Structured, Technology Independent VLSI Design[1]

J. Gu[2] and K.F. Smith

UUCS-89-008

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

August 29, 1990

[1]This work has been supported in part by DARPA under Contract No. DAAC-11-84-K-0017, in part by 1987-1988 ACM/IEEE Design Automation Award, and is currently supported in part by 1988-1989 ACM/IEEE Design Automation Award.

[2]Jun Gu is now in Dept. of Electrical Engineering, gu@enel.UCalgary.CA

Structured, Technology Independent VLSI Design[1]

J. Gu and K. F. Smith

UUCS-TR-89-008

VLSI Group
Department of Computer Science
University of Utah
Salt Lake City, UT 84112
gu@cs.utah.edu

April 5, 1989
Rev. Aug. 1990

Summary

Rapid advancement in new semiconductor technologies has created a need for the design of existing integrated circuits using these new technologies. These new technologies are required to provide improved performance, smaller feature sizes and lower costs. The conversion of an integrated circuit from an existing technology to a new technology, however, is very difficulty with existing CAD tools.

In this research, we have concentrated on developing a structured, technology independent VLSI design methodology, with the goal of theoretically quantifying technology independence and systematically performing technology transformation. We have identified the nature of the problems, using techniques developed during our past research, within the context of particular semiconductor technologies such as CMOS and GaAs technologies.

Keywords: Computer-Aided Design (CAD), Path Programmable Logic (PPL), Technology Independence, Technology Transformation.

---

[1]This work has been supported in part by DARPA under Contract No. DAAC-11-84-K-0017, in part by 1987-1988 ACM/IEEE Design Automation Award, and is currently supported in part by 1988-1989 ACM/IEEE Design Automation Award.

Jun Gu is now in Dept. of Electrical Engineering, gu@enel.UCalgary.CA

Contents

1 Introduction

There are two ways to provide more processing power for an architecture, either at the system level or at the processor level. A powerful architecture is achievable if the capability of a single processor can be extended through massive concurrent execution. An architecture can also be made more powerful by relying on technological improvements to push the processor beyond its current maximum capabilities, e.g., small geometry CMOS, fast ECL, GaAs, etc.

Although there are many CAD systems currently being developed, most of them are technology (or process) dependent. An enormous amount of effort has been made to convert an existing circuit to new technologies in order to take advantage of faster circuit speeds, small geometrical sizes and low costs.

The term *technology (or process) independence* could be defined as existing between variations in a semiconductor process. For example, the differences which exist between 3.0 μ, 2.0 μ, and 1.5 μ CMOS processes. In this research we take a much more global view and define technology independence to mean independence between major technologies. For example, independence between NMOS, CMOS and GaAs technologies. Technology independence provides several important benefits to integrated circuit design:

- Module library portability — circuit modules that have been designed in one technology can be made available in many different technologies;
- Upgradability — circuits designed in older, less attractive technologies can be

easily reimplemented in newer technologies that have desirable characteristics such as higher speed or lower power consumption;

- Cost-effective prototyping — prototype circuits can be implemented using the least expensive (or fastest turnaround) technology to evaluate functionality and then implemented using the desired technology; and
- Tradeoff analysis — a circuit can be functionally evaluated before decisions that depend on the fabrication process are made. It is also possible to obtain relative performance data using a relatively inexpensive process.

This research has concentrated on building a structured, technology independent VLSI tool so that we will be able to increase architecture computing throughputs by taking advantage of the state-of-the-art semiconductor technologies with little or no modifications of the existing CAD system [17, 18]. In Section 2, a structured VLSI design methodology called Path Programmable Logic (PPL) is introduced. In Section 3, logic partitioning strategy and its role to achieve effective technology independence, as well as the PPL logic partitioning scheme, are discussed. Our previous and recent work in developing PPL technology independence and technology transformation methodologies for NMOS, CMOS, and GaAs technologies are described in Sections 4 and 5. Finally in Section 6, the conclusions are made.

2 PPL – A Structured VLSI Design Methodology

In has long been realized that to facilitate integrated circuit design, a CAD tool which supports different levels of design procedures, ranging from composite layout, through schematic and logic design, to symbolic manipulation is necessary. To be useful for rapid VLSI design, we must have such a tool which completely operates on a symbolic layout level and avoids the lower level details. Research by the VLSI Group at the University of Utah during the last several years has resulted in a symbolic, structured, and technology independent IC design methodology, known as PPL [12, 15].

2.1 The Principles of PPL Design Methodology

PPL is a structured VLSI design technique using a *cell matrix* methodology where cells are placed at arbitrary locations. The cells are connected on all four sides and interconnection is accomplished by placing cells next to each other. There are elementary cells which perform AND and OR operations for a PLA (Programmable Logic Array). The AND condition of the input signals are formed on the rows of the PPL and the OR conditions are formed on the columns. Cells in addition to those needed to perform the sum of products realization of a function are also provided and may be inserted into the grid at *arbitrary locations.* These include flip-flops, inverters, loads, row and column connections, and pass transistors. Consequently both combinational and sequential circuits can be easily designed.

Using this methodology, design of a circuit is performed by placing cells which can be represented by logical symbols on a grid representing the integrated circuit

(see Figure 4). When the grid is completely populated, it is both the logic representation and topological layout of the circuit. The cells have predefined schematic and composite representations. They are custom designed to optimize performance and size for the chosen integrated circuit process.

The detailed discussion of the PPL logic partitioning strategy will appear in Section 3.2.

2.2 An Example

The design of PPL circuits can best be understood by examing a simple design example, e.g., a binary counter (or divide by 4) circuit design. The counter states transitions are given in Figure 1.

This counter circuit can be implemented using PLA design, as shown in Figure 2. This circuit consists of three structures: the AND plane, the OR plane, and the state variable memory (memory plane). The inputs to the AND plane enter on column wires and the outputs exit on row wires. The inputs to the OR plane enter on row wires and the outputs exit on column wires. It is assumed that the column signals

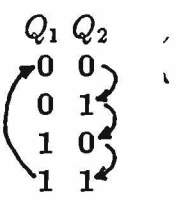

Figure 1: The Binary Counter Sequence

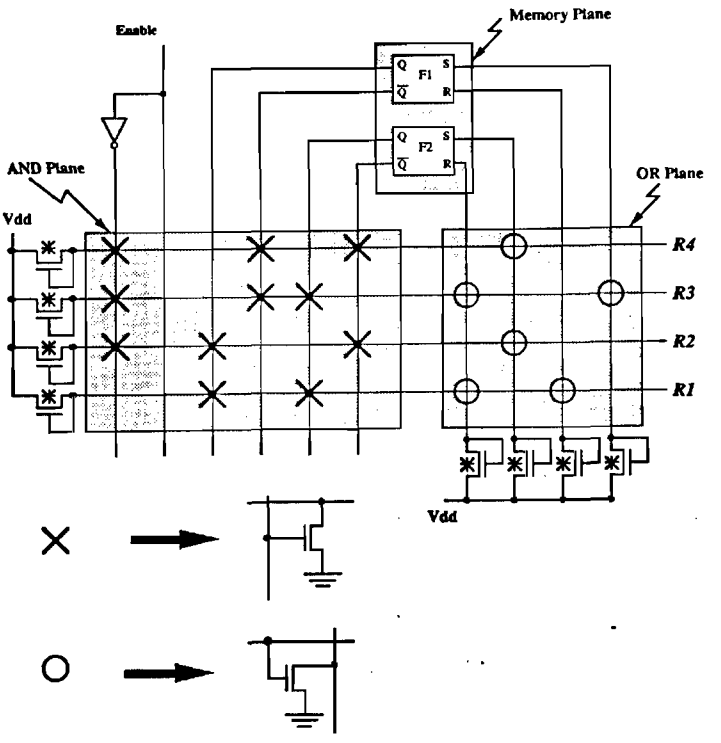

Figure 2: The PLA Implementation of a Binary Counter

will be true when low, i.e., 1 = *low*. Each of these three parts is compact in itself, but they cannot be merged into a single, coherent structure.

If we now fold the AND and OR planes onto one plane, a better structure which is illustrated in Figure 3 can be obtained. Note that the output columns have been interspersed with the input columns. The folded planes can now be viewed as a single grid, with each location containing either AND elements, OR elements, or memory elements as shown by the shaded boxes. This concept was extended to allow arbitrary circuits elements to be placed in any grid location of the folded plane.

