APPLICATION OF MODEL DRIVEN ARCHITECTURE DESIGN METHODOLOGIES TO MIXED-SIGNAL SYSTEM DESIGN PROJECTS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

John Sheridan Fisher, B.S.E.E., B.S. Mathematics, M.S.E.E.

* * * * *

The Ohio State University

2006

Dissertation Committee:

Prof. Steven B. Bibyk, Adviser

Prof. M. Ismail

Prof. J. DeGroat

Approved by

Adviser

Graduate Program in

Electrical Engineering

ABSTRACT

Mixed-signal system design is a complex task with many levels of deliverables, including layout, schematics, simulation results, functional models, design specifications, and process design kits. The ability to create mixed-signal design content has lagged behind digital design for decades, largely because the digital methodology has been able to make efficient use of design abstraction and implementation automation. In this work, we argue that these digital design methods are a form of Model Driven Architecture (MDA), which could be applied to mixed-signal design at a number of levels to create the next generation of mixed-signal design flow. To illuminate this point, we have created a novel translation engine to generate deliverable content in an automated manner from an abstraction of the desired content.

We show two implementations of this MDA process. One is the creation of a process design kit for mixed-signal EDA toolsets. As the element that interfaces the EDA toolset with the fabrication process, the construction of the PDK fills a critical role in the design process. The time-saving enhancements added to the PDK improve design efficiency and decrease time-to-market (TTM). However, creating the proper PDK files to implement various enhancements is a time-consuming task. We propose a novel method that these improvements could developed once, then have portability to any fabrication process. This process provides a direct path for PDK construction

reuse between processes. In fact, the fabrication process information is also portable between EDA toolset within our proposed methodology.

The second implementation is the translation of design specifications into AHDL models, corresponding to existing schematics of appropriate topology. This schematic topology is automatically selected from the design specifications and is wired to the rest of the design hierarchy by the novel translation engine developing in this work. This second example fits into a larger mixed-signal design process where the schematic are optimized for performance and layout generated in an automated fashion using available EDA tools, such that, with virtual component (VC) libraries, the design can develop from specification to layout in an automated design flow. Certainly, the development of VC libraries is part of a larger goal for design reuse within mixed-signal design. However, the developing EDA tools also offer methods of abstract design intent within the tool framework to enable IP capture and reuse. We further that intent with our translation engine and offer further abilities to abstract design intent for automated reuse.

Dedicated to my family

VITA

November 24, 1974	Born - Columbus, OH
March of 1997	B.S. Mathematics, The Ohio State University, Columbus, OH.
June of 1997	B.S. Electrical Engineering, The Ohio State University, Columbus, OH.
August of 1999	M.S. Electrical Engineering, The Ohio State University, Columbus, OH.
September of 1997 to August of 2004	Graduate Research Assistant, Graduate Teaching Assistant, Micrys Industrial Fellow, Texas Instruments Industrial Fellow, The Ohio State University, Columbus, OH.
August of 2004 to present	Mixed-Signal Design Engineer, Kenet, Inc., Reading, MA.

PUBLICATIONS

Research Publications

Cerny, C.; Blumgold, R.; Cook, J.; Bibyk, S.; Fisher, J.; Siferd, R.; Si-Yu Ren, "Detailed Analysis of Enhancement-Mode Technologies for the Development of High Performance, Power Conserving, Mixed-Signal Integrated Circuits," *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference, 2000 (NAECON 2000)*, pages 532-540, October 2000.

John Sheridan Fisher and Steven B. Bibyk, "Design Methods For System-On-A-Chip Control CODECs To Enhance Performance And Reuse," *Proceedings of the 14th Annual IEEE International ASIC/SOC Conference*, 2001, pages 423-427, September 2001.

John Sheridan Fisher, Robert Henz, Deepika Devarajan, Jason Abele, and Steven B. Bibyk, "System-on-a-Chip Design Methodologies and Issues for Transducer-to-Pico-Network Applications," *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems (MWSCAS 2001)*, pages 930-934, Volume 2, October 2001.

John Sheridan Fisher, Jason Abele, Steven B. Bibyk, "Application of Model Driven Architecture Design Methodologies to Mixed-Signal System Design Projects." *Transactions on Computer Aided Design*, submitted for publication.

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in:

Studies in Mixed-Signal Design Methodologies: Prof. Steven Bibyk

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

When mixed-signal microchips involved only a handful of transistors, a single engineer could design the entire chip. Consequently, the designer could use any ad-hoc design methodology to complete the given project at hand. However, as mixed signal systems have gotten exponentially more complex, TTM constraints require that large teams of engineers divide the design into smaller and smaller sections. It is at this partition and at the re-integration of the large array of sections that there is a weakness in the mixed-signal system design flow. In addition, TTM constraints are so fierce that efficiencies introduced in any part of the design cycle are worth considering.

By and large, mixed-signal electronic design automation (EDA) tools have been successful at setting up a design methodology to develop topology driven circuits and have allowed designers to integrate these together without regard to their mutual interactions. However, in large system integration, topological design methods are less efficient than performance driven design methods; from a system's point-of-view, we do not care specifically whether the opamp in our switched-cap integrator is of a current mirror topology or a Miller-compensated topology, but rather that its gain

and bandwidth meet certain performance metrics.

This focus of functional specifications as the motivation to drive design goals represents an abstraction of the target implementation. Digital design tools have had such an abstraction provided by their behavioral hardware description languages (HDLs), which can be synthesized into structural netlists and then further implemented into finished layout. Most importantly, this whole implementation process in digital design is automated. While there is non-trivial setup and design choices to make about the design flow, there is less opportunity for the translation process to be broken my manual error.

This abstraction and implementation process has been missing from mixed-signal design, in part because of the complexity and interconnectedness of analog block behaviors and in part because of a lack of EDA tools to capture and handle analog behaviors. As such, we will apply a formal abstraction and implementation methodology to the mixed-signal design flow. We will propose the application of developing commercial EDA tools to achieve equivalent abstraction as in digital design. Then we will further apply a novel model driven architecture (MDA) process, including a novel translation engine, to further abstract behavioral descriptions to specifications, such that system design can be developed from design specifications with an automated path to schematics to layout.

Chapter 2 will introduce the concept of MDA and in what form we consider it. Chapter 3 will introduce the concept of a process design kit (PDK) for the EDA toolset

as a simple, but often overlooked example of deliverable in the design process. Chapter 4 will show how we have been able to generate a PDK for design in multiple EDA toolsets and foundries by using our MDA process. The necessary abstractions, models, sources, and targets will be also be developed in this chapter.

Having developed the idea of our MDA process through a simple implementation, we will begin to explore the mixed-signal design flow. Chapter 5 will discuss traditional design methodologies and available commercial EDA tools. Some of the issues with the traditional methods will be illuminated through three fabricated and tested mixed-signal design examples in Chapter 6. We will present our envisioned natural next step of evolution of mixed-signal design flows in Chapter 7. To that, we will add our proposed MDA process in Chapter 8. Then in Chapter 9, we will present a design example of our novel MDA process and its models and translation engine. Finally, we will discuss our contributions and areas of future work in Chapter 10.

CHAPTER 2

INTRODUCTION TO THE MODEL DRIVEN ARCHITECTURE

The Object Management Group (OMG) formally released its Model Driven Architecture (MDA) methodology standard in 2001 [2]. At its core, MDA argues for an automated translation from higher-level models to implementation [3]. These higher-level models are platform-independent models (PIMs) and model the application at a functional level. The automated translation creates platform specific models (PSMs), using architectural rules for implementation on any given platform. Figure 2.1 shows a block diagram of the relationship between models in the MDA methodology. One of the primary emphases of MDA is the partitioning of the design effort into functionality and implementation. This partition allows for compartmenting of the work into areas of design expertise. Experts in both the application design and implementation design can work concurrently, without dependence on or interference with the other group. Additionally, the system design is given a framework in which to capture important features, requirements, and specifications, which will be used to generate the final implementation [4]. That implementation can be re-targeted to a different platform by the same automated translation process, without changes to application model, and only dependant on the availability of architecture rules to generate the

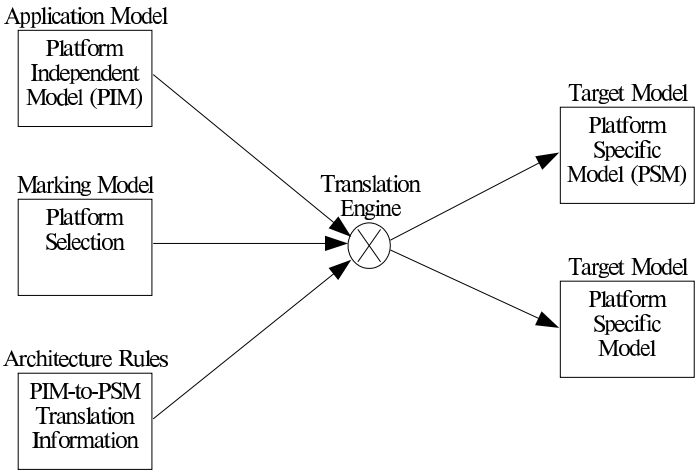


Figure 2.1: Generalized Model Driven Architecture Block Diagram.

appropriate platform-specific target models. Also, because of the automated translation process, the application model can be developed to later point in the design cycle. Specification refinements and changes do not cause wasted implementation cycles until the project due-date is within the time window for automated implementation.

While the primary motivation for this methodology has been software design, the OMG was careful to try to leave the process generalized such that it could be applied to any process where an abstraction of some implementation is possible. As such, we believe that MDA provides one path to the next generation of microchip design, and in particular, we will explore possible applications into mixed-signal design. Without explicitly calling for the use of MDA, Keutzer and others have indicated that hardware design is in dire need of the methodology that MDA brings to bear [5, 6, 7]. Keutzer

specifically indicates the need to model the application at its most abstracted to explore the design space; to make use of all possible forms of design reuse; to implement reliable and consistent implementation methods; and to partition the behavior and implementation as their intertwining complicates the design [5]. We argue that one path to addressing these observations is the use of the MDA methodology wherever possible to abstract any (non-trivial) deliverables to be generated in the mixed-signal design process.

To be clear, our implementation of the MDA process is based on these core philosophies outline. Specifically, we will not be using the OMG standardized Universal Modeling Language (UML) [8] to graphically capture the system specifications. Nor will we be using the OMG standardized Meta-Object Facility (MOF) [9] to create a model of the modeling language (meta-models). While these are fine standards, their implementation dilute the intent of this particular work. While the existing constructs in UML can handle a state-machine, the constructs required to handle signal modeling and data flow are not as developed. However, the core ideas of the MDA methodology are highly applicable in a number of mixed-signal design tasks.

Digital design has already made use of a number of MDA methods. Using a hardware description language (HDL) to describe the behavior of a digital system is an abstraction of that system. Because this form of digital design can be synthesized into an FPGA or various fabrication processes using an automated synthesis, place, and route engine, digital design is a form of MDA. The behavioral model is an application model that can be reused. Likewise the fabrication process information and

leafcell libraries are architecture rules and can also be reused. Separate groups can develop the behavioral models and the process libraries concurrently, allowing for the behavioral models to be developed until the time window for automated translation runs into the project deadline. Digital design may not use the MDA framework to discuss their methods, but digital design's fundamental methodology is an example of an MDA process.

We argue that the MDA process can be anywhere in the mixed-signal design process where non-trivial deliverables can be abstracted to their application level functionality. We will present two separate and distinct example implementations to develop the point. The first example is the generation of the process design kit (PDK) for the electronic design automation (EDA) toolset and is presented in Chapter 4. This work is necessary for any mixed-signal design and is a simple MDA example, in that its process is complete and encapsulated. In contrast, our second implementation is the abstraction of mixed-signal virtual components (VCs) into performance metrics to enable design space exploration and to capture analog design work, knowledge, and organization. This MDA process is developed within the larger framework of a full mixed-signal design flow, in which it plays an important role but is not all-encompassing. This work is presented in Chapter 8.

CHAPTER 3

INTRODUCTION TO A PROCESS DESIGN KIT

One implementation of the MDA methodology for mixed-signal design that we have developed is a translation engine to create a process design kit (PDK) customized for given a fabrication process and for an electronic design automation (EDA) toolset. In this chapter we discuss what elements compose a PDK and how they aid in the design process.

3.1 PDK Fundamentals

A PDK is a set of files that the EDA toolset uses as a framework for design in a particular fabrication process. Table 3.1 expresses some functionality that a basic PDK defines.The EDA toolset merely provides a general skeleton for design work, so the PDK is necessary to specify which features will be used in what manner. At its most basic, the PDK determines which design layers have what purposes and properties.

In our PDK generation example, we chose to focus on the Cadence® mixed-signal design flow and the Laytools® custom design flow. The Cadence® toolset is expansive and expensive, with the cost of a single seat of each available package in the toolset

Tool functionality	PDK component
Schematic design	primitive library, callback code
Schematic simulations	fabrication device models
Layout design	layer information, pcells, routing information
DRC physical verifcation	design rules
LVS physical verification	extraction rule, LVS rules
Layout extracted simulations	process electrical rules

Table 3.1: EDA Toolset Functionality Defined by PDK Components.

running in the millions of dollars. However, the Cadence® tool is designed to handled the largest design projects in the world, with hundreds of engineers working together. In contrast, Laytools® is meant for smaller teams of a few engineers each working on smaller design projects that do not stress the computational limits of traditional physical and functional verification, as well as not requiring some of the design efficiency enhancements that Cadence® offers. As such the cost of a complete suite of Laytools® is in the tens of thousands of dollars. Of course, the package of tools in the Laytools® suite is not as expansive as the Cadence® package, but sufficient to handle complete mixed-signal design work. The conventional wisdom is that the suite tools from Laytools®can be purchased for the annual maintenance cost of the similar set of tools from Cadence®.

3.2 Handling Design Layers

The defacto standard for exchanging layout design data is the GDS-II file, which contain a record for each module in the hierarchical design. Each module is composed of instances of other modules in the hierarchy, as well as paths and polygons of layout

design on the various layers used at that level of the hierarchy. These design layers are referenced only by number in the GDS-II file. Likewise, EDA tools internally reference each layer in layout design by number. However, each layer can also be given a name, which is far more convenient for designers to visualize the layers. So at a minimum, a PDK needs to indicate which layers are available for any given fabrication process, what the name of each layer is, and how that layer is to be displayed on screen. There also needs to be some mapping for translating these internal layer numbers to foundry specified ones.

The Laytools® method for handling methodology is simpler, although not as flexible as in Cadence®. Here is a a code snippet for one file in a Laytools® PDK:

```
layeralias 1 NWELL
    ## nwell implant
layeralias 3 PSD
    ## p-active/diffusion implant
layeralias 2 NSD
    ## n-active/diffusion implant
layeralias 10 POLY
    ## gate poly
layeralias 11 POLYF
    ## gate poly fill
```

The *layeralias* command maps an internal database layer number to an alias name for design work. The # tags begin comments for documentation. In contrast, a section of a Cadence® formatted technology file is provided:

```
layerDefinitions(
 ...

 techLayers(
 ;;( LayerName                  Layer#     Abbreviation )
  ( NWELL       1    NWEL )
  ( NSD         2    NSD )
  ( PSD         3    PSD )
  ( POLY        10    POLY )
  ( POLYF       11   PLYF )
  ...
 )
```

```
techLayerPurposePriorities(
;;( LayerName                 Purpose    )
 ( NWELL                     drawing    )
 ( NSD                       drawing    )
 ( PSD                       drawing    )
 ( POLY                      drawing    )
 ( POLYF                     drawing    )
 ( POLY                      pin        )
 ( NWELL                     net        )
 ( NSD                       net        )
 ( PSD                       net        )
 ( POLY                      net        )
 ( POLYF                     net        )
 ...
)

techDisplays(
;;( LayerName   Purpose      Packet         Vis Sel Con2ChgLy DrgEnbl Valid )
 ( NWELL        drawing      NWELL            t t t t t )
 ( NSD          drawing      NSD              t t t t t )
 ( PSD          drawing      PSD              t t t t t )
 ( POLY         drawing      POLY             t t t t t )
 ( POLYF        drawing      POLYF            t t t t t )
 ( POLY         pin          POLYpin          t t t t t )
 ( NWELLnet     drawing      NWELLnet         t t t t t )
 ( NSDnet       drawing      NSDnet           t t t t t )
 ( PSDnet       drawing      PSDnet           t t t t t )
 ( POLYnet      drawing      POLYnet          t t t t t )
 ( POLYFnet     drawing      POLYFnet         t t t t t )
 ( POLY2net     drawing      POLY2net         t t t t t )
 ( POLY2Fnet    drawing      POLY2Fnet        t t t t t )
 ...
)
) ;;layerDefinitions
```

In the Cadence® methodology, comments are started with the ; tag. Also, there is an entire section for defining layers, which is encapsulated inside the *layerDefinition(...)* tag. [This tag might also be written *(layerDefinition...)* as all of the Cadence® PDK files are based in the LISP scripting language, which Cadence® has expanded upon and named SKILL™.] The fundamental subsection is in the *techLayers(...)* tag. It is this subsection that provides the equivalent functionality, mapping layer name to internal layer number, as well as providing a layer name abbreviation (four characters or fewer) for layer palette. However, in the Cadence® layout design tool, each internal layer number can be used in multiple ways. Each number/name pair has a series of

bindings to indicate purpose. Each layer has a drawing purpose binding, which is used during layout design. In addition, each physical layer also has a net binging, which is used by the extraction tool to visually represent merged and connected nets. Additionally, a pin binding is provided for connecting or routing layers, such as metal layers. Each binding for each layer appears separately in the layer palette. This palette is heavily used during layout design to select active, visible, and selectable layers. As such, there are layers that may be available in the PDK, which are not visible or not selected by default. The *techDisplays(...)* tag controls these default behaviors, as well as naming a color and fill packet to be applied to each layer/purpose pair. The packet will be defined to indicate how each layer/purpose is to be displayed.

A display resource file snippet for Cadence® shown here:

```
;(drDefinePacket (DisplayName PacketName Stipple LineStyle Fill Outline) ...)
(drDefinePacket
 ( display  NWELL     dots1      dashed      gray       gray       )
 ( display  NSD       dots       dashed      lime       lime       )
 ( display  PSD       dots       dashed      tan        tan        )
 ( display  POLY      slash      solid       red        red        )
 ( display  POLYF     vZigZag    dashed      red        red        )
 ( display  POLYpin   solid      solid       red        red        )
 ( display  NWELLnet  blank      dashed      gray       gray       )
 ( display  NSDnet    blank      dashed      lime       lime       )
 ( display  PSDnet    blank      dashed      tan        tan        )
 ( display  POLYnet   blank      solid       red        red        )
 ( display  POLYFnet  blank      dashed      red        red        )

 ...
)
```

For each packet, a fill pattern, outline line-style, fill color, and outline color is specified by the *(display ...)* tag within the *(drDefinePacket ...)*. Packet fill, color, and line style combinations need not be unique and are often reused, as trying to create entirely unique patterns can create designs schemes that are difficult to use during

layout design, because the color scheme makes the display too busy. There is a trade-off between providing more information regarding which layers have been used in the design and the readability of the color scheme on the display. Generally, schemes are creates to enable visualizing active areas, polysilicon layers, and routing layers primarily. Implants and logical marking layers are secondary.

Similar methods are used in designing display schemes for the Laytools® layout design tool. Here is the syntax for their display definition:

```
layercolor  11 NWELL
layercolor  22 NSD
layercolor  30 PSD
layercolor   2 POLY
layercolor   2 POLYF
fill  39 NWELL
fill  40 NSD
fill  40 PSD
```

Again, the format is much simpler, simply assigning a color number to each layer name with the *layercolor* tag and a fill number to each layer name with the *fill* tag. The fill patterns and colors are defined separately, as is also done in the Cadence® display resource file. These commands provide the mapping from layer numbers to layer names and layer display schemes.

3.3 Physical Verification

Additionally, foundries impose design rules on what layer patterns and relations are legal for fabrication, so PDKs almost universally have a design rule check (DRC) file to validate the layout design against the foundry rules. This level of physical verification only ensures that the layout design is legal for fabrication, and does not

validate the functionality. In mixed-signal design, functional verification is done at the schematic level. These schematics provide a structural abstraction of the active and passive devices in the design within a hierarchical topology. Specifically, the layout design is dense on information that does not affect functionality to the first-order; there will be parasitic elements introduced by the layout design, which will often be monitored and minimized in specific instances with extreme care. Otherwise, these parasitic elements are lower-order effects.

Since the functionality is designed at the schematic level, and the physical design is at the layout level, there needs to be a physical verification that the schematic structure is identical to the layout structure. This validation is referred to as layout versus schematic (LVS) and requires an intermediate extraction step of both the layout and the schematic to generate structural netlists of devices.

For both the DRC and the LVS physical verification processes, the PDK files provide the rules that the need to be enforced by the EDA tool. For example, one DRC might be the required minimum spacing between two polygons of the same layer, say the first metal layer. The DRC file specifies that there is a rule of that type and what its minimum value is. The EDA tool's base engine understands how to interpret that rule and enforce it across the entire layout design being checked.

A snippet of a Laytools® DRC file is provided:

```
...

!!
!! routingOnly DRC commands
!!
```

```
$LAYOP
  !! poly gate spacing
  *drcPolyGateSp = poly WITH DISTANCE LT 0.45 /DRC /OUT
/LIST="POLY.4: min POLY spacing = 0.45um"
  *drcPolyN = poly WITH NOTCH LT 0.45 /DRC /OUT
/LIST="POLY.3: min POLY spacing = 0.45um"

  !! diff (source/drain) enclosure of poly (gate)
  *drcPolyDiffEnc = poly WITH ENCLOSURE LT 0.60 IN diff /DRC /OUT
/LIST="POLY.5: min diffusion (source/drain) enclosure of POLY
(gate) = 0.60um" /NOT_CUTTING

  !! gate extension
  *drcDiffPolyEnc = diff WITH ENCLOSURE LT 0.45 IN poly /DRC /OUT
/LIST="POLY.6: min POLY enclosure (gate extension) of
diffusion = 0.45um" /NOT_CUTTING

  !! poly to diff spacing
  *drcPolyDiffSp = poly WITH DISTANCE LT 0.20 TO diff /DRC /OUT
/LIST="POLY.7: min POLY spacing to diffusion = 0.20um" /NOT_CUTTING

  ...
END !! end of LAYOP section
```

In this snippet, we check the spacing (and the notch spacing of a single polygon to itself) of polysilicon polygons, the enclosure of the source and drain diffusion around polysilicon, the gate extension of polysilicon beyond diffusion, and the spacing of polysilicon to unrelated (not cutting) diffusion. Assumptions are made to ease the rule design. For example, it is assumed that any overlap of polysilicon and diffusion is a MOSFET gate. These assumptions may limit the possible designs to be considered but do not hinder work in standard CMOS design. It should also be noted that there are intermediate rules prior to the code snippet that define layers, such as *diff* appropriately.

A equivalent set of rules is provided for the Diva™ tool in Cadence®:

```
drcExtractRules(
  ...
  drc( gate (sep < 0.45) "POLY.4: min POLY gate spacing = 0.45um" )
  drc( gate (notch < 0.45) "POLY.4: min POLY gate spacing = 0.45um" )
  drc( diff poly (enc < 0.60) "POLY.5: min diffusion (source/drain)
enclosure of POLY (gate) = 0.60um"
```

```
)
  drc( poly diff (enc < 0.45) "POLY.6: min POLY enclosure (gate
extension) of diffusion = 0.45um" )
  drc( poly diff (sep < 0.20) "POLY.7: min POLY spacing to diffusion = 0.20um" )
  ...
)
```

While the syntax is different in Cadence®, the sense of the rule implementation is unchanged between toolsets.

In the LVS process, the rules files need to specify what combination of layers in what relation form which devices during extraction. The tools is able to use this information to check the entire layout design for all of the specified devices and form a netlist. The schematic netlist is more direct to generate, as the schematic contains discrete devices. However, there are still conventions regarding which device parameters are translated into the netlist for LVS, as opposed to which parameters are used by the netlist for simulation. Once the two netlists have been created, the tool requires further LVS rules to dictate how parallel and series connected devices can be combined. Additionally, the LVS tool needs to understand which parameters of matched devices need to be compared; the two netlists can be determined to have been wired identically but fail to match if the devices being compared have mismatched parameters.

A snippet of the Laytools® extraction step of LVS is provided:

```
...

!!
!! ***** Extract Rules *****
!!
$LAYNET
  !! nmos device
  DEVICE m_nmos /MODEL = &nmosModel
  REGION %g ngate
  REGION %sd nsd /NUMBER = 2
  REGION %b wirePsubs
```

```
  RELATION %g TOUCHING %sd
  RELATION %b ENCLOSING %g
  TERMINAL d %sd wireNdiff /NUMBER = 1
  TERMINAL g %g wirePoly
  TERMINAL s %sd wireNdiff /NUMBER = 1
  TERMINAL b %b wirePsubs

  ...
END !! end of LAYNET device extraction commands

!!
!! ***** Parameter Extraction *****
!!
$NETPAR
  !! nmos device
  DEVICE m_nmos
  PARAMETER W = (EDGE(g,sd) / 2) * 1u
  PARAMETER MIN$W = W * (1 - &wtol)
  PARAMETER MAX$W = W * (1 + &wtol)
  PARAMETER L = ( (CIRCUMFERENCE(g) / 2) * 1u ) - W
  PARAMETER MIN$L = L * (1 - &ltol)
  PARAMETER MAX$L = L * (1 + &ltol)
  PARAMETER M = 1
  PARAMETER MIN$M = M * (1 - &mtol)
  PARAMETER MAX$M = M * (1 + &mtol)

  ...
END !! end of NETPAR parameter extraction
```

Within the *$LAYNET* tag, various devices geometries can be declared for extraction. Each *DEVICE* tag begin a new set of rules of a device. Each *REGION* tag indicates which regions to consider for the device when the *RELATION* tags are true for the polygon relationships. Then the *TERMINAL* tag is used to assign the ports of the device being extracted. This set of rules is used over the entire design being verified, and each instance where the set of rules is valid is considered a device to be extracted. Combining the extracted devices with the routing layers (of metal) forms a complete netlist. The one addition is the set of parameters for each device. Within the *$NETPAR* tag, parameters and their ranges are derived and assigned to the instances. So for each instance of a device extracted, *$NETPAR* applies its rules to assign each instance its parameters. This example code extracts an NMOSFET by looking for a gate touching two source/drain regions enclosed within a p-substrate. Then the

length of the common edge of the gate and the source/drain regions divided by two becomes the transistor width. The circumference of the gate divided by two then less the width is the length. We must assign a default multiplier of one, which may be modified by later verification when parallel devices are merged.

