Silicon compiler for implementation of systolic arrays using finite ring arithmetic.

Martin Jaekel
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/3458

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
SILICON COMPILER FOR
IMPLEMENTATION OF SYSTOLIC ARRAYS
USING FINITE RING ARITHMETIC

by

Martin Jaekel

A Thesis
Submitted to the Faculty of Graduate Studies through the
Department of Electrical Engineering in Partial Fulfilment
of the Requirements for the Degree of
Master of Applied Science at the
University of Windsor

Windsor, Ontario
May, 1991
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ABSTRACT

The purpose of this thesis is to design an automatic layout generator for high speed systolic VLSI integrated circuits using the Residue Number System. This entire process can be completed with minimum user intervention. The WINDsor SILicon Compiler (WINDSILC) is part of a system to automate the design of high speed application specific VLSI chips. It creates a user friendly environment for generating and simulating such chips. Finally, WINDSILC also provides facilities for automatically creating the files required for testing the chip after fabrication on ASIX tester.
To my Parents.
ACKNOWLEDGEMENTS

I would like to acknowledge the guidance and support provided by Dr. G. A. Jullien and Dr. W. C. Miller throughout the progress of this thesis. I would also like to thank Dr. S. Bandyopadhyay for his role as my supervisory committee member. Thanks must also go to the members of the systolic compiler group, Dimitris, Arunita, Wai and Bruce for their assistance. Finally, I would like to thank our system administrators Mr. Stephen Karamatos and Mr. Mike Drouillard for their help and patience in solving many of the problems I have encountered.
# TABLE OF CONTENTS

Abstract iv  
Dedication v  
Acknowledgements vi  
List of Figures x  
List of Tables xiii 

1 Introduction 1  
1.1 Introduction 1  
1.2 Motivation 1  
1.3 Solution Outline 2  
1.4 Thesis Organization 4  

2 General Principles of Silicon Compilation 6  
2.1 Introduction 6  
2.2 Silicon Compilers 7  
2.3 Ideal Silicon Compiler 10  
2.4 DSP Silicon Compilers 12  
2.5 Meet in the Middle Design Strategy 13  
2.6 Systolic Cells Versus Standard Cells 14  
2.7 Conclusions 15  

3 Implementation of Systolic Compiler 16  
3.1 Introduction 16  
3.2 General Outline of the Systolic Compiler 16  
3.3 Encoder 25  
3.4 FIR Filter 28  

vii
3.5 Scaling
3.6 Rounding
3.7 Mixed Radix
3.7 Conclusions

4 Automatic Layout Generation in Systolic Compiler
4.1 Introduction
4.2 SDA Environment
4.2.1 Design Framework
4.3 Customized Layout Generator Environment
4.4 Overview of Layout Generator
4.5 Specification from Transformer
4.5.1 MDL
4.5.2 EDIF
4.6 Module Generator
4.6.1 Structure Compiler
4.7 Schematics, Symbol and Block Diagram Generator
4.8 Place & Route
4.9 File Manager
4.9.1 Relative I/O Placement
4.9.2 Net Property
4.10 Generation of ASIX Test Vectors
4.11 Conclusions

5 Results
5.1 Introduction
5.2 Switch Level Simulation on BIPSP Cell
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>Circuit Level Simulation on BIPSP Cell</td>
<td>78</td>
</tr>
<tr>
<td>5.4</td>
<td>Test Results From Fabricated Chip</td>
<td>79</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusions</td>
<td>86</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Conclusion and Recommendations</td>
<td>87</td>
</tr>
<tr>
<td>6.1</td>
<td>Conclusion</td>
<td>87</td>
</tr>
<tr>
<td>6.2</td>
<td>Recommendations</td>
<td>88</td>
</tr>
</tbody>
</table>

References

Appendix I.    SKILL Programs          94
Appendix II.   C Programs               166
Appendix III.  Physical Layout         195
Appendix IV.   Schematics for FIR Filter and Systolic Cell 199
Appendix V.    User Manual             205

Vita Auctoris  216
### LIST OF FIGURES

| Figure 1.1 | Systolic Compiler | 3 |
| Figure 2.1 | Level of Design | 8 |
| Figure 2.2 | Behavioural Representation of Full Adder | 9 |
| Figure 2.3 | Structure Representation of Full Adder | 9 |
| Figure 2.4 | Physical Representation of Full Adder | 10 |
| Figure 2.5 | Ideal Silicon Compiler | 11 |
| Figure 2.6 | Different Implementation Styles for DSP Algorithm | 12 |
| Figure 2.7 | Meet in the Middle Design Strategy | 14 |
| Figure 3.1 | Overview of Systolic Compiler | 17 |
| Figure 3.2 | Overview of Designer's Assistant | 18 |
| Figure 3.3 | Overview of Transformer | 20 |
| Figure 3.4 | Overview of the Layout Generator | 21 |
| Figure 3.5 | Y chart of Systolic Compiler | 22 |
| Figure 3.6 | General Systolic Cell | 23 |
| Figure 3.7 | Overview of DSP System | 24 |
| Figure 3.8 | Block Diagram of 18 Bit to 5 Bit Encoder | 26 |
| Figure 3.9 | Block Diagram of 9 Bit to 5 Bit Encoder | 27 |
| Figure 3.10 | Internal Block Diagram of 9 to 5 bit Encoder | 27 |
| Figure 3.11 | Modified Systolic Cell for FIR Filter | 29 |
| Figure 3.12 | Single Modulus Nth Order FIR Filter | 30 |
| Figure 3.13 | Scaling Block Diagram | 31 |
| Figure 3.14 | Flow Diagram of Scaling | 33 |
| Figure 3.15 | Scaling Structure using Four Moduli | 34 |
| Figure 3.16 | Scaling Structure using Five Moduli | 35 |
| Figure 3.17 | Scaling Array for Four 5-Bit Moduli | 37 |
| Figure 3.18 | Systolic Cell used for B1 | 38 |
| Figure 4.1 | Simplified SDA/EDGE System Overview | 42 |
| Figure 4.2 | WINDSILC Fixed Menu | 45 |
| Figure 4.3 | WINDSILC Pop-Up Menu | 46 |
| Figure 4.4 | Flowchart of Layout Generator | 49 |
| Figure 4.5 | Array Structure Template | 56 |
| Figure 4.6 | Example of Structure Array | 58 |
| Figure 4.7 | Flowchart of Schematic, Symbol and Block Diagram Generator | 60 |
| Figure 4.8 | Block Diagram of FIR Filter | 62 |
| Figure 4.9 | Place & Route Design Flow | 64 |
| Figure 4.10 | Flowchart of File Manager | 69 |
| Figure 4.11 | Flowchart of Pre-Placement I/O | 70 |
| Figure 4.12 | I/O Placement File | 71 |
| Figure 4.13 | STL Code Generation | 73 |
| Figure 5.1 | Silos Simulation on BIPSP_m cell | 77 |
| Figure 5.2 | Circuit Level Simulation on BIPSP_m Cell | 78 |
| Figure 5.3 | Phi1 Clock Signal | 79 |
| Figure 5.4 | Phi2 Clock Signal | 80 |
| Figure 5.5 | Aout0 Line at 16MHz | 81 |
| Figure 5.6 | Aout0 Line at 20MHz | 82 |
| Figure 5.7 | Rise Time at an Output Line | 83 |
Figure 5.8  Fall Time at an Output Line  84
Figure 5.9  Output From a Linear Array of Cells  85
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1</td>
<td>Multiplier Values with Four Moduli</td>
<td>35</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Multiplier Values with Five Moduli</td>
<td>36</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Introduction

The objective of this thesis is to design an automatic layout generator for high speed systolic VLSI integrated circuits using the Residue Number System. The VLSI circuits being realized are high speed signal processing circuits. The entire process is done with minimum or no user intervention as opposed to the traditional custom made layout.

1.2 Motivation

The traditional approach to designing any VLSI integrated circuit is to use custom layout techniques. Many hours are required to design large integrated circuits such as FIR Filters using custom layout techniques. With the rapid approach of the ULSI era, the time it takes to layout a large DSP chip is often comparable to the lifetime of the chip itself. Thus it is becoming increasingly important to find faster, more efficient ways of designing chips. One obvious approach is to try to automate the chip design process. This is the method investigated in this thesis.

In addition to the long design time, the chances of making mistakes using custom layout techniques are high. In fact custom layout techniques lends
itself to the bottom up approach. Thus the designer can easily get confused about the overall picture of the design. Automating the design process would greatly reduce the "human error factor" and thus help to increase the reliability of the chip. Obviously there is a trade-off involved here. Using automatic layout techniques means sacrificing some amount of silicon area which can be saved by cleverly designing, dense custom layout. But as the complexity of the circuits increase, automating the layout becomes an increasingly attractive alternative.

VLSI covers a wide range of disciplines. It includes computer science topics such as knowledge base techniques, database and functional programming, electrical engineering fields such as digital and analog design, computer architecture, and physics in particular device physics. As we enter into new technologies such as Gallium Arsenide areas such as material science are falling inside the scope of VLSI. A method is needed to separate these disciplines and special skills from each other, so that it is not necessary for a single individual to be proficient in all these fields. A top down or meet in the middle approach is more suitable for achieving this separation than the bottom up approach associated with custom layouts. In the former two design methodologies it is possible to separate the system designer from the silicon design (as explained in chapter 2).

1.3 Solution Outline
Our solution is to develop a method for automatically generating VLSI chips by using a silicon compiler that has fast design time using the meet in the middle (MIM) approach. This approach integrates the advantages of both top down and bottom up design strategies. The various stages of chip design can
easily be subdivided using this strategy. The Windsor Silicon Compiler (WINDSILC) was designed in such a way as to use the MIM design strategy.

The purpose of the systolic compiler (WINDSILC), is to create an environment for the design of high speed integrated circuits. The problem with slow design time can also be solved if an automatic layout generator is used. A number of chips can be designed in this way in a matter of days while custom layout techniques could take months.

![Diagram of Systolic Compiler](image)

**Figure 1.1 Systolic Compiler**
The systolic compiler consists of three main sections. The Designer's assistant involves knowledge base techniques, prolog and DSP arithmetic. The Transformer uses formal verification methods and the layout generator makes use of CAD tools and device physics. The layout generator uses the MIM approach to make the CAD tool development portion of building large chips independent from the layout of the individual cells. The use of a special purpose systolic array [Kun'82] enables the generated chips to meet the speed and area requirements. Targeting the chips for a specialized architecture allows the design to be optimized in terms of area and speed.

1.4 Thesis Organization

Chapter 2 discusses some of the basic concepts of Silicon compilation. It mentions the three representations of the design: Behavioral, Structural and Physical representation and contains a brief description of the methods used by several present day silicon compilers.

Chapter 3 provides an outline of the systolic compiler as shown where the layout generator fits in the entire project. The hardware for the systolic compiler is described in detail. Chapter 4 discusses in detail about the layout generator and provides a description for each of its modules. This chapter also provides the filter. In addition, a comparison between reading from the chip to the circuit level simulation is completed.

Chapter 5 discusses the conclusions reached and the limitation of the current version of the systolic compiler and layout generator with possible extensions for future work.
Appendices I & II list SKILL and C programs used in WINDSILC. Appendix III shows layouts generated by the systolic compiler as well as the systolic cell itself. Appendix IV shows the schematics for Systolic Cell and FIR Filter. Appendix V is a user manual for running the layout generator.
CHAPTER 2

GENERAL PRINCIPLES OF SILICON COMPILATION

2.1 Introduction

The 1980's saw the introduction of Very Large Scale Integrated (VLSI) Circuit technology with its high chip density and complexity. The design time of such chips in many instances surpassed the chip lifetime. With Ultra Large Scale Integrated (ULSI) Circuit in the foreseeable future this problem will just be compounded.

Two approaches can be taken to reduce the design time of the chip. One is to improve Computer Aided Design (CAD) tools in aiding experienced mask layout designers with symbolic compactors, design rule checkers, etc. Another approach is to try to capture the human design knowledge, and use it in silicon compilers. This chapter deals with the general principles and background of silicon compilation which is the main focus of the thesis.
2.2 Silicon Compilers

Silicon compilers represent a translation process from a higher level description into a layout data. The higher level description can be schematics, boolean equations, register level descriptions or even algorithms. Silicon compilers contain knowledge or rules required to synthesize a higher level description into layout data.

The main advantage of silicon compilers is that they reduce design time and complexity. They reduce complexity by allowing the user to adopt a top down approach to design thus removing much of the complexity of physical representation at the higher levels of the design process.

A silicon compilation can be viewed as a translation process from the structural or behavioral axis to the physical axis in the tripartite representation of a design as depicted by the Y chart of fig 2.1. The Y-chart contains three different domains that is the behavioral, structural and physical. The farther the design is from the centre of the Y chart the more complex it becomes. The three domains are described below:
**Behavioral representation:** The behavioral domain contains the purpose of the design or the functionality. This representation can be looked upon as a black box. The only information given about the black box is the I/O information (Input and output pins) and the function describing the relationship between the input and output pins as described in Figure 2.2.
.MODULE full_add a_in b_in c_in / s_out c_out;
BIT a_in, b_in, c_in, s_out, c_out;
DELAY s_out c_out:
  a_in (3,3)
  b_in (3,3)
  c_in (3,3);
FOREVER
  {WAITFOR changed(a_in) or changed(b_in) or changed(c_in);
    s_out = c_in^(a_in^b_in);
    c_out = (a_in&b_in) | ((a_in^b_in)&c_in);
  }
.EOM;

Figure 2.2 Behavioral Representation of Full Adder

Structural representation: This representation contains information about the composition of the cells and other components as well as the interconnections between them as shown in Figure 2.3. This representation bridges the gap between the functionality and the physical representation.

Figure 2.3. Structure Representation of Full Adder
Physical Representation: The physical representation ignores the purpose and functionality of the design and maps a structural component in space to mask layout. This information is used in the manufacture or fabrication of the VLSI chip shown in figure 2.4.

![Figure 2.4 Physical Representation of Full Adder](image)

### 2.3 Ideal Silicon Compiler

An ideal silicon compiler is represented in Fig 2.5 according to Gajski and Thomas [Gaj88]. It consists of four compilers viz. the system compiler, processor compiler, module compiler and cell compiler. The arc between the behavioral representation and structural representation is a synthesis process. Arcs between the structural representation and geometric representation denote the placement and routing phase.
Cell compilers synthesize cells into gates and transistors and then transform these gates and transistors into a mask layout. A cell is viewed as a circuit of SSI or MSI complexity. Module compilers generate regular or semi-regular arrays. Examples of modules are PLAs, ROMs, RAMs, etc. A processor compiler decomposes each process into a set of modules. A system compiler decomposes a program or an algorithm into a set of communicating processes.

Figure 2.5 Ideal Silicon Compiler
2.4. DSP Silicon Compilers

Ideal silicon compilers are very difficult to implement. To achieve optimization in speed and area, one must target the silicon compiler to a specific algorithm. DSP algorithms are very suitable for realization using such special purpose silicon compilers [Gin'86] [Rab'87] [Dem'86] [Hil'85] [Gles'84]. Several different implementation styles for DSP algorithms are shown in fig 2.6 [Rab'88].

Figure 2.6 Different Implementation Styles for DSP Algorithm
There are basically two types of DSP systems: the compiled silicon and software programmable. Compiled silicon refers to specific architectures that are not programmable, but fixed. These architectures, for example systolic architectures, are dedicated to a specific application. They have the advantage of very high sampling rates. Programmable architectures, such as the microprocessor are flexible in that programs can be loaded into it but they have slow sampling rates. The silicon compiler developed in this thesis, the WINDsor SILicon Compiler (WINDSILC) is targeted towards high speed DSP applications in areas such as radar. Thus the systolic bit/parallel architecture is chosen. To further increase the speed of computation, all computations are performed using the residue number system (RNS). The details regarding this will be discussed in the following chapters.

2.5. Meet In The Middle Design Strategy

As the silicon world moves into ULSI technology, it will become infeasible to know both system and silicon design. Therefore there is a need to separate the two fields. Ideally the system designer should not be required to have any knowledge about silicon. This would give the system designer greater flexibility and lead to more efficient designs. This method is known as meet in the middle design methodology, and is shown in fig. 2.7 [Rab’88].

As one can see from the figure, the system designer is strictly separated from the silicon design. The system designer only needs to work at the behavioral and structural level, while the silicon people are only concerned with the silicon technology.
2.6 Systolic Cells Versus Standard Cells

A systolic architecture in most cases only contains a few cells. This has the advantage of a much smaller cell library. Designing with standard cells or macro cells, on the other hand, requires a much greater number of cells. As the technology becomes more complex, more standard cells are needed to meet the timing and driving requirements. Systolic cells only communicate with their
nearest neighbors, thus reducing interconnections or wire delay. Many other advantages are brought out with systolic cells such as the reduced need for compiler timing analysis tools, fault simulation and testability tools. If the systolic cell is designed for optimum speed, area and designed for testability and fault simulation very simple tools can be developed for it.

2.7 Conclusions

The goal of this chapter was to give a brief description of the general principles of silicon compilation. The different types of silicon compilers have been profiled. In particular DSP compilers were studied. Finally design methodologies were outlined such as meet in the middle.
CHAPTER 3

IMPLEMENTATION OF SYSTOLIC COMPILER

3.1 Introduction

The physical implementation of the systolic compiler will be discussed in this chapter. A general outline of the systolic compiler is first given. The general systolic cell with modifications for the encoder, FIR Filter and scaling is then described. A schematic and layout is shown for each of these modifications. Finally the theory and physical realization for the Encoder, FIR Filter and scaling are discussed.

3.2 General Outline of the Systolic Compiler

The systolic compiler is a software system for automatically generating VLSI chips for application in high speed digital signal processing based on systolic architectures and finite ring arithmetic. It helps the user to implement vector DSP algorithms such as FIR filters, transforms and matrix operations.

The compiler is composed of three software components as shown in fig. 3.1

i) A Designer's Assistant.

ii) A Transformer.

iii) An Automatic Layout Generator.
Figure 3.1 Overview of Systolic Compiler

**Designer's Assistant**

The designer's assistant [SAR 90] takes in user information such as filter parameters (cutoff frequency, stop band attenuation, etc). It also receives hardware information such as the maximum number of chips, dynamic range and sampling frequency. The designer's assistant produces necessary information for hardware implementation of the filter in the form of a Filter Architecture...
Design Language (FADL) file. This is essentially a structural description of the filter. The FADL file includes values of the filter coefficients, the moduli used, and the number of bits required for input to the encoder and the outputs of the scalar.

![Diagram](image)

**Figure 3.2 Overview of Designer's Assistant**

**Transformer**

The transformer [Wai'91] receives a FADL file and provides a functional model of the filter to the Automatic Layout Generator [Bau'89]. The transformer consists of three parts. The first part of the transformer is implemented using the Windsor Attribute Grammar Environment (W/AGE) which was developed in the department of Computer Science, University of Windsor, by Dr. Frost [Fro'90] [Fro'91]. W/AGE is based on attribute grammar introduced by Kereth in 1968. W/AGE facilitates the construction of program as language interpreter. The first part takes in the FADL file and extracts any connectivity information. The
connectivity is expressed in PARA which is based on Miranda, a functional programming language [Tur'85]. A behavioral model of the filter is produced by the second part of the transformer. PARA is a hardware description language. It consists of a set of higher order functions added to the Miranda Standard environment. PARA is an executable specification. By providing appropriate signal vectors one can verify the correctness of the filter realized. The third part of the transformer takes in the netlist described in PARA and translates this to Electronic Design Interchange Format (EDIF), which is a standard interchange format to describe a circuit [Edi'87]. EDIF describes the cells used in the netlist and the interconnection of these cells. The transformer also produces a Module Description Language (MDL). MDL specifies the values of the ROM contents, the filter coefficients and the moduli used and passes this information to the Automatic Layout Generator.
Figure 3.3 Overview of Transformer

The transformer consists of a set of routines developed using functional programming paradigm and an Attribute Grammars environment. Functional programming facilitates algebraic manipulation and transformation resulting in formal proofs for functional correctness [Lam'85]. The attribute grammar framework is useful for translation to foreign representation and parsing of prototype language specification [Far'84]
Automatic Layout Generator

The Automatic Layout Generator is the final component of the systolic compiler. It takes both MDL and EDIF and produces a mask layout. As shown in fig 3.4, MDL runs the layout module generator. The layout module generator generates library cells that will be used by the place and route tools. Place and route takes in an EDIF netlist and places and routes cells generated from the layout module generator as well as cells that were created by the layout editor such as I/O cells. More details regarding the Automatic Layout Generator are given in chapter 4.

![Diagram showing the process flow of the layout generator](image_url)

**Figure 3.4 Overview of Layout Generator**

As mentioned in chapter two a silicon compiler can be described in a Y chart which consist of three domains: Behavioral domain, Structural domain and Physical domain. The systolic compiler can also be mapped on this Y chart as shown in fig 3.5.
In a behavioral level description, the user is only concerned about the intended use or functionality of the design. The structural domain describes the structural components in the design while the physical domain specifies the physical implementation for each of the structural components. The designer's assistant is represented by an arc starting from the behavioral level to a structural level producing a FADL file. The transformer is represented by an arc within the same domain. It starts with the FADL description and produces a MDL and EDIF
description. Finally the layout generator takes in EDIF and MDL and produces a layout in the physical domain.

The systolic cell used in the systolic compiler is based upon a bit level inner product step processor (BIPSP<sub>m</sub>) [Maj'88] [Bir'87]. The block diagram of this cell is shown in fig. 3.6. The operation of this cell is explained by eqn. 3.1

\[ Y_{out} = Y_{in} \oplus_m A \oplus_m X_{in} \]  

(3.1)

![Figure 3.6 General Systolic Cell](image-url)
In this cell $Y_{\text{in}}$ and $X_{\text{in}}$ are 5 bit buses while A is a constant which is stored in a 8 x 4 word ROM where each word is 5 bits. One bit of $X_{\text{in}}$ (steering bit) is used to steer $Y_{\text{in}}$ around the ROM or through the ROM as required. If the steering bit is logic 1 $Y_{\text{in}}$ acts as an address to the ROM and the content of that address is sent out to $Y_{\text{out}}$. If the steering bit is logic 0 $Y_{\text{in}}$ is steered around the ROM. Five basic systolic cells implement equation 3.1. A different bit of $X_{\text{in}}$ acts as the steering bit in each cell. The words come into the systolic pipeline architecture serially, with all the bits in a particular word coming in parallel. However, the computation is done bit serially. Since this basic cell is used in the pipeline structure the latches are used for each input.

A modification to this basic cell is applied to the encoder, Fir Filter and scaling block both in functionality and physical implementation. This will be explained in greater detail in this section. The block diagram for the entire system designed by the systolic compiler is depicted in fig 3.7. The Physical representation for the systolic cell is shown in Appendix III. The physical dimension of this cell is about 2000 by 1000 (design) microns.

![Figure 3.7 Overview of DSP System](image)

The shaded boxes are components not designed by the systolic compiler. They are shown here to give an overview of the entire system. As shown in Fig 3.7 an
analog signal enters the system, it is processed and a analog signal is sent out. In this thesis the signal is digitally processed using finite ring arithmetic, in particular, using the Residue number system. The analog signal is converted to a digital signal by a A/D converter and encoded into a RNS number by a encoder. It is processed by a digital signal processor and scaled and converted to digital binary signal using mixed radix conversion [Jul'78]. Finally the digital output is converted back to an analog signal. The digital processor can implement the FIR Filter, FFT, etc. An example of a FIR Filter is shown in this section.

3.3 Encoder

The encoder maps a weighted binary number to a non-weighted RNS number, given a specified set of moduli. The mapping operation which is a modulo reduction of an S bit bit number, is shown in equation 3.2.

\[ |X|_m = \sum_{b=0}^{S-1} 2^b \otimes_m x[b] \]  

(3.2)

The block diagram for the encoder is shown in fig 3.8.
The equation being computed is

\[1 \times m = \sum_{b=0}^{8} 2^b \phi_m x[b] \oplus_m \sum_{b=9}^{17} 2^b \phi_m x[b]\]

\[= \left[ \sum_{b=0}^{3} 2^b \phi_m x[b] \oplus_m \sum_{b=4}^{8} 2^b \phi_m x[b] \right] \oplus_m \left[ \left( \sum_{b=0}^{3} 2^b \phi_m x[b] \oplus_m \sum_{b=4}^{8} 2^b \phi_m x[b] \right) \phi_m 2^9 \right]\]

(3.3)

The first two summations are performed by the top block and the last two by the bottom block. The final block multiplies the output of the bottom block by \(2^9\) and adds it to the output from the top block. Notice that the first and third summations in equation (3.3) need not be reduced modulo \(m\). This is due to the fact that the summation results in a four bit binary number which is already
reduced modulo m if m is a five bit number. In this encoder structure a binary number, containing a maximum of eighteen bits, may be encoded into a five bit RNS number. If the number of bits of the input is less than ten, a simpler structure may be used which is shown in fig 3.9.

Figure 3.9 Block diagram of 9 bit to 5 bit encoder

The generic cell in the encoder is the same as the general cell in fig. 3.6.

Figure 3.10 Internal Block Diagram of 9 bit to 5 bit Encoder
Physical Realization

The physical realization of the encoder contains 5 \( B\text{IPSP}_m \) cells per modulus. Typically we would use 4 moduli. This means 20 cells are used. On a chip this would occupy 7500\( \mu \) by 7500\( \mu \) as shown in Appendix III.

3.4 FIR Filter

The FIR filter is the middle block of fig.3.7. A \( N^{th} \) order FIR filter using residue number system is expressed by equation (3.4).

\[
|Y(n)|_m = \sum_{i=0}^{N-1} A(i) \Phi_m X(n-i) \tag{3.4}
\]

In equation (3.4) \( X \) and \( A \) are the input and constant multiplier respectively and \( Y \) is the output. \( A(i) \) represents the impulse response of the filter. The output is obtained by the convolution of the inputs and the filter coefficients. Equation (3.4) can be represented more efficiently as a bit-level inner product shown in equation (3.5).

\[
|Y(n)|_m = \sum_{i=0}^{N-1} \sum_{b=0}^{4} 2^b \Phi_m A(i) \Phi_m x^{[b]}(n-i) \tag{3.5}
\]

A single multiplication requires five \( B\text{IPSP}_m \) cells. In order to perform the convolution, an extra latch is required on the steering bit (refer to fig. 3.11). This extra latch provides the required sliding effect between the coefficients and the data. The outer summation can be implemented by one single adder or the
addition can be distributed among the cells as a partial sum [Tah '87]. This latter method is the most efficient and is used in WINDSILC.

Figure 3.11 Modified Systolic Cell for FIR Filter

A single modulus Nth order FIR filter is shown in fig.3.12.
Physical Realization

Each block contains five systolic cells. Fig 3.12 represents a single coefficient of the FIR Filter. The physical layout is shown in Appendix III. It is apparent that more than one chip is needed to realize this FIR filter. Depending on the order of the filter a chip can easily become I/O bound. This will happen if a chip contains a certain number of coefficients, where each is represented by a full set of moduli. The optimum solution for minimum area is one modulus per chip. This would require ten input pads and ten output pads (excluding lines for power, ground, clocks, etc). A physical layout for one chip is shown in Appendix III.

3.5 Scaling

The scaling scales down the dynamic range and decodes a number from RNS to binary using a mixed radix conversion (see fig. 3.13).
The internal word length of the filter is large enough to ensure sufficient precision in arithmetic operations. It would be uneconomical and unnecessary to bring this large word length to the user. The scaling is used to reduce this dynamic range. If scaling is done in the proper order a free conversion to binary can be achieved by using mixed radix conversion. Scaling in RNS is a division of an arbitrary dividend by any factor of $M$, where $M$ is a product of all the moduli expressed in equation (3.6). [Sza 67]

$$M = \prod_{i=0}^{N-1} m_i$$  \hspace{1cm} (3.6)

A division in any number system is defined by the equation (3.7)

$$x = \left[ \frac{x}{y} \right]_y + \lfloor x \rfloor_y$$  \hspace{1cm} (3.7)
where \( X \) is the dividend, \( Y \) is the divisor, \( \frac{X}{Y} \) is the integer portion of the quotient, and \( \lfloor X \rfloor Y \) is the remainder. The above equation may be rewritten as equation (3.8)

\[
\left\lfloor \frac{X}{Y} \right\rfloor = \left\lfloor \frac{X - \lfloor X \rfloor Y}{Y} \right\rfloor
\]

(3.8)

Thus, the residue representation of equation (3.8) is:

\[
\left\{ \left\lfloor \frac{X - \lfloor X \rfloor Y}{Y} \right\rfloor m_0, \left\lfloor \frac{X - \lfloor X \rfloor Y}{Y} \right\rfloor m_1, \ldots, \left\lfloor \frac{X - \lfloor X \rfloor Y}{Y} \right\rfloor m_{N-1} \right\}
\]

(3.9)

If \( Y \) is restricted to a product defined by \( M \), the division can be done very efficiently. In general scaling can be represented by the flow diagram shown in figure 3.14.
Figure 3.14 Flow Diagram of Scaling

It is also expressed by the recursive relationship in equation (3.10).

\[ \tilde{X}^{(1)} = X \]  \hspace{1cm} (3.10a)

\[ \tilde{X}_i = \left| \tilde{X}^{(i)} \right|_{m_i} \]  \hspace{1cm} (3.10b)

\[ \tilde{X}^{(n)} = \left( \tilde{X}^{(n-1)} - \tilde{X}_{n-1} \right) \left| \frac{1}{m_{n-1}} \right|_{m_i} \]  \hspace{1cm} (3.10c)
The flow diagram in figure 3.15 does not map directly to the operations realizable by the BIPSP$_m$ cell. The reason for this discrepancy is that the BIPSP$_m$ cell performs a multiplication with accumulation $x \leftarrow X + YA$ while the flow diagram requires its blocks to perform the equation $x \leftarrow [X + (-Y)] A$. This problem can be overcome by preceding the scaling and decoder block with pre-multipliers. Each pre-multiplier multiplies the incoming number by the required amount. A single BIPSP$_m$ cell is required for each pre-multiplier. For a structure that consists of four moduli, three pre-multiplier cells are required (refer to fig 3.15).

Each premultiplier is basically a $32 \times 5$ bit ROM. The Contents of the ROM can be determined before design time. Premultipliers and multiplying factors for each node in fig. 3.15 are shown in table 3.1. The variable $Y_i$ represents a constant multiplier for node $N_i$ and $P_i$ denotes the $i^{th}$ premultiplier.
<table>
<thead>
<tr>
<th>P₁</th>
<th>( \frac{1}{m₀} )</th>
<th>m₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₂</td>
<td>( \frac{1}{m₀} )</td>
<td>m₂</td>
</tr>
<tr>
<td>P₃</td>
<td>( \frac{1}{m₀} )</td>
<td>m₃</td>
</tr>
<tr>
<td>Y₁</td>
<td>( \frac{1}{m₀} )</td>
<td>m₁</td>
</tr>
<tr>
<td>Y₂</td>
<td>( \frac{1}{m₀} )</td>
<td>m₂</td>
</tr>
<tr>
<td>Y₃</td>
<td>( \frac{1}{m₁} )</td>
<td>m₂</td>
</tr>
<tr>
<td>Y₄</td>
<td>( \frac{1}{m₀} )</td>
<td>m₃</td>
</tr>
<tr>
<td>Y₅</td>
<td>( \frac{1}{m₁} )</td>
<td>m₃</td>
</tr>
<tr>
<td>Y₆</td>
<td>( \frac{1}{m₂} )</td>
<td>m₃</td>
</tr>
</tbody>
</table>

Table 3.1: Multiplier values with four moduli.

The scaling structure for five moduli contains four pre-multipiers as in Fig 3.16.

![Scaling Structure Using Five Moduli](image_url)

Figure 3.16 Scaling Structure Using Five Moduli
The values $P_1 - P_3$ and $Y_1 - Y_6$ for multipliers for five moduli are shown in table 3.1 while values $P_4$, $Y_7 - Y_{10}$ are shown in table 3.2.

<table>
<thead>
<tr>
<th>P4</th>
<th>$\frac{1}{m_0}$</th>
<th>$\frac{1}{m_1}$</th>
<th>$\frac{1}{m_2}$</th>
<th>$\frac{1}{m_3}$</th>
<th>$\frac{1}{m_4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y7</td>
<td>$\frac{1}{m_0}$</td>
<td>$\frac{1}{m_1}$</td>
<td>$\frac{1}{m_2}$</td>
<td>$\frac{1}{m_3}$</td>
<td>$\frac{1}{m_4}$</td>
</tr>
<tr>
<td>Y8</td>
<td>$\frac{1}{m_1}$</td>
<td>$\frac{1}{m_2}$</td>
<td>$\frac{1}{m_3}$</td>
<td>$\frac{1}{m_4}$</td>
<td></td>
</tr>
<tr>
<td>Y9</td>
<td>$\frac{1}{m_2}$</td>
<td>$\frac{1}{m_3}$</td>
<td>$\frac{1}{m_4}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y10</td>
<td>$\frac{1}{m_3}$</td>
<td>$\frac{1}{m_4}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Multiplier values with five moduli

A viable physical design for four 5 bit moduli scaling is shown in fig 3.17 [Maj'88].
Figure 3.17 Scaling Array For Four 5-bit Moduli

The lightly shaded blocks represent 5 linearly connected cells. The solid block represents just 5 linearly connected latches. The line superimposed over the half tone block is the recurrent parallel data path or is X bus of BISP$_m$ of fig 3.18.
The line between block1 and block2 implies block1 uses the steering bit of block2 rather than its own. The output is shown at the bottom of the array.

The systolic cell for B3 - B6 uses the general cell as shown in figure 3.6. A modification was made to B1 & B2 since B1 steering bit comes from B2 thus an extra bit must be added to B1 as shown in fig. 3.18.

Figure 3.18 Systolic Cell Used for B1
There is no functional difference between B2 and B3 - B6 but there is a physical difference because B2 must share its steering bit with B1.

3.6 Rounding

Usually an integer division implies a truncation. A truncation can cause large errors. This is because the bias is always to one side and can accumulate when a large number of operations are carried out. This problem can be remedied to a certain extent by using rounding rather than truncation. The constant $\gamma = (m_1, m_2)/2$, where $m_1$ & $m_2$ are the moduli being scaled, is used to perform rounding to the nearest integer. This can be implemented by adding to the input $X_1 \rightarrow X_4$ by modifying pre-multiplier ROMs. This is shown by mapping the input $X_1, X_2, X_3, X_4$ to the scaling $\tilde{X}_1 \tilde{X}_2 \tilde{X}_3 \tilde{X}_4$

\[
\begin{align*}
\tilde{X}_0 &= (X_1 \#_{m_0} \gamma) \#_{m_4} P_0 \\
\tilde{X}_2 &= (X_2 \#_{m_2} \gamma) \#_{m_4} P_2 \\
\tilde{X}_3 &= (X_3 \#_{m_3} \gamma) \#_{m_4} P_3 \\
\tilde{X}_4 &= (X_2 \#_{m_4} \gamma) \#_{m_4} P_4
\end{align*}
\]  

(3.11a)  
(3.11b)  
(3.11c)  
(3.11d)

variables $\tilde{X}_0$ to $\tilde{X}_4$ is shown in figure 3.15 & 3.16.

3.7 Mixed Radix conversion

The structure in fig 3.16 and 3.17 contains mixed radix conversion which is an efficient method to convert RNS to binary. In order to acquire this efficient
conversion to binary using the given architecture, a few simple rules must be followed:

a) The binary output can't be more than 10 bits.
b) 32 must be chosen as one of the moduli.
c) The output from modulus 32 of the FIR Filter stage is fed to the second last line in the scaling structure. Mixed Radix representation is described by equation (3.12):

\[ X = a_N \prod_{i=1}^{N-1} m_i + a_3 m_1 m_2 + a_2 m_1 + a_1 \]  

(3.12)

which simplifies for a five bit modulus to

\[ X = a_2 m + a_1 \]  

(3.13)

\[ = a_2 \times 32 + a_1 \text{ (for } m=32) \]

a2 represent the five most significant bits because the radix is 32 while a1 represents the five least significant bits. a1 and a2 are shown in fig 3.6 & fig 3.7 and are expressed in equations (3.14).

\[ a1 = \left| x \right| \bmod m \]  

(3.14a)

\[ a2 = \left| \frac{x-a1}{m} \right| \]  

(3.14b)

3.8 Conclusions
The intention of this chapter is to give a theoretical and physical explanation of the hardware used in the filter. The functional and physical representation of the systolic cells were discussed.
CHAPTER 4

AUTOMATIC LAYOUT GENERATION IN SYSTOLIC COMPILER

4.1 Introduction
This chapter is concerned with the implementation of the Automatic Layout Generator. First the SDA environment and some of its important features used in the development of the systolic compiler are discussed. Next detailed descriptions of the different components of the layout generator are given. Finally a method to convert test vectors to the ASIX format is shown.

4.2 SDA Environment
The WINDsor SILicon Compiler (WINDSILC) was developed using the SDA environment provided by Cadence. In order to understand the silicon compiler one must have a good understanding of the environment being used. This section gives a short overview of the SDA environment and provides the necessary background for understanding the remainder the chapter.
4.2.1. Design Framework

SDA is a set of individual software components tied together by a single user interface, called the SDA Design Framework, allowing a complete circuit design, simulation layout and specification cycle [Sda'39].

![Figure 4.1 Simplified SDA/EDGE System Overview](image)

The purpose of SDA Design Framework is to provide the user with one consistent graphical user interface, through which we access the different components of the SDA system which are the individual windowing system (mouse and keyboard), and three types of cursors;

a) main cursor
b) snap cursor, snaps to grid closest to main cursor.
c) Text cursor appears only in an unix video|es or text entry window.

**SDA Database:** The SDA database contains the design data such as the layout, the schematic and the extracted layout, etc. It uses a hierarchical data structure of directories and files within the unix system. The generic form of the design directory hierarchy in the SDA system is of the form *Block/Representation/Revision*.

A *Block* is a design directory for the particular chip that is being designed. It contains one or more directories below called representations. A *Representation* is a design type directory such as layout, schematic, etc. It contains one or more files called revisions. It may also contain one or more sub-directories. A *Revision* is a file that contains the actual design data. The most current data is always saved in the revision file named *current*. The previous revision is named *backup*. An example of the directory hierarchy within the SDA database is shown below.

![Diagram of directory hierarchy]

In this example *inv* is the blockname, *schematic* is the representation and *current* is the most recent revision.
Conversion Programs: The Conversion Programs are translation programs to get data in and out of the SDA system. The SDA system contains conversion programs such as GDS II <-> SDA and EDIF <-> SDA. Storage formats other than Cadence database format include Electronic Design Interchange Format (EDIF), calma GDSII and MIF.

Graphics Editor: The GraphicsEditor allows one to create layouts, schematics, menus, netlists and extractors. After creating schematic and layout representations of the circuit a simulation is run on it. The Simulation Environment (SE) controls the entire simulation process and provides a graphical display of results.

Place & Route: Powerful and efficient tools such as Place and Route can place and route cells automatically. More about this tool will be explained later. After completing a layout, its correctness can be checked by using SDA Design Rule Checker.

4.3 Customized Layout Generator Environment
The layout generator requires a large amount of data or information. It also produces schematics, block diagrams and reports. In order to make the environment user friendly, several pop up and fixed menus were designed and added to it. When a user first enters Cadence, four desks appear. In WINDSILC one of the desks contains the WINDSILC logo, another desk contains a unix window, the third one contains fixed menus for WINDSILC and the fourth is empty. It is recommended that most compiler jobs run in the third desk. The fixed menu in this desk is shown in figure 4.2.
Figure 4.2  WINDSILC Fixed Menu

This menu contains results for the encoder, FIR Filter and scaling section. Each section, except for FIR filter contains four icons. The encoder and scaling sections contain icons for block diagram layout, chip information and I/O information. The FIR Filter section has an icon for Block diagram and another one for chips.

To start up the layout generator one must go to the Design Manager menu (Pop up menu) and click on WINDSILC. A flowchart should appear as shown in Figure 4.3.
There are two modes to the layout generator indicated by the two paths coming from the WINDSILC icon in figure 4.4. The path to the left requires the user to enter moduli and coefficients. The path on the right allows these values to be calculated by the Designer’s Assistant and Transformer. The reason for these two different modes is to allow the user to informally "experiment" with his own coefficients and moduli. Obviously, in this case, the user will not get the Designer Assistant's help and formal verification is not performed on the design.

When the user clicks the WINDSILC icon, a window form will appear as shown below.

*Do you want to use Designer’s Assistant and Transformer?*

*Enter chipname=*


FIR Filter type (lowpass, highpass or bandpass)=
Enter purpose of chip (encoder, FIR Filter or scaling)=
Enter chip number=
1'st cutoff frequency in hertz=
2'nd cutoff frequency in hertz=
sampling rate in hertz=
wordlength of input=
wordlength of output=
# of moduli=
# of coefficients=
cancel generation=

If the user wants to use the coefficient and moduli from the systolic compiler
then he should answer yes to the question "Do you want to use Designer's
Assistant and Transformer?". If the answer is no then the number of moduli,
coefficients, chipname, etc. must be entered. Otherwise a default value will be
used. The next step is to click on the Design information icon. The form
shown below appears on the screen.

Enter width(dx) of input pad (design microns)=
Enter width(dx) of input pad (design microns)=
Enter width(dx) of input pad (design microns)=
Enter height(dy) of input pad (design microns)=
Enter width(dx) of output pad (design microns)=
Enter height(dy) of output pad (design microns)=
Enter width(dx) of vdd pad (design microns)=
Enter height(dx) of vdd pad (design microns)=
Enter width(dx) of gnd pad (design microns)=
Enter height(dx) of gnd pad (design microns)=
Enter width(dx) of BIPSP pad (design microns)=
Enter height(dx) of BIPSP pad (design microns)=
Enter width(dx) of chip pad (design microns)=
Enter height(dx) of chip pad (design microns)=
Enter Maximum power consumption of chip (watts)=
Enter supply voltage (in volts)=
Enter technology (cmos3, cmos4)=
cancel?
The user has the option of entering the width and length of the input, output, ground, vdd pads, systolic cell and the power dissipation among other things. This information is used by the file manager to place the I/O pads relative to each other.

If the user wants to enter the coefficients and moduli the next step is to click on the moduli and coefficient icons. Depending on the number of moduli and coefficients the following window appears.

\[
\begin{align*}
\text{Enter modulus } 1 &= \\
\text{Enter modulus } 2 &= \\
& \quad \ddots \\
\text{Enter modulus } N &= \\
\text{Enter coeff1} &= \\
\text{Enter coeff2} &= \\
& \quad \ddots \\
\text{Enter coeffN} &= 
\end{align*}
\]

The user simply enters integers for the moduli and coefficients.

The final step is to click on the run compiler icon. This will start the actual generation of the chips using Place & Route.

4.4 Overview of Layout Generator

The layout generator is the component responsible for converting the structural description of a chip provided by the transformer into an actual layout. This comprises the heart of the systolic compiler.

Fig. 4.4 shows the various modules of the automatic layout generator as well as the outputs obtained from these modules. The ellipses represent different
modules and the rectangles show their outputs. Three of these modules, namely, the Place & Route, the PDCompare and DRC were already provided by Cadence. All other modules (indicated by shaded ellipses) were created by the author. These modules are described in this chapter in sections 4.6 to 4.10. The modules were created and bound together by procedures or functions written in a programming language called SKILL.
SKILL [Wood'86] is a powerful graphical interpretive functional language which allows one to create any procedure that can be performed interactively on the system. Its syntax is very close to that of C. SKILL also enables one to design sophisticated procedures by providing direct access to the design database and the full Design Framework toolset.

The layout generator receives both Module Design Language (MDL) and Electronic Design Interchange Format (EDIF). MDL and EDIF will be explained in more detail in section 4.5. The following modules receive only MDL: Schematic, Symbol and Block diagram generator, Module generator and File Manager. Place and Route is the only module that accepts EDIF.

Schematic, Symbol generator provides Schematic and symbol representations for the Physical Design Compare Module. The block diagram generator produces block diagrams which give the user a clear and concise description of the filter realization. The user only needs to study the block diagram to acquire sufficient information about the design.

The Module Generator [Six'86a] [Six'86b] produces library cells for place and route. The main constituents of this module are SKILL programs and the structure compiler. The Layout editor is used to create custom made layouts of individual cells. These layouts should be created before running the layout generator.

Place and Route is provided by Cadence. This tool is meant to be an interactive tool. Programs written in SKILL are used to automate Place and Route. This module accepts an EDIF design which is converted into SDA
database format. Place and Route requires library cells. Each library cell is created by the module generator and the layout editor tool. It uses a preplacement file to place Input/Output (I/O) pads relative to each other. A net information file assigns properties to critical nets. For example the clock net is the longest and contains the most capacitance, thus this net will have a higher priority for placement than other nets. This will be stated in the net information file.

The File Manager is used to generate preplacement and net property files automatically. This module reads a design information file to determine the physical dimensions of the systolic cells and I/O cells. It also reads the maximum perimeter of the chip and power dissipation. Based on these parameters it creates the preplacement and net property files.

The verification tools are used to do circuit level and gate level simulations on the filter layout. HSPICE is used for circuit simulations and Silos for Gate level simulations. The circuit is verified by comparing the expected values to the actual values. This procedure is accomplished by writing a file in a Stimulus Test Language (STL) Format.

The test vectors used for circuit level and gate level simulations are converted to ASIX format. Thus the same test vectors that were used to test the mask layout on Cadence can be used to test the actual chip. This allows a comparison to be made between the actual results and the expected results based on HSPICE models.
4.5 Specification from Transformer

This section discusses the two different types of inputs (MDL & EDIF) that are taken from the transformer.

4.5.1 MDL

The Module Design Language contains information such as the ROM contents for each individual BiPSPl cell. It also contains the purpose (encoder, FIR Filter, scaling) and the name of the chip. An example of a MDL description of a single chip is shown below.

Purpose: "FIR-Filter"
Chipname: "Chip 1"
Moduli: '(32)
Rom: '((2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1) (4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 1 2 3) (8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 1 2 3 4 5 6 7) (16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 1 2 3 4 5 6 7) (0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15) (0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15) )

The moduli and the contents of the ROM are given in a list format. In this example, the modulus is 32. There is one coefficient in this example with a value of 2. Since there is one coefficient and one modulus the contents of five ROM's must be specified. The following information is extracted from MDL.

i) Purpose of chip
ii) Name of chip
iii) Moduli values
iv) Rom content values
v) Number of moduli
vi) Number of coefficients
vii) Coefficient values

This information is sufficient along with EDIF to run the layout generator.

4.5.2 EDIF

Electronic Design interchange Format (EDIF) is a language to describe and interchange electronic design information regarding mask layouts netlists, etc.

WINDSILC uses the EDIF netlist option. The netlist provides connectivity information and states the cells used. These cells are defined in an external library. WINDSILC creates these cells with the module generator and layout editor. A typical format of EDIF is shown below.

```
{external library}
{design library}
{cell definition}
   {interface}
   {instances}
{nets}
```

The external library contains information about any cell referenced by the netlist. The external library only needs to specify the I/O pins and the name of each library cell. The design library contains cells created by the user. The Cell definition contains the name of the cell being designed, while the interface holds I/O information of the cell. The instance field contains the instances used in the cell. For example a NAND gate may be used several times in a design. To
differentiate between the NAND gates, different instance names are assigned to each gate. The interconnection of instances is provided in the nets section. The format shown in the above example is a flat netlist.

4.6 Module Generator

The purpose of the module generator is to create library cells for Place and Route. It accomplishes this by programming each BIPSP\textsubscript{m} cell and generating arrays or semi arrays of systolic cells. By exploiting the regular structure of systolic arrays, dense layouts can be produced from the module generator. An example to generate one IPSP cell library is shown below.

\begin{verbatim}
Purpose: "FFT"
Chipname: "Chip 1"
Instance: 'IPSPI'
Rom: '((2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 21 22 23 24 25 26 27 28 29 30 0 1))
\end{verbatim}

The module generator is composed of SKILL procedures and uses a structure compiler. The SKILL procedure is used to extract relevant information from MDL and run the structure compiler.

4.6.1 Structure Compiler

Regular and Semi-regular structures are being used more widely in integrated circuits [Bam'85]. Cadence provides a very powerful tool to automate the layout of such structures. This tool, the structure compiler, uses an extremely compact data format to store these structures. Thus the problem with limited data storage can be avoided.
The structure compiler is a tool that automatically generates a layout from an array structure template, a library of master cells and an optional personality file. There are two types of arrays: homogeneous arrays and heterogeneous arrays.

The *Homogeneous array* only contains one master cell. The only information needed to compile this array is the master cell and the number of rows and columns. *Heterogeneous array* also called complex array, contains more than one master cell. A personality file is used to specify the relative positions of these cells in the array.

The Array Structure Template provides the block diagram (fig 4.5) of the layout to be produced from the structure compiler. The blocks in the array structure template are called array blocks. The relative positions are specified by describing which edge should abut other edges. The relative size of the array blocks do not determine their actual locations.
Each array block has the following parameters:

**Name Field**: This indicates the array blockname.

**Type Field**: This indicates the type of array; homogeneous or heterogeneous.

**Personality Field**: The personality field specifies a personality file which is used only for a heterogeneous array. This file specifies which masters cells are used in the heterogeneous array and the relative positioning.

**Map Field**: This field provides a list of mapping from individual symbols in the personality data to strings of symbols, each of which corresponds to a single master cell. For example, the symbol 'O' is mapped onto two symbols: "N" followed by "T".
**Master Field:** The field contains the master cells used in the structure compiler. This shows a mapping between a symbol to a master cell. This defines the mapping between the symbols in the personality file and master cells.

Example:

masters: 0:rompat0 layout; 1: rompat1 layout;

**Orientation Pattern Field:** The orientation of the master cell is specified in this field. The orientation is specified for cells in the array block. The pattern is repeated throughout the array block. The following eight possible transformations can be applied to a cell.

- R0  Rotate 0
- R90 Rotate 90
- R180 Rotate 180
- R270 Rotate 270
- MX Mirror in the x-direction (about a vertical axis)
- MY Mirror in the Y-direction (about a horizontal axis)
- MR90 Mirror and rotate 90
- MR270 Mirror and rotate 270

**Procedure Field:** A procedure file is used to further manipulate the array.

**Pin Field:** A list of Pins can be promoted to a higher level. This permits one to include modules created by the structure compiler and use it in place and route. An example of a property of an array block is shown below.
There are five array structures, Bit0, Bit1, Bit2, Bit3 and Bit4, in this diagram. Each of these structures corresponds to a 32 X 1 bit ROM, and collectively they form the 32 X 5 bit ROM of the BIPSP cell. The personality file for each structure is used to program a particular ROM. In fig. 4.6 the array template for Bit4 is shown. The corresponding personality file, Bit4.Personality, specifies the contents of a 32 X 1 bit ROM. A personality file is used because the array structure is heterogeneous. It consists of two master cells, diffcell
and empty cell denoted by 1 and 0 respectively. There was no need to supply the set and map fields in this example.

4.7 Schematic, Symbol and Block Diagram Generator
The purpose of the schematic and symbol generator is to provide schematic and symbol representations for FD compare. The user can examine the schematic rather than mask layout for any detailed information such as the construction of BIPSP_m cell or the ROM contents. As mentioned before, the block diagram provides general information. The block diagram improves the user friendliness of WINDSILC. A flow diagram of this module is shown in figure 4.7.
An example of SKILL functions to generate schematics and symbols is shown below.

```
Sch_rom_gen ("FIR_Filter" "test design" "mod 32" "BIPSP0" 0 12 3 ... 3l)
Sch_rom_gen ("FIR_Filter" "test design" "mod 32" "BIPSP1" 2 3 4 5 ... 1)
Sch_rom_gen ("FIR_Filter" "test design" "mod 32" "BIPSP2" 4 5 6 7 ... 3)
Sch_rom_gen ("FIR_Filter" "test design" "mod 32" "BIPSP3" 8 9 10 11 ... 7)
Sch_rom_gen ("FIR_Filter" "test design" "mod 32" "BIPSP4" 16 17 18 19 ... 15)
```
Sch_rom_gen ("FIR_Filter" "test design" "mod 32" 5)
Sch_rom_gen ("FIR_Filter" "test design" 1)

This example creates one filter coefficient consisting of BIPSP_m cells connected linearly. The function Sch_chip_gen connects the linear array to its I/O pads.

The block diagram generator reads both MDL and a parameter file. This parameter file is generated from the expert system. Information entered by the user is processed and transferred into this file. An example of this parameter file is shown below.

Cutfreq1=1000
Cutfreq2=2000
Samrate=20 MHZ
nummod=4
purpose="FIR_Filter"
coeffnum=4
inwordlen=9
outwordlen=10
chipnum=5

In this example the filter is a lowpass with a sampling rate of 20 MHZ, at cutoff frequency 1000 and 2000, the number of bits encoded is 9 and number of bit scaled to is 10. An example of a generated block diagram is shown in figure 4.8.
The block diagram consists of a number of icons. The icons represent coefficient table, filter specifications, and filter architecture. The information within each of these icons is displayed by clicking the mouse button over it. Examples are given in the user manual in Appendix V.

The coefficient table contains relevant information about the number and values of the coefficients. The coefficients values are the value before converting to residue number system.

The filter specification icons contain information about the type of filter (lowpass, highpass, bandpass), sampling rate, input word length, output word length.
The filter architecture icon shows the structure of the filter. The parallel nature of the architecture is shown on the icon. By clicking the icon the block diagram of a particular coefficient can be displayed. To go deeper into the structure the user should click again on the icon. This will display the block diagram of each individual \( \text{BIPSP}_m \) cell.

4.8 Place&Route

Cadence place and route features tools for the design of Application Specific Integrated Circuits (ASICs) and high volume integrated circuits using cell libraries [Dill'88]. The Place & Route tool uses powerful algorithms such as the cluster placement and contour routing. Thus denser layouts can be achieved than with conventional systems. Place & route allows designs containing up to 10,000 cells to be routed. It also supports a variety of chip topologies such as straight-row cells, mixed and standard cells and macro cell IC's. The design flow of Place & Route is depicted below in fig 4.9. Each module of the flowchart is discussed in this section.
Figure 4.9 Place & Route Design Flow

From the flowchart it can be seen that the cell library data must be created along with the connectivity information before Place & Route can be activated.
The Cell Library must contain all library cells referenced in any design used by Place & Route. If a schematic is being used to capture a design, and the connectivity information obtained from the schematic capture is being utilized by Place & Route, then a minimum of three representations are necessary. These are the symbol, the layout and the abstract. A symbol is an external view of a block and contains only terminal information. It does not contain any internal information such as a schematic representation. A physical representation is used for the final mask. Abstract is the representation used by Place & Route. It contains only pin and boundary information from the layout. If the design is entered as an EDIF netlist and flattened, then only two representations are needed: layout and abstract.

Special cells may also be included in the cell library. These are not referenced in the netlist but are used by the special routing function. There are three primary "glue" cells: I/O corners, Feedthrough cells and Fill cells. The I/O corners cell is used to complete the power bus bends in the corner of the I/O frame. A Feedthrough cell is placed in the standard cell row to allow routing across the row. This reduces routing net length and routing area. A Fill cell is identical to the feedthrough cell but has no terminal to route across the standard cell row. This cell type replaces feedthrough cells that are not assigned to nets after feedthrough insertion.

Two technology files are used to describe the basic design rules for the place and route system. The Rule.tech is a technology file that describes the routing layers, their width, separators, parasitic elements, etc., while the contact.tech file describes the contacts used during routing.
A design must be entered in the SDA Design Framework in a netlist. This netlist describes the cells used and the interconnection between these cells. The netlist can be captured by a schematic or a netlist can be a EDIF netlist. Both of these netlists must be transferred to autoLayout rep which is a flat netlist version stored in SDA database.

A net has certain properties associated with it. The main properties are the type, priority and width. The net type property determines how it is treated by the place and route system. The basic net type are supply, ground and signal. The net priority property is used to provide more importance to a higher priority net. This means the router will route a higher priority net before a lower priority net. The net width property defines a maximum width a net can have. These properties can be defined in a netlist being read in or they can be updated by the place and route algorithm. Most of the time the net type is set in the netlist and priorities are set before placement to get a better placement. The net property file is created automatically by the layout generator during run time.

