file captures the information related to all the standard cells present inside each feedstock. It lists down the cell instance and reference names, count, input, output and inout pins.

3. –feedstock_summary <Feedstock summary file>

This file is generated by the feedstock library generation process. This file captures the

summary and key metrics of results generated by the feedstock library generation script. This file is fed as an input to perform the sanity checks on the feedstock library by verifying its contents with that of the feedstock output file.

4. –coordinate_file <Coordinate file>

This switch is an optional argument. The coordinate file contains the coordinates of standard cell valid, and invalid sites. The feedstocks are placed at the valid sites, and invalid sites are marked with blockages. In the case of complex floorplan shapes like T-shaped, L shaped, notches, etc., there might arises a need to shift the feedstock cells in horizontal or vertical direction to resolve legalization placement issue. Thus, a mechanism to shift specific feedstocks in the horizontal or vertical direction is provided in this file.

5. –semi_placement_aware

This switch is an optional argument. The flow supports two different flows – full placement aware and semi-placement aware. By default, full placement flow is enabled. This switch can be used in case the user wants to enable the semi-placement aware flow.

6. –no_place_all

This switch is an optional argument. By default, the flow places the feedstock cells in all the valid sites, which might result in the placement of redundant feedstocks in the design. In order to place only the required feedstocks in the design, this switch should be enabled.

7. –waive

This switch is an optional argument. In case, the user wants to ignore the warning and errors given by script, and wants to generate the output files, this switch should be enabled.

- ***Outputs***

1. Summary report – summary.rpt

The summary of all the key metrics obtained in the feedstock pre-placed design generation process, which is discussed in section 3.6 is listed in this file. The file gets generated in this directory: ${PWDIR}/REPORTS directory, where ${PWDIR} is the present working directory.

2. *Golden design map report – design_map.rpt*

The mapping information of the instance and library cell names for all the feedstock cells instantiated in the design is listed in this output file. For example, if there are 5 different types of feedstocks, which are instantiated 100 times in the design, then instance names of feedstock varies in range 1-100, whereas the feedstock library cell names ranges from 1-5. Each instance of feedstock cell is mapped to 1 of the 5 feedstock cell types. This file gets generated in this directory: ${PWDIR}/REPORTS directory, where ${PWDIR} is the present working directory.

3. *Golden design placement report – design_placement.rpt*

This output file specifies the location of all the feedstocks instantiated in the design. In order to generate the DEF file of the feedstock preplaced golden design, the respective locations of all the feedstock instances are captured in this file. The file gets generated in this directory: ${PWDIR}/REPORTS directory, where ${PWDIR} is the present working directory.

4. *Group Statistics report – group_stats.rpt*

This output file lists down the information for all the groups created in the design while scanning and partitioning the design. This file can be used by user for analyzing the feedstock cells distribution in the design. The file gets generated in this directory: ${PWDIR}/REPORTS directory, where ${PWDIR} is the present working directory.

5. *Design scan report – scan_revised_design.rpt*

This output file contains the standard cell reference, instance names and locations for all the cells present in the revised ASIC design. This file is fed as an input to *assign_ionames.pl* script (discussed in chapter 5). The file gets generated in this directory: ${PWDIR}/REPORTS directory, where ${PWDIR} is the present working directory.

Chapter 4: Design Collaterals Generation for Synthesis and P&R Flow

In this chapter, the collaterals generation process for synthesis and backend physical design process is presented. The overview and need of these design collaterals is described first in section 4.1. Then, the approach and methodology adopted by author to generate these design collaterals is discussed in section 4.2. The implementation details are captured in section 4.3. Finally, the deliverables i.e. EDA software information in terms of usage model, purpose, inputs and outputs are listed down in section 4.4.

4.1 Overview and Need of design collaterals

There are different design phases in ASIC design implementation process, which make use of design tools provided by multiple EDA vendors. The major design implementation steps involved in our project work or any typical ASIC design flow include synthesis, floor planning, power planning, placement, clock tree synthesis, routing, etc. In order to effectively implement the design using different EDA tools, standard IEEE design format files are used. In our work flow, we have followed the standard industry approach and are making use of these standard files to integrate our feedstock based flow with the existing EDA tools.

The design collaterals required in our feedstock based synthesis and placement & route (P&R) flow include liberty files, netlist and DEF file. All these collaterals are used in design and capture some of the essential information required by the EDA tools to interpret the design. The purpose of all these collaterals along with their description is mentioned in the following sub-sections.

4.1.1 Liberty (.lib) file

The liberty file is an ASCII representation of the timing and power parameters associated with standard cell in a particular technology node. The timing and power parameters for each standard cell are represented either in non-linear delay model (NLDM) tables or current composite source (CCS) formats. These parameters are obtained by simulating the standard cell under a variety of corners, i.e. different process, voltage and temperature (PVT) conditions [28]. This information is required to perform the timing analysis of circuit, to calculate the input output (IO) delays, other path delays, and power information. During the process of synthesis and P&R, gate

delays are evaluated from this file, based on the slew rate and capacitance load values.

4.1.2 Netlist (.v) files

In the ASIC design, the standard cells, macro cells, etc. are connected in a unique way to implement the desired functionality and meet timing constraints [29]. The connectivity information capturing the way these cells are connected is mentioned in netlist file. The synthesis tool generates synthesized netlist from RTL. The register transfer level (RTL) is a design abstraction which models circuit in terms of flow of data between hardware registers. This logic gets mapped to standard cells in an optimal way to meet timing, area, and power constraints, and generates the synthesized netlist. The netlist file can be flattened or hierarchal. The flat netlist has all the instances in one level. In a hierarchical netlist, the instances are present in different levels of abstraction. The top level in design instantiates all the modules, i.e. instances of modules are accessed from top level.

4.1.3 DEF (.def) files

Design Exchange Format (DEF) is a specification for representing the physical layout of an integrated circuit in an ASCII format. It represents the netlist and circuit layout. DEF is used in conjunction with Library Exchange Format (LEF) to represent complete physical layout of an integrated circuit while it is being designed. The DEF file contains both the physical information and the connectivity information which is also captured in netlist file. Apart from this, all the pins, ports, nets physical information which includes location of cells, macros, metal layers, and non-default rules (NDR) are captured in this file.

4.2 Collaterals Generation Approach and Methodology

In this section, the approach and methodology adopted to generate design collaterals, i.e. liberty files, netlist and DEF files is discussed. The standard backend CAD methodology is adopted to generate these design collaterals. The collaterals generation flow has dependencies on feedstock generation flow (discussed in Chapter 2) and scan design (feedstock pre-placed design generation) flow (discussed in Chapter 3). The reference DCG flow diagram is shown in Figure 17. The reader can clearly observe the inputs and outputs from each process. The usage, inputs, outputs and other relevant details of the DCG EDA software are mentioned in section 4.4.

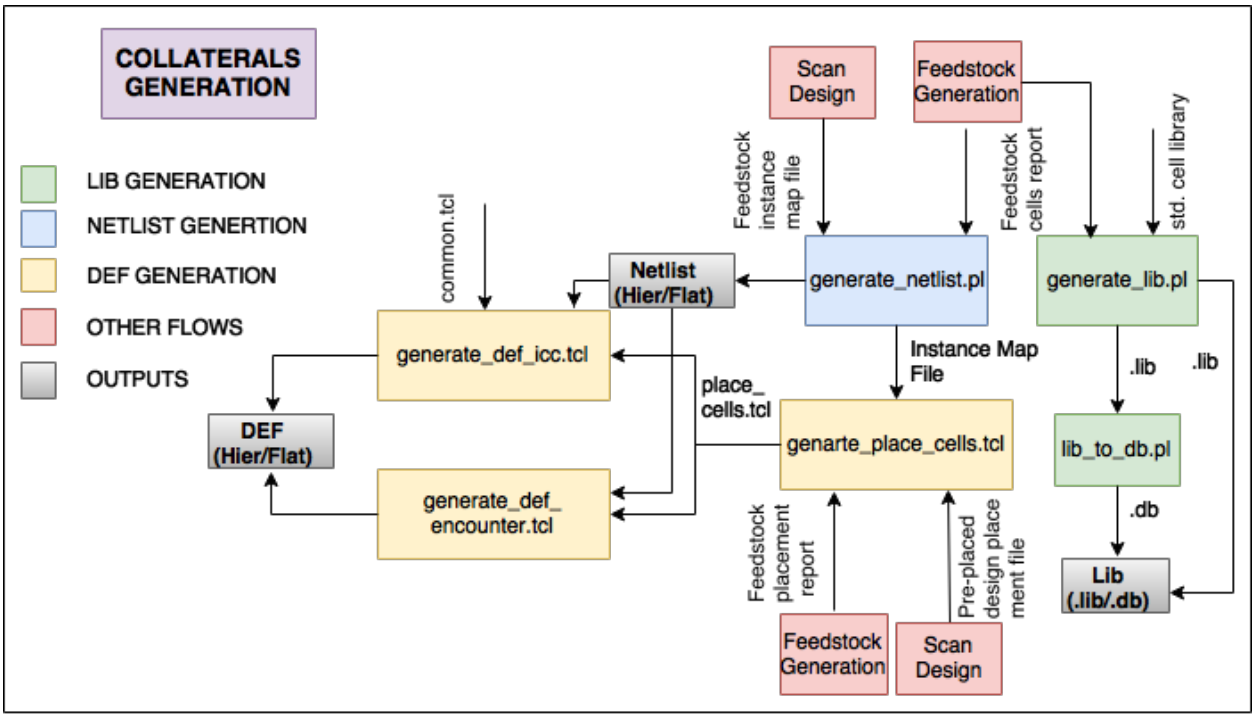


Figure 17: Design Collaterals Generation Flow.

The liberty files generation flow has input dependencies on feedstock library generation flow and standard cell library. The feedstock generation flow generates the feedstock cells report which lists down the different types of standard cells along with their count, pin names, which are instantiated in the feedstock. The liberty generation flow derives information for the required standard cells (liberty view needs to be generated for these cells) from this report. It extracts the liberty view information for these cells from the standard cell library to generate the output liberty file for all the feedstocks. The output feedstock library is actually a subset of existing standard cell library. The intent of generating feedstock library is to reduce the turnaround time (TAT) of manufacturing system on chip (SoC) by reducing the time taken by different EDA tools to load design library.

The netlist generation of feedstock pre-placed design is discussed below. The purpose of this flow is to generate the netlist file for the feedstock pre-placed design which forms the base layer in final SoC. The DEF file being generated from this output netlist is fed as the golden design DEF in Conformal ECO flow to meet the functionality of customer ASIC design. It uses feedstocks as primitive cells in design instead of standard cells, which is the conventional ASIC approach. It also instantiates memory macros and hard IPs in the golden design, similar to ASIC design

approach. In other words, post mask ECO synthesis is performed using pre-placed feedstock design as the reference golden design. The netlist generation flow has input dependencies on feedstock generation and scan design flow. The feedstock preplaced design generation flow generates the feedstock instance map file, which lists down all the types and count of feedstocks instantiated in the pre-placed feedstock design. The feedstock instance and library cell names are reported in this file. The feedstock cell instance name is unique for each feedstock cell placed in design. However, feedstock cell library name determines the type of feedstock cell. For example, assume there are 5 types of feedstock cells which are instantiated 100 times in design. In this case, there will be 100 instance names, unique for each instance, whereas the feedstock library cell name for all these instances is not unique, and will be one of the five possible feedstock library cell names. The information about the standard cells which constitute each feedstock is obtained from the feedstock cell report, which is generated by feedstock library generation flow.

The DEF generation process for the feedstock pre-placed design is similar to the netlist generation process. The DEF file represents the netlist and physical layout information. This flow again has dependencies on feedstock generation and scan design flows. The feedstock placement report listing down the physical coordinates for all the standard cells present inside each feedstock is generated by feedstock library generation process. The feedstock pre-placed design placement report lists down the location of all the feedstock cells instantiated in design. Based on these two reports, physical location of all the standard cells in the feedstock pre-placed design is obtained. The instance names of all these standard cells are obtained from the netlist generation process. Based on this information, Tcl command file is generated to place the standard cells in design, which is compatible with both Synopsys and Cadence tools. Then, the DEF file is generated by reading the netlist file and sourcing the Tcl command file in either Synopsys IC Compiler or Cadence Encounter/Innovus tools. The output DEF file is used as the golden design DEF file in the Conformal ECO flow to perform post mask ECO synthesis, i.e. to map functionality of customer RTL design using feedstock pre-placed design which forms the customized base layer in the final SoC.

4.3 Implementation Details and Features

In the following sub-sections, the implementation details of all the collaterals generation steps are discussed.

4.3.1 Liberty files generation

Following are the main features of the utilities developed in the liberty files generation flow:

- The output feedstock library is a subset of existing standard cell library. This library is design dependent and is unique for each design, unlike the conventional static standard cell library in ASIC flow. There are no redundant standard cells in feedstock library. This works well in our flow because the base layer is fixed. However, in conventional ASIC flow, the standard cells can be added in any design phase for optimizations. Thus, there is a need to keep all the standard cells in design library.
- The liberty files can be generated for specific corners, if required. The corners represent simulation conditions: process, voltage and temperature (PVT). By default, the flow generates the liberty files for all the corners present in existing standard cell library area. If required, liberty files can be generated for specific corners, thereby saving disk space and reduces library load time in EDA tools to save the NRE costs.
- The liberty files are generated for each feedstock separately. The information about the type of standard cells present in each feedstock is obtained from the feedstock cell report, which is generated by the feedstock library generation flow, as discussed in chapter 2. This is done to ensure that only required liberty files are loaded in tool while performing synthesis of specific partitions. This again saves the run time, and reduces the overall turnaround time (TAT).
- The standard cell library can be obtained from multiple standard cell library paths. The author has provided wild character support in paths, i.e. * character is supported in path name.
- Various sanity checks are added in the flow to ensure high quality of output collaterals. Some of the sanity checks include checking if the liberty view for all the cells present in design exists in standard cell library area, if file paths and corner information provided by user is valid or not.
- The Synopsys Design Compiler and IC Compiler also reads library in db format. Therefore, utility is developed to convert liberty file from .lib format to .db format.