The equivalent Diva™ Cadence® extraction snippet is shown:

```
drcExtractRules(
  ...

  ;;***** Extract Standard Gate NMOSFETs ***************
  extractDevice( ngate (wirePoly "G") (wireNdiff "S" "D") (pwellNet "B")
     "nmos4 auLvs PRIMITIVES" physical )

    ngateWidth = measureParameter( length (ngate coincident wirePoly) 0.5E-6 )
     saveParameter( ngateWidth "w" )

    ngateLength = measureParameter( length (ngate inside wirePoly) 0.5E-6 )
     saveParameter( ngateLength "l" )

    ngateSDArea = measureParasitic( area (wireNdiff) 1e-12 figure )
     attachParasitic( ngateSDArea ("as" "S") ("ad" "D") ngate )

    ngateSDPer = measureParasitic( perimeter (wireNdiff) 1e-6 figure )
     attachParasitic( ngateSDPer ("ps" "S") ("pd" "D") ngate )

    saveRecognition( ngate "POLY" )

  ...
)
```

In the Cadence® rules, pairs of conduction layers and device ports to be assigned that layer are given after a single device recognition region. So we logically derive a region, *ngate*, which only occurs where NMOSFET devices are, then specify on which conduction layers the ports would connect to the device. If the recognition region touches or overlaps each of the port region, then Cadence® extracts an instance of the device. Since we define gate as where polysilicon and diffusion are, the edge length that is coincident between the gate and the entire polysilicon strip is twice the device width. And Cadence® provides an *inside* function that measures the

edge length of one polygon inside of another. That length is twice the NMOSFET length. Cadence® also allows for the diffusion areas and perimeters to be measured for parasitic modeling. It should be noted that only the nominal parameter values are measured in Cadence®, as the matching tolerances will be applied at the LVS step of the physical verification. A sample of that code is given:

```
(lvsRules
  ...

    (permuteDevice parallel     "nmos4"         parallelMOS)
    (permuteDevice parallel     "pmos4"         parallelMOS)
     ...
    (permuteDevice parallel     "rpoly2HR"      parallelRES)
    (permuteDevice series       "rpoly2HR"      seriesRES)
    (permuteDevice parallel     "cpoly2poly"    parallelCAP)
    (permuteDevice series       "cpoly2poly"    seriesCAP)

    (compareDeviceProperty  "nmos4"     compareMOS)
    (compareDeviceProperty  "pmos4"     compareMOS)
     ...
    (compareDeviceProperty  "rpoly2HR"  compareRES)
    (compareDeviceProperty  "cpoly2poly" compareCAP)
) ;end lvsRules
```

The Diva™ Cadence® engine understands how to compare the schematic extracted netlist with the layout extracted netlist, however some work must be done to get them to match in terms of the number of devices and nets). Since the layout-extraction matches single devices, its netlist must have its parallel and series devices combined by the *permuteDevice* tag. The *parallelMOS, seriesRES, and parallelCAP* procedures iteratively combine the specified devices in the netlists to try to match them. But even once the netlists match logically–same ports of the same devices wired in the same manner–the *compareDeviceProperty* tag is used to compare the device parameters. It is in these procedures–*compareMOS, compareRES, and compareCAP*–that the property tolerances are applied.

The equivalent Laytools® code snippet is provided:

```
!!
!! ***** combine devices *****
!!
$COMBINE m_nmos m_nmos /NOPREFIX /P_EQUAL = (W,L) /P_MIN = (W,L) /P_ADD = M
$COMBINE m_pmos m_pmos /NOPREFIX /P_EQUAL = (W,L) /P_MIN = (W,L) /P_ADD = M
$COMBINE xcpoly xcpoly /NOPREFIX /LABEL_DEVICE /P_EQUAL =
(CPIP_AREA,CPIP_PERI) /P_MIN = (CPIP_AREA,CPIP_PERI) /P_ADD = M
$COMBINE xrpoly2HR xrpoly2HR /NOPARALLEL /NOPREFIX /LABEL_DEVICE
/P_EQUAL = R1K_WIDTH /S_MIN = R1K_WIDTH /S_ADD = R1K_LENGTH /TX = 1
/TY = 2
!!   now combine parallel devices of the same width AND length
$COMBINE xrpoly2HR xrpoly2HR /NOPREFIX /LABEL_DEVICE /P_EQUAL =
(R1K_LENGTH,R1K_WIDTH) /P_ADD = R1K_WIDTH /P_MIN = R1K_LENGTH

!!
!! ***** launch LVS *****
!!
$NMC /COORDINATES
```

As in the Cadence® rules, methods for combining the instances within the netlist are provided by the *$COMBINE*. Then, the *$NMC* tag launches the actual LVS tool, which will compare combined-instance netlists, as well as parameters when they have been attached to the extracted devices.

3.4 Enabling Layout Design Efficiency

Aside from providing physical verification rules, the PDK often provides rules that enable routing assistance in the layout design. These rules specify which layers are for routing between points in the layout and which layers provide a connection between two layers (i.e. via12 connects first level metal and second level metal). In addition, the tool needs to understand the minimum width allowed for the routing layers and connecting layers, as well as minimum spacing between two polygons in the same layer. A set of Cadence® rules for this purpose follow:

```
;;*********************************
```

```
;; DEVICES
;;*********************************
devices(

  symContactDevice(
 ;;( name viaLayer viaPurpose layer1 purpose1 layer2 purpose2
 ;;   w l (row column xPitch yPitch xBias yBias) encByLayer1
 encByLayer2 legalRegion )
  ( VIA12 V12 drawing M1 drawing M2 drawing
     0.45 0.45 (1 1 0.9 0.9 center center) 0.15 0.15 _NA_ )
  ( VIA23 V23 drawing M2 drawing M3 drawing
     0.45 0.45 (1 1 0.9 0.9 center center) 0.15 0.15 _NA_ )
  ...
  ) ;;symContactDevice

  ...
) ;;devices

;;*********************************
;; PHYSICAL RULES
;;*********************************
physicalRules(
 spacingRules(
 ;;( rule   layer1   [layer2]   value  )
  ( minWidth "M1" 0.45 )
  ( minWidth "V12" 0.45 )
  ( minWidth "M2" 0.50 )
  ( minWidth "V23" 0.45 )
  ( minWidth "M3" 0.60 )
  ...
 ) ;;spacingRules
 ...
) ;;physicalRules
```

The *devices(...)* tag contains the *symContactDevice(...)* tag, which define the vias between routing layers. Each contact/via is given a new for the appropriate menu. Then the layers and purposes are given. So for the *VIA23* via, layer *V23* is used at the via layer between layers *M2* and *M3*, with all layers using the *drawing* purpose. The width and length of a single via follow on second line of the tag. Then the number of rows and columns to be default when generating a via are given, along with the horizontal and vertical pitch, and default horizontal and vertical origin point of the via instance in the layout design tool. Finally, the required enclosure of the via layer by the routing layers is specified (and a generally unused specifier on where this contact is legal to use in the layout design). Of course, this complex tag only

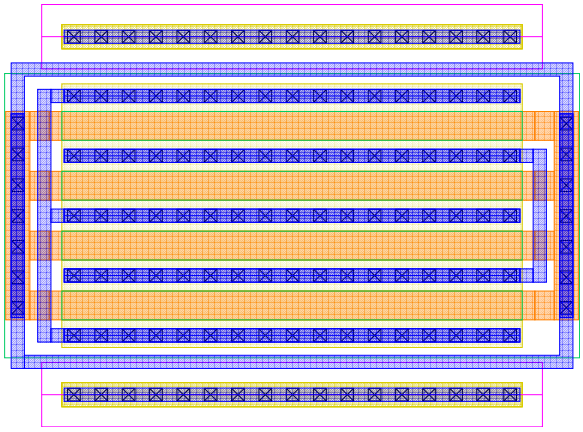


Figure 3.1: Complex Transistor Pcell Figure.

defines the via itself, such that the tool understands what element to place into the layout design when changing between layers. Within the *physicalRules(...)*, there is a *spacingRules(...)* tag to specify a number of DRC rules. One rule is the minimum width requirements on various layers, especially routing layers. This minimum number become the initial default width of layers used in the routing tool. Once all of the devices have been designed into a layout cell, the remainder of the design task is to connect all of the ports together. The routing tool provides an efficient method to generate all the necessary paths and vias to create the connections.

Adding parameterized cells (pcells) to layout assistance also decreases the time required for layout design. A parameterized cell is a layout module that uses the structural specifications for the exact device that is (or the set of devices that are) needed in the layout, and produces such an instance to be placed in the layout design. For example, drawing every layer for every transistor in the design would be painful because of the number of exacting design rules that need to be satisfied for

EDA Template	PDK Primitive
nmos4	nmos4
pmos4	pmos4
res	rpoly
res	rpoly2
res	rpoly2hr
res	rm1
res	rm2
res	rm3
cap	cpoly2poly
ind	indm1
ind	indm2
ind	indm3
presistor	presistor
pcapacitor	pcapacitor

Table 3.2: EDA Primitive Template to PDK Primitive Mapping.

each individual transistor. A pcell draws all of the layers automatically given the parameters–width and length, among others–such that the device instance satisfies all of the design rules of an individual transistor. Figure 3.1 shows how complex transistor pcells can easily become. Consider the number of rectangles necessary to generate the pcell shown. Outside the pcell, there are still issues with inter-device spacing rules and routing design rules, but there are generally far fewer of these rules, so the design load is much lessened and the designed devices are far more uniform (which aids in device matching).

In order to use the pcells, the PDK needs two additional elements. The first is a library of available structural devices that can be used in the schematic set. These primitives would include transistors and passive devices for mixed-signal design. While the EDA vendor generally will provide a base template for each type of transistor and each

passive device, the PDK must customize a copy of these templates for each device available in the fabrication process. For example, while the PDK might only provide one resistor template, the process could easily have ten or more resistors available when all of the diffusion, polysilicon, and metal layers have been considered. Table 3.4 shows how these EDA provided templates are mapped to PDK primitive cells. Because the passive device templates are abstracted from implementation method, they are the ones that get copied to multiple different PDK primitives. Of course, there are fabrication processes with multiple different MOSFET types, such as low or medium threshold devices or extended-drain devices. In such instances, the MOSFET templates would also map to multiple PDK primitives.

The second element required is the parameter set available on each unique structural device. These parameters include both functional parameters (i.e. width of a transistor or capacitance of a capacitor) and physical (layout-only) parameters (i.e. whether the transistor pcell should connect all of the gates together and in what layer and whether or not at both ends). A set of call-back functions that translate structural parameters into layout-only parameters (among other parameter setting automation) can also be provided in the PDK to give the designer useful feedback when generating the schematics. For example, a call-back function that translates the requested resistor width and length into resistance makes the designer more efficient.

A code snippet for a Cadence® formatted call-back procedure is given:

```
/*************************************************************/
/*  rpoly2hrResCallBack()                                    */
/*  sets: [w l r]                                            */
/*************************************************************/
procedure( rpoly2hrRCallBack()
```

```
  let( (width length resistance res delta_width grid)
    grid = 5e-08
    res = 1200
    delta_width = 0
    width=round(cdfParseFloatString(cdfgData->w->value)/grid)*grid
    length=round(cdfParseFloatString(cdfgData->l->value)/grid)*grid
    resistance = res*length/(width-delta_width)
    cdfgData->r->value = sprintf(s "%g" (resistance * 1))
    cdfgData->w->value = sprintf(s "%g" (width * 1))
    cdfgData->l->value = sprintf(s "%g" (length * 1))
  ) ; let
)
```

This procedure calculates and writes back the resistance of a passive *rpoly2hr* resistor cell based on a width and length, as well as the static variable of resistance per drawn square. Also, the width and length are rounded to the manufacturing grid specified by the foundry. All of these calculated values are written back to the instance of the *rpoly2hr* cell being edited. A snippet of that cell's properties for Cadence® are given:

```
/*****************************************************/
 LIBRARY = "PRIMITIVES"
 CELL    = "rpoly2hr"
/*****************************************************/

let( ( libId cellId cdfId )
    unless( cellId = ddGetObj( LIBRARY CELL )
        error( "Could not get cell %s." CELL )
    )
    when( cdfId = cdfGetBaseCellCDF( cellId )
        cdfDeleteCDF( cdfId )
    )
    cdfId  = cdfCreateBaseCellCDF( cellId )

    ...

    cdfCreateParam( cdfId
        ?name           "l"
        ?prompt         "Length"
        ?units          "lengthMetric"
        ?defValue       "2e-5"
        ?type           "string"
        ?display        "artParameterInToolDisplay('l)"
        ?callback       "rpoly2hrResCallBack()"
        ?parseAsNumber  "yes"
        ?parseAsCEL     "yes"
    )
    cdfCreateParam( cdfId
        ?name           "w"
        ?prompt         "Width"
        ?units          "lengthMetric"
```

```
        ?defValue       "2e-6"
        ?type           "string"
        ?display        "artParameterInToolDisplay('w)"
        ?callback       "rpoly2hrLengthCallBack()"
        ?parseAsNumber  "yes"
        ?parseAsCEL     "yes"
    )
)
```

After the preamble of the properties, there are a series of *cdfcreateParam* tags that define the specific properties of each parameter of a cell. A resistor has a length l, a width w, and a resistance r. Each can be assigned a common name (prompt), units, a default value, various entry types, an entry display method, and a call-back function. This call-back is executed each time the entry is modified. So in the snippet, whenever the length is modified, the call-back to recalculate resistance is executed. However, when the width is modified, it is assumed that the resistance needs to remain fixed and the length is modified.

The equivalent Laytools® methods is much simpler and less flexible:

```
EDITCELLATTRIBUTE RPOLY2HR l 20U
EDITCELLATTRIBUTE RPOLY2HR w 2U
EDITCELLATTRIBUTE RPOLY2HR r (1.2*#[l]/#[w])K
```

There is no external call-back function. The equivalent functionality is built into the primitive cell's properties. Also, there is a separation between static and dynamic variables. In this resistor example, the width and length are always static entries, and the resistance is always a dynamically calculated. And there is no understanding of what sort of entry each parameter is, nor a common name field to explain the meaning of the various parameter names. These extras are certainly not necessary and may only marginally improve design efficiency, however it is indicative of the sort

of differences that exist between the toolsets at the different levels.

As a final point on PDK features, we will note that there are many other customizations that can be made to a PDK to decrease design time. The set outlined here is family common and dramatically improves efficiency without becoming overly dependant on any one fabrication process' uniqueness or any one EDA toolset's specializations, at least in terms of what functionality is attainable.

3.5 The PDK File Set

In both example toolsets, as is generally common, the described PDK features can be defined in a set of text files. The exception is that the library of primitive devices is a set of binary files. As mentioned, these primitives can be copied directly from the vendor provided templates. Then, the default parameters of the copied primitive library must be set appropriately in order to properly interface with the physical verification, pcells, and simulation engine. This setup can either be done through the toolset's graphical user interface (GUI) or by executing and compiling a customized text file of PDK code into the primitive library.

Other text files must also be compiled into the PDK libraries, such as the pcell code. In the Cadence® toolset, the technology file, containing the layer definition and routing automation, must be compiled into the technology library of the PDK. In contrast, the corresponding file in Laytools® is sourced and parsed each time the toolset is launched. In both toolsets, the layer color definition file is likewise loaded

at start-up. However, the physical verification rules are parsed and executed for each DRC run, such that the rules can be improved or augmented between runs with the toolset still running.

The structure of these files is straight-forward but often cumbersome and lengthy to create. As shown in the various code snippets, the commands within these files are generally self-documenting. Additionally, differences in layers and base design rules are usually in name only, as all bulk CMOS fabrication process are based on the same physics and general fabrication methods. So, the difficulty in creating these PDK files is primarily in ensuring that the foundry given name for each layer, as well as each design rule value, is applied in every appropriate place. Manually creating a new file set or re-targeting an existing file set to a new fabrication process is a tedious and time consuming process with no automated verification to ensure that every entry has been created or updated correctly.

The exceptions to this rote file set are the pcells. The pcells are intended to ease layout by encapsulating the complex design rules as well as by allowing the construction of the pcell instance with any legal set of parameters. Consequently, their construction is as complex as the set of design rules and device parameters. In a number of fabrication processes, the number of layers involved in the generation of a single transistor pcell is more than half those available in the process. So developing pcells can be a complicated coding task. Unfortunately, the environment for developing the pcells is not ideal. In order to evaluate that a piece of code is producing the desired result, a layout cell must be compiled from code, updated in the library, and

instantiated in (or refreshed within) another test layout. Additionally, actual syntax error reporting for pcells in Cadence® or Laytools® can often be cryptic at best.

Consider the following code snippet from a Cadence® pcell, which generates only the contacts in the source drain regions:

```
;; CON
conLeft = diffLeft + conDiffSurr
conLower = diffLower +conDiffSurr
conRight = conLeft + conSize
conUpper = conLower + conSize
conInitYShift = floor( (wUm - (numConPerSD * conPitch) - (2 * conDiffSurr)
                        + conSpacing) / (2 * mfgGrid) ) * mfgGrid
conXShift = 0
mDrawn = 0
while( mDrawn < (m + 1)
  conYShift = conInitYShift
  conDrawn = 0
  while( conDrawn < numConPerSD
    dbCreateRect( cv "CON"
       list( (conLeft + conXShift):(conLower + conYShift)
             (conRight + conXShift):(conUpper + conYShift) ) )
    conYShift = conYShift + conPitch
    conDrawn = conDrawn + 1
  ) ;;end while conDrawn
  conXShift = conXShift + SDInnerL + (s * lUm) + ( (s - 1) * polySpacing )
  mDrawn = mDrawn + 1
) ;;end while mDrawn
```

Consideration must be given to the offset of the contacts, such that they are centered both horizontally and vertically in the diffusion. Just the initial offset become dependant on how many contacts can legally (by DRC rules) fit into the width of the diffusion, the manufacturing grid, the required amount of enclosure of diffusion around a contact, and the contact spacing and pitch. Then the distance between and size of diffusion regions must be properly considered given the length of each interior diffusion region, the number of polysilicon stripes between diffusion regions, the length of each stripe, and the spacing between stripes. These number must be set into a set of nested control loops to generate both column of rows of contacts,

incrementing and reseting offsets and shifts appropriately. The functional errors introduced during development are actually easier to debug (given a sufficiently rich set of test cases to exercise all unique pcell functionality) because the error is visually apparent. Syntactical errors usually manifest themselves by specifying the name of the nearest control statement to the actual error (although not which instance, nor on what line of the procedure). Debugging syntax errors can be a lengthy sleuthing process.

Equivalent code from Laytools® is no easier to manage:

```
##
## cnt
##
'cntYShiftInit = int( (wUm - (numCntPerSD * cntPitch)
 - (2 * cntDiffEnc) + cntSp) / (2 * mfgGrid) ) * m
fgGrid'
'cntLeft = diffLeft + cntDiffEnc'
'cntLower = diffLower + cntDiffEnc'
'cntRight = cntLeft + cntW'
'cntUpper = cntLower + cntW'
'cntXShift = 0.0'
'mDrawn = 0'

'mDrawM:'

    'cntDrawn = 0'
    'cntYShift = cntYShiftInit'

    'cntDraw:'

        SETLAYER 'cnt'
        RECTANGLE 'cntLeft + cntXShift' 'cntLower + cntYShift'
'cntRight + cntXShift' 'cntUpper + cntYShift'
        'cntYShift = cntYShift + cntPitch'
        'cntDrawn = cntDrawn + 1'

        'goif (cntDrawn < numCntPerSD, "cntDraw")'

    'cntXShift = cntXShift + SDPitch'
    'mDrawn = mDrawn + 1'

    'goif (mDrawn < (m + 1), "mDrawM")'
```

While syntactical errors are a bit better noted, the method for entering the code into the system is by a GUI only, into which entries must be typed or copied. Additionally, the Laytools® environment offers fewer control statements to manage the complex geometries. Essentially, only *if* and *branch* statements are available, further complicating the pcell design effort. These shortcoming tends to limit the number of test cases and design complexity for any given pcell.

CHAPTER 4

APPLYING AN MDA PROCESS TO PDK DEVELOPMENT

In this chapter, we will discuss how we have applied an MDA process to the problem of PDK development.

4.1 Introducing an MDA Process for Generating a PDK

To address these issues of developing a PDK quickly and accurately, we have developed a novel translation engine that allows for a PDK to be created from the fabrication process rules/data and a library of PDK templates. The fabrication process information is the application model of the PDK, as it represents the PDK abstracted from any toolset requirements. The marking model for this MDA process simply states for which EDA toolset the PDK is to be generated. The architecture rules are composed of two parts–PDK file structure rules and source files. The rules indicate which PDK files are to be generated from which source files. The source modules are PDK file templates for the files to be created in the MDA process. Specifically, a template file exists for each PDK file in each EDA toolset available, and each has been tagged such that the translation engine can place the necessary

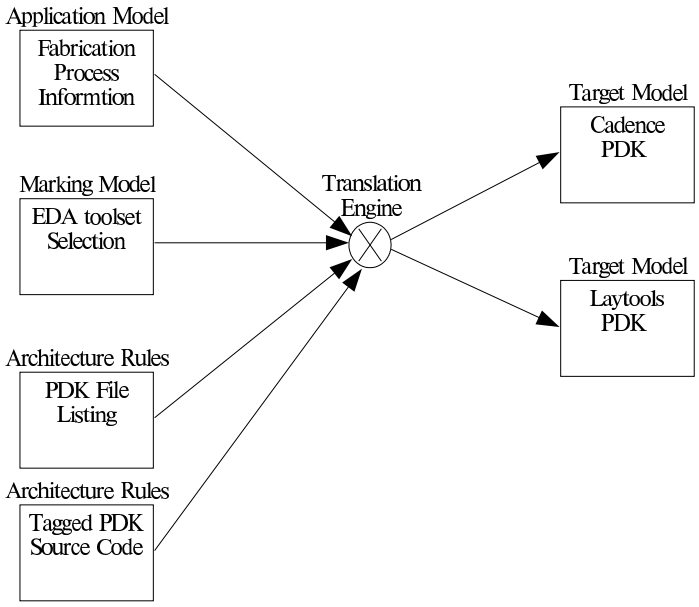


Figure 4.1: Model Driven Architecture Block Diagram for PDK Development.

fabrication process information at all of the required points within the templates to form the customized PDK files. As the ultimate goal of this MDA process is the creation of a PDK, these PDK files are the target models. In our example, the target models can be generated in either Cadence® format or Laytools® format. Figure 4.1 shows a block diagram of relationship of the models involved in this MDA process.

Once again, the pcells are the one exception. In our MDA process, the pcell source modules are actually developed in MATLAB®. Because of the aforementioned difficulties in developing pcells in an EDA toolset, it is more efficient to develop the algorithms first in another language. Any scripting language that has accessible graphing capabilities, mathematical manipulation functions, and specific debugging messages would be a reasonable choice. We chose MATLAB® for this purpose because it both

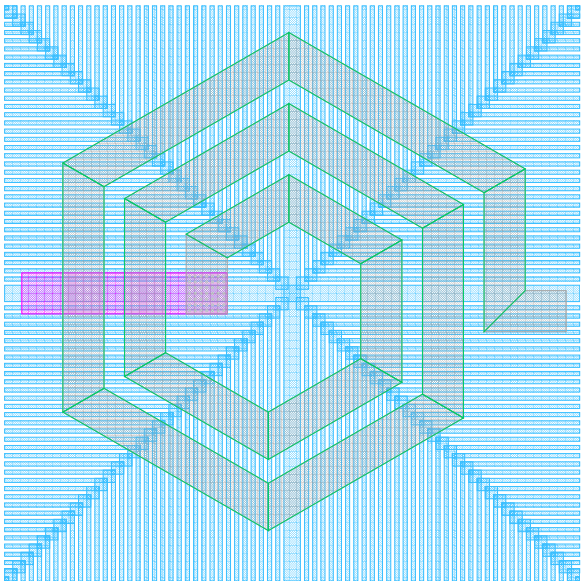


Figure 4.2: Six-Sided Spiral Inductor with Ground Shield Pcell Figure.

met the criterion and is ubiquitous within the engineering community. Additionally, by using a separate scripting language, the pcell algorithm is not bound to any one EDA toolset. The MDA process can be used to translate the pcell to any EDA toolset that we might be using. This design reuse helps to reduce the PDK development time.

Figure 4.2 shows a pcell that was developed in MATLAB® initially before being translated into an EDA toolset language. Both the metal inductor and ground shield entail a degree of geometric complexity that would make coding only in the EDA toolset language arduous. The shield forms a single piece of metal, made of horizontal and vertical fingers connected to a single X-shaped backbone. In fact the backbone is made of small overlapping rectangles to keep all points on the manufacturers design grid and to avoid acute angles in the design. Using the fingered shield (instead of a

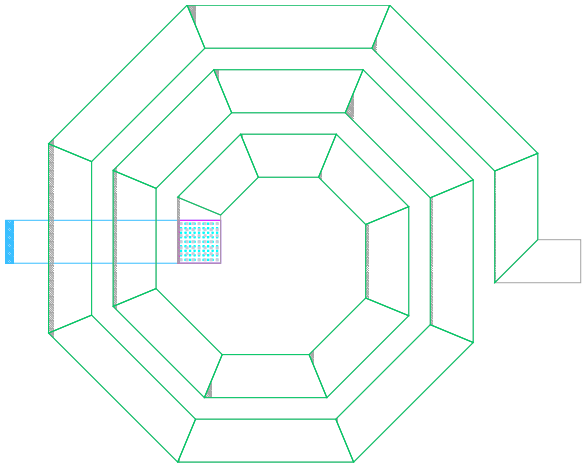


Figure 4.3: Eight-Sided Spiral Inductor Pcell Figure.

flat plate or a weave) prevents eddy currents and provide a shield for the inductor against the substrate. The shield width and length, as well as the finger width and gap, are configurable parameters of the pcell.

The inductor pcell is designed to generate any s-sided spiral inductor. Figure 4.3 shows an eight-sided example of just the spiral inductor. In the figure, the first metal layer is used as a cross-under out of the center of the inductor. The main segments are marked with a inductor recognition layer for physical verification purposes. The inductor width (w), the gap (g), the inner diameter (d), the number of turns (n), and the number of sides (s) are parameters of the pcell. The spiral is not a proper spiral because pain has been taken to keep the edge-to-edge gap constant throughout the inductor. Common spiral inductor pcell examplesmap the center line of an s-sided inductor path onto an ideal spiral at the inductor corners. One consequence is that the gap between turns varies, which degrades the designed impedance of the inductor.

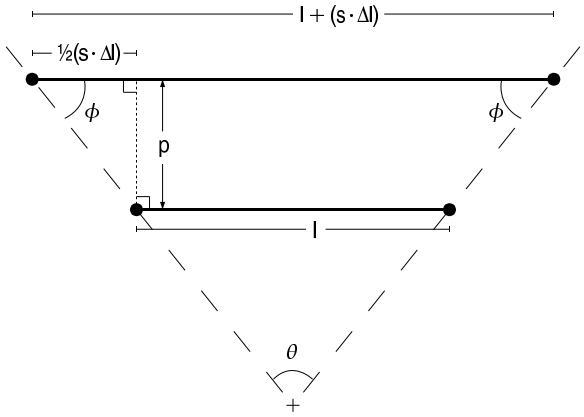


Figure 4.4: Geometry of Two Sections on One Side of an s-sided Metal Inductor with Constant Gap Spacing.

Additionally, the inductor section are then at odd angles, such that, even for a four-sided or eight-sided inductor, the sections are not on Manhattan-geometry (multiples of 45-degrees). For these reasons, spiral inductor are often painstakingly hand-drawn to avoid these issues.

We have created a novel pcell that implements the generalized geometry for the constant-gap, s-sided spiral inductor. The geometry involved for two sections from one side of the s-sided inductor are shown in Figure 4.4. For this figure, the pitch (p) is the sum of the inductor path width (w) and the gap (g).

$$p = w + g \tag{4.1}$$

and

$$\theta = \frac{360^o}{s} \tag{4.2}$$

so

$$2\phi + \theta = 360^o \tag{4.3}$$

thus

$$\phi = 90^o - \frac{180^o}{s} \tag{4.4}$$

Now, instead of mapping corners to an ideal spiral, we want to determine the increase in length of each segment that will get the desired geometry. Specifically, each subsequent segment will be a fixed amount longer than the previous one. And the angle between each subsequent segment and each previous one will be θ. We have θ, so we need to find the change in length between each successive segments. Having the change in length be a constant help to keep our spiral inductor as close as possible to an ideal spiral. As such, the change in length between two concentric segments on one side will be s-times the change in length of any two successive segments. Thus,

$$\frac{s \cdot \Delta l}{2} = \frac{p}{\tan \phi} \tag{4.5}$$

and finally

$$\Delta l = \frac{2p}{s \tan(90^o - \frac{180^o}{s})} \tag{4.6}$$

Given this complex geometry, the ability to prototype the design in MATLAB® improves design efficiency, and also decouples the algorithm from any specific EDA toolset.

Since the pcells also have dependencies on fabrication information, the architecture rules also contain a set of tagged MATLAB®-based pcells files. The translation of these MATLAB® models into the Laytools® pcell language or into the Cadence® pcell language involves an intermittent step of creating a fabrication process customized MATLAB® file of the pcell. This customized MATLAB® model is then directly translated into the target model, a pcell file for the selected EDA toolset.

It is important to understand that these tagged MATLAB® source modules are just another part of the architecture rules for the MDA process. The only difference is that the pcell code is in MATLAB®, whereas the the rest of the architecture rules' target source is in the EDA toolset language(s).

4.2 MDA Advantages and Limitations for PDK Development

The MDA process has the advantage that the PDK for an EDA toolset must be fully developed only once. That development is to generate the tagged set of files for that particular toolset. Thereafter, only the fabrication process information has to be aggregated into the application model format in order to generate a PDK for a new process. Since each element of this information is entered into the application model exactly once, the repetition required to create a PDK manually is eliminated. Additionally, once an application model has been created for a fabrication process, the PDK can be generated for that process for any of the EDA toolsets for which the architecture rules have been written. For example, once the AMI C5N process application model has been created, the same application model can be used to create

PDKs for Cadence®, Mentor Graphics®, and Laytools®, assuming that the architecture rules exist for each of these EDA toolsets.

Further, as new PDK features are added to any one EDA toolset, only the files in the architecture rules for that toolset must be modified. In contrast, without this MDA process, such new features added to a toolset would have to be repeatedly added to the PDK that is being maintained for *each* process. In the MDA process, the upgrade is added in one place, and thereafter a refreshed PDK can be generated for each fabrication process. Likewise if an error is discovered in the PDK template or if the syntax for some set of commands changes, it can be fixed in one place and each PDK refreshed.

Similarly, if a design rule value changes, the application model can be modified for that one element, and the PDK refreshed. Depending on the design rule that might change, it is possible that the fix will impact the majority of the files and in a number of places in some of these files. Additionally, unless the designer who creates the PDK is dedicated to PDK development, re-learning everywhere that a changing design rule is used in the PDK during the later stages of a project would be particularly time-consuming.

With the MDA process, each application model for a fabrication process are reusable for any EDA toolset. Additionally, the architecture rules for an EDA toolset are also entirely reusable and expandable for any fabrication process.

It is important to acknowledge that each fabrication process has slightly different design rules and that the architecture rules would have to be modified to fully implement any specialized deviations. Such modifications would only appear in the appropriate PDKs. Alternatively, the MDA process can be used to generate a base design kit, to which such specializations could be added manually. The time to add such specializations to the architecture rules is about twice as high as to simply add them manually to an MDA-created base PDK.

However depending on the process specializations, it may be appropriate to leave them out of the PDK. For example, a process may offer six metal layers, and the design may only require three. As long as the top metal layer in the process is mapped to the third metal layer in the PDK, the other upper three metal layers, their vias, and their design rules could be safely ignored in any dedicated fabrication run. Or if certain intricate design rules for transistors can be embedded by construction into the transistor pcell, then these design rules can be left out of the physical verification deck. Certainly, these unusual scenarios need to be evaluated on a case-by-case basis for risk assumed versus schedule impact to develop full verification methods, whether using the MDA process or not to build the PDK.

To put a fine point on the issue, the MDA process for generating PDKs is intended to save design time on the project. The PDK is absolutely necessary to complete the project, however there is little creative or novel work involved in creating a PDK. Some pcell code may be particularly clever and merits reuse, which is one reason that its source code is written in a scripting language in our process. The intention is to

develop a fully functional PDK for a fabrication process with as little designer effort as possible, such that the design team can focus its effort on the creative design work and decisions of the project.

The importance of this automation is starting to be understood within the EDA community. Efforts to develop a standard for a common design database system, known as Open Access, are starting to reach maturation, and such databases are now being supported by major EDA tool vendors, such as Cadence® and Mentor Graphics®. Additional efforts to develop a standard for a common PDK framework, known as Open Kit, are also underway [10]. With only one PDK to generate, foundries would be more apt to produce an Open Kit PDK, instead of developing, maintaining, and supporting sufficiently complicated documentation to allow everyone to build a PDK for their chosen EDA toolset.

4.3 Developing MDA models for PDK Development

To develop the MDA process to generate a PDK, a series of steps had to be taken. Firstly, the implementation and the specification of the PDK has to be separated, such that specification could be abstracted. The abstraction becomes the application model. So for a PDK, the implementation is the language and file structure for the EDA toolset. But without the fabrication process information, that framework is as empty as the EDA toolset is without the PDK. Thus, the fabrication process information is the application model. We chose to code our translation engine in PERL,

because of the language's robust ability to parse and manipulate text files. Consequently, we chose to make the format of the fabrication model information a single associative array, which contains the fabrication process information. Likewise, the marking model, which determines the format of the target models, is another associative array containing the EDA toolset for which the PDK is being generated.

The application model references the fabrication process information to be read, as well as naming the project, naming the application, and defining the application function:

```
## name of this project (perhaps like an EDA tool library name)
$project{'name'} = 'ami06cdspdk'; ## no spaces

{ ## Cadence PDK application model
    ## name of this application
    my $application = 'cadencePdk';  ## no spaces
    push @applications,$application;
    $application->{'name'} = $application;
    ## function of this application
    $application->{'function'} = 'pdk';

    ## fabrication process
    $project{'fabProcess'} = 'ami06typical'; ## required
}
```

In this example, we create a project called *ami06cdspdk* (which will be the name of the new output directory where PDK files will be generated) and are modeling an application named *cadencePdk*. The function of the application model is PDK generation for the AMI C5N process, with typical electrical parameters (such as sheet resistances and threshold voltages). The fabrication process information is also part of the application model, and a sample code snippet for its DRC rules and primitive device information is given:

```
    $fab{'drcM1WName'} = 'M1.1';
    $fab{'drcM1W'} = '0.45';
    $fab{'drcM1SpName'} = 'M1.2.';
```

```
    $fab{'drcM1Sp'} = '0.45';

    ...

    $fab{'devicePmos4Name'} = 'pmos4';
    $fab{'devicePmos4Model'} = 'pch';

    $fab{'deviceRnwellName'} = 'rnwell';
    $fab{'deviceRnwellModel'} = 'rnwell';
    $fab{'deviceRnwellRsh'} = '120'; ## ohms/sq
    $fab{'deviceRnwellJ'} = '20'; ## current density, uA/um
```

Here the associative array for fabrication process information is assigned the width and spacing rules for the first metal layer; the device name and model name for the PMOSFET; and the device name, model name, sheet resistance, and current density for an n-well resistor.

The full marking model is given and simply defines the EDA toolset for which the PDK is being generated:

```
{ ## PDK marking model
    my $application = 'cadencePdk';  ## no spaces
    ## target model platform
    $application->{'target'} = 'cadence'; ## required
}
```

While we named the application *cadencePdk*, we have not committed to the output being for the Cadence® toolset. This choice is explicitly declared in the marking model, as this model declares which form of implementation is to be created. The names of the application and project have were chosen to reflect intention of the marking model, but do not have any causality. In fact, we could have named the project *ami06pdk* and created two application models within it–one for Cadence® and one for Laytools®–to generate both PDKs at once. The only practical problem (beyond why someone might want to do so) is that the output files would be co-mingled in

the same project directory.

The last set of models necessary for our MDA process are the architecture rules. These are customized per implementation, so we have models for Cadence® and Laytools® in our example. A section of the rules for Laytools® is given:

```
  ## declare a hash for each module and its 'moduleName' entry
  foreach $module (@{$application->{'moduleNames'}}){
    $application->{'modules'}->{$module}->{'project'} = $project{'name'};
    $application->{'modules'}->{$module}->{'application'} = $application;
    $application->{'modules'}->{$module}->{'module'} = $module;
  }

   ## Laytools color allocation file
   $application->{'modules'}->{'display'}->{'targetSrc'}
= 'pdkLaytools/display.ini';

   ## Laytools layer initialization file
   $application->{'modules'}->{'layers'}->{'targetSrc'}
= 'pdkLaytools/layers.ini';

  ## Laytools process configuration file
   $application->{'modules'}->{'configs'}->{'targetSrc'}
= 'pdkLaytools/configs.ini';

   ## Laytools LayVer dB plotting rules file
   $application->{'modules'}->{'lvDBPLOT'}->{'targetSrc'}
= 'pdkLaytools/decks/lvDBPLOT.ver';

   ## Laytools LayVer verification DRC file
   $application->{'modules'}->{'lvDRC'}->{'targetSrc'}
= 'pdkLaytools/decks/lvDRC.ver';

   ## Laytools LayVer verification LVS file
   $application->{'modules'}->{'lvLVS'}->{'targetSrc'}
= 'pdkLaytools/decks/lvLVS.ver';
```

The equivalent section of architecture rules code for Cadence® is also given:

```
  ## declare a hash for each module and its 'moduleName' entry
  foreach $module (@{$application->{'moduleNames'}}){
    $application->{'modules'}->{$module}->{'project'} = $project{'name'};
    $application->{'modules'}->{$module}->{'application'} = $application;
    $application->{'modules'}->{$module}->{'module'} = $module;
  }

  ## Cadence display resource file
  $application->{'modules'}->{'display'}->{'targetSrc'}
= 'pdkCadence/TECHLIB/display.drf';
```

```
  ## Cadence technology library SKILL code file
  $application->{'modules'}->{'techfile'}->{'targetSrc'}
= 'pdkCadence/TECHLIB/techfile.rul';

  ## Cadence DIVA verification DRC file
  $application->{'modules'}->{'divaDRC'}->{'targetSrc'}
= 'pdkCadence/TECHLIB/divaDRC.rul';

  ## Cadence DIVA verification Extraction file
  $application->{'modules'}->{'divaEXT'}->{'targetSrc'}
= 'pdkCadence/TECHLIB/divaEXT.rul';

  ## Cadence DIVA verification LVS file
  $application->{'modules'}->{'divaLVS'}->{'targetSrc'}
= 'pdkCadence/TECHLIB/divaLVS.rul';
```

These portions of the architecture rules specify which source modules are to be translated into which target models and are another PERL associative array. The only substantiative difference between the two pieces of code is that the file structures are different between the two PDKs. Each has a display control file, DRC file, and LVS file. Laytools® has a layer definition file, and Cadence® combines that function with a number of other process declaration functions into a single technology file, as well as requiring a separate extraction file for Diva™ physical verification.

The other section of the architecture rules are the tagged source modules. These files are coded in the EDA toolset language, except for the tags that are to be parsed and manipulated by the translation engine. In fact, the source modules were developed from a set of files for a working PDK, which were abstracted of fabrication process information. In each file, special tags were substituted for the fabrication process information.

So the translation engine reads and stores the application model, then uses marking model to select which listing of source modules from the architecture rules need to

be used. Specifically, the architecture rules contain file listings for both a Laytools® PDK and a Cadence® PDK, as shown above. With a list of files to be parsed and fabrication information to substitute, the translation can be completed, except for the pcells. For the pcells, the intermittent MATLAB® models are created first to customize them for the fabrication process, then each one is directly translated from MATLAB® code into EDA toolset language code.

In Section 3.2, we presented a code snippet for each PDK defining the layer mapping. Now, we can present a code snippet to demonstrate the architecture rules source module:

```
layerDefinitions(
 techLayers(
  ( <! eval when "1">$fab{"layerM3Name"}</!>
<! eval when "1">$fab{"layerM3Num"}</!>
<! eval when "1">$fab{"layerM3Abbrev"}</!>  )
  ( <! eval when "1">$fab{"layerIndm3rName"}</!>
<! eval when "1">$fab{"layerIndm3rNum"}</!>
<! eval when "1">$fab{"layerIndm3rAbbrev"}</!> )
 )
 techLayerPurposePriorities(
  ( <! eval when "1">$fab{"layerM3Name"}</!>                        drawing )
  ( <! eval when "1">$fab{"layerIndm3rName"}</!>                    drawing )
  ( <! eval when "1">$fab{"layerM3Name"}</!>                        pin     )
  ( <! eval when "1">$fab{"layerM3Name"}</!>                        net     )
  ( <! eval when "1">$fab{"layerIndm3rName"}</!>                    net     )
 )
 techDisplays(
  ( <! eval when "1">$fab{"layerM3Name"}</!>        drawing
<! eval when "1">$fab{"layerM3Name"}</!>              t t t t t )
  ( <! eval when "1">$fab{"layerIndm3rName"}</!>    drawing
<! eval when "1">$fab{"layerIndm3rName"}</!>          t t t t t )
  ( <! eval when "1">$fab{"layerM3Name"}</!>        pin
<! eval when "1">$fab{"layerM3Name"}</!>pin          t t t t t )
  ( <! eval when "1">$fab{"layerM3Name"}</!>net     drawing
<! eval when "1">$fab{"layerM3Name"}</!>net          t t t t t )
  ( <! eval when "1">$fab{"layerIndm3rName"}</!>net drawing
<! eval when "1">$fab{"layerIndm3rName"}</!>net       t t t t t )
 )
)
```

Once translated, this code section will define the third metal layer and an inductor recognition layer properties in the EDA tool. Primarily, we need to substitute the

proper fabrication process name for these equivalent layers, as well as the layer number to be used. The `<! eval when "CONDITION">...</!>` tag is evaluated by the translation engine. When the `CONDITION` is true, the payload of the tag is evaluated. In these example instances, fabrication process information from the application model is substituted by the evaluation. For example, the variable `$fab{"layerM3Name"}` has been defined as *M3* in the application model, so everywhere that the `eval` tag with this variable is evaluated, only the text *M3* will remain after translation. The resulting output from the translation engine is given:

```
layerDefinitions(
 techLayers(
  ( M3          27   M3  )
  ( INDM3R      95   LM3R )
 )
 techLayerPurposePriorities(
  ( M3                      drawing    )
  ( INDM3R                  drawing    )
  ( M3                      pin        )
  ( M3                      net        )
  ( INDM3R                  net        )
 )
 techDisplays(
  ( M3           drawing      M3              t t t t t )
  ( INDM3R       drawing      INDM3R          t t t t t )
  ( M3           pin          M3pin           t t t t t )
  ( M3net        drawing      M3net           t t t t t )
  ( INDM3Rnet    drawing      INDM3Rnet       t t t t t )
 )
)
```

After translation, all of the tags have been substituted for customized values. The equivalent code snippet is provided for the Laytools® layer mapping:

```
layeralias <! eval when "1">$fab{"layerM3Num"}</!>
<! eval when "1">$fab{"layerM3Name"}</!>
    ## metal3
layeralias <! eval when "1">$fab{"layerIndm3rNum"}</!>
<! eval when "1">$fab{"layerIndm3rName"}</!>
    ## m3 inductor recognition layer
layeralias 113 <! eval when "1">$fab{"layerM3Name"}</!>PIN
    ##  m3 net marking layer
```

As well as the translation output:

```
layeralias 28 M3
    ## metal3
layeralias 95 INDM3R
    ## m3 inductor recognition layer
layeralias 113 M3PIN
    ##  m3 net marking layer
```

As also mentioned in the Section 3.2, the Laytools® EDA tools does not use layer purposes, so a separate pin layer on a unique layer number is require for the third metal layer in the target code. Otherwise the pairs of code provide equivalent functionality.

We had further mentioned the need for each PDK to have a display scheme definition. A section of the architecture rules source file for Laytools® is shown:

```
layercolor  15 <! eval when "1">$fab{"layerM3Name"}</!>
layercolor  15 <! eval when "1">$fab{"layerM3Name"}</!>PIN
layercolor   3 <! eval when "1">$fab{"layerIndm3rName"}</!>
fill  15 <! eval when "1">$fab{"layerM3Name"}</!>
```

This section of a source file corresponds to the layer mapping presented in this section and translates into:

```
layercolor  15 M3
layercolor  15 M3PIN
layercolor   3 INDM3R
fill  15 M3
```

Already the name of the third metal layer has been used repeatedly, so our MDA process helps to prevent errors by enforcing consistency.To further stress this point, we present the equivalent display resource file for Cadence®:

```
(drDefinePacket
 ( display       <! eval when "1">$fab{"layerM3Name"}</!>
backSlash  solid       winColor2  winColor2  )
 ( display       <! eval when "1">$fab{"layerM3Name"}</!>pin
solid      solid       winColor2  winColor2  )
 ( display       <! eval when "1">$fab{"layerM3Name"}</!>net
blank      solid       winColor2  winColor2  )
 ( display       <! eval when "1">$fab{"layerIndm3rName"}</!>
```

```
blank      solid       green      green      )
 ( display       <! eval when "1">$fab{"layerIndm3rName"}</!>net
blank      solid       green      green      )
)
```

along with its translated output:

```
(drDefinePacket
 ( display  M3         backSlash  solid       winColor2  winColor2  )
 ( display  M3pin      solid      solid       winColor2  winColor2  )
 ( display  M3net      blank      solid       winColor2  winColor2  )
 ( display  INDM3R     blank      solid       green      green      )
 ( display  INDM3Rnet  blank      solid       green      green      )
)
```

As with the Laytools® method, colors and fill patterns are reused to present a pleasing display scheme with some loss of information. However, we try to use the same color for all third metal layers, and differentiate them with fill patterns. As the pins are small and used sparingly, we fill them solid. And since nets are only viewed when examining an extracted view containing the merged outline of physical layers only (no implant layers or recognition layers), we use no fill for net layers. Of course, the primary substitution has been for the base layer name, *M3*.

In Section 3.4, we discussed routing assistance in the Cadence® PDK. From the architecture rules source code, a section for the second metal layer-to-third metal layer via is shown:

```
  ( VIA23 <! eval when "1">$fab{"layerV23Name"}</!> drawing <! eval
when "1">$fab{"layerM2Name"}</!> drawing <! eval when
"1">$fab{"layerM3Name"}</!> drawing
     <! eval when "1">$fab{"drcV23W"}</!> <! eval when
"1">$fab{"drcV23W"}</!> (1 1 <! eval when
"1">$fab{"drcV23W"}+$fab{'drcV23Sp'}</!> <! eval when
"1">$fab{"drcV23W"}+$fab{'drcV23Sp'}</!> center center) <! eval when
"1">$fab{"drcV23M2Enc"}</!> <! eval when "1">$fab{"drcV23M3Enc"}</!>
_NA_ )
```

and translates into:

```
  ( VIA23 V23 drawing M2 drawing M3 drawing
     0.45 0.45 (1 1 0.9 0.9 center center) 0.15 0.15 _NA_ )
```

In this source/target code pair, DRC rules are used to provide some of the necessary values to define the symbolic pin, along with the necessary layer names. Additionally, some of the substituted number are the result of mathematical combinations of DRC rules. The `eval` tag handles these computations as well during evaluation.

Similarly, the MATLAB® architecture rules source code references a number of DRC rules (although without computation):

```
function <! eval when "1">$fab{'deviceIndm3Name'}</!> (w,g,d,n,s,crossunder)
%%============================================================================
%% Process:  <! eval when "1">$fab{"description"}</!>
%% Contents: Parameterized Cell <! eval when "1">$fab{'deviceIndm3Name'}</!>
%% File: <! eval when "1">$fab{'deviceIndm3Name'}</!>.m
%% Author: John Sheridan Fisher
%% Created by 'misimda' on: <! eval when "1">`date`</!>
%% Maintainer: John Sheridan Fisher
%%============================================================================
%%
%% Update History:
%% Date        Update Details
%%----------------------------------------------------------------------------
%%
%%----------------------------------------------------------------------------
%%

    %% design rules
    mfgGrid = <! eval when "1">$fab{"drcMfgGrid"}</!>;
    m1Width = <! eval when "1">$fab{"drcM1W"}</!>;
    m1Spacing = <! eval when "1">$fab{"drcM1Sp"}</!>;
    v12Width = <! eval when "1">$fab{"drcV12W"}</!>;
    v12Spacing = <! eval when "1">$fab{"drcV12Sp"}</!>;
    v12M1Surr = <! eval when "1">$fab{"drcV12M1Enc"}</!>;
    v12M2Surr = <! eval when "1">$fab{"drcV12M2Enc"}</!>;
    m2Width = <! eval when "1">$fab{"drcM2W"}</!>;
    m2Spacing = <! eval when "1">$fab{"drcM2Sp"}</!>;
    v23Width = <! eval when "1">$fab{"drcV23W"}</!>;
    v23Spacing = <! eval when "1">$fab{"drcV23Sp"}</!>;
    v23M2Surr = <! eval when "1">$fab{"drcV23M2Enc"}</!>;
    v23M3Surr = <! eval when "1">$fab{"drcV23M3Enc"}</!>;
    m3Width = <! eval when "1">$fab{"drcM3W"}</!>;
    m3Spacing = <! eval when "1">$fab{"drcM3Sp"}</!>;
```

These rules form the basis for the geometric constructs from the pcell. By coding the pcell based on DRC rules, rather than aggregated values, the code is more portable to other processes. In fact, this method sometimes avoid conceptual errors where multiple sets of DRC rules may govern some specific spacing or size in a pcell, and which set of rules needs to be obeyed varies depending on the parameters given to the pcell. Since an exhaustive search of input parameters is never possible, capturing this duality in code based on specific DRC rules is paramount.

As an example, we might consider the spacing between NMOSFETs, where the diffusion spacing, metal spacing, and contact spacing might all have input into the actual proximity two adjacent NMOSFETs. Depending on the current density expected for the devices, the number of contact columns and width of metal contacting the source/drain regions will shift the minimum possible spacing between the devices.

The translated output of the present source code is also given:

```
function indm3 (w,g,d,n,s,crossunder)
%%============================================================================
%% Process:  AMI 2-poly 3-metal 0.6um CMOS Process
%% Contents: Parameterized Cell indm3
%% File: indm3.m
%% Author: John Sheridan Fisher
%% Created by 'misimda' on: Wed Feb  1 18:10:12 EST 2006
%% Maintainer: John Sheridan Fisher
%%============================================================================
%%
%% Update History:
%% Date        Update Details
%%----------------------------------------------------------------------------
%%
%%----------------------------------------------------------------------------
%%

    %% design rules
    mfgGrid = 0.05;
    m1Width = 0.45;
    m1Spacing = 0.45;
    v12Width = 0.45;
```

```
    v12Spacing = 0.45;
    v12M1Surr = 0.15;
    v12M2Surr = 0.15;
    m2Width = 0.50;
    m2Spacing = 0.50;
    v23Width = 0.45;
    v23Spacing = 0.45;
    v23M2Surr = 0.15;
    v23M3Surr = 0.15;
    m3Width = 0.60;
    m3Spacing = 0.60;
```

This target code section include a file header, where the process to which the target has been customized is noted, along with a date stamp for the translation process.

In Section 3.3, we presented code snippets from PDK files for physical verification. DRC code was presented, and here we present the architecture rules source code for another section of a Cadence®DRC file:

```
 ;;**********  Metal3 Layer DRCs ************************************
 drc( m3 (width < <! eval when "1">$fab{"drcM3W"}</!>)
"<! eval when "1">$fab{"drcM3WName"}</!>: min <! eval when "1">
$fab{"layerM3Name"}</!> width = <! eval when "1">$fab{"drcM3W"}</!>um" )
 drc( m3 (sep < <! eval when "1">$fab{"drcM3Sp"}</!>)
"<! eval when "1">$fab{"drcM3SpName"}</!>: min <! eval when "1">
$fab{"layerM3Name"}</!> spacing = <! eval when "1">$fab{"drcM3Sp"}</!>um" )
 drc( m3 (notch < <! eval when "1">$fab{"drcM3Sp"}</!>)
"<! eval when "1">$fab{"drcM3SpName"}</!>: min <! eval when "1">
$fab{"layerM3Name"}</!> spacing = <! eval when "1">$fab{"drcM3Sp"}</!>um" )
 drc( widem3 (sep < <! eval when "1">$fab{"drcWidem3M3Sp"}</!>)
"<! eval when "1">$fab{"drcWidem3M3SpName"}</!>: min wide
<! eval when "1">$fab{"layerM3Name"}</!> spacing =
<! eval when "1">$fab{"drcWidem3M3Sp"}</!>um" )
```

It should be noted that in our implementation of the DRC file architecture rules, we opted to create an internal layer statically named *m3*. The fabrication process name for this layer might be *M3*, *metal3*, *ML3*, or *MTOP*, so here may be some difference between the internal DRC layer names and the process layer name. This difference

could be coded out using our MDA process, but would further complicate the verification files without changing the output messages.

Regardless, the translation engine reads the `eval` tags and uses the fabrication information from the appropriate associative array to produce the DRC target file, where the corresponding code snippet is given:

```
 ;;**********  Metal3 Layer DRCs ************************************
 drc( m3 (width < 0.60) "M3.1: min M3 width = 0.60um" )
 drc( m3 (sep < 0.60) "M3.2: min M3 spacing = 0.60um" )
 drc( m3 (notch < 0.60) "M3.2: min M3 spacing = 0.60um" )
 drc( widem3 (sep < 1.00) "M3.3: min wide M3 spacing = 1.00um" )
```

Continuing our theme in this section, these DRC rules are for the third metal layer. The parallel DRC code for the Laytools® PDK is also shown:

```
  !! *** m3 Layer DRCs ***

  !! m3 width
  *drcM3W = m3 WITH WIDTH LT <! eval when "1">$fab{"drcM3W"}</!> /DRC
/OUT /LIST="<! eval when "1">$fab{"drcM3WName"}</!>: min
<! eval when "1">$fab{"layerM3Name"}</!> width
= <! eval when "1">$fab{"drcM3W"}</!>um"

  !! m3 spacing
  *drcM3Sp = m3 WITH DISTANCE LT <! eval when "1">$fab{"drcM3Sp"}</!>
/DRC /OUT /LIST="<! eval when "1">$fab{"drcM3SpName"}</!>: min
<! eval when "1">$fab{"layerM3Name"}</!>
spacing = <! eval when "1">$fab{"drcM3Sp"}</!>um"
  *drcM3N = m3 WITH NOTCH LT <! eval when "1">$fab{"drcM3Sp"}</!> /DRC
/OUT /LIST="<! eval when "1">$fab{"drcM3SpName"}</!>: min
<! eval when "1">$fab{"layerM3Name"}</!> spacing =
<! eval when "1">$fab{"drcM3Sp"}</!>um"

  !! wide m3 spacing
  *drcWidem3M3Sp = widem3 WITH DISTANCE LT
<! eval when "1">$fab{"drcWidem3M3Sp"}</!> TO m3 /DRC /OUT
/LIST="<! eval when "1">$fab{"drcWidem3M3SpName"}</!>: min wide
<! eval when "1">$fab{"layerM3Name"}</!> spacing =
<! eval when "1">$fab{"drcWidem3M3Sp"}</!>um"
```

and translated:

```
  !! *** m3 Layer DRCs ***
```

```
  !! m3 width
  *drcM3W = m3 WITH WIDTH LT 0.60 /DRC /OUT /LIST="M3.1: min M3 width = 0.60um"

  !! m3 spacing
  *drcM3Sp = m3 WITH DISTANCE LT 0.60 /DRC /OUT /LIST="M3.2: min M3
spacing = 0.60um"
  *drcM3N = m3 WITH NOTCH LT 0.60 /DRC /OUT /LIST="M3.2: min M3
spacing = 0.60um"

  !! wide m3 spacing
  *drcWidem3M3Sp = widem3 WITH DISTANCE LT 1.00 TO m3 /DRC /OUT
/LIST="M3.3: min wide M3 spacing = 1.00um"
```

We want to re-emphasize here that these rules check the same DRCs and provide the same output messages, as those from Cadence®. The language is different, but each phrase asks the exact same question and provides a one-to-one relation.

