Overview of Register-transfer Level Design Flow

Prof. Kurt Keutzer
EECS
keutzer@eecs.berkeley.edu

Goals of Course

Communicate the essence of the Register-Transfer Level synthesis design flow in a semester
  - Goal: “If Avanti, Cadence, and Synopsys were all abducted by aliens, their software could be recreated by this class.”
Prepare you for performing publishable research – aim high, a real publication!
Processing, Devices students – understand the tool flow, examine ways of bridging the gap between processing, design, and CAD
Circuits students – understand how the tools that you will be using for the rest of your life work
CAD students – give you foundation material for the field, prepare you for preliminary examinations
Theory types – understand how algorithms are applied in this algorithm rich area
Approach of the Course

Each week

- Examine a portion of the RTL design flow
- Identify one or more key problems
- Formulate the problem mathematically
- Solve the problem, examining trade-offs between
  - The computational efficiency of the algorithms
  - The quality/optimality of the result
- Look at contemporary practice
- See how close the classroom work approaches industrial practice

Course logistics

EECS 244
Cory 521, Tues, Thursday 3:30 – 5:00PM
Prof. Kurt Keutzer, Cory 566, Office hour: Thurs 2:30 – 3:30PM
keutzer@eecs.berkeley.edu - or by appointment
Instructor Michael Orshansky, Cory 463, Office hour: Friday 11-12 am
omisha@eecs.berkeley.edu - or by appointment
TA: Pinhong Chen, pinhong@cadence.com, office hours TBD
Exam 1: 30%
Exam 2: 30%
Final project: 40% (20% general content, 10% content in presentation, 10% content in written report)
Web page: http://www-cad.eecs.berkeley.edu/~pinhong/ee244/
Course notes in pdf on web page by noon day of lecture (on a good day!)
Project Outline

Motivation
Problem statement
Prior work
Investigative approach
Results
Summary
Conclusions
Future Work

Tips for a Great Project

Use your skill set
• Circuits, devices, processing, system-level applications

Great idea:
• Topical – e.g. system level, deep submicron effects, power
• Tractable – can make an impact in a semester, have all the software, examples, data files that you need

Get started early
Get mentorship (senior grad students, post-docs, prof of course)
Follow deadlines
Formulate the problem clearly
Formulate your results clearly
Some Successful 244 Projects


A System Level Project

``Bus Encoding to Prevent Crosstalk Delay'', B. Victor, K. Keutzer.

Problem:

- Delay in global wires is a large factor in overall circuit performance
- Capacitive coupling (cross-talk) between wires may greatly increase this delay

Solution:

- Old solutions: place shielding wires between signal wires
  - Large (2X) area penalty
- Proposed solution: use data encoding to eliminate coupling between neighboring bus lines
  - Smaller (1.25X) area penalty
A Circuit Level Project

“Getting to the Bottom of Deep Submicron”, D. Sylvester, K. Keutzer

Problem:
- Total chip wiring length increase dramatically
- It is claimed that wire delay contribution becomes dominant
- Traditional design flows (as will see) treat wire contribution as something secondary
- It is claimed that traditional design flows thus will fail in deep sub-micron

Solution:
- Need to perform careful analysis of local and global interconnect
- If design is done on blocks of 50K-100K gates, traditional flows will work

A Transistor and Process Level Project

“Impact of [...] Variability on Performance of High-speed Digital Circuits” M. Orshansky, L. Milor, P. Chen, K. Keutzer, C. Hu

Problem:
- Chip performance depends on path delay distribution and on critical path values
- Manufacturing of advanced technologies inevitably creates large variations in properties of transistors
- Neglecting this variability and uncertainty leads to underperforming and malfunctioning circuits

Solution
- A probabilistic model is derived to predict critical path degradation due to uncertainty
- Circuit and process-level fixes proposed for most critical factors
Possible Projects This Year

Efficient optimization algorithms for low-power design using dual-Vdd and Vth circuit design
- **Motivation:** By using 2nd Vdd and Vth can greatly reduce power consumption
- **Challenge:** How to optimally select gates and transistors to be high/low Vth/Vdd?