4.3.2 Netlist files generation

Following are the main features of the utilities developed in the netlist files generation flow:

- The output netlist file is generated by extracting information on feedstock cells used in design from scan design (F2DG) flow, and information on standard cells present in each feedstock from feedstock library generation (FLG) flow.

- The output netlist can be either hierarchical or flattened, as desired by the user. This is controlled by providing a command line switch to the user while running the netlist generation script. In hierarchical netlist, there exists a top level with design name, and all the feedstocks are considered as sub-modules of top level. In the flattened approach, all the feedstocks are considered at top level. The design information remains the same in both of these formats. However, the cells instance naming convention differs and is generated based on standard CAD approach followed in industry. By default, the flat netlist is dumped by DCG flow.
- The flow generates the instance map file in addition to the netlist file. This instance map file serves as an input to the DEF generation process. This file captures the instance names of all the cells present in output netlist file. Based on this information, the output DEF file assigns locations to all the cells present in design.
- While performing ECO based synthesis, it is required in some cases that netlist should not have ports at the top level. This is done to ensure the correct mapping of cells in Conformal ECO flow. To meet this requirement, an option is provided in the netlist generation flow to skip top level ports in the output netlist. By default, the DCG flow includes all the ports at top level.
- Various sanity checks are added in the flow to ensure high quality of output collaterals. Some of the sanity checks include checking if input file paths exist, if the input files are readable or not, if any input collateral is missing or not, etc.

4.3.3 DEF files generation

Following are the main features of the utilities developed in the DEF files generation flow:

- The DEF generation flow comprises of 2 steps. In step 1, the standard cells placement command file is generated. This is an intermediate output which is fed as an input to the step 2 of the flow in order to generate the output DEF file.
- The DEF file generated can be either hierarchical or flat, and depends on the type of input netlist.
- The flow supports DEF generation from both Cadence Encounter/Innovus tool and Synopsys IC Compiler tools.
- In step 1, the instance map file generated by the netlist generation process, feedstock placement report generated from FLG flow, and pre-placed design placement report generated from scan design (FP2DG) flow are fed as inputs. The flow extracts the instance names, physical coordinates of all the standard cells present inside each feedstock and their relative physical

location from these files. The step 1 generates a Tcl command file which contains the standard cell placement commands.

- In step 2, IC compiler or Encounter/Innovus tool is invoked. Then, the step 2 Tcl utility, i.e *generate_def_icc.tcl* or *generate_def_encounter.tcl* is sourced to generate the output DEF file. This utility reads in the step 1 output command file to place the cells in design. By default, the output DEF file has all the information related to floorplan, ports, PG straps, standard cell, memory macros and hard IP blocks locations, nets, NDRs, etc.
- The step 2 Tcl utility comprises of 2 sections. The section 1 lists down the input parameters which can be configured by user. Some of the configurable parameters include IO to core boundary margin, power rail metal width, design settings file path, standard cell row height, etc. The design commands are mentioned in the section 2 and should not be changed by user.
- Various sanity checks are added in the flow to ensure high quality of output collaterals. Some of the sanity checks include checking if input file paths exist, if the input files are readable or not, if any input collateral is missing or not, etc.

4.4 Deliverables

In this section, the EDA software including the perl and Tcl scripts/utilities developed in collaterals generation process is mentioned. The usage model, paths, inputs, outputs and other important information is also captured for the better understanding of flow to the reader.

4.4.1 Liberty files generation

This sub-section describes the liberty files generation process.

4.4.1.1 Script #1: generate_lib.pl

- ***Function:***

The purpose of this perl script is to develop the liberty files for each feedstock. The list of standard cells which are used in the feedstock is obtained by feedstock report. The liberty view information for the required cells is then obtained from standard cell area to generate the feedstock library, which is a subset of the existing standard cell library. This library contains only the required number of cells for each feedstock. Thus, the synthesis and P&R tool takes less time to load feedstock based cell library.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/lib_genertion/scripts/generate_lib.pl

-file <Feedstock output report> (REQUIRED)

-libDirs List of lib directories, i.e. <libDir1> <libDir2> Wild character (*) is supported in the library directory names (REQUIRED)

-corners List of corners for which you want to generate lib files i.e. <corner1> <corner2> If not specified, lib files are generated for all corners present in libDirs (OPTIONAL)

- ***Inputs:***

1. *-file <Feedstock output report>*

This file is generated by the feedstock generation script. This file lists down the type and count of standard cells present inside each feedstock. This file also captures the input, output and inout pin names for each standard cell present inside feedstock.

2. *-libDirs List of lib directories, i.e. <libDir1> <libDir2> ...*

The user needs to specify the list of standard cell library directory paths in this argument. The user can specify multiple library directory paths in a list format (separated by spaces). The wild character is also supported in paths, i.e. the user can mention * in library directory paths.

3. *-corners List of corners for which user wants to generate lib files i.e. <corner1><corner2>..*

This switch is an optional argument. In case, the user wants to generate the liberty files for some specific corners i.e. for specific PVT conditions, user can use this option and provide a list of corners. By default, lib files are generated for all the corners present in standard cell library.

- ***Outputs***

1. *.lib files for each feedstock*

The output lib files containing all the required standard cells information for each feedstock are generated in this directory, where ${PWDIR} is the present working directory - ${PWDIR}/ LIB/ ${feedstock}.

4.4.1.2 Script #2: lib_to_db.tcl

- ***Function:***

The function of this utility is to convert liberty file information from .lib format to .db format. In some of the synthesis and backend design tools, .db format is used. It is the compressed binary format to represent the same information which exists in liberty files.

Usage: *source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/lib_genertion /scripts/lib_to_db.tcl in ICC shell*

- ***Inputs:***

1. The user needs to specify the correct liberty file names in the editable section of the script, which need to be converted in .db format.

- ***Outputs:***

1. The user needs to specify the correct output .db file paths in the editable section of the script.

4.4.2 Netlist generation

This sub-section presents the netlist files generation process.

4.4.2.1 Script #3: generate_netlist.pl

- ***Function:***

The purpose of this script is to generate the netlist file of the input design. The input design can consist of any number of feedstocks. The output netlist can be generated either in a flattened mode or in a hierarchical mode, as per user needs. In case the top level port names are not required in the output netlist, an option is provided in the utility to skip them.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/netlist_genertion/scripts/generate_ netlist.pl

-file	*<Feedstock output report>*	*(REQUIRED)*
-design_name	*<Top module name in output netlist file>*	*(REQUIRED)*
-map_file	*< Feedstock Map file to map cells properly. Format is:*	
	<Feedstock instance name>:<Feedstock lib cell name>	*(REQUIRED)*
-hierarchical	*(Specify this switch in case you want to dump hierarchical netlist)*	
	By default: flat netlist is dumped	*(OPTIONAL)*
-noports	*<In case user do not want to specify in/out ports in top level>*	

- ***Inputs:***

1. -file <Feedstock output report>

This file is generated by the feedstock library generation script. This file lists down the type and count of standard cells present inside each feedstock. This file also captures the input, output and inout pin names for each standard cell present inside feedstock.

2. -design_name <Top module name in output netlist file>

The top level module name of the output design is mentioned in this switch. The generated netlist has a top level module named after this argument value.

3. -map_file < Feedstock Map file to map cells properly>

The purpose of this file is to provide the mapping information of the instance and library cell names for all the feedstock cells instantiated in design. For example, if there are 5 different types of feedstocks, which are instantiated 100 times in design, then instance names of each feedstock instance is unique, whereas feedstock library cell names is going to be one of the 5 possible values.

4. -hierarchical

This switch is an optional argument. In case the user wants to generate a netlist file in an hierarchical format, please use this switch. By default, a flat netlist is generated by this utility.

5. -noports

In case the user does want to skip the top level ports information in the output netlist, please specify this switch. By default, the top level ports are generated in the output netlist.

- ***Outputs:***

1. ${designName}.v

The utility generates an output netlist file in this path: ${PWDIR}/NETLIST /${designName}.v, where ${PWDIR} is the working directory.

2. ${designName}_cellmap.rpt

The utility also generates the cell map file, which lists down the library and instance cell names, pin names for all the standard cells used in each feedstock. This file captures this information for all the feedstocks instantiated in design.

4.4.3 DEF generation

This sub-section discusses the DEF file generation process.

4.4.3.1 Script #4: generate_place_cells_tcl.pl

- ***Function:***

The purpose of this script is to generate an intermediate output Tcl file which gets sourced in DEF generation script to generate the output DEF file. This Tcl file is compatible with both Synopsys IC Compiler and Encounter/Innovus tools, and contains standard cells placement commands.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/def_generation/scripts/generate_place_cells_tcl.pl

-feedstock_place_file	*<Feedstock cell placement report>. Specifies where cells are placed in each feedstock*	*(REQUIRED)*
-design_place_file	*<Design placement report>. Specifies where feedstocks are placed in design*	*(REQUIRED)*
-instance_map_file	*<Design lib cell-instance map file>. Maps lib cells for all the feedstocks with instance names*	*(REQUIRED)*
-design_name	*<Top module name in netlist file>*	*(REQUIRED)*
-synopsys	*Specify this switch if DEF needs to be dumped by ICC By default: disabled this switch, ports are dumped.*	*(OPTIONAL)*

- ***Inputs:***

1. -feedstock_place_file <Feedstock placement report>

This file is generated by the feedstock library generation script. This file lists down the locations for all the instances of standard cells present inside each feedstock.

2. -design_place_file <Design placement report>

This file is generated by FP2DG algorithm, script. This file lists down the locations for all the feedstock cells instantiated in design. In order to generate the DEF file for a feedstock preplaced golden design, the respective locations for all the feedstock instances are captured in this file.

3. -instance_map_file <Design libcell-instance map file>

The file is generated by the netlist generation script. It lists down the library and instance cell names, pin names for all the standard cells used in each feedstock. This information is captured for all the feedstock instances in this file.

4. -design_name <Top module name of netlist file>

The top level module name of the output design is mentioned in this switch. The generated DEF has a top level module named after this argument value.

5. -synopsys <Specify this switch if DEF needs to be dumped by ICC>

This is an optional argument. The user can specify this switch if output the DEF file needs to be generated from Synopsys ICC tool. By default, Cadence Encounter/Innovus tool is supported.

- ***Outputs:***

1. place_cells.tcl

This utility generates *place_cells.tcl* Tcl file which gets sourced in the DEF generation script, in order to generate the DEF file. This Tcl file is sourced either in Synopsys ICC or in Cadence Encounter/Innovus tools. It contains the standard cell placement commands for all the standard cells instantiated in all the feedstock cells placed in design.

2. golden_design.rpt

This file is required while assigning the cells connected to IO ports during the synthesis process. This file contains the following information for the feedstock preplaced design – standard cell instance and library cell names, locations, pins, for all the standard cells instantiated in design.

4.4.3.2 Script #5: generate_def_icc.tcl

- ***Function:***

The purpose of this script is to generate the DEF file for the feedstock pre-placed design. This script gets sourced in Synopsys IC Compiler tool. The output DEF file supports both flattened and hierarchical design formats.

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/def_genertion /scripts/generate_def_icc.tcl in ICC shell

- ***Inputs:***

1. place_cells.tcl

This Tcl file gets dumped by *generate_place_cells_tcl.pl* script. It contains the standard cell placement commands for all the standard cells instantiated in all the feedstock cells placed in design.

2. common.tcl

This Tcl file contains all the settings and library information required to load the design in ICC shell.

- ***Outputs:***

1. ${design_name}.def

The utility generates the DEF file. The output DEF file has all the information related to floorplan, ports, PG straps, standard cell and macro cells locations, nets, NDRs, etc.

4.4.3.3 Script #6: generate_def_encounter.tcl

- ***Function:***

The purpose of this script is to generate the DEF file of the feedstock pre-placed design. This script gets sourced in Cadence Encounter/Innovus tool. The output DEF file supports both flattened and hierarchical design formats.

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/def_genertion /scripts/generate_def_encounter.tcl in Encounter/Innovus shell

- ***Inputs:***

1. place_cells.tcl

This Tcl file gets dumped by *generate_place_cells_tcl.pl* script. It contains the standard cell placement commands for all the standard cells instantiated in all the feedstock cells placed in design.

Note: This script has editable section where user can provide configurable parameters to meet design needs.

- ***Outputs:***

1. ${design_name}.def

The utility generates the DEF file. The output DEF file has all the information related to floorplan, ports, PG straps, standard cell and macro cells locations, nets, NDRs, etc.

Chapter 5: Synthesis of Feedstock Based Design

This chapter deals with the synthesis of ASIC design using feedstock cells in the design library, unlike conventional ASIC flow approach which makes use of the standard cells in design. In this chapter, the synthesis approach, methodology and flow implementation details are presented. The problem statement and synthesis objectives are described first in section 5.1. Then, the proposed synthesis approach and methodology adopted to synthesize design is discussed in section 5.2. The implementation details of Cadence Conformal ECO flow are captured in section 5.3. The custom solutions required and developed for the partitioned synthesis flow are presented in section 5.4. Finally, the deliverables, i.e. EDA software information in terms of usage model, purpose, inputs and outputs are listed down in section 5.5.

5.1 Problem Statement and Objectives

The synthesis of M2A2 design deals with the mapping of standard cells present in ASIC design using standard cells placed in feedstock pre-placed design. Unlike conventional ASIC flow which makes use of cells present in standard cell library without any restriction on the number of times any cell can be used, the M2A2 synthesis performs exhaustive mapping of cells where the ASIC design can be synthesized only using the cells present in the feedstock pre-placed design. Thus, M2A2 synthesis is a more constrained problem. In a typical ASIC flow, placement of cells is performed after synthesis process. However, in M2A2 flow, the placement of cells is performed first. Then physical aware synthesis is performed to meet the functionality requirements and optimize design performance. Thus, it becomes a very challenging problem where functionality and optimization need to be considered, when the standard cells are already placed in the design.