Our final pairs of code are for layout extraction during physical verification. Here will show the architecture rules source code for extracting a third metal layer inductor from layout in the Cadence® tool:

```
  ;;***** Extract m3 inductor device ***********
  extractDevice( indM3 (wireM3 "PLUS" "MINUS")
     "<! eval when "1">$fab{'deviceIndm3Name'}</!> auLvs PRIMITIVES" physical)

    saveRecognition( indM3 "<! eval when "1">$fab{"layerM3Name"}</!>")
```

There are only two tags in this source snippet. One is for the netlist primitive to be generated and the other is for the net layer on which to bind the extracted primitive. The other logical layer names are internal only and do not need to be translated. So the given target model is almost identical:

```
  ;;***** Extract m3 inductor device ***********
  extractDevice( indM3 (wireM3 "PLUS" "MINUS")
     "indm3 auLvs PRIMITIVES" physical)

    saveRecognition( indM3 "M3")
```

Likewise, the Laytools® equivalent actually has no tags for the translation engine to evaluate. The netlist primitive has already been abstracted by the `&indm3Model` tag.

And Laytools® does not create an extracted view that requires net layer onto which the netlist primitive needs to be bound. As such the source file snippet and target file snippet are identical for this section of code.

This concludes our discussion of the MDA process as applied to PDK development. This first example of an MDA process in mixed-signal system design was chosen because it is a focused example. The models, abstractions, sources, and targets are obvious with no previous methodologies to supplant. The results and improvement, as well as limitations, can be clearly presented. In Chapter 5, we begin our discussion of mixed-signal system design flow in an effort to develop how the MDA process can be applied. Hopefully, the MDA process presented in this chapter will help to maintain clarity within the complexities of the mixed-signal system design flow.

CHAPTER 5

INTRODUCTION TO EXISTING MIXED-SIGNAL DESIGN METHODOLOGIES

5.1 Established Design Methodologies

Existing microchip design methodologies are reflected in the structure of the EDA tools on the market. With time-to-market (TTM) often being the primary factor in a product's profitability [11, page 3], the microchip design market demands that their EDA tools are constantly improved with more efficient design automation. Every aspect of microchip design where human intervention is involved can be evaluated to discern if a computer could perform the task, or some part of it, and at what cost (i.e. performance, power dissipation, die area).

Since the early 1990s [12], the digital design flow has been automated from the point of user-generated behavioral hardware definition language (HDL) models. These models are reusable because they are process-independent and entirely abstracted from the implementation details. The entire digital system can be re-spun to a completely different technology by simply synthesizing the behavioral information, then placing and routing these process-abstracted structural models to generate completed layout

implementation, which is an entirely automated process; although there are many choices for the user to make about how the structure will be implemented in this automated flow, the design flow is structured to prevent the engineer from corrupting the structural design so that it would no longer match the behavioral specifications. Straight-forward (but not inconsequential) improvements, responding to arising issues, have been made subsequently to improve digital performance, handle increased design complexity, and decrease necessary user input. In contrast, the analog portion of the mixed-signal design flow is only now moving beyond transistor-level schematics and full custom layout. Both are hand-generated and entirely process-dependent, because the design is still done at the structural implementation level. The transistors are painstakingly sized through approximate calculations, followed by many iterative simulations. Further, only structural topologies can be reused when re-targeting a design to a new process, because the device physics of a particular process govern the exact transistor sizes. And since design is done at the structural implementation level, even derivative designs from the original for the same fabrication process will require significant re-design and re-simulation effort. This disparity in available toolsets is referred to within the mixed-signal design community as the Gap [13].

5.2 The Gap

The Gap has developed for two reasons. First, the abstraction of the relevant functionality from restoring digital logic is much more straight-forward than the same task for analog circuitry. For digital blocks, latency and power dissipation are the primary concerns beyond logical operations. In contrast, each analog block may have

Project type	Digital gates	Analog functionality
Timing-driven designs	$5,000 - 250,000$	none
Block-based designs	$150,000 - 1,500,000$	ADC, DAC, PLL, AFE blocks
Platform-based design	$300,000+$	complete analog systems and blocks

Table 5.1: Type and Complexity of SoC Projects.

pages design specifications that are of primary concern, which all affect each other and are all caused by an overlapping set of devices.

The second factor in the growth of the Gap is that pure digital designs dominated the system-on-a-chip (SoC) projects of the 1990s and accelerated digital SoC design methodologies. Analog circuitry was purchased as off-the-shelf components to supplement the digital core, or designed separately using classical analog design methods. Only in the last few years have design teams moved beyond timing-driven designs (TDD) with from 5,000 to 250,000 gates and no mixed-signal capability, as well as block-based design (BBD) with from 150,000 to 1,500,000 gates and only ancillary mixed-signal analog-to-digital converter (ADC), analog front-end (AFE), and phase lock loop (PLL) blocks. Now design teams are trying to deal with platform-based design (PBD) with greater than 300,000 gates and mixed-signal functions and interfaces fully integrated into the heart of the system [12, page 7]. In addition, the total number of new ASIC designs beginning each year is declining [14, 15], while the number new mixed-signal ASIC designs beginning each year is actually *increasing* [15]. As would be expected, the increasing complexity of integrated systems is also driving up the size of design teams necessary to deliver the SoC while maintaining or decreasing TTM.

Therefore, EDA tool vendors are finding that, while the number of EDA tool seats is not changing substantially, the demographic landscape of these seats is shifting. There are fewer (less vocal) pure digital design seats and substantially more (more vocal) mixed-signal design seats pushing the mixed-signal design methods forward.

Further compelling the need to close the Gap is the industry expectation that mixed-signal design experts, or analog gurus, take 10–15 years to mature [16], while the number of new mixed-signal design students is waning [17]. EDA tool vendors are being asked to make more efficient design tools that less experienced engineers can successfully operate efficiently.

Of course, the Gap not only presents a problem of low efficiency in mixed-signal design compared to its digital counterpart, but also an serious issue of broken block interfaces [18]. Without a unified simulation environment to verify that the connections between the digital blocks and the mixed-signal blocks have been properly implemented, design teams are left with assuming that the members in charge of one block will coordinate with those in charge of the other, even if it is in coordinating the interpretation of a written specification. As SoC designs migrate more to PBD, where there are more such interfaces, this issue becomes more serious.

5.3 AHDL Benefits & Traps

To address this issues, extensions to existing digital HDLs have been created to handle mixed-signal systems. These (digital and) analog HDLs (AHDLs) are a superset of the

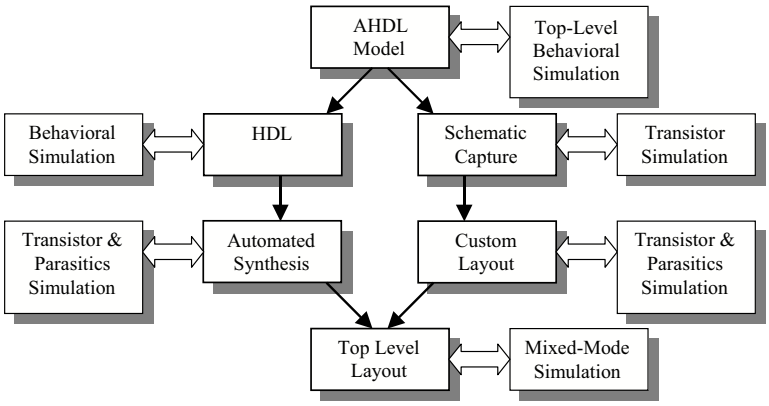


Figure 5.1: Top-Down Mixed-Signal System Design Flow Diagram.

digital HDLs and are intended to fix the connectivity issue. Other reasons to model in AHDL are to allow system design exploration & test and to encourage development of virtual component (VC) intellectual property (IP) for design reuse [18]. The digital HDL Verilog has been expanded to VerilogA then subsequently to VerilogAMS, while the digital HDL VHDL has been expanded to VHDL-AMS. Both EDA tool vendors with complete front-to-back mixed-signal microchip design packages, Cadence® and Mentor Graphics®, have completely implemented the AHDL extensions as an option into their base multi-mode simulators. Mentor Graphics®'s ADVancd-MS™ simulator, Cadence®'s Spectre™ simulator, Dolphin®'s SMASH™ simulator, and ELANIX®'s SystemView™ simulator all accept SPICE, Verilog, VHDL, VerilogA, VerilogAMS, VHDL-AMS, and C-code formats. Some simulators accept MATLAB® code as well.

With the ability to simulate the entire system behaviorally, the present top-down design approach is shown in Figure 5.1. It is important to note that AHDL simulations take longer than HDL simulations. So pure digital blocks should still be modeled in pure digital HDLs, which also lend themselves to automated synthesis, place, and route. Also while coding the complete SoC into an AHDL at the system top-level does solve the connectivity dilemma, there is still the issue of efficiency. In fact, some engineers argue that adding a design level to capture the top-level in the AHDL increases TTM and decreases system accuracy [18]. There is a non-trivial effort involved in developing the AHDL models for the mixed-signal blocks that does not seem to reduce the subsequent design work necessary to advance these blocks to layout. And as mentioned previously, these models are difficult to generate for mixed-signal blocks, because of their specification complexity.

There is a tremendous issue in understanding what properties can be safely ignored and which must be abstracted to the behavioral view. Failing to properly identify and model some crucial aspect of the mixed-signal implementation will lead to a false sense of comfort with the blocks and can allow the design to be implemented in a detrimental manner, unbeknownst to the design team. For example, consider designing a digital-to-analog converter (D/A) using a resistor divider topology. If the matching of the resistors is evaluated at the AHDL level, but the abstraction into an entirely behavioral design of these passive devices misses some details, the accuracy of the D/A will be compromised when implemented as a result of the dubious AHDL simulation output. Even if the missed details are discovered later in the design, TTM

will be impacted as the top-level behavioral models have to be re-designed and re-simulated. Clearly, the initial AHDL models need to be as accurate as possible and to allow that for additions as the design process illuminates more understanding of the block's behavior.

5.4 Optimization & Automation

In addition to providing AHDL modeling, tools are now beginning to provide help with mixed-signal schematic design and layout. While both Mentor Graphics® and Cadence® have provided tools that (to varying extents) blindly place and route the layout for the designer, the mismatching and parasitics devices incurred with these packages have made them a risk in the mixed-signal design flow. Neolinear® has developed IP and software that intelligently places and routes layout for a designer, with a specific concern for the matching and parasitic devices in a design. Cadence® has purchased and incorporated this technology into their package bundle. Schematics in Cadence® are designed with tags on the devices that indicate how they should be related (i.e. common centroid, interdigitation, dummy-ends, etc.), as well as a level of concern about their parasitic coupling. The mask design engine then automates the layout procedure. The purchased Neolinear® technology also includes libraries of common mixed-signal block topologies, which are parameterized and can automatically be customized to meet design specifications. For example, a user can pick an opamp topology and his/her relevant design specifications, and the package will optimize the device sizes in an attempt to meet the her/his specifications. This style of optimization software is not unique to Cadence®, as a number of EDA vendors have

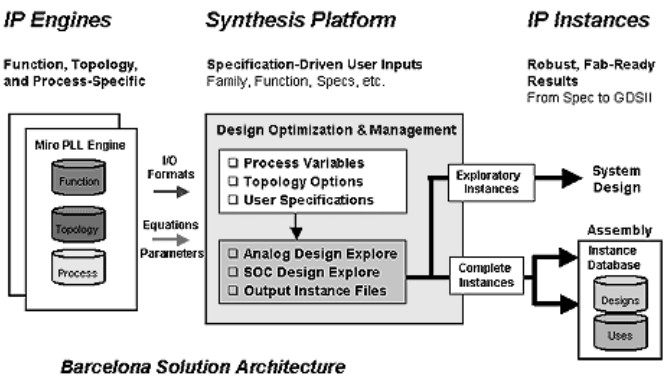


Figure 5.2: Automated Design Flow by Barcelona Design® [1].

developed similar technology.

A slightly different technology has been developed by Barcelona Design®, as shown in Figure 5.2. Their product actually takes just the design specifications for a given mixed-signal block and process technology, then automatically decides what topology to use, optimizes the devices within the topology, and generates a completed layout cell (along appropriate ancillary files). The Barcelona Design® product design team has derived exacting equations for each of their mixed-signal blocks, then can map any specified process technology into the equations and derive parameterized device sizes based on the design specifications.

5.5 AHDL Synthesis

Both schematic automation tools are dependent on a library of design blocks, but the Barcelona Design® approach is actually closer to being able to automatically

generate layout from an HDL specification. Presently, that specification would be a structural one, but libraries could easily be developed hierarchically on top of what Barcelona Design® already has created to accept behavioral input specifications. In fact, the next logical step is to develop parameterized AHDL behavioral models from the tested mixed-signal functional blocks that are available, thus creating a VC with exact AHDL model. A user would customize the VC blocks to their design specifications in AHDL, then the tool would synthesize the topology, optimize the structural schematics, and generate the layout, giving careful consideration to parasitic devices. This methodology allows the entire SoC to be synthesized from a completed, behavioral AHDL top-level model. It also removes the increased design effort where engineers have to generate AHDL models and port them to sized schematics, which was a concern for product TTM. Concerns about matching the schematic design and implementation to the initial model are also mitigated, because the AHDL was generated from VC test results. Of course, this methodology is most effective in scenarios where an IP repository of topologies can be developed or purchased for reuse.

CHAPTER 6

ISSUES WITH EXISTING MIXED-SIGNAL DESIGN METHODOLOGIES

In this chapter, we will discuss work already completed that motivates the development of mixed-signal design flows. Specifically, we will describe each of three projects and the weaknesses in existing design methodologies that caused issues in the project.

6.1 Control CODEC Design Example

For our first design example of the methodology that we have discussed and its weaknesses, we will examine one application of a Control CODEC. While the conventional concept of a signal coder-decoder (CODEC) is of a homogeneous device, such that the output decoding is simply an inverse of the input decoding, the heterogeneous CODEC decouples the input and output decoding; this variation generalizes the system. A Control CODEC is an application of a heterogeneous CODEC, which replaces a traditional discrete analog control loop [19]. The analog control loop includes an ADC (coder) plus a DAC (decoder), as well as other peripheral signal processing algorithms. Our application of the Control CODEC is to improve the control of RF transponders in satellites. We consider this control situation, in which case it

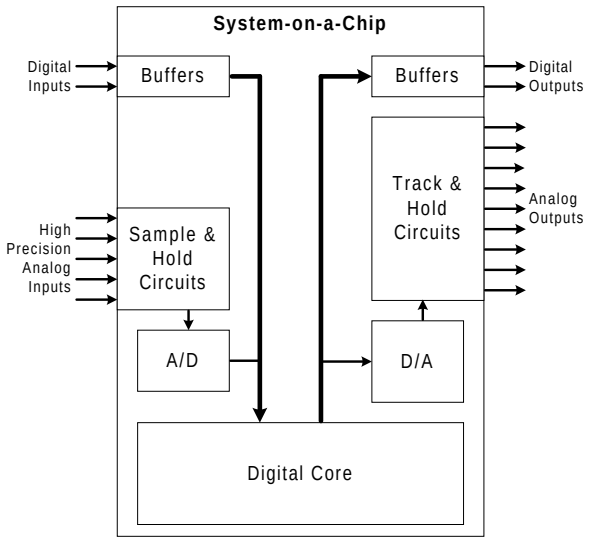


Figure 6.1: System-on-a-Chip CODEC Block Diagram.

is typical that the signal structure of the analog inputs (temperature, RF power) is significantly different from the analog outputs (RF gain control). Nevertheless, the ADC and DAC converters are optimized for the problem phenomenology at hand (the control loop), so it is consistent to consider the problem in the mixed signal domain as the design of a Control CODEC, as an application of a heterogeneous CODEC. A generalized system-on-a-chip CODEC block diagram is shown in Figure 6.1. Our Control CODEC is a specific application of the generalized CODEC where the input and output characteristics are decoupled, and where the design replaces an analog control loop.

6.1.1 Control CODEC Application

We choose the satellite application as our design example because we feel that it illustrates a number of important aspects of the developing design methodology. In space-borne microelectronics, system weight and challenging environmental conditions are primary motivations to condense large discrete systems onto a single chip in one of the most advanced fabrication processes available. Combining the system onto a single die saves weight, which is a serious design consideration; the cost to launch a just single kilogram into outer-space is measured in the thousands of dollars (US) [20]. Using more advanced fabrication processes both decreases the weight (by increasing die density) and improves environmental immunity; more advanced technologies have thinner gate oxides, which are more resistant to long term radiation damage [21]. So, there is a strong motivation to migrate satellite discrete analog control loops to control CODEC designs, as well as to continually re-target these completed designs to the most advanced fabrication available.

Re-targeting designs to new fabrications lines has traditionally been an expensive and time consuming process. In addition, satellite microelectronics need to work reliably as a system, as there is often no possibility to repair or replace faulty hardware once in orbit. Further, the satellite control systems of our application are commodity designs, not pushing the upper frequency boundaries and lower noise limits of a given fabrication technology. Finally, with the input signal and output signal characteristics completely decoupled reuse of cell blocks (or even sub-blocks) between the input signal path and output signal path may not be viable, which will increase expense

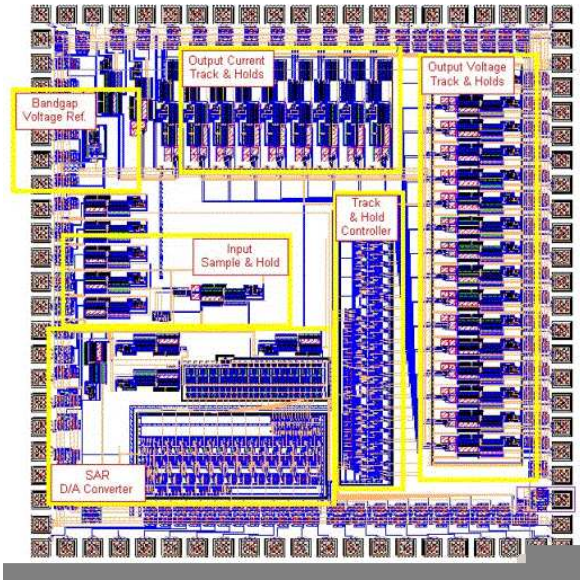


Figure 6.2: Full Custom Satellite Control CODEC Layout.

and design time.

Figure 6.2 is the layout for the first silicon of a relatively simple control CODEC, which took three months to generate with the equivalent of six full time engineers. The top-level specifications were generated on paper in a couple days, with only minor additions during the project. Subsequently, the design was partitioned into blocks among the design team. While the digital section was designed by a separate group, the mixed-signal design team dealt with methodology outlined in Section 5.1.

6.1.2 Control CODEC Design Methodology Issues

One issue that caused the first silicon not to function as specified was in the design partitioning. The polarity of the digital inputs for the DAC control logic did not

match the digital output buffering that had been implemented. In addition, on the day before submitting the layout to the foundry, another design partitioning issue was discovered. The DAC analog output range did not match the expected input range for the track & hold (T/H) circuits, and a level shifter & gain block had to be hastily added at the last possible minute. Had any sort of AHDL modeling been done, both of these issues would have been discovered and eliminated at the beginning of the design process.

System modeling would also have ensured the reliability of the system after top-level integration. And if the models used to describe this system are specified from generalized IP libraries of VCs, then there is a direct and expedient synthesis path for the entire layout. Re-targeting the design to a new foundry becomes as simple as integrating the new IP library VCs into the system modeling. A design that took months to target to one foundry by a team of engineers could be accomplished in just days, targeting multiple processes, by a single engineer working with a set of IP libraries.

Creating AHDL models would also have forced the DAC to have been abstracted and the accuracy of the resistors may have been considered. Within our design team, the DAC schematic design was done by a different engineer than its mask design was. In this hand-off, any consideration of resistor matching was not communicated, so the layout was done with regularity for quick construction as the primary factor. Consequently, the resistor segments were too far apart, and the accuracy of the DAC did

not meet the specification.

Finally, the design methodology broke down again when reporting the bonding diagram to the customer. The die pin-out was transcribed by hand to a different piece of software for documentation, which led to the primary power pin being swapped with a test pin. The introduction of engineer intervention in the design flow without automated verification caused an error to be made. A similar comparison error could just as easily be made when evaluating whether the AHDL simulation results match the schematic simulation results. Some form of automated comparison is necessary to ensure the integrity of the design flow.

6.2 LVA1 Process Evaluation Design Example

Our second example is a multi-team process evaluation design. The joint project between Ohio State, Wright State University, and the Air Force Research Laboratory was intended to evaluate the experimental MITLL AST LVA1 $0.25\mu m$ fully-depleted silicon-on-insulator (FDSOI) CMOS process.

Because this SOI process was fully-depleted, the source and drain diffusions of MOSFETs extended all the way down to the insulator of the wafer. While this feature is ideal for low leakage digital designs, mixed-signal layout suffers when trying to place body contacts. The only place to contact the body terminal is at the ends of the gate, because the gate shields the underlying active from being implanted as diffusion. In addition, all active regions are mesa islands of silicon on a wafer of insulator, so the

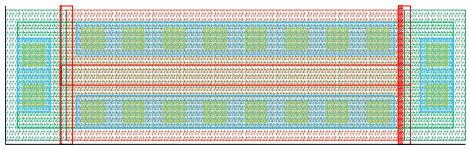


Figure 6.3: MITLL AST LVA1 H-gate Layout Example.

entire wafer is silicided after the polysilicon layer is in place. The silicide sharply lowers the surface resistivity of all silicon, diffusion, and polysilicon and shorts together any of these co-located regions on the same mesa-island.

Specifically, diodes become difficult to make, because even making one island of active with two different doping polarities only forms a diode under the surface, which is shorted by the silicide applied to the diffusion surface. The only manner in which to eliminate the silicide shorting is to place a polysilicon path on top of the diode boundary. With the polysilicon path on the diode boundary, the diffusion implants are no longer directly applied to the boundary, but its junction potential changes too.

Now, since body contacts must be placed at the ends of the MOSFET, the gate no longer is able to be simply defined just as the area where polysilicon and active overlap. In fact, body diffusion must directly abut the source/drain diffusion to end the channel width. However, in order to prevent both the source and drain from being shorted to the body, a polysilicon path must be placed over their diodes' boundaries. The resulting structure is referred to as an H-gate device, and is shown in Figure 6.3. The H-gate arms separate the source/drain diffusion (between the arms) from the body diffusion (outside the arms). Of course, device width is now a function of the

horizontal (relative to a flat wafer) traversal of source/drain diffusion implant under the H-gate arms and is thus poorly controlled.

This contrived structure was one of two recommended methods for creating MOSFETs with body contacts. The other had even poorer controlled device width and shorted the body contacts to the source. Body diffusion implant (just slightly overlapping the gate) was interspersed within the source implant. Device width in the source-sided body contacted (SBC) MOSFET is said to be the sum of the widths of all of the source implants and is thus controlled by the accuracy of the vertical diffusion implant masks!

The cost of not using a body-contacted device is that in the saturation region, there is a kink effect, which causes the device current to increase exponentially with V_{DS}. Regenerative CMOS digital logic is unaffected by this phenomenon because one half of the devices are always off with very little leakage current. Even in switching, the transition current will only find a path through transistors in the linear region.

These sorts of issues were the purpose of our evaluation designs. The process had been well qualified and used for commercial digital designs, but the foundry wanted to expand into the mixed-signal market as well. This lack of experience in the mixed-signal domain was plain in sight of the body-contact issue as well as the fact that there were no precisely controlled resistors or capacitors.

6.2.1 LVA1 Process Evaluation Blocks

To evaluate the process for mixed-signal purposes, Wright State submitted a fully-differential opamp, a fully-differential comparator, and a buffer, which all simulated for GPS speeds. Our team at Ohio State submitted baseband elements including: an experimental opamp, a classical opamp, a voltage bias generator, a first-order continuous-time delta-sigma modulator (DSM), a simple sample & hold (S/H), a offset-independent S/H, and a track & hold (T/H). The T/H is a S/H combined with a moving-average (MA) filter, and was tuned so that its output would converge half-way to the input on each clock cycle. The DSM had a continuous-time integrator, instead of a switched-capacitor implementation, and had an $RC = 62.5ns$ (16MHz). Each block was to be bonded out separately, so Wright State assembled all of the blocks into individual pad frames with electro-static discharge (ESD) protection.

6.2.2 LVA1 Design Methodology Issues

Unfortunately, the limited ESD protection documentation was not understood and all of the pads were wired with power and ground shorted together. This severe problem illuminates a major weakness in many design flows. While each group was able to verify with automated tools that the layout of each block matched its corresponding schematic, there was no schematic generated for any of the pad/ESD blocks within pad frames. Since ESD protection can be extremely complicated, automated layout-versus-schematic (LVS) at the top-level is essential; visual inspection is not sufficient and can cause the entire design project to completely fail, as was the case for this

project.

Of course, the design kit must be set-up to handle this verification. The foundry provided both the ESD protection cells and the design kit. In this example, the tool could perform the verification, but the foundry did not provide schematics to correspond to the ESD protection layout provided. In haste to submit the designs, Wright State was not able to go through the complicated and time intensive process of making schematics to match these provided layouts.

In addition, our team's work was done in Cadence®, while Wright State and MITLL AST both worked with Mentor Graphics® tools. Since our toolsets did not match, our schematics would not have been useful to Wright State, who would have had to re-enter them to complete the LVS verification discussed above. We were fortunate that MITLL AST was able to provide any form of Cadence® design kit, since their development was done entirely in Mentor Graphics® tools. This problem illustrates the possible side-effect that needs to be understood when using best-in-class tools, instead of using all of the packages from one EDA tool vendor.

Even the design process of this example was not without difficulty. The only resistor available was extended lengths of polysilicon. Since this material can be used for routing as well, the design kit could not differentiate between polysilicon routing and polysilicon resistors, and thus did not extract polysilicon resistors. This problem was especially difficult in the DSM block where a resistor divider created a bias voltage between power and ground. Without extracting the resistor devices, the verification

deck dubiously flagged the block as having shorted power and ground together. In addition, the LVS path was broken, because schematics for LVS (without resistors) had to be generated by hand to complete the verification. This manual process introduces a new range of possible errors, because there is no automated manner to verify that the schematics were changed correctly. In a proper verification flow, the schematics are simulated, then verified to match the layout. Here, the schematics had no resistors, so ground and power were shorted, and thus would not simulate. The design kit should have been created with a logical layer (as opposed to a physically fabricated layer) that would be used to identify polysilicon that is being used as a resistor. Such a logical layer would have fixed our local verification issues.

However, we also had some concerns about the veracity of our simulations involving the polysilicon resistors. Because the resistors are just bare polysilicon snaking across the etched-out wafer insulation, there was concern that the simplest resistor model of only a resistance value was not sufficient, even at room temperature (where temperature coefficients do not have an effect). Some foundries release sub-circuit models and temperature & voltage coefficients for their resistive devices, but rarely is attention given to how much effect different models of the parasitic distributed devices have on the resistive device performance. For example, should the distributed parasitic capacitance of a resistor to its substrate or backplane be lumped into one capacitor at one of the ports or one capacitor in the middle of the resistance value or some other configuration? Working to evaluate the mixed-signal performance of an experimental process caused these sorts of issues to be questioned.