Cross-Talk Aware Static Timing Analysis
- **Motivation:** accuracy of standard timing analyzers is in danger due to interconnect cross-talk effects
- **Challenge:** incorporate good cross-talk models into STA, be computationally efficient

Possible Projects This Year

Statistical Static Timing Analysis
- **Motivation:** uncertainty about exact values of gate and wire delay (manufacturing variations, etc...). affects quality of static timing analysis results
- **Challenge:** incorporate statistical information into static timing analysis

Synthesis techniques for improved yield
- **Motivation:** circuit yield is influenced by circuit structure (e.g. cell layouts). Yield loss may be both due to particles as well as from manufacturing variations
- **Challenge:** propagate information about yield preferences higher in design cycle
**Possible Projects This Year**

**MESCAL – some background**

The Teepee framework allows users to design systems in a schematic editor. Components are designed using a constraint-based language we have developed. After designing the components, the Teepee environment allows for the automatic generation of the instruction set of the machine, a primitive assembler, and a simulator.

**Teepee PE RTL Code Generation**

- Implement Verilog code generation in the Teepee PE framework and push the resulting RTL description through traditional logic synthesis (i.e. Design Compiler). This would involve writing a visitor to walk the abstract syntax tree of the design and produce code.

**Possible Projects This Year**

**Teepee Syntax-Directed Editor**

- Implement an interactive editor that assists users in designing components in the Teepee PE framework. Details would include assisting the component writer in a syntax-directed manner, adding debugging capabilities, and helping in completing the definition of the language.

**Teepee Assembly View**

- Implement a view in the Teepee modeling environment that allows for the generation of an assembler. Preliminary work has been completed on building a simple parameterized assembler, but more work is needed to enhance the utility of the assembler. Debugging capabilities need to be implemented to assist assembly code writers and the compiler.
Key Dates

- Project teams + topics (1 paragraph) due 9/12/2002
- Full project proposals due 10/1/2002
- Exam 1 Handed out 10/3/2002
- Exam 1 collected at BEGINNING of class 10/8/2002
- Preliminary project report to be posted on class site 10/29/2002
- Exam 2 Handed out 10/31/2002
- Exam 2 collected at BEGINNING of class 11/5/2002
- Final project presentations TBD – between 12/6 and 12/13

The Inverted Pyramid

Electronic systems > $1Trillion
Semiconductor > $220B
CAD $3B
For example

![Investment Bar Chart]

Source: Dataquest

Where does CAD fit in?

![CAD in Silicon Foundries, EDA Industry, SEMiconductor Industry, Real World, Electronic Systems]

Electronic Systems
Silicon Foundries
EDA Industry
Semiconductor Industry
Real World
CAD at every level ...

Real World
Speech processing
Signal processing
Performance analysis

Electronic Systems
System synthesis
HW/SW co-design

Semiconductor Industry
Formal verification
Logic synthesis
Place and route
Circuit simulation

Silicon Foundries
Device simulation
Process modeling

Influencing CAD: Moore’s Law

Transistors

10M

1M

100K

10K

1K

100

10

1


Microprocessors

8080 68020 80386 8050

Pentium

Pentium Pro

MIPS R4000

80486

80486

10x/6 years
**NRTS: Chip Frequency (Ghz)**

![Graph showing chip frequency over time]

- On-chip, global clock, high performance
- On-chip, local clock, high-performance

---

**Increasing Device and Context Complexity**

Exponential increase in device complexity—increasing with Moore’s law (or faster)!

System context in which devices are deployed (e.g. cellular radio) are increasing in complexity as well exponential increases in design productivity

*We have exponentially more transistors!*
Deep Submicron Effects

Smaller geometries are causing a wide variety of effects that we have largely ignored in the past:
- Cross-coupled capacitances
- Signal integrity
- Resistance
- Inductance

Design of each transistor is getting more difficult!

Heterogeneity on Chip

Greater diversity of on-chip elements
- Processors
- Software
- Memory
- Analog

More transistors doing different things!
**Stronger Market Pressures**

- Decreasing design window
- Less tolerance for design revisions

*Exponentially more complex, greater design risk, greater variety, and a smaller design window!*

---

**A Quadruple-Whammy**

- Complexity
- Heterogeneity
- Time-to-Money
- DSM Effects
Role of CAD: Helping humans cope

How are we doing?

Source: SEMATECH
**Evolution of the EDA Industry**

- **1978**: Transistor entry - Calma, Computervision
- **1985**: Schematic Entry - Daisy, Mentor, Valid
- **1992**: Synthesis - Cadence, Synopsys

What's next?

- McKinsey S-Curve

**Design Process**

- **Design**: specify and enter the design intent
- **Verify**: verify the correctness of design and implementation
- **Implement**: refine the design through all phases
Application

Motivated by:

- A bright idea
- A market opportunity
  - An emerging market
  - A high growth market
- A technological breakthrough

For example - wireless telephony

World’s Cellular Subscribers

Will provide a ubiquitous infrastructure for wireless data as well as voice

Source: Ericsson Radio Systems, Inc.
**Specification**

Function, performance (power, delay, area), cost

- A competitor’s
  - Integrated circuit
  - Data sheet
- A napkin
- An industry standard

For example, GSM standard for cellular telephony
Targeting an IC Implementation

Mapping onto a system on a chip
Full Wireless Phone Organization

A more complex design – SOC architecture

Courtesy Synopsys
**Design Process**

*Design*: specify and enter the design intent

*Verify*: verify the correctness of design and implementation

*Implement*: refine the design through all phases

---

**Design Productivity by Approach**

<table>
<thead>
<tr>
<th>DOMAIN SPECIFIC</th>
<th>GATES/WEEK (Dataquest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEHAVIORAL</td>
<td>8K - 12K</td>
</tr>
<tr>
<td>RTL</td>
<td>2K - 10K</td>
</tr>
<tr>
<td>GATE</td>
<td>1K - 2K</td>
</tr>
<tr>
<td>TRANSISTOR</td>
<td>100 - 200</td>
</tr>
<tr>
<td></td>
<td>10 - 20</td>
</tr>
</tbody>
</table>
To Design, Implement, Verify 10M transistors

- Staff Months
  - 62.5
  - 125

Implementations here are often not good enough

- 625

Because implementations here are inferior/unpredictable

- 6250
- 62,500

Power
Delay
Area

Level of Acceptance of Synthesis Techniques

Behavioral Synthesis
Sequential Synthesis
Combinational Synthesis
Expert designer

Acceptance curve

Source: A. DeGeus
### Current Practice: HDL at RTL Level

```verilog
module foobar (q, clk, s, a, b);
    input clk, s, a, b;
    output q; reg q; reg d;
always @(a or b or s) // mux
    begin
        if( !s )
            d = a;
        else if( s )
            d = b;
        else
            d = 'bx;
    end // always @(a or b or s)
always @(clk) // latch
    begin
        if( clk == 1 )
            q = d;
        else if( clk !== 0 )
            q = 'bx;
    end // always @(clk)
endmodule
```

### RTL Level

An RTL description is always implicitly structural -
- the registers and their interconnectivity are defined
- Thus the clock-to-clock behavior is defined
- Only the control logic for the transfers is synthesized.

This approach can be enhanced:
- Register inferencing
- Automating resource allocation

**RTL implies**
- \[ a = b + c; \]
- \[ d = a + 1; \]
- \[ e = d * 2; \]

**Behavioral implies**
- \[ e = 2 * (b + c + 1) \]
**Verification**

**Design**: specify and enter the design intent

**Verify**: verify the correctness of design and implementation

**Implement**: refine the design through all phases

---

**Design Verification**

Library/module generators

- HDL
- RTL Synthesis
- