Overview of Design Methodology

Lecture 1

Put things into perspective

A Few Points Before We Start

All About Handling The “Complexity”

• Design and manufacturing of semiconductor products are very complex processes
  - Design methodologies
  - Manufacturing methodologies

• All knowledge are to overcome complexity

• A good engineer has effective ways to overcome complexity in a problem
  - Yet, complex problem usually doesn’t require complex solution
Two Scientific Thinkings

- **Realists** - The results of scientific discovery are real laws in nature
  - Find the fundamental laws governing the complexity we see

- **Instrumentalists** - The results of scientific discovery are instruments for making good prediction. The actual laws in nature can be very different from the ones discovered
  - Find ways to predict the behavior we observe, without understanding the laws behind

Two fundamental approaches

- **Deduction**
  - Moving from general to particular
  - Eg. Learn theories and algorithms, apply to solve complex problems
  - Eg. Develop theories, models, and algorithms, and apply them to overcome the complexity
  - The approach in the past 40 years

- **Induction (Engineering)**
  - Moving from particular to general
  - Eg. Learn from the behavior, find a solution to deal with the behavior
  - Eg. Learn from multiple behaviors, generalize a solution to deal with a class of behaviors
  - The problem to face in the next 10 years

Observation

- In the past 40 years, design and manufacturing processes are dominated by the thinking of deduction
  - Develop the theories, models first
  - Work well when the complexity is low

- In the next 10 years, the processes will mostly be dominated by induction methodologies
  - Too difficult to find theories and models behind complex phenomenon we encounter
• In the engineering world, there are three types of skill
  - Don’t have implementation skills -> marketing
  - Don’t have domain knowledge -> solving a wrong problem
  - Don’t have theoretical sense -> re-inventing the wheel

How to learn the three skills?

• Implementation skill
  - Basic constructs can be taught
  - Skill to implement cannot be taught
  - Depend on self motivation and learning

• Theoretical sense
  - Theories and concepts can be taught
  - Application of theories and concepts cannot be taught
  - It is based on experience

• Application domain knowledge
  - Facts and phenomenon can be described
  - Empirical judgments are hard to teach

Lectures

• In addition to Verilog, I also talk about “application domain knowledge”

• Some that you can learn here are hard to get by self-study (and also most irrelevant to your grade)
  - Trends (facts and phenomenon)
  - Experience

• You can always learn more by self-studying
  - Use the Internet
  - Practice your Verilog skills
  - Read papers and other books
Design Begins With Models and Simulation

In a design process, people rely on models and simulation tools to construct a design.
- Keep in mind, in design we do NOT deal with the real things. We only deal with models and simulations.

In a manufacturing process, the final design model is used to produce the real product.

Design and manufacturing

The need for models

- You need to worry about two main things
  - Logical functionality
  - Electrical properties or characteristics
  - How to ensure those before manufacturing?

- How do we model a design?
  - So that we can simulate the design to check its behavior (logical & electrical)?
  - So that we can be sure about its functionality and properties before sending the design for manufacturing?
First, Model for Functionality

- We need some sort of "formal model" to describe logical functionality
  - This formal model is understandable by some software tools that can do some required analyses on the design

- To build a design model, you need to have a **modeling language**
  - A language to describe hardware functionality
  - This language is important because everything else in the design process depends on it

Hardware Description Language (HDL)

- HDL is just a modeling language for people to construct a formal model to describe **functionality** of their designs

- Popular HDLs are Verilog and VHDL

- A HDL model is interpreted by a simulation tool for its behavior
  - ModelSIM

- The interpretation is as important as the language itself

Hardware Description Languages (HDLs)

- Highly portable
- Describe multiple levels of abstraction
- Provide many descriptive styles
  - Structural
  - Register Transfer Level (RTL)
  - Behavioral
- Serve as input to simulator & synthesis
- Support structured design methodology
What is a design methodology

• A design methodology is like a flow chart
  - Each component in this flow chart is performing a specific design task, such as logic synthesis

• A design methodology is determined by the Electronic Design Automation (EDA) software tools you purchased
  - And based on some in-house (software) tools

• In a design flow, you may have >50 such tasks to perform

Fig 1-1 – Design Methodology

• Typical design flow for an ASIC (Page 3)

HDL-based Design Flow Example

- Create behavior
  - The initial design model for functional simulation
  - Verify/simulate functionality
  - Pre-synthesis signoff
- Synthesize into gate-level model
  - E.g., using Design Compiler
- Post-synthesis design validation
  - Compare gate-level model and behavior model
  - Logic equivalence checking
- Fault simulation and test generation
  - Design For Test (DFT)
  - Based on gate-level model
- Post-synthesis timing verification
  - E.g., using PrimeTime, static timing analyzer
- Place & route (physical synthesis)
- Verify physical and electrical design rules
  - Physical rule-based manufacturability
- Extract parasitics
- DfC extraction
- Design signoff

RTL Model

Gate Netlist Model

Layout Model
Design flow depends on style

- Not everyone follows the same flow
  - No exactly same flow for two companies

- Different design styles result in different flows

- An FPGA-style is very different from ASIC-style, and very different from processor-style

- Every design process is unique in some sense
  - Flow is proprietary to a company

Design Styles

- ASIC
  - examples: controllers
  - typical characteristics
    - low cost
    - slower speed
    - specific applications
    - short development time
    - older manufacturing technologies
    - mass market
  - ASIC designs can be large in size
    - millions of transistors and more

Processor Designs

- Designs that follow architectural SPECs
  - microprocessors, signal processors, etc.
- Typical characteristics
  - architectural specification
  - long development time
  - expensive
  - fixed market
  - high performance
  - state-of-the-art manufacture facilities
Memory Designs

- On-chip / off-chip storage devices
  - video RAM, flash memory, etc.
  - high density
  - high performance
  - regular structure
  - special design methodology
  - special manufacture facilities
  - growing market
- Memory designs are crucial in many high-performance applications

SOC (System on Chip)

- Core-based design or IP design
- Current hot market
- New challenges in tools/methodologies
- System-on-a-chip

Design Styles

- Considerations (Tradeoff)
  - Performance (Power, Speed)
  - Area
  - Cost
  - Functionality
  - Time to Market
- Design approaches
  - Custom - design circuit by hand
  - Standard cell - use synthesis tool like Design Compiler
  - IP - purchase design from another company
  - FPGA - hardware simulation of your Verilog
Examples

• For fastest time-to-market, rapid prototyping
  - Use FPGA

• For fast time-to-market, low cost ASIC
  - Use commercial standard EDA design flow
  - Make everything as automatic as possible

• For high-performance, high volume design
  - No high-performance and low volume (why?)
  - Use custom design flow, optimize in every step

Post-Silicon

• For complex design, it is virtually impossible to do everything correct first time
  - Yield is usually low at 1st silicon

• Design effort continues into post-silicon stage
  - Fix design to improve yield
  - Develop an effective test methodology

• Top two concerns
  - Yield
  - Test Cost

Design - Post-silicon stage

• Before design - setup manufacture line
  - Test chip, yield learning
  - Either done by foundry or before/during design stage

• After design sign-off, obtain silicon samples
  - Diagnosis of bugs, Yield improvement
  - Silicon calibration, On-chip repair
  - Determine mass production test strategies
  - Speed sorting method
  - System bring-up test
  - Performance validation
  - Re-spin

• After several re-spins, start mass production
  - Manufacturing test (for defects)
  - Performance sorting
  - System test
  - Reliability test (burn-in)
  - Ship to customers
Post-production stage

- Continuous yield improvement
  - Identify common failure modes
  - Root cause identifying
  - Generate fix(es)

- Test cost reduction
  - Identify redundant tests
  - Adapt test processes to save cost by removing
    redundant tests

- Failure analysis
  - Obtain customer returns
  - Diagnosis and silicon debug (root cause)
  - Feedback to design (if worthy)

Recall: HDL-based Design Flow

- Create behavior
  - The initial design model for functional simulation
  - Verify/simulate functionality
  - Pre-synthesis Signoff

- Synthesize into gate-level model
  - E.g., using Design Compiler
  - Post-synthesis design validation
  - Compare gate-level model and behavior model
  - Logic equivalence checking

- Fault Simulation and Test Generation
  - Design for Test (DFT)
  - Based on gate-level model

- Post-synthesis Timing Verification
  - E.g., using PrimeTime, static timing analyzer

- Place & route (physical synthesis)
- Verify physical and electrical design rules
- Layout rules for manufacturability
- Extract parasitics
- RC (L) extraction

- Design sign-off

RTL Design Entry - Processor

- No explicit behavior model
- RTL is the formal SPEC for the design
  - Provides cycle accurate information
- Schematic is the model of actual implementation
  - Synthesize
  - Custom design
- Gate-level model is for test preparation
  - Model for ATPG tool
Design Data Hierarchy

- **Behavior**
  - Like a C program
- **RTL (Register Transfer Level)**
  - Explicitly define all state-holding elements
  - Provide a cycle-accurate model
- **Gate**
  - All high-level constructs (if, case, assign) are converted to gates (and, or, not).
- **Transistor**
  - All are converted with nmos, pmos, etc.
- **Layout (final picture)**