Bearing this structural difference in mind, examination of the PPL design will illustrate its differences from conventional circuit design. In a conventional IC circuit design, we generally first make a logic design by drawing a circuit schematic, then simulate the design (this processes usually repeats itself many times to refine

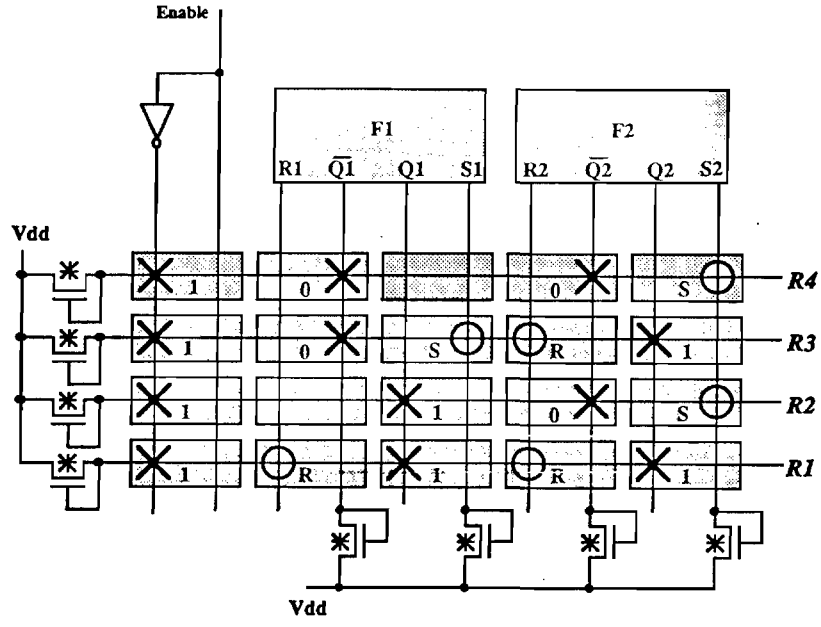

Figure 3: The PPL Design of a Binary Counter

the original design), finally we layout the composite patterns on a graphical VLSI workstation.

In PPL design, we do not have to experience the tedious procedures that were encountered in conventional custom design. Instead, the counter sequence in Figure 1 can immediately be mapped into PPL symbolic layout as shown in Figure 4. Note that we have combined rows $R2$ and $R4$ into a single row because $F2$ is always set whenever $F2$ and *Enable* are both 1, i.e., $F2$ toggles regradless of the state of $F1$.

Note that in PPL design, two distinct operations are performed on the layout of Figure 3. First, a symbol is assigned to each shaded box. Second, we made all of the shaded boxes an integral multiple of a unit cell. In this case, the **1**, **0**, **R**, and **S** cells are unit cells. The flip-flop cell **F** is composed of 2×3 or 6 total unit cells. This flip-flop would now be considered to be a "macro" cell.

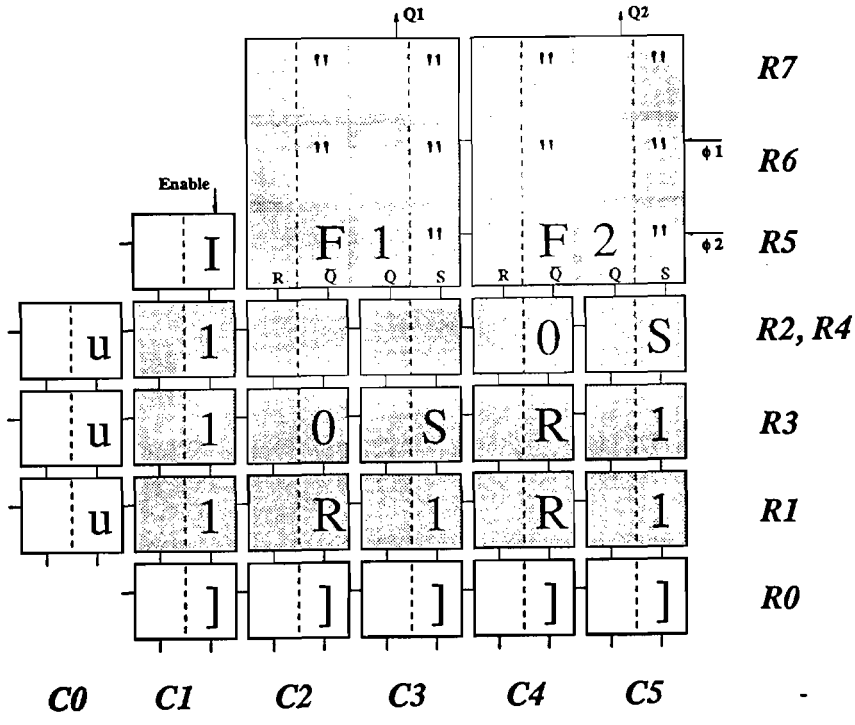

Figure 4: PPL Symbolic Layout for a Binary Counter

In this binary counter example, six PPL cells, i.e., **1**, **0**, **S** (*set*), **R** (*reset*), **I** (inverter), and **F** (flip-flop), are used. For simplicity, we only show the PPL symbols. The PPL symbol, the NMOS schematic and the NMOS composite layout of two representative PPL cells, i.e., the PPL cell "**1**" and cell "**R**", are shown in Figure 5. The logic synthesis including the PPL layout for this binary counter is governed by some simple If-Then rules which can be derived directly from the counter sequence in Figure 1. For example, to place PPL cells on the second row (i.e., *row 2* in Figure 4) of the PPL layout plane, the two simple If-Then rules which are shown in Figure 6 are applied. The counter circuit schematic (using NMOS for this example) described by PPL symbolic layout is illustrated in Figure 7.

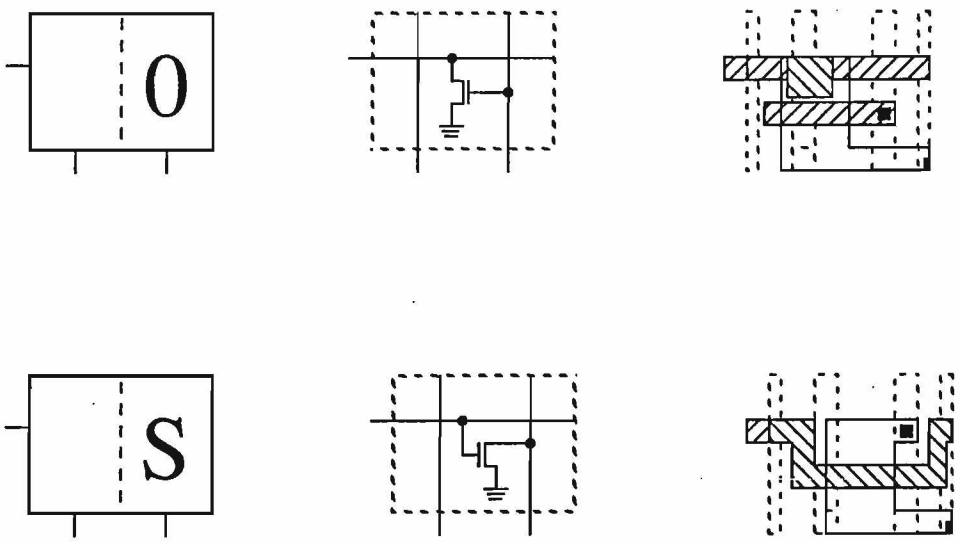

Figure 5: Symbol, Schematic, and Composite Layout of Two PPL Cells, **1** and **R**

Rule 1: *If* $Q_1 = \mathbf{0}$ and $Q_2 = 1$, *Then* Cell **S** sets Q_1 to *1*.
Rule 2: *If* $Q_2 = \mathbf{1}$ and $Q_1 = 0$, *Then* Cell **R** reset Q_2 to *0*.

Figure 6: PPL Layout Rules for Binary Counter at the Second Row

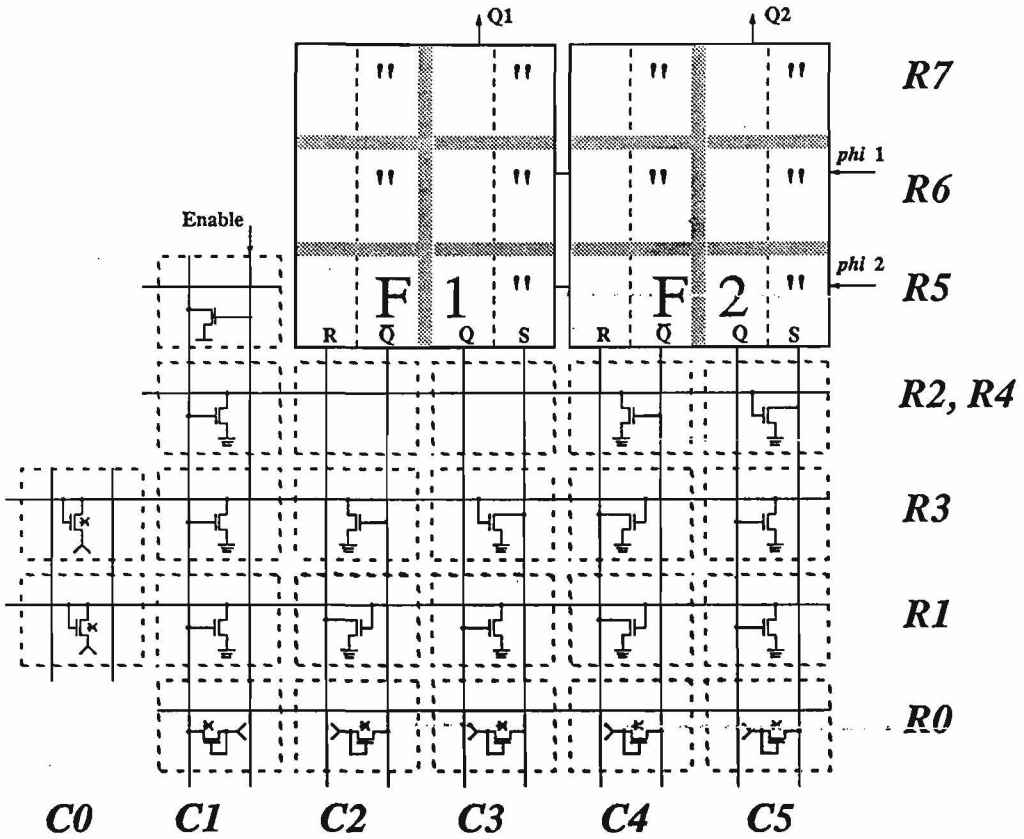

Figure 7: Binary Counter Circuit Schematic (NMOS)

2.3 The Basic Features of the PPL Design

PPL cells are the key to the performance of VLSI circuits and systems designed using the PPL methodology. At the lowest composite layout level, they must meet circuit schematic design, various electrical specifications, and multiple topological connection requirements. At the higher symbolic logic design level, the cell partition scheme for the PPL symbolic cell representation must support a general, rule-based, hierarchical, and technology independent IC design methodology. For details of NMOS, CMOS and GaAs PPL cell implementations see [5, 6, 7, 8, 13].

Among many unique features of the PPL design methodology, the following aspects are of significant importance.

1. LOGIC SYNTHESIS USING PPL.

As shown in the binary counter example given in Section 2.2, circuit synthesis including layout using PPL can be done, in most cases, based on a straightforward and *common understanding* of the given circuit functional specification.

Referring to the above counter design, from the counter sequence in Figure 1, a set of PPL design rules for the binary counter can be derived. Two of these PPL rules are shown in Figure 6 which represent the counter state transition from counter state $Q_1\ Q_2 = 01$ to the next counter state $Q_1\ Q_2' = 10$. The PPL symbolic counter circuit layout can be directly implemented in terms of these rules, as shown in the second row in Figure 4.

It is interesting to note that circuit logic synthesis using PPL is a *functionally con-*

ceptualized mapping rather than a *schematically conceptualized* design. For example, cell **0**, or cell **1**, senses column logic states and invokes a new row logic state, which forms an *AND* term in the PPL circuit and presents a PPL rule in an **If** statement (like a *noun* which represents an object in a rule statement). Similarly, cell **S** (i.e., *set*), or cell **R** (i.e., *reset*), sets a column logic state according to the current row logic state, which produces an *OR* term in a PPL circuit and generates a rule in a **Then** statement (like a *verb* which represents an action in a rule statement).

2. UNIVERSAL AND FLEXIBLE IC BUILDING BLOCKS.

The philosophy of the PPL logic partitioning described in Section 3.2 and in [7] can be summarized as: (1) partitioning the circuit logic into the *simplest and smallest blocks* so that any general integrated circuit and system can be built hierarchically; (2) maintaining a set of necessary functional macro cells so that the redundant efforts for building those frequently used circuits can be avoided. The simple PPL cells can be as small as a single transistor or one metal layer connection. The functional macros in PPL may be as large as a flip-flop or a full adder. Indeed, any circuit or system block could be used in this scheme as a PPL macro, e.g., a RISC system could be a macro in the PPL cell data base. Based on the generality and flexibility of the PPL design methodology, designers are capable of designing and implementing any VLSI circuit with a high degree of freedom. They can use simple PPL cells to build any desired circuit module which forms part of a larger architecture.

3. ONE STEP, MULTIPLE LEVEL CIRCUIT DESIGN.

PPL has the unique advantage of giving a *simultaneous high-level view* of the

symbolic, the logic, the schematic, the composite layout and the interconnect of a VLSI circuit in a single step. This provides not only the technical background to train designers who become familiar with all levels of the IC circuit design, but also frees them from many time-consuming, labor-intensive tasks when constructing a real IC architecture. For example, they do not have to be stuck on the low level details of composite layout, the extraction of circuit schematics, design rule checks (DRC), etc.

4. TECHNOLOGY INDEPENDENCE.

A clever feature of PPL design is its *technology independence.* For example, by symbolically replacing the PPL cells in the technology database, a PPL chip designed using an NMOS technology can be easily transferred into chips which will be fabricated using CMOS and/or GaAs technologies, or vise versa. The higher level PPL symbolic representation for NMOS, CMOS, and GaAs technologies could remain the identical.

5. USER-FRIENDLY INTERACTION.

User-friendly interaction of the PPL system is a direct consequence of the above several features. As will be discussed in Section 2.5, an order or more of magnitude reduction in design time (as compared to other semi-custom IC design techniques) for integrated circuits can be achieved using the PPL design methodology.

2.4 PPL Design Tools

The PPL design system, or PPL design tools, is a specific embodiment of the PPL IC design methodology. There are primarily three software packages in the PPL system,

a structured logic editor *Tiler* [11], a PPL circuit extractor *Simpplex* [10], and a unit delay simulator *Simppl* [10], which are used alternatively in the design process.

Tiler is a special interactive editor used for designing VLSI circuits using the PPL methodology. PPL tiles are inserted into a rectangular grid by typing characters that represent the tiles. When *Tiler* is first started, the user is prompted for the technology to be used for designing a PPL circuit. When a technology name is given by the user, a technology file is loaded into *Tiler*.

Simpplex is a PPL circuit extractor (with electrical rules check). The output of *Simpplex* is fed into a PPL circuit simulator.

Simppl is a switch level simulator designed for simulating circuits built using the PPL methodology. In addition to using a switch-level model, *Simppl* makes use of multiple logic-value (six for MOS) to accurately model the operation of soft and hard nodes, bussing, wired logic, and dynamic logic. There are both soft and hard values for the true and false logic levels. Since the complete circuit schematics have been automatically defined when PPL symbols were populated on the entire plane, simulation by *Simppl* is straightforward without requiring the extraction of circuit schematics for the simulator.

2.5 The Practical Assessment of PPL Design Methodology

To evaluate the efficiency and effectiveness of the PPL design methodology, some statistical data has been accumulated and several benchmark experiments comparing the PPL design method with other IC design methodologies for various performance

and sizing figures have been performed during the past several years.

Benchmark 1: During March 1988, three Japanese electrical engineers from Oki Data Inc. came to discuss the PPL design methodology. Rentek (a Salt Lake City local IC design company) implemented an existing Oki Data circuit which was designed by Oki Data using full custom layout methods. The new chip designed using PPL was completed in about $3 \sim 4$ man-weeks and was approximately $1/2$ the size of the original full custom chip was implemented in about one man-year.

Benchmark 2: During the spring of 1988, the VLSI group at the University of Utah implemented a Haugenhauer Filter using the PPL methodology and CAD tools. This filter was initially designed by an agency of the federal government using conventional techniques including standard cells and a silicon compiler. These designs resulted in chips which were approximately three times as large as that of the PPL design. The PPL circuit design was accomplished in a few weeks as compared to several months which were required for the design with more conventional techniques.

Figures 8 and 9 show two major comparisons in terms of design time and density of the circuits designed using PPL and other design methodologies.

3 Logic Partitioning Strategy and Technology Independence

Technology independence is generally considered difficult to achieve in integrated circuit design. A silicon compiler or a CAD design environment can only be as technology independent as the circuit implementation technique it uses. For example,

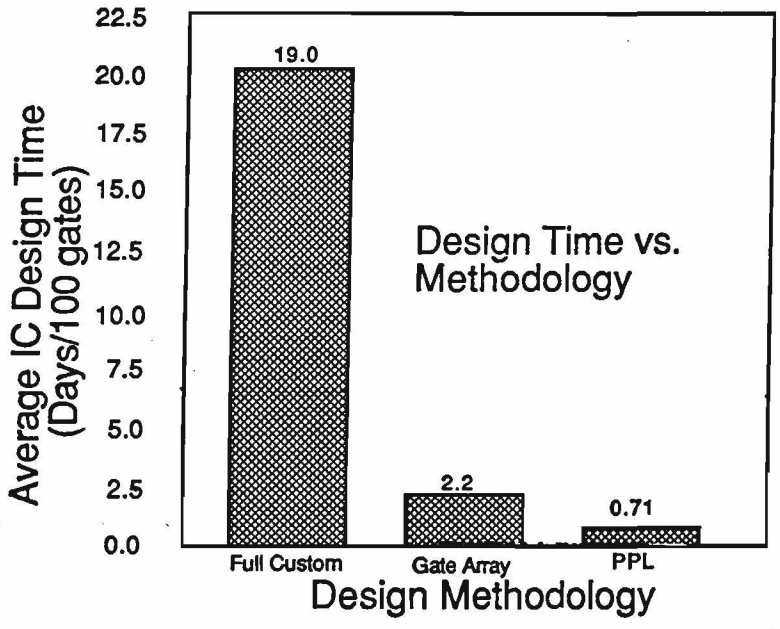

Figure 8: Comparison of Design Time with Different Methodologies (Courtesy of Rentek, Inc.)