The M2A2 design synthesis aims to make an optimal use of cells placed in the feedstock pre-placed design. The first and foremost objective is that the synthesis tool should be able to implement the functionality of ASIC design using the pre-placed cells in design. Then, the synthesizer tool should map the cells considering their locations in design. In other words, it should perform physical aware mapping of cells. This is of utmost importance because the design performance is highly dependent on how synthesizer is mapping the logic to these placed cells. The third objective is to ensure that all the desired optimization knobs of synthesizer tool are enabled in order to get the optimal synthesized results. It is desired that turnaround time in synthesizing the design is less, since higher TAT increases the NRE cost.

In order to synthesize the design with the constraints discussed above, Cadence Conformal ECO tool is used. The idea is based on functional post mask ECO approach [30], which is widely adopted in industry for fixing post silicon bugs and feature enhancement. Typically, functional ECO approach is adopted for fixing limited number of paths in design. However, the M2A2 synthesis flow makes use of this flow for synthesizing the complete design. This approach has few drawbacks, which will be explained later along with the possible ways to overcome them.

5.2 Proposed Synthesis Methodology

In this section, the synthesis methodology adopted to resolve the problem mentioned in previous section is discussed. The proposed synthesis methodology is based on functional post mask ECO approach. A high level overview of M2A2 synthesis methodology is shown in Figure 18. The functional post mask ECO synthesis is a process of improving or fixing the bugs in design after the design is implemented in silicon. In other words, synthesizing the design which is already taped out. The synthesis tool relies on the spare cells to implement additional functions or to correct the design functionality. It is a typical way of fixing post-silicon design bugs which is followed in the industry. However, the M2A2 main stream synthesis is based on this approach. The golden design in M2A2 comprises of feedstocks in the base layer. Each feedstock in golden design consists of sea of spare gates, with no connections among the cells. In other words, the golden design consists of standard cells, all of them are marked as spare cells. The spare cells have their input and output ports dangling. The ASIC design which needs to be implemented using feedstock cells is fed to Cadence Conformal ECO tool as the revised design. The Conformal ECO tool makes use of the golden design to form the base layer which is used to implement the functionality of the revised design. The timing constraints, physical libraries, and other setup files are also fed to Cadence Conformal ECO tool. These collaterals are already generated in ASIC design flow, and can be used without any modifications. Thus, synthesizing design using this approach ensures that exhaustive cell mapping is performed where ASIC design is implemented using standard cells instantiated in specific feedstock cells which are pre-placed in design.

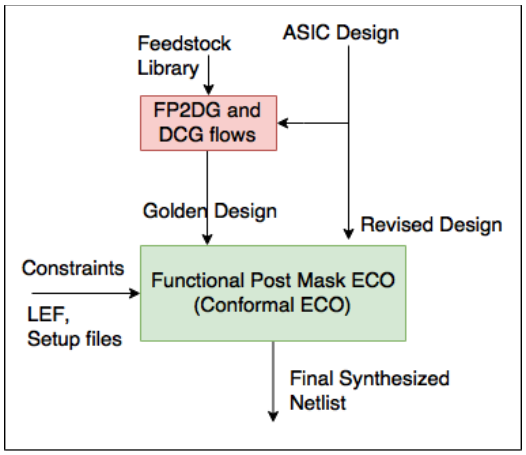


Figure 18: M2A2 Synthesis Methodology.

One of the main concerns in performing synthesis of a full design using this approach is that it might result in sub-optimal design performance. The ECO tool generates the patch file which captures the changes in the functionalities of the golden and revised design. In our case, the functionality of the revised design is implemented. Hence the size of patch file is dependent on revised design size. In case the design to be implemented comprises of large number of standard cells, the patch file size can be very large. This might result in failure of synthesis. Further, it might lead to sub-optimal design performance due to high critical path lengths. These are some of the problems which need to be resolved. The first problem related to patch file size can be resolved using Hierarchical Functional ECO flow. This flow is similar to flattened ECO flow, but generates multiple patch files, one for each partition in the hierarchy. This addresses the patch size and synthesis failure problem. However, hierarchical flow is not physical aware. The mapping of cells is based on greedy patch approach. The spare cells in patch 1 are first used to map the cells in design, and then patch 2 spare cells are used. The spare cell matching is not based on physical location of cells in revised design. Thus, it degrades the performance of design by elongating the length of critical paths, thereby adding significant interconnect delay.

In order to address both the problems, the partitioned synthesis flow is proposed in this work. This flow addresses both the patch file size and localized physical aware mapping of cells to partitions in order to ensure low interconnect delays. However, there is an overhead in terms of EDA flow development required for the partitioned synthesis flow, which is discussed in detail in section 5.4. The idea of partitioning is based on Cadence Common Power Format (CPF) flow [31]. In CPF flow, the design is divided into multiple partitions. Each partition is synthesized separately. However, CPF flow is used in industry to implement designs when multiple voltage domains/islands are used. The design is segmented into multiple partitions, each of which is powered by different voltage. The synthesis tool does the mapping of cells based on the respective design technology and physical libraries. However, this flow does not perform exhaustive cell mapping, and cannot be used directly to perform M2A2 synthesis. Hence, the best of both the worlds of Conformal ECO flow and CPF flow are taken to perform synthesis of M2A2 design, as shown in Figure 19.

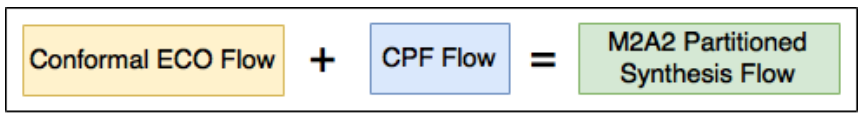


Figure 19: M2A2 Partitioned Synthesis Flow Overview.

5.3 ECO Conformal Flow Implementation

In this section, the implementation details of Cadence Conformal ECO flow are discussed. The subsection 5.3.1 deals with the overview of Conformal ECO flow. The physical aware Conformal ECO flow is discussed in subsection 5.3.2. The subsections 5.3.3, 5.3.4, and 5.3.5 discuss the flattened, hierarchical and partitioned synthesis flows respectively.

5.3.1 Overview

This sub-section deals with the Conformal ECO flow implementation details. The conventional post mask functional ECO flow is shown in Figure 20 [30]. According to Figure 20, G1 represents the golden design, i.e. the original design which is the feedstock pre-placed design in M2A2 design approach. The netlist and DEF files are required for this type of design, which is generated by DCG process. The RTL2 represents the customer ASIC design which needs to be implemented using M2A2 technology. The ASIC design implementation of RTL2 is performed to generate the G2 design netlist. This netlist is also referred to as revised design netlist. The Conformal ECO tool reads in both the revised and golden design collaterals, maps the key points which is basically mapping the primary input and output ports, and analyzes the design to generate

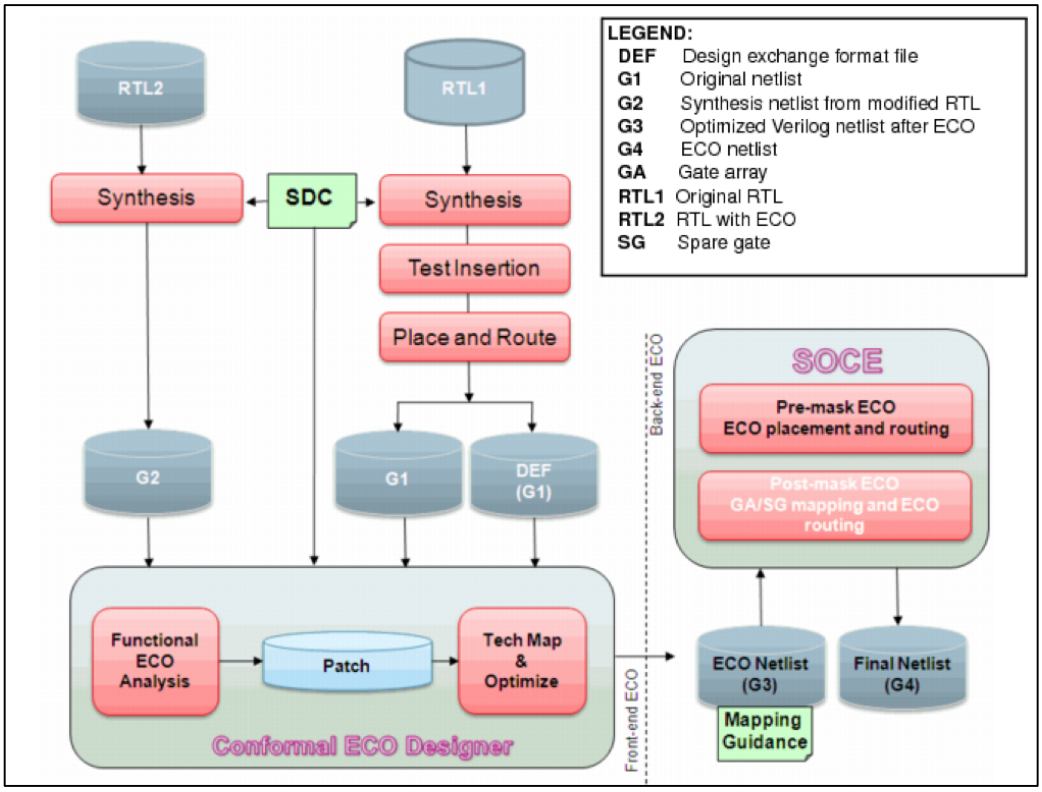


Figure 20: Post Mask Functional ECO flow [30].

a patch file. The patch file is generated based on the differences in functionality of both designs. The designer needs to specify the list of cells in golden design which can be used by ECO tool to implement functionality of revised design. These cells are also known as spare cells. The patch file generated by ECO tool is then fed to RTL Compiler, also known as Genus which is Cadence synthesis tool to perform synthesis.

The functional ECO flow also requires other collaterals in order to perform synthesis. The key points must exist in both the designs. Since M2A2 design approach makes use of this flow to map two different designs unlike conventional functional ECO approach where revised design is derivative of golden design, there is a need to ensure that the ports of revised design exist in golden design. This is done by developing custom scripts discussed in section 5.4. Next, all the physical and technology design libraries of standard cells instantiated in feedstock are required by tool in liberty (LIB) and library exchange format (LEF). The design constraints need to be specified in Synopsys design constraints (SDC) format. The SDC file lists down all the clock definitions, input and output timing delays, clock attributes such as uncertainty, etc.

As highlighted in section 5.2, it is highly recommended to perform physical aware synthesis. The traditional synthesis tools use wire-load models based on fanouts, which do not provide accurate wire delay information especially for designs where a significant portion of the delay is contributed by the wires like in deep submicron designs. In M2A2 design, the interconnect delay problem is more critical since the cells are already placed in design. In order to solve this problem, Physical Location Estimation (PLE) synthesis flow approach is adopted.

5.3.2 PLE ECO Synthesis Flow

"Physical Layout Estimation", shown in Figure 21 [32] is a physical modeling technique to capture timing closure P&R tool behavior that can be used during the RTL structuring process in synthesis process. This flow is developed by Cadence. It uses actual physical library info and capacitance tables that provide length-based capacit ance info, adjusts based on changing design sizes & structures, throughout optimization process.

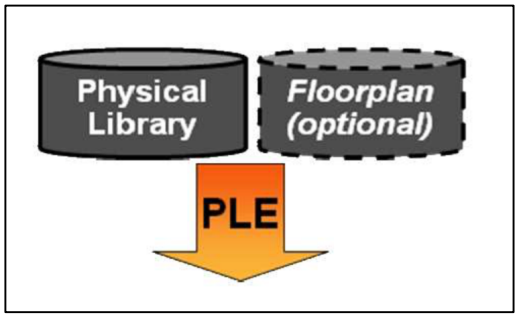


Figure 21: PLE Synthesis Flow [32].

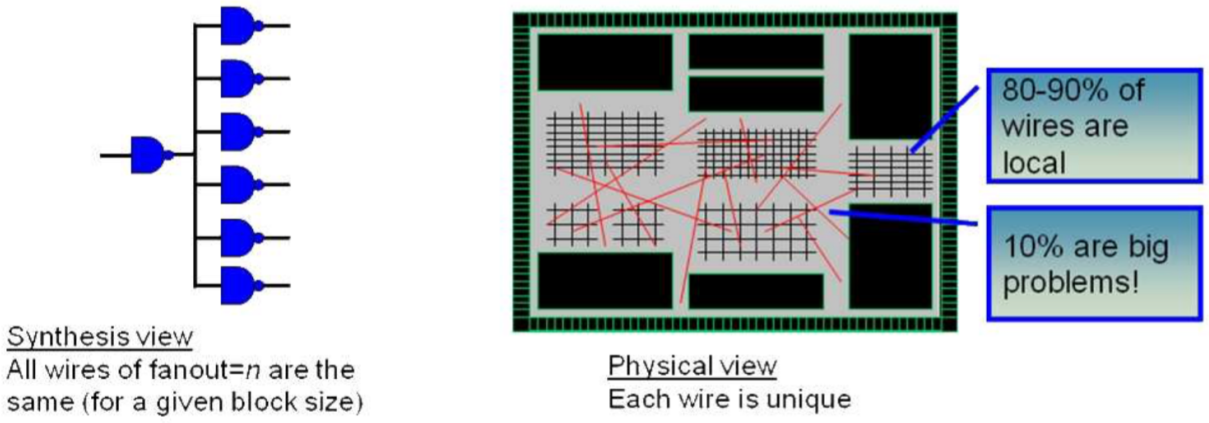


Figure 22: PLE aware synthesis [32].

Because it can dynamically adapt, it results in the best timing results. Since the design cells are already placed, it is aware of what the long wires will be in the design, and it does a good job in modeling the local interconnect, as shown in Figure 22 [32]. The PLE flow uses technology information and cell areas from the LEF libraries instead of using it from the synthesis technology libraries. The PLE flow uses parasitic resistance and capacitance values from the LEF libraries and the capacitance tables when estimating the wire lengths and the interconnect delays.

However, this approach is not used in this work to obtain the synthesis results, since the capacitance table for 32nm technology node is unavailable. This work has been listed in scope of future work. The partitioned PLE flow complement each other and it is expected that merging both flows will result in optimal interconnect delays.

5.3.3 Flattened Synthesis Flow

The flattened synthesis flow is same as the conventional functional ECO flow. It makes use of all those collaterals which are discussed in section 5.3.1. The merits of performing synthesis using this approach is that it is simple to implement, and no mapping issues are generally observed. However, the patch file size can be big which can result in long run times and sub-optimal results.

5.3.4 Hierarchical Synthesis Flow

In the hierarchical functional ECO synthesis flow (shown in Figure 23), multiple partitions are generated of a design [33]. With this approach, Conformal ECO tool uses the module boundaries to create a small patch. The static hierarchical compare is recommended.

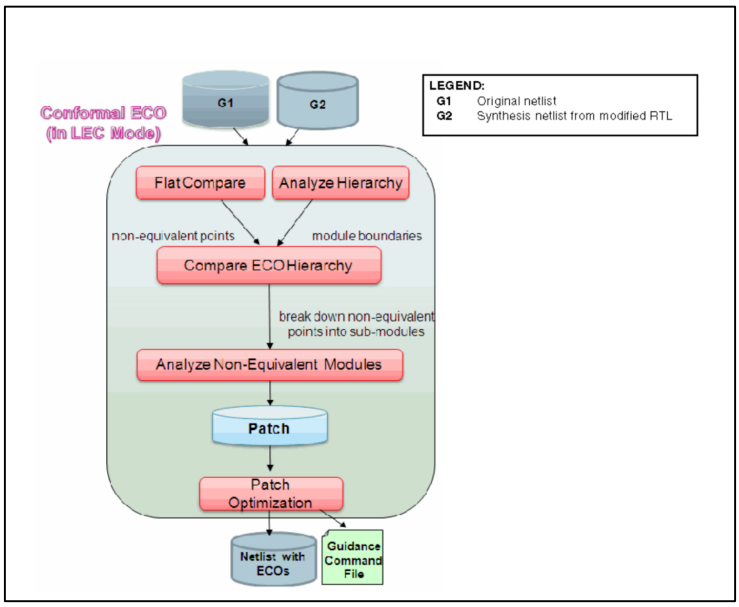


Figure 23: Hierarchical Functional ECO Synthesis Flow [33].

However, if there are false non-equivalences due to boundary optimization, phase inversions, or hierarchical clock gating, dynamic hierarchical compare should be used. Although hierarchical ECO flow requires additional setup, the runtime is faster and the patch is potentially smaller. However, hierarchical ECO flow does greedy patch optimization. The physical aware mapping of spare cells to revised cells design is not performed. The spare cells in patch 1 are exhausted first, and then spare cells of patch 2 are used to map cells lying anywhere in the design. This results in a very high interconnect delay. Hence this synthesis approach is not implemented.

5.3.5 Partitioned Synthesis Flow

This approach is similar to the flattened synthesis flow. In order to reduce the patch file size and improve localized physical aware mapping of cells, the design is segmented into multiple partitions. Each partition is fed to the flattened synthesis flow separately. Doing so ensures that patch file size is reduced. The partition is smaller in size in comparison to a full design, which also ensures more localized mapping of cells. Further, the synthesis runtime is reduced due to smaller patch file size. In cases where synthesis failure occurs due to missing spare cells, the synthesis won't be performed for the full design again. This will save turnaround time, which lowers down the NRE cost. However, this flow requires generation of collaterals for each partition. The section 5.4 discusses the collaterals generation process of partitioned synthesis flow. However, the constraints generation flow for partitioned synthesis flow is yet to be developed. But, the partitioned PLE synthesis flow is the recommended synthesis flow to achieve optimal results.

5.4 Partitioned Synthesis Flow Custom Scripts

In this section, the customizations for partitioned synthesis flow are discussed. As discussed in section 5.3, there is a need to perform synthesis for multiple partitions of a single design, rather than performing synthesis on a complete chip. This is required to ensure better physical aware synthesis of a design which is especially critical in M2A2 synthesis flow due to long interconnects. The partitioned synthesis flow is similar to flattened synthesis flow. In the partitioned synthesis flow, the design is segmented into multiple partitions and each partition is synthesized separately using flattened synthesis flow. Since, each partition is smaller in size in comparison to the complete design, the critical path lengths in design are expected to be less, which will ensure optimal performance of design by lowering down the interconnect delay.

As discussed in section 5.2, the Conformal ECO flow requires golden design netlist and DEF files, revised design netlist, list of spare cells, IO ports list for revised design, technology and physical libraries, capacitance table, constraints and other setup tables. To perform partitioned synthesis flow, all these collaterals are required. In flattened synthesis flow, all these collaterals are generated in FP2DG, DCG flows. In partitioned synthesis flow, there is a need to generate revised design netlist, golden design netlist and DEF, list of IO ports in revised design, spare cells list and design SDC constraints for each partition. The custom flow is developed to meet this requirement. Figure 24 shows the partitioned synthesis flow which generates the golden design partition DEF and netlist, revised partition netlist, IO ports and spare cells list. However, constraints generation for each partition using IO budgeting flow is still not developed. This work is listed as a scope of future work in this document. Thus, the partitioned synthesis flow is not deployed and synthesis results are based on flattened synthesis flow. The results obtained, discussed in chapter 6 show that the net length associated with critical paths is significantly higher (~6X to ~10X) in M2A2 design to that in ASIC design. This clearly highlights the need of partitioned synthesis flow.

The overview of partitioned synthesis flow is discussed below. The detailed explanation mentioning functionality, usage, input and output dependencies of EDA software for partitioned synthesis flow is mentioned in section 5.5.2. In order to generate the revised design netlist, first the netlist is flattened. This removes all the hierarchies in the design and all instances are defined in top level. Next, the flattened design is segmented into multiple partitions, which are connected at top level to ensure that there is no change in the functionality. This two-step process makes use

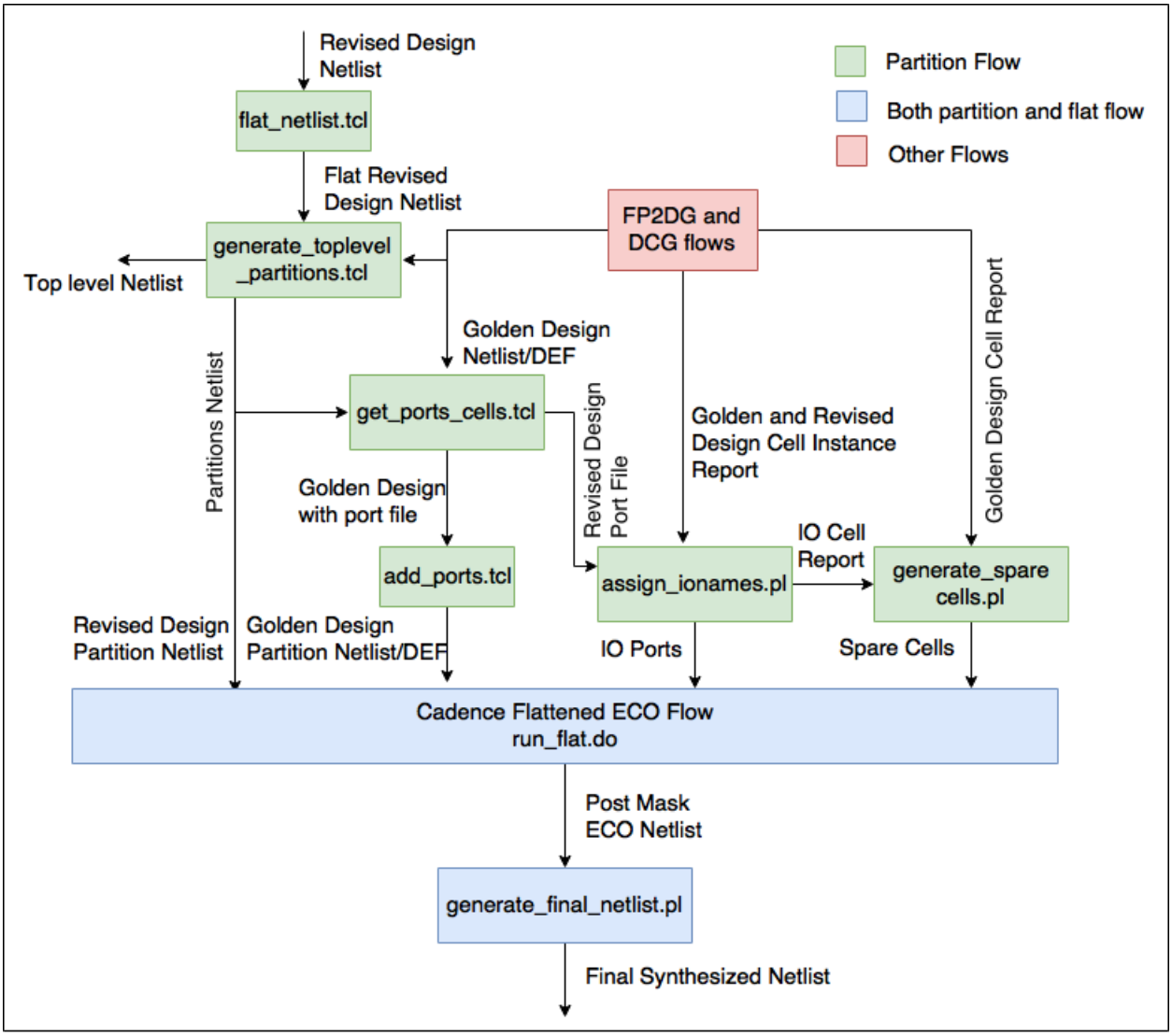


Figure 24: Partitioned Synthesis Customization Flow.

of *flat_netlist.tcl* and *generate_toplevel_partitions.pl* scripts to generate netlist files for all the partitions in design. The instance list of each partition is generated in FP2DG process. Based on this information, instances are grouped to generate the partition and top level module netlist files. The partition netlist generated is fed to Conformal ECO flow as the revised design netlist. The list of ports in partition netlist is given as input to *get_ports_cells.tcl* script, which reads the golden design partition collaterals generated by FP2DG and DCG processes to generate golden design netlist and DEF files. These files contain the ports of the revised design. This is fed to the Conformal ECO tool as the revised design collaterals. In order to perform physical aware port assignment of partitions, revised design port file is fed to *assign_ionames.pl* script to generate IO port list of golden design which lists down 1-1 mapping of ports in revised and golden design

partitions. The spare cells are also generated based on this list. The spare cell list in golden design consists of all the cells except the cells connected to input/output ports. This list is generated by *generate_sparecell.pl* script, which takes golden design information from DCG flow. Thus, all the collaterals except SDC constraints are generated for partitioned synthesis flow. The constraints generation requires IO budgeting flow where input and output delay budgets need to be calculated accurately for all the partitions. The detailed explanation of EDA software developed for performing partitioned synthesis is discussed in the next section.

5.5 Deliverables

In this section, the EDA software in form of dofiles, perl and Tcl scripts/utilities developed in flattened and partitioned synthesis flow is mentioned. The usage model, paths, inputs, outputs and other important information is also captured for better understanding of the flow to the reader.

5.5.1 Synthesized Netlist generation

This sub-section describes the dofiles and synthesized netlist generation process.

5.5.1.1 Script #1: run_flat.do

- ***Function:***

The script is sourced in Conformal ECO to perform flattened functional ECO synthesis of M2A2 design. The script generates the CFM file in the Conformal ECO tool which gets invoked on the fly in RTL Compiler to perform synthesis. The RTL Compiler generates netlist file of the patch which captures logical connections information among cells pre-placed in feedstock design to implement the ASIC design functionality. This is the main script which performs flattened post mask functional ECO synthesis.

Usage:

dofile /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/dofiles/run_flat.do in Conformal ECO shell

- ***Inputs:***

1. The user needs to specify technology library, LEF library, golden design DEF and netlist, revised design netlist, capacitance table, and spare cells list to perform flattened post mask functional ECO synthesis.

- ***Outputs:***

1. swapcells.tcl

The utility dumps out swap cells tcl file which contains ecoSwapSpareCell commands for all the instances in revised design which will be replaced by golden design cell names. Since, all the cells are defined as spare cells in the golden design, this file captures the list of cells which are used to implement the functionality. The remaining cells are still marked as spare cells.

2. ${design_name}.postECO.v

The utility also generates the netlist file of the patch. The patch captures the incremental changes done in the golden design in order to implement the functionality of the revised design. This file gets sourced in Encounter/Innovus tool along with *swapcells.tcl* file to generate the final synthesized design netlist.

5.5.1.2 Script #2: run_hier.do

- ***Function:***

The script is sourced in Conformal ECO to perform hierarchical functional post mask ECO synthesis of M2A2 design. The script generates the CFM file in the Conformal ECO tool which gets invoked on the fly in RTL Compiler to perform synthesis. The RTL Compiler generates netlist files for multiple patches, one for each module in hierarchy, each captures the logical connections information among cells pre-placed in feedstock design to implement the ASIC design functionality. This is the main script which performs hierarchical functional post mask ECO synthesis. The synthesis is performed using greedy patch approach, i.e. the cells in different modules won't be mapped to spare cells of their patches based on their locations in design, but all the spare cells in patch 1 will be used first. This can degrade the performance of the synthesized design.

Usage:

dofile /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/dofiles/run_hier.do in Conformal ECO shell

- ***Inputs:***

1. The user needs to specify technology library, LEF library, golden design DEF and netlist, revised design netlist, capacitance table, and spare cells list to perform hierarchical post mask functional ECO synthesis.