Purpose of Data Model

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Functional Simulation</th>
<th>Specification</th>
<th>Less Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL</td>
<td>Synthesis/Simulation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gate Level</td>
<td>Test Generation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transistor/</td>
<td>Physical Properties</td>
<td>Implementation</td>
<td>Very Complex</td>
</tr>
<tr>
<td>Layout</td>
<td>Area/Timing Optimization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Verification Challenges

- **Functional Verification (with slow simulation)**
  - What functions do we need?
  - How do we know the design performs all functions we want?
- **Equivalence Checking (with discrepancy in modeling perspective)**
  - Specification = Implementation?
- **Test Preparation (with unpredictable defects)**
  - A chip is manufactured correctly?
- **Performance Constraints (with inaccurate modeling)**
  - Timing, power, noise are ok?
Verification Perspective

- Functional verification is most time consuming
- Timing and power validation cannot be 100% in the pre-silicon phase due to model inaccuracy

Quality Issues

- How do we know we have done a good job?
  - Never can be sure?
  - Can our customers find out?

- How do we know what it is going wrong?
  - Locate a design error or manufacturing defect
  - Analysis the cause
  - Go back and fix the design or the process
  - can be costly

Do Everything Early

- Because the later you have a problem, the higher price it will cost you
- However, the earlier the stage, the less accurate the models are
  - Fundamental question: With a somewhat accurate but not 100% accurate model, how can you NOT over-doing it?
Design Challenges

How difficult to do a design?
- How big a HDL model can be?
  - Millions of lines
- How long to simulate a design
  - Days, weeks
- How long to do a design
  - Months, years
- How do we know it is done
  - No body truly know

No body truly knows?
- Why?
  - Because the design is too complex
  - No body has the power to know
- Also ...
  - There is no need to know
- In the real world, you don’t need to be perfect to be successful
  - You need to be much better than all your competitors
- In the real world, there is always someone else to solve the problems left by you
How Complex?

- Over the years, the design industry follows this so-called “Moore’s Law”
- By advancing the technologies so fast, we are making chips today that are so complex
- The complexity of making chips is so high today that it can go beyond control soon if we don’t do something about it

Moore’s law?

- Gordon Moore, Intel co-founder (1968)
- Moore’s Law – rule of thumb in semiconductor industry (a reference for competition)
  - He said that the number of “components” on a single silicon chip would double every 1.X year
  - Somebody says double every 1 year
  - Somebody says double every 1.5 years
  - Most recently, we say “double every 2 years
- Some people also say the “performance double every 1.x year”
  - In short, “industry should move fast!”
Trend – Frequency (Speed)

We are stuck below 5 GHz

Trend – Performance

MIPS: million instructions per second
TIPS: 1 million MIPS

Trend – Transistor Size
Change in 10 years

Intel® Pentium® Processor on 350nm Technology

1995 - 133MHz, 3.3M transistors

Intel® Pentium® 4 Processor on 90nm Technology

2004 - 3.4GHz, 125M transistors

But, if we keep going like this ...

We are going to have ...

Power Dissipation
Power Density

Transistor – the basic building block

- Essentially, it is a switch
  - When voltage on GATE is high (above Vt), current flows between SOURCE and DRAIN
  - When GATE voltage is low (below Vt), supposedly no current flows between them
- But …

Experimental transistors …

Source: Intel

We need to do something to control the leakage power …
Transistor is smaller ...

Small devices leak current!

Sub-Threshold Leakage Current

- Why this is bad?
  - Current flow = Power dissipation
  - A transistor is sitting there doing nothing and still consumes Power
  - One consequence: running out of battery quickly

Leakage power consumption

Power is a key limiter of Moore’s Law
3D Transistor – Year 2007

- 45nm production
- Improve performance
- Lower off current flow

High-k Dielectrics for leakage reduction

- high-k dielectrics are defined as those with a dielectric-constant greater than that of silicon nitride
- This is required for 32nm and below …

Power Wall

- While manufacturing guys are busy improving their process, ECE people are developing new designs and design methods
For example, multi-voltage design

- More power = faster
- So, if we don’t need to run too fast in some part of the chip, just give it lower voltage to save power
  - Is this also happening in the car industry as well?
  - We have seen more hybrid models because of the rising gas price
- This has implications to many design methods

For example - multi-threading design

Multi-threading improves performance by better utilizing power
So we can boost performance without increasing power