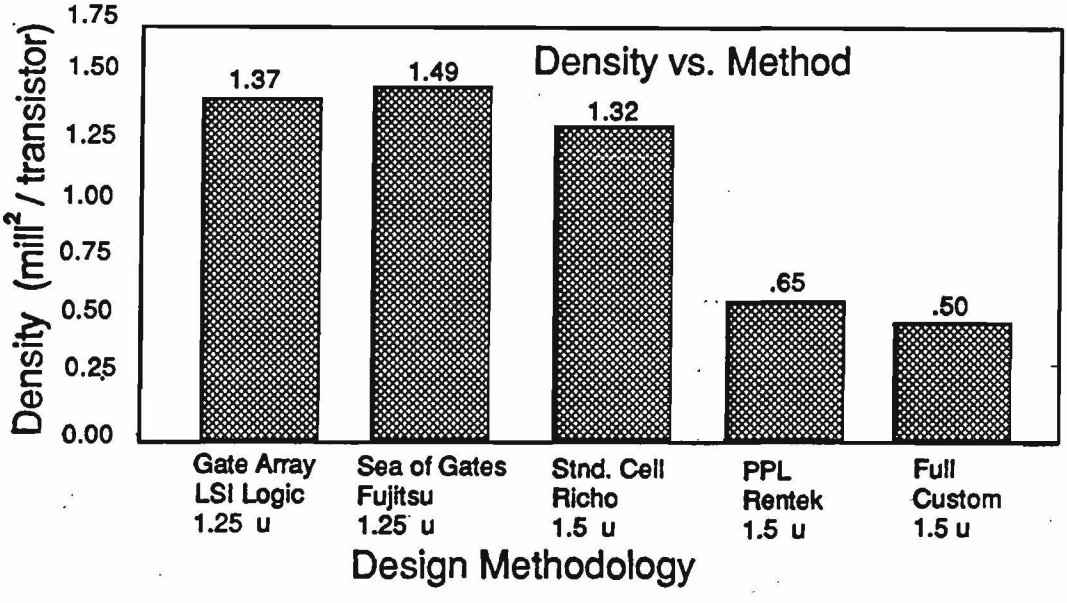

Figure 9: Comparison of Design Density with Different Methodologies (Courtesy of Rentek, Inc.)

a silicon compiler written for a CMOS process cannot be easily derived from one for an NMOS process [1].

3.1 Logic Partitioning, Circuit Abstraction and Design Generality

A general practice to achieve technology independence in a symbolic layout system is to keep the description of the primitives separate from the code that builds the layout. Furthermore, it is desirable that the symbolic layout system program read technology fabrication information at execution time to allow designers to update a symbolic layout by binding it to a new technology. The primitive descriptions are kept in the technology database.

So far there have been several categories of design strategies to implement a technology independence property into an existing CAD system (unfortunately there have been very few publications). We can roughly classify them into **low-level technology independence** - at the primitive symbolic and stick diagram level, or, **high-level technology independence** - at or beyond the functional cell and macro level. A CAD system implemented with this low-level technology independence simply generates circuit layout for each primitive symbol (vias, wires, etc.) at the layout level in terms of a given set of technology specifications. This category of design systems must manage numerous classes of information where each class is associated with a tremendous amount of data [16]. Conversely, a CAD system implemented with high-level technology independence generates macros which are usually inappropriate for the particular application, space-inefficient, and inflexible.

A representative example of a CAD system with high-level technology independence, for example, might be the standard cell design methodology, where functional cells and macros at or beyond certain higher circuit design levels are generated for different technologies. Functional macros representing the higher level circuits, however, lack the generality for versatile, area-efficient circuit design. They can only construct a circuit at or beyond the level at which the macro was designed. Each minor modification in the macro circuit schematic design, or in its physical geometrical layout, requires the re-generation of a complete new full custom macro. Therefore, it is not unusual to see a typical standard cell library that contains several hundred complex cells in order to accomplish most circuit designs, and is usually equipped with a standard cell module generator for generating the rest of necessary cells. The construction of such a CAD system with high-level technology independence usually requires a large amount of work, which may actually be equivalent to building an entire new system.

In short, a dominant factor which determines the optimum (in terms of design abstraction, computing efficiency, and universality for circuit design) technology independence and technology transformation is the granule size for partitioning the circuit logic. The lower level technology independence (**fine granule**) possesses design generality but lacks abstraction power for low level physical details, therefore, greatly increases computation overhead. In contrast, the higher level technology independence (**coarse granule**) possesses excessive high level abstraction and loses design generality, therefore, a tremendous amount of effort is required to design new macros.

3.2 PPL Logic Partitioning Scheme

A structured technology independent VLSI design methodology such as PPL is an attractive engineering direction in which an appropriate logic partition scheme at a **physically distributed, simple logic level** is used. In what follows next, we first describe in general the PPL logic partitioning philosophy with the emphasis on solving the following two problems:

1. How can we design a logic partitioning scheme that supports **general** integrated circuit design ?

2. How can we design a logic partitioning scheme that makes **less redundant** integrated circuit design ?

The PPL logic partitioning philosophy could be informally illustrated by an overview of four categories of the leaf PPL cells.

1. THE SMALLEST AND SIMPLEST LOGIC PARTITIONS.

In order to generalize a design methodology for a wide range of integrated circuit designs, a logic partitioning scheme which allows the decomposition of the circuit logic into its simplest and most general components is appropriate. The simpler each partitioned logic component is, the more general the logic function associated with this logic component. A similar analogy can be found in construction of an English word. For instance, in order to compose *any* English words, a partitioned character set consisting of 26 simple alpha characters (i.e., a, b, c, ..., x, y, z) are necessary and sufficient. A cell-set with a collection of the smallest and simplest cells

permits circuit design at or beyond any design levels, ranging from a transistor level, to gate level, macro level, circuit level, and system architecture level. Other CAD systems which could not support this simple logic partitioning are not able to allow the implementation of an intergrated circuit at an arbitrary design level, as discussed in Section 3.1.

Several primitive operations in the PPL logic partitioning scheme have been identified as:

- **Signal Passing:** This logic function indicates a signal interconnection where a logic value input into and output from this partioned logic component remains unchanged. There are two categories of PPL cells designed for this operation: **wiring cell**, e.g., blank cells, and **connection cell**, such as #, &, *, and $, etc (see Figure 10).

- **Signal Inversion:** The PPL design methodology is based on a distributed logic system made up of a plane of *AND* gates interleaved with a plane of *OR* gates. There are two important characteristics in this design approach. First, each logic variable of a vertical column bus can act as a stored binary variable or perform a logical *OR* function of the row inputs. When the column segment represents a stored variable, the values can be *set* and *reset* by the row's logic variable value. Second, each logic value of a row bus is a conjunction or product term over the selected column variables; a column input may be either the column value or its complement, or there may be no connections from the column to that row. These variables are ANDed to form the row value. In a

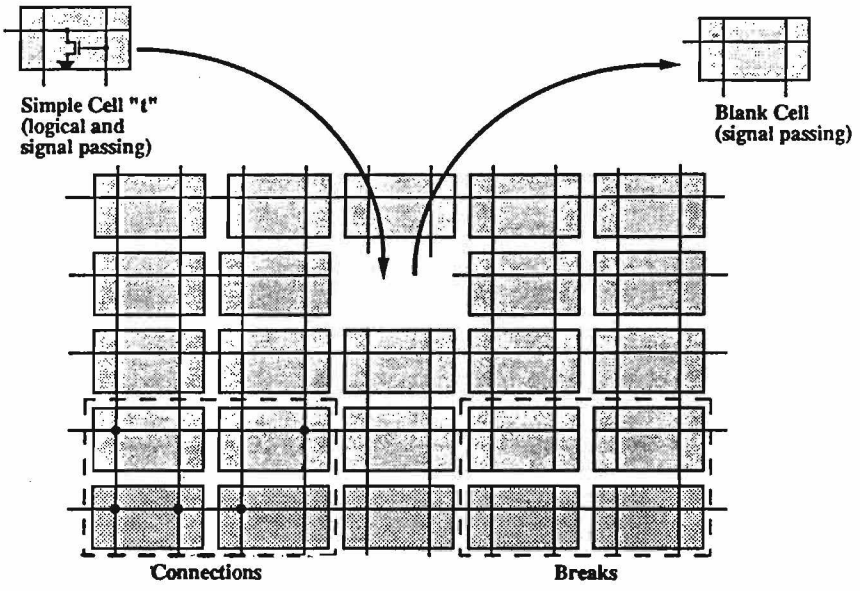

Figure 10: Some PPL Connection and Breaking Cells

more general sense, these logic operations can be viewed as signal inversion for different permutations of input and output over vertical and horizontal signal busses. They are partitioned as simple logic components as **1**, **0**, **S**, **R**, and **I**, etc.