- ***Outputs:***

1. swapcells.tcl

The utility dumps out swap cells tcl file which contains ecoSwapSpareCell commands for all the instances in revised design which will be replaced by golden design cell names to implement the functionality. Since, all the cells are defined as spare cells in the golden design, this file captures the list of cells which are used to implement the functionality.

2. ${design_name}.postECO.v

The utility generates the netlist file for all the patches. All the patches in design when merged together capture the incremental changes done in golden design in order to implement the functionality of the revised design. This file gets sourced in Encounter/Innovus tool along with *swapcells.tcl* file to generate the final synthesized design netlist.

5.5.1.3 Script #3: generate_final_netlist.tcl

- ***Function:***

The script is sourced in Encounter/Innovus shell to generate the final synthesized netlist. The post mask ECO netlist generated by RTL compiler contains the cell instances from revised design, which need to be replaced by golden design instance names. This script requires post ECO

netlist file and *swapcells.tcl* file generated by RTL compiler to generate synthesized netlist, when sourced in Encounter/Innovus shell. This is the final output of the synthesis flow to the P&R flow.

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/generate_final_netlist.tcl in Encounter/Innovus shell

- ***Inputs:***

1. The user needs to specify technology LEF library, post ECO netlist, spare cells list and swap cells tcl file generated by RTL compiler, to generate final synthesized functional ECO netlist.

- ***Outputs:***

1. ${design_name}.v

This is the final synthesized netlist which implements the ASIC design functionality using feedstocks preplaced in design. This netlist file is fed to backend design flow to perform clock tree synthesis, routing and backend design optimizations.

5.5.2 Partitioned Synthesis Flow Custom scripts

In this subsection, the details on custom scripts for partitioned synthesis flow are discussed.

5.5.2.1 Script #4: flat_netlist.tcl

- ***Function:***

The script is sourced in RTL Compiler tool to generate flattened version of netlist from the input design netlist. This script is required in partitioned and flattened synthesis flow. In both

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/flat_netlist.tcl in RTL Compiler shell

flattened and partitioned synthesis flow, it is used to generate the flattened version of design netlist. In partitioned flow, it is further used to generate netlist files for different partitions from the flattened version of design netlist.

- ***Inputs:***

1. The user needs to specify technology library, and the design netlist which needs to be flattened.

- ***Outputs:***

1. ${design_name}.flat.v

The utility generates the flattened version of the design netlist. This is the output file which will be fed to the Conformal ECO flow as revised netlist, and fed to *generate_toplevel_partitions_netlist.tcl* in partitioned synthesis flow to generate netlist files for partitions.

5.5.2.2 Script #5: generate_toplevel_partitions_netlist.tcl

- ***Function:***

The script is sourced in RTL Compiler tool to generate hierarchical netlist from the input design netlist. This script is sourced to generate hierarchical version of input design netlist, which is required in partitioned synthesis flow. It generates the top level netlist, netlist files for each partition and complete netlist file which has top level and partitions instantiated in design.

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/generate_toplevel_partitions.tcl in RTL Compiler shell

- ***Inputs:***

1. The user needs to specify technology library, design netlist and instance partition report to generate netlist files for the partitions. The instance partition report is generated in the FP2DG process, which captures the instance names of all the cells which belong to the particular partition in design.

- ***Outputs:***

1. ${design_name}.v

The utility generates the hierarchical version of input design netlist. This netlist file has top and partitions module definitions instantiated in this netlist file.

2. ${design_name}.top.v

The utility also generates the top level netlist of the design. This file includes only the top level of design. It includes the instantiations to all the partitions in the design, but partitions modules are not included in this file.

3. ${design_name}.${partition}.v

The utility also generates the netlist file for each partition. Multiple netlist files are generated, one for each partition. This file includes the module definition for each partition. This file will be fed as the revised design netlist in partitioned synthesis flow where each partition is synthesized separately and then stitched back at top level.

5.5.2.3 Script #6: add_ports.tcl

- ***Function:***

The script is sourced in RTL Compiler tool to add top level ports in design netlist. In the Conformal ECO flow, it is required to map the names of primary inputs and outputs of golden and revised design. In order to ensure that mapping can be performed, the revised design netlist ports are added to the golden design netlist using this utility. The utility reads the input design netlist and port list file, and adds all the ports in the top level.

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/add_ports.tcl in RTL Compiler shell

- ***Inputs:***

1. The user needs to specify the technology library, golden design netlist and IO cell report to add ports to golden netlist file. The IO cell report is generated by *get_cells_reports.tcl* script, which captures the primary input and output port names as well as cells associated with these ports.

- ***Outputs:***

1. ${design_name}.v

The utility generates the golden design netlist, with additional ports at top level specified by user in IO cell report.

5.5.2.4 Script #7: get_ports_cells.tcl

- ***Function:***

The script is sourced in RTL Compiler tool to generate an IO cell report for the revised design which is fed to *add_ports.tcl* script to add ports in the golden design netlist. The utility generates the list of primary input, output and inout ports in input design netlist and list of associated cells connected with these ports.

Usage:

source /home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis /scripts/get_ports_cells.tcl in RTL Compiler shell

- ***Inputs:***

1. The user needs to specify the technology library, and revised design netlist to generate IO cell report.

- ***Outputs:***

1. ${design_name}.iocell.rpt

The utility generates the IO cell report, which captures the primary input and output port names as well as cells associated with these ports, for input revised netlist file. This file is used as an input file in *add_ports.tcl*, *assign_ionames.pl* and *generate_sparecells.pl*.

5.5.2.5 Script #8: assign_ionames.pl

- ***Function:***

This purpose of this script is to generate *ports.do* file for each partition. In order to map the primary input and output ports of golden and revised design, the port list of golden design needs to be generated. This ensures physical aware mapping of ports in design.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis/scripts/assign_ionames.pl

-golden_design_file	*<Golden Design File>*	*(REQUIRED)*
-revised_design_iocells_file	*<Revised Design IO file>*	*(REQUIRED)*
-revised_design_placement_file	*<Revised Design Placement file>*	*(REQUIRED)*
-feedstock_cell	*<Feedstock instance name>*	*(REQUIRED)*

- ***Inputs:***

1. -golden_design_file <Golden Design File>

This file is generated by DEF generation script *generate_place_cells_tcl.pl* script. This file contains information regarding the golden design standard cell instance names, cell location, pins, standard cell library names for all the cells instantiated in golden design.

2. –revised_design_iocells _file <Revised Design IO file >

This file is generated by *get_ports_cells.tcl* utility. This file captures the primary input and output port names as well as cells associated with these ports, for input revised netlist file.

3. –revised_design_placement _file <Revised Design Placement file>

The file is generated by FP2DG script. It lists down the location of all the cells instantiated in the revised design.

4. –feedstock_cell <Feedstock Instance name>

The feedstock instance name of the partition to which IO cells are assigned, is mentioned in this switch value.

- ***Outputs:***

1. ${partition}.ports.do

The utility generates the *ports.do* file which is sourced in *run_flat.do* file to perform the flattened post mask ECO synthesis of a partition. This file lists down the port mapping information for each partition, i.e. mapping correlation between ports present in golden and revised design.

2. ${partition}.iocells.rpt

The utility generates the report which captures the list of cells associated with primary input, and output ports. This report is fed to *generate_sparecells.pl* script to generate the list of spare cells for each partition.

5.5.2.6 Script #9: generate_sparecells.pl

- ***Function:***

The script is used to generate list of spare cells for each partition in golden design. In order to perform ECO synthesis of each partition, list of spare cells need to be generated which gets sourced in *run_flat.do* file to perform ECO synthesis of each partition.

Usage:

/home/projects/courses/spring_16/ee382m-16745/feedstock/aseem_scripts/synthesis/scripts/generate_sparecells.pl

-feedstock_cellmap_file	*<Feedstock cell map file>*	*(REQUIRED)*
-golden_io_cell_rpt	*<Golden netlist IO cell ports file>*	*(REQUIRED)*
-feedstock_cell	*<Feedstock instance name>*	*(REQUIRED)*

- ***Inputs:***

1. –feedstock_cellmap_file <Feedstock Cell Map File>

This file is generated by netlist generation script *generate_netlist.pl* script. It lists down the libcell name and corresponding instance names, pin names for all the cells for each instance of feedstock.

2. –golden_io_cell_rpt <Golden Netlist IO cell ports file >

This file is generated by *assign_ionames.pl* utility. This file captures the list of cells which are connected to primary input and output ports of the golden design partition.

3. –feedstock_cell <Feedstock Instance name>

The feedstock instance name of the partition, which is used to provide spare cells in order to perform mapping of revised design, is mentioned in this switch value.

- ***Outputs:***

1. ${partition}.sparecells.do

The utility generates the spare cells dofile which is sourced in *run_flat.do* file to perform the flattened post mask ECO synthesis of partition. This file lists down all the spare cells present in golden design partition which are used to implement functionality of revised design partition.

Chapter 6: Performance Evaluation

In this chapter, the performance evaluation of M2A2 circuits is discussed. The FLG, FP2DG and synthesis results for different feedstock library constraints and feedstock sizes are presented. The ASIC implementation results for all 3 design corners – high performance, balanced performance and low power are discussed in section 6.1. In the Specific Design Library (SDL) library approach, the feedstock library is generated based on a specific design, and then synthesis is performed using the same design. The synthesis results for designs using M2A2 SDL approach are described in section 6.2. To generalize the library and reduce number of overall feedstocks in library to synthesize all the designs, the common design library (CDL) approach is adopted. In this approach, library is trained using all the designs, and this common library is fed to the synthesizer. The synthesis results for M2A2 CDL approach are captured in section 6.3. In order to further constrain the library, different design library (DDL) approach is adopted. In this approach, the library is designed using some set of designs and different design is synthesized using this library. The synthesis results for M2A2 DDL approach are mentioned in section 6.4. The performance of M2A2 SDL and M2A2 CDL is compared and presented in section 6.5. Similarly, the M2A2 DDL and M2A2 CDL performance is compared and discussed in section 6.6. The synthesis results for different number of feedstocks in M2A2 CDL approach are also evaluated, and mentioned in section 6.7. The performance of M2A2 CDL design is compared with ASIC and FPGA designs and described in sections 6.8 and 6.9 respectively. Finally, the key inferences drawn from all the synthesis experiments are captured in section 6.10.

6.1 ASIC Standard Cell Results

The standard cell ASIC design implementation is based on 32nm technology node. The design uses ARM 32nm standard cell library. The standard cell ASIC implementation is performed on 3 different corners – high performance, balanced power and performance, and low power. The Synopsys Design Compiler (DC) and Synopsys IC Compiler (ICC) tools are used for the synthesis and backend physical design respectively. The quality of results (QoR) of ASIC design for all 3 corners is evaluated in terms of die area, power consumption, critical timing path delay, critical timing path length, and number of logic levels in critical path. The critical path is the path which consumes the most time in the design, and decides its maximum operating frequency. The QoR of ASIC design for all corners is shown in Table 3. The design floorplans are shown in Figure 25.

QoR	High Performance	Balanced Performance	Low Power
Die Rectilinear area	1.02 mm^2	1.07 mm^2	1.25 mm^2
Die Minimum area	0.99 mm^2	0.96 mm^2	0.89 mm^2
Total Power Consumption	218.4 mW	27.2 mW	11.8 mW
Critical Timing Path Delay	1.25 ns	2.74 ns	4.94 ns
Critical timing path length	785.83 μm	1068.3 μm	772.1 μm
#Logic levels in critical path	45	55	60

Table 3: ASIC Standard Cell Design QoR.

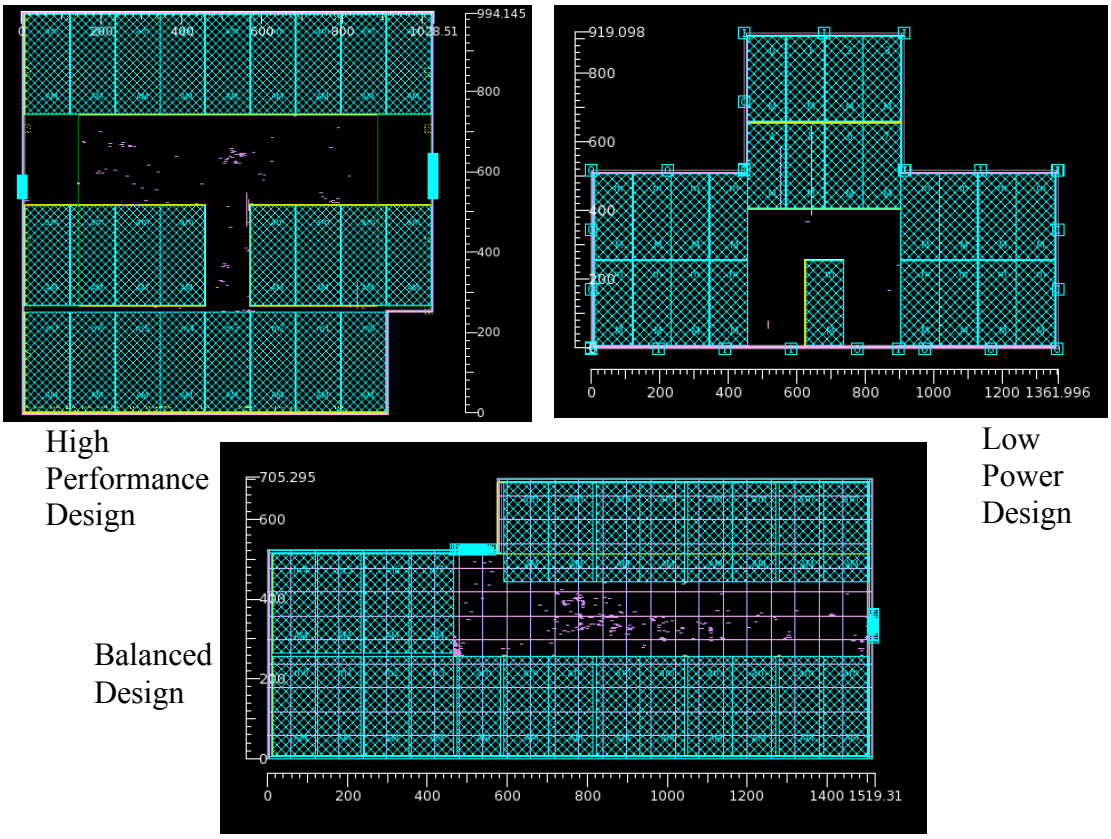


Figure 25: ASIC Implementation Design Floorplans.

6.2 M2A2 SDL Results

This section deals with the performance evaluation of designs implemented based on M2A2 SDL approach. In Specific Design Library (SDL) approach, the feedstock library is generated based on a specific design, and then this library is used to synthesize the same design. Thus, library is generated separately for all 3 different design corners, as shown in Figure 26.

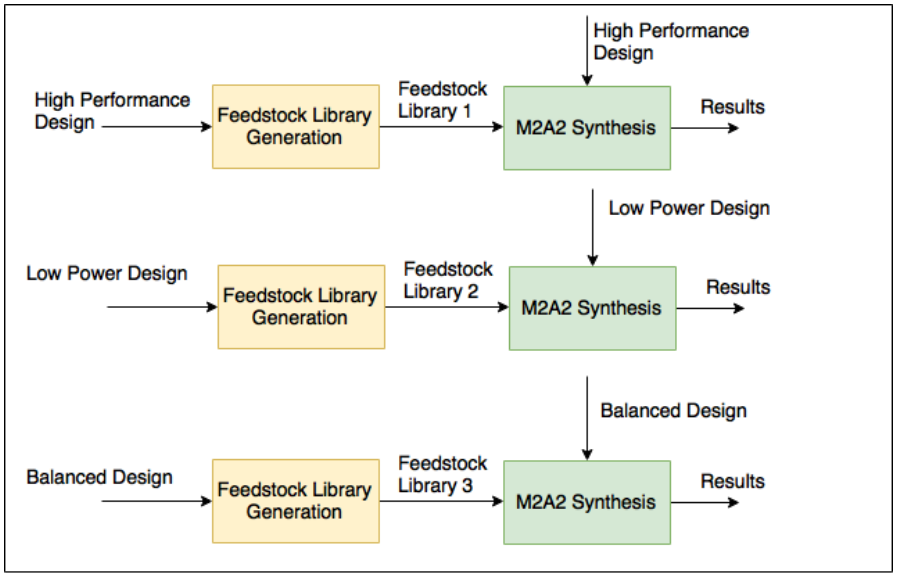


Figure 26: M2A2 SDL Approach.

This is certainly not a good approach to synthesize the design, since this approach leads to high number of feedstocks in the library. However, this problem is not trivial. For example, there are ~300 types of standard cells instantiated ~50000 times in a high performance design. Thus the total number of combinations to place the 300 types of cells at 50000 different locations is 300^{50000}. Out of all these possible combinations, an optimal combination is chosen. The SDL approach is implemented for two different feedstock sizes, i.e. for 50 μm * 50.86 μm and 71 μm * 71.86 μm. It makes use of 5 and 3 feedstock cells respectively, and are called SDL50 and SDL70 approaches. The heights 50.86 μm and 71.86 μm are non-integer numbers, because they are integral multiples of standard cell height. The SDL70 feedstock size is twice the SDL50 feedstock size.

The Feedstock Library Generation (FLG) metrics are analyzed for all the design corners using SDL50 approach, and are shown in Table 4. The FLG metrics analyzed include feedstock size, #feedstocks, average number of times each feedstock is used, mapping similarity percentage, mapped coverage, standard cell utilization in feedstocks and average deterministic placement of cells percentage for each feedstock. All these metrics are explained in section 2.7.

The feedstock pre-placed design generation (FP2DG) metrics are also analyzed for this design approach, and are shown in Table 5. The metrics include standard cell count in design, cell count and area ratio with ASIC design approach, mapping distribution of cells within feedstock site, neighbor feedstock site, group, neighbor group and overall design, and unmapped percentage

of cells in the design. These metrics are explained in section 3.6.

QoR	High Performance	Balanced Performance	Low Power
Feedstock Size (μm* μm)	50*50.16	50*50.16	50*50.16
#Feedstocks in library	5	5	5
Feedstock Avg. Repetition (FAR)	~24	18	~14
Mapping Similarity (%)	~86	~80	~84
Design cells Mapped Coverage (%)	~85	~82	~80
Standard Cell Area Utilization (%)	78 – 93	92 – 94	78 – 93
Deterministic placement of cells (%)	~87	~88	~90

Table 4: FLG Analysis for M2A2 SDL50 Designs.

QoR	High Performance	Balanced Performance	Low Power
Standard cell count in M2A2 design	99183	70580	58749
#Feedstocks placed in design	125	90	80
Standard cell area ratio (M2A2 vs. ASIC)	1.19	1.21	1.33
Standard cell count ratio (M2A2 vs. ASIC)	1.92	1.51	1.55
Mapping of cells within feedstock (%)	80.80	80.80	80.30
Mapping of cells with neighbor feedstock (%)	8.30	9.80	9.30
Mapping of cells within group (%)	8.10	4.90	8.80
Mapping of cells with neighbor group (%)	0	0	0
Mapping of cells in overall design (%)	2.80	4.50	1.60
Unmapped cells in design (%)	0	0	0

Table 5: FP2DG Analysis for SDL50 Designs.

QoR	High Performance	Balanced Performance	Low Power
Die Rectilinear area	1.12 mm^2	1.12 mm^2	1.40 mm^2
Die Minimum area	1.08 mm^2	1.02 mm^2	0.94 mm^2
Critical Timing Path Delay	1.25 ns	2.82 ns	4.80 ns
Critical timing path length	7672 μm	8637 μm	9370 μm
#Logic levels in critical path	37	53	56

Table 6: M2A2 SDL50 Synthesis Results.

Table 6 lists down the synthesis results for all the design corners implemented using M2A2 SDL50 design approach. The die rectilinear and minimum area, critical timing path delay, length and logic levels are evaluated to get the timing and area metrics for all the design corners. The floorplans of M2A2 SDL50 designs are shown in Figure 27. The timing and area metrics of M2A2 SDL50 results are compared with ASIC standard cell QoR metrics. The die rectilinear and minimum area of both M2A2 and ASIC designs are compared, and whichever area ratio gives higher (worse) number, is reported. The critical timing path delay is also evaluated at synthesis stage, which does not consider the interconnect delay. In order to account for the interconnect delay, the critical path length, and #logic levels are evaluated for the M2A2 designs. The critical path length ratio of M2A2 vs. ASIC design, along with additional interconnect length is reported. The interconnect delay is estimated using delay of 300ps/mm. The expected interconnect delay is added to the synthesis timing to get the final delay for M2A2 circuits. The critical timing path determines the circuit speed. This timing delay of M2A2 SDL design is compared with that of the ASIC design to get the circuit speed ratio for both the implementation techniques. Table 7 and Table 8 list down the performance comparison of SDL50 and SDL70 designs with ASIC design respectively.

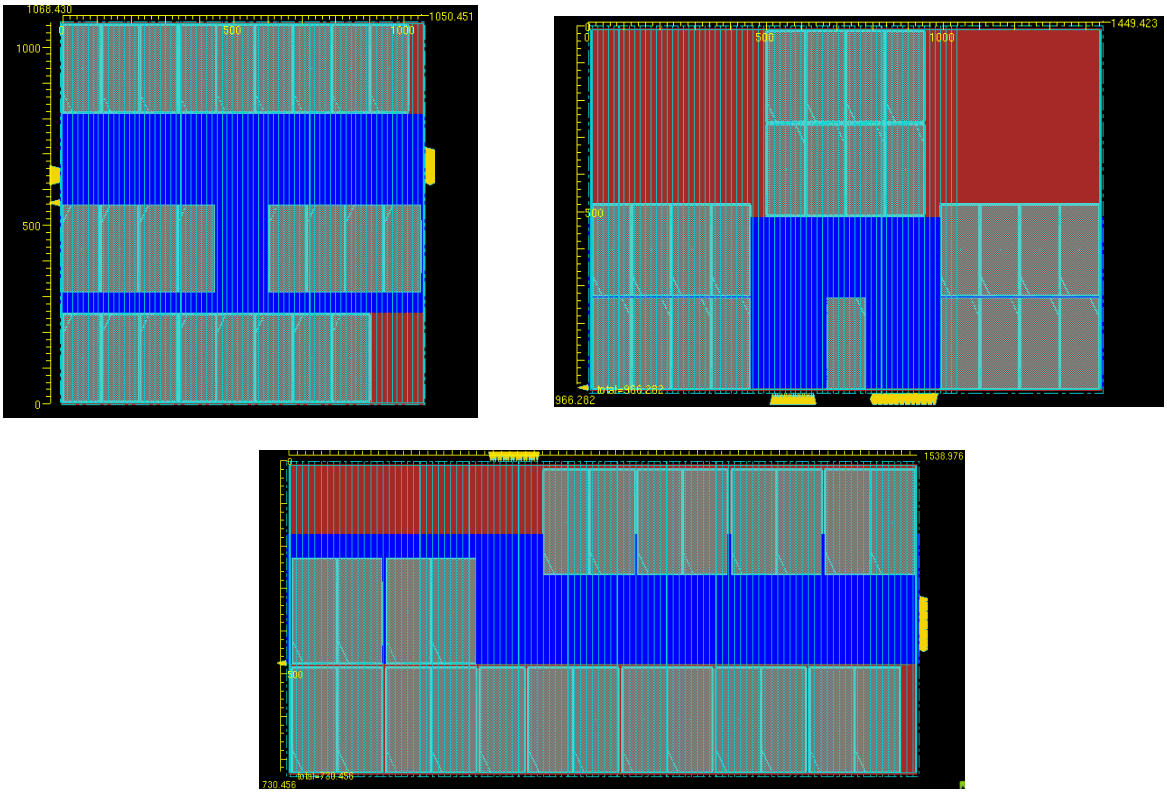


Figure 27: M2A2 SDL50 Design Floorplans.

QoR	High Performance	Balanced Performance	Low Power
Worst area ratio (M2A2 vs. ASIC)	1.10	1.06	1.12
Synthesis critical path delay ratio	1.00	1.03	0.97
Critical path length ratio	9.76	8.08	12.13
Additional critical path length (μm)	6886	7569	8600
Additional interconnect delay* (ns)	2.06	2.27	2.58
Expected P&R critical path delay (ns)	3.31	5.09	7.38
Timing Delay ratio (M2A2 vs. ASIC)	2.60	1.80	1.50

Table 7: M2A2 SDL50 vs. ASIC Synthesis Results.

QoR	High Performance	Balanced Performance	Low Power
Worst area ratio (M2A2 vs. ASIC)	1.18	1.08	1.28
Synthesis critical path delay ratio	1.00	1.02	0.98
Critical path length ratio	6.50	6.18	9.74
Additional critical path length (μm)	4232	5543	6753
Additional interconnect delay* (ns)	1.27	1.66	2.02
Expected P&R critical path delay (ns)	2.52	4.45	6.89
Timing Delay ratio (M2A2 vs. ASIC)	2.02	1.62	1.39

Table 8: M2A2 SDL70 vs. ASIC Synthesis Results.

The synthesis results for M2A2 SDL50 are also compared with SDL70 results, and shown in Table 9. As the feedstock size increases, the number of feedstocks in design decreases. However, the die area increases. Thus, there is a cost tradeoff between silicon die area and #feedstocks.

QoR	SDL50	SDL70
Feedstock Size (μm * μm)	50*50.16	71*71.86
#Feedstock cells used	5	3
Worst area ratio (M2A2 vs. ASIC)	1.06 – 1.12	1.08 – 1.28
Synthesis critical path delay ratio	0.97 – 1.03	0.98 – 1.02
Critical path length ratio	7.68 – 12.13	6.18 – 9.74
Timing Delay ratio (M2A2 vs. ASIC)	1.50 – 2.60	1.39 – 2.02

Table 9: M2A2 SDL50 vs. M2A2 SDL70 Synthesis Results.

6.3 M2A2 CDL Results

This section deals with the performance evaluation of designs implemented based on M2A2 CDL design approach. In Common Design Library (CDL) approach, the feedstock library is generated based on all the reference designs, and then this library is used to synthesize each design separately. Thus, a common library is generated for all 3 different design corners, and fed to the synthesizer tool to perform synthesis of design, as shown in Figure 28.

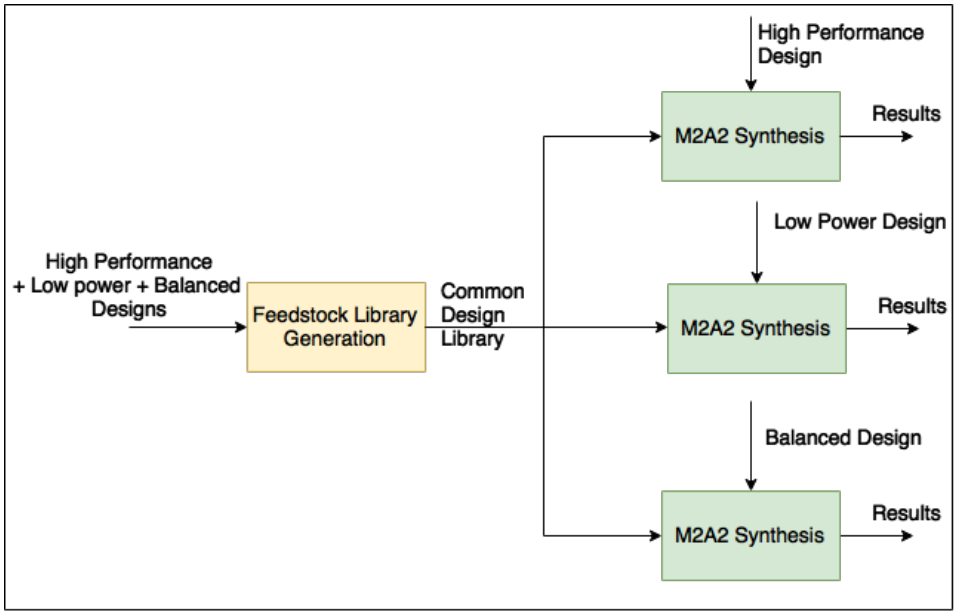


Figure 28: M2A2 CDL Approach.

Unlike SDL approach, CDL approach uses common feedstocks across all the designs, thus limits the count of feedstocks in library. However, CDL is a more challenging problem than SDL, since number of cells in all reference designs are added which increases the possible combinations. The CDL approach is also implemented for the same two feedstock sizes, i.e. for 50 μm * 50.86 μm and 71 μm * 71.86 μm. It makes use of 5 and 3 feedstocks respectively, and are called CDL50 and CDL70 approaches. The FLG, and FP2DG analysis is also performed for M2A2 CDL designs. The synthesis performance in terms of area and circuit speed metrics is also evaluated, and then compared with ASIC design. This methodology is similar to what author has adopted in SDL design approach. The performance comparison of the M2A2 CDL designs with the ASIC designs is shown in Table 10. Similarly the performance comparison of M2A2 CDL70 designs with ASIC designs is made, as shown in Table 11.

QoR	High Performance	Balanced Performance	Low Power
Worst area ratio (M2A2 vs. ASIC)	1.10	1.10	1.12
Synthesis critical path delay ratio	1.00	0.96	0.88
Critical path length ratio	10.12	6.98	10.14
Additional critical path length (μm)	7160	6389	7060
Additional interconnect delay* (ns)	2.14	1.91	2.12
Expected P&R critical path delay (ns)	3.39	4.54	6.50
Timing Delay ratio (M2A2 vs. ASIC)	2.71	1.65	1.31

Table 10: M2A2 CDL50 vs. ASIC Synthesis Results.

QoR	High Performance	Balanced Performance	Low Power
Worst area ratio (M2A2 vs. ASIC)	1.18	1.08	1.28
Synthesis critical path delay ratio	1.00	0.99	0.91
Critical path length ratio	11.30	7.82	11.24
Additional critical path length (μm)	8104	7287	7912
Additional interconnect delay* (ns)	2.43	2.18	2.37
Expected P&R critical path delay (ns)	3.68	4.89	6.89
Timing Delay ratio (M2A2 vs. ASIC)	2.94	1.78	1.39

Table 11: M2A2 CDL70 vs. ASIC Synthesis Results.

The synthesis results for M2A2 CDL50 are also compared with CDL70 results, and shown in Table 12. As the feedstock size increases, number of feedstocks in design decreases. However, the die area increases. Thus, there is a cost tradeoff between silicon die area and #feedstocks.

QoR	CDL50	CDL70
Feedstock Size (μm * μm)	50*50.16	71*71.86
#Feedstock cells used	5	3
Worst area ratio (M2A2 vs. ASIC)	1.10 – 1.12	1.08 – 1.28
Synthesis critical path delay ratio	0.88 – 1.00	0.91 – 1.00
Critical path length ratio	6.98 – 10.14	7.82 – 11.24
Timing Delay ratio (M2A2 vs. ASIC)	1.31 – 2.71	1.39 – 2.94

Table 12: M2A2 CDL50 vs. M2A2 CDL70 Synthesis Results.

6.4 M2A2 DDL Results

This section deals with the performance evaluation of designs implemented based on M2A2 DDL design approach. In Different Design Library (DDL) approach, the feedstock library is generated based on all reference designs except the design which needs to be synthesized, and then this library is used to synthesize the target design. Thus, DDL approach tests the generalization of feedstocks present in design library, which can be used to synthesize different designs. The DDL approach is also implemented for the same two feedstock sizes, i.e. for 50 μm * 50.86 μm and 71 μm * 71.86 μm. It makes use of 5 and 3 feedstocks respectively, and are called DDL50 and DDL70 approaches respectively.

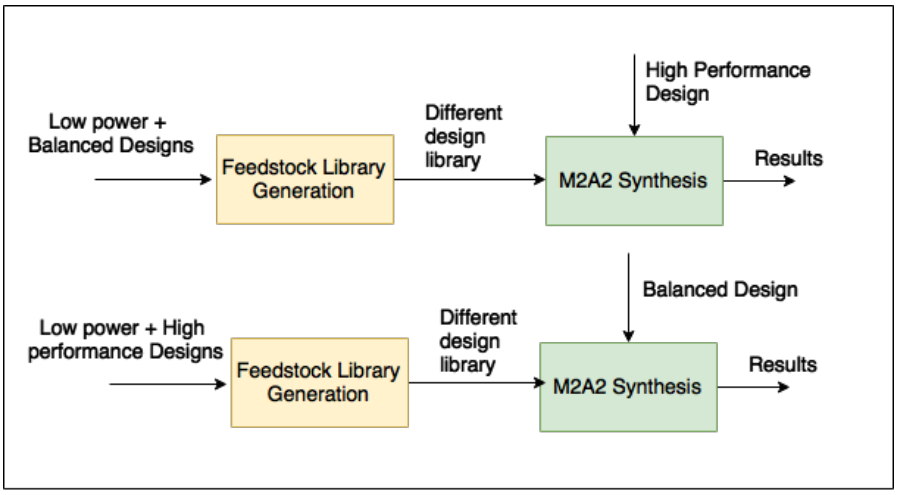


Figure 29: M2A2 DDL Approach.

The M2A2 DDL50 implementation is performed for High performance design whereas M2A2 DDL70 implementation is done for balanced performance design. The synthesis performance in terms of area and circuit speed metrics is also evaluated, and then compared with the ASIC design. The performance comparison of M2A2 DDL design with ASIC design implementation for both designs is shown in Table 13. The synthesis timing results for M2A2 DDL50 High performance design got degraded by 48%. This is due to the fact that feedstock library was designed based on balanced and low power design corners. The feedstocks designed using FLG algorithm made the maximum use of HVT and RVT cells, due to which the timing of critical path got degraded. This also proves that FLG algorithm is learning appropriately from the reference designs. This approach has shown worst results, since different corners were used in generating the feedstock library and synthesizing design.

QoR	High Performance DDL50	Balanced Performance DDL70
Worst area ratio (M2A2 vs. ASIC)	1.10	1.08
Synthesis critical path delay ratio	1.48	0.96
Critical path length ratio	14.78	8.23
Additional critical path length (μm)	10821	7721
Additional interconnect delay* (ns)	3.24	2.31
Expected P&R critical path delay (ns)	5.09	4.94
Timing Delay ratio (M2A2 vs. ASIC)	4.07	1.80

Table 13: M2A2 DDL50/70 vs. ASIC Synthesis Results.

6.5 Comparison of M2A2 SDL and M2A2 CDL Results

In this section, the M2A2 results obtained for all the design corners using SDL and CDL approach are compared. Table 14 and Table 15 lists down the comparison of SDL50 vs. CDL50 and SDL70 vs. CDL70 design performance.

QoR	SDL50	CDL50
Feedstock Size (μm * μm)	50*50.16	50*50.16
#Feedstock cells used	5(*3=15)	5(*1=5)
Worst area ratio (M2A2 vs. ASIC)	1.06 – 1.12	1.10 – 1.12
Synthesis critical path delay ratio	0.97 – 1.03	0.88 – 1.00
Critical path length ratio	7.68 – 12.13	6.98 – 10.14
Timing Delay ratio (M2A2 vs. ASIC)	1.50 – 2.60	1.31 – 2.71

Table 14: M2A2 SDL50 vs. M2A2 CDL50 Synthesis Results.

QoR	SDL70	CDL70
Feedstock Size (μm * μm)	71*71.86	71*71.86
#Feedstock cells used	3 (*3 = 9)	3
Worst area ratio (M2A2 vs. ASIC)	1.08 – 1.28	1.08 – 1.28
Synthesis critical path delay ratio	0.98 – 1.02	0.91 – 1.00
Critical path length ratio	6.18 – 9.74	7.82 – 11.24
Timing Delay ratio (M2A2 vs. ASIC)	1.39 – 2.02	1.39 – 2.94

Table 15: M2A2 SDL70 vs. M2A2 CDL70 Synthesis Results.

The synthesis results obtained for both SDL and CDL approaches did not vary much, which is good from the perspective that feedstocks designed by FLG algorithm are generic. The performance of a high performance design is also not degraded much, for the same area. However, CDL approach makes use of lesser number of feedstocks in design. In SDL approach, 9 feedstocks (3 for each design) are used, whereas in CDL approach only 3 feedstocks are used. It is possible that results obtained with SDL and CDL match to some extent, due to the fact that same design is implemented for 3 different corners. The feedstocks generated in SDL approach for different corners might be similar to each other. Nonetheless, the feedstock design looks generic to serve different design needs, since the cells are packed differently in the design corners.

6.6 Comparison of M2A2 DDL and M2A2 CDL Results

In this section, the M2A2 results obtained for specific corners of design using DDL and CDL approach are compared. Table 16 and Table 17 list down the comparison of DDL50 vs. CDL50 results obtained for High performance design synthesis, and DDL70 vs. CDL70 design results obtained for balanced design synthesis respectively.

QoR	DDL50-5	CDL50-5
Feedstock Size (μm * μm)	50*50.16	50*50.16
#Feedstock cells used	5	5
Worst area ratio (M2A2 vs. ASIC)	1.10	1.10
Synthesis critical path delay ratio	1.48	1.00
Critical path length ratio	14.78	10.12
Timing Delay ratio (M2A2 vs. ASIC)	4.07	2.71

Table 16: M2A2 DDL50 vs. ASIC High Performance Design Synthesis Results.

QoR	DDL70-3	CDL70-3
Feedstock Size (μm * μm)	71*71.86	71*71.86
#Feedstock cells used	3	3
Worst area ratio (M2A2 vs. ASIC)	1.08	1.08
Synthesis critical path delay ratio	0.96	0.99
Critical path length ratio	8.23	7.82
Timing Delay ratio (M2A2 vs. ASIC)	1.80	1.78

Table 17: M2A2 DDL50 vs. ASIC Balanced Design Synthesis Results.

The synthesis timing result obtained for DDL50 in high performance design corner is 1.5x of timing results to the CDL50 design. The DDL timing is degraded due to the fact that synthesis is performed using HVT and RVT standard cells which are mostly present in the reference designs of DDL design library. However, timing results almost match for DDL70 and CDL70 designs, since the HVT and LVT cells are present in reference designs to generate correct feedstock library. The interconnect length did not vary much which shows that FLG and FP2DG algorithms are working correctly.

6.7 M2A2 CDL Results for different #feedstocks

In this section, the CDL50 synthesis results are presented for different number of feedstocks. The number of feedstocks drives the NRE cost of M2A2 circuits. Higher the number of feedstocks, more will be the NRE cost. The CDL50 synthesis is performed for all 3 design corners with different number of feedstocks, i.e. for 1, 3 and 5, named as CDL50-1, CDL50-3 and CDL50-5 respectively. Table 18 lists down the synthesis results comparison for all 3 design corners when synthesized using CDL approach with 1, 3 and 5 feedstocks in design library.

It can be observed that synthesis timing is not affected much in all 3 cases. It is due to the fact that FP2DG algorithm is not timing aware. The same floorplans are used for all 3 cases, and the timing delay is expected to be the same. However, it is strongly believed that the final timing for three cases will be different, since backend physical design will be optimizing the design by swapping the cells to reduce the interconnect delay, since the synthesis is not made physical aware as of now. The CDL50-5 design is expected to perform better due to low interconnect delays in design in comparison to CDL50-1 design. The power is also expected to be lower for CDL50-5 in comparison to CDL50-1, since it offers more flexibility in placing HVT cells in the design.

QoR	CDL50-1	CDL50-3	CDL50-3
Feedstock Size (μm * μm)	50*50.16	50*50.16	50*50.16
#Feedstock cells used	1	3	5
Worst area ratio (M2A2 vs. ASIC)	1.10 – 1.12	1.10 – 1.12	1.10 – 1.12
Synthesis critical path delay ratio	0.85 – 1.00	0.95 – 1.00	0.88 – 1.00
Critical path length ratio	5.90 – 11.88	6.60 – 15.26	6.98 – 10.14
Timing Delay ratio (M2A2 vs. ASIC)	1.21 – 3.05	1.34 – 3.68	1.31 – 2.71

Table 18: M2A2 CDL50-1 vs. CDL50-3 vs. CDL50-5 Synthesis Results.

6.8 Comparison of M2A2 and ASIC Performance

In this section, the area and circuit speed comparison of M2A2 and ASIC design is presented. Based on all the synthesis experiments, the M2A2 based circuit is expected to run 1.5x-3x slower than ASIC design, while the area increases in the range of 1.05x-1.12x. This performance is based on implemented algorithms and expected interconnect delay, without considering any backend design optimizations. The power consumption comparison is not made in this work, since physical design of M2A2 circuit is still not done, and power consumption cannot be correctly determined at synthesis stage.

6.9 Comparison of M2A2 and FPGA Performance

In this section, the area and circuit speed performance of M2A2 and FPGA design is presented. According to the experimental results presented in ICCAD'07 [34], FPGAs occupy 18x-35x area in comparison to the ASIC designs. FPGAs operate at the speed which is typically 3x-5x slower than ASICs. The power consumption in FPGAs varies in the range of 7x-15x to that of ASICs. Using this data and M2A2 synthesis results discussed above, the area and circuit speed comparison between M2A2 and FPGA is made. The M2A2 circuits are 1x-3x faster in comparison to the FPGA circuits. The devices made using M2A2 technology are expected to be 15x-30x smaller than the FPGA devices. The power comparison is not made, since power consumption of the M2A2 circuits is not available. However, it is expected that the M2A2 circuits will be consuming power similar to that of ASIC circuits, and will be much better than FPGAs.