A final consideration for this design example is how the design kit construction impacted our efficiency in generating the layout. Because of the complexity inherent in a FDSOI process, the design rules for layout were substantially more cumbersome. However, this complexity mainly effected the layout of MOSFETs, so if the foundry had been able to provide MOSFET parameterized cells (p-cells), our layout efficient would have been greatly improved. Layout p-cells automatically place a device into the layout, whose various properties (i.e. width, length, multiplier) can be entered into a form to update the layout. Being conscious of our lack of p-cells, we designed schematics with a minimal number of device sizes and reused them heavily. Once all of the devices are placed, the only task remaining is routing, for which modern layout editors provide some routing path automation. The standard user-interface allows the designer to route a signal path by indicating where each corner will be in the layout with the mouse, and allows the designer to change routing layers along the path. While our LVA1 design kit allowed for the former, the latter was not possible; everywhere that routing layers changed, a contact had to be placed by hand. A few simple lines of code to add this contact automation would have greatly enhanced the design kit's efficiency.

6.3 Transducer-to-Digital-Pico-Network Design Example

We present the design of a transducer-to-digital-pico-network interface for our final example of the design methodologies. This design is a difficult and complex task,

primarily because the phenomenology of the transducer is generally not well characterized over electrical and atmospheric (temperature, pressure, humidity, etc.) variations. In contrast, the digital pico-network or local area network (LAN) protocol is generally well defined in a specification, such as IEEE 1451 [22]. For our discussion, we will assume that a sensor generates an analog electrical signal that we wish to send over a digital network. This topology has a large variety of practical applications where size, power, or weight constraints merit a single chip design solution and include wireless audio acquisition, blimp mounted video surveillance, and remote environment monitoring all via a wireless connection to a self-configuring pico-network or LAN.

Our design example is known as the Nautilus chip from the Digital Exportation of an Established Protocol from Sensing Encoded Analog (DEEP SEA) Project; this unique chip is intended to be an integrated microchip solution for receiving a standard 4–20mA analog communications protocol, converting the received analog to 16-bit digital words and transmitting the digital words on a standard SPI (Serial Peripheral Interface) pico-network connection. The ADC uses a second-order Delta-Sigma ADC topology and is followed by a third-order sinc filter. The SPI core transmits the 16-bit digital words across a standardized, fully configurable SPI serial interface as 16-bit/8-bit data. In addition, the SPI core can receive data for the ADC, such as filter coefficients. The chip was fabricated through MOSIS on AMI's $0.6\mu m$ process. The mixed-signal sections of our chip were implemented through a standard custom design flow, while the digital sections were synthesized from VHDL using the Alliance

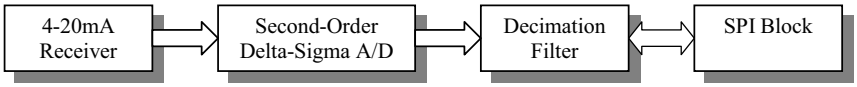


Figure 6.4: Nautilus Chip Block Diagram.

toolset.

6.3.1 4–20mA Receiver

The analog section of our mixed-signal example senses an incoming current and converts its magnitude to a digital format. Specifically, we have designed a receiver for a standard 4–20mA current loop and a second-order Delta-Sigma ADC to generate a digital pulse density stream of the data on the current loop. We chose the standard 4–20mA current loop as our "sensor" data because its electrical signal behaves like sensor data which we can explore for its design methodologies without the overhead of uncertain sensor phenomenology.

The 4–20mA receiver is a simplified version of a topology developed by Burr- Brown, now Texas Instruments [23] and is shown in Figure 6.5. The transfer function for the 4–20mA receiver is given by Equation 6.1 and Equation 6.2. As shown by Equation 6.1, V_{REF} can be adjusted to center the range of the receiver's output V_{IN} within the ADC's input range. If V_{REF} is made to be tunable, then each individual receiver can be tuned within its assembled 4–20mA system to account for inconsistencies in the

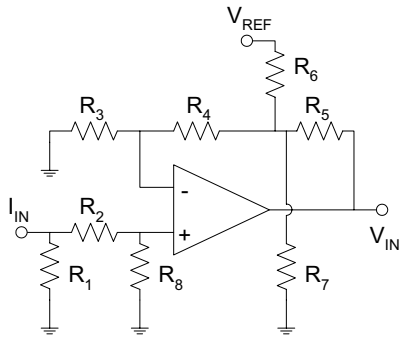


Figure 6.5: 4–20mA Receiver Schematic.

sensor input or analog block fabrication.

$$V_{IN} = \frac{R_1 R_8}{R_2 + R_1 + R_8} k I_{IN} - \frac{R_5}{R_6} V_{REF}, \tag{6.1}$$

where

$$k = 1 + \frac{R_4}{R_3} + \frac{R_5}{R_3} + \frac{R_5}{R_7}\left(1 + \frac{R_4}{R_3}\right) + \frac{R_5}{R_6}\left(1 + \frac{R_4}{R_3}\right) \tag{6.2}$$

6.3.2 Second-Order Delta-Sigma Modulator ADC

Our example ADC design is a DSM ADC whose output is to be decimated to 16-bit resolution. A DSM was chosen because of its high resolution, low die area consumption, and unique tolerance to fabrication process variations. Also, using a DSM forces the use of digital filtering, which can be used to correct a spurious response that might

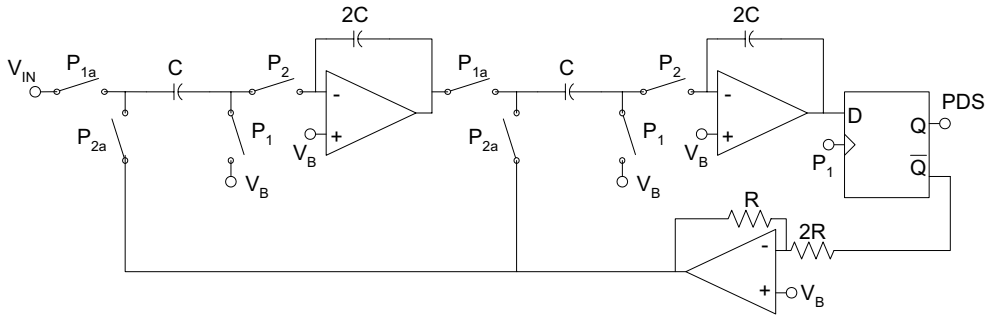


Figure 6.6: Second-Order Delta-Sigma Modulator ADC Schematic.

be found in the analog section as the design progressed. If other ADC topologies suffered this problem during design, a separate DSP would have to be added to correct spurious response; this addition would represent a major change in the design specification and have a serious impact on the TTM. The design for our second-order DSM was developed by combining aspects of a third-order DSM by Johns & Martin [24] and a second-order design discussed by Mandyam [25], as well as general topological considerations from Boser [26] and Shenoi [27].

We chose a single-ended design because it is sufficient to explore design methodologies involved without the being encumbered by the design overhead of a fully differential architecture. We use a D-flip-flop as a single bit quantizer, which will guarantee linear quantization. Our DSM is designed with the AC ground at a mid-rail bias voltage of V_B. Thus the feedback voltage is actually midway between a rail and V_B, which minimizes the maximum quantization error. We choose an over-sampling rate (OSR) of 128. This topology yields a theoretical maximum signal-to-noise ratio (SNR_{MAX}) of $100.2dB$ (just over 16.5-bits), which is a $34.6dB$ increase in SNR_{MAX} compared with

the same design choices using a first-order modulation scheme. It might then seem natural to use an even higher-order topology, however third-order and higher topologies have diminishing returns for SNR_{MAX} and do not have guaranteed stability [24].

In our design in Figure 6.6, the switched-capacitor structures each form a delaying non-inverting integrator. The signals $P1$ and $P2$ are a non-overlapping clocking pair, as are $P1a$ and $P2a$. The signal $P1a$ is a slightly delayed version of $P1$, which helps attenuate signal feed-through [25]. Assuming ideal integration, the signal transfer function (STF) is given in Equation 6.3, while the noise transfer function (NTF) is given by Equation 6.4. Of course, the pulse density stream (PDS) must be converted into 16-bit words by a decimation filter, which will be discussed in Section 6.3.3.

$$STF = \frac{z^{-2}}{1 - z^{-1} + z^{-2}} \tag{6.3}$$

$$NTF = \frac{(1 - z^{-1})^2}{1 - z^{-1} + z^{-2}} \tag{6.4}$$

6.3.3 Decimation Filter

The decimation filter section of our example takes the pulse density stream provided by the ADC as an input. It then decimates the clock rate by a factor of 128 and yields a 16-bit value as an output. The 16-bit output value is finally passed to a buffer which can be read by the SPI unit at any time. To achieve this decimation,

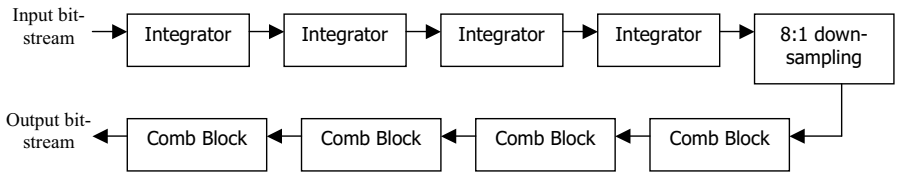


Figure 6.7: Decimation Filter Block Diagram.

we implemented a fourth-order sinc filter based on a design by Hogenauer [28]. The sinc filter was chosen for its inherent implementation stability, requiring only digital adders instead of multipliers and adders.

Our design modified the design example by Hogenauer to take a bit stream input instead of a 16-bit input. It also uses 16-bit wide columns of integrators and comb filters to do the decimation. Each integrator column is comprised of four, 4-bit carry-look-ahead adders tied together to create a 16-bit adder which takes the previous value (stored in a register) and adds the new input. Similarly the comb-filter columns are made of four, 4-bit comb filters with a differential delay of one. The pulse density stream (our input) is placed on the least significant bit of the first column of integrators.

6.3.4 SPI Block

The SPI is essentially a three-wire serial bus for eight or sixteen bit data transfer applications. The three wires carry information between devices connected to the bus. Each device on the bus acts simultaneously as a transmitter and receiver. Two of the three lines transfer data (one line for each direction) and the third is a serial

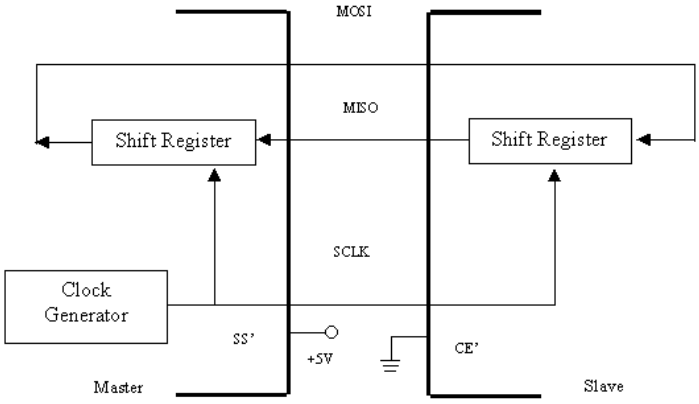


Figure 6.8: SPI Block Diagram.

clock. Some devices may be only transmitters while others are only receivers [29]. Generally, a device that transmits usually possesses the capability to receive data also. An SPI display is an example of a receive-only device while EEPROM is a receive-and-transmit device.

The devices connected to the SPI bus may be classified as Master or Slave devices. A master device initiates an information transfer on the bus and generates clock and control signals. A slave device is controlled by the master through a slave select (chip enable) line and is active only when selected. Generally, a dedicated select line is required for each slave device. The same device can possess the functionality of a master and a slave but at any point of time, only one master can control the bus in a multi-master mode configuration. Any slave device that is not selected must release (make high impedance) the slave output line.

The SPI bus employs a simple shift register data transfer scheme; data is clocked out of and into the active devices in a first-in, first-out fashion. It is in this manner that SPI devices transmit and receive in full duplex mode.

All lines on the SPI bus are unidirectional: The signal on the clock line (SCLK) is generated by the master and is primarily used to synchronize data transfer. The master-out, slave-in (MOSI) line carries data from the master to the slave and the master-in, slave-out (MISO) line carries data from the slave to the master [30], [31]. Each slave device is selected by the master via individual select lines. Information on the SPI bus can be transferred at a rate of near zero bits per second (bps) to 1 Mbps. Data transfer is usually performed in eight/sixteen bit blocks. All data transfer is synchronized by the serial clock (SCLK). One bit of data is transferred for each clock cycle. Four clock modes are defined for the SPI bus by the value of the clock polarity and the clock phase bits. The clock polarity determines the level of the clock idle state and the clock phase determines which clock edge places new data on the bus. Any hardware device capable of operation in more than one mode will have some method of selecting the value of these bits. This multi-mode capability combined with the simple shift-register architecture makes the SPI bus very versatile, and allows many non-serial devices to be used as SPI slaves.

6.3.5 Nautilus Design Methodology Issues

All of the system pieces were assembled, and immediately, a problem was discovered. The synthesized digital sections dwarfed the mixed-signal sections. In fact, they were

so large that they would not fit within just one reticle space that MOSIS provides and had to be broken into separate dice. The problem arose from the design kit that we had to use for the digital synthesis, place, and route tools that we used. The design kit that interfaces with the synthesis tool logically defines all of the necessary primitive devices as a series of one-dimensional paths with expansion rules for each type of segment to achieve design rule minimums. Once we were able to map their output layers to the ones necessary for Cadence®, we had to make the tool satisfy the design rules. This process caused all the layer to become substantially bloated. In particular, the one-dimensional paths with expansion created innumerable notches in the layout. The path segments, and thus the entire digital layout had to be bloated until these notches were absorbed.

At synthesis, we also discovered that the tool did not accept behavioral VHDL, so our code had to be ported to structural VHDL. Unfortunately, this synthesis tool did not well match our other EDA tools. Had this been known since the beginning of the project, another tool could have been chosen, but the design had already progressed too far to change methodologies by synthesis time. Although it was later discovered that four reticle spaces could be combined for one design, the original digital blocks were still too large and would still have had to be on separate dice each, as well as separate from the mixed signal blocks. This misunderstanding about combining reticle spaces was so prevalent among the users for this process that MOSIS changed their online documentation after a discussion with our design team.

A separate problem occurred at top-level integration that was not discovered until the initial dice were fabrication and tested. At testing, the mixed signal sections seemed to be non-functional, and the digital sections seemed to function extremely slowly. We then learned that MOSIS allows designs to be submitted to the AMI foundry through two design flows. One is the natural AMI design flow; the other is the MOSIS SCMOS design flow, which is intended to base all design rules off of the process feature size to make designs entirely portable. Not understanding that there was a difference in how the data for these flows was handled differently, we submitted to the AMI flow, because we knew that the chips would be fabricated at AMI. This action indicates that the n-diffusion implant in the layout data is a put everywhere that its polygons are **not** drawn, which unfortunately was not our understanding from either MOSIS or the design kit we used that had been recommended by MOSIS. Thus, while our PMOSFETs were bit weak, the NMOSFETs had no diffusion implant and were extremely slow. The description of this design flow and the layers associated with it were also updated in the MOSIS online documentation after a discussion of this issue. To properly re-fabricate our designs, no changes were made in the project database; the designs were just streamed out differently and submitted again.

This problem illustrates a weakness in most any design flow that is difficult to overcome: transferring layout data to the foundry is a vulnerable point in any design flow. The standard methodology for sending data to a foundry to stream the layout to a GDS-II file, where every design layer is assigned a number. If the layer numbers assigned do not match those expected, the design would be mis-fabricated. Hopefully, the foundry will open the data and inspect it to ensure that what they have

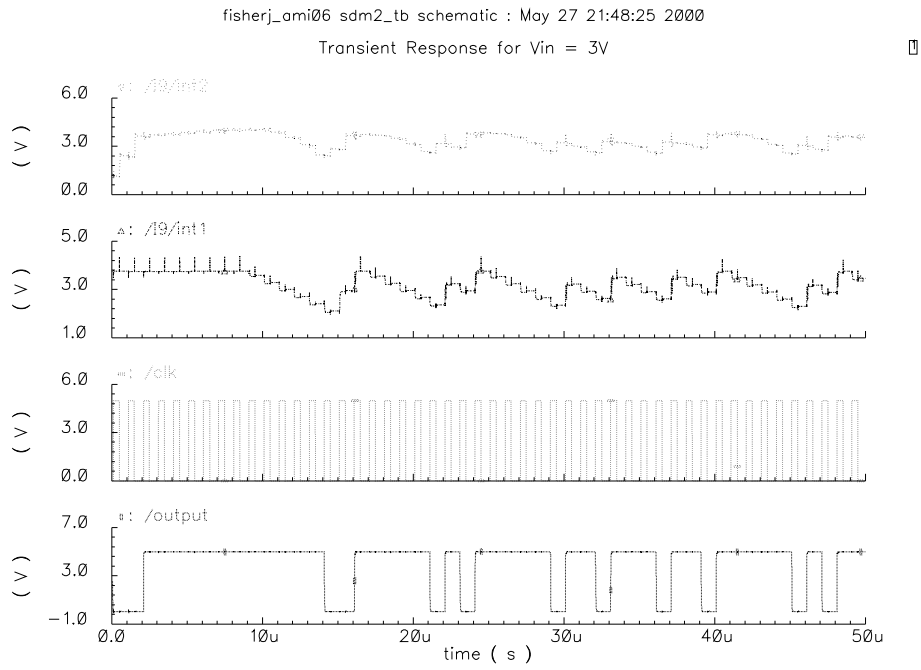


Figure 6.9: Sample Transient Simulation Results for the DSM2 Block.

received makes sense. While this process could be automated to report back to the design team what sorts of devices were recognized, foundries do extensive manual post-processing and seem to visually verify the design at that point. The best way to avoid problems in this situation is for the foundry to provide a design kit that is guaranteed to properly stream data in and out for their purposes.

While the designs were being re-fabricated, a topological problem was also discovered in the DSM ADC block. The coefficients used in the feedback path were not properly identified for their effect on the system performance. Examining the STF of the DSM in Equation 6.3 will show that a pole has been introduced within the space where quantization noise is usually substantially attenuated. Because of our oversampling

rate and sinc filter cut-off frequency, it should be possible to apply subsequent filtering to remove the effect of the misplaced pole, unless the pole induces saturation of integrators. Our simulations did not show any indications that any blocks would saturate, as the example simulation in Figure 6.9

The misplaced pole problem would not have occurred if our MATLAB® designs and our block diagram DSM specification had been properly synchronized. The DSM was implemented exactly as our hand-drawn block diagram showed but failed to match our design work in MATLAB®. We had used MATLAB® to evaluate various constructions for the DSM block and had used block diagrams to conceptualize our design implementations for Cadence®. MATLAB® offers a easy to access experimenting with a variety of design flows without the cumbersome overhead associated with an HDL.

While the DSM ADC block would need some rework to be reused in a future design, the other mixed signal blocks and sub-block could be directly used in future projects. And although the Nautilus designs have been carefully archived, they are not easily accessible to engineers who might be able to reuse them. The schematics and their test-benches have been saved, but the simulation set-up and resultant data is difficult to archive in Cadence®. Other designers may be able to find our designs but would have to entirely re-simulate them to evaluate their performance. If we had generated an initial AHDL model, then updated it with results from test, we would have created much more effective VC IP. The AHDL could contain all the relevant specifications for each block and could be easily added and simulated to other designs. With AHDL models for our blocks, designers creating highly complex systems

could use our AHDL model with their transistor-level schematics and/or their AHDL models, a mixed-mode simulator such as ADVancd-MS^{TM} could accept any of these formats and would expedite their initial simulations to establish functionality. If one of our blocks meets their functional needs, detailed simulations can be done with its transistor-level schematic counterpart substituted in for the AHDL model. Having AHDL models enhances reuse and acts as a form of documentation for the design.

CHAPTER 7

TOP-DOWN/BOTTOM-UP MIXED-SIGNAL SYSTEM DESIGN METHODOLOGY

In our envisioned mixed-signal design flow possible with the existing generation of EDA tools, a developed IP library of VCs would include a full set of characterized mixed-signal building blocks, such as switched-capacitor integrators, buffers, sample-and-hold circuits, bi-quad filters, analog switches, opamp gain stages, reference voltages, and bias currents. Also, each module would have various interchangeable views available, such as an AHDL model, a schematic, and a layout. And each block's views would have some parameters to be optimized for the desired performance, power, and die size requirements. While this methodology may not be able to deliver the optimal performance possible at a given process node, any digital system synthesized from basic logic gates makes this concession to be able to create complex systems in a narrow TTM window. We refer to this type of process as commodity system design, where the integration of a complex system is driven primarily by TTM with performance specifications safely within the constraints of the selected fabrication process.

Figure 7.1 shows the proposed top-down/bottom-up mixed-signal design flow. The intent of this design flow is that the analog work can developed with the same degree

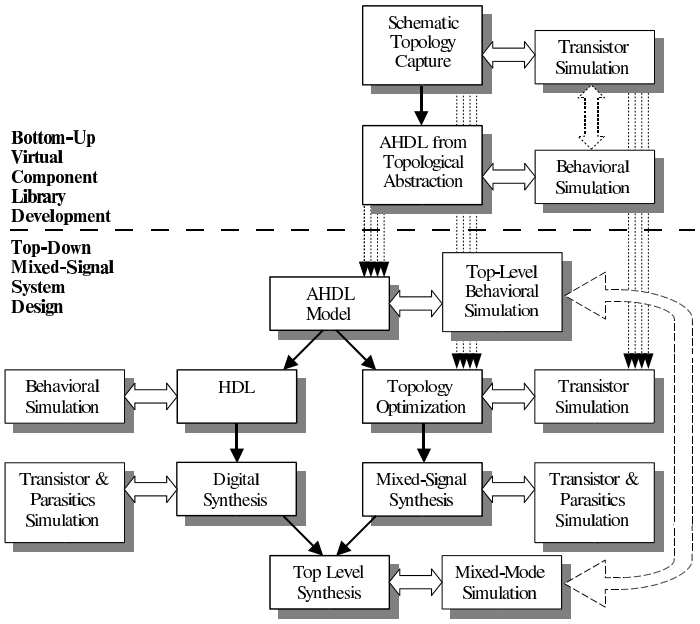


Figure 7.1: Top-Down/Bottom-Up Mixed-Signal System Design Flow Diagram.

of automation as digital design is presently, provided a sufficient library of VCs. So instead of initially developing a top-level behavioral model then lower-level AHDL modules, we create or procure the base modules first. The performance of the base modules is thus based on characterized layout and detailed schematic simulations.

In this chapter, we will discuss the use of virtual component libraries to the design flow, as well as how that enables the mixed-signal design flow to move closer the automation available in the digital design flow.

7.1 The Need for Virtual Component Libraries

There has been resistance within the design community to adopt AHDL modeling methods, or at least to limit its usage to a sub-domain of the design flow with the bulk of the design flow remaining in more traditional methods. We argue that a primary reason for this entrenchment has been a view of AHDL modeling as a method used in top-down-only design flows and, as such, there is a lack of IP libraries.

Without a library of VCs with which to begin, AHDL models are created as simple as possible at first, then more details and limitations are added as the project develops. One consequence is that design difficultly may get pushed into a less desirable point in the system.

For example, consider a system that will operate with a supply voltage that is known to be particularly noisy at low frequencies. It is simple enough to assert that each module in the design is required to have a certain minimal power-supply rejection ratio (PSRR) to handle the near DC noise source. If the system is primarily composed of ground referenced opamp-based blocks, the DC gain of the opamps would provide low frequency PSRR and suggest that design is safe. However, the bias voltage source module with its current mirrors will not have the same degree of PSRR (for a given design time, power, or die area budget). Plus, these bias voltages are set for a given supply voltage on the system. There is no margin from the supply voltage to add a regulator for just this bias block. Either a regulator must be created for this block and all of its clients, forcing all of its clients to adjust to the lower supply voltage, or the design of what was thought to be a simple, low impact block becomes a difficult

one and the linchpin of system performance. Depending on when this non-ideality is discovered in the design cycle, there may be no choice but to meet the difficult PSRR specification, which may slip the design schedule, expand the pad ring because of too much area for the module, and exceed the power budget. These consequences could have been avoided if the design difficultly was placed in a more tenable place

The top-down-only design flow forces the creation of a set of design specifications for the base modules before these modules have been implemented. A further consequence is that the developed performance metrics not only may be difficult but also may be mutually exclusive. For example, in the top-down-only design flow, a base module for a regulator might get a specification for a maximum load, a minimal PSRR, and a maximum output noise, which are each independently achievable but not mutually possible at a particular fabrication node, given a power and area budget.

The consequence of this initial idealistic view is that fundamentally important non-idealities do not express their impact until late in the design cycle. At which point, affected modules must be re-designed or the non-idealities must be minimized enough to eliminate their impact. Continuing our example, consider an AFE gain stage, which relies on a voltage source, which will actually be delivered by a regulator. If the initial AHDL model is an ideal voltage source, where noise, ripple, and PSRR are not considered, the assumed performance is not valid. Then late in the design cycle, these non-idealities are introduced and performance becomes inadequate. As mentioned, it is possible that the required performance metrics could combine to create an untenable design, either one not possible or more likely one that is possible when designer

are willing to pay unexpected design time, power, and die area penalties.

In order to avoid setting mutually exclusive performance metrics, a system designer would be required to have the understanding, insight, and experience necessary to design that base module. However, as aforementioned, analog gurus are a precious commodity that are most efficiently tasked to developing base schematic modules exclusively, and our intention is to create a tool flow where less experienced engineers are able to generate more of the content. The system design work needs to be decoupled from the development of VCs. Certainly, such libraries are not always going to be available or complete to begin a project, so some exception must be given to allow the system design to proceed in parallel with the library development. But even here, we would propose that the analog gurus generate placeholder AHDL modules in conjunction with an understanding of the needs of the system designers.

7.2 Developing Virtual Component Libraries

So analog gurus are responsible for the bottom-up virtual component library development shown in Figure 7.1. They create schematic topologies that can have their device sizes optimized to achieve as wide a range of performance as possible. They create a series of testbenches that capture these performance metrics and annotate in the tool which devices to tune to optimize which metrics. From these testbenches and annotations, the schematic optimization tool is able to generate an sized schematic to meet a set of specifications that fall within the range of those achievable. For example, the classic stage opamp is well understood for which devices can be sized to

achieve certain gain and bandwidth specifications [24, page 240] [32, page 372] [33, page 361]. The analog gurus also annotate the schematic topology with layout requirements, such that the layout automation tool is able to create a complete layout module with appropriate matching and parasitics. The schematic designer has always needed to indicate which devices need common-centroid matching and which nets are sensitive to parasitic capacitance. As discussed in Section 5.4, the developing EDA tools simply provide a uniform (per EDA tool) method of specifying these needs, along with an automated path to implement them.

The work outlined so far for the analog gurus comprises the top row of the design flow in Figure 7.1, specifically the schematic topology capture and transistor simulation blocks. The other major responsibility of the analog gurus is to abstract the structural schematic topology into a behavioral AHDL model. These model must be a drop-in replacement for the schematic. It is important for the ports to match between the views such that either view can be used in mixed-mode simulation. The AHDL model will have the topology's performance specifications as model parameters, each with a range (often dependent on the other parameters) that is achievable by the abstracted topology. This point is key. The system designer will be counting on the fact that if an opamp AHDL model allows the gain parameter to be set to $90dB$ and the bandwidth to be set to $1GHz$, then the abstracted topology can be optimized to meet both (and other) specifications. A code snippet of a latched comparator with module specifications as parameters is given:

```
module compLatch (q, qbar, clk1, vdd, vinm, vinp, vss);
```

```
  //--------------------------------
  // I/O ports and input parameters
  //--------------------------------

  // comparator hysterysis and offset
  parameter real param_vos = 5m from [2m:40m);
  parameter real param_vhyst = 100m from [20m:800m);

  // digital output delay, rise, and fall times
  parameter real param_tdel = 800p from [200p:8n];
  parameter real param_trise = 500p from (50p:4n];
  parameter real param_tfall = 500p from (50p:4n];

  ...
```

Here the hysteresis and offset voltage of a comparator are parameters of the AHDL model that can optimized in the schematic domain by adjusting transistor sizes within the developed topology.

An AHDL model could instead be created with schematic device sizes as parameters. However, this method voids the purpose of the topological abstraction. It voids the automation that the schematic topology optimization engine provide, removes the analog gurus' knowledge of regarding mapping behavioral to implementation, and forces the system designer to have the knowledge necessary to size the devices to achieve the actual desired performance metrics. Hence, we argue that the AHDL model parameters are the topology's key specifications. A code snippet of an input switch with device sizes as parameters is provided:

```
module inputSwitch (voutn, voutp, invert_sig, vdd, vinn, vinp, vss);

  //--------------------------------
  // I/O ports and input parameters
  //--------------------------------
  // PMOSFET width (m)
  parameter real param_sw_wp = 48u from (1.2u:1m);
  // PMOSFET length (m)
  parameter real param_sw_lp = 0.6u from (0.6u:20u);
  // NMOSFET width (m)
  parameter real param_sw_wn = 30u from (0:1m);
  // NMOSFET length (m)
```

```
parameter real param_sw_ln = 0.6u from (0.6u:20u);
...
```

The transistor simulation testbenches used for schematic topology capture will be reused for behavioral simulations of the AHDL model. As such, the behavioral model must capture the functionality of the structural schematic sufficiently that the performance metrics are the same when each module is simulated in the same testbench. Ideally, even the output waveforms would be identical, such that a designer could overlay the plots to visually understand the correspondence between the two or subtract the waveforms and find that the maximum difference is below a minimum threshold. This point is also important. Because the two views are in different domains the only validation that they model the same module is that one set of testbenches responds the same for both views. Getting the same results out of each view is the basis for validation.

With a set of testbenches that validate both the schematic topology and AHDL behavioral model, an annotated schematic for layout automation and for device size optimization based on performance metrics, and an AHDL model parameterized for these performance metrics, the module is ready for use within an VC library. As sea of modules is created and collected into the library and forms the bottom-up path of our envisioned design flow.

7.3 Applying Virtual Component Libraries to a Design Flow

With a VC library from which to work, the system designer develops upper-level AHDL models. The base AHDL models in the behavioral hierarchy are VC modules from the library. So the system designer partitions the design and forms a hierarchy until each base functional unit corresponds to a VC module. With a sufficiently rich VC library to develop most commodity projects, the system designer must then select the most appropriate VC module to meet the specifications a particular functional unit. Also, the testbenches for each level of hierarchy and the top-level AHDL model must be created and exercised by the system designer. These upper-level testbenches allow the VC module specifications to be determined and provide functional verification of the hierarchy and complete system. The VC module testbenches need to be run against the AHDL model from the library to validate that the output produces the expected functionality and is based on the specified parameters.

Following up on the Control CODEC example given in Figure 6.1, the system designer would first partition the digital sections from the analog one. As in many applications, the digital and analog interface is clean, so such a partition is possible. Certainly, if such a break where not clean, then the AHDL behavioral hierarchy would have several levels of mixed mode models. In the Control CODEC example, the buffers and DSP can be partitioned into a digital section, and the S/H, ADC, DAC, and T/H would form upper-levels in the analog hierarchy. As mentioned in Section 6.1.2, the DAC block required a opamp-based gain and level-shifting block for its output. So the DAC subsection of the design would be composed of its back-end and its actual hybrid thermocode/R-2R DAC [24, page 483], thus actually using at

least two modules from the VC library. The S/H subsection for a weak signal would need a low pass filter (LPF), signal buffer, then proper S/H, requiring at least three modules from the VC library. Certainly, the two design interface issues mentioned with this project would be discovered at the top-level AHDL behavioral simulation. And the DAC schematic from the VC library would have attributes embedded to ensure that the layout design did not compromise the necessary resistor matching.

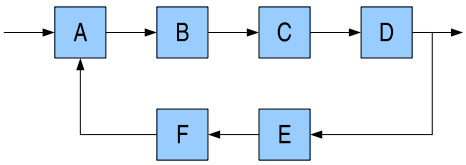
Simulation testbenches would be created for the S/H, ADC, DAC, and T/H subsections; analog and digital sections; and top-level. Through these testbenches, each VC module's parameters would be optimized to meet system performance requirements. It should be noted that blindly using the most aggressive set of specifications for any one block might meet the system performance requirements but will cost power and die area.

In fact, this trade-off is one of the primary reasons that the EDA companies developed schematic topology optimization engines. One of the classic examples is a pipelined ADC architecture, where each stage resolves $1.5bits$ and has an identical topology. However, the design requirements for noise relax for each stage in the pipeline, such that the first stage has the most stringent specifications [34]. The same primary opamp from the first stage could be used for all subsequent stages, but this would represent a substantial waste of power. So the same schematic topology is optimized to just meet the design requirements in each stage.

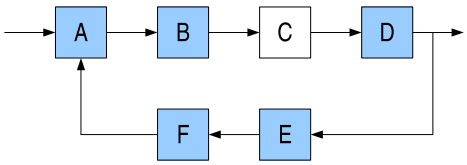
7.4 Integration, Optimization, Synthesis, and Validation

Once that model is complete with the base AHDL modules optimized for performance, implementation begins. The digital implementation follows a standard synthesis, place, route, back-annotate, and timing-closure path to develop its completed layout modules. During analog implementation, the corresponding base schematic topology is customized by an automated schematic optimization engine, as discussed in Section 5.4 and Section 7.2, to meet the AHDL module specifications. This customization may not simply involve the schematic to be optimized, as the performance of one analog block is a function of those around it in more than just timing. In analog design, loads play a critical role in the function and performance. For example, the load on an opamp will change it stability and slew rate. Thus, mixed-mode simulations will be used during optimization, with the base schematic view being simulated in conjunction with other AHDL modules. This method is also used to validate that the system specification are being met at the schematic level. The same testbenches developed for the top-level AHDL models can be run in mixed-mode simulation with various key sets of schematics replacing their AHDL counterparts.

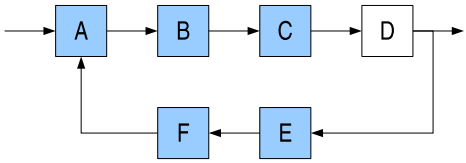
As an example, consider the generalized system shown in three views in Figure 7.2. This system could represent any number of systems that use feedback with clocked data, such as a PLL. Simulation time for clocked feedback system is much longer than strictly feed-forward systems, because each (non-trivial) module must reach a steady state operating point. This convergence takes a number of clock cycles, which is dependant on the loop transfer function. So even once every module has been translated from an AHDL model to an optimized schematic, the AHDL models will still



(a) All AHDL Models



(b) AHDL Models Except Module C



(c) AHDL Models Except Module D

Figure 7.2: Mixed-Mode Simulation Variations.

be used in a mixed mode simulation because simulating the entire feedback loop in a schematic-only simulation would take prohibitively long. Instead one module will be simulated as a schematic, while the others remain modeled in the AHDL. The same clocked feedback system testbench will be used for each simulation. So Figure 7.2(a) is intended to show simulating the entire clocked feedback system with AHDL models. Then Figure 7.2(b) shows simulating the system with AHDL models, except for module C, which is simulated with a schematic view. Likewise Figure 7.2(c) shows only module D using a schematic view, with the other modules using AHDL views. The latter two figures could represent simulating an PLL with all AHDL views, except for the voltage controlled oscillator (VCO), which uses a schematic view because of the required accuracy for this module.

Because simulating time climbs exponentially with linearly increasing design complexity, there are many clocked feedback systems where the sum of simulation time to run the same testbench for *each* module as a schematic (with the other modules using AHDL views) is actually less than running the same testbench for a single schematic-only simulation. Ideally, a schematic-only simulation of the top-level testbenches would be run, however this level of functional verification may not be worth the time impact, given its limited decrement of design risk with all of the other AHDL-only and mixed-mode simulations that have at that point validated the design. Certainly design integration is not possible with the schematic-only simulation. Often, design team will run one schematic-only simulation on the project due date as the other verification steps are running, just as a final check, even if the result will not be known for a number of days. This task puts the sense final closure on the functional verification.

Once the analog schematics have been optimized, the layout engine uses the tags in the schematic to generate the base module blocks. Assuming that the signal and supply nets are also appropriately tagged, the engine can also layout the hierarchy, even the pad ring. Certainly, as in the digital domain, some work may be required to guide the hierarchical layout appropriately. With the layout complete, parasitic extracted simulations validate that the design has been implemented in layout as designed in AHDL. Again, a layout-only simulation at the top-level may not offer enough risk reduction to be worth the time impact and, in fact, may not be possible with existing computer resources. Certainly, each block needs to have its base level testbenches from the VC library run against the layout extracted netlist of that module to complete the base module functional verification. Additionally, key performance blocks

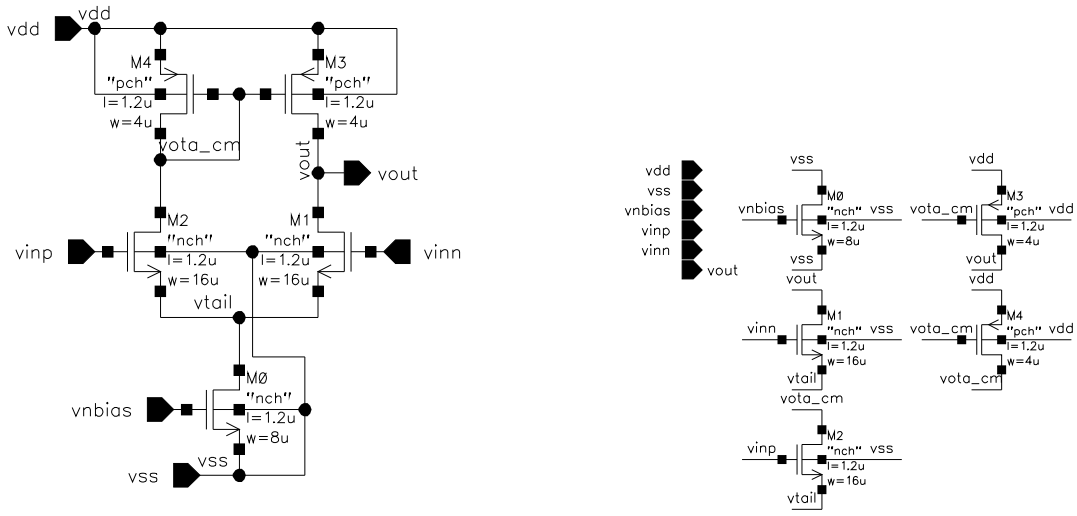
should be inserted into the mixed-mode simulation environment as parasitic extracted netlists and run with the top-level testbenches, then evaluated against the original AHDL-only top-level simulations. This final set of steps closes the loop on the design cycle to mitigate risk as much as design time allows.

7.5 Advantages and Limitations

In this methodology, the development of libraries of VCs is an essential task, as the extent of these libraries bounds the possible system design space. As discussed in Section 7.2, each schematic topology has its own AHDL model, with model parameters corresponding to those available to the schematic optimization engine. The AHDL model would then have the same set of parameters available to be adjusted to achieve optimal system performance when simulating with the upper-level system hierarchy and top-level AHDL models. The schematic topology would also have its devices tagged as appropriate such that an automated engine could synthesize a layout view.

If the topological schematic optimization characteristics and the AHDL model are created with reuse in mind, both the schematic and AHDL model could be re-targeted without rework to a different fabrication process. In fact, the AHDL model can often be sufficiently generalized that with only minimal rework it can be ported to an entirely different topology.

By offering AHDL modeling, topology optimization, and layout automation, the present generation of EDA tools are an evolutionary step forward from the previous



(a) Standard Wiring

(b) Nets Named Only

Figure 7.3: Functionally Equivalent OTA Schematics.

generation in terms of abstraction and documentation. Specifically, schematics and structural models are both an abstraction of the fabricated design and a fundamental form of documentation. While reams of text and figures can be used to describe a design, the design itself cannot be recreated. Even with a netlist dump, the position and relation of the devices in the schematic is important documentation to the design.

A differential pair is drawn in a specific standardized manner, including where the device sizes and models name are placed. Figure 7.3 shows two functionally and logically equivalent schematics of an operational transconductance amplifier (OTA), and the following code snippet is also equivalent:

```
MM4 vota_cm vota_cm vdd vdd PM W=4u L=1.2u M=1
MM3 vout vota_cm vdd vdd PM W=4u L=1.2u M=1
MM2 vota_cm vinp vtail vss NM W=16u L=1.2u M=1
MM1 vout vinn vtail vss NM W=16u L=1.2u M=1
```

```
MM0 vss vnbias vss vss NM W=8u L=1.2u M=1
```

Figure 7.3(a) is known even to undergraduate circuits students as an OTA stage. In contrast, even the most experience analog gurus would have to ponder and map Figure 7.3(b) or the code snippet in order to divine that it is a simple OTA stage. The placement of devices on the sheets has a standardized understanding for functionality and must be treated with great care. Given the testbench schematics and an understanding of the functionality, any external documentation could be recreated, eventually. By adding AHDL modeling, there is an obvious behavioral abstraction available, but there is also a level of documentation provided in this abstraction. The relevant functionality of the block is captured in the behavioral format, which documents the module.

The topology optimization is another major step in the development of EDA tools. The ability to move beyond just developing and archiving testbenches and writing text to understand a schematic is a serious advancement. In the previous generation of EDA tools, absent the original designer, the schematic and testbenches (if archived) were often all that was available to divine the design intent of a schematic. Anyone who has ever been handed a previously unseen schematic on a C-size sheet and been asked to become responsible for it can appreciate the intent of the topology optimization. Not only are all relevant performance measurements and testbenches captured by the tool, but also the designer's intent must be documented for the optimization to succeed. Specifically, once a simulation has been run, nodes measured, and performance metrics calculated, the engine must be given a methodology for varying

specific device sizes to allow the performance metrics to meet specification. Thus, the original designer's understanding of how performance is affected by the schematic choices is documented.

So if we reconsider the OTA schematic from Figure 7.3(a), the (approximate) gain is well known [33, page 152] and given in Equation 7.1.

$$|A_v| = g_{m1,2}\left(\left(r_{o1,2} + \frac{1}{2g_{m4}} \parallel r_{o4}\right) \parallel r_{o3}\right) \tag{7.1}$$

If we assume that the devices are designed to be in saturation, then the transconductance is given by Equation 7.2, and the output resistance is given by Equation 7.3.

$$g_m = \sqrt{\frac{2\mu C_{ox}\frac{W}{L} I_D}{1 + \lambda V_{DS}}} \tag{7.2}$$

$$r_o = \frac{1}{\lambda I_D} \tag{7.3}$$

Thus $|A_v|$ is approximately inversely proportional to the square-root of the drain current. This relation would suggest that gain could be arbitrarily increased by decreasing the tail current, which would of course save power as well. However, there are other parameters that are dependant on the drain current. Equation 7.4 [33, page 239] gives the noise power of the OTA.

$$\overline{V_{n,in}^2} = 8kT\left(\frac{2}{3g_{m1}} + \frac{2g_{m3}}{3g_{m1}}\right) + \frac{2K_N}{C_{ox}(WL)_1 f} + \frac{2K_P}{C_{ox}(WL)_3 f}\frac{g_{m3}^2}{g_{m1}^2} \tag{7.4}$$

In the noise equation, the thermal noise power is also inversely proportional to the square root of the drain current, except that increasing noise is rarely desirable. And

the slew rate is proportional to the drain current. Thus, there is a performance balance to be achieved in decreasing the size of the $M0$ transistor that controls the tail current. Increasing differential pair size will increase the transconductance (until the device goes into subthreshold) and decrease the flicker noise, but costs area. Increasing the load pair size increases output swing but decreases output impedance. Using these exacting equations is one methodology to design device sizes based on performance metrics. Barcelona Design® uses this methodology for blocks far more complicated than a simple OTA. Plus they use more complicated equations that include secondary and tertiary effects, such that they can resolve exact device sizes for precise performance specifications. Without these higher order effects, the equations are just approximate and still need iteration to fully optimize the sizing.

Of course, many analog gurus will have the experience necessary to understand which devices they would choose to leave a fixed size and which devices they would vary to achieve the desired performance. Regardless of the method of understanding how to map specification to devices to be sized, it is key to capture this information in the tool. This annotation provides a profoundly new level of documentation in analog design.

The layout automation tool also provides a level of documentation. Until this tool's development, ad-hoc methods of documenting layout intent have been used irregularly in schematics. For example, matched devices often get circled. However, this tool allows tags to be applied to sets of devices to indicate how they should be handled. Since this information, combined with the fabrication process information, is all the

tool uses to generate the actual layout, then this information can also be properly considered as a model or abstraction of the layout. That model documents all of the designer's intent for the layout design.

One issue with this tool as implemented by Cadence®, and the topology optimization as well, is that there is neither a standardized nor even intuitive visualization of these tags. Perhaps alternate schematic views will develop over time in the industry to give a visualization page for these tools' tags. A text file summary of these tags is cumbersome because it references the devices by instance name. For example, in order to ensure that the input differential pair is going to get common centroid layout by reading the text summary of layout tags, the designer would have to correlate the schematic instance names with those in the (lengthy) text file. However, these improvements will develop as these new tools mature into the design flow.

Using these developing EDA tools with our envisioned design flow methodology shown in Figure 7.1 does constitute a major step forward in system design. Designs can be abstracted to allow for a partition of resources between module development and system design. With performance abstracted to the specification domain by the analog gurus, the system designer is able to develop more complex systems with less detailed knowledge of schematic design. However, there are still issues with this methodology. Firstly, the system designer may be presented with a number of options with which to implement an individual module. Without a deeper understanding of the analog design behind these modules, the decision as to which one to use is arbitrary. However, the optimal choice may be entirely subjective and apparent to

the analog gurus. Additionally, as the VC library develops, there will develop a vast sea of modules. Unlike digital leaf cells where there are a known finite number of functions (and hence possible leaf cells) of any given drive strength and number of inputs, the range of analog functional blocks is unbounded. Just managing this sea of modules is a difficult task. To address these concerns and to expand on this envisioned methodology, we will present our proposed methodology which uses our novel translation engine to manage the translation from specification to topology in Chapter 8.

CHAPTER 8

MIXED-SIGNAL SYSTEM DESIGN IN AN MDA PROCESS

In this chapter, we will discuss our proposed introduction of an MDA process into mixed-signal system design to develop an automated design flow from specification to layout.

8.1 Proposed Mixed-Signal Design Methodology

Our proposed design flow would expand the top-down/bottom-up design flow beyond the capacity of existing EDA toolsets. We have created a novel EDA tool which provides a framework for abstracting the topological selection, such that system design can be done at a specification level without the same degree of in depth analog knowledge required to translate a set of detailed system specifications into customized hierarchical topology. To create this abstraction, we apply an MDA methodology to decouple specifying the design from topological implementation.

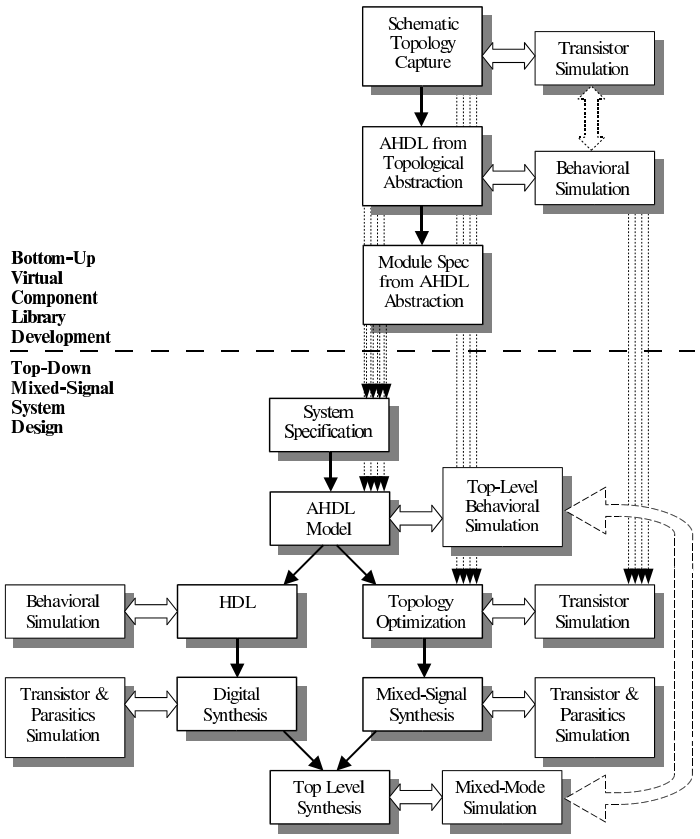


Figure 8.1: MDA Methodology Mixed-Signal System Design Flow Diagram.

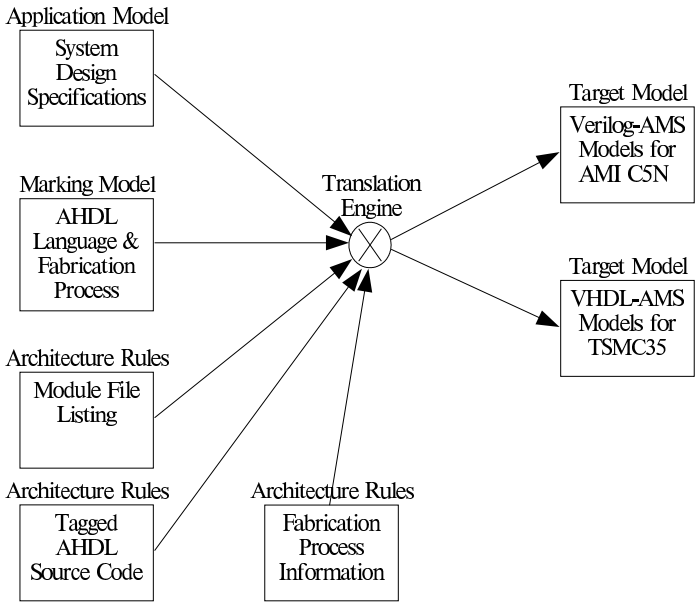


Figure 8.2: Model Driven Architecture Block Diagram for Mixed-Signal System Design.

As such, the system designers using this methodology are less concerned with the exact implementation developed and more focused on the functionality and specifications. For a commodity mixed-signal system, the system designer who wants a system with an analog-to-digital converter (ADC) often does not care what sort of ADC is used, so long as it meets various criteria. While those criterion might suggest one family of converters, the designer does not care which exact topology is used within that family, as long as the specifications are met. So while a designer might know that the ADC specifications imply some sort of delta-sigma modulator (DSM) ADC would be appropriate, whether a DSM211 or a DSM22 is used is relatively unimportant.

In this implementation of MDA methodology, experts who know how the system specifications translate into an exact topology create the architecture rules, as well as VC

libraries of characterized mixed-signal modules that the engine uses to translate the system designer's application model into the target models. The application model is the system specification, the abstracted model. The target models are detailed AHDL models. Of these target models, the base building block modules are behavioral and based on characterization data from schematic/layout extracted simulations and silicon testing. The other modules are structural and form the hierarchy of the system. The system designer also specifies a marking model to dictate what code language is used for the target models, as well as which fabrication process will be used. Our implementation uses VerilogAMS code as the target model customized for the AMI C5N fabrication process. Figure 8.2 shows a block diagram of relationship of the models involved in this MDA process. It is especially important to notice that the system designer is isolated from the development of the implementation methods.

8.2 The Design Space: Explore Rapidly and Develop More

This separation allows the system designer to explore the space of specification options, without being encumbered by the details of exact implementation. In Section 6.1, we discussed the design of a Control CODEC, as an application of a generalized heterogenous CODEC. A system designer for such a project would want to be able to select and evaluate any possible set of sensors and actuators, without being encumbered by the exact circuit implementation to form these choices into a mixed-siganl control loop. The focus when exploring this design space is most effectively spent considering the sensors, actuators, and control algorithms that are the core of the the control loop. In our proposed method, the system designer can use the sensor

and actuator electronic interface specifications as the specifications for the analog sections of the Control CODEC. Then the tool uses these choices to automatically select the implementation, the exact VC module with a schematic topology and AHDL model. The focus can remain at the specification level for the analog sections, allowing more time to consider the optimal set of control elements and their digital algorithms.

Likewise in Section 6.3, we discussed a transducer-to-digital-pico-network system. We indicated that the transducer phenomenology would not be well characterized until late in the design process and that the choice of digitial pico-network interface standard was not fixed. So for an application in this project, the focus when initially exploring the design space will be more on the digital work. The choice of a communication standard for the pico-network can be given full focus within the same framework. Digital VCs may not have the same design content as analog ones, but our novel tool is still applicable. If we indicate that one of the modules within our design is an SPI client with some set of specification, the tool is unbiased that these specifications and the resultant VC library entry are digital. The topological selection simply maps specifications to a VC module, so the process is unchanged. In this manner, any variety of communication protocols can be considered within the system design space.

Further, the analog front-end must be able to adapt to changes in our understanding of the transducer phenomenology, as the project develops. The TTM penalty for changing these specifications is simply to rerun the automated implementation path. In the top-down/bottom-up methodology developed in Chapter 7, changes in the design specification could be tolerated to a certain point where the selected topology is

no longer valid. At which point, a new topology must be selected manually. More importantly, while a previously topology might be made to work by the schematic optimization engine, the result may be a seriously sub-optimal design. In contrast, in our proposed methodology, the automated topological selection is rerun when the specification changes, to ensure the the optimal module is used. The focus is to spend as much design effort as possible in the design space and minimze the implementation time while decreasing TTM.

In addition, the design space can be explored more quickly, since the target models are automatically configured and connected appropriately. This feature allows specifications to be validated or shown to be impossible or in conflict sooner in the design process. So for our pico-network system, we can validate that a continuous time sensor transducer signal's integrity will be maintained through the AFE, ADC, decimation, and communication standard chosen. The AHDL and HDL models enable a rapid validation of the functionality beyond the raw specifications of a single module.

Similarly, the Control CODEC replaces an analog control loop, so we can quickly find loop stability issues. Suppose that a particular combination of a sensor, an AFE, an ADC, a digital gain, a DAC, a T/H, an actuator, and the physical plant (the reality being actuated) form an underdamped loop due to the latency through the loop. The initial system level simulations would illuminate the issue, allowing the offending blocks to be re-specified with a consideration for latency. The topological selection can then be rerun and the fix verified.

Additionally, the automatic translation helps find conceptual design errors and avoid manual translation errors. And the mixed-signal design experts knowledge is shared and reused more effectively across many projects through the libraries of modules. In addition, the application model can continue to develop through a much later point in the design specification cycle, since the application model is automatically translated into the target models. This is not only an important feature in designs like the pico-network example where the transducer phenomonology is not well understood initially, but also in general projects where the initial specification are not frozen. Despite designer's best intentions, "feature creep" occurs in projects of every size. Designers have been known to quip that new features only stop being added once the tape-out date passes and that the introduction of these new feature late in the process causes the creation of a design resembling Frankenstein's monster. Late design additions must be grafted onto the existing implementation, often in less than ideal manners.

The last minute addition of the DAC backend buffer on the Control CODEC project discussed in Section 6.1.2 created a less than ideal signal path in the layout design, as well as introducing unnecessary complication in the system. The DAC reference voltages could have been easily chosen such that the output buffer design would have been far less complicated and shared one such reference voltage. While the need for this buffer is not strictly an example of feature creep, it does illuminate some of the results for feature creep additions. So a process that encourages specification development until a later stage in the design process and produces equally graceful results is clear advantage for complex system design. This advantage is leveraged by the

automated implementation path from specification to layout.

8.3 Moving Towards Implementation

In our model, the VerilogAMS system is a first pass at the design appropriate for the system. Various parameters are provided with which to optimize the design. And the base target models are designed to be exactly matched to schematic modules in the library, such that once the VerilogAMS code is optimized, the schematic devices can be optimized to meet the VerilogAMS code specifications and form a complete schematic set. In addition, target models and schematics can be co-simulated with sensitive blocks in schematic form and insensitive blocks in VerilogAMS form to balance simulation time and required accuracy, as developed in Section 7.4 with Figure 7.2.

In our MDA implementation, we do not consider the issues of parameter optimization or mixed-signal layout automation, as others have made significant contributions there already, such as Neolinear® and Barcelona Design®. However it is worth noting that mixed-signal layout automation methods can be considered as another MDA example. In every sense, a schematic is an application model–an executable model that abstracts the layout design. The designer applies tags to various sets of elements to indicate how they need to be related in the layout, such as common-centroid techniques. These tags are the marking model. And the commercial tools use foundry rules and programmed understanding of layout techniques as the architecture rules

to feed into the tool's translation engine to produce completed layout.

As discussed with Figure 8.2, the set of system specifications is the application model in our implementation. The marking model is a set of specifications on which format of AHDL code will be generated for which EDA tool and for which fabrication process. The architecture rules are composed of three parts. The first is a set of rules for how the system specifications translate into module family and topology selections, followed by rules on which modules are to be included in the design for each possible topology selection. The second is a set of rules from the foundry, regarding the fabrication process. And the third is a library of tagged AHDL modules with the behavior and much characterization data already included.

The translation engine uses the application model specifications and the architecture rules on family and topology selections to decide which topology is needed. For example, an ADC application model specification of signal-to-noise ratio (SNR) greater than 60dB with a maximum clock frequency of no more than 16 times the maximum input bandwidth would imply at least a fourth order DSM (such as a DSM211 or a DSM22) is required. Then the rules on the required modules for the selected topology are used to generate a list of tagged source modules to parse. Continuing our implementation, we will need modules for switched-capacitor integrators, comparators, buffers, and others, as well as the hierarchy to connect these base modules together. Source modules may be used repeatedly with different parameters to produce distinct target models. In a DSM22, the second integrator in each stage has only two inputs,

while the first input in the second stage has three inputs. In addition, the first integrator's non-idealities have a dominant effect on system performance relative to the other integrators, so it has more of its possible noise sources modeled than the other integrators. However, we are able to use the same tagged source module for all of the target model integrators. Some of the tags in this integrator source module, as well as in the other base modules, will be processed by the translation engine using data from the fabrication rules. For example, the gate-oxide capacitance causes a charge injection effect on the integrator when the switched-capacitor switches open and that effect can be modeled in the integrator module with the fabrication process parameter for *cox* as a value inserted into the target module by the translation engine at the appropriate source module tag. The other tags are parsed with system information on which target module is being generated for which system topology. The second integrator in each stage of a DSM22 has only two inputs, so the third available input in the tagged source integrator module will be eliminated in the target model.

8.4 Encouraging Design Reuse Through Content Management

As in the PDK MDA implementation, using an MDA process for mixed-signal system design encourages design reuse. While the process of creating characterized VCs with various views is intended to provide the ability to reuse design effort, the MDA process both gives those modules a context and helps to manage the libraries of information. With a fully developed MDA process, the VCs are no longer a sea of available topologies; there is a selection process coded into the architecture rules that shows the intent of the analog gurus that created a new topology, when assumedly an existing

one was not sufficient. With an automated translation from specification to topology, the most appropriate module is always reused in the development a new system. So not only are the VCs designed to be reused, but also there is documentation of how functional design specifications map to a topological selection. The MDA process allows for the understanding of this next level of abstraction to be documented by the analog guru. Existing tools have finally given the ability to document how a schematic topology relates to its performance, however the ability to document how a functional specification relates to a choice of topologies has to date been at best unstructured.

Continuing our Control CODEC example, there are a large variety of S/H topologies available to say nothing of the number of topologies available for a commidity ADC. A developed VC library may contain at least one of implementation of each topology. Keeping track of which modules are available and appropriate can become a non-trivial task that introduces an element of manual error. Further, a number of VC modules may meet the basic S/H specifications, such as bandwidth, linearity, and noise. However, one of the options might incur an unintended penalty beyond design specifictions, such as the use of a deep NWELL layer. Of the options available, this design might yield the best possible performance for no die area or power penalty. However, another topology might be sufficient to meet the specifications without requiring the special fabrication layer and hence increasing the cost of every die in manufacturing. The topological selection algorithm would be written to only use such a specialized topology where the specifications demand it.

Additionally, the MDA process helps to manage the project and library data. Beyond aiding reuse, the context that the process gives the sea of modules provides an organizational framework. The rules indicate exactly which modules are associated with which topology. It also structures which testbenches are relevant for understanding the performance of the designated topology. These testbenches are developed at both the sub-module and super-module levels. So once again, the MDA process aids in a process of documentation, which would otherwise be unstructured or non-existent.

When the DAC in the Control CODEC example is selected, it is not just the DAC schematic that is added to the design library from the VC library. The DAC is composed of hierarchy. The resistor ladder, the opamp, and the hierarchy all require AHDL models and schematics. Additionally, there are a slew of testbenches that get added to the design library for this particular DAC application. Outside of any testbenches to validate the entire DAC super-module, the opamp would have several distinct testbenches for AC gain and bandwidth, DC range, slew rate, PSRR, and start-up transients. A VC module is not just a schematic and its AHDL module; each module has a whole slew of ancillary files that must accompany it.

CHAPTER 9

AN MDA PROCESS MIXED-SIGNAL SYSTEM DESIGN EXAMPLE

In this chapter, we will explore a design example of using our MDA process for mixed-signal system design.

9.1 A Conceptual System Design Example

As a conceptual example of a complex mixed-signal system, we offer the heterodyne transceiver shown in Figure 9.1. The high speed input is an RF signal which is amplified, filtered, then down-converted to a filtered intermediate frequency (IF) signal. A second down-conversion stage amplifies, filters, down-converts, and filters to a baseband signal. This baseband signal is then digitized and processed by the digital signal processing (DSP). The high speed output chain is simply the reverse process. Additionally a baseband path is shown that converts an amplified baseband sensor signal to digital, as well as driving a baseband actuator, such as a speaker or servo. This familiar structure and its many variants [35] have application in systems that communicate wirelessly with any network.

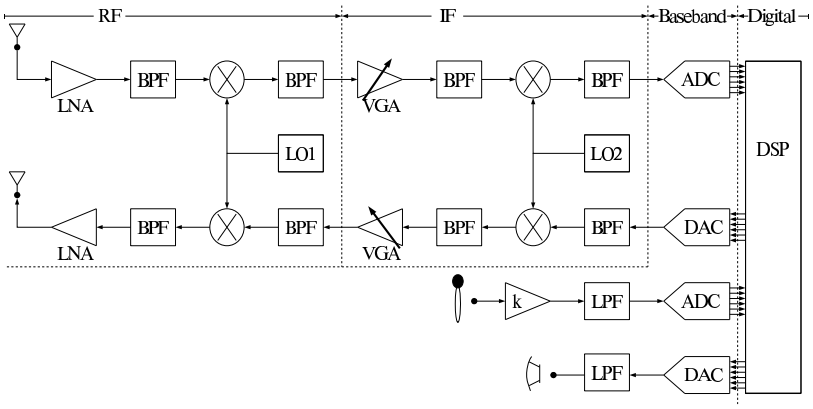


Figure 9.1: Generalized Transceiver System Block Diagram.

Our MDA process allows for the transceiver's specification to be varied to evaluate design trade-offs and immediately simulated at a base module level. For example, in a frequency division multiple access (FDMA) application channel selection can be done either digitally after the ADC or with the second down-conversion stage by adjusting the phase local loop (PLL) generating the second local oscillator (LO) signal. Digital channel selection is a straight-forward problem to solve, but requires that the digital logic and the ADC operate at a frequency high enough to accommodate the bandwidth of all of the channels, instead of just one channel. Adjusting the LO frequency with the require accuracy is also not a trivial problem. However, in our MDA process, only a few specifications would need to be modified to cause the translation engine to select the appropriate topologies and generate the full set of VerilogAMS modules for each scenario. The space of these trade-offs are fundamental to the project and can be explored more quickly in our MDA process.

Additionally, the application model could be ported to an entirely new project with a different wireless standard specification. So if this process was used to create a $900MHz$ handset, the same application model could be used to port the design to the $2GHz$ range. Each high speed block would needs a few specifications adjusted, but then the translation could be rerun and a new full set of modules generated. Again, the engine would select the most approriate topologies for the new specifications.

9.2 Focusing on a DSM ADC VC

The creation of the VC libraries and architecture rules necessary to synthesize the entire transceiver would require a few man-years of effort, so for the purposes of this work we have focused our attention the ADC modules within the design, and particularly DSM ADCs. We chose DSM ADCs because they are a super-module composed of integrators, comparators (for single level quantization), single-bit digital-to-analog converters (DACs), reference voltages, voltage buffers, current biases, and clock generators. Further, ADCs in general, and DSM ADCs in particular, represent a challenge with certain advantages for behavioral modeling. As discussed in Section 7.4, clocked feedback systems are difficult to develop when confined to being able to simulate in a schematic-only environment. Being able to isolate individual modules for schematic development and simulate with the other blocks as AHDL modles as shown in Figure 7.2 helps decrease TTM. Additionally, each sub-module of a DSM ADC has complexities of its own. Unlike the PLL example, none of these sub-modules within the feedback loop can be accurately represented with a simple digital function or a

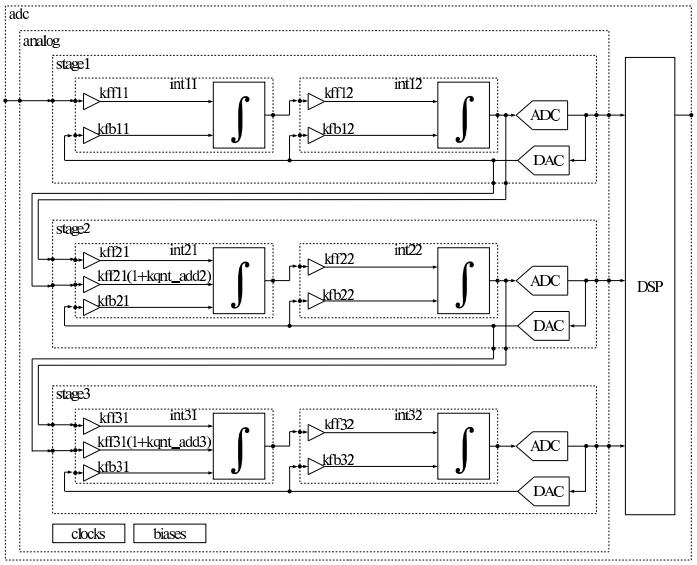


Figure 9.2: Generalized Sixth-Order Cascaded DSM (DSM222) ADC Block Diagram.

single LaPlace function. Each module has analog behavior, although our primary focus is on the integrators' behavior and their non-idealities. Given our single-bit quantization, the integrators' non-idealities will be the dominant ones of all the sub-modules.

Each of these sub-modules could be an independent module in our MDA process as well. Figure 9.2 shows a DSM222 block diagram, where three stages of second-order DSM ADCs are cascaded, and their outputs combined by digital error correction (DEC) logic in the digital signal processing (DSP) block. The effect is that each subsequent stage's effective input is the quantization error from the previous stage. The DEC logic then subtracts the subsequent stage's output from the previous stage's output. The output of this DEC stage is the previous stage's digitized input signal

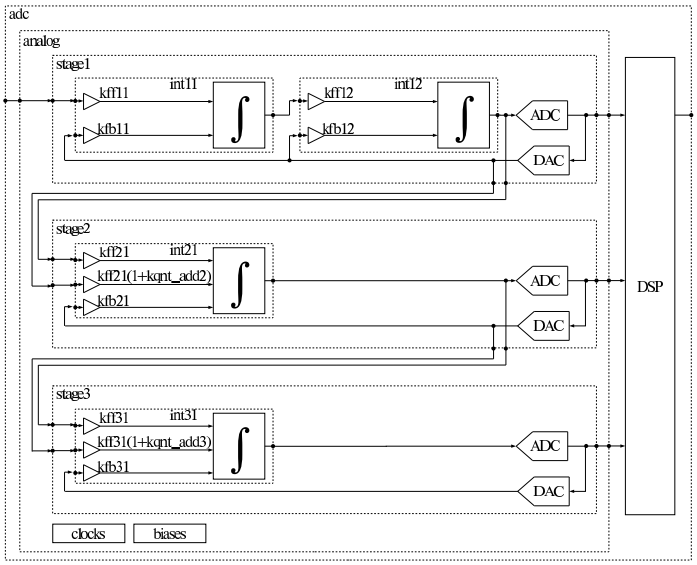


Figure 9.3: Generalized Fourth-Order Cascaded DSM (DSM211) ADC Block Diagram.

and the subsequent stage's lower quantization noise.

The DSM1, DSM2, DSM21, DSM22, DSM211, DSM221, and DSM222 topologies are developed in this work. Figure 9.3 shows a DSM211 topology, which cascaded two first order DSM ADCs after a single second order DSM ADC. It is important to notice that the only difference between the two topologies are two fewer integrators in the DSM211. Certainly, the DSM222 is the superset of the designs developed for this work.

9.3 DSM ADC Theory and Modeling

By superposition, the z-domain output of a DSM ADC ($Y_{DSM}(z)$) is given by Equation 9.1, where we combine the signal transfer function (STF) and the noise transfer function (NTF) with the inputs ($X(z)$) and the quantization error ($E(z)$), respectively. The quantization error is of course introduced by the ADC in the loop. Another error could be introduced by the feedback DAC in the loop, if we were using multi-bit quantization. Since we are using single-bit quantization, the linearity of the transfer is guarenteed, so the only noise that we will add to the base z-domain analysis is the quantization noise. A generalized STF equation for the DSM2 as shown in the first stage of Figures 9.3 and 9.2 is given by Equation 9.2. Likewise, the corresponding NTF equation is given by Equation 9.3. In order to eliminate the poles from the STF and NTF, the denominator of each equation much be equal to one, as indicated by Equation 9.4. Consequently, our choices for the feedback gain coefficients are limited, as given by Equations 9.5 and 9.6. With these limits, we can find the general z-domain output of DSM2 in Equation 9.7, where kff is the product of the feedforward gain coefficients. A similar analysis for a DSM1 yields is output in Equation 9.8. In both cases, the quantization noise has been moved out of the bandwidth of interest (baseband in our case) by the noise shaping properties of the DSM ADC.

$$Y_{DSM}(z) = STF_{DSM}(z) \cdot X(z) + NTF_{DSM}(z) \cdot E(z) \tag{9.1}$$

$$STF_{DSM2}(z) = \frac{kff11 \cdot kff12 \cdot z^{-2}}{(1 - kfb1 \cdot kff12 + kfb2)z^{-2} + (-2 - kfb2)z^{-1} + 1} \tag{9.2}$$

$$NTF_{DSM2}(z) = \frac{(1-z^{-1})^2}{(1-kfb1\cdot kff12+kfb2)z^{-2}+(-2-kfb2)z^{-1}+1} \tag{9.3}$$

$$(1-kfb1\cdot kff12+kfb2)z^{-2}+(-2-kfb2)z^{-1}+1=1 \tag{9.4}$$

$$kfb2=-2 \tag{9.5}$$

$$kfb1=-\frac{1}{kff12} \tag{9.6}$$

$$Y_{DSM2}(z) = kff\cdot z^{-2}X(z)+(1-z^{-1})^2E(z) \tag{9.7}$$

$$Y_{DSM1}(z) = kff\cdot z^{-1}X(z)+(1-z^{-1})\cdot E(z) \tag{9.8}$$

Of course, the DSM ADC designs shown in Figures 9.3 and 9.2 are cascaded. The input of the second stage can be the difference between first stage's output before quantization ($Y_1 - E_1$) and after quantization (Y_1), which would thus be just the quantization noise. The output of the second stage is then the non-shaped quantization noise of the first stage (E_1) and the shaped quantization noise of the second stage($(1-z^{-1})E_2$). We can then digitally apply the same noise shaping to the output of the second stage (Y_2) as the quantization noise in the first stage received ($(1-z^{-1})^2$) and add it from the output of the first stage. The result of the addition is that the all terms containing E_1 are eliminated, the already nosie shaped E_2 receives a higher order shaping, and the digitized input is unchanged. In effect, cascaded stages simply remove the previous stage's shaped noise and replace it with their own, which gets a

higher order of noise shaping.

In theory, this technique effectively achieves the same performance as a single stage DSM whose filter order is the sum of all filter orders in all of the cascaded stages. Each of the integrators in our baseband design has a filter order of one, so the DSM211 achieves the performance of a DSM4. This analysis does assume that each stage uses the same number of quantization bits, which is by no means required. While this technique does achieve higher order DSM performance without suffering the serious stability issues of third order or greater single stage DSM ADCs, there are issues with the topology. The most serious is that the analog filters must be matched by the digital error correction to completely eliminate the quantization noise of stages before the last one. Since the analog filter will not perfectly match the digital one, there will be some residual quantization noise from ealier stages to degrade performance.

Performance can be further degraded by filter saturation. The gain coefficients are used to prevent this issue. In particular, the $kff21$ coefficient is used to scale $E_1(z)$ for the second stage. Additionally, there is a technique [36, page 225] to add $Y_1(z)$ to the input of the second stage times a gain coefficient $kqnt\_add2$. This addition helps to spread out the input voltages of the second stage across the valid range, and can improve performance. As such, the second stage input is given by Equation 9.9. The same technique cannot be as easily used on the third stage because the digital correction becomes more complicated. Consequently, $kqnt\_add3 = 0$, as implied by Equation 9.10.

$$X_{stage2}(z) = -E_1(z) + kqnt\_add2 \cdot Y_{stage1}(z) \tag{9.9}$$

$$X_{stage3}(z) = -E_2(z) \tag{9.10}$$

Having established the DSM2 and DSM1 output equations and the inputs to each stage, we can express the digital error correction filtering necessary to combine the individual stage outputs. Equation 9.11 shows the filters applied to the output of the first two stages to create a DSM21 output. This output of course still requires filtering to create proper digital words. The DSM211 output is created by combining the DSM21 output with the third stage output, and its filtering is given by Equation 9.12. These two equations give the filters necessary to do the digital error correction.

$$\begin{aligned} Y_{DSM21}(z) &= (z^{-1} - kqnt\_add2 \cdot z^{-1}(1-z^{-1})^2)Y_{stage1}(z) \\ &\quad + \frac{1}{kff21}(1-z^{-1})^2 Y_{stage2}(z) \end{aligned} \tag{9.11}$$

$$Y_{DSM211}(z) = z^{-1}Y_{DSM21}(z) + \frac{1}{kff21 \cdot kff31}(1-z^{-1})^3 Y_{stage3}(z) \tag{9.12}$$

Solving these equations, we find see in Equation 9.13 that the DSM21 has the same performance as a DSM3. Likewise, Equation 9.14 indicates that the DSM211 has the same performance as a DSM4.

$$Y_{DSM21}(z) = kff11 \cdot kff12 \cdot z^{-3}X(z) + \frac{1}{kff21}(1-z^{-1})^3 E_2(z) \tag{9.13}$$

$$Y_{DSM211}(z) = kff11 \cdot kff12 \cdot z^{-4}X(z) + \frac{1}{kff21 \cdot kff31}(1-z^{-1})^4 E_3(z) \tag{9.14}$$

A similar analysis can be performed for a DSM222 architecture. Equations 9.15 and 9.16 show the filters necessary for digital error correction of a DSM222 ADC. Solving these equations yields Equations 9.17 and 9.18.

$$\begin{aligned} Y_{DSM22}(z) &= (z^{-2} - kqnt\_add2 \cdot z^{-2}(1-z^{-1})^2)Y_{stage1}(z) \\ &\quad + \frac{1}{kff21 \cdot kff22}(1-z^{-1})^2 Y_{stage2}(z) \end{aligned} \tag{9.15}$$

$$\begin{aligned} Y_{DSM222}(z) &= z^{-2}Y_{DSM22}(z) \\ &\quad + \frac{1}{kff21 \cdot kff22 \cdot kff31 \cdot kff32}(1-z^{-1})^4 Y_{stage3}(z) \end{aligned} \tag{9.16}$$

$$Y_{DSM22}(z) = kff11 \cdot kff12 \cdot z^{-4}X(z) + \frac{1}{kff21 \cdot kff22}(1-z^{-1})^4 E_2(z) \tag{9.17}$$

$$\begin{aligned} Y_{DSM222}(z) &= kff11 \cdot kff12 \cdot z^{-6}X(z) \\ &\quad + \frac{1}{kff21 \cdot kff22 \cdot kff31 \cdot kff32}(1-z^{-1})^6 E_3(z) \end{aligned} \tag{9.18}$$

Our model for the DSM ADC is taken from the work of Jian-Yi Wu [37]. His work developed a MATLAB® model of a DSM211 by combinding the works of Yin and Medeiro with del Rio. Yin developed a DSM211 topology with some non-idealities, which Medeiro expanded upon and generalized to both a DSM211 and DSM22. The work of del Rio provided an analytical analysis of integrator modeling and non-idealities. Wu's work coded the models in MATLAB® and expanded the scope of non-idealities

considered.

Since the intent of our work is to develop the MDA methodology for mixed-signal design, instead of creating a new DSM model, we used Wu's model as a basis. Since our method is based on VC modules, we needed to entirely reorganize and sometimes rewrite Wu's models to make them modular. To shown the difference in intent, a sample of Wu's code is given:

```
%1st stage
vr1=v_fb1;
v_os1=(cs1/ci1*(vin1(i)-kf1*vr1)+vout1_n_1)/(gm1*ro1)+v_offset1;
vin1_org=vin1(i);
vin1(i)=(v_har_rc(v_settle_rc(vin1(i),ts,cs1),fin,fs,cs1)+v_clk_jitter(Ain,fin,t(i)))-kf1*v_se
   +v_ch_inj(0,v_os1-vss,ci1,v_os1)+v_opsw_n(kf1,cs1,c_eq1,nt1);             %input signal wit
v_int_out1(i)=integrator3(ci1,cs1,cin1,cl1,ro1,gm1,io_p1,io_n1,psrr,vin1(i),vin1_org,vout1_n_1
%2nd stage
vr2=v_fb1;
%v_os2=(cs2/ci2*(vout1_n_1-kf2*vr2)+vout2_n_1)/(gm2*ro2)+v_offset2;
vin2_org=vout1_n_1;
vin2(i)=vout1_n_1-kf2*vr2;
v_int_out2(i)=integrator3(ci2,cs2,cin2,cl2,ro2,gm2,io_p2,io_n2,psrr,vin2(i),vin2_org,vout2_n_1
%1st comparator
[v_comp_out1(i),index1]=comparator(v_int_out2(i),index1);

%============================================================
%3rd stage
vr3=v_fb2;
%v_os3=(cs3/ci3*(vout2_n_1-0.375*vr2-kf3*vr3)+vout3_n_1)/(gm3*ro3)+v_offset3;
vin3_org=vout2_n_1-0.375*vr2;
vin3(i)=vout2_n_1-0.375*vr2-kf3*vr3;
v_int_out3(i)=integrator3(ci3,cs3,cin3,cl3,ro3,gm3,io_p3,io_n3,psrr,vin3(i),vin3_org,vout3_n_1
%2nd comparator
[v_comp_out2(i),index2]=comparator(v_int_out3(i),index2);

%============================================================
%4th stage
vr4=v_fb3;
%v_os4=(cs4/ci4*(vout3_n_1-0.25*vr3-kf4*vr4)+vout4_n_1)/(gm4*ro4)+v_offset4;
vin4_org=vout3_n_1-0.25*vr3;
vin4(i)=vout3_n_1-0.25*vr3-kf4*vr4;
v_int_out4(i)=integrator3(ci4,cs4,cin4,cl4,ro4,gm4,io_p4,io_n4,psrr,vin4(i),vin4_org,vout4_n_1
%3rd comparator
[v_comp_out3(i),index3]=comparator(v_int_out4(i),index3);

%============================================================
vout1_n_1=v_int_out1(i);     %keep previous output of the integrator
vout2_n_1=v_int_out2(i);     %keep previous output of the integrator
vout3_n_1=v_int_out3(i);     %keep previous output of the integrator
     vout4_n_1=v_int_out4(i);        %keep previous output of the integrator
```

```
      v_fb1=v_comp_out1(i);
      v_fb2=v_comp_out2(i);
 v_fb3=v_comp_out3(i);
```

Now we show the equivalent code, developed for modularity:

```
%%----------------------------
%% 1st stage
%%----------------------------

%% 1st stage, feedback DAC
vfb1 = fbdac(param.fbdac1, dquant1(i));

%% 1st stage, 1st integrator
vint11(1,i+1) = int2input(param.int11, fs, vint11(i), vsupplyNoise(i), vin(i), vfb1);

%% 1st stage, 2nd integrator
vint12(1,i+1) = int2input(param.int12, fs, vint12(i), vsupplyNoise(i), vint11(i), vfb1);

%% 1st stage quantizer
dquant1(1,i+1) = quantizer(param.quant1, dquant1(i), vint12(i+1));


%%----------------------------
%% 2nd stage
%%----------------------------

%% 2nd stage, feedback DAC
vfb2 = fbdac(param.fbdac2, dquant2(i));

%% 2nd stage, 1st integrator
vint21(1,i+1) = int3input(param.int21, fs, vint21(i), vsupplyNoise(i), vint12(i), vfb1, vfb2);

%% 2nd stage quantizer
dquant2(1,i+1) = quantizer(param.quant2, dquant2(i), vint21(i+1));


%%----------------------------
%% 3rd stage
%%----------------------------

%% 3rd stage, feedback DAC
vfb3 = fbdac(param.fbdac3, dquant3(i));

%% 3rd stage, 1st integrator
vint31(1,i+1) = int3input(param.int31, fs, vint31(i), vsupplyNoise(i), vint21(i), vfb2, vfb3);

%% 3rd stage quantizer
dquant3(1,i+1) = quantizer(param.quant3, dquant3(i), vint31(i+1));
```

Each command in this section of code calls a subroutine that corresponds to an actual module necessary to implement the design in hardware. The arguments correspond

to the abstraction of the actual module ports as much as possible. Since MATLAB® modules do not inherit parameters in the manner that AHDL modules do, we use the first argument to pass a structure of parameters, including parameters for that module and revelant fabrication parameters. While fs is a parameter for the clock frequency, we pass it separately as if it were the clock signal requiring its own port. Because MATLAB® is inherently a discrete time simulation environment, passing the actual clock would be senseless. The data for each voltage signal is effectively sampled data. Additionally, adding current bias or power-down ports does not add useful behavior in this discrete time environment.