The placement of a design is basically done in two steps: I/O circuit placement and cell placement. The I/O circuit can be placed from a text file, automatically or manually. In WINDSILC the placement of the I/O cell is done through a text file that is automatically created by the file manager. Placement is done both interactively and automatically. A powerful placement algorithm called cluster is used for automatic placement. SDA also added an automatic macro cell placement. This placement algorithm handles blocks with large variations of shapes and sizes.
The Channel Generation algorithm [Dai'85] divides the chip into two areas. One area is where the cell are placed and the other area is where the channels are routed [Yos'82]. There are two types of channels: one that is routed vertically and the other one is routed horizontally. These channels can be rectangular or rectilinear in shape. There are pins on the border of this channel. A horizontal channel has pins with access direction up and down while a vertical channel has pins with access directions left and right. The channel generation algorithm is an integral part of Place & Route and is thus utilized in the systolic compiler.

In the global routing phase all nets are assigned a channel and global routing optimization is performed. There are three type of routing styles a global router can take. They are Expansion Routing, Tree Routing and Bias File Routing or Enhanced Tree Routing. Expansion Routing is a generalized Lee router [Rub'74] [Hig'83] that expands from one channel into another using cost functions such as distance or congestion. Tree Routing is mainly used by the power and ground nets. The router follows the tree structure of the channels, connecting to pins on one side of the channel. If there are power and ground nets on opposite side of the channel boundary then a planar single layer wiring of these nets will be done. If routing is not possible with Tree Routing then Expansion Routing is done. Bias File routing or Enhanced Tree Routing strategy gives the user the capability to specify on a net per net or channel bias per channel whether to use Expansion Routing or Tree Routing. Once global routing has assigned all nets to channels then detail routing is performed.
The *detail router* [Deu'76] [Heyn'82] creates the final chip layout by routing channel by channel. The router goes through each channel, compacts or expands the channel, propagates pins to adjacent channels until the entire chip is routed and automatically compacted. The detail router routes in three distinct routing styles: Straight routing, Manhattan routing and 45 degree contour routing. The straight routing on primary layers is made as straight as possible. The manhattan router bends wire(90 degree) to route around contacts and uneven channels contours. The 45 degree contour router bends wires on 45 degree angles to achieve shorter net lengths.

4.9 File Manager

The File Manager produces a number of files necessary for viewing Place and Route. These files are preplacement file, net property file and possibly an EDIF file depending on whether the user wants to enter coefficients and moduli directly to the layout generator or generate them through the expert system. If the user chooses to enter Moduli and coefficients directly to the layout generator the file manager will produce the EDIF netlist. The file manager also reads a design information file which contains physical dimensions of I/O pads, systolic cells, etc. A flowchart of the File Manager is shown in fig. 4.10.
4.9.1 Relative I/O Placement

The preplacement file contains the relative placement of I/O pads. This file is generated by receiving only the number of coefficients and moduli along with the design information file (refer to fig. 4.11)
The parameter of the I/O frame can be calculated from the number of input, output, vdd and gnd pads and size of the I/O pads. The I/O pads are placed in the proper position taking into account the Aspect ratio. The Aspect ratio is calculated from the dimensions of the core of chip. The area and parameter of core are calculated from the number of rows and columns of systolic cells.

Although Place and Route would still run if these properties were not specified the resulting chip would not function. So it is very important that the properties be assigned correctly. An example of the relative I/O placement file is shown in figure 4.12.
In this example there are four input and output pads, one vdd and one ground pad. The two inputs INA, INB are placed to the left. INC, IND and gnd placed at the bottom. Output OUTA and VDD placed to the right OUTD, OUTC and OUTB placed at top.

4.9.2 Net property

The purpose of the net property file is to provide additional properties for special nets. For example vdd and gnd nets have different widths in the I/O frame and core of chip. This requires the net to be split and given different widths. This can be specified in a net property file. An example of net splitting is shown below.

```
NET "gnd!" PRIOR 115 R.WIDTH 0.000000 MAX.LEN 0.00000 CAP 0 TYPE GROUND Conn RELATED_NET EON
```
Vdd and gnd are the main nets. They are split into VddInt, VddRing, GndInt, GndRing respectively. The "Int" represent internal nets while the suffix Ring represent I/O Ring nets. The internal net width in this example is 5 microns for both ground and vdd. The Ring net for vdd is 181 micron while the ground net is 166 micron.

4.10 Generation of ASIX Test Vectors.
In this section the generation of test vectors for the ASIX tester is discussed. As mentioned before STL is a higher level language that produces test vectors for HSPICE, SILOS, and other simulators and testing equipment. Since, the ASIX tester will be used to functionally verify the fabricated chips it is beneficial to write a procedure to convert the STL test language to ASIX format.

In order to convert STL to ASIX three functions or procedures must be created: deftest, genvec and endtest. The information passed to these functions is shown in figure 4.13.
It is necessary to create three specific functions, `deftest`, `genvec` and `endtest`, because the code generation is operating under open simulation system which automatically looks for these functions. Cadence provides detailed descriptions of the steps to be followed for generating these function. Even if such facilities were not provided by Cadence, the ASIX test vectors could still be generated. However a much larger and more complex SKILL program would then be required to achieve the same results.

From figure 4.13 `deftest` receives the pin definitions, clock definitions and pulse definitions from the STL program. `Deftest` takes these definitions and
writes the pins, clocks and pulses in a ASIX format. For example, an STL code for defining pin name "A" is: defpin A in. The general ASIX format is

PIN [I/O] [Device Pin] [Tester Pin] [Vector Bits] [Name]

For example, PIN I 1 1 80, 1 "A"

The function deftest extracts direction and pin name from the defpin construct.

The next function in figure 4.13 is genvec function. It receives the time steps and stimulus values. The time steps are extracted from the deftiming construct of STL. The general format for the deftiming construct is:

deftiming [resolution] [strobe time] [vector time]

For example, deftiming 1ns 1ns 100ns

The vector time is the time period for each vector. From the vector construct the stimulus values are obtained. The vector construct is shown below.

\[
\begin{align*}
\text{xv(1 0)} \\
\text{xv(0 1)} \\
\text{xv(1 0)}
\end{align*}
\]

The stimulus values are converted to ASIX equivalent logic value characters. A logic 0 is equivalent to character A, and logic 1 is equivalent to character I for an input pin. Logic 0 and logic 1 for an output pin are equivalent to characters S and Q respectively. The last function, endtest, receives the endtime and the number of vectors generated.

Three files must be created on the ASIX tester. These are filename.tvf, filename.dif and filename.tpf. Filename.tvf is a test vector file. ASIX reads this file for the input vectors and expected output vectors.
Filename.dif is a device definition file. The name of the chip is recorded along with the date, pin names and directions and duct numbers for pins.

Filename.tpf is the test pattern file. This file contains a list of programs to test the chip. It checks for short circuit to ground or power, the functionality, etc.

The files filename.dif and filename.tpf are produced by the code generator. The user creates the test pattern file. Also the user supplies the expected output vectors in the test vector file.

4.11 Conclusions

The purpose of this chapter was to present and explain the different modules used to make up the layout generator in the Cadence environment. Descriptions of the built-in modules as well as of those created by the author were given.
CHAPTER 5

RESULTS

5.1 Introduction

This chapter discusses the results obtained from switch and circuit level simulations on the BIPSP cell. It also reports on the performance of an actual chip which was designed using the systolic compiler and fabricated. The test data obtained from the fabricated chip show that the systolic compiler is able to design chips which operate satisfactorily at the desired speeds.

5.2 Switch Level Simulations on BIPSP Cell

The SILOS [Sil'88] simulator was used to perform the switch level simulations on the BIPSP cell. These simulations were carried out mainly to verify the functional correctness of the circuit. They are not concerned with the timing or delays associated with the circuit components. The waveforms obtained from a typical SILOS simulation are shown below in figure 5.1.
Fig 5.1 SILOS Simulation of BISPm Cell.

The waveforms shown here correspond to the two phase non-overlapping clocks (GPHI1 and GPHI2) and the Yin and Yout buses (shown here as Ain and Aout) of the BISP cell in figure 3.6. The output is latched when GPHI1 is high and the input is latched into the cell when GPHI2 is high. For example, at 260ns GPHI2 is high and the data on the Ain bus, 00001 (1), is ready to be latched in. At 300ns GPHI2 is high and the output is latched on to the Aout bus. The value of Aout at this time is 01011 (11). In other words the contents of address location 1 is 11.
5.3 Circuit Level Simulation on BIPSP Cell

A circuit level simulation using HSPICE [Hsp'88] is shown below. Fig. 5.2 shows the waveforms obtained from the HSPICE simulations of the BIPSP\textsubscript{m} cell. Here GPHI1 and GPHI2 represent the two phase non-overlapping clocks, \textit{Ain0} - \textit{Ain4} represent the five bits of \textit{Yin} (ref fig 3.6) which acts as the address to the ROM and \textit{Aout0} - \textit{Aout4} represent \textit{Yout} of the BIPSP\textsubscript{m} cell. It can be seen from this figure that the clock period is 40\textit{ns}. This means that the circuit functions at 25 MHz. During the \Phi_1 phase data is loaded into the BIPSP\textsubscript{m} cell and during the \Phi_2 phase data is taken out. The \textit{Aout} corresponding to a particular \textit{Ain} comes out one clock period after \textit{Ain} has entered the cell. For the given example, it can be seen that the contents of the ROM corresponding to location 10101 is 11111.

Fig 5.2 Circuit Level Simulation of BIPSP\textsubscript{m} Cell.
5.4 Test Results From Fabricated Chip

The fabricated chip was tested on the ASIX tester. A number of waveforms corresponding to various signals on the chip are shown in this section. These waveforms were obtained directly from measurements taken with the Tektronix digital oscilloscope. Figures 5.3 and 5.4 show the two phase non-overlapping input clock signals to the chip. In this example the clocks are running at 20MHz.

![Phi1 clock at 20MHz](image)

Figure. 5.3 Phi1 Clock Signal.
Figure 5.4 Phi2 Clock Signal.

The data on a single line of the Aout bus of a BIPSP cell is shown at two different frequencies, 16MHz and 20MHz in figures 5.5 and 5.6 respectively. These outputs demonstrate that the cell operates satisfactorily even at speeds of up to 20MHz. The simulations done on the cell indicated that the chip would operated at a maximum speed of around 22-23MHz. From the actual test results we found that the chip operates at 20MHz, but fails at 25MHz. So, the simulations and actual results agree quite well in this case.
Output from BIPSP cell at 16MHz

Time(sec)

Volts

Aout0

-2.00e-7 -1.00e-7 1.29e-26 1.00e-7 2.00e-7 3.00e-7 4.00e-7 5.00e-7

Figure 5.5 Aout0 Line at 16MHz.
Output from a BIPSP at 20MHz

Figure 5.6 Aout0 Line at 20 MHz.

The delays associated with the cell can be obtained by looking at the above two figures more closely. The rise and fall times at the output are shown below in
figures 5.7 and 5.8 respectively. These measurements were taken at an operating frequency of 20MHz.

![Rise Time of BIPSP cell](image)

**Figure. 5.7 Rise Time at an Output Line.**

It can be seen from fig. 5.7 that the rise time in this case is approximately 30ns.
Fall Time of BIPSP cell

Figure 5.8 Fall Time at an Output Line.

From fig. 5.8, the fall time at an output line is approximately 22ns. It should be kept in mind that the rise and fall times reported here were measured at an output pin on the chip. This corresponding values within the chip should be even smaller.
So far we have discussed results from a single BIPSP cell. The final waveform shown in fig. 5.9 shows the output obtained from an array of six such cells. The entire chip was designed by the systolic compiler and tested at 16MHz.

**Output from array of 6 BIPSP cells**

![Graph showing output from array of 6 BIPSP cells.]

*Figure 5.9 Output From a Linear Array of Cells.*
5.5 Conclusions

The purpose of this chapter was to present the results of simulations as well as actual test results obtained from a fabricated chip. These results demonstrate the viability of using the systolic compiler for generating high speed DSP chips. One such chip was fabricated and tested and found to work satisfactorily at speeds of upto 16MHz.
CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

An automatic layout generator for high speed VLSI systolic circuits has been implemented. This layout generator is part of the systolic compiler developed at the University of Windsor for automating the design of high performance VLSI chips in the area of digital signal processing. The silicon compiler was developed within the Cadence environment. Various VLSI tools were used to create the modules in the compiler. The Cadence programming language, SKILL, was used extensively along with a number of C programs. The following modules were developed as part of the complete systolic compiler.

1) Module Generator
2) Schematic and symbol generator
3) Block diagram generator
4) File Manager
5) Conversion between test vectors to ASIX test vector format.
6) Silicon compiler environment
The user interacts with the layout generator through popup and fixed menus. He has the option to enter moduli and coefficients or let the Designer's assistant and Transformation section of the systolic compiler provide the necessary information.

The user can view the mask layout or report files through menus. He can also substitute the systolic cell or I/O pads with other ones by supplying the physical dimensions of the new cells through a popup menu.

A facility is provided whereby the test vectors used to simulate circuit and switch level designs can be converted to ASIX format. Thus a comparison between circuit level simulation and the actual chip measurements can be made.

6.2 Recommendations

The current version of the layout generator receives a FIR filter description from MDL. In order to accommodate other DSP applications MDL & FADL must be made more flexible. Thus a modification to the MDL & FADL languages is required.

The present version of the layout generator is a cell based silicon compiler. It places, programs and interconnects cells. The cells must exist in the library beforehand. This method is called meet in the middle approach. It was adopted by Cathedral II and WINDSILC to separate the CAD development discipline from the details of device physics. As we move into the VLSI era and approach giga hertz clock rates the number of library cells increases to meet specific needs of the technology. It would become unfeasible to store
thousands of library cells. A method to create a layout down to the transistor level is required. Thus an extension of the cell based to transistor based layout would be useful.

The systolic compiler presently captures high level design through the designers assistant which uses Prolog. An improvement to the systolic compiler would be to make it capable to accepting VHDL input. Many of the VHDL descriptions available in the commercial market can then be used in the systolic compiler.
References


APPENDIX I

SKILL Programs
procedure(changeRepType(block))

; This skill program converts representation
; type and representation status in the current
; window rep to logicCell and bounded respectively.

 ******************************************

prog(blockname)

; edit Representation/type and Representation/status of layout.

sprintf(blockname " cmoslib/%s layout" block
graphEdit(blockname)
setLayer(bkngd "drawing")
subAll()
thisRep= getWindowRep()
    property(thisRep,"Representation/type","logicCell")
    property(thisRep,"Representation/status","bounded")
save()

; edit Representation/type and Representation/status of abstract.

sprintf(blockname " cmoslib/%s abstract" block
graphEdit(blockname)
setLayer(bkngd "drawing")
subAll()
thisRep= getWindowRep()
    property(thisRep,"Representation/type","logicCell")
    property(thisRep,"Representation/status","bounded")
save()
)

procedure(compiler)

******************************************

; This procedure runs the compiler automatically.
;

******************************************

makeForm("Please fill in this form with appropriate values"
    (middle "Do you want to use Designer's assistant and Transformer " "Boolean")
    (chipname "Enter chipname" "filename")
    (FIRtype "FIR filter type( lowpass, highpass or bandpass)" "file" "lowpass")
    (type "Enter purpose of chip [encoder FIR_Flter scaling ]" "filename")
    (chipnum "Enter chip number" "int")
    (cutfreq1 "1st Cutoff Frequency in hertz" "int" 1000 (1 100000))
    (cutfreq2 "2nd Cutoff Frequency in hertz" "int" 5000 (1 100000))
    (samrate "Sampling Rate in hertz" "int" 2000 (1 2000000))
    (inwordlen "Wordlength of input" "int" 9 (1 18))
    (outwordlen "Wordlength of output" "int" 10 (1 10))
    (nummod "# of modulo number" "int" 4 (1 10))
'(numcoeff "# of infinite coefficient" "int")
'(cancel "Cancel generation" "Boolean" nil)
)

if(cancel then return())
  if(middle then
    return()
  else
    ModCoeff(cutfreq1 cutfreq2 samrate inwordlen outwordlen FIRtype type chipname nummod numcoeff)
    return()
  )
)

procedure( runCompileruser() )
  prog()
    load(" skill/coeff.file")
    setup_tech()
    FIRblockgen()
    openSDA("read skill/schematic.log")
    edif2sda(type chipname)
    main_array(type chipname)
    place_route(type chipname)
    ; doPDCmpare(type chipname)
  )
)

procedure( runCompilerexp() )
  prog()
    load(" skill/MDL.par")
    setupTechMid()
    ; FIRblockgen()
    ; edif2sda(purpose chipname)
    ; main_array(purpose chipname)
    ; main_schematic(purpose chipname)
    ; place_route(purpose chipname)
    ; doPDCmpare(purpose chipname)
  )
)

procedure( ModCoeff(cutfreq1 cutfreq2 samrate inwordlen outwordlen FIRtype type chipname nummod numcoeff) )

; This procedure creates a ModVal and CoeffVal functions.
; These functions will bring up a makeform window for the user
; to enter mod and coefficient values.
;
  prog()
    modCoeffFile = outfile( " skill/modCoeffFile" "w" )
    fprintf( modCoeffFile "procedure( ModCoeffVal() \n"
    fprintf( modCoeffFile " prog() \n"
    fprintf( modCoeffFile " declare(modnum[5]) \n"
    fprintf( modCoeffFile " declare(coeffnum[300]) \n"
printf( modCoeffFile "values\n")
for(modcount 1 nummod
sprint(modstring "\nEnter values for mod%d\n" int\n modcount modcount)
    printf(modCoeffFile modstring)
)
for(coefcount 1 numcoef
sprint(coeffstring "\nEnter values for coeff%d\n" int\n coefcount coeffcount)
    printf(modCoeffFile coeffstring)
)
printf( modCoeffFile "\n")
for(modcount 1 nummod
sprint(modstring "modnum%d=%d \n" modcount modcount)
    printf(modCoeffFile modstring)
)
for(coefcount 1 numcoef
sprint(coeffstring "coeffnum%d=coeff%d \n" coefcount coeffcount)
    printf(modCoeffFile coeffstring)
)
printf( modCoeffFile "cutfreq1=\n" cutfreq1)
printf( modCoeffFile "cutfreq2=\n" cutfreq2)
printf( modCoeffFile "samrate=\n" samrate)
printf( modCoeffFile "inwordlen=\n" inwordlen)
printf( modCoeffFile "outwordlen=\n" outwordlen)
printf( modCoeffFile "FIRtype=\n" FIRtype)
printf( modCoeffFile "type=\n" type)
printf( modCoeffFile "chipname=\n" chipname)
printf( modCoeffFile "nummod=\n" nummod)
printf( modCoeffFile "numcoeff=\n" numcoeff)

printf( modCoeffFile "loadmodcoeff(cutfreq1 cutfreq2 samrate inwordlen outwordlen FIRtype type chipname nummod numcoeff) \n")
printf( modCoeffFile "\n")
printf( modCoeffFile "\n\n")
close(modCoeffFile)

load(" skill/modCoeffFile")
}
}

procedure( DesignInfo() )
    prog((inputx inputy outputx outputy gndx gndy vddx vddy BIPSx BISPy chipx chipy chipPow supVol tech cancel)

makeForm( "Please fill in this form with appropriate values"
    (inputx "Enter width(dx) of input pad (design microns)" float 345.0 )
    (inputy "Enter height(dy) of input pad (design microns)" float 1072.0 )
    (outputx "Enter width(dx) of output pad (design microns)" float 445.0 )
(outputy "Enter height(dy) of output pad (design microns)" "float" 1072.0 )
(gndx "Enter width(dx) of gnd pad (design microns)" "float" 340.0)
(gndy "Enter height(dy) of gnd pad (design microns)" "float" 1072.0)
(vddx "Enter width(dx) of vdd pad (design microns)" "float" 340.0)
(vvddy "Enter height(dy) of vdd pad (design microns)" "float" 1072.0)
(BIPSPx "Enter width(dx) of BIPSP pad (design microns)" "float" 1874.0)
(BIPSPy "Enter height(dy) of BIPSP pad (design microns)" "float" 1184.0)
(chipx "Enter width(dx) of chip (design microns)" "float" 10000.0)
(chipy "Enter height(dy) of chip (design microns)" "float" 10000.0)
(chipPow "Enter maximum power consumption of chip(watts)" "float" 0.5)
(supVol "Enter supply voltage(in volts)" "float" 5.0)
(tech "Enter technology [cmos3,cmos4]" "file" "cmos3")
(cancel "Cancel generation" "Boolean" nil)

if(cancel then return())

designInfo.file=outfile("skill/designInfo.tech" "w")

fprint(designInfo.file ";;\n")
(fprint(designInfo.file ";;This file supplies length,width of I/O pads. \n")
(fprint(designInfo.file ";;It also supplies max perimeter, power consumption, supply voltage. \n")
(fprint(designInfo.file ";;\n")
(fprint(designInfo.file ";;input pad \n")
(fprint(designInfo.file "inputx=\%f \n\n" inputx)
(fprint(designInfo.file "inputy=\%f \n\n" inputy)
(fprint(designInfo.file ";;output pad \n")
(fprint(designInfo.file "outputx=\%f \n\n" outputx)
(fprint(designInfo.file "outputy=\%f \n\n" outputy)
(fprint(designInfo.file ";;gnd pad \n")
(fprint(designInfo.file "gndx=\%f \n\n" gndx)
(fprint(designInfo.file "gndy=\%f \n\n" gndy)
(fprint(designInfo.file ";;vdd pad \n")
(fprint(designInfo.file "vddx=\%f \n\n" vddx)
(fprint(designInfo.file "vvdyy=\%f \n\n" vvdyy)
(fprint(designInfo.file ";;BIPSP cell\n")
(fprint(designInfo.file "BIPSPx=\%f \n\n" BIPSPx)
(fprint(designInfo.file "BIPSPy=\%f \n\n" BIPSPy)
(fprint(designInfo.file ";;max chip perimeter (in microns)\n")
(fprint(designInfo.file "chipx=\%f \n\n" chipx)
(fprint(designInfo.file "chipy=\%f \n\n" chipy)
(fprint(designInfo.file ";;Maximum power consumption (in watts)\n")
(fprint(designInfo.file "chipPow=\%f \n\n" chipPow)
(fprint(designInfo.file ";;Supply voltage\n")
(fprint(designInfo.file "supVol=\%f \n\n" supVol)
(fprint(designInfo.file ";;technology\n")
(fprint(designInfo.file "tech=\"%s\" \n\n" tech)
close(designInfo.file)
)
)

procedure( loadmodcoeff(cutfreq1 cutfreq2 samrate inwordlen outwordlen FIRtype type chipname nummod numcoeff)
    prog()
        coeffFile=outfile("skill/coeff.file")
procedure( compile_c() )

; This procedure compiles all c programs needed in the WINDSILC.
;
;
;
;

system("cc new_c_dir/masromrow.c -lm -o ex_c_dir/masromrow.out ")
system("cc new_c_dir/masrom.c -lm -o ex_c_dir/masrom.out")
system("cc new_c_dir/newmodplace.c -lm -o ex_c_dir/newmodplace.out ")
system("cc new_c_dir/edif.c -lm -o ex_c_dir/edif.out")
system("cc new_c_dir/placeio.c -lm -o ex_c_dir/placeio.out")
system("cc new_c_dir/dectomod.c -lm -o ex_c_dir/dectomod.out")
system("cc new_c_dir/sch_skill.c -lm -o ex_c_dir/main_sch_array.out")

compile_c()

procedure( create_bit_rom_symbol(block_symbol bit) )

; This procedure creates a symbol for a single rom bit.
;
;
; ARGUMENTS:
;    block_symbol: blockname of ROM.
;    bit: The bit number.
;
;
;
graphEdit("%s symbol current" block_symbol)
procedure( create_symbol_rom(block_symbol )

;*******************************************************************************
; This procedure creates a symbol for each mod of the FIR_Filter.
; This symbol is used in the cmos3lib library.
; Edit2sda checks the symbol in each library component.
;
; ARGUMENTS:
;    block_symbol: Name of mod cell.
;
;*******************************************************************************

    graphEdit(" cmos3lib/cell%# symbol current" block_symbol)
    setLayer("bkgnd")
    openSDA("GraphicsEditor addalldi")
    openSDA("GraphicsEditor rm")
    setLayer("instance")
    rectangle(0.0 0.875:1.625)
    setLayer("device")
    rectangle(0.0 0.875:1.625)
    setMaster("sympin symbol current")
    pinInstance(0.150 "R7" "r0" "input")
    pinInstance(0.1375 "R6" "r0" "input")
    pinInstance(0.125 "R5" "r0" "input")
    pinInstance(0.125 "R4" "r0" "input")
    pinInstance(0.100 "R3" "r0" "input")
    pinInstance(0.875 "R2" "r0" "input")
    pinInstance(0.750 "R1" "r0" "input")
    pinInstance(0.625 "R0" "r0" "input")
    pinInstance(0.500 "C3" "r0" "input")
    pinInstance(0.375 "C2" "r0" "input")
    pinInstance(0.250 "C1" "r0" "input")
    pinInstance(0.125 "C0" "r0" "input")
    pinInstance(0.438:1.625 "PREBAR" "r0" "input")
    pinInstance(0.875:0.875 "EVAL" "r0" "output")
    setLayer("bkgnd")
    label(1.000:1.500 "@instanceName" "leftBottom" "stick" 0.1 "r0" "")
    sprintf(romnum "ROM%d" bit)
    label(0.25:0.750 romnum "leftBottom" "stick" 0.1 "r0" ")
    save()
)
pinInstance(0.000:1.250 "Ain1" "r0" "input")
pinInstance(0.000:1.125 "Ain2" "r0" "input")
pinInstance(0.000:1.000 "Ain3" "r0" "input")
pinInstance(0.000:0.875 "Ain4" "r0" "input")
pinInstance(0.000:0.625 "Pin0" "r0" "input")
pinInstance(0.000:0.500 "Pin1" "r0" "input")
pinInstance(0.000:0.375 "Pin2" "r0" "input")
pinInstance(0.000:0.250 "Pin3" "r0" "input")
pinInstance(0.000:0.125 "Pin4" "r0" "input")
pinInstance(1.500:1.375 "Aout0" "r0" "output")
pinInstance(1.500:1.250 "Aout1" "r0" "output")
pinInstance(1.500:1.125 "Aout2" "r0" "output")
pinInstance(1.500:1.000 "Aout3" "r0" "output")
pinInstance(1.500:0.875 "Aout4" "r0" "output")
pinInstance(1.500:0.625 "Bout0" "r0" "output")
pinInstance(1.500:0.500 "Bout1" "r0" "output")
pinInstance(1.500:0.375 "Bout2" "r0" "output")
pinInstance(1.500:0.250 "Bout3" "r0" "output")
pinInstance(1.500:0.125 "Bout4" "r0" "output")
save()

procedure( create_5rom_symbol(block_symbol )

;**************************************************************************
;This procedure creates a symbol for a five bit rom
;
;ARGUMENT:
;    block_symbol: Name of BIPSP rom cell.
;
;**************************************************************************

graphEdit("%s symbol current" block_symbol)
setLayer("bknd")
openSDA("GraphicsEditor addallidi")
openSDA("GraphicsEditor rm")
setLayer("instance")
rectangle(0:0 1.375:1.500)
setLayer("device")
rectangle(0:0 1.375:1.500)
setMastert("sympin symbolr current")
    pinInstance(0.000:1.250 "PREBAR" "r0" "input")
    pinInstance(0.000:1.125 "EVAL4" "r0" "output")
    pinInstance(0.000:1.000 "EVAL3" "r0" "output")
    pinInstance(0.000:0.875 "EVAL2" "r0" "output")
    pinInstance(0.000:0.625 "EVAL1" "r0" "output")
    pinInstance(0.000:0.500 "EVAL0" "r0" "output")
    pininstance(0.000:0.375 "C3" "r0" "input")
    pininstance(0.000:0.250 "C2" "r0" "input")
    pinInstance(0.000:0.125 "C1" "r0" "input")
    pinInstance(1.375:1.250 "C0" "r0" "input")
pinInstance(1.375:1.125 "R7"   "r0" "input")
pinInstance(1.375:1.000 "R6"   "r0" "input")
pinInstance(1.375:0.875  "R5"  "r0" "input")
pinInstance(1.375:0.625  "R4"  "r0" "input")
pinInstance(1.375:0.500  "R3"  "r0" "input")
pinInstance(1.375:0.375  "R2"  "r0" "input")
pinInstance(1.375:0.250  "R1"  "r0" "input")
pinInstance(1.375:0.125  "R0"  "r0" "input")
setLayer("bkgrd")
save()

procedure( create_mod_symbol( type chip mod )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This procedure creates a symbol for a single mod circuit.
;
; ARGUMENTS:
;    type: Three types encoder,FIR_Filer and scaling.
;    chip: chip name
;    mod: Name of mod circuit.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

prog((modblock BIPSP count offset)
;
; Creating a symbol for mod circuit.
;

sprintf(chipblock " %s/%s/%s symbol current" type chip mod)
graphEdit(chipblock)
setLayer("bkgrd")
onopenSDA(“GraphicsEditor addalldi”)
onopenSDA(“GraphicsEditor rm”)
setLayer("instance")
rectangle(0:0 2.125:2.625)
setLayer("device")
rectangle(0.125:0 2.250)
line(0.125 0.125 0.625)
line(0.125 0.250 0.750)
line(0.125 0.875 0.875)
line(0.125 1.000 1.000)
line(0.125 1.125 1.125)
line(0.125 1.375 1.375)
line(0.125 1.500 1.500)
line(0.125 1.625 1.625)
line(0.125 1.750 1.750)
line(0.125 1.875 1.875)
openSDA("GraphicsEditor setenv #Bits=1 ")
openSDA("DisplayControl setlayer wire pin ")
openSDA("GraphicsEditor setenv io=input")

; Create input pins.
;

openSDA("GraphicsEditor setenv labelText=Bin4 ")
rectangle(-0.025:0.600 0.025:0.650)
label(0.025:0.650 "Bin4" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bin3 ")
rectangle(-0.025:0.725 0.025:0.775)
label(0.025:0.775 "Bin3" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bin2 ")
rectangle(-0.025:0.850 0.025:0.900)
label(0.025:0.900 "Bin2" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bin1 ")
rectangle(-0.025:0.975 0.025:1.025)
label(0.025:1.025 "Bin1" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bin0 ")
rectangle(-0.025:1.100 0.025:1.150)
label(0.025:1.150 "Bin0" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Ain4 ")
rectangle(-0.025:1.350 0.025:1.400)
label(0.025:1.400 "Ain4" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Ain3 ")
rectangle(-0.025:1.475 0.025:1.525)
label(0.025:1.525 "Ain3" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Ain2 ")
rectangle(-0.025:1.600 0.025:1.650)
label(0.025:1.650 "Ain2" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Ain1 ")
rectangle(-0.025:1.725 0.025:1.775)
label(0.025:1.775 "Ain1" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Ain0 ")
rectangle(-0.025:1.850 0.025:1.900)
label(0.025:1.900 "Ain0" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=GPHI1 ")
rectangle(0.850:2.600 0.900:2.650)
label(0.900:2.650 "GPHI1" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=GPHI2 ")
rectangle(1.225:2.600 1.275:2.650)
label(1.225:2.650 "GPHI2" "" "stick" 0.05 )

; Create output pins
;
openSDA("GraphicsEditor setenv io=output")
openSDA("GraphicsEditor setenv labelText=Bout4 ")
rectangle(2.100:0.600 2.150:0.650)
label(2.150:0.675 "Bout4" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout3 ")
rectangle(2.100:0.725 2.150:0.775)
label(2.150:0.800 "Bout3" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout2 ")
rectangle(2.100:0.850 2.150:0.900)
label(2.150:0.925 "Bout2" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout1 ")
rectangle(2.100:0.975 2.150:1.025)
label(2.150:1.050 "Bout1" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout0 ")
rectangle(2.100:1.100 2.150:1.150)
label(2.150:1.175 "Bout0" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Aout4 ")
rectangle(2.100:1.350 2.150:1.400)
label(2.150:1.400 "Aout4" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Aout3 ")
rectangle(2.100:1.475 2.150:1.525)
label(2.150:1.525 "Aout3" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Aout2 ")
rectangle(2.100:1.600 2.150:1.650)
label(2.150:1.650 "Aout2" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Aout1 ")
rectangle(2.100:1.725 2.150:1.775)
label(2.150:1.775 "Aout1" "" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Aout0 ")
rectangle(2.100:1.850 2.150:1.900)
label(2.150:1.900 "Aout0" "" "stick" 0.05 )

; Label mod symbol.
;
label(0.625:2.125 mod "leftBottom" "stick" 0.1 "r0" "")
save()
}
)
procedure( create_BISP_symbol() )

;*******************************************************************************************************************
; This procedure creates a symbol for one BISP cell.
;
;*******************************************************************************************************************

prog()

;

Creating a symbol for mod circuit.
;

graphEdit(" BISP.s symbol current")
setLayer("bkgrd")
  openSDA("GraphicsEditor addalldi")
openSDA("GraphicsEditor rm")
setLayer("instance")
rectangle( 0:0 2.125:2.625)
setLayer("device")
rectangle(0.125:0 2.250)
line(0:0.625 0.125:0.625)
line(0:0.750 0.125:0.750)
line(0:0.875 0.125:0.875)
line(0:1.000 0.125:1.000)
line(0:1.125 0.125:1.125)
line(0:1.375 0.125:1.375)
line(0:1.500 0.125:1.500)
line(0:1.625 0.125:1.625)
line(0:1.750 0.125:1.750)
line(0:1.875 0.125:1.875)
line(2.0.625 2.125:0.625)
line(2.0.750 2.125:0.750)
line(2.0.875 2.125:0.875)
line(2:1.000 2.125:1.000)
line(2:1.125 2.125:1.125)
line(2:1.375 2.125:1.375)
line(2:1.500 2.125:1.500)
line(2:1.625 2.125:1.625)
line(2:1.750 2.125:1.750)
line(2:1.875 2.125:1.875)
line(0.875:2.500 0.875:2.625)
line(1.250:2.500 1.250:2.625)

openSDA("GraphicsEditor setenv #Bits=1 ")
openSDA("DisplayControl setlayer wire pin ")
openSDA("GraphicsEditor setenv io=input")

;

Create input pins.
;
Create output pins

openSDA("GraphicsEditor setenv io=output")

openSDA("GraphicsEditor setenv labelText=Bout4 ")
rectangle(2.100:0.600 2.150:0.650)
label(2.150:0.650 "Bout4" " "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout3 ")
rectangle(2.100:0.725 2.150:0.775)
label(2.150:0.775 "Bout3" " "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout2 ")
rectangle(2.100:0.850 2.150:0.900)
label(2.150:0.900 "Bout2" " "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=Bout1 ")
rectangle(2.100:0.975 2.150:1.025)
rectangle(0.350:0.850 0.400:0.900)
  label(0.400:0.900 "C3" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=PREBAR ")
rectangle(0.350:1.725 0.400:1.775)
  label(0.400:1.775 "PREBAR" "stick" 0.05 )

openSDA("GraphicsEditor setenv io=input")

openSDA("GraphicsEditor setenv labelText=EVAL0 ")
rectangle(0.350:0.975 0.400:1.025)
  label(0.400:1.025 "EVAL0" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=EVAL1 ")
rectangle(0.350:1.100 0.400:1.150)
  label(0.400:1.150 "EVAL1" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=EVAL2 ")
rectangle(0.350:1.350 0.400:1.400)
  label(0.400:1.400 "EVAL2" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=EVAL3 ")
rectangle(0.350:1.475 0.400:1.525)
  label(0.400:1.525 "EVAL3" "stick" 0.05 )
openSDA("GraphicsEditor setenv labelText=EVAL4 ")
rectangle(0.350:1.600 0.400:1.650)
  label(0.400:1.650 "EVAL4" "stick" 0.05 )

save()

)
)
detail("FIR_Filter" "test_design")

******************************************************************************
+ DSP DESK
******************************************************************************
(0, 0, 0, "set infix");
(0, 0, 0, "newdesk");
(0, 99, 0, "close");
(1, 1, 0, "resize");
(0, 0, 2, "");
(1151, 22, 2, "");
(1150, 21, 0, "DisplayControl read layers/display/current ; DisplayControl setlayers 0 0");
(0, 23, 0, "open");
(1151, 261, 2, "");
(0, 262, 0, "open");
(1151, 614, 2, "");
(0, 615, 0, "open");
(1151, 836, 2, "");
(1150, 835, 0, "GraphicsEditor edit DSPlib/palllete schematic");
(0, 0, 0, "set noinfix");
procedure(FIRblock())

prog((Type FIRtype char FIRtype cutfreqchar1 cutfreqchar2 samratechar inwordlenchar outwordlenchar modcountchar chipnumchar)

;**********************************************************************************************
;This procedure creates a block diagram for a FIR Filter. 
;The input for this procedure is stored in a param.file .
;The parameter file contains information such as modulo 
;numbers, type of FIR Filter(low, high or band pass), 
;the number of coefficient, word length of input and 
;output, the number of chips needed to implement the 
;FIR Filter.
;
;**************************************************************************************

declare(coef[20])
declare(modnum[10])

load("skill/coef.file")

;Create a number of modulo rows of FIR symbol.
;
graphEdit("FIR_Filter/block_diagram_schematic")
addAllDI()
deleteShapes()

setMaster("FIR_Filter/block_diagram/modulo_tap symbol current")
for(i 1 nummod
    instance(1.875:5-i)
)

;Create the symbol for the type of filter.
;
setMaster("FIR_Filter/block_diagram/graph/%s symbol" FIRtype)
instance(7.125:0)

;Create the symbol for the coefficient table
;
setMaster("FIR_Filter/block_diagram/Table symbol")
instance(8.125:3.2)

;label the modulo number next to the FIR filter symbol.
;
label(0.5:5 "FIR Filter" "leftBottom" "gothic" 1)
label(-0.8:4.5 "modulo" "leftBottom" "stick" 0.2)

for(j 1 nummod
    sprintf(modnumchar "%d" modnum[j])
    label(-0.5:5-j modnumchar "leftBottom" "stick" 0.2)
)
save()

; label information about FIR filter

sprintf(Type " FIR_Filter/block_diagram/graph/%s schematic" FIRtype)
graphEdit(Type)

addAllDOI()
deleteShapes()

sprintf(FIRtypechar "%s filter" FIRtype)
label( 3:10 FIRtypechar "leftBottom" "stick" 1)

if( (strcmp(FIRtype "bandpass") == 0)
    then
    sprintf(cutfreqchar1 "%d" cutfreq1)
    label( 3:0 "1st Cutoff Frequency" "leftBottom" "stick" 0.5)
    label( 12:0 cutfreqchar1 "leftBottom" "stick" 0.5)

    sprintf(cutfreqchar2 "%d" cutfreq2)
    label( 3:1 "2nd Cutoff Frequency" "leftBottom" "stick" 0.5)
    label( 12:1 cutfreqchar2 "leftBottom" "stick" 0.5)
else
    sprintf(cutfreqchar1 "%d" cutfreq1)
    label( 3:0 "Cutoff Frequency" "leftBottom" "stick" 0.5)
    label( 12:0 cutfreqchar1 "leftBottom" "stick" 0.5)
)

sprintf(samratechar "%d" samrate)
label( 3:6 "Sampling Rate" "leftBottom" "stick" 0.5)
label( 12:6 samratechar "leftBottom" "stick" 0.5)

sprintf(inwordlenchar "%d" inwordlen)
label( 3:5 "Word length of input" "leftBottom" "stick" 0.5)
label( 12:5 inwordlenchar "leftBottom" "stick" 0.5)

sprintf(outwordlenchar "%d" outwordlen)
label( 3:4 "Word length of output" "leftBottom" "stick" 0.5)
label( 12:4 outwordlenchar "leftBottom" "stick" 0.5)

label( 3:3 "Modulo used" "leftBottom" "stick" 0.5)

for(i 1 nummod
    sprintf(modcountchar "%d" modnum[i])
label(10.5+i*1.5:3 modcountchar "leftBottom" "stick" 0.5)
)

sprintf(coefnumchar "%d" coefnum)
label(3:2 "Number of taps used" "leftBottom" "stick" 0.5)
label(12:2 coefnumchar "leftBottom" "stick" 0.5)

sprintf(chipnumchar "%d" chipnum)
label(3:1 "Number of chips" "leftBottom" "stick" 0.5)
label(12:1 chipnumchar "leftBottom" "stick" 0.5)

save()

; Create coefficient table.
;

graphEdit("FIR_Filter/block_diagram/Table schematic")
addAIID1()
deleteShapes()

for(k 1 coefnum

    sprintf(coefchar "%d" coeff[k])
    sprintf(Tap "%d" k)

    label(0:10 "Tap quantized coefficients" "leftBottom" "stick" 0.5)
    label(0:8-k Tap "leftBottom" "stick" 0.5)

    label(8:8-k coefchar "leftBottom" "stick" 0.5)

)

save()

graphView("FIR_Filter/block_diagram schematic")
setWEnvVar("displayAxes?" "no")
setWEnvVar("displayGrid?" "no")

procedure(FIRblockgen())

prog((Type FIRtypechar cutfreqchar1 cutfreqchar2 samratechar inwordlenchar outwordlenchar modcountchar chipnumchar)

;********************************************************************
;This procedure creates a block diagram for a FIR Filter.
;The input for this procedure is stored in a param.file.
;The parameter file contains information such as modulo
;numbers, type of FIR Filter(low, high or band pass),
;the number of coefficient,word length of input and
;output, the number of chips needed to implement the
;FIR Filter.
;
;********************************************************************
declare(coef[300])
declare(modnum[10])

sprintf(coeff " skill/coff file")
load(coeff)

; ;Create a number of modulo rows of FIR symbol.
;
graphEdit(" FIR_Filter/block_diagram schematic")
addAllID()
deleteShapes()

setMaster(" FIR_Filter/block_diagram/modulo tap symbol current")
for(i 1 nummod
    instance(1.875:5-i)
)

; ;Create the symbol for the type of filter.
;
setMaster(" FIR_Filter/block_diagram/graph/%s symbol" FIRtype)
instance(7.125:0)

; ;Create the symbol for the coefficient table
;
setMaster(" FIR_Filter/block_diagram/Table symbol")
instance(8.125:3.2)

; ;label the modulo number next to the FIR filter symbol.
;
label(0.5:5 "FIR Filter" "leftBottom" "gothic" 1)
label(-0.8:4.5 "modulo" "leftBottom" "stick" 0.2)

for(i 1 nummod
    sprintf modnumchar "%d" modnum[i])
    label(-0.5:5-j modnumchar "leftBottom" "stick" 0.2)
)
save()

; ;label information about FIR filter
;
sprintf(Type " FIR_Filter/block_diagram/graph/%s schematic" FlRtype)
graphEdit(Type)

addAllDI()
deleteShapes()

sprintf(FlRtypechar "%s filter" FlRtype)
label( 3:10 FlRtypechar "leftBottom" "stick" 1)

if( (strcmp(FlRtype "bandpass") == 0)
then
sprintf(cutfreqchar1 "%d" cutfreq1)
label( 3:0 1'st Cutoff Frequency" "leftBottom" "stick" 0.5)
label( 12:0 cutfreqchar1 "leftBottom" "stick" 0.5)
sprintf(cutfreqchar2 "%d" cutfreq2)
label( 3:1 2'nd Cutoff Frequency" "leftBottom" "stick" 0.5)
label( 12:-1 cutfreqchar2 "leftBottom" "stick" 0.5)
else
sprintf(cutfreqchar1 "%d" cutfreq1)
label( 3:0 "Cutoff Frequency" "leftBottom" "stick" 0.5)
label( 12:0 cutfreqchar1 "leftBottom" "stick" 0.5)
)

sprintf(samratechar "%d" samrate)
label( 3:6 "Sampling Rate" "leftBottom" "stick" 0.5)
label( 12:6 samratechar "leftBottom" "stick" 0.5)

sprintf(inwordlenchar "%d" inwordlen)
label( 3:5 "Word length of input" "leftBottom" "stick" 0.5)
label( 12:5 inwordlenchar "leftBottom" "stick" 0.5)

sprintf(outwordlenchar "%d" outwordlen)
label( 3:4 "Word length of output" "leftBottom" "stick" 0.5)
label( 12:4 outwordlenchar "leftBottom" "stick" 0.5)

label( 3:3 "Modulo used" "leftBottom" "stick" 0.5)
for(1 nummod
sprintf(modcountchar "%d" modnum[il])
label( 10.5+i*1.5:3 modcountchar "leftBottom" "stick" 0.5)
)

sprintf(coefnumchar "%d" numcoeff)
label( 3:2 "Number of taps used" "leftBottom" "stick" 0.5)
label( 12:2 coefnumchar "leftBottom" "stick" 0.5)

sprintf(chipnumchar "%d" chipnum)
label( 3:1 "chip number" "leftBottom" "stick" 0.5)
label( 12:1 chipnumchar "leftBottom" "stick" 0.5)
setWEnvVar("displayAxes?" "no")
setWEnvVar("displayGrid?" "no")
save()
graphEdit("FIR_Filter/block_diagram/Table schematic")
addAllDI()
deleteShapes()

for k 1 numCoeff
    sprintf(coef->(char) "%d" coeffnum(k))
    sprintf(Tap "%d" k)
    label( 0:10 "Tap quantized coefficients" "leftBottom" "stick" 0.5)
    label( 0:8-k Tap "leftBottom" "stick" 0.5)
    label( 8:8-k coef->(char) "leftBottom" "stick" 0.5)
)

setWEnvVar("displayAxes?" "no")
setWEnvVar("displayGrid?" "no")
save()

graphView("FIR_Filter/block_diagram schematic")
setWEnvVar("displayAxes?" "no")
setWEnvVar("displayGrid?" "no")
)

procedure( icheck() )
    prog(( rep outPort)

;untracev( c)

/* open output file */
printMessage("expecting output filename \n");

; outPort = outfile( GetString( "\nEnter output filename : " ) )
outPort = outfile( "justice")

/* open and load rep to be checked */
rep = nil
printMessage("expecting database representation name\n")
while( rep == nil
    rep = dbOpen( GetString("\nEnter rep id( block rep [rev] : ") )
; rep = dbOpen("/home/sda/mcarthur/newmod5a/library/dflop_test schematic")
)

/* if schematic, make sure rep has been extracted */

if rep->repName == "schematic" then
    printf("\nlastSchematicExtraction : %s\n" rep->lastSchematicExtraction)
    printf("\ninstancesLastChanged : %s\n" rep->instancesLastChanged)
if((compareTime(rep->lastSchematicExtraction,rep->instancesLastChanged) <=
0)
    then
        printf("\nError - connectivity data is not current.\n")
        printf("Please extract rep and try again.\n\n");
        return();
    )

/* write header information */

printf("\nRunning current check on %s \n" rep->fullPathName)
fprintf(outPort "\nRunning current check on %s \n" rep->fullPathName)
fprintf(outPort "\nBlock path is : \n\n%\n\n" dbGetPath())
if(rep->NLPRepresentationList != nil then
    printf("NLPRepresentationList : %s\n" rep->NLPRepresentationList))

printf("*********** ~~~~~~~~~~\n")

/*loop through each net in the rep invoking the check */

foreach( net rep->nets ichecknet(net))

/* flush and close output port */

close( outPort)

)

procedure( ichecknet( net)
    prog( (termprop childnet term inst outputTerminals totalSink oterm io maxCur command

    totalSink = 0
    outputTerminals = ()

    if((net->name != nil) then printf("net name is : %s \n" net->name)
        else printf("net name is : nil \n")
    printf("global net : %s \n" net->isGlobalIP)
    if((net->width != nil) then printf("net width : %d \n" net->width)
    if((net->type != nil) then printf("net type : %s \n" net->type)
    if((net->prior != nil) then
        if((net->prior != nil) then
            printf("net priority : %d \n" net->prior)
        else
            printf("net priority : %s \n" net->prior)
        )
    )
    if((net->nets != nil) then printf("net children of the net are listed below \n")
        foreach( childnet net->nets if((childnet->name != nil) then printf("child net is : %s \n" childnet->name))
    )
    if((net->instances != nil) then printf("net composed on instances listed below\n")

)
foreach( inst net->instances
    if((inst->masterName != nil) printf("instance name : %s 
"
inst->masterName))
    if((inst->name != nil) printf("name : %s " inst->name))
    if((inst->mosaic != nil) printf("parent mosaic : %s " inst->mosaic))
    if((inst->purpose != nil) then printf(" purpose : %s\n" inst->purpose))
)

if((net->terminals != nil) then printf("\nnet is connected to the following terminals \n")
    foreach( term net->terminals
        printf("terminal : %s io : %s width : %d instance name : %s 
"
        term->name
        term->fTerm->io
        term->fTerm->width
        term->fTerm->rep->blockName
        )
        printf("instance# : ")
        if(((termprop = get_instance(term->fTerm->rep)) != nil) then
            mystery_printf(" " termprop)
        else printf("no props")
    )
    printf("\n")
    if((term->fTerm->prop != nil) then
        foreach( termprop term->fTerm->prop
            printf(" formal prop %s : " termprop->name)
            mystery_printf(" value : " termprop->value)
            printf("\n")
        )
    )
)

if((net->pins != nil) then printf("\npins associated with the net are : \n\n")
    foreach( pin net->pins printf("pins terminal name : %s \n pin->term->name))

printf("++++++++++++++++++++++++++++++\n")

; visit all actual terminals connected to the net
foreach( term net->terminals
    inst = term->inst
    case( term->io
        ("output" outputTerminals = cons( list( inst->name
            term->name term->maxCur) outputTerminals))
        ("input" if((term->maxCur!=nil) then totalSink = totalSink +
        term->maxCur))
    )
    )

; visit all formal terminals connected to the net
; notice that a formal terminal whose io field is "input" is
; actually an output on the net and vice versa

foreach( pin net->pins

function maxCur = nth(x, n)
    if(maxCur == nil) then maxCur = 0
    if(maxCur < totalSink) then
        printf("Inadequate current drive.\n")
        printf("Net %s\n", net->name)
        printf("Terminal %s\n", term->name)
        printf("Max drive current = %f\n", maxCur)
        printf("Max sink current = %f\n", totalSink)
    end
end

procedure GetString(prompt)
    l = nil
    printMessage("Get String")
    makeForm(list(c prompt "string"))
    return(c)
end

procedure mystery_printf(prompt value)
    printf(prompt)
    if fixp(value) then printf("%d", value)
    else
        if bigp(value) then printf("%d", value)
        else
            if symstrp(value) then printf("%s", value)
            else
                if floatp(value) then printf("%f", value)
                else
                    if null(value) then printf("nil")
                    else
                        printMessage("unknown type")
                    end
                end
            end
        end
    end
print("format currently not supported \n")

procedure(get_instance(rep_ptrn)
    prog(
        (property)
        if((rep_ptrn->prop == nil) then return(nil)
        else
            foreach( property rep_ptrn->prop
                    if((property->name == "instance#") then
                        return(property->value)
                    )
            )
            return(nil)
    )
)

procedure(MDL(rownm))
    prog((modcount romcount incindex modindex BIPSPcount lay)
        load(" skill/MDL.par")
    )
    ; Create layout skill files.
    ;
    ; sprintf(lay "%s/%s/main_array.skill.n" purpose chipname )
    ; portlay=outfile(lay "w")
    ; incindex=1
    ; modellcount=0
    ; for(BIPSPcount 0 (length(rom)-1)
    ;     modindex=fix(BIPSPcount/(length(rom)/length(moduli)))
    ;     if((modellcount == (length(rom)/length(moduli))) then
    ;         modellcount=0
    ;     )
    ;     if (modindex == incindex) then
    ;         ++incindex
    ;         col = (length(rom)/length(moduli))/rownm
    ;     fprintf(portlay "create_array(""mod%d" %d %d) 
    ; purpose nth(modindex-1 moduli) romnum col)
    ;     fprintf(portlay "create_rom(""mod%d" %d " chipname purpose
    ; nth(modindex moduli) romcount)
        for(romcount 0 31
            fprintf(portlay "%d " nth(romcount nth(BIPSPcount rom)))
        )
fprintf(portlay "\n")
++modellcount

fprintf(portlay "create_array\"%s\" \"%s\" \"mod%d\" %d %d \n\n" chipname purpose
nth(modindex moduli) rownum col)
close(portlay)

; Create schematic skill files.
;
    sprintf(sch "%s/%s/schematic.skill" purpose chipname )
portsch=outfile(sch "w")
incindex=1
modellcount=0
for(BIPSPcount 0 (length(rom)-1)
    modindex=fix(BIPSPcount/(length(rom)/length(moduli)))
    if((modellcount == (length(rom)/length(moduli))) then
        modellcount=0
    )
    if( modindex == incindex ) then
        ++incindex
        fprintf(portsch "sch_mod_gen\"%s\" \"%s\" \"mod%d\" %d \n\n" purpose
chipname nth(modindex-1 moduli) (length(rom)/length(moduli)))
    )
fprintf(portsch "sch_rom_gen\"%s\" \"%s\" \"mod%d\" \"BIPSP%d\" " chipname purpose nth(modindex moduli) modellcount)
for(romcount 0 31
    fprintf(portsch "%d " nth(romcount nth(BIPSPcount rom)))
    )
    fprintf(portsch "\n")
++modellcount
    )
    fprintf(portsch "sch_mod_gen\"%s\" \"%s\" \"mod%d\" %d \n\n" purpose chipname
nth(modindex moduli) (length(rom)/length(moduli)))
    fprintf(portsch "sch_chip_gen\"%s\" \"%s\" %d \n\n" purpose chipname
    length(moduli))
close(portsch)
)

procedure( makedir(directory1 directory2)
;********************************************************************************
; This skill routine simple creates a directory.
; The user only needs to supply sub directories that exist under
; /home2/sda/jackel/struc_comp_dir.
;
;********************************************************************************
;
    prog((status )
    {
        status=sys("mkdir new_struc_comp_dir/%s/%s",directory1 directory2)
    )
procedure( netprop(type chip rownum)

; This procedure creates a net property file. This file adds
; properties to nets such as net width, priority.
;
;

proo((rowcount netpropfile)

sprintf( netpropfile " %s/%s/netprop" type chip)
  outport=outfile(netpropfile "w")
  fprintf(output "printfout_for_rep \" %s/%s/autoLayout/current\" \n" type chip)
  fprintf(output "NET \"vdd\" PRIOR 115 R_WIDTH 166.000000 MAX_LEN 0.000000 CAP 0
  TYPE SIGNAL conn LOGICAL_NET EON \n")
  fprintf(output "NET \"gnd\" PRIOR 115 R_WIDTH 186.000000 MAX_LEN 0.000000 CAP 0
  TYPE SIGNAL conn LOGICAL_NET EON \n")

  for(rowcount 0 rownum-1
    fprintf(output "NET \"gnd%d\" PRIOR 115 R_WIDTH 40.000000 MAX_LEN 0.000000 CAP 0
    TYPE SIGNAL conn LOGICAL_NET EON \n" rowcount)
    fprintf(output "NET \"vdd%d\" PRIOR 115 R_WIDTH 40.000000 MAX_LEN 0.000000 CAP 0
    TYPE SIGNAL conn LOGICAL_NET EON \n" rowcount)
  )

  close(output)
)

)

procedure( ModCoeffVal()

proo()
  declare(modnum[5])
  declare(coeftnum[300])
    makeForm("Please fill in this form with appropriate values"
      'mod1 "Enter values for mod1" "int"
      'coeft1 "Enter values for coeff1" "int"
    )

    modnum[1]=mod1
    coeftnum[1]=coeft1

cutfreq1=1000

cutfreq2=5000

samrate=2000

inwordlen=9

outwordlen=10

FIRtype="lowpass"

type="FIR_Fter"

chipname="test_design"

nummod=1

numcoeff=1
loadmodcoeff(cutfreq1 cutfreq2 samrate inwordlen outwordlen FIRtype type chipname
nummod numcoeff)
)
)

;This skill routine programs one newmod cell. The user supplies the contents
;of Rom and the blockname of the stored layout. Rom contains a eight by four word.
;Each word is five bits.

procedure mod_create_rom( block_name count n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12 n13 n14 n15 n16
n17 n18 n19 n20 n21 n22 n23 n24 n25 n26 n27 n28 n29 n30 n31)

prog((output template personality setValue status status1 thisRep directory)

;The system command lets us run "C" programs. We first compile the "C" program
;personnyskill.c and we execute the executable file called a.out.
; sprintf(directory "%s" block_name)
; mkdir("romcell" directory)

printf(output "new_struc_comp_dir/%s/layout" block_name )
printf(template "new_struc_comp_dir/%s/template" block_name)
printf(personality "new_struc_comp_dir/%s/personality%d" block_name count )

;setting the graphicsEditorUnits per userUnit of layout rep to 2.

graphEdit(output)
thisRep= getWindowRep()
property(thisRep,"graphicsEditorUnits per userUnit",2 )
save()

graphEdit(template)

; change the masters property of template planes.