6.10 M2A2 Results Inferences & Claims

In this section, the key inferences drawn from M2A2 synthesis results and their comparisons with ASIC and FPGA designs are mentioned. Following are the important inferences and claims of this work:

1. The synthesis timing, i.e. timing without considering the interconnect delay, matches the ASIC design timing. This proves the correct functioning of FLG and FP2DG algorithms.
2. The area of M2A2 circuits increases by $\leq$12% as compared to ASIC circuits.
3. The overall timing including the expected interconnect delay varies in the range of 1.5x-3x to that of ASIC design timing.

4. The actual interconnect timing result might improve, since backend optimizations have not considered in this work.
5. The critical path length for M2A2 design varies from 6x-10x to that of ASIC design.
6. The feedstocks designed using FLG algorithm are generic, since the synthesis timing results for CDL, SDL and DDL approaches did not vary much. This ensures scalability of M2A2 design approach.
7. The FLG and FP2DG algorithms are working correctly in terms of adding required number of cells, since synthesis tool did not choke or synthesis did not fail.
8. There is a cost tradeoff between the silicon die area and number of feedstocks used in design. By increasing the feedstock size to increase its area by a factor of 2, the feedstock count decreases by 40% (from 5 to 3). However, the die area increases by ~20%.
9. There is a need to perform partitioned PLE synthesis in order to get low interconnect lengths, by making synthesis more localized and physical aware.
10. The FLG algorithm needs to be improved by adding buffer insertion mechanism, drive strengths and VT estimation from ASIC design slack profiles, and to make it critical path aware i.e. feedstock design for a timing aware mode.
11. The FP2DG algorithm needs to developed for timing mode. It currently supports only area mode. This will lead to pre-placement of feedstocks in the design which have critical paths.
12. The FLG and FP2DG algorithms should be developed based on machine learning techniques. The greedy techniques might not work well with the diverse set of designs.
13. The improvements mentioned in #9, #10, #11, #12 are expected to reduce the high interconnect lengths in design, which will improve M2A2 device performance.
14. Further meaningful insights can be drawn for the M2A2 circuits by testing it on more diverse set of designs. It will likely lead to identification of new problems, which will eventually improve the EDA design approach of M2A2 circuits.

Chapter 7: Conclusion and Scope of Future Work

This chapter concludes the research work done in this project and mentioned in this document. The conclusion of this work is presented in section 7.1 The scope of future work which includes improvements and next steps are discussed in section 7.2.

7.1 Conclusion

A new type of ASIC named as Microscale Modular Assembled ASIC (M2A2) is proposed in this work. This technology is based on the selective vacuum based assembly technique for fabrication of ASICs using a limited number of mass-produced feedstock logic circuits. This document has discussed the EDA design for M2A2 circuits. The EDA implementation steps for M2A2 design include designing of feedstock cells, pre-placement of feedstock cells in design, generating the design collaterals in ASCII format, and then synthesizing design using feedstock library. The performance of M2A2 circuits is evaluated in terms of die area and circuit speed. The performance and area metrics of M2A2 based design are compared with both standard cell ASIC and FPGA designs to establish the value of this technology.

One of the main risks involved in this project was to find out if a limited number of generic feedstocks can be designed which can meet the functionality of standard cell based ASIC design, and provide significantly better performance in terms of area, power and timing relative to that of FPGAs. The NRE cost of M2A2 technology is expected to be significantly lower (see Figure 1) than ASICs only if there are limited number of feedstocks in design library, since NRE costs of M2A2 circuits includes cost of masks for all feedstocks. The synthesis experiments for different feedstock library combinations such as SDL, CDL and DDL for different feedstock sizes made use of minimal, i.e. 1, 3 or 5 feedstock cells in addition to the macro cells to synthesize the ASIC customer designs and met the functionality requirement with reasonable performance. The promising synthesis results have clearly addressed this risk. For diverse set of designs and design specifications, this solution may be further improved by making use of even less number of feedstocks.

The other main issue with ASIC design is the high turnaround time (TAT). This high TAT not only delays the launch of a product into market, but it also increases the design costs in terms of the EDA tools licenses, engineering costs, etc. The M2A2 circuits also make use of EDA tools,

but the design cycle time is expected to be lower than ASICs. This reduction is due to the fact that M2A2 design approach makes use of a limited number of feedstock cells to implement and optimize the design. This leads to a substantially reduced number of design variables, leading to low number of design optimization iterations, which decreases the turnaround time and NRE cost.

Besides addressing the high NRE cost issue, the M2A2 design approaches ASICs in terms of design performance, and die area. The synthesis of ASIC designs based on M2A2 feedstock approach resulted in a circuit speed, which almost matches (within 5% increase) with the ASIC design circuit speed. The die area increase is within 12% for all the runs, performed under different feedstock library constraints such as SDL, CDL, DDL and feedstock sizes, i.e. for 50um*50.86um and 71um*71.86um. The M2A2 synthesis results significantly beat FPGA design results in terms of circuit speed and die area. Typically, FPGAs are 3x-5x slower and have 18x-35x bigger die area in comparison to ASICs [34]. This comparison shows that M2A2 based circuits are significantly better in terms of die area and circuit speed than FPGAs. The power consumption of M2A2 circuits is not measured, since the results are obtained at the synthesis stage. However, it is expected that M2A2 circuits will not result in very high power consumption in comparison to ASICs. The M2A2 devices are expected to have lower power consumption than FPGAs, which consume 7x-15x of the power consumed by ASICs [34].

The timing results obtained after the synthesis stage do not include interconnect delay, which can be significantly higher in M2A2 circuits in comparison to the ASIC design. The critical path lengths for M2A2 designs are compared with those of ASIC designs to estimate the expected increase in interconnect delay. The results show 6x-10x increase in critical path lengths, which can add significant interconnect delay to make M2A2 device runs 1.5x – 3x slower in comparison to ASIC. This problem can be addressed by synthesizing the design based on partitioned PLE synthesis design approach, critical path consideration in FP2DG process, optimal buffer insertion feature in FLG, and optimizing the physical design to use cells placed nearby in design. Thus, critical path length increase is not expected to pose any serious threat to degradation in circuit speed. In a nutshell, this technology has tremendous potential to enable low volume, low cost ASIC production, if the issues highlighted in EDA domain are resolved.

7.2 Scope of Future Work

This section discusses the improvements which can be made to the solutions implemented in this thesis document. Following are the main tasks which can address the risks involved in EDA design and improve M2A2 design performance.

7.2.1 Physical Design Flow Development

The backend physical design flow needs to be developed and optimized to perform backend design optimizations. This backend flow can then be integrated with the existing M2A2 flow. The P&R results can then be obtained for diverse designs, and compared with standard cell based ASICs to get an accurate comparison of circuit speed and power for both the technologies.

7.2.2 FP2DG Improvements

The current FP2DG algorithm works in an area recovery mode only. The algorithm is designed in such a way that it results in optimal placement and selection of feedstock cells for minimal increase in area. The algorithm needs to be improved, to operate in the timing aware mode. In timing aware mode, feedstock placement and selection would be governed by the critical path gate delays and critical path lengths in order to optimize gate and interconnect delays.

7.2.3 FLG Improvements

The current FLG is implemented based on greedy and heuristic techniques. The solution can be further improved by devising intelligent buffer insertion methodology, better drive strength and VT class assignment to cells based on critical paths and slack profiles of ASIC design. The algorithm can be implemented using machine learning techniques to achieve the global optimal solution. These improvements will ease the design closure process.

7.2.4 Synthesis Improvements

The synthesis of M2A2 circuits has been performed using flattened ECO approach, which led to high critical path lengths. The critical path lengths can be significantly improved using partitioned PLE synthesis flow. The design constraints for all the partitions and capacitance table need to be generated in order to perform partitioned PLE synthesis for M2A2 designs.

References

[1] Tennant, D. M. "Limits of conventional lithography." Nanotechnology Springer New York, 1999. 161-205.

[2] X-Celeprint, “Micro-Transfer-Printing,” 2014. Available: www.x-celeprint. com.

[3] T. Westphalen, S. Hengesbach, C. Holly, M. Traub, and D. Hoffmann, “Automated alignment of fast- axis collimator lenses for high-power diode laser bars,” Proc. SPIE, vol. 8965, no. 0, p. 89650V, Mar. 2014.

[4] A. Das, R. Murthy, D. Popa, and H. Stephanou, “A multiscale assembly and packaging system for manufacturing of complex micro-nano devices,” IEEE Trans. Autom. Sci. Eng., vol. 9, no. 1, pp. 160–170, 2012.

[5] R. Ghadiri, T. Weigel, C. Esen, and a Ostendorf, “Microassembly of complex and three-dimensional microstructures using holographic optical tweezers,” J. Micromechanics Microengineering, vol. 22, no. 6, p. 065016, Jun. 2012.

[6] T. Huang, E. Nilsen, M. Ellis, K. Kim, K. Tsui, G. Skidmore, C. Goldsmith, A. Nallani, and J. B. Lee, “3-D, Self-aligned, Micro-assembled, Electrical Interconnects for Hetrogenous Integration,” in MEMS Components and Applications for Industry, Automobiles, Aerospace, and Communications, 2003, vol. 4981, pp. 189–201.

[7] Pelrine, Ronald E., Annjoe Wong-Foy, and Brian K. McCoy. "Manufacturing using levitated manipulator robots." U.S. Patent No. 8,941,270. 27 Jan. 2015.

[8] These plots and illustrations are the courtesy of Paras Ajay, PhD. Student, Department of Mechanical Engineering at The University of Texas at Austin.

[9] Bob Zeidman, “The Death of Structured ASIC”, Electronic Systems Design Engineering, Chip Design Magazine, 2006.

[10] Rick Mosher, “Structured ASIC Based SoC Design”, Design & Reuse, AMI Semiconductor, Dallas, US.

[11] Patel, Chetan, et al. "An architectural exploration of via patterned gate arrays." Proceedings of the 2003 international symposium on Physical design. ACM, 2003.

[12] Ho, Sam MH, et al. "Structured ASIC: Methodology and comparison." Field-Programmable Technology (FPT), 2010 International Conference on. IEEE, 2010.

[13] “FPGA vs. ASIC, What to Choose?”, Available: www.anysilicon.com, 2016.

[14] Moon, Euclid Eberle. Interferometric- spatial- phase imaging for sub- nanometer three-

dimensional positioning. Diss. Massachusetts Institute of Technology, 2004.

[15] L. Bening and H. Foster, "Optimizing multiple EDA tools within the ASIC design flow," in IEEE Design & Test of Computers, vol. 18, no. 4, pp. 46-55, Jul/Aug 2001.

[16] O. Coudert, "Timing and design closure in physical design flows," Proceedings International Symposium on Quality Electronic Design, 2002, pp. 511-516.

[17] A. H. Salek, Jinan Lou and M. Pedram, "An integrated logical and physical design flow for deep submicron circuits," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 9, pp. 1305-1315, Sep 1999.

[18] N. A. Modi and M. Marek-Sadowska, "ECO-Map: Technology remapping for post-mask ECO using simulated annealing," 2008 IEEE International Conference on Computer Design, Lake Tahoe, CA, 2008, pp. 652-657.

[19] N. A. Modi and M. Marek-Sadowska, "ECO-Map: Technology remapping for post-mask ECO using simulated annealing," 2008 IEEE International Conference on Computer Design, Lake Tahoe, CA, 2008, pp. 652-657.

[20] Mark McDermott, “Project Description, VLSI II Spring 2016 Course”, The University of Texas at Austin, 2016.

[21] N. S. Chaudhari, "Intelligent systems and polynomial solvability of NP-complete problems," 2010 IEEE Conference on Cybernetics and Intelligent Systems, Singapore, 2010, pp. 132-137.

[22] M. Zhao, Z. Li, Y. Wang and Q. Xu, "Longest Common Sub-sequence Computation and Retrieve for Encrypted Character Strings," 2016 19th International Conference on Network-Based Information Systems (NBiS), Ostrava, 2016, pp. 496-499.

[23] W. Y. Wu, "A Method for Fuzzy String Matching," 2016 International Computer Symposium (ICS), Chiayi, 2016, pp. 380-383.

[24] V. W. Ajin and L. D. Kumar, "Big data and clustering algorithms," 2016 International Conference on Research Advances in Integrated Navigation Systems (RAINS), Bangalore, 2016, pp. 1-5.

[25] Rui Xu and D. Wunsch, "Survey of clustering algorithms," in IEEE Transactions on Neural Networks, vol. 16, no. 3, pp. 645-678, May 2005.

[26] I. Hong, S. H. Sohn, J. K. Lee and J. M. Kim, "DFS algorithm with ranking based on Genetic Algorithm in UHF portable system," 2009 9th International Symposium on Communications and Information Technology, Icheon, 2009, pp. 454-459.

[27] C. Wang, Y. Lu, B. Zhang, T. Gao and P. Cheng, "An optimized BFS algorithm: A path to load balancing in MIC," 2015 IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2015, pp. 199-206.

[28] A. B. Kahng, Bao Liu and Xu Xu, "Constructing current-based gate models based on existing timing library," 7th International Symposium on Quality Electronic Design (ISQED'06), San Jose, CA, 2006, pp. 6 pp.-42.

[29] D. Baez-Lopez, J. L. Ballesteros and J. Pedraza-Chavez, "Conversion of circuit schematics from a graphic display to a netlist and its applications," Proceedings of 36th Midwest Symposium on Circuits and Systems, Detroit, MI, 1993, pp. 1159-1161 vol.2.

[30] “Post-Mask ECO using Conformal ECO”, Conformal-ECO (CECO) Version 15.10-p100, Cadence, September 2015.

[31] “Advanced Low Power RAK”, Conformal CPF Flow, Cadence, October 2016.

[32] “RTL Compiler-Physical Application Note”, Product Version RTL Compiler 11.2, Cadence, June 22, 2012.

[33] “Enabling RTL-to-GDSII Flows”, ECO Challenges, Version 2.4, Cadence, November 2014.

[34] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 203-215, Feb. 2007.