- **Signal Breaking:** Several signal breaking operations, such as %, :, &, =, and !, have been identified which allow the multiple logic formulation and the insertion of any distinct circuit module at an arbitrary plane position. This permits much denser packaging of logic circuits into an array and the execution of much more complex functions on a single chip (see Figure 10).

These simple logic components have the flexibility to handle any general integrated circuit design. There is, however, a large degree of design redundancy which could

be introduced when constructing a circuit using simple, primitive logic components. The following three categories of logic partitions are aimed at minimizing circuit redundancy occurring during circuit design.

2. THE FUNCTIONAL LOGIC PARTITIONS.

When implementing the simplest cells to support a general circuit design, a number of functional logic components, such as flip-flop cells, pullup cells, RAM cells, and pad cells, are created to increase design functionality, productivity, and portability. Similar situations can be found in studying the English language. To spell out a character "cell", it is better use the word "cell" directly rather than compose it by using simple and primitive characters like "c," "e," "l," and "l", since this would be too redundant.

3. THE PARAMETERIZED LOGIC PARTITIONS.

Those logic components are designed under the same PPL symbol but with different electrical characteristics for the particular application. Pullups components (column pullups are represented as symbol] and row pullups are represented as symbol u in NMOS technology, as shown in Figure 4) are an example of this kind of cells where the different sizes of the pullup transistor can be selected by defining a modifier.

4. THE LOGIC PARTITIONS WHICH ARE IMBEDDED WITH EFFICIENT HARDWARE ALGORITHMS.

There are many highly concurrent VLSI computing algorithms and area-efficient hardware algorithms which are formulated in a regular (bit-sliced, cell-oriented, or,

mesh and tree structured) layout fashion. They can be finally decomposed into a number of the relatively individual and simple circuit modules. For example, systolic array is such a category of parallel architecture. This category of cells are used to implement those regular and individual functions. Both the parameterized logic partitions and the regular cell partitions may have analogies, in the English language, such as an affix and the roots of words.

In summary, each partitioned logic component produced by the PPL logic partitioning scheme is:

- **an abstracted PPL symbol,** i.e., a simple and general abstraction representing a building block to construct a circuit.
- **a 2-dimensional, topologically distributed logic function.** The distributed logic function can be specified by *function (symbol, X-coordinate, Y-coordinate).* Note that in general a circuit logic schematic, could be partitioned although, has not been associated with any topologically spatial dimension or spatial information.
- **a PPL cell,** which can physically and feasibly be implemented based on the available semiconductor technologies.

A PPL cell-set is a collection of the PPL cells which is a concrete embodiment of PPL logic partitioning strategy. A detailed discussion of the design and implementation of the PPL cell-set will appear in Section 4.

3.3 PPL Technology Independence

The technology independent property in PPL design differs considerably from conventional technology independent implementations which were described above (in Section 3.1). As described in Section 3.2, the fundamental logic partition of primitive PPL circuits focuses on structured, single transistors and their connection level, although some complex cells or macros have also been designed and implemented. This small granule partition scheme provides the background for low-level technology independence design and has several unique advantages: (1) The structured nature of the VLSI design provides high-level abstraction of low level details, in addition to fast design and computational efficiency. (2) Since this partition offers the most general (smallest and simplest) building blocks, any circuit design can be accomplished using a minimum set of some 15 to 20 simple, small cells.

Due to its simple and appropriate schematic level logic partition and high level symbolic abstraction, the PPL design methodology is able to avoid the manipulation of large amounts of information at the lower physical layout level while keeping abreast of new technology advancements with a minimum amount work (for redesign of a small number of cells). To update a newer semiconductor technology for PPL circuit design requires only the redesign of this small number of cells, which usually can be completed in one man-week. Therefore, the PPL design methodology can provide a broad spectrum of technology independence. Circuits designed using the PPL methodology are fundamentally technology independent. The PPL circuit design methodology offers a CAD design environment in which tradeoffs based on technology

can be easily considered and handled.

The technical effectiveness of the PPL technology independence can be seen from two aspects:

- **Technology Representation.** At or above the PPL symbolic representation level, the same circuit design based on different NMOS, CMOS, or GaAs technologies appears essentially identical. There are actually very few circuit design differences which can be detected.

- **Technology Transformation.** A circuit designed using PPL in one technology could be transformed into a similar circuit based on another technology, with little or no human interactions.

The original study of PPL technology independence in circuit design and an earlier experiment can be found in [2, 4] which were reported by Gu and Smith in [7]. Jacobs also discussed PPL technology independence and many details in [9]. Further developments in this direction are described in [4, 14].

4 Design and Implementation of Small Granule, Low-Level Technology Independence

Two simple and straightforward strategies for the design and implementation of the small granule, low-level technology independence for PPL design methodology are the **generic PPL cell-set** and the **technology-specific PPL cell-set**[2]. To begin

[2]More detailed discussion of the other methods are in [3].

our discussion, we need the following definition to distinguish our implementation strategies.

Definition:

A PPL cell, i.e., an embodiment of a partitioned logic component, is a *black box* consisting of the following 10-tuples:

- **A PPL Cell** *(a black box)*
  - External
    1. Symbol
    2. Geometrical Structure
    3. Appearance
    4. Local and Global Connectivity
    5. Power Distribution
  - Internal
    6. Logic Schematic
    7. Electrical Characteristics
    8. Internal Connectivity
    9. Physical Composite Layout
    10. Internal Power Distribution

4.1 The Generic PPL Cell-set

The ideas for the generic PPL cell-set are simple. When this cell, at the lowest layout level, is implemented using different semiconductor (e.g., NMOS, CMOS, and GaAs) technologies, a generic version of the cell could be developed. That is, the cell's (the external portion of a black box) higher level symbolic representation, its geometrical structure and appearance, its near-neighbor connection and global communication, and its power supply (both Vdd and Gnd), might remain identical. Thus, from a designer (a user) point of view, a PPL symbol (cell), at the symbolic level, is functionally and topologically equivalent in different semiconductor technologies.

Inside each black box, however, the construction of the same PPL symbol in different technologies involve many differences:

- different **logic schematics** for NMOS, CMOS, and GaAs circuits are chosen.
- different **electrical characteristics** (e.g., worst-case driving capability, clock speed and timing delay, loading effect, etc.) for different technologies are designed and simulated.
- different **internal connectivity** (cell's local connection and global communication patterns) for different technologies are defined.
- different integrated circuit **physical composite rules** are implemented.
- different **power supply** designs (power routing, power layout schemes) are used.

Obviously, the internal design and implementation of each PPL symbol is technology specific, or technology dependent. The generic nature of the high level PPL symbolic representation makes those technical and implementation details invisible to the designers.

The PPL cell-set designed with generic features has flexibility and portability when introducing a new semiconductor technology into the existing CAD system without modifying and rewriting the available system.

4.2 The Technology Specific PPL Cell-set

As a matter of fact, the design and implementation of a generic PPL cell-set is a multiple semiconductor technology based constraint optimization problem. Some fundamental constraints in PPL generic cell-set implementation may come from logic schematic, circuit electrical properties, physical composite layout, cell's local and global connectivity, and many other factors. Since constraint-based optimization is NP-complete in nature, an optimal solution for designing a PPL generic cell-set is hard to obtain. Therefore, one main drawback of a generic PPL cell-set implementation is that different constrains are encountered when competing multiple technologies enforce the worst-case penalties in design time, layout area, etc.

We have built several sets of technology specific cells for NMOS, CMOS and GaAs technologies[3]. These technology specific cell sets can be expanded to include a generic set. In terms of the definition, the external portion of a PPL cell, for example, its symbolic representation, its appearance (e.g., aspect ratio), local and global connectivity, and power routing, might not be the same for different technologies.

The major difference between a technology-specific PPL cell-set and a generic PPL cell-set is its performance improvement. The technology-specific PPL cell-set is designed and implemented with maximum optimization freedom in terms of the specific semiconductor technology used. That is, each cell representing a PPL symbol is specially implemented for optimal performance and minimum overhead in their schematic design, circuit speed simulation, physical layout, and connection band-

[3]There are currently two NMOS technologies, four CMOS technologies, and two GaAs technologies for the PPL cell sets that are available.

width design. Instead of structure and connection equivalence, there might be minor technology-specific modifications that are associated with each newer technology introduced. Thus, few corresponding adaptations must then be added into the existing CAD tools.

The implementation of a technology-specific PPL cell-set achieves much better performance figures than the pure generic one by avoiding numerous constraints in multiple-technology based cell design.

5 Technology Transformation Among PPL Circuits

Transforming PPL circuits from one symbolic representation to another can be done at several design levels: high-level circuit functional transformation, middle-level circuit schematic transformation, or low-level circuit structure transformation. Any circuit technology transformation for different technologies must produce circuit with functional consistency.

The higher level technology transformation involves functional (semantic) transformation and is closely related to the **re-generation** of an entire new circuit module. The low level circuit transformation, on the other hand, is more involved with circuit structure (syntactic) **symbolic mapping** and therefore more efficient.

The advantage of PPL technology independence is its small-granule based, structured partitioning scheme which allows direct, low level, structured technology transformations, without rebuilding the new circuit functional modules. A large system

consisting of simple PPL circuit modules can be automated for technology transformation. Most complex circuits designed using the PPL methodology in MOS or GaAs can usually be transformed manually by symbolic manipulations. Both are based on low level syntactic, structured technology transformations. Only in some unusual cases (where the high-level technology transformation is depend on functional redesign) will this be needed.

Deeper level electrical constraints, even under the same PPL symbolic representation pattern, might not be the same for different semiconductor technologies. In addition to structured mapping, symbolic level technology transformation among PPL circuits must also comply with the lower level electrical constraints of the new technology. Fortunately, most of these problems can be handled by built-in technology-specific design *rules* represented as structured placement and mapping rules.

5.1 Technology Transformation for Generic PPL Cell-set

By definition, the external portion of the generic PPL cell-set (i.e., its higher level symbolic representation, its geometrical structure and appearance, its near-neighbor connection and communication, and its power supply) might remain identical. Therefore, technology transformation among different PPL circuits constructed using generic PPL cell-set could simply be done by replacing different cell-sets corresponding to different technologies. The symbolic PPL circuit layout representation remains unchanged during the technology transformation procedure.

5.2 Technology Transformation for Technology Specific PPL Cell-set

In contrast, technology transformation for the PPL circuits which were designed based on technology specific cell-set may require the corresponding changes in the PPL symbolic layout representation. The problem, before technology transformation is taken, is knowing (usually at the symbolic level) beforehand the portions of the circuit which should be modified.

In a 2-dimensional integrated circuit layout plane which may contains thousands and millions of circuit components, there must be some common portions (schematic, circuit structure, or symbolic representation, etc.) and some singular portions of the circuits. One of the auxiliary products resulting in the PPL logic partitioning scheme is the separation of the **common property** (technology independent property) from the **singular property** (technology dependent property). A technology specific PPL cell-set is a combination of both circuit properties. This result is the basis for technology transformation based on technology specific PPL cell-set. When a circuit is transformed from one technology to another technology, its common property usually remains untouched, the only required changes which should be made are in the singular portion of the circuit.

We use the following example, a simple CMOS binary counter PPL design, to illustrate what we mean by the common (singular) property, and how to perform technology transformation among different technologies. This example corresponds to the NMOS binary counter PPL design example described in Section 2.2.

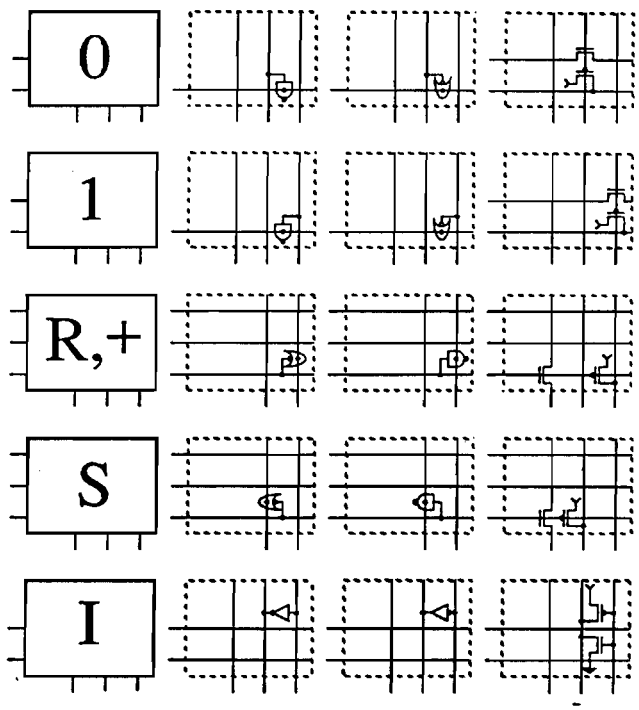

Figure 11: Symbol and Schematic of Some PPL CMOS Cells

In Figure 11, several CMOS PPL cells are given which are produced from the PPL logic partitioning scheme. Instead of decomposition of the circuit logic into a small schematic granule as in NMOS and GaAs technologies, a pair of complementary transistors are grouped together to form the simplest and smallest logic partition for CMOS technology.

Referring back to the NMOS PPL symbolic layout illustration in Figure 4, if we drop the row pullups (symbol **u** on the leftside of the layout) and column pullups (symbol] on the bottom of the layout), the NMOS binary counter representation in Figure 4 can then becomes a PPL representation of a CMOS binary counter layout as illustrated in Figure 12. The underneath CMOS schematic for the binary counter is shown in Figure 13 (short connections and Gnd connections are automatically routed by the editor ptogram).

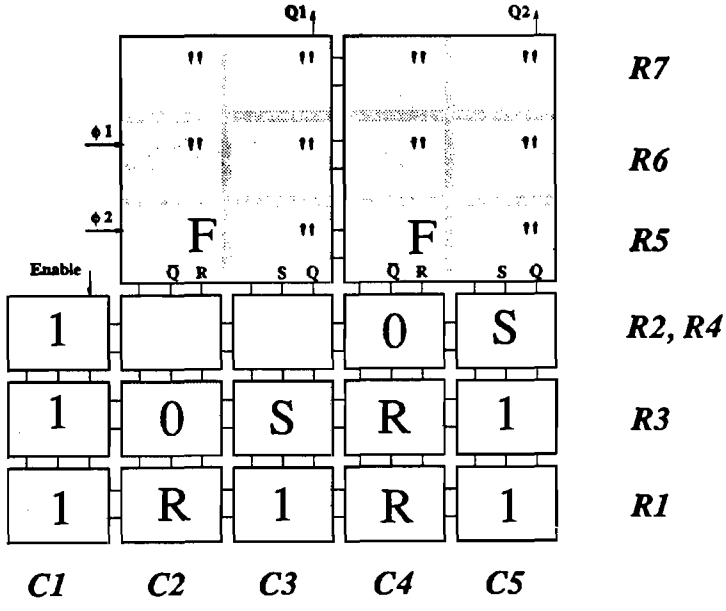

Figure 12: PPL Symbolic Layout for a CMOS Binary Counter

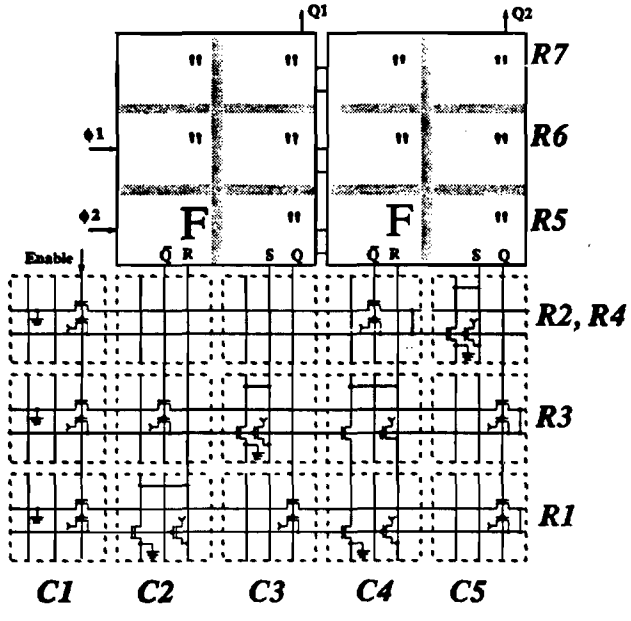

Figure 13: Binary Counter Circuit Schematic (CMOS)

In this example, those identical representations of the binary counter (i.e., **1**, **0**, **R**, **S**, **I**, and **F**) are the common representative properties, and those unique representations of pullup loads (i.e., symbol] and symbol **u**) and cell's near-neighbor connections are the singular properties in circuit transformation. Technology transformation of a PPL circuit symbolic representation involves only the static deletion or addition of those singular representations which are associated with a specific semiconductor technology. In most cases, these syntactic symbolic mapping can be done automatically.

5.3 A Case Study of PPL Technology Independence

There is a large number of design and fabrication experiments being done which demonstrate the suitability of the PPL methodology to technology independent design over the NMOS, CMOS, and GaAs technological spectrum. In what follows next, we give one case study based on PPL technology independence. This design example of a 4-bit linear feedback shift register has been fabricated using NMOS, CMOS, and GaAs technologies.

A representative circuit, a 4-bit LFSR (Linear Feedback Shift Register) (with some additional circuit and pad-testing circuitry), was designed by Ran Ginosar (a visiting professor from the Technion in Isreal). This test chip based on PPL technology independent design was fabricated using NMOS, CMOS, and GaAs technologies, at various features. Comparisons between all these test chips were made based on the similarity of the design and the resulting performance characteristics.

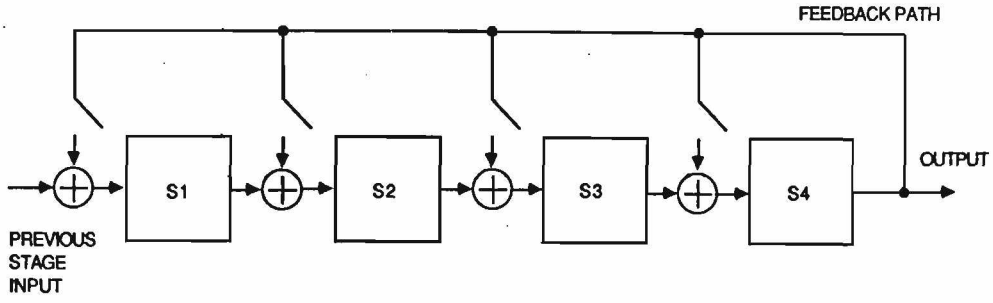

Figure 14: A Four Stage Cascadable LFSR Circuit

Linear Feedback Shift Registers consist of a shift register with Exclusive-OR (XOR) gates separating each successive bit register. The shift register output is fed back to some or all of the XOR gates. LFSRs are used to generate binary codes and sequences. A cascadable LFSR which allows dynamic configuration of the LFSR by means of switches is shown in Figure 14. Each switch determines whether the corresponding gates generates the XOR function of the feedback signal and the previous bit, or just passes the previous bit through, ignoring the feedback.

The design is cascadable, since multiple identical units can be chained to produce a longer LFSR. The output of the last stage serves as the feedback signal for all previous stages. The detailed descriptions of logic design of LFSR can be found in [14].

The complete GaAs PPL design of the *core* of the LFSR circuit (i.e., without the

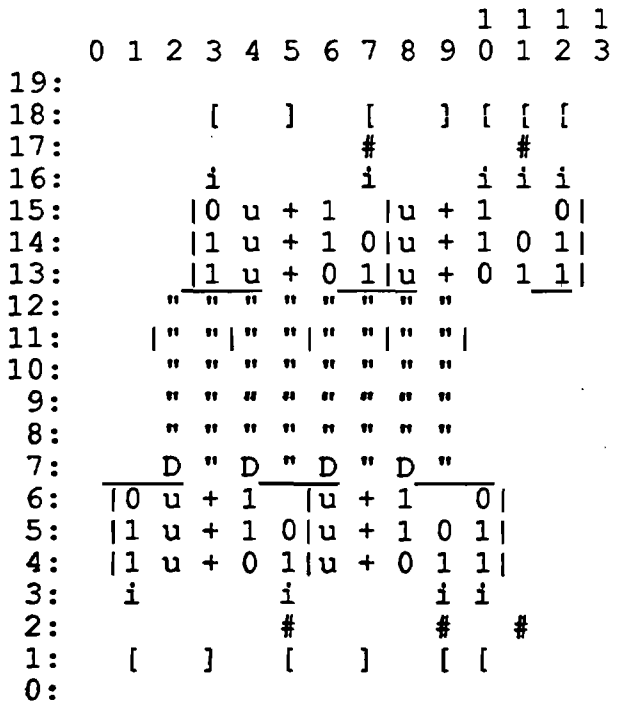

Figure 15: A GaAs PPL Layout of a 4-bit LFSR Circuit

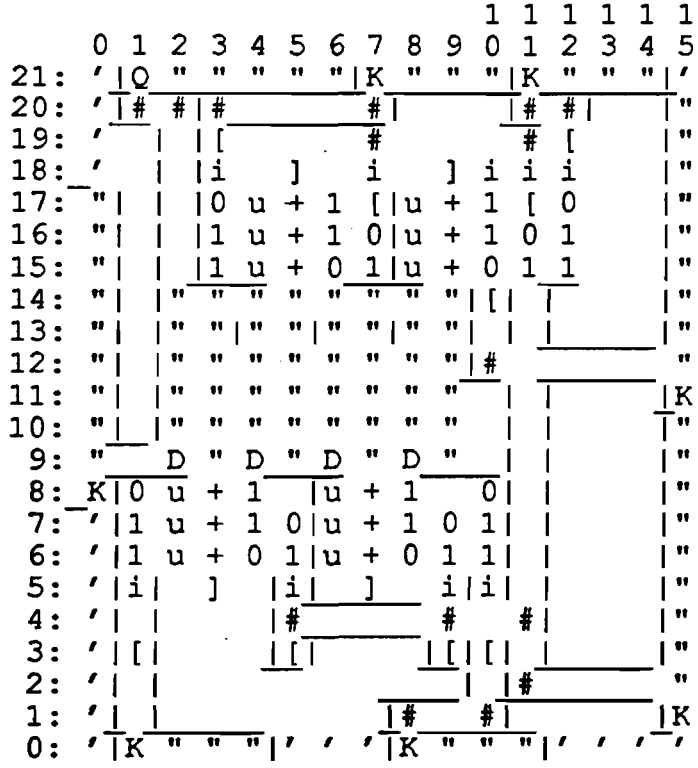

Figure 16: A PPL Layout of a 4-bit LFSR GaAs Chip

additional circuitry and the I/O pads) is shown in Figure 15. The PPL designs of the LFSR chip (the *core* circuit with the I/O pads) in NMOS, CMOS, and GaAs are shown in Figures 16, 17, and 18. They differ from each other, and from GaAs design, in only minor details which could be easily manipulated in most cases (e.g., deleting those static pull-up loads in NMOS or GaAs circuit design produces a corresponding CMOS circuit).

The same LSFR chips have been fabricated using NMOS, CMOS, and GaAs processes and found functionally correct in recent chip testing.

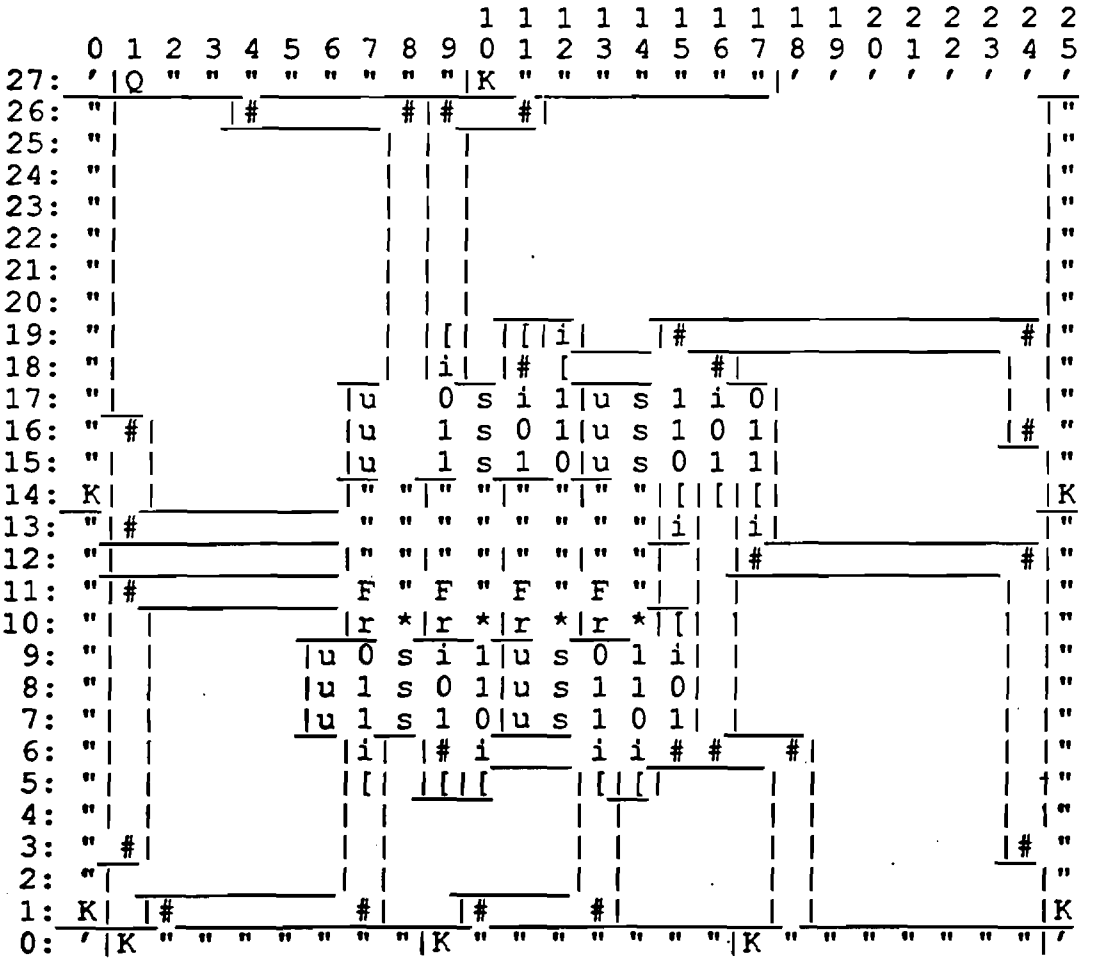

Figure 17: A PPL Layout of a 4-bit LFSR NMOS Chip

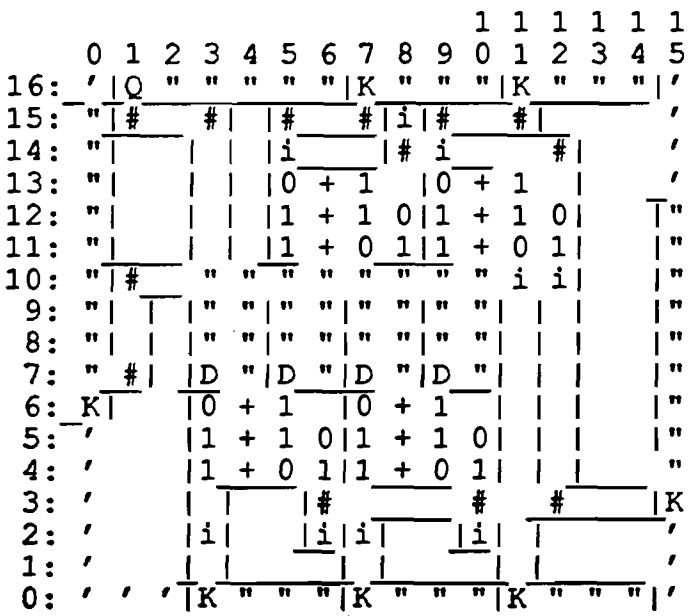

Figure 18: A PPL Layout of a 4-bit LFSR CMOS Chip

5.4 Critical Technical Issues in PPL Circuit Transformation

Much of our current research work for technology transformation has concentrated on GaAs-based circuits. However we have spent the majority of our efforts on MOS processes. The following partial listing illustrates some of the difficulties which we have observed that must be overcome when transforming a circuit among different technologies, in particular from a MOS circuit to a GaAs circuit. Some of our primitive ideas and research strategies for the corresponding item in the list are also presented below.

(1) Since passive load devices are used in NMOS, the outputs of most NMOS gates can be *wire-ORed* together to form a more complex gate. In conventional CMOS design, however, no passive device exist – explicit gates must be used. In this case the NMOS *wired-OR* gate must be identified, a CMOS equivalent designed, and a modified PPL implementation generated. Generating an efficient, modified PPL implementation is the principal problem.

In our logic partition scheme, the passive pullup loads are partitioned as a separate cell in NMOS PPL symbolic representation, they can be deleted when transforming an NMOS circuit to a CMOS circuit [7].

(2) CMOS, being an insulated gate technology, has a large fanin and fanout load on each gate, especially when speed is not a problem, since no DC current flows through the gate, and stages are isolated from each other for DC characteristics. In GaAs, on the other hand, with its Schottky diode gate which becomes forward biased, there is a significant DC interaction between the input and output of a stage. Therefore,

fanout must be limited to a few gates, regardless of performance requirements.

The implementation of this requirement when transforming a CMOS circuit to a GaAs circuit can be done in a straightforward way by limiting the fanout and fanin on each gate. In the worst case, a minor redesign, i.e., split the large fanout into several groups would work.

(3) In MOS technologies we often exploit the dynamic charge storage on gates as intermediate data registers. In GaAs the gate draws current and therefore dynamic data storage is generally prohibited.

In the worst case we could prohibit dynamic logic in CMOS.

(4) Power distribution requirements pose a major difficulty in GaAs circuits. Transforming from a MOS circuit to a conventional (non-PPL) GaAs circuit may involve a complete redesign of the floorplan.

Several forms of a GaAs power distribution meshes in the PPL grid plane have been developed which have low inductances and DC resistances. So far this class of solutions work well for GaAs circuits running about 300 MHz. This solution can be applied to both CMOS and GaAs arrays.

(5) MOS circuit designs allow high level of flexibility regarding placement and routing. GaAs circuits are extremely sensitive to length and relative placement of signal wires. Long lines could have serious reflections which may cause circuits to malfunction. Again, this may require a redesign for many CAD methodologies.

In PPL, the designer exploits placement locality and maintains control of wire length in all technologies, and as a result PPL designs are more amenable to automatic

porting from MOS to GaAs technologies.

(6) In MOS technologies, the most significant reactive impedances are capacitances to the substrate. In GaAs, where inter-layer capacitance and self- and mutual-inductances cause major disturbances, we must design modular cells that keep these difficulties manageable, so that a PPL array originally designed for MOS can be implemented in GaAs.

6 Conclusion

Rapid advancement in semiconductor technology has made integrated circuits designed using older technologies no longer competent. An enormous amount of effort has been taken to manually convert an existing circuit to new technologies in order to take advantage of faster speed, small chip size and low cost. Technology independent VLSI design is an attractive and useful technique that is capable of increasing computing system throughouts by taking advantage of state-of-the-art semiconductor technology with little or no modifications to the existing CAD systems.

We have developed a structured, technology independent VLSI design methodology based on small granule logic partitioning scheme and low-level technology independence implementation. Previous and recent research results based on a large number of experiments indicate the suitability of the PPL methodology for technology independent VLSI design over a wide technological spectrum including NMOS, CMOS and GaAs technologies.

Further study and investigation of some open technical problems (in Section 5.4)

will mature for the building of an advanced technology transform system for MOS and GaAs circuits.

References

[1] D. D. Gajski. *Silicon Compilation. VLSI Systems Design*, pages 48–64, Nov. 1985.

[2] J. Gu. *PEditor: A Rule-Based Prototype PPL Editor.* 1985.

[3] J. Gu. *Intelligent and Stuctured Techniques for MOS and GaAs VLSI Circuit Design.* PhD Thesis Proposal, to appear 1989.

[4] J. Gu. *Intelligent Silicon Compilation.* Research Proposal, March 12, 1985.

[5] J. Gu and K. F. Smith. *Design and Implementation of Optimal PPL Cell Set using 2.0 μ Metal Gate CMOS Process.* Technical report, Dept. of Computer Science, Univ. of Utah, June 1985.

[6] J. Gu and K. F. Smith. *Design and Implementation of Optimal PPL Cell Sets using 3.0 μ, 2.0 μ, and 3.0/1.2 μ Scalable CMOS Processes.* Technical report, Dept. of Computer Science, Univ. of Utah, Sept. 1985.

[7] J. Gu and K. F. Smith. *An Intelligent Circuit Module Generator.* In *Proceedings of 1986 IEEE International Conference on Computer Design: VLSI in Computers*, pages 470–475, Oct. 1986.

[8] J. Gu and K. F. Smith. *Design and Implementation of 1.0 μ GaAs PPL System for Ultra Fast VLSI Architectures.* Technical report, Dept. of Computer Science, Univ. of Utah, July 1987.

[9] S. R. Jacobs. *Automatic Technology Translation of Very Large Scale Integrated Circuits.* Thesis Proposal, Sept. 1986.

[10] S. R. Jacobs and K. F. Smith. *SIMPPL User's Guide.* Technical Report UUCS-TR-86-004, Rev.2, Dept. of Computer Science, Univ. of Utah, Nov. 1987.

[11] S. R. Jacobs and K. F. Smith. *TILER User's Guide.* Technical Report UUCS-TR-86-006, Rev.2, Dept. of Computer Science, Univ. of Utah, Nov. 1987.

[12] B. E. Nelson, D. R. Morrell, C. J. Read, and K. F. Smith. *PPL Integrated Circuit Design Methodology. Computer Aided Design*, 18(9):481–488, Nov. 1986.

[13] K. F. Smith. *Implementation of SLA's in NMOS Technology.* In *Proceedings of the VLSI 81 International Conference*, Aug. 1981.

[14] K. F. Smith, T. Carter, J. Gu, S. R. Jacobs, and R. Ginosar. *PPL: A Technology Independent VLSI Design Tool for NMOS, CMOS and GaAs.* Technical report, Dept. of Computer Science, Univ. of Utah, 1990, to appear.

[15] K. F. Smith, T. M. Carter, and C. E. Hunt. *Structured Logic Design of Integrated Circuits Using the Storage/Logic Array (SLA). IEEE Trans. on Electron Devices*, Ed-29(4):765–776, Apr. 1982.

[16] P. Smith and S. Daniel. *The VIVID System Approach to Technology Independence: the Master Technology File System.* In *Proceedings of 22nd ACM/IEEE Design Automation Conference*, pages 76–81, June 1985.

[17] E. Von and J. Gayle. *Role of Technology Design Rules in Design Automation.* In *Proceedings of 20th ACM/IEEE Design Automation Conference*, page 395, June 1983.

[18] W. H. Wolf and A. E. Dunlop. *Symbolic Layout and Compaction. In* Physical Design Automation of VLSI Systems, *B. T. Preas and M. J. Lorenzetti (Eds.)*, chapter 6. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1988.