sprintf(setvalue1 " new_struc_comp_dir/leftside layout;")
sprintf(setvalue2 " new_struc_comp_dir/topside layout;")
sprintf(setvalue3 " new_struc_comp_dir/rightside layout;")
sprintf(setvalue4 " new_struc_comp_dir/bottomside layout;")
sprintf(setvalue5 " new_struc_comp_dir/int_space2 layout;")

; A: new_struc_comp_dir/H0_pat layout; B: new_struc_comp_dir/H1_pat
; layout; C: new_struc_comp_dir/V0_pat layout; D: new_struc_comp_dir/V1_pat layout; E:
; new_struc_comp_dir/H_space1 layout; F: new_struc_comp_dir/H_space2 layout; G:
; new_struc_comp_dir/V_space1 layout; H: new_struc_comp_dir/V_space2 layout; I:
; new_struc_comp_dir/int_space1 layout; J: new_struc_comp_dir/int_space2 layout;

property("leftside" "masters" setvalue1)
property("topside" "masters" setvalue2)
property("rightside" "masters" setvalue3)
property("bottomside" "masters" setvalue4)
property("inamespace" "masters" setvalue5)
property("inamespace1" "masters" setvalue5)
property("inamespace2" "masters" setvalue5)
property("inamespace3" "masters" setvalue5)
property("bit0" "masters" setvalue6)
property("bit1" "masters" setvalue6)
property("bit2" "masters" setvalue6)
property("bit3" "masters" setvalue6)
property("bit4" "masters" setvalue6)

save()

;The structure compiler is executed.

compileArray(output template personality)
save()
}

procedure printpersal(n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12 n13 n14 n15 n16 n17 n18 n19 n20 n21 n22 n23 n24 n25 n26 n27 n28 n29 n30 n31)
prog((blockname rom_add)

declare(n[32])
declare(rem[32])

n[0]=n0
n[1]=n1
n[2]=n2
n[3]=n3
n[4]=n4
n[5]=n5
n[6]=n6
n[7]=n7
n[8]=n8
n[9]=n9
n[10]=n10
n[12]=n12
n[13]=n13
n[14]=n14
n[15]=n15
n[16]=n16
n[17]=n17
n[18]=n18
n[19]=n19
n[20]=n20
n[21]=n21
n[22]=n22
n[23]=n23
n[24]=n24
n[25]=n25
n[26]=n26
n[27]=n27
n[28]=n28
n[29]=n29
n[30]=n30
n[31]=n31

out_port:=outfile("      new_struc_comp_dir/newromb/personality1" "w")

for(bit 0 4)
    fprintf(out_port ",.bias row \n")
    fprintf(out_port ",.plane bit%d \n", bit)
    fprintf(out_port "\n")

if(bit == 0)
    for(rom_add 0 31
        rem[rom_add]=mod(n[rom_add] 2)
    )

if(bit > 0)
    for(rom_add 0 31
        n[rom_add]=n[rom_add] / 2
        rem[rom_add]=mod(n[rom_add] 2)
    )

    fprintf(out_port "G")
    if(rem[15] == 0 then
        fprintf(out_port "C")
    else
        fprintf(out_port "D")
    )
    fprintf(out_port "H")
    if(rem[4] == 0 then
        fprintf(out_port "C")
    else
        fprintf(out_port "D")
    )
    fprintf(out_port "G \n")

if(rem[30] == 0 then
    fprintf(out_port "A")
else
    fprintf(out_port "B")
}

fprintf(out_port "E")
if(rem[31] == 0 then
    fprintf(out_port "A")
else
    fprintf(out_port "B")
}
if(rem[21] == 0 then
    fprintf(out_port "A")
else
    fprintf(out_port "B")
)
fprintf(out_port "E")
if(rem[20] == 0 then
    fprintf(out_port "A
")
else
    fprintf(out_port "B
")
)

fprintf(out_port "G")
if(rem[14] == 0 then
    fprintf(out_port "C")
else
    fprintf(out_port "D")
)
fprintf(out_port "H")
if(rem[5] == 0 then
    fprintf(out_port "C")
else
    fprintf(out_port "D")
)
fprintf(out_port "G
")
fprintf(out_port "I
")

!fprintf(out_port "G")
if(rem[7] == 0 then
    fprintf(out_port "C")
else
    fprintf(out_port "D")
)
fprintf(out_port "H")
if(rem[12] == 0 then
    fprintf(out_port "C")
else
    fprintf(out_port "D")
)
fprintf(out_port "G
")

if(rem[22] == 0 then
    fprintf(out_port "A")
else
    fprintf(out_port "B")
)
fprintf(out_port "F")
if(rem[23] == 0 then
    fprintf(out_port "A")
else
    fprintf(out_port "B")
)
fprintf(out_port "F")
if(rem[29] == 0 then
   fprintf(out_port "A")
else
   fprintf(out_port "B")
)
fprintf(out_port "E")
if(rem[28] == 0 then
   fprintf(out_port "A \n")
else
   fprintf(out_port "B \n")
)
fprintf(out_port "G")
if(rem[6] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "H")
if(rem[13] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "G \n")
fprintf(out_port "I \n")
fprintf(out_port "G")
if(rem[3] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "H")
if(rem[0] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "G \n")
if(rem[18] == 0 then
   fprintf(out_port "A")
else
   fprintf(out_port "B")
)
fprintf(out_port "E")
if(rem[19] == 0 then
   fprintf(out_port "A")
else
   fprintf(out_port "B")
)
fprintf(out_port "F")
if (rem[17] == 0 then
   fprintf(out_port "A")
else
   fprintf(out_port "B")
)
fprintf(out_port "E")
if (rem[16] == 0 then
   fprintf(out_port "A \n")
else
   fprintf(out_port "B \n")
)

fprintf(out_port "G")
if (rem[2] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "H")
if (rem[1] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "G \n")
fprintf(out_port "I \n")

fprintf(out_port "G")
if (rem[11] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "H")
if (rem[8] == 0 then
   fprintf(out_port "C")
else
   fprintf(out_port "D")
)
fprintf(out_port "G \n")

if (rem[26] == 0 then
   fprintf(out_port "A")
else
   fprintf(out_port "B")
)
fprintf(out_port "E")
if (rem[27] == 0 then
   fprintf(out_port "A")
else
   fprintf(out_port "B")
)
fprintf(out_port "F")
if(rem[25] == 0 then
    fprintf(out_port "A")
else
    fprintf(out_port "B")
)
fprintf(out_port "E")
if(rem[24] == 0 then
    fprintf(out_port "A \n")
else
    fprintf(out_port "B \n")
)
fprintf(out_port "G")
if(rem[10] == 0 then
    fprintf(out_port "C")
else
    fprintf(out_port "D")
)
fprintf(out_port "H")
if(rem[9] == 0 then
    fprintf(out_port "C")
else
    fprintf(out_port "D")
)
fprintf(out_port "G \n")
fprintf(out_port "\n")
}
close(out_port)
)

procedure( maincomp(type chipname)
    prog((PDir sienv replist replistblock sienvblock compenv compenvblock )
    );*****************************************************************************
;open si.env file AND ENVIRONMENTAL INFORMATION SUCH AS SIMMASTER,
; SIMULATOR, blockPath.
;*****************************************************************************
sprintf(PDir " %s/\%s/comprun" type chipname)
system("cd ; mkdir %s" PDir)
sprintf(sienv "%s/\%s/env" PDir)
sienvblock = outfile(sienv "w")
sprintf(sienvblock "SIMMASTER = \" %s/\%s/schema\current\%n\n" type chipname)
sprintf(sienvblock "SIMULATOR = \"PCompare\n")
sprintf(sienvblock "blockPath = \" /usr/local/cds/etc/sdalib/schema
/usr/local/cds/etc/sdalib/schema/sheets/38ths\"
")
close(sienvblock)
);*****************************************************************************
; open .replist file
;**********************************************************************
sprintf(replist " %s/%s/comprun/.replist" type chipname)
replistblock = outfile(replist)
fprintf(replistblock "[representationPath , ,string,\"lvs schematic gate.sch cmos.sch \"] \n ; ")
close(replistblock)

;**********************************************************************
;open comp.env file and write comp.env which contain
;PDirectory control parameters.
;**********************************************************************
sprintf(compenv "%s/comp.env" PDir)
compenvblock = outfile(compenv "w")
fprintf(compenvblock "compareLayout = \" %s/%s/extracted/current\"\n" type chipname)
fprintf(compenvblock "compareSchematic = \" %s/%s/schematic/current\"\n" type chipname)
fprintf(compenvblock "compareRules = \" rules/rules.lvs\"\n")
fprintf(compenvblock "createXref = nil\n")
fprintf(compenvblock "useTerminalCorrespondence = t\n")
fprintf(compenvblock "applyDeviceFixing = nil\n")
fprintf(compenvblock "compareNetlistLayout = t\n")
fprintf(compenvblock "compareNetlistSchematic = t\n")
close(compenvblock)
)
)

procedure(extractcomp( type chipname )
    prog((PDir replist replistblock)

;**********************************************************************
;open .replist file and write representation switch list.
;**********************************************************************
sprintf(PDir " %s/%s/extracted/comprun/.replist" type chipname)
system("cd ; mkdir %s" PDir)
sprintf(replist " %s/%s/extracted/comprun/.replist" type chipname)
replistblock = outfile(replist)
fprintf(replistblock "[representationPath , ,string,\"lvs extracted gate.sch cmos.sch \"] \n ; ")
close(replistblock)
)
)

procedure(doPDcompare(type chipname)
    prog(( startcomp procl readport string printstring)

;This procedure automatically compares a schematic and
;extracted rep under the same blockname.
;
compextract(type chipname 0)

;
;create main PDcompare run directory.
;
maincomp(type chipname)

;
;create run directory under extracted rep.
; extractcomp(type chipname)

; run PDCmpare in the unix environment.

; sprint(startcomp "si-batch %s/%s/comrun & " type chipname)
proc1=startProcess(startcomp)
readport=readFromChild(proc1->poport)
sprint(string "%s" readport)
while( strcmp("eof" string) != 0)
  if((strcmp("blanked" string) != 0) then
    sprint(printstring "%s" readport)
  )
readport=readFromChild(proc1->poport)
sprint(string "%s" readport)
)
cleanProcess(proc1->pid)
)

procedure(placeObjects(list xLoc yLoc dirFlag)
  foreach(elem list)
    inst = cadr(elem)
    placeInst(inst xLoc yLoc)
    x = abs(xCoord(upperRight(inst->bBox))
    - xCoord(lowerLeft(inst->bBox)))
    y = abs(yCoord(upperRight(inst->bBox))
    - yCoord(lowerLeft(inst->bBox)))
    delta = min(x y)
    placeInst(inst xLoc yLoc)
    case dirFlag
      ("x" xLoc = xLoc + delta)
      ("y" yLoc = yLoc + delta)
      (t)
    )
  )
)

procedure(placeInst(inst xPos yPos)
  transforms instance in R0 orientation
  case inst->orient
    ("0" orient = "r0")
    ("90" orient = "r90")
    ("180" orient = "r180")
    ("270" orient = "r270")
    ("sideways" orient = "sideways")
    ("sideways&90" orient = "sideways&90")
    ("upsideDown" orient = "upsideDown")
    ("sideways&270" orient = "sideways&270")
    (t)
  )
  transformObject(list(inst) list(-xCoord(inst->xy) -yCoord(inst->xy)) orient)
moveObject( list(inst) list( xPos->xCoord( inst->xy ) yPos->yCoord( inst->xy ) )
     property( inst "Instance/status" "placed" )
)

procedure( placeIo(type chipname)
    leftList = ncons( nil )
    bottomList = ncons( nil )
    topList = ncons( nil )
    rightList = ncons( nil )

    minSize = 1e20
    maxSize = 0.0
    nCells = 0
    exitFlag = 0

    sprintf( fileName " %s/%s/preplaceio" type chipname )

    if( fileName != "" then
        foreach( inst getWindowRep()->instances
        if( inst->master->type=="ioCircuit" then
            property( inst "Instance/status" "unplaced" )
        )
        graphicsEditor( "AutoLayout layoutenv fileName=%s" fileName )
        graphicsEditor( "AutoLayout placement justifyFrame" ) else
    )

    printMessage( "Resetting io-frame ... " )

    foreach( inst getWindowRep()->instances

        if( inst->master->type=="ioCircuit" then

            location = inst->PRioLocation
            offset = inst->PRioOffset
            side = inst->side

            x = abs(xCoord(upperRight(inst->bBox))
                - xCoord(lowerLeft(inst->bBox))
            y = abs(yCoord(upperRight(inst->bBox))
                - yCoord(lowerLeft(inst->bBox))

            minSize = min( x min( minSize y )
            maxSize = max( x max( maxSize y )
            nCells = nCells + 1

            if( inst->PRioLocation == nil || inst->PRioLocation == "none"
            then
                exitFlag = 1
                printf( " Property 'PRioLocation' is not specified for inst "%s\n", inst->name )
            else
                if( inst->PRioOffset == nil
                then
                    exitFlag = 1

printf( " Property 'PRioOffset' is not specified for inst '%s'", inst->name )
  else

  case( inst->PRioLocation
  ("left" property( inst "PRioSide" "left" "string" "no-help" nil "read-only"
  tconc( leftList list( inst->PRioOffset inst )))
  ("right" property( inst "PRioSide" "right" "string" "no-help" nil "read-only"
  tconc( rightList list( inst->PRioOffset inst )))
  ("top" property( inst "PRioSide" "top" "string" "no-help" nil "read-only"
  tconc( topList list( inst->PRioOffset inst )))
  ("bottom" property( inst "PRioSide" "bottom" "string" "no-help" nil "read-only"
  tconc( bottomList list( inst->PRioOffset inst )))
  (t
  printf(" Property 'PRioLocation' incorrectly specified on inst '%s'
  Value must be one out of (left right top bottom) - '%s'
  inst->name inst->PRioLocation )
  exitFlag = 1 )

  )
  )
  )

  if( exitFlag == 1)
  then
  printf(" Please specify the side (left top right bottom) and the location (>=0) for the io-
  circuits listed above in file '%s'
  Program aborted
  exitFlag = 1 )
  else

  leftList = sortcar( car(leftList) 'lessp )
  rightList = sortcar( car(rightList) 'lessp )
  topList = sortcar( car(topList) 'lessp )
  bottomList = sortcar( car(bottomList) 'lessp )

  setLayer( "bknd" "drawing" )
  graphicsEditor( "suball" )
  setLayer( "bknd" "drawing" )
  graphicsEditor( "suball" )
  setLayer( "prbdary" "boundary" )
  graphicsEditor( "addall" )
  l = length( selectedSet() )
  if( l == 1 )
  then
  box = car(selectedSet())--bBox
  setLayer( "bknd" "drawing" )
  graphicsEditor( "suball" )
  lowerleft = lowerLeft( box )
  upperright = upperRight( box )
  else
  lowerleft = list( 0.0 0.0 )
  upperright = list( nCells)*minSize (nCells)*minSize )

  placeObjects( leftList xCoord(lowerleft) yCoord(lowerleft)+maxSize "y" )
  placeObjects( rightList (xCoord(upperright)+100)-maxSize yCoord(lowerleft)+maxSize "y" )
  placeObjects( topList xCoord(lowerleft)+maxSize yCoord(upperright)-maxSize "x" )
placeObjects( bottomList xCoord(lowerleft)+maxSize yCoord(lowerleft) "x")

graphicsEditor( "AutoLayout placeenv ioJustificationStyle=free" )
graphicsEditor( "AutoLayout placement justifyFrame" )
graphicsEditor( "AutoLayout tools boundary" )

)

procedure( deleteIoProp() 
    foreach( inst getWindowRep()--instances 
        if( inst->master->type=="ioCircuit" then 
           deleteProperty( inst "FRioLocation" ) 
           deleteProperty( inst "FRioOffset" )
        )
    )

procedure( flatten_sch(type chipname)
    ; This procedure runs the flatnetlist in the background 
    ; and checks if it is done.
    ;
    prog((startflat proc1 readport string printstring replist replistblock sienv sienvblock Flatdir)
        setSkillPath("","","","martin")
        sprintf(Flatdir " %s/%s/flatten.run1" type chipname)
        system("cd; mkdir %s Flatdir")
        sprintf(sienv "%s/si.env" Flatdir)
        sienvblock=outfile(sienv "w")
        fprintf(sienvblock "SIMMASTER = \" %s/%s/netlist/current\" \n" type chipname)
        fprintf(sienvblock "SIMULATOR = \"flatten\" \n")
        fprintf(sienvblock "blockPath = \". %s/%s /usr/local/cds/etc/sdalib/schema
/usr/local/cds/etc/sdalib/schema/sheets/38ths\"\n" type chipname)
        close(sienvblock)

        sprintf(replist "%s/%s/flatten.run1/.replist" type chipname)
        replistblock=outfile(replist)
        fprintf(replistblock "[representationPath, ,string,"abstract schematic"] \n; ")
        close(replistblock)

        sprintf(startflat "si -batch %s/%s/flatten.run1 &" type chipname)
        proc1=startProcess(startflat )
        readport=readFromChild(proc1->poport)
        sprintf(string "%s" readport)
        while( (strcmp("eof" string) != 0)
            if((strcmp("blanked" string) != 0) then
                sprintf(printstring "%s" readport)
            }
            readport=readFromChild(proc1->poport)
            sprintf(string "%s" readport)
        )
        cleanProcess(proc1->pid)
    )
)
procedure(open(type chipname)
open place and route rep
prog(openrep)
    fprintf(openrep "%s/%s" type chipname)
setEnvVar("blockName" openrep )
setEnvVar("representationName" "autoLayout")
setEnvVar("revisionName" "current")
)
)

procedure(setup_report_dir(type chipname)
; sets up a report directory

    openSDA("GraphicsEditor AutoLayout layoutenv channelsBlockNamePrefix=
    " name_chipname"
    openSDA("GraphicsEditor AutoLayout placeenv outputCmdFile=clusterPlaceCmd ")
    openSDA("GraphicsEditor AutoLayout routeenv glrStatsOutputFile=glrStat ")
    openSDA("GraphicsEditor AutoLayout routeenv glrTempPinFile=Name= tempPins ")
    openSDA("GraphicsEditor AutoLayout routeenv glrSticksFile=Name= sticks ")
)

procedure(macroplace())
; place macro cells automatically
    openSDA("GraphicsEditor AutoLayout placement macroPlace")
)

procedure(ionotplace())
; ionotplace
    openSDA("GraphicsEditor AutoLayout placeenv placeo?=yes
    swapPlace.Ilo?=yes fastPlacement?=no
    ioJustificationStyle=free")
    openSDA("GraphicsEditor AutoLayout layoutenv mainTreeDir=ignore")
    openSDA("GraphicsEditor AutoLayout placeenv fastPlacement?=no
    ioToCellMinDist=50 ioToMinGap=0 alignLayer=prboundary"
    openSDA("GraphicsEditor AutoLayout placement justifyFrame")
    openSDA("GraphicsEditor AutoLayout placement clusterPlace")
)

procedure(iocorners())
; io corners
    openSDA("GraphicsEditor setenv masterRepresentationName=abstract")
    openSDA("GraphicsEditor AutoLayout placeenv ioCornerStartPosition=BL
    glueInstancePrefix=corInst numGluePieceInstances=4
    gluePieceNetAssociation=matchTermName")
    setEnvVar("masterBlockName" "cmos3lib/iocorner")
    openSDA("GraphicsEditor AutoLayout placement glueIn")
)
procedure( iojust()
; i/o justification

  openSDA("GraphicsEditor AutoLayout placeenv
  ioJustificationStyle=correspondsOnly")
  openSDA("GraphicsEditor AutoLayout
  placeenv alignLayer=supplyPins ioToCellMinDist=0")
  openSDA("GraphicsEditor AutoLayout placement justifyFrame")
)

procedure( changen()
; channel generation

  openSDA("GraphicsEditor AutoLayout placement closePlacement")
  openSDA("GraphicsEditor AutoLayout tools boundary")
  openSDA("GraphicsEditor AutoLayout channels treeGenerator")
  openSDA("GraphicsEditor AutoLayout channels channelGenerator")
  openSDA("GraphicsEditor AutoLayout tools boundary")
  ;openSDA("GraphicsEditor AutoLayout tools default(filePower")
  prigrlChannelInit()
)

procedure( splitpow()
; split power

  openSDA("GraphicsEditor AutoLayout tools splitPower")
  openSDA("GraphicsEditor AutoLayout layout env cancelCommand? = no")
)

procedure( getprop(type chipname)
; get net properties
  prog((netpropfile)
  sprint(netpropfile "GraphicsEditor AutoLayout layout env fileName = %.5s/%.5s/netprop"
  type chipname)
  openSDA(netpropfile)
  openSDA("GraphicsEditor AutoLayout tools eletToDBase")
)

procedure( global()
; global route

  openSDA("GraphicsEditor AutoLayout route env glrMinRoutingPriority=0")
  openSDA("GraphicsEditor AutoLayout route env glrCongestionCost=0
  glrStatsOutputFile = report STATS
  glrStats = detailed grpSpeed = bypass grpMirrorCells? = yes")
  openSDA("GraphicsEditor AutoLayout place env stubRoute? = no")
  openSDA("GraphicsEditor AutoLayout checkPlace connUpdate")
  openSDA("GraphicsEditor AutoLayout globalRoute glrinit")
)
prIglrGlobalInit()
openSDA("GraphicsEditor AutoLayout globalRoute routeAll")
openSDA("GraphicsEditor AutoLayout globalRoute glrClose")
openSDA("GraphicsEditor AutoLayout globalRoute optimize")
)

procedure( detail(type chipname)
; detail route
    prIglrClearNet()
    prDisplayHiErase()
    prIglrDetailInit()
    system("rm -r %s/%s/abstract" type chipname)
    openSDA("GraphicsEditor AutoLayout layoutenv savedRepName=saved")
    openSDA("GraphicsEditor AutoLayout route env detRRectangularIoFrame?=yes detRRoutingStyle=manhattan detRCenterContacts=offCentered detRContactAspectRatio=1 detRStaggeredExitContacts?=yes detRChanCompactionMode=automatic areaCertaintyFact=.3 contactCertaintyFact=.5 RCsCertaintyFact=.4")
    openSDA("GraphicsEditor AutoLayout detailRoute autoRouter")
    openSDA("GraphicsEditor AutoLayout tools bindMasters")
    openSDA("GraphicsEditor setwenv displayStartLevel=0 displayStopLevel=20")
)

procedure( save_p_r0)
; save place and route abstract rep.
    openSDA("GraphicsEditor AutoLayout layoutenv savedRepName=abstract")
    openSDA("GraphicsEditor AutoLayout anyLevel save")
)

procedure( abstract_to_layout(type chipname)
; generate abstract to layout.
    system("SOPATH=.: %d ; replaceReps %s/%s/saved1/current " type chipname)
)

procedure( close_p_r0)
; close place and route rep.
    openSDA("GraphicsEditor AutoLayout layoutManager close")
)

procedure( absLay(type chipname)
;
; This procedure runs the flatnetlist in the background
; and checks if it is done.
;
prog((starttabs readport printstring proc1 string)
        sprintf(starttabs "replaceReps %s/%s/saved/current &" type chipname)
        proc1=startProcess(starttabs )
        readport=readFromChild(proc1->poport)
        sprintf(string "%s" readport)
        while( (strcmp("eof" string) != 0) )
        if((strcmp("blanked" string) != 0) then
          sprintf(printstring "%s" readport)
        )
        readport=readFromChild(proc1->poport)
        sprintf(string "%s" readport)
    )
    ;printf("readport=%s \n" printstring)
    cleanProcess(proc1->pid)
    )
 */

procedure( setupPR(type chipname)
    prog((doDetailFile doPropFile doPlaceFile)
        doDetailFile=outfile(" skill/dodetail.skill "w")
        fprintf(doDetailFile "detail(%"%s" "%s") " type chipname)
        close(doDetailFile)
        doPropFile=outfile(" skill/dogetprop.skill "w")
        fprintf(doPropFile "getprop(%"%s" "%s") " type chipname)
        close(doPropFile)
        doPlaceFile=outfile(" skill/doplaceio.skill "w")
        fprintf(doPlaceFile "placeio(%"%s" "%s") " type chipname)
        close(doPlaceFile)
    )
)

procedure( place_route(type chipname)

;******************************************************************************
 ; This procedure runs place and route automatically.
 ;******************************************************************************

emptyWindow()
flatten_sch(type chipname)
setup_report_dir(type chipname)
open( type chipname )
setupPR(type chipname)
openSDA("read skill/pr.log")
absLay(type chipname)
)

procedure( rename_instance( modcount mod0 mod1 mod2 mod3 mod4)

; This routine removes the character "1" for every instance
; in the autolayout rep.

prog((inpadnum outpadnum IO_cir IO_name)

declare(mod[4])

; rename inpad instance .

for(inpadnum 0 (5*modcount)
   sprintf(IO_name "input%d" inpadnum)

sprintf(IO_cir "1%s" IO_name)

getWindowRep()-->instances-->TrueInstance-->numInst=IO_cir
   property(IO_cir "TrueInstance/name" IO_name)
 )

; rename outpad instances .

for(outpadnum 0 (10*modcount)
   sprintf(IO_name "output%d" outpadnum)

sprintf(IO_cir "1%s" IO_name)

getWindowRep()-->instances-->TrueInstance-->numInst=IO_cir
   property(IO_cir "TrueInstance/name" IO_name)
 )

; rename mod instances.

sprintf( IO_name "%s" mod0)

sprintf(IO_cir "1%s" IO_name)

getWindowRep()-->instances-->TrueInstance-->numInst=IO_cir
   property(IO_cir "TrueInstance/name" IO_name)

sprintf( IO_name "%s" mod1)

sprintf(IO_cir "1%s" IO_name)

getWindowRep()-->instances-->TrueInstance-->numInst=IO_cir
   property(IO_cir "TrueInstance/name" IO_name)

sprintf( IO_name "%s" mod2)

sprintf(IO_cir "1%s" IO_name)

getWindowRep()-->instances-->TrueInstance-->numInst=IO_cir
   property(IO_cir "TrueInstance/name" IO_name)

sprintf( IO_name "%s" mod3)

sprintf(IO_cir "1%s" IO_name)

getWindowRep()-->instances-->TrueInstance-->numInst=IO_cir
   property(IO_cir "TrueInstance/name" IO_name)
sprintf("1%<$s" mod4)
sprintf("1%<$s" IO_name)

getWindowRep()-->instances-->TrueInstace-->numInst=IO_cir
property(IO_cir "TrueInstace/name" IO_name)

; rename glo instances.

getWindowRep()-->instances-->TrueInstace-->numInst="lgpx"
property("lgpx" "TrueInstace/name" "gpx")

; rename power instances.

getWindowRep()-->instances-->TrueInstace-->numInst="lppx"
property("lppx" "TrueInstace/name" "ppx")

; rename phi1 instances.

getWindowRep()-->instances-->TrueInstace-->numInst="lphi1"
property("lphi1" "TrueInstace/name" "phi1")

; rename phi2 instances.

getWindowRep()-->instances-->TrueInstace-->numInst="lphi2"
property("lphi2" "TrueInstace/name" "phi2")

; rename pho1 instances.

getWindowRep()-->instances-->TrueInstace-->numInst="lpho1"
property("lpho1" "TrueInstace/name" "pho1")

; rename pho2 instances.

getWindowRep()-->instances-->TrueInstace-->numInst="lpho2"
property("lpho2" "TrueInstace/name" "pho2")

)

)

procedure( setup_tech() )

;*******************************************************************************
;This procedure creates preplacement file, a skill file that
;creates library cells used by place and route, and a edif
;file.
;
;*******************************************************************************

system(" ex_c_dir/main sch array.out -c 1 8 -m 1 31 > scaling/B1/main sch array skill")
system(" ex_c_dir/main sch array.out -c 1 17 -m 1 31 > scaling/B2/main sch array skill")
system(" ex_c_dir/main sch array.out -c 1 19 -m 1 31 > scaling/B3/main sch array skill")
procedure: sch_chip_gen( type chip nummod)

; This procedure creates a schematic for a chip.
; Argument:
; type: There is three types Encoder,FIR_Filter,
;     scaling.
; chip: Name of chip.
; nummod: Number of modulo used in this chip.
;
;
; prog((modblock BIPSPcount offset)
declare( modnum[10])
setSkillPath(""," "))
change_EXT_SRC
load(" skill/coefficient.file")

; Creating schematic for chip.
;

sprintf(modblock "%s/%s " type chip )
graphEdit(" %s schematic current" modblock)
setLayer("bkgnd")
openSDA("GraphicsEditor addalldi")
openSDA("GraphicsEditor rm")

; Place i/o pad instances.
;
setMaster("cmos3lib/gphi1pad symbol current")
instance(-3.600:6.125+7.500*nummod - 7.500)
setMaster("opin symbol current")
pinInstance(-3.000:6.125+7.500*nummod - 7.500 "GPHI1!" "" R0 output")
setMaster("cmos3lib/gphi2pad symbol current")
instance(-3.600:5.375+7.500*nummod-7.500)
setMaster("opin symbol current")
pinInstance(-3.000:5.375+7.500*nummod - 7.500 "GPHI2!" "" R0 output")

for(iocount 0 nummod-1
setMaster("cmos3lib/inpad symbol current")
instance(-3.600:4.625+7.500*iocount)
instance(-3.600:3.875+7.500*iocount)
instance(-3.600:3.125+7.500*iocount)
instance(-3.600:2.375+7.500*iocount)
instance(-3.600:1.625+7.500*iocount)
instance(-3.600:0.875+7.500*iocount)

);
instance(-3.600:0.125+7.500*iocount)
instance(-3.600:-0.625+7.500*iocount)
instance(-3.600:-1.375+7.500*iocount)
instance(-3.600:-2.125+7.500*iocount)
}

for(iocount 0 nummod-1
    setMaster("cmos3lib/outline symbol current")
    instance(5.000:4.625+7.500*iocount)
    instance(5.000:3.875+7.500*iocount)
    instance(5.000:3.125+7.500*iocount)
    instance(5.000:2.375+7.500*iocount)
    instance(5.000:1.625+7.500*iocount)
    instance(5.000:0.875+7.500*iocount)
    instance(5.000:0.125+7.500*iocount)
    instance(5.000:-0.625+7.500*iocount)
    instance(5.000:-1.375+7.500*iocount)
    instance(5.000:-2.125+7.500*iocount)
}

; Place mod symbol. If mod = 1 then place one mod symbol(mod32)
; If mod = 2 place two mod symbol (mod32 and mod31).
; If mod = 3 place three mod symbol (mod32, mod31 and mod29)
; If mod = 4 place four mod symbol (mod32, mod31, mod29 and mod27)

if nummod == 1 then
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[1])
    instance(-0.500:0.000)
}

if nummod == 2 then
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[1])
    instance(-0.500:0.000)
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[1])
    instance(-0.500:3.000)
}

if nummod == 3 then
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[1])
    instance(-0.500:0.000)
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[2])
    instance(-0.500:3.000)
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[3])
    instance(-0.500:6.000)
}

if nummod == 4 then
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[1])
    instance(-0.500:0.000)
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[2])
    instance(-0.500:3.000)
    setMaster(" %s/%s/mod%d symbol current" type chip modnum[3])
    instance(-0.500:6.000)
setMaster(" %s/%s/mod%d symbol current" type chip modnum[4])
instance(-0.500:9.000)
)
;
;
for(modcount0 nummod-1
  setMaster("ipin symbol current")

  printf(Ain0 "Ain%d0" modcount)
  printf(Ain1 "Ain%d1" modcount)
  printf(Ain2 "Ain%d2" modcount)
  printf(Ain3 "Ain%d3" modcount)
  printf(Ain4 "Ain%d4" modcount)
  printf(Bin0 "Bin%d0" modcount)
  printf(Bin1 "Bin%d1" modcount)
  printf(Bin2 "Bin%d2" modcount)
  printf(Bin3 "Bin%d3" modcount)
  printf(Bin4 "Bin%d4" modcount)

  pinInstance(0.3753*modcount + 2.625 "GPHI1" "R0" "input")
  pinInstance(0.7503*modcount + 2.625 "GPHI2" "R0" "input")
  pinInstance(-0.5003*modcount + 1.875 Ain0 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 4.625 Ain0 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 1.750 Ain1 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 3.875 Ain1 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 1.625 Ain2 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 3.125 Ain2 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 1.500 Ain3 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 2.375 Ain3 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 1.375 Ain4 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 1.625 Ain4 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 1.125 Bin0 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 0.875 Bin0 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 1.000 Bin1 "R0" "input")
  setMaster("opin symbol current")

  pinInstance(-3.0007500*modcount + 0.125 Bin1 "R0" "input")
  setMaster("ipin symbol current")

  pinInstance(-0.5003*modcount + 0.875 Bin2 "R0" "input")
  setMaster("opin symbol current")
pinInstance(-3.000:7.500*modcount - 0.625 Bin2 "R0" "input")
setMaster(" opin symbol current")

pinInstance(-0.590:3*modcount + 0.750 Bin3 "R0" "input")
setMaster(" opin symbol current")

pinInstance(-3.000:7.500*modcount - 1.375 Bin3 "R0" "input")
setMaster(" opin symbol current")

pinInstance(-0.500:3*modcount + 0.625 Bin4 "R0" "input")
setMaster(" opin symbol current")

pinInstance(-3.000:7.500*modcount - 2.125 Bin4 "R0" "input")

printf(Aout0 "Aout%d0" modcount)
printf(Aout1 "Aout%d1" modcount)
printf(Aout2 "Aout%d2" modcount)
printf(Aout3 "Aout%d3" modcount)
printf(Aout4 "Aout%d4" modcount)
printf(Bout0 "Bout%d0" modcount)
printf(Bout1 "Bout%d1" modcount)
printf(Bout2 "Bout%d2" modcount)
printf(Bout3 "Bout%d3" modcount)
printf(Bout4 "Bout%d4" modcount)

setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.875 Aout0 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 4.625 Aout0 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.750 Aout1 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 3.875 Aout1 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.625 Aout2 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 3.125 Aout2 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.500 Aout3 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 2.375 Aout3 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.375 Aout4 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 1.625 Aout4 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.125 Bout0 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 0.875 Bout0 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 1.000 Bout1 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount + 0.125 Bout1 "R0" "output")
setMaster(" opin symbol current")

pinInstance(1.625:3*modcount + 0.875 Bout2 "R0" "output")
setMaster(" opin symbol current")

pinInstance(4.500:7.500*modcount - 0.625 Bout2 "R0" "output")
setMaster(" opin symbol current")
pinInstance(1.625:3*modcount + 0.750 Bout3 " " R0 " output")
setMaster("ipin symbol current")
pinInstance(4.500:7.500*modcount - 1.375 Bout3 " " R0 " output")
setMaster("opin symbol current")
pinInstance(1.625:3*modcount + 0.625 Bout4 " " R0 " output")
setMaster("ipin symbol current")
pinInstance(4.500:7.500*modcount - 2.125 Bout4 " " R0 " output")
}

openSDA("GraphicsEditor skill schExtract")
save()

})

procedure (sch_mod_gen( type chip mod romnum)

;*******************************************************************************;
; This procedure creates a schematic for a mod cell.
; ARGUMENT:
;   type: Filter type( encoder, FIR_Filter and scaling.
;   chip: Name of chip.
;   mod: Name of mod cell.
;   romnum: Rom number cell.
;*******************************************************************************;

prog((modblock BIPSPcount offset)

change_EXT_SRC

; Create a symbol for BIPSP cell

create_BIPSP_symbol()

; Creating schematic for mod.

sprintf(modblock "%s/%s/%s " type chip mod)
graphEdit(" %s schematic current" modblock)
setLayer("bkgnd")
openSDA("GraphicsEditor addalldi")
openSDA("GraphicsEditor rm")

setMaster("ipin symbol current")
pinInstance(-0.500:3.000 " GPH11 " " R0 " input")
pinInstance(-0.500:2.875 " GPH12 " " R0 " input")
pinInstance(-0.500:1.875 " Ain0 " " R0 " input")
pinInstance(-0.500:1.750 " Ain1 " " R0 " input")
pinInstance(-0.500:1.625 " Ain2 " " R0 " input")
pinInstance(-0.500:1.500 "Ain3" "R0" "input")
pinInstance(-0.500:1.375 "Ain4" "R0" "input")
pinInstance(-0.500:1.125 "Bin0" "R0" "input")
pinInstance(-0.500:1.000 "Bin1" "R0" "input")
pinInstance(-0.500:0.875 "Bin2" "R0" "input")
pinInstance(-0.500:0.750 "Bin3" "R0" "input")
pinInstance(-0.500:0.625 "Bin4" "R0" "input")

setMaster("opin symbol current")

pinInstance(romnum*3+2.625:1.875 "Aout0" "R0" "output")
pinInstance(romnum*3+2.625:1.750 "Aout1" "R0" "output")
pinInstance(romnum*3+2.625:1.625 "Aout2" "R0" "output")
pinInstance(romnum*3+2.625:1.500 "Aout3" "R0" "output")
pinInstance(romnum*3+2.625:1.375 "Aout4" "R0" "output")
pinInstance(romnum*3+2.625:1.125 "Bout0" "R0" "output")
pinInstance(romnum*3+2.625:1.000 "Bout1" "R0" "output")
pinInstance(romnum*3+2.625:0.875 "Bout2" "R0" "output")
pinInstance(romnum*3+2.625:0.750 "Bout3" "R0" "output")
pinInstance(romnum*3+2.625:0.625 "Bout4" "R0" "output")

setLayer("wire")
line(-0.5:2.875 romnum*3+1.250:2.875)
line(-0.5:3.000 romnum*3+0.875:3.000)
line(-0.5:1.875 0:1.875)
line(-0.5:1.750 0:1.750)
line(-0.5:1.625 0:1.625)
line(-0.5:1.500 0:1.500)
line(-0.5:1.375 0:1.375)
line(-0.5:1.125 0:1.125)
line(-0.5:1.000 0:1.000)
line(-0.5:0.875 0:0.875)
line(-0.5:0.750 0:0.750)
line(-0.5:0.625 0:0.625)
line(romnum*3+2.125:1.875 romnum*3+2.625:1.875)
line(romnum*3+2.125:1.750 romnum*3+2.625:1.750)
line(romnum*3+2.125:1.625 romnum*3+2.625:1.625)
line(romnum*3+2.125:1.500 romnum*3+2.625:1.500)
line(romnum*3+2.125:1.375 romnum*3+2.625:1.375)
line(romnum*3+2.125:1.125 romnum*3+2.625:1.125)
line(romnum*3+2.125:1.000 romnum*3+2.625:1.000)
line(romnum*3+2.125:0.875 romnum*3+2.625:0.875)
line(romnum*3+2.125:0.750 romnum*3+2.625:0.750)
line(romnum*3+2.125:0.625 romnum*3+2.625:0.625)

; Placing BIPSP symbol in schematic.
;

for(BIPSPcount 0 romnum

setLayer("bkgnd")
setMaster(" BIPSP.s symbol current")
instance(3*BIPSPcount:0)
sprintf(BIPSPblock "  %s/%s/%s/%s/BIPSP%d symbol current" type chip mod
BIPSPcount )
setMaster(BIPSPblock)
;
setMaster("  FIR_Filter/chip1.s/mod31/BIPSP1 symbol current")
instance(3*BIPSPcount+0.375:0.500)
;
Creating wires.
;
setLayer("wire")

line(0.875+3*BIPSPcount:2.500 0.875+3*BIPSPcount:3.000)
line(1.250+3*BIPSPcount:2.500 1.250+3*BIPSPcount:2.875)
;if( (BIPSPcount < romnum) then
  line(2.125+3*BIPSPcount:1.750 3.000+3*BIPSPcount:1.750)
  line(2.125+3*BIPSPcount:1.250 3.000+3*BIPSPcount:1.250)
  line(2.125+3*BIPSPcount:1.000 3.000+3*BIPSPcount:1.000)
  line(2.125+3*BIPSPcount:0.875 3.000+3*BIPSPcount:0.875)
  line(2.125+3*BIPSPcount:0.750 3.000+3*BIPSPcount:0.750)
  line(2.125+3*BIPSPcount:0.625 3.000+3*BIPSPcount:0.625)
  else
  )
openSDA("GraphicsEditor skill schExtract")
save()
;
; create a symbol for mod cell
;
create_mod_symbol(type chip mod)
)
)

procedure( sng_rom_gen(rom_address)
  prog()
  ; This procedure creates a schematic for each rom.
  ; ARGUMENT:
  ;     rom_address: create a transistor in the rom location
  ;
  ;
  ;**************************************************

}
setMaster("sdalib/nmos symbol")

case(rom_address
  (0 instance(2:0.188 "" "r0"
    setLayer("text")
    label(2.375:0.25 "romloc 0" "" "stick" .05 )
  )
  (1 instance(2:1.188 "" "r0"
    setLayer("text")
    label(2.375:1.25 "romloc 1" "" "stick" .05 )
  )
  (2 instance(7:0.188 "" "r0"
    setLayer("text")
    label(7.375:0.25 "romloc 2" "" "stick" .05 )
  )
  (3 instance(7:1.188 "" "r0"
    setLayer("text")
    label(7.375:1.25 "romloc 3" "" "stick" .05 )
  )
  (4 instance(1:0.188 "" "r0"
    setLayer("text")
    label(1.375:0.25 "romloc 4" "" "stick" .05 )
  )
  (5 instance(1:1.188 "" "r0"
    setLayer("text")
    label(1.375:1.25 "romloc 5" "" "stick" .05 )
  )
  (6 instance(5:0.188 "" "r0"
    setLayer("text")
    label(5.375:0.25 "romloc 6" "" "stick" .05 )
  )
  (7 instance(5:1.188 "" "r0"
    setLayer("text")
    label(5.375:1.25 "romloc 7" "" "stick" .05 )
  )
  (8 instance(3:0.188 "" "r0"
    setLayer("text")
    label(3.375:0.25 "romloc 8" "" "stick" .05 )
  )
  (9 instance(3:1.188 "" "r0"
    setLayer("text")
    label(3.375:1.25 "romloc 9" "" "stick" .05 )
  )
  (10 instance(6:0.188 "" "r0"
    setLayer("text")
    label(6.375:0.25 "romloc 10" "" "stick" .05 )
  )
  (11 instance(6:1.188 "" "r0"
    setLayer("text")
    label(6.375:1.25 "romloc 11" "" "stick" .05 )
  )
  (12 instance(0:0.188 "" "r0"
    setLayer("text")
    label(0.375:0.25 "romloc 12" "" "stick" .05 )
  )
)
(26 instance(6.2.188 "" ""r0"")
   setLayer("text")
   label(6.375:2.25 "romloc 26" ""stick" .05)
)
(27 instance(6.3.188 "" ""r0"")
   setLayer("text")
   label(6.375:3.25 "romloc 27" ""stick" .05)
)
(28 instance(0.2.188 "" ""r0"")
   setLayer("text")
   label(0.375:2.25 "romloc 28" ""stick" .05)
)
(29 instance(0.3.188 "" ""r0"")
   setLayer("text")
   label(0.375:3.25 "romloc 29" ""stick" .05)
)
(30 instance(4.2.188 "" ""r0"")
   setLayer("text")
   label(4.375:2.25 "romloc 30" ""stick" .05)
)
(31 instance(4.3.188 "" ""r0"")
   setLayer("text")
   label(4.375:3.25 "romloc 31" ""stick" .05)
)
)
)
)

procedure sch_rom_gen(type chipname modnum BIPSNum n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12 n13 n14 n15 n16 n17 n18 n19 n20 n21 n22 n23 n24 n25 n26 n27 n28 n29 n30 n31)
;
; This procedure creates a schematic for each 8 by 4 ROM.
; ARGUMENT:
; type: Filter type. Three types is allowed (encoder,
;      FIR_Filter, scaling)
; chipname: Name of chip.
; modnum: Name of mod cell.
; BIPSNum: Name of BIPS cell.
; n0,n1 .. : rom content for rom location 0,1 ..
;
;
;*******************************************************************

prog( blockname rom_add)
change_EXT_SRC

declare(n[32])
declare(rem[32])
n[0]=n0
n[1]=n1
n[2]=n2
n[3]=n3
n[4]=n4
n[5]=n5
n[6]=n6
n[7]=n7
n[8]=n8
n[9]=n9
n[10]=n10
n[12]=n12
n[13]=n13
n[14]=n14
n[15]=n15
n[16]=n16
n[17]=n17
n[18]=n18
n[19]=n19
n[20]=n20
n[21]=n21
n[22]=n22
n[23]=n23
n[24]=n24
n[25]=n25
n[26]=n26
n[27]=n27
n[28]=n28
n[29]=n29
n[30]=n30
n[31]=n31

printf(blockname " %s/%s/%s/%s" type chipname modnum BIPSNum)

for(bit 0 4
    printf(blockname " %s/%s/%s/%s/ROM%d" type chipname modnum BIPSNum bit)

;
;Generating a symbol for each bit of ROM.
;
create_bit_rom_symbol(blockname bit)

;
; Generate schematic for ROM.
;

graphEdit("%s schematic current" blockname)
setLayer("bkgnd")
openSDA("GraphicsEditor addalldi")
openSDA("GraphicsEditor rm")

setMaster("sdalib/nmos symbol")
instance(7.5:.188 "IC0" "r0")
instance(7.5:0.812 "IC1" "r0")
instance(7.5:1.812 "IC2" "r0")
instance(752.812 "IC3" "r0")

setMaster("sdalib/pmos symbol")
instance(353.562 "lpmos" "r0")

setMaster("gnd symbol")
instance(775-0.375)
instance(775:0.625)
instance(775:1.625)
instance(775:2.625)

setMaster("ipin symbol")
pinInstance(0175 "R0" "r0" "input")
pinInstance(1175 "R1" "r0" "input")
pinInstance(2175 "R2" "r0" "input")
pinInstance(3175 "R3" "r0" "input")
pinInstance(4175 "R4" "r0" "input")
pinInstance(5175 "R5" "r0" "input")
pinInstance(6175 "R6" "r0" "input")
pinInstance(7175 "R7" "r0" "input")
pinInstance(75:0.188 "C0" "r0" "input")
pinInstance(75:0.812 "C1" "r0" "input")
pinInstance(75:1.812 "C2" "r0" "input")
pinInstance(75:2.812 "C3" "r0" "input")
pinInstance(353.562 "PREBAR" "r0" "input")

setMaster("opin symbol")
pinInstance(853.375 "EVAL" "r0" "output")

setMaster("vdd symbol")
instance(375:3.75)

;  Createing wires for rom schematic.
;
setLayer("wire")
line(0.25:3.375 8.562:3.375)
line(0.25:3 7.75:3)
line(0.25:2.375 8.375:2.375)
line(0.25:2 7.75:2)
line(0.25:1.375 8.375:1.375)
line(0.25:1 7.75:1)
line(0.25:0.375 8.375:0.375)
line(3.375:3.375 8.375:3.375)
line(0.25:0 7.75:0)

setLayer("bkgrnd")

if(bit == 0
for(rom_add 0 31
   rem[rom_add]=mod(n[rom_add] 2)
   if(rem[rom_add] == 0
sng_rom_gen(rom_add)
)

if(bit>0
for(rom_add 0 31
    n[rom_add]=n[rom_add]/2
    rem[rom_add]=mod(n[rom_add]/2)
    if(rem[rom_add] == 0
        sng_rom_gen(rom_add)
    )
)

setLayer("wire")

rem[12]=0
rem[13]=0
rem[28]=1
rem[29]=0

if (orow == 0) then
    line(0.3188 0:0.188)
else

if (orow == 0) then
    line(1.3188 1:0.188)
else

if (orow == 0) then
    line(2.3188 2:0.188)
else

if (orow == 0) then
    line(3.3188 3:0.188)
else

if (orow == 0) then
    line(4.3188 4:0.188)
else

.
if ( orrow == 0 )
    line(5:3.188 5:0.188)
else
    
    if ( orrow == 0 )
        line(6:3.188 6:0.188)
    else
        
    if ( orrow == 0 )
        line(7:3.188 7:0.188)
    else
        
setLayer("text")
sprint(from "ROM%d" bit)
    label(3:5 rom == "stick" .6)

setLayer("bkgrd")
save()

;
; create schematic for BIPSP cell.
;

sprintf(BIPSP_sch " %s/%s/%s/%s" type chipname modnum BIPSPnum)

;
; Create symbol 5 roms.
;
create_5rom_symbol(BIPSP_sch)

;
; Create schematics for 5 roms.
;
graphEdit("%s schematic current" BIPSP_sch )
openSDA("GraphicsEditor addalldi ")
openSDA("GraphicsEditor rm ")
setMaster("%s/ROM0 symbol current" BIPSP_sch )
instance(0:3.275 "ROM0" "R0")
setMaster("%s/ROM1 symbol current" BIPSP_sch )
instance(0:1.250 "ROM1" "R0")
setMaster("%s/ROM2 symbol current" BIPSP_sch )
instance(0:-0.875 "ROM2" "R0")
pinInstance(0.000:0.625 "C1" "r0" "input")
pinInstance(0.000:0.750 "C0" "r0" "input")
pinInstance(0.438:0.750 "PREBAR" "r0" "input")

setMaster("opin symbol")
pinInstance(0.875:0.000 "EVAL2" "r0" "output")

setMaster("ipin symbol")
pinInstance(0.000:-1.500 "R7" "r0" "input")
pinInstance(0.000:-1.625 "R6" "r0" "input")
pinInstance(0.000:-1.750 "R5" "r0" "input")
pinInstance(0.000:-1.875 "R4" "r0" "input")
pinInstance(0.000:-2.000 "R3" "r0" "input")
pinInstance(0.000:-2.125 "R2" "r0" "input")
pinInstance(0.000:-2.250 "R1" "r0" "input")
pinInstance(0.000:-2.375 "R0" "r0" "input")
pinInstance(0.000:-2.500 "C3" "r0" "input")
pinInstance(0.000:-2.625 "C2" "r0" "input")
pinInstance(0.000:-2.750 "C1" "r0" "input")
pinInstance(0.000:-2.875 "C0" "r0" "input")
pinInstance(0.438:-1.375 "PREBAR" "r0" "input")

setMaster("opin symbol")
pinInstance(0.875:-2.125 "EVAL3" "r0" "output")

setMaster("ipin symbol")
pinInstance(0.000:-3.625 "R7" "r0" "input")
pinInstance(0.000:-3.750 "R6" "r0" "input")
pinInstance(0.000:-3.875 "R5" "r0" "input")
pinInstance(0.000:-4.000 "R4" "r0" "input")
pinInstance(0.000:-4.125 "R3" "r0" "input")
pinInstance(0.000:-4.250 "R2" "r0" "input")
pinInstance(0.000:-4.375 "R1" "r0" "input")
pinInstance(0.000:-4.500 "R0" "r0" "input")
pinInstance(0.000:-4.625 "C3" "r0" "input")
pinInstance(0.000:-4.750 "C2" "r0" "input")
pinInstance(0.000:-4.875 "C1" "r0" "input")
pinInstance(0.000:-5.000 "C0" "r0" "input")
pinInstance(0.438:-3.500 "PREBAR" "r0" "input")

setMaster("opin symbol")
pinInstance(0.875:-4.250 "EVAL4" "r0" "output")
save()

forromcount 0 4
sprintf(romblock " %s/%s/%s/%s ROM%d schematic current" type chipname modnum BIPSPnum romcount)
graphEdit(romblock)
openSDA("GraphicsEditor skill schExtract")
save()
)
sprintf(BIPSPblock " %s/%s/%s/%s schematic current" type chipname modnum BIPSPnum)
graphEdit(BIPSPblock)
openSDA("GraphicsEditor skill schExtract")
save()
)
)

procedure( setupTechMid()  

; This procedure creates preplacement file, a skill file that  
; creates library cells used by place and route, and a edif  
; file.
; 
; 
; ;
; 
;

prog((stDIR sys1 sys2 sys3 sys4 sys5))

sprint(directory "  %s/%s" type chipname)
stDIR=isDir(directory)
if( stDIR then  else createDir(directory))

sprint(directory "  %s/%s.s" type chipname)
stDIR=isDir(directory)
if( stDIR then  else createDir(directory))

load("  skill/MDL.par")

; ;Run system command to create preplacement I/O file.
; 
numcoeff=length(rom)/length(moduli)
nummod=length(moduli)
preplaceio(purpose chipname numcoeff nummod)

; ; create skill file for schematic,symbol and module generator.
; 
MDL(rownum)

; ; Create a net property file
; 
netprop(purpose chipname rownum)
)
)
procedure( setup_tech()  

;**************************************************************************************************  
; This procedure creates preplacement file, a skill file that  
; creates library cells used by place and route, and a edif  
; file.  
;  
;**************************************************************************************************  

prog((stDIR sys1 sys2 sys3 sys4 sys5)  

declare(coefnum[300])  
declare(modnum[10])  

sprintf(directory "  
stDIR=lsDir(directory)  
if( stDIR then else createDir(directory))  

sprintf(coeffF "  
load(coeffF)  

;  
; Run system command to create skill file for module generator.  
;  
; sprintf(sys1 "  
; sprintf(sys2 "  
for(i 1 numcoeff  
; sprintf(temp "%d " coefnum[i])  
; sys2=strcat(sys2 temp)  
)

sprintf(sys3 "-m %d " nummod)  

sprintf(sys4 "  
for(i 1 nummod  
; sprintf(temp "%d " modnum[i])  
; sys4=strcat(sys4 temp)  
)

sprintf(sys5 "%s %s >  
system1=strcat(sys1 sys2 sys3 sys4 sys5)  
system(system1)  
system("  

; Run system command to create preplacement I/O file.  
;  
; preplaceio(type chipset numcoeff nummod)  

;  
; Run system command to create edif netlist file.  


; sprintf(sys1 " ex_c_dir/edif.out -m %d " nummod)
sprintf(sys2 "")
for(i 1 nummod
  sprintf(temp "%d " modnum[i])
  sys2=strcat(sys2 temp)
)
sprintf(sys3 "-%d %s > %s/%s/edif.netlist" rownum chipname type chipname)
system3=strcat(sys1 sys2 sys3 )
system(system3)
;system(" ex_c_dir/edif.out -m 1 27 -r 2 %s > %s/%s/edif.netlist" chipname type chipname )

;Run system command to create skill routine to create schematic automatically.
;

sprintf(directory " %s/%s.s" type chipname)
stdIR=isDir(directory)
if( stdIR then else createDir(directory))

sprintf(sys1 " ex_c_dir/main_sch_array.out -c %d " numcoeff)
  sprintf(sys2 "")
for(i 1 numcoeff
  sprintf(temp "%d " coeffnum[i])
  sys2=strcat(sys2 temp)
)
sprintf(sys3 "-m %d " nummod)
  sprintf(sys4 "")
for(i 1 nummod
  sprintf(temp "%d " modnum[i])
  sys4=strcat(sys4 temp)
)
sprintf(sys5 "%s %s > %s/%s/schematic.skill" type chipname type chipname)
system4=strcat(sys1 sys2 sys3 sys4 sys5)
system(system4)
;
; create doschematic.skill file.
;
  doschematic=outfile(" skill/doschematic.skill" "w")
  fprintf(doschematic "main_schematic(" "%s" \ "%s")" type chipname)
  close(doschematic)

;
; Create a net property file
;
  netprop(type chipname rownum)
)
)

procedure( create_array(block_name row col))

;This routine creates a programmed array of newmod cells.
;The user only needs to enter the rows and columns of the
; newmod array.

prog((output template personality setValue colcount rowcount count parity directory))

; Declare column and row array.

declare(c[10])
declare(r[10])

for(colcount 0 (col-1)
 c[colcount]=colcount
)

for(colcount col 9
 c[colcount]=0
)

rowcount=0
parity=0

for(count 1 row

; Run and compile "C" program masromrow.c.

sprintf(directory "%s" block_name)

makedir("masterromrow" directory)

status=system("cc c_dir/masromrow.c -lm -o c_dir/a.out; c_dir/a.out -c %d -p %d > struc_comp_dir/masterromrow/%s/personality" col parity block_name)

; Define output layout rep, template and personality file.

sprintf(output "masterromrow/%s/layout%d layout" block_name rowcount)

sprintf(template "masterromrow template")

sprintf(personality "masterromrow/%s/personality" block_name)

; Setting the graphicsEditorUnits per userUnit of layout rep to 2

graphEdit(output)

thisRep= getWindowRep()

property(thisRep,"graphicsEditorUnits per userUnit",2)

save()

; Edit template.

graphEdit(template)

; Editing template masterromrow masters property.

sprintf(setValue "P:masRomSp1 layout; T:masRomSp2 layout; 0:romcell/%s/rom%d layout; 1:romcell/%s/rom%d layout; 2:romcell/%s/rom%d layout; 3:romcell/%s/rom%d layout; 4:romcell/%s/rom%d layout; 5:romcell/%s/rom%d layout; 6:romcell/%s/rom%d layout; 7:romcell/%s/rom%d layout; 8:romcell/%s/rom%d layout; 9:romcell/%s/rom%d layout;"
    property("masterRomRow" "masters" setValue)

    if((parity==0) then
        parity=1
    else
        parity=0
    )

    for(colcount 0 col
        c[colcount]=c[colcount] + col
    )

    save()

;Execute structure compiler.

    compileArray(output template personality)

    save()
    rowcount++
    )

    for(rowcount 0 (row-1)
        if(rowcount=rowcount
    )

    for(rowcount row 9
        if(rowcount=0
    )

;Run and compile "C" program masrom.c.

    makedir("masterRom" directory)

    status=system("cc c_dir/masrom.c -lm -o c_dir/a.out; c_dir/a.out -r %d > struc_comp_dir/masterRom/%s/personality" row block_name)

;Define output layout rep, template and personality file.

    sprintf(output "masterRom/%s layout" block_name)
    sprintf(personality "masterRom/%s/personality" block_name)
    sprintf(template "masterRom template")

;setting the graphicsEditorUnits per userUnit of layout rep to 2

    graphEdit(output)
    thisRep= getWindowRep()
    property(thisRep,"graphicsEditorUnits per userUnit",2 )
    save()

;Edit template plane.
```c
graphEdit(template)

sprintf(setValue "0:masterromrow/%s/layout%d layout; 1:masterromrow/%s/layout%d layout;
2:masterromrow/%s/layout%d layout; 3:masterromrow/%s/layout%d layout;
4:masterromrow/%s/layout%d layout; 5:masterromrow/%s/layout%d layout;
6:masterromrow/%s/layout%d layout; 7:masterromrow/%s/layout%d layout;
8:masterromrow/%s/layout%d layout; 9:masterromrow/%s/layout%d layout; "

property("masterRom" "masters" setValue)

save()

; Execute structure compiler.

compileArray(output template personality)

save()

create_newmod(directory row col)

)

)

procedure(create_newmod( block_name rows cols))

prog((output template personality setValue setValue1 setValue2 setValue2 thisRep directory))

sprintf(directory "%s" block_name)

mkdir("masternewmod" directory)

status=system("cc c_dir/newmodplace.c -lm -o c_dir/a.out; c_dir/a.out -c %d -r %d >
struc_comp_dir/masternewmod/cell%s/personality" cols rows block_name)

sprintf(output "masternewmod/%s layout" block_name)

sprintf(template "masternewmod template")

sprintf(personality "masternewmod/cell%s/personality" block_name)

; Setting the graphicsEditorUnits per userUnit of layout rep to 2

graphEdit(output)

thisRep= getWindowRep()

property(thisRep,"graphicsEditorUnits per userUnit",2)

save()

graphEdit(template)

sprintf(left "X:space2 layout; P:leftpath layout; W:onemiconL layout;")

sprintf(newmod "L:newmod3L layout; R:newmod3R layout; N:newmod3 layout; l:newmod3L layout; r:newmod3R layout; n:newmod3 layout;")
```
sprint(right  "X:space1 layout; W:onemicroR layout; Q:rightpath layout;")

property("leftside"  "masters" left)
property("masternewmod"  "masters" newmod)
property("rightside"  "masters" right)

save()

; Promoting pin to a higher level.

if(mod(rows, 2) != 0 then

    sprint(setValue "in=right<rows-1:rows-1>
Ain0,Ain1,Ain2,Ain3,Ain4,Bin0,Bin1,Bin2,Bin3,Bin4,PHI1,PHI2;
out=left<0:0>
Aout0,Aout1,Aout2,Aout3,Aout4,Bout0,Bout1,Bout2,Bout3,Bout4,PHO1,PHO2,gnd!,vdd!;")
    property("masternewmod"  "pins" setValue)

else

    sprint(setValue "out=right<0:0>
Aout0,Aout1,Aout2,Aout3,Aout4,Bout0,Bout1,Bout2,Bout3,Bout4,PHO1,PHO2,gnd!,vdd!;
in=right<rows-1:rows-1>
Ain0,Ain1,Ain2,Ain3,Ain4,Bin0,Bin1,Bin2,Bin3,Bin4,PHI1,PHI2;")
    property("masternewmod"  "pins" setValue)

)

save()

compileArray(output template personality)

save()

; delete previous made layout reps.

status=system(" rm -r  struc_comp_dir/mastercell/%s layout" block_name)

sprint(output "mastercell/%s layout current" block_name)
    sprint(template "mastercell template")
    sprint(personality "none")

graphEdit(output)
    thisRep= getWindowRep()
    property(thisRep,"graphicsEditorUnits per user!Unit",2)
    save()

graphEdit(template)

; Promoting pins to a higher level.

if(mod(rows, 2) != 0 then
sprintf(setValue "Ain0=right<rows-1:rows-1> Ain0; Ain1=right<rows-1:rows-1> Ain1; Ain2=right<rows-1:rows-1> Ain2; Ain3=right<rows-1:rows-1> Ain3; Ain4=right<rows-1:rows-1> Ain4; Bin0=right<rows-1:rows-1> Bin0; Bin1=right<rows-1:rows-1> Bin1; Bin2=right<rows-1:rows-1> Bin2; Bin3=right<rows-1:rows-1> Bin3; Bin4=right<rows-1:rows-1> Bin4; PH1=right<rows-1:rows-1> PH1; PH2=right<rows-1:rows-1> PH2; Aout0=left<0:0> Aout0; Aout1=left<0:0> Aout1; Aout2=left<0:0> Aout2; Aout3=left<0:0> Aout3; Aout4=left<0:0> Aout4; Bout0=left<0:0> Bout0; Bout1=left<0:0> Bout1; Bout2=left<0:0> Bout2; Bout3=left<0:0> Bout3; Bout4=left<0:0> Bout4; PHO1=left<0:0> PHO1; PHO2=left<0:0> PHO2; gnd1=left<0:0> gnd1; vdd1=left<0:0> vdd1;")
  property("masternewmod" "pins" setValue)
else
  sprintf(setValue "Ain0=right<rows-1:rows-1> Ain0; Ain1=right<rows-1:rows-1> Ain1; Ain2=right<rows-1:rows-1> Ain2; Ain3=right<rows-1:rows-1> Ain3; Ain4=right<rows-1:rows-1> Ain4; Bin0=right<rows-1:rows-1> Bin0; Bin1=right<rows-1:rows-1> Bin1; Bin2=right<rows-1:rows-1> Bin2; Bin3=right<rows-1:rows-1> Bin3; Bin4=right<rows-1:rows-1> Bin4; PH1=right<rows-1:rows-1> PH1; PH2=right<rows-1:rows-1> PH2; Aout0=right<0:0> Aout0; Aout1=right<0:0> Aout1; Aout2=right<0:0> Aout2; Aout3=right<0:0> Aout3; Aout4=right<0:0> Aout4; Bout0=right<0:0> Bout0; Bout1=right<0:0> Bout1; Bout2=right<0:0> Bout2; Bout3=right<0:0> Bout3; Bout4=right<0:0> Bout4; PHO1=right<0:0> PHO1; PHO2=right<0:0> PHO2; gnd1=right<0:0> gnd1; vdd1=right<0:0> vdd1;")
  property("masternewmod" "pins" setValue)
)

; Indicate layout rep for masternewmod and masterRom plane. These
; planes are located in mastercell template rep.

  sprintf(setvalue1 "masternewmod/%s layout", block_name)
  property("masternewmod" "masters" setvalue1)

  sprintf(setvalue2 "leftpathspace layout")
  property("leftpathspace" "masters" setvalue2)

  sprintf(setvalue3 "masterRom/%s layout", block_name)
  property("masterRom" "masters" setvalue3)

save()

compileArray(output template personality)

save()

; This part of the program creates a prboundary layer that will be
; placed over the macrocell. This is needed for the conversion
; between layout to abstract rep.

  x_cord=244.5 + (cols*709 )
  y_cord=4 + rows*742

  graphEdit(output)
;This skill routine programs one newmod cell. The user supplies the contents
;of Rom and the blockname of the stored layout. The Rom contains an eight by four word.
;Each word is five bits.

procedure( create_Rom( block_name count n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12 n13 n14 n15 n16 n17
 n18 n19 n20 n21 n22 n23 n24 n25 n26 n27 n28 n29 n30 n31)

 prog((output template personality setValue status status1 thisRep directory)

 ;The system command lets us run "C" programs. We first compile the "C" program
 ;personmskil c and we execute the executable file called a.out.
 ;printf(directory "%s" block_name)
 ;mkdir("romcell" directory)

 status=system(" ex_c_dir/prog_rom.out -n %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %
property("space2" "masters" setvalue2)
property("bottomspace" "masters" setvalue3)
property("topspace" "masters" setvalue3)
property("col0" "masters" setvalue4)
property("col1" "masters" setvalue4)
property("col2" "masters" setvalue4)
property("col3" "masters" setvalue4)

save()

;The structure compiler is executed.

compileArray(output template personality)

save()

)

)

# Log file for program 'opencds -layers layers/display/current'.
# # Wed Apr 10 09:20:03 1991
# # Process id = 9569
#

# This log file run place and route automatically.
#
#

(614, 619, 1, "GraphicsEditor AutoLayout layout env interruptUser?=-no");
(10, 10, 2, "");
(10, 10, 2, "");
#(614, 614, 1, "GraphicsEditor setenv blockName= encoder representationName=autoLayout
revisionName= current ");
#(10, 10, 2, "");
#(10, 10, 2, "");
(614, 614, 1, "GraphicsEditor AutoLayout placeenv placementSnapGrid=1");
(10, 10, 2, "");
(10, 10, 2, "");
(614, 614, 1, "GraphicsEditor AutoLayout layoutManager open");
(10, 10, 2, "");
(10, 10, 2, "");
(614, 614, 1, "setmenu AutoLayout/");
(10, 10, 2, "");
(10, 10, 2, "");

(614, 614, 0, "GraphicsEditor skill load \\
skill/doplaceio.skill\"");
(10, 10, 2, "");
(614, 614, 0, "GraphicsEditor skill macroplace");
(10, 10, 2, "");
(614, 614, 0, "GraphicsEditor skill iocorners");
(10, 10, 2, "");
(614, 614, 0, "GraphicsEditor skill iouj")
(10, 10, 2, ")
(614, 614, 0, "GraphicsEditor skill chagn")
(10, 10, 2, ")
(614, 614, 0, "GraphicsEditor skill load \"
(10, 10, 2, ")
(614, 614, 0, "GraphicsEditor skill global")
(300, 750, 2, ")
(10, 10, 2, ")
(614, 614, 0, "GraphicsEditor skill load \"
(10, 10, 2, ")
(skill/dogetprop.skill"");
APPENDIX II

C Programs
#include <stdio.h>
#include <math.h>

/* The following program creates a Edif file   */

main(argc,argv)
   int argc;
   char *argv[];
{
   int coeffcount[1], modcount[1], coeff[100], modulo[5];
   int count,count1;
   char *a;
   void Edif();

   if(argv[1][0] == '-' & & argv[1][1] == 'c')
   {
      coeffcount[1]=atoi(argv[2]);
      for(count=2, count1=0; count+1<3+coeffcount[1]; count++, count1++)
      {
         coeff[count1]=atoi(argv[count+1]);
      }
   }

   if(argv[3+coeffcount[1]][0] == '-' & & argv[3+coeffcount[1]][1] == 'm')
   {
      modcount[1]=atoi(argv[4+coeffcount[1]]);
      for(count=4+coeffcount[1],count1=0; count+1<5+coeffcount[1]+modcount[1]; count++, count1++)
      {
         modulo[count1]=atoi(argv[count+1]);
      }
   }

   Edif(modulo,modcount,coeff,coeffcount,argv[7]);
}

void Edif(modulo,modcount,coeff,coeffcount,cellname)

int modcount[1],coeffcount[1],modulo[5],coeff[100];
char *cellname;
{
   int count1,macrocount,count2,incount,modulocount,outcount;
   int iocount,inputcount,outputcount,count;

   printf("(EDIF example \n");
   printf(" (EdifVersion 2 0 0) \n");
   printf(" (EdifLevel 0) \n");
   printf(" (keywordMap \n");
print("(keywordLevel 0)\n");
print(" )\n");
print(" (status \n");
print(" (written \n");
print(" (timestamp 1990 8 21 4 0 \n");
print(" (program \"test_design\") \n");
print(" )\n");
print(" )\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf(" external cmoslib \n");
printf(" (ediLevel 0) \n");
printf(" (technology \n");
printf(" (numberDefinition \n");
printf(" ) \n ");
printf(" ) \n");
printf("\n");
printf(" (cell inpadx \n");
printf(" (celltype GENERIC) \n");
printf(" (view symbol \n");
printf(" (viewType NETLIST) \n");
printf(" (interface \n");
printf(" (port OUT (direction output)) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf("\n");
printff(\n");
printf(" (cell outpadx \n");
printf(" (celltype GENERIC) \n");
printf(" (view symbol \n");
printf(" (viewType NETLIST) \n");
printf(" (interface \n");
printf(" (port IN (direction input)) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf("\n");
printf("\n");
printf(" (cell gp\n");
printf(" (celltype GENERIC) \n");
printf(" (view symbol \n");
printf(" (viewType NETLIST) \n");
printf(" (interface \n");
printf(" (port (rename gnd \"gnd\") (direction input)) \n");
printf(" (port gndcore (direction input)) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf("\n");
printf("\n");
printf(" (cell pp\n"));
printf(" (celltype GENERIC) \n");
printf(" (view symbol \n");
printf(" (viewType NETLIST) \n");
printf(" (interface \n");
printf(" (port (rename vdd \"vdd\") (direction input)) \n");
printf(" (port vddcore (direction output)) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");

for(macrocount=0; macrocount<modcount[1]; macrocount++)
{
  printf(" (cell mod%d \n", modul=[macrocount]);
  printf(" (celltype GENERIC) \n");
  printf(" (view symbol \n");
  printf(" (viewType NETLIST) \n");
  printf(" (interface \n");
  printf(" (port PHI1 (direction input)) \n");
  printf(" (port PHI2 (direction input)) \n");
  printf(" (port Ain0 (direction input)) \n");
  printf(" (port Ain1 (direction input)) \n");
  printf(" (port Ain2 (direction input)) \n");
  printf(" (port Ain3 (direction input)) \n");
  printf(" (port Ain4 (direction input)) \n");
  printf(" (port Bin0 (direction input)) \n");
  printf(" (port Bin1 (direction input)) \n");
  printf(" (port Bin2 (direction input)) \n");
  printf(" (port Bin3 (direction input)) \n");
  printf(" (port Bin4 (direction input)) \n");
  printf(" (port PHO1 (direction output)) \n");
  printf(" (port PHO2 (direction output)) \n");
  printf(" (port Aout0 (direction output)) \n");
  printf(" (port Aout1 (direction output)) \n");
  printf(" (port Aout2 (direction output)) \n");
  printf(" (port Aout3 (direction output)) \n");
  printf(" (port Aout4 (direction output)) \n");
  printf(" (port Bout0 (direction output)) \n");
  printf(" (port Bout1 (direction output)) \n");
  printf(" (port Bout2 (direction output)) \n");
  printf(" (port Bout3 (direction output)) \n");
  printf(" (port Bout4 (direction output)) \n");
  printf(" ) \n");
  printf(" ) \n");
  printf(" ) \n");
  printf("\n");
  printf("\n");
}

printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf(" (library CHIP_dir \n");
printf(" (ediLevel 0) \n");
printf(" (technology \n");
printf(" (numberDefinition \n");
printf(" ) \n");
printf(" ) \n");
printf("\n");
printf(" (Cell %s \n", cellname);
printf(" (celltype GENERIC) \n");
printf(" (view netlist \n");
printf(" (viewType NETLIST) \n");
printf(" (interface) \n");
printf(" (contents) \n");
printf("\n");

for(macrocount=0; macrocount<modcount[1]; macrocount++)
{
    printf(" (instance mod%d \n", modulo[macrocount]);
    printf(" (viewRef symbol \n");
    printf(" (cellRef mod%d \n", modulo[macrocount]);
    printf("    (libraryRef cmos3lib) \n");
    printf(" ) \n");
    printf(" ) \n");
    printf(" ) \n");
    printf("\n");
}

for(inputcount=0; inputcount<modcount[1]*5; inputcount++)
{
    printf(" (instance input%d \n", inputcount);
    printf(" (viewRef symbol \n");
    printf(" (cellRef inpadx\n");
    printf("    (libraryRef cmos3lib) \n");
    printf(" ) \n");
    printf(" ) \n");
    printf(" ) \n");
    printf("\n");
}

for(outputcount=0; outputcount<modcount[1]*10; outputcount++)
{
    printf(" (instance output%d \n", outputcount);
    printf(" (viewRef symbol \n");
    printf(" (cellRef outpadx\n");
    printf("    (libraryRef cmos3lib) \n");
    printf(" ) \n");
    printf(" ) \n");
}
printf("   ) \n");
printf("  \n");
}

printf("   (instance phi1 \n");
printf("   (viewRef symbol \n");
printf("   (cellRef inpady) \n");
printf("   (libraryRef cmos3lib) \n");
printf("   ) \n");
printf("  \n");
printf("   ) \n");
printf("  \n");

printf("   (instance phi2 \n");
printf("   (viewRef symbol \n");
printf("   (cellRef inpady) \n");
printf("   (libraryRef cmos3lib) \n");
printf("   ) \n");
printf("  \n");
printf("   ) \n");
printf("  \n");

printf("   (instance pho1 \n");
printf("   (viewRef symbol \n");
printf("   (cellRef outpady) \n");
printf("   (libraryRef cmos3lib) \n");
printf("   ) \n");
printf("  \n");
printf("   ) \n");
printf("  \n");

printf("   (instance pho2 \n");
printf("   (viewRef symbol \n");
printf("   (cellRef outpady) \n");
printf("   (libraryRef cmos3lib) \n");
printf("   ) \n");
printf("  \n");
printf("   ) \n");
printf("  \n");

printf("   (instance gpx \n");
printf("   (viewRef symbol \n");
printf("   (cellRef gpx) \n");
printf("   (libraryRef cmos3lib) \n");
printf("   ) \n");
printf("  \n");
printf("   ) \n");
printf("  \n");

printf("   (instance ppx \n");
printf("   (viewRef symbol \n");
printf("   (cellRef ppx) \n");
printf("   (libraryRef cmos3lib) \n");
printf("   ) \n");
printf("  \n");
printf("   ) \n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");

for(modulocount=0; modulocount<modcount[1]; modulocount++)
{
    for(count2=0,incount=0; count2<5; count2++)
    {
        printf(" (net in%d \n",incount+(modulocount*10));
        printf(" (joined \n");
        printf(" (portRef OUT (instanceRef input%d) \n",incount+(modulocount*5));
        printf(" (portRef Ain%d (instanceRef mod%d)) \n",count2,modulo[modulocount]);
        printf(" ) \n");
        printf(" ) \n");
        printf("\n");
        printf("\n");
incount++;
    }

    for(count2=0,outcount=0; count2<5; count2++)
    {
        printf(" (net out%d \n",outcount+(modulocount*10));
        printf(" (joined \n");
        printf(" (portRef IN (instanceRef output%d)) \n",outcount+(modulocount*10));
        printf(" (portRef Aout%d (instanceRef mod%d)) \n",count2,modulo[modulocount]);
        printf(" ) \n");
        printf(" ) \n");
        printf("\n");
        printf("\n");
        outcount++;
    }

    for(count2=0; count2<5; count2++)
    {
        printf(" (net out%d \n",outcount+(modulocount*10));
        printf(" (joined \n");
        printf(" (portRef IN (instanceRef output%d)) \n",outcount+(modulocount*10));
        printf(" (portRef Bout%d (instanceRef mod%d)) \n",count2,modulo[modulocount]);
        printf(" ) \n");
        printf(" ) \n");
        printf("\n");
        printf("\n");
        outcount++;
    }
}

printf(" (net PHI1 \n");
printf(" (joined \n");
printf("  (portRef OUT (instanceRef phi1)) \n");
for(modulocount=0; modulocount<modcount[1]; modulocount++)
{
  printf("  (portRef PHI1 (instanceRef mod%d)) \n",modulo[modulocount]);
  incout++;
}
printf("  ) \n");
printf("  ) \n");
printf("\n");
printf("  (net PHI2 \n");
printf("  (joined \n");
printf("  (portRef OUT (instanceRef phi2)) \n");
for(modulocount=0; modulocount<modcount[1]; modulocount++)
{
  printf("  (portRef PHI2 (instanceRef mod%d)) \n",modulo[modulocount]);
  incout++;
}
printf("  ) \n");
printf("  ) \n");
printf("\n");
printf("  (net PHO1 \n");
printf("  (joined \n");
printf("  (portRef IN (instanceRef pho1)) \n");
for(modulocount=0; modulocount<modcount[1]; modulocount++)
{
  printf("  (portRef PHO1 (instanceRef mod%d)) \n",modulo[modulocount]);
  incout++;
}
printf("  ) \n");
printf("  ) \n");
printf("\n");
printf("  (net PHO2 \n");
printf("  (joined \n");
printf("  (portRef IN (instanceRef pho2)) \n");
for(modulocount=0; modulocount<modcount[1]; modulocount++)
{
  printf("  (portRef PHO2 (instanceRef mod%d)) \n",modulo[modulocount]);
  incout++;
}
printf("  ) \n");
printf("  ) \n");
printf("\n");
printf("\n");
printf("  (net (rename vdd "vddl") \n");
printf("  (joined \n");
printf("  (portRef vdddcore (instanceRef ppx)) \n");
printf("  ) \n");
printf(" ) \n");
printf(" (net (rename gnd "gnd!" ) \n");
printf(" (joined \n"));
printf(" (portRef gndcore (instanceRef gp)) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
printf(" ) \n");
}

#include <stdio.h>
#include <math.h>

/*****************************************************************************/
/*
 /* This program creates a skill program called main_array.skill. This */
 /* skill program is used to automatically create library components */
 /* that place and route will use. */
/*
 /* OPTIONS: -c coeffnum coef0 coef1 ... */
 /* coeffnum: The number of coefficients used in FIR */
 /* coef0 coef1 ...: The actual coefficients used. */
/*
 /* -m modnum mod0 mod1 ... */
 /* modnum: The number of modulo used in FIR Filter. */
 /* mod0 mod1 ...: The actual modulo used. */
/*
 /* Written by Martin Jaekel */
/*
/*****************************************************************************/

tmain(argc,argv)
  int argc;
  char *argv[];
{
  int coeff[100], modulo[5];
  void Dec_to_Mod();
  int temp,count1,count,coeffnum,modnum;

  /******************************************************************************/
  /* options to c program is declared here. */
  /******************************************************************************/

  if( argv[1][0] == '-' && argv[1][1] == 'c' )
```c
void Dec_to_Mod(coeff, modulo, modnum, coeffnum)

/***************************************************************
/* The Dec_to_Mod routine simple converts decimal to modulo      */
/* The decimal can be positive or negative.                   */
/* Arguments of function is                                    */
/* coeff: The actual coefficients used in FIR Filter.        */
/* modulo: The actual modulo used in FIR Filter.             */
/* modnum: The number of modulo used in FIR_Filter.         */
/* coeffnum: The number of coefficients used in FIR Filter. */
/***************************************************************

int coeff[100], modulo[5];
int *modnum,*coeffnum;

{ int i,row,modcount,coeffcount,count3,count4,count5,romcount;
```
printf("procedure( n_in_array())\n");
printf("\n");

for(modcount=0; modcount<modnum; modcount++)
{
    for(coeffcount=0; coeffcount<coeffnum; coeffcount++)
    {
        for(count5=0; count5<5; count5++)
        {

            printf("\ncreate_rom("mod%d" %d ", modulo[modcount],coeffcount5+count5);
            if(coeff[coeffcount]>0)
            {
                for(i=0; coeff[coeffcount]>modulo[modcount]; i++)
                {
                    coeff[coeffcount]=coeff[coeffcount]-modulo[modcount];
                }
            }else
            {
                for(i=0; coeff[coeffcount]<0; i++)
                {
                    coeff[coeffcount]=coeff[coeffcount]+modulo[modcount];
                }
            }

            for(count3=0; count3<32; count3++)
            {
                romcount=coeff[coeffcount]+count3;
                if(romcount>=modulo[modcount])
                {
                    romcount=romcountmodulo[modcount];
                }
                printf("%d ", romcount);
            }
            printf("\n");
        }
        printf("\ncreate_array(\"mod%d" %d )\n", modulo[modcount], *coeffnum);
        printf("\n");

    }
}

#include<stdio.h>
#include<math.h>

main()

/* This program converts decimal to binary */

{ int num[2][2], row, col, iteration, temp;
char charray[2][2][6];

num[0][0]=30;
num[0][1]=12;
num[1][0]=10;
num[1][1]=25;

for(col=0;col<2;col++)
  for(row=0;row<2;row++)
  {
    iteration=4;
    printf("iteration= %d\n",iteration);
    temp=num[row][col];

    while(iteration)
    {
      if ( temp>=(int)pow(2.0,(double)iteration))
      {
        charray[row][col][4-iteration]='1';
        temp=temp - (int)pow(2.0,(double)(iteration));
        iteration--;
      } /* end if statement */
      else
      {
        charray[row][col][4-iteration]='0';
        iteration--;
      } /* end else if statement */
    } /* end while loop */

    if(temp==0)
      charray[row][col][4] = '0';
    else
      charray[row][col][4] = '1';

    printf("charray[%d][%d]=%s\n",row,col,charray[row][col]);
  } /* end for statement */

for(col=0;col<2;col++)
  for(row=0;row<2;row++)
    printf("binnum[%d][%d]= %s\n",row,col,charray[row][col]);
} /* end Dec_to_Bin */

#include <stdio.h>
main(argc,argv)
  int argc;
  char *argv[];
/* This program creates a personality file for masterRom template plane. */
/* The purpose of this personality file is to program the entire newmod array. */
/* It uses the rompattern created for each row in masterromrow directory. */
/* The symbols used in the personality file are numbers from 0 to 9. */

int count, nrsrows, row;
int array[10][2];

/* Option -r is the number of rows of the newmod array. */
/* The value entered must be numerical */

for(count=1; count<argc; count++, count++)
{
  if(argv[count][0] == '-' && argv[count][1] == 'r')
  {
    nrsrows=atoi(argv[count+1]);
  }
}

/* This part of the program prints out character to the standard output. */

printf("bias row \n");
printf("plane masterRom \n");

for(row=0; row<nrsrows; row++)
{
  printf("%d\n", row);
}

} /* end main */

#include <stdio.h>
#include <math.h>

void dec_to_mod(coefcount, coeff, modcount, modulo)

int coeff[100], modulo[5];
int modcount, coefcount;

{ int i, row, count1, count2, count3, count4, count5, romcount;

  printf("procedure( main_array()\n");
  printf("\n");

  for(count1=0; count1<modcount; count1++)
  {
    for(count2=0; count2<coefcount; count2++)
{ 
    for(count5=0; count5<5; count5++)
    {
        if(count5>0)
        {
            coeff[count2]=coeff[count2]*2;
            for(i=0; coeff[count2]>=modulo[count1]; i++)
            {
                coeff[count2]=coeff[count2]-modulo[count1];
            }
        }
        printf("\ncreate_Rom("mod%d" %d ", modulo[count1],count2*5+count5);
        if(coeff[count2]>0)
        {
            for(i=0; coeff[count2]>modulo[count1]; i++)
            {
                coeff[count2]=coeff[count2]-modulo[count1];
            }
        }
        else
        {
            for(i=0; coeff[count2]<0;i++)
            {
                coeff[count2]=coeff[count2]+modulo[count1];
            }
        }
    }

    for(count3=0; count3<32; count3++)
    {
        romcount=coeff[count2]+count3;
        if(romcount>=modulo[count1])
        {
            romcount=romcount-modulo[count1];
        }
        printf("%d ",romcount);
        printf("\n");
    }
    printf("\ncreate_array("mod%d" %d 5 \n", modulo[count1],coeffcount);
    printf("\n");
}

#include <stdio.h>
#include <math.h>

void edif(coeffcount,coeff,modcount,modulo)

int modcount,coeffcount,modulo[5],coeff[100];
{
int count1, macrocount, count2, incount, modulocount, outputcount;
int locount, inputcount, outputcount, count;

printf("(edef example\n");
printf(" (status \n");
printf(" (EdifVersion 1 1 0)\n");
printf(" (version 0)\n");
printf(" (written \n");
printf(" (timestamp 1990 2 26 12 00 00)\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" )\n");
printf(" (library edif3lib \n"));
printf(" (Cell filename \n"));
printf(" (view netlist autolayout \n"));
printf(" (interface)\n");
printf(" )\n");
printf(" )\n");
printf(" (contents)\n");

for(macrocount=0; macrocount<modcount; macrocount++)
{
    printf(" (instance (qualify edif3lib mod%d) abstract mod%d(property sda status \"unplaced\")\n", modulo[macrocount], modulo[macrocount]);
}

for(inputcount=0; inputcount<modcount*5; inputcount++)
{
    printf(" (instance (qualify edif3lib inpax) abstract inpax%d(property sda status \"unplaced\")\n", inputcount);
}

for(outputcount=0; outputcount<modcount*10; outputcount++)
{
    printf(" (instance (qualify edif3lib outpax) abstract outpax%d(property sda status \"unplaced\")\n", outputcount);
}

    printf(" (instance (qualify edif3lib inpax) abstract phi1(property sda status \"unplaced\")\n");
    printf(" (instance (qualify edif3lib inpax) abstract phi2(property sda status \"unplaced\")\n");
    printf(" (instance (qualify edif3lib outpax) abstract pho1(property sda status \"unplaced\")\n");
    printf(" (instance (qualify edif3lib outpax) abstract pho2(property sda status \"unplaced\")\n");
    printf(" (instance (qualify edif3lib gpax) abstract gpax(property sda status \"unplaced\")\n");
printf(" (instance (qualify edif3lib ppx) abstract ppx(property sda status "unplaced")).")
"
";  
printf(" \n");  
printf("\n");

for(modulocount=0; modulocount<modcount; modulocount++)
{
    for(count2=0; count2<count2<5; count2++)
    {
        printf(" (net in%d (joined (qualify input%d OUT))(qualify mod%d Ain%d))\n",  
            incount+(modulocount*10),incount+(modulocount*5),modulo[modulocount],count2);  
        printf(" (property sda type "signal")\n");  
        printf(" \n");  
        incount++;  
    }
}

for(count2=0; outcount=0; count2<5; count2++)
{
    printf(" (net out%d (joined (qualify output%d IN))(qualify mod%d Aout%d))\n",  
        outcount+(modulocount*10),outcount+(modulocount*5),modulo[modulocount],count2);  
    printf(" (property sda type "signal")\n");  
    printf(" \n");  
    outcount++;  
}

for(count2=0; count2<5; count2++)
{
    printf(" (net out%d (joined (qualify output%d IN))(qualify mod%d Bout%d))\n",  
        outcount+(modulocount*10),outcount+(modulocount*5),modulo[modulocount],count2);  
    printf(" (property sda type "signal")\n");  
    printf(" \n");  
    outcount++;  
}

printf("\n");  
printf("\n");  
printf("\n");

printf(" (net vdd \n");  
printf(" (joined (qualify ppx vddcore)(qualify ppx vdd)(qualify gpv vdd))\n");  
printf(" (qualify phi1 vdd)(qualify phi2 vdd)\n");  
printf(" (qualify pho1 vdd)(qualify pho2 vdd)\n");
printf("\n");

count=0;  
for(count<count<count++)
{
    printf(" (qualify input%d vdd)", count);  
    printf(" (qualify input%d vdd)", count+1);  
    printf(" (qualify input%d vdd)\n", count+2);  
    printf(" (qualify input%d vdd)\n", count+3);  
    printf(" (qualify input%d vdd)\n", count+4);
count=0;
for(count<count;count++)
{
    printf("(qualify output%d vdd)\n", 10*count);
    printf("(qualify output%d vdd)\n", 1+10*count);
    printf("(qualify output%d vdd)\n", 2+10*count);
    printf("(qualify output%d vdd)\n", 3+10*count);
    printf("(qualify output%d vdd)\n", 4+10*count);
    printf("(qualify output%d vdd)\n", 5+10*count);
    printf("(qualify output%d vdd)\n", 6+10*count);
    printf("(qualify output%d vdd)\n", 7+10*count);
    printf("(qualify output%d vdd)\n", 8+10*count);
    printf("(qualify output%d vdd)\n", 9+10*count);
}

printf("\n");
printf("(property sda type \"supply\")\n");
printf(")(Global vdd)\n");
printf(")(criticalSignal 105 vdd)\n");
printf("\n");

printf("(net gnd \n");
printf("(joined (qualify gp[x gndcore)(qualify gp[x gnd)(qualify pp[x gnd)\n");
printf(")(qualify phi1 gnd)(qualify phi2 gnd)\n");
printf(")(qualify phi1 gnd)(qualify phi2 gnd)\n");
printf("(\n");

for(count2=0;count2<count;count2++)
{
    printf("(qualify mod%d Bin0)(qualify mod%d Bin1)(qualify mod%d Bin2)\n", 
modulo[count2],modulo[count2],modulo[count2]);
    printf("(qualify mod%d Bin3)(qualify mod%d 
modulo[count2],modulo[count2]);
}

count=0;
for(count<count;count++)
{
    printf("(qualify input%d gnd)\n", 5*count);
    printf("(qualify input%d gnd)\n", 1+5*count);
    printf("(qualify input%d gnd)\n", 2+5*count);
    printf("(qualify input%d gnd)\n", 3+5*count);
    printf("(qualify input%d gnd)\n", 4+5*count);
}

count=0;
for (count=count;count++)
{
    printf("(qualify output%d gnd)\n", 10 *count);
    printf("(qualify output%d gnd)\n", 1+10*count);
    printf("(qualify output%d gnd)\n", 2+10*count);
    printf("(qualify output%d gnd)\n", 3+10*count);
    printf("(qualify output%d gnd)\n", 4+10*count);
}
printf("(qualify output%d gnd)\n", 5+10*count);
printf("    (qualify output%d gnd), 6+10*count);
printf("(qualify output%d gnd), 7+10*count);
printf("(qualify output%d gnd)\n", 8+10*count);
printf("    (qualify output%d gnd)\n", 9+10*count);
}

printf("\n");
printf("(property sda type \"ground\")\n");
printf("\n");
printf("(criticalSignal 105 gnd)\n");
printf("\n");

printf("\n");
printf("\n");
printf("(property sda purpose \"cell\")\n");
printf("(property sda type \"standard\")\n");
printf("(property user NLPRepresentationList \"abstract schematic\")\n");
printf("(property user NLPStoppingPointList \"abstract\")\n");
printf("\n");
printf("\n");

}

*******************************************************************************/
/**
*/
/** ARGUMENTS    'L' represents newmod3 cell with left side pins   */
/**    'R' represents newmod3 cell with right side pins 
*/
/**    'N' represents newmod3 cell with no pins */
/**    'T' represents newmod3 cell mirrored about the Y */
/**    axis with left side pins */
/**    'r' represents newmod3 cell mirrored about the Y */
/**    axis with right side pins */
/**    'n' represents newmod3 cell mirrored about the Y */
/**    axis with no pins */
/**    'P' represents left path of newmod cell array */
/**    'Q' represents right path of newmod cell array */
/**    'X' represents space which is the height of newmod cell */
/**
*/
/**

*******************************************************************************/
```c
#include <stdio.h>
main(argc, argv)
int argc;
char *argv[];

/* This program creates a personality file used by the structure compiler */
/* for an array of newmod3 cells. The symbols used in the personality file */
/* is described below. */

/* 'L' represents newmod3 cell with left side pins */
/* 'R' represents newmod3 cell with right side pins */
/* 'N' represents newmod3 cell with no pins */
/* 'I' represents newmod3 cell mirrored about the Y axis with left side pins */
/* 'L' represents newmod3 cell mirrored about the Y axis with right side pins */
/* 'n' represents newmod3 cell mirrored about the Y axis with no pins */
/* 'P' represents left path of newmod cell array */
/* 'Q' represents right path of newmod cell array */
/* 'X' represents space which is the height of newmod cell */

{ int maxcols, maxrows, row, col, maxRrow, maxLrow, limit, count;
  char newmodarray[30][30][4], leftside[15][3][4], rightside[15][3][6];

  for(count=1; count<argc; count++, count++)
  {
    if( argv[count][0] == '-' && argv[count][1] == 'c')
    {
      maxcols= atoi(argv[count+1]);
    }
    else
    if( argv[count][0] == '-' && argv[count][1] == 'r')
    {
      maxrows= atoi(argv[count+1]);
    }
  }

  for(row=0; row<maxrows; row++)
  for(col=0; col<maxcols; col++)
  {
    if(row % 2) /* checks if row in newmod array is even or odd */
    {
      /* place odd row of newmod cells */
      if(col==0)
      {
        newmodarray[row][col][0]='l';
      }
      /* place even row of newmod cells */
      if(col==1)
      {
        newmodarray[row][col][1]='r';
      }
      /* place odd row of newmod cells */
      if(col==2)
      {
        newmodarray[row][col][2]='n';
      }
      /* place even row of newmod cells */
      if(col==3)
      {
        newmodarray[row][col][3]='P';
      }
    }
  }

```
else
    if(col==(maxcols-1))
    {
        newmodarray[row][col][0]='r';
    }
    else
    {
        newmodarray[row][col][0]='n';
    }
}
else
{
/* place even rows of newmod cells */
    if(col==0)
    {
        newmodarray[row][col][0]='L';
    }
    else
    if(col==(maxcols-1))
    {
        newmodarray[row][col][0]='R';
    }
    else
    {
        newmodarray[row][col][0]='N';
    }
}
}

if(maxrows % 2) /* checks if total # of rows of newmod cells is even or odd */
{
    maxLrow=(maxrows+1)/2;
    maxRrow=maxLrow;
}
else
{
    maxLrow=maxrows/2;
    maxRrow=(maxrows+2)/2;
}

limit=(maxrows+2)/2;

for(row=0; row<limit; row++)
{
    if(maxrows % 2)
    { /* place side path on odd # of rows */
        if(row==0)
        {
            leftside[row][0][0]='P';
            rightside[row][0][0]='X';
        }
        else
        if(row==(maxLrow-1))
```c
{
    leftside[row][0][0] = 'X';
    rightside[row][0][0] = 'Q';
} else
{
    leftside[row][0][0] = 'P';
    rightside[row][0][0] = 'Q';
}
else
    /* place side path on even # of rows */
if(row == 0)
{
    leftside[row][0][0] = 'P';
    rightside[row][0][0] = 'X';
} else
    if(row == (maxRow - 1))
    {
        rightside[row][0][0] = 'X';
        leftside[row][0][0] = '
    } else
    {
        leftside[row][0][0] = 'P';
        rightside[row][0][0] = 'Q';
    }
}
/* end for */

if(maxRows == 1)
{
    leftside[0][0][0] = 'X';
    rightside[0][0][0] = 'X';
}

    printf(".plane leftside masternewmod rightside \n");
for (row = 0; row < maxRows; row++)
{
    if(row < maxLRow)
    {
        printf("%s ", leftside[row][0]);
    } else
    {
        printf("W ");
    }
    for (col = 0; col < maxCols; col++)
    {
        printf("%s", newmodarray[row][col]);
    } /* end for statement */
    if(row < maxRow)
    {
```
    printf(" %s\n",rightside[row][0]);
} else {
    printf(" W\n");
} /* end for statement */

    printf("\n\n.termalias in<0>=Ain0; in<1>=Ain1; in<2>=Ain2; in<3>=Ain3; in<4>=Ain4;
    in<5>=Bin0; in<6>=Bin1; in<7>=Bin2; in<8>=Bin3; in<9>=Bin4; in<10>=PHI1;
    in<11>=PHI2;\n");

    printf(".termalias out<0>=Aout0; out<1>=Aout1; out<2>=Aout2; out<3>=Aout3;
    out<4>=Aout4; out<5>=Bout0; out<6>=Bout1; out<7>=Bout2; out<8>=Bout3; out<9>=Bout4;
    out<10>=PHO1; out<11>=PHO2; out<12>=gnd; out<13>=vdd;\n");
} /* end main */

#include <stdio.h>
#include <math.h>

main()
{
    int x1,x2,x3,x4,i;
    int x1bar,x2bar,x3bar,x4bar;

    printf("create_rom("premul1" 0 ");
    for(i=0; i<32; i++)
    {
        x1=12*i;
        x1bar=x1%29;
        printf(" %d", x1bar);
    }
    printf("\n");

    printf("create_rom("premul2" 0 ");
    for(i=0; i<32; i++)
    {
        x2=i;
        x2bar=x2%27;
        printf(" %d", x2bar);
    }
    printf("\n");

    printf("create_rom("premul3" 0 ");
    for(i=0; i<32; i++)
    {
        x3=23*i;
        x3bar=x3%31;
        printf(" %d", x3bar);
    }
    printf("\n");

    printf("create_rom("premul4" 0 ");
    for(i=0; i<32; i++)
#include <stdio.h>
#include <math.h>

/**************************************************************************/
/*
/* This program creates a skill program called main_sch_array_skill. This */
/* skill program is used to automatically create schematics */
/*
/* OPTIONS: -c coeffnum coeff0 coeff1 ...
/* coeffnum: The number of coefficients used in FIR Filter.
/* coeff0 coeff1 .. : The actual coefficients used.
/*
/* -m modulus mod0 mod1 ...
/* modulus: The number of modulo used in FIR Filter.
/* mod0 mod1 .. : The actual modulo used.
/*
/* purpose: Purpose of chip
/* chipname: Name of chip.
/*
/* Written by Martin Jaekel
/*
/**************************************************************************/

main(argc, argv)
int argc;
char *argv[];

[
    int coeff[100], modulo[5];
    void Dec_to_Mod();
    int temp,count1,count,coeffnum,modnum;
    char *purpose, *classname;

    /**************************************************************************/
    /* options to c program is declared here. */
    /**************************************************************************/

    if( argv[1][0] == '-' & & argv[1][1] == 'c' )
    {
        coeffnum=atoi(argv[2]);

        for(count=2,count1=0;count+1<3+coeffnum;count++,count1++)
            printf("%d", x4bar);
        printf("\n");
    }
{
    coeff[count1]= atoi(argv[count1+1]);
}
}

if(argv[3+coeffnum][0] == '\-' & & argv[3+coeffnum][1] == 'm')
{
    modnum=atoi(argv[4+coeffnum]);
}

for(count=4+coeffnum,count1=0; count+1< 5+coeffnum+modnum; count++,count1++)
{
    modulo[count1]= atoi(argv[count1+1]);
}
}

purpose=argv[argc-2];
chipname=argv[argc-1];

Dec_to_Mod(coeff,modulo,&modnum,&coeffnum,purpose,chipname); /* call routine to convert
decimal to mod */

} /* end main program */

void Dec_to_Mod(coeff,modulo,modnum,coeffnum,purpose,chipname)

/*******************************************************************************/
/*
/* The Dec_to_Mod routine simple converts decimal to modulo */
/* ?he decimal can be positive or negative. */
/*
/* Arguments of function is */
/* coeff: The actual coefficients used in FIR Filter. */
/* modulo: The actual modulo used in FIR Filter. */
/* modnum: The number of modulo used in FIR Filter. */
/* coeffnum: The number of coefficients used in FIR Filter. */
/*/ 
/*******************************************************************************/

int coeff[100],modulo[5];
int *modnum,*coeffnum;
char *purpose,*chipname;

{
    int j,row,modcount,coeffcount,count3,count4,count5,romcount;
    int tempcoeff;

    /*******************************************************************************/
    /* Print procedure and procedure name */
    /*******************************************************************************/
for(modcount=0; modcount < *modnum; modcount++)
{
    for(coefcount=0; coefcount < *coeffnum; coefcount++)
    {
        tempcoeff=coeff[coefcount];
        for(count5=0; count5<5; count5++)
        {

            printf("\nsch_rom_gen( \"%s\" \"%s.s\" \"mod%ld\" \"BIPSP%d\" ", purpose, chipname, 
modulo[modcount],coeffcount*5+count5);

            /**************************************************************************
            /* If decimal number is positive decrement it by mod. Basically */
            /* This part of the program finds modulo of positive numbers. */
            /**************************************************************************
            if(coeff[coefcount]>0)
            {
                for(i=0; tempcoeff>modulo[modcount]; i++)
                {
                    tempcoeff=tempcoeff-modulo[modcount];
                }
            }
            else

            /**************************************************************************
            /* This part of the program finds the modulo of negative numbers. */
            /**************************************************************************
            {
                for(i=0; tempcoeff<0;i++)
                {
                    tempcoeff=tempcoeff+modulo[modcount];
                }
            }

            /**************************************************************************
            /* Print out the rom contents. */
            /**************************************************************************
            for(count3=0; count3<32; count3++)
            {
                romcount=tempcoeff+count3;
                if(romcount>=2*modulo[modcount])
                {
                    romcount=romcount-2*modulo[modcount];
                }
                if(romcount==modulo[modcount])
                {
                    romcount=romcount-modulo[modcount];
                }
                printf("\%d ",romcount);
            }
            printf("\n");
tempcoeff=tempcoeff*2 % modulo[modcount];
}

/** Print out create_array function call. */

#include <stdio.h>
#include <math.h>

/* The following program creates a personality file used by the structure compiler */

main(argc, argv)
int argc;
char *argv[1];
{
    int romarray[10][5];
    int romloc, modulo, newmodrows, newmodcols, count, col, row;
    void Rom_Pat(), Dec_to_Bin(), OutputFun();
    char chararray[10][5][80];

    if( argv[1][0] == '-' & & argv[1][1] == 'n')
    {
        count=2;
        if( count < argc)
        {
            for(col=0; col<4; col++)
            {
                for(row=0; row<8; row++)
                {
                    romarray[row][col]=atoi(argv[count]);
                    count++;
                }
            }
        }
    }

    Rom_Pat(romarray); /* arrange rows and columns in correct order */
    Dec_to_Bin(romarray,chararray); /* call routine to convert decimal to binary */
    OutputFun(chararray); /* call routine to store ROM values in personality file */
void Rom_Pat(num)

int num[8][4];

{
    int col, row, temp, countf, countr;

    /* The rows and columns must now be changed to conform to the ROM */
    /* pattern of Peter Bird’s cell. */
    for(col=0; col<4; col++)
    {
        /* switch row 0 <-> 1 */
        temp=num[0][col];
        num[0][col]=num[1][col];
        num[1][col]=temp;

        /* switch row 4 <-> 5 */
        temp=num[4][col];
        num[4][col]=num[5][col];
        num[5][col]=temp;
    }

    for(row=0; row<8; row++)
    {
        /* switch column 0 <-> 1 */
        temp=num[row][0];
        num[row][0]=num[row][1];
        num[row][1]=temp;
    }
}

/* end rompat */

void Dec_to_Bin(num,charray)

/* This program converts non binary number into a 5 bit binary number */

int num[8][4];
char charray[10][4][80];

{
    int iteration,row,col,temp;

    for(col=0; col<4; col++)
for(row=0; row<8; row++)
{
    iteration=4;
    temp=num[row][col];

    while(iteration)
    {
        if ( temp>=(int)pow(2.0,(double)iteration))
        {
            chararray[row][col][4-iteration]=1;
            temp=temp - (int)pow(2.0,(double)(iteration));
            iteration--;
        } /* end if statement */
    } /* end else if statement */
} /* end while loop */

if(temp==0)
    chararray[row][col][4]= '0';
else
    chararray[row][col][4]= '1';

} /* end for statement*/

} /* end Dec_to_Bin */

void OutputFun(chararray)
{
    /* This function prints ROM pattern into personality file which will be used */
    /* by the structure compiler. */

    char chararray[10][4][80];

    int row, col;

    printf(".plane col1 col0 col2 col3\n");
    for (row=0; row<8; row++)
    {
        for (col=0; col<4; col++)
        {
            printf("%s ",chararray[row][col]);
        }
        printf("\n");
    }
}
} /* end OutputFun */
APPENDIX III

Mask Layouts
Title: **FINITE IMPULSE RESPONSE FILTER**

Size: (Design Scale microns) 7572×7967

Number of Bonding Pads: 64

Designer: Martin Jaekel, Research Assistant

Description: This chip contains a modulo Bit-Level Systolic array implemented with a single generic cell. The generic cell computes over finite rings and provide fixed coefficient. The architecture is composed of linear arrays of systolic, steerable dynamic ROM.
Scaling Chip
APPENDIX IV

Schematics for FIR Filter and Systolic cell
ROM Drivers

D3

D2

D1

D0

D4

D5

D6

D7

D8

D9

D10

D11

C3

C2

C1

C0

R0

R1

R2

R3

R4

R5

R6

R7
Latch

Clocks
Steer-bit

BIN0

PHI2BAR

PHI1BAR

PHI2BAR

PHI1BAR

I15

I16

I79

I77

BOUT4

INV

INV

INV

INV

CS0NEG

CS0PL

CS1NEG

CS1PL

A4B

A4

I19

I24

I20

I25

I21

I86

I87

I22

INV

INV

INV

INV

PREBAR

TGNEG

TGPL

PHI2

PHI1

PHI2

PHI1
2 to 4 Row Decoder
APPENDIX V

User Manual
USER MANUAL

The automated layout generator is the third component of the systolic compiler, which consists of the designers assistant, FCST and the layout generator. This Appendix describes the steps to be followed in order to run the layout generator and view the results of the run afterwards. This manual is divided into three sections. The first section describes how to start up the layout generator. The second section describes how to enter the required input information correctly in order to run it. The third describes how to hierarchically view the results of the run.

V.1 Starting up the Layout Generator

If WINDSILC is properly installed the layout generator can be started by following the steps outlined below.

i) In the open window environment on a unix station(sun4, SPARC1 or SPARC) enter the command wisc<CR>.

ii) A menu appears as shown below

1) Designer's assistant
2) Transformer
3) Layout Generator

Enter 3<CR>.
iii) Four Cadence desks appears on the screen. Choose the desk containing the WINDSILC fixed menu by clicking a mouse button inside that desk. The screen will appear as shown in fig. V.1.

![Figure V.1 Initial WINDSILC Screen.](image)

iv) Pop up the Design Manager menu by clicking the left button in the empty window.

v) Click on the WINDSILC icon. Another popup menu will appear as shown below in fig. V.2. This puts the user in the WINDSILC environment.
V.2 Running the Layout Generator

i) Place cursor over the WINDSILC icon on the WINDSILC popup menu and click the left mouse button. A fill form appears depicted in fig V.3.

Do you want to use Designer’s Assistant and Transformer?

Enter chipname=
FIR Filter type (lowpass, highpass or bandpass)=
Enter purpose of chip (encoder,FIR Filter or scaling)=
Enter chip number=
1’st cutoff frequency in hertz=
2’nd cutoff frequency in hertz=
sampling rate in hertz=
wordlength of input=
wordlength of output=
# of moduli=
# of coefficients=
cancel generation=

Figure V.3 WINDSILC Fill Form.
ii) The fill form shows the default answer to each question. In order to change any answer move the cursor over the question and click the middle mouse button. The default answer appears in the edit window. It can be erased and a new value entered.

iii) In order to use the coefficients and the moduli from the systolic compiler the user should answer yes to the following question "Do you want to use the designer’s assistant and transformer". If the user enters no to this question then the moduli, coefficients, chipname and other relevant information must be supplied. Otherwise the default values will be used. After all the information has been entered place the cursor outside the fill form and click any mouse button.

iv) The next step is to click on the Design information icon in the appropriate path. The form shown below appears on the screen.

```
Enter width(dx) of input pad (design microns)=
Enter width(dx) of input pad (design microns)=
Enter width(dx) of input pad (design microns)=
Enter height(dy) of input pad (design microns)=
Enter width(dx) of output pad (design microns)=
Enter height(dy) of output pad (design microns)=
Enter width(dx) of vdd pad (design microns)=
Enter height(dx) of vdd pad (design microns)=
Enter width(dx) of gnd pad (design microns)=
Enter height(dx) of gnd pad (design microns)=
Enter width(dx) of BIPSP pad (design microns)=
Enter height(dx) of BIPSP pad (design microns)=
Enter width(dx) of chip pad (design microns)=
Enter height(dx) of chip pad (design microns)=
Enter Maximum power consumption of chip (watts)=
Enter supply voltage (in volts)=
Enter technology (cmos3, cmos4)=
```

Figure V.4 Design Information Form.
The default values of the Design information fill form are the actual size and technology of the most recent revision of the systolic cell. Move cursor outside of fill form and click any mouse button.

v) If the user wishes to supply the coefficients and moduli then he should click on the load coefficient and moduli icon. The fill form shown below in Figure V.5 appears.

\begin{verbatim}
Enter modulus 1=
Enter modulus 2=
.
Enter modulus N=
Enter coeff1=
Enter coeff2=
.
Enter coeffN=
\end{verbatim}

Figure V.5 Moduli and Coefficients Fill Form

If the information is taken from the designer's assistant then this step may be omitted.

vi) The final step is to actually run the layout generator by clicking on the run compiler icon corresponding to the chosen path.

V.3 Viewing the Compiler Run Results

The WINDSILC fixed menu, shown in fig. V.6 can be used to hierarchically view the internal structure of each component of the designed system. There
are three main components, the encoder, the FIR filter and the decoder or scaling.

![Image of WINDSILC fixed menu]

Figure V.6 WINDSILC fixed menu.

V.3.1 Encoder and Scaling

Viewing the structure of the encoder and scaling blocks are carried out by following the same steps. Hence, they are discussed together in this section. There are four icons associated with each of these blocks on the fixed menu. These are the chip layout, block diagram, I/O information and chip information icons.

**Chip Layout**: Clicking this icon will automatically display the mask layout of the corresponding block. Examples of such layouts are shown in Appendix III.
**Block Diagram:** Clicking this icon displays the internal structure of the encoder or scaling block. This allows the user to see how the individual BIPSP cells are connected within each block. Fig. V.7 shows the internal structure of the scaling block for a particular example.

![Block Diagram of Scaling](image)

**Figure V.7 Block Diagram of Scaling.**

**I/O Information:** This icon displays the names of the I/O pads and their positions. This information will be useful for testing a fabricated chip. The I/O information for an encoder chip is shown below in fig. V.8.
Chip Information: Selecting this icon displays certain information about a particular chip. These include its name, size, number of inputs and outputs and number of Vdd and Gnd pads.

V.3.2 Viewing the FIR Filter Block
The FIR filter section of the WINDSILC menu contains only two icons, the block diagram and the chip layout.

Chip Layout: As in the previous section, this icon simply displays the mask layout of a particular chip. An example of a layout of a FIR filter is shown in Appendix III.
Block Diagram: Selecting this icon displays the structural representation of the FIR filter as shown in fig. V.9.

![Block Diagram of FIR Filter](image)

Figure V.9 Block Diagram of FIR Filter.

Clicking on the coefficient table icon in fig. V.9 will show the actual values of the coefficients used in the most recent run. An example of this is shown in fig. V.10.

<table>
<thead>
<tr>
<th>Tap #</th>
<th>Quantized Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>51</td>
<td>23</td>
</tr>
</tbody>
</table>

Figure V.10 Coefficient Table.
Clicking on the filter icon displays information about the frequency characteristics of the filter such as the cutoff frequency, etc. An example of this is shown in fig. V.11.

<table>
<thead>
<tr>
<th>Lowpass Filter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling Rate</td>
<td>2000Hz</td>
</tr>
<tr>
<td>Input Word Length</td>
<td>9 bits</td>
</tr>
<tr>
<td>Output Word Length</td>
<td>10 bits</td>
</tr>
<tr>
<td>Modulus Used</td>
<td>32</td>
</tr>
<tr>
<td>Number of Taps</td>
<td>51</td>
</tr>
<tr>
<td>Chip Number</td>
<td>1</td>
</tr>
<tr>
<td>Cutoff Frequency</td>
<td>500Hz</td>
</tr>
</tbody>
</table>

Figure V.11 Example of Filter Characteristics.

Selecting the modulo icon from fig. V.9 displays the schematic shown in fig. V.12 which illustrates the functionality of an array of five BIPSP cells.

Figure V.12 Functionality of a Linear Array of 5 BIPSP Cells.
VITA AUCTORIS

Martin Jaekel was born on May 15, 1964 in Niagara Falls, Canada. He completed his high school education at Harrow High School. He graduated from the University of Windsor in 1989 with a Bachelor of Applied Science in Electrical Engineering. In May 1991 he finished his Masters of Applied Science in Electrical Engineering at the University of Windsor.