However when we port the code to VerilogAMS, the modules' port defintions must exactly match with their schematic counterparts. Every bias line and differential must be accounted. A code snippet from the beginning of the first DSM stage module is provided:

```
module dsm222stage1 (vfb1n, vfb1p, vint11n, vint11p, vint12n, vint12p, vquant1,
    clk1, clk1a, clk2, clk2a,
    ibias11, ibias12, ibias13, ibias14, ibias15, ibias_nin11,
    pdbar, vdd, vin_swn, vin_swp, vmid, vrefn, vrefp, vss);

  //---------------------------------
  // I/O ports and input parameters
  //---------------------------------
  //% parameter constraints
  // feedforward path gain
  parameter real param_kff11 = 0.25 from (0:inf);
  parameter real param_kff12 = 0.50 from (0:inf);
  // quantized feedback path gain
  parameter real param_kfb11 = 2 from (0:inf);
  parameter real param_kfb12 = 2 from (0:inf);
  // feedback DAC scaling gain
  parameter real param_kdac1 = 0.125 from (0:inf);
```

The port definition is shown with every differential signal, each clock phase, each bias input, and a power-down signal. Additionally, some of the parameters are shown

to highlight that AHDL models allow instances to inherit paramters. As such any instance call must define each parameter to be changed from default. A sample of this call is given:

```
dsm222int11 #(
    .param_fab_cox(`fab_cox),
    .param_fab_un(`fab_un),
    .param_fab_up(`fab_up),
    .param_fab_vth0n(`fab_vth0n),
    .param_fab_vth0p(`fab_vth0p),
    .param_fab_phib(`fab_phib),
    .param_fab_gamman(`fab_gamman),
    .param_fab_gammap(`fab_gammap),

    .param_kff(`kff11),
    .param_kfb(`kfb11),
    .param_kdac(`kdac1),
    .param_fs(`freq_samp),
    .param_seed(`seed11),

    .param_ci(1e-12),
    .param_ro(2.544e6),
    .param_A0(6309.12),
    .param_kioutp(20),
    .param_kioutn(20),
    .param_vdelta_outp(0),
    .param_vdelta_outn(0),
    .param_cin(208e-15),
    .param_psrr_zero(`M_PI * 2e3),
    .param_psrr_pole(`M_PI * 2e8),
    .param_cmrr_zero(`M_PI * 2e3),
    .param_cmrr_pole(`M_PI * 2e8),
    .param_vos(10e-3),
    .param_nt(0.457),
    .param_gain_a1(0),
    .param_gain_a2(-0.171),
    .param_cl(1e-12),
    .param_cap_a1(50e-6),
    .param_sw_wp(48e-6),
    .param_sw_lp(0.6e-6),
    .param_sw_wn(30e-6),
    .param_sw_ln(0.6e-6),
    .param_sw_kratio(0.5)
  )
  xint11 (
    .voutn(vint11n),
    .voutp(vint11p),
    .clk1(clk1),
    .clk1a(clk1a),
    .clk2(clk2),
    .clk2a(clk2a),
    .ibias_int(ibias11),
    .ibias_vbuf(ibias12),
    .pdbar(pdbar),
    .vdd(vdd),
```

```
    .vinFbn(vfb1n),
    .vinFbp(vfb1p),
    .vinSign(vin_swn),
    .vinSigp(vin_swp),
    .vmid(vmid),
    .vss(vss)
  );
```

This code snippet represents an instantiation of one integrator module into the first stage of the DSM ADC. While VerilogAMS allows positional port mapping as MATLAB® requires for its arguments, we elect to use explicit port mapping to ensure proper connectivity. Additionally, each integrator also has a couple dozen parameters that can each be overriden. For the same reason, we elect to use explicit paramter mapping. And while defining multiple mappings per line might make the code more compact, we belive that it would also make the code less intelligible. As the code serves the same purpose as schematics, clear documentation is a priority.

At the most base levels, the math for the behavior algorithms is largely unchaged for these clocked circuits. While we can define the signals in a continous time and provide appropriate transistions and delays for more discrete signals, the fundamental math is unchanged. Even the commands within the two languages are similar, so the base algorithms are similar. The biggest change in porting the code from MATLAB® to VerilogAMS is to create realistic interfaces. The MATLAB® abstracts the interfaces to the bare essentials, and the VerilogAMS requires accurate interfaces, as discussed.

9.4 Developing MDA Models for Mixed-Signal System Design

In order to add a VC to the MDA process, we begin with a working VerilogAMS model and its corresponding schematic. Ideally, the model has been characterized

from fabricated layout, as most of our modules have. Table 9.4 and Table 9.4 show which source modules are mapped to which target models for both architectures. It is important to note that the integrator source module is translated into into target models with different topologies. Specifically, the first integrator in cascaded stages have three-inputs, and the others have two-inputs. Consequently, there are two different schematics that correspond to the same source module. The architecture rules provide for specifying this situation. The reason for this choice is to keep only one switched-capacitor integrator behavioral model, such that updates and improvements only need to happen in one place.

The VerilogAMS model also should be written so as to keep the process characteristics as model parameters, which will aid in re-targeting to different fabrication processes. The model is then tagged, where parameters to be replaced are substituted for a tag to be evaluated, as shown in this source code snippet:

```
  //--------------------------------
  // I/O ports and input parameters
  //--------------------------------
  // fabrication parameters -- <! eval when "1">$fab{'fab'}</!>
  parameter real param_fab_cox = <! eval when "1">$fab{'cox'}</!> from (0:inf);
  parameter real param_fab_un = <! eval when "1">$fab{'un'}</!> from (0:inf);
  parameter real param_fab_up = <! eval when "1">$fab{'up'}</!> from (0:inf);
  parameter real param_fab_vth0n = <! eval when "1">$fab{'vth0n'}</!>
from (0:inf);
  parameter real param_fab_vth0p = <! eval when "1">$fab{'vth0p'}</!>
from (-inf:0);
```

Here the code snippet is for fabrication parameters. The fabrication process information is defined in the architecture rules, a sample of which is given:

```
    $fab{'fab'} = 'ami06typical'; ## AMI06 corner: Typical
    $fab{'cox'}= 2.484e-3; ## (F / m^2) from (0:inf);
    $fab{'un'} = 0.00468; ## (m^2 / V s)  from (0:inf);
    $fab{'up'} = 0.00155; ## (m^2 / V s)  from (0:inf);
    $fab{'vth0n'} = 0.69; ## (V) from (0:inf);
    $fab{'vth0p'} = -0.89; ## (V) from (-inf:0);
```

Module Function	Source Name	DSM211 Target Name
DSM ADC Testbench	dsmAdcTest/adcTest.vams	dsm211AdcTest
DSM ADC Top-Level	dsmAdc/adc.vams	dsm211Adc
Simulation POR Source	benchTools/bench_vpor.va	dsm211Bench_vpor
Simulation Pulse Source	benchTools/bench_vpulse.va	dsm211Bench_vpulse
Analog Section	dsmAnalog/analog.va	dsm211Analog
Clock Generator	clockGen/clockGenNonOverlap.va	dsm211ClockGenNonOverlap
Current Bias	ibias/ibiasGen.va	dsm211IbiasGen
Fully Differential Switch	diffSwitch/invertSignal.va	dsm211InvertSignal
DSM ADC Stage 1	dsmAnalog/stage1.va	dsm211Stage1
DSM ADC Stage 2	dsmAnalog/stage2.va	dsm211Stage2
DSM ADC Stage 3	dsmAnalog/stage3.va	dsm211Stage3
Stage 1, Integrator 1	integrator/intSWCap3in.va	dsm211Int11
Stage 1, Integrator 2	integrator/intSWCap3in.va	dsm211Int12
Stage 2, Integrator 1	integrator/intSWCap3in.va	dsm211Int21
Stage 3, Integrator 1	integrator/intSWCap3in.va	dsm211Int31
One-bit DAC	fbdac/dac1bit.va	dsm211Dac1bit
Latched Comparator	quantizer/regenCompLatch.va	dsm211RegenCompLatch
Digital Error Correction	dsmErrorCorr/errorCorr.v	dsm211ErrorCorr
DEC H1 Filter	dsmErrorCorr/digDelay2.v	dsm211ErrorCorrH1
DEC H2 Filter	dsmErrorCorr/digDiff2.v	dsm211ErrorCorrH2
DEC H3 Filter	dsm211dsmErrorCorr/digDelay2.v	dsm211ErrorCorrH3
DEC H4 Filter	dsm211dsmErrorCorr/digDiff4.v	dsm211ErrorCorrH4

Table 9.1: DSM211 Topology Source to Target Mapping.

Module Function	Source Name	DSM222 Target Name
DSM ADC Testbench	dsmAdcTest/adcTest.vams	dsm222AdcTest
DSM ADC Top-Level	dsmAdc/adc.vams	dsm222Adc
Simulation POR Source	benchTools/bench_vpor.va	dsm222Bench_vpor
Simulation Pulse Source	benchTools/bench_vpulse.va	dsm222Bench_vpulse
Analog Section	dsmAnalog/analog.va	dsm222Analog
Clock Generator	clockGen/clockGenNonOverlap.va	dsm222ClockGenNonOverlap
Current Bias	ibias/ibiasGen.va	dsm222IbiasGen
Fully Differential Switch	diffSwitch/invertSignal.va	dsm222InvertSignal
DSM ADC Stage 1	dsmAnalog/stage1.va	dsm222Stage1
DSM ADC Stage 2	dsmAnalog/stage2.va	dsm222Stage2
DSM ADC Stage 3	dsmAnalog/stage3.va	dsm222Stage3
Stage 1, Integrator 1	integrator/intSWCap3in.va	dsm222Int11
Stage 1, Integrator 2	integrator/intSWCap3in.va	dsm222Int12
Stage 2, Integrator 1	integrator/intSWCap3in.va	dsm222Int21
Stage 2, Integrator 2	integrator/intSWCap3in.va	dsm222Int22
Stage 3, Integrator 1	integrator/intSWCap3in.va	dsm222Int31
Stage 3, Integrator 2	integrator/intSWCap3in.va	dsm222Int32
One-bit DAC	fbdac/dac1bit.va	dsm222Dac1bit
Latched Comparator	quantizer/regenCompLatch.va	dsm222RegenCompLatch
Digital Error Correction	dsmErrorCorr/errorCorr.v	dsm222ErrorCorr
DEC H1 Filter	dsmErrorCorr/digDelay2.v	dsm222ErrorCorrH1
DEC H2 Filter	dsmErrorCorr/digDiff2.v	dsm222ErrorCorrH2
DEC H3 Filter	dsm222dsmErrorCorr/digDelay2.v	dsm222ErrorCorrH3
DEC H4 Filter	dsm222dsmErrorCorr/digDiff4.v	dsm222ErrorCorrH4

Table 9.2: DSM222 Topology Source to Target Mapping.

In this section of code, we define various process parameters, such as oxide capacitance, base mobility, and threshold voltage. The units are given for clarity. Each value will be instantiated at translation into the source code. The section of fabrication parameters directly matches the tags in the source code given above. Each complete tage will be replaced by its corresponding value.

Sections of source code that may or may not be included, such as specific noise sources that may not be relevant in every integrator or extra input ports, are also tagged for elimination or inclusion based on the architecture rules. This code snippet shows the tagged source code to be included or not:

```
    input vinFbp;
    electrical vinFbp;
  <! verbatim when "($module{'module'} =~ /int[23]1$/i)">  input vinQntn;
    electrical vinQntn;
    input vinQntp;
    electrical vinQntp;
  </!>  input vinSign;
    electrical vinSign;
```

And this snippet shows the architecture rule that controls the previous source code tag set:

```
## declare a hash for each module and its 'moduleName' entry
foreach $module (@{$application->{'moduleNames'}}){
  $application->{'modules'}->{$module}->{'project'} = $project{'name'};
  $application->{'modules'}->{$module}->{'application'} = $application;
  $application->{'modules'}->{$module}->{'topology'} =
$application->{'topology'};
  $application->{'modules'}->{$module}->{'module'} =
$application.ucfirst($module);
  $application->{'modules'}->{$module}->{'simulator'} = $project{'simulator'};
    }
```

Specifically, the command to set the appropriate variable where `$module='int31'`, `$module='int12'`, or some such is:

```
 $application->{'modules'}->{'module'} = $application.ucfirst($module);
```

The target source `%module` associative array is a reference to the architecture rules associative array:

```
%application->{'modules'}->{$module}
```

If `$module='int21'` from a DSM222 topology, then the respective translation engine output of the two source code snippets are:

```
  // fabrication parameters -- ami06typical
  parameter real param_fab_cox = 0.002484 from (0:inf);
  parameter real param_fab_un = 0.00468 from (0:inf);
  parameter real param_fab_up = 0.00155 from (0:inf);
  parameter real param_fab_vth0n = 0.69 from (0:inf);
  parameter real param_fab_vth0p = -0.89 from (-inf:0);
```

and

```
    input vinFbp;
    electrical vinFbp;
    input vinQntn;
    electrical vinQntn;
    input vinQntp;
    electrical vinQntp;
    input vinSign;
    electrical vinSign;
```

The abstraction that we are making in our MDA process is to extract the functional specifications from the VerilogAMS view of the VC. As the VerilogAMS model is already behavioral, this abstraction is the next logical step. The layout is the lowest level implementation, and the schematics are a structural model for the layout. The VerilogAMS model is a behavioral abstraction of that structure. The specifications are the abstraction of the behavioral model, and a sample application model section (of just the ADC) is provided:

```
## name of this project (perhaps like an EDA tool library name)
$project{'name'} = 'xm_rx_tx'; ## no spaces

{ ## ADC application model
    ## name of this application
    my $application = 'wifi_adc';  ## no spaces
    push @applications,$application;
    $application->{'name'} = $application;
```

```
    ## function of this application
    $application->{'function'} = 'adc';
    ## minimum resolution (bits)
    $application->{'specs'}->{'minimum resolution'} = 16;
    ## minimum signal-to-noise ratio (dB)
    $application->{'specs'}->{'minimum SNR'} = 84;
    ## minimum spurious free dynamic range (dB)
    $application->{'specs'}->{'minimum SFDR'} = 92;
    ## minimum effective number of bits (dB)
    $application->{'specs'}->{'minimum ENOB'} = 14.1;
    ## minimum integral non-linearity (bits)
    $application->{'specs'}->{'minimum INL'} = 0.5;
    ## minimum differential non-linearity (bits)
    $application->{'specs'}->{'minimum DNL'} = 0.5;
    ## maximum total power consumtion (W)
    $application->{'specs'}->{'maximum quiescent power'} = 1.4;
    ## single-ended supply voltage (V)
    $application->{'specs'}->{'supply voltage'} = 5;
    ## maximum sampling/clock frequency (Hz)
    $application->{'specs'}->{'maximum sampling frequency'} = '40M';
    ## maximum input bandwidth (Hz)
    $application->{'specs'}->{'maximum input bandwidth'} = '1.25M';
    ## maximum input amplitude (V)
    $application->{'specs'}->{'maximum input amplitude'} = 1;
    ##  reference voltage, vref (V)
    $application->{'specs'}->{'reference voltage'} = 2;
    ##  reference current, iref (A)
    $application->{'specs'}->{'reference current'} = '50u';
    ##  POR (power-on-reset) reset time, tpor (s)
    $application->{'specs'}->{'POR reset time'} = '1u';
}
```

Of course, a marking model is necessary to indicate in what format the target models will be, as well as which simulator and fabrication process will be used. A sample marking model to correspond to the application model snippet is given:

```
{ ## ADC marking model
    ## fabrication process
    $project{'fabProcess'} = 'ami06typical'; ## required
    $project{'simulator'} = 'cadence';

    my $application = 'wifi_adc';  ## no spaces
    ## target model platform
    $application->{'target'} = 'verilog'; ## required
}
```

Certainly, the abstraction process is not trivial, as it requires that the analog gurus document and code the specification metrics of an AHDL module, as well as the performance range that a VC can achieve. Without the possible performance range,

the topological selection cannot be achieved. While these requirements may take development time, their consideration is risk mitigating. Design errors caused by using a module outside of its valid region of operation are greatly reduced, as those valid regions are explicitly determined and coded into the topological selection. Further, the abstraction only need be done once, then the VC can be used in system design until fabrication process become obsolete. Also, the VC AHDL modules can be aggregated into super-modules of many sub-modules, bundling more functionally into one application. For example, until the DSM222 super-module is created, the system designer would have to have an understanding of how to create a DSM222, to combine all of the submodules together in various application module sections, and to create test benches for the DSM222.

The intention of the MDA methodology is to shield the system designer from this process as much as possible, so super-modules are vital to being able to develop ever more complex systems.

CHAPTER 10

CONTRIBUTIONS AND FUTURE WORK

In this work, we have discussed the application of the MDA methodology to mixed-signal design. We have shown how for two distinctly different implementations an MDA process can be applied to achieve useful abstraction and automated implementation, which enables design development, documentation, reuse, and decreased TTM. In both implementations, we discussed the design processes as they exist presently, the abstractions available, and how these examples fit within the MDA methodology.

In Chapter 3, we have shown how the PDK development process both is necessary to system design to provide fundamentally necessary design components, as well as efficiency improving elements, and takes development time that would better be spent on system development, instead of rote PDK generation. In Chapter 4, we have discussed how our MDA process allows reuse of PDK elements, such that once they have been implemented, the time impact of re-targeting to a new fabrication process is minimal. This implementation clearly exhibits the impact that an MDA process can have to generate content for a mixed-signal design project, because the PDK generation process is isolated from other processes; the development process begins

and ends within our proposed MDA process.

For a second implementation, we have discussed mixed-signal design methodologies. Firstly, we discussed the existing methodologies and developing EDA toolsets in Chapter 5. The issues with some of these traditional methods have been shown in Chapter 6 through design examples that we developed within the span of this work. Then in Chapter 7, we have given a view of the next generation of design methods that we envision will naturally emerge to take full advantage of developing EDA tools. Finally, in Chapter 8, we have proposed an iteration of design methodologies. Our proposed method uses an MDA process to further abstract the mixed-signal system design flow. The abstraction of schematic topology, even when represented by a corresponding behavioral AHDL model, to a set of functional specifications allows for a partitioning of the design. This partitioning has been shown to allow the analog gurus to specialize their precious talents in the development of VC libraries. These libraries can be used by system designers with less specific knowledge of analog design to build more complex systems in less time. The automation in the process from specification to layout allows for the increased complexity with decreased TTM.

These gains are possible because the MDA process engenders design abstraction and automated implementation. We have fit the abstractions and their implementations into application models, marking models, and architecture rules necessary to generate target models. We have shown how we have generated these necessary models along with a translation engine which is able to generate the target models. The abstraction and model model generation process for the PDK development has been shown

in Chapter 4, and in Chapter 9 for the system design process. In these chapters, we have also shown how this work generates useful content to the mixed-signal system design process.

We have given an argument that anywhere in the mixed-signal design process that non-trivial deliverables are generated, have an abstraction possible, and will be re-generated in some form, a MDA methodology can applied to achieve better design in a shorter time, taking mixed-signal design flows into the next generation.

Of course, this body of work has been focused in its intent to show the MDA process for the mixed-signal design. Specifically, we did not set out to create a commercial EDA tool. As such, there are variety of avenues left open upon which to expand and iterate. Within PDK development, only Cadence® and Laytools® were considered. Architecture rules could be written for a variety of other EDA tools, such as Dolphin® and Mentor Graphics®. Even within the Cadence® and Laytools® architecture rules and source files, not every possible PDK feature was implemented.

The scale of MATLAB® code that could translated was limited for the purpose of demonstration. A canonical list of control statements could be implemented. Additionally, instead of only translating MATLAB®'s *plot* command into a layout polygon, subroutines corresponding to actual layout generation methods (rectangle, polygon, path, etc) could be written for MATLAB® and added to the architecture rules for translation. Further, only the architecture rules were written for Cadence® pcell generation from MATLAB®. The rules for Laytools® and other EDA tools could be

added.

In the system design sections, our focus was on the DSM ADC example. A large VC library would be necessary to consider complete system design projects, such as the one shown in Figure 9.1. Further, improvements could be made in the manner that application and marking models are captured. Files containing a sea of specifications may not be the most efficient manner to implement these models. One suggestion would be building a MATLAB® SimuLink® front end for these models. Each application in the project could have its own block where specifications could be captured in a more standard framework. The SimuLink® modules would generate the actual application and marking models for the translation engine. In fact, if the SimuLink® models had a deeper understanding of the meaning of the specifications then a simulation could be run on the application model in MATLAB®. Being able to simulate at this level of abstraction would preclude the need to translate the system, optimize its AHDL models, then simulate them to get an initial understanding of the performance.

Finally, the architecture rules were captured in PERL code, except for the tagged source AHDL models. The capture of topological selection, hierarchical structure, and parameter mapping in PERL may not be ideal. The design abstraction from topology to specification may be unnecessarily obfuscated. And the structures necessary to make the rules consistent may be unnecessarily cumbersome. While the methodology is sound, the exact implementation could be improved to be more obvious. Specifically, it has been difficult enough to get analog gurus to write AHDL

models. Trying to force them to write this style of PERL code may be unreasonable. In this work, the focus on the translation method, so the user interfaces were a secondary consideration and are less ideal as such. In fact, the raw code for the architecture rules is plausible, if a wrapper could be written to generate it from a more intuitive user interface. Our novel methodology has been soundly implemented through the translation engine, and its range can be expanded with within the VC library, especially with some clever user interface enhancements.

BIBLIOGRAPHY

[1] Barcelona Design, "Automated design flow," URL: `http://www.barcelonadesign.com/corporate/products/default.asp`, Dec. 2001.

[2] Object Management Group, "Model driven architecture - a technical perspective," Tech. Rep., OMG, July 2001.

[3] Franck Fleurey, Jim Steel, and Benoit Baudry, "Validation in model-driven engineering: testing model transformations," *Proceedings of the First International Workshop on Model, Design and Validation*, pp. 29–40, Nov. 2004.

[4] David Hearnden, Kerry Raymond, and Jim Steel, "Anti-Yacc: MOF-to-text," *Proceedings of the Sixth International Enterprise Distributed Object Computing Conference*, pp. 200–211, Apr. 2002.

[5] Kurt Keutzer, Sharad Malik, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1523–1543, Dec. 2000.

[6] Ken Kundert, Henry Chang, Dan Jefferies, Gilles Lamant, Enrico Malavasi, and Fred Sendig, "Design of mixed-signal systems-on-a-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1561–1571, Dec. 2000.

[7] R. Sommer, I. Rugen-Herzig, E. Hennig, U. Gatti, P. Malcovati, F. Maloberti, K. Einwich, C. Clauss, P. Schwarz, and G. Noessing, "From system specification to layout: seamless top-down design methods for analog and mixed-signal applications," *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pp. 884–891, Mar. 2002.

[8] Object Management Group, "Unified modelling language (uml) version 1.4," Tech. Rep., OMG, Sept. 2001.

[9] Object Management Group, "Meta-object facility (mof) version 1.3," Tech. Rep., OMG, Apr. 2000.

[10] Open Kit Technical Committee, "Design objective document," URL: `http://www.eda.org/openkit/doc/OK_DOD_100103.pdf`, Oct. 2003.

[11] Peter Conradi, *Reuse in Electronic Design*, John Wiley & Sons, 1999.

[12] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, and Andrew McNelly Lee Todd, *Surviving the SOC Revolution*, Kluwer Academic Publishers, 1999.

[13] Gary Pratt, "Automated Top-Down Design Rapidly Becoming Essential For Mixed-Signal Circuitry," URL: `http://www.planetee.com/planetee/servlet/DisplayDocument?ArticleID=14521`, Mar. 2001, Mentor Graphics Corp.

[14] Walden C. Rhines, "Design reuse: Great promise, great difficulty, great decisions ahead," VSIA Conference, Oct. 2000.

[15] Henry Samueli, "Designing in the new millennium: It's even harder than we thought," Opening Keynote Address at the 38th Design Automation Conference, June 2001.

[16] George Gielen, "When will the analog design flow catch up with th digital methodology?," Session 26 Panel at the 38th Design Automation Conference, June 2001.

[17] Caitlin Kelly, "Ingenious new engineering programs stress teamwork, creativity, business, society: Teaching from a clean slate," *IEEE Spectrum*, Sept. 2001.

[18] Ron Vogelsong, "Ams behavioral modeling workshop," International Cadence Usergroup Conference, Dec. 2001.

[19] John Sheridan Fisher and Steve Bibyk, "Design Methods for SOC Control CODECS to Enhance Reuse," *IEEE Proceedings of the 14th Annual ASIC/SOC Conference*, pp. 423–427, 2001.

[20] NASA Quest, "Space team online presents: International space awareness day forum with nasa and space science experts," URL: `http://ltp.arc.nasa.gov/space/chats/archive/05-27-99isw.html`, May 1999.

[21] Joseph M. Benedetto, "Radiation field sets design limits," *Electronic Engineering Times*, p. 52, Jan. 1999.

[22] Kang Lee, "A standard in support of smart transducer networking instrumentation and measurement," *Proceesing of the 17th IEEE IMTC Technical Conference*, pp. 525–528, 2000.

[23] Burr-Brown, "RCV420, Precision 4mA to 20mA Current Loop Receiver," URL: http://focus.ti.com/docs/prod/productfolder.jhtml?genericPartNumber=RCV420.

[24] David A. Johns and Ken Martin, *Analog Integrated Circuit Design*, John Wiley & Sons, Inc, New York, first edition, 1997.

[25] Bharath Mandyam, "Design issues in i and ii order sigma-delta modulators," M.S. thesis, The Ohio State University, 1999.

[26] B.E. Boser, *Design and Implementation of Oversampled Analog-to-Digital Converters*, Ph.D. thesis, Stanford, 1989.

[27] Kishan Shenoi, *Digital Signal Processing in Telecommunications*, Prentice Hall PTR, New Jersey, 1995.

[28] Eugene B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, pp. 155–162, Apr. 1981.

[29] ATMEL, "SPI Serial EEPROMs 1M (131,072 x 8) AT25P1024 Preliminary," URL: http://www.atmel.com/atmel/acrobat/doc1082.pdf.

[30] Roger L Stevens, *Serial PIC'n*, Square 1 Electronics, Kelseyville, CA, 1999.

[31] Inc. Microchip Technologies, "PIC16C63A / 65B / 73B / 74B, 8-Bit CMOS Microcontrollers with A/D Converter, 10.0 Synchronous Serial Port (SSP) Module," .

[32] Phillip E. Allen and Douglas R. Holberg, *CMOS Analog Circuit Design*, Oxford University Press, New York, first edition, 1987.

[33] Behzad Razavi, *Design of Analog CMOS Integrated Circuits*, Prentice Hall PTR, first edition, 2000.

[34] Anup Salva, "Low-power design approaches for programmable-speed pipelined analog-to-digital converters," M.S. thesis, The Ohio State University, 2002.

[35] Behzad Razavi, *RF Microelectronics*, Prentice Hall PTR, first edition, 1997.

[36] Steven Norsworthy, Richard Schreier, and Gabor Temes, Eds., *Delta-Sigma Data Converters*, IEEE Press, Piscataway, NJ, 1997.

[37] Jian-Yi Wu, *Round Trip Design Methods for DSM Design*, Ph.D. thesis, The Ohio State University, 2001.