The general idea - valued performance

The overall performance is what we really care about
So more processors together, multi-threading make better sense
In Summary
- In the past 20 years, manufacturing technology improved so fast
  - That we were busy building more and more complex designs
- We didn't actually have time to really think hard to optimize our design to better utilize the resource such as Power
- In these few years, people have started working on optimizing the designs
  - Continue to push for better performance
  - But not utilizing more power

Two more walls
- **Variation (reliability) wall**
  - When transistors become so small, it is hard to control their behavior
  - Transistor behavior (sizes) vary significantly
- **Verification wall**
  - We said we wanted to build more complex designs
  - Today's microprocessors consists of millions of lines of RTL codes
  - How do we know that our designs do not have a bug?
    - We need to verify their correctness
    - Very tough task!!

Process variations
This is not new … it just that they are affecting our designs now …
With-in die variation

Depending on where the transistors are located, their sizes differ

Did I mention Power density varies?

- Power is related to speed
- Temperature is related to speed
- Power is related to temperature
- How do I know that my design can really run at 3GHz?
  - You need to consider all the uncertainties

How do I know power density does vary?
Temperature variation

Variations and Defects

- People worry about "process" "temperature" "voltage" variations
- We need to design our chips with these variations in mind
- Not only that, the process uncertainties can cause many unintentional errors on the silicon
  - Manufacturing defects!
- We need to **test** our silicon to screen for any defect
- The bottom line is
  - You don't ship a bad product out of the door!
  - Even though there are so many variations and uncertainties

Modeling all defects is impossible

Credits: Greg Spirakis, Intel Corp. 7th European Test Workshop
A Real Latent Defect Example

- SOC controller for automotive
- Start to fail after driving 15000 miles
- Show failure only under 40°C
- Failure is also frequency dependent
- Determine to be a latent defect!

Test equipment are expensive

S9000 tester, costs millions each

Off-shore test facility – Intel
Test is complex and costs a lot!

Verification

- Verification is to make sure that what you design is what your customer wants and expects
- How do you do it?
  - Simulate your design before you have it as real silicon
  - Problem: Design is too complex and simulation is very “slow”

Functional Verification Demand

- Functional verification cost grows faster than design complexity
Pre-Silicon functional verification

not that we don't try

Verification Crisis

• More than 50% of the project budget already goes to verification
• Simulation and preparation time already drags the time-to-market
• Design complexity grows tremendously with the use of multiple cores
• Cost of chip re-spin is high
  - > $100K for medium chips
  - > $1M or more for a complex designs

In summary, we have 3 big walls

Power Wall
Variation Wall
Verification Wall

Future design
Some Questions to Think

• The three walls are just problems that manifest some fundamental problem(s) of how we do design
  - What is the fundamental problem?

• Will we see a fourth wall in the near future?

• What will be the key enabling technology to succeed five years from now?

In Summary

• We are hitting the verification wall
  - The simulation time drags time-to-market
  - Don’t know if we have simulate enough

• We are hitting the power wall
  - Too many transistors on a single die
  - Leakage is serious for 65nm or below
  - Wireless applications require low power

• We are hitting the process (variation, reliability) wall
  - 22nm will be in mass product
  - Most people stays at 90 or 65nm
  - Analog world stays at .13 or higher
  - Process variations are too high for today’s design methods

Cost Vs. Performance

• In the past 20 years, performance is the king to make big money

• Now, it is “cost” and “application”
  - Reduce design cost, improve turn-around time
  - Improve margin
  - Expand applications to sell more chips
  - Eg. Intel is driving for “total performance” on a single chip (with n processors)

• The world has changed
Design Art

• Design is a "constrained optimization" process
  - Optimize for your goals
  - Timing, power, area, functionality, time to market
  - Under some constraints
    • Constraints may not be well defined
    • Limited time, resources, expense, chip area, etc.

• Constrained optimization is to explore some tradeoff in the design space
  - Understanding the tradeoff is the key
  - The space may not be convex and hence, you may have many local optimal but not global optimal points

Jobs

• Designs
  - System level
  - Logic level
  - Circuit level
  - Verification and debug
  - DFT and test

• Tools
  - Methodologist
  - Tool Developer
  - Application Engineer

Example - Design Companies

• Processors (SoC)
  - IBM, Intel, Freescale, TI, Qualcomm, SamSung, nVidia, Apple, AMD, ARM, Oracle, Broadcom, MediaTek, etc.

• Analog
  - National Semiconductor, Dialog, Qualcomm, TI, etc.

• Memory
  - SamSung, Micron, etc.

• Embedded Cores
  - You can buy them from tool companies such as Synopsys, Cadence Design Systems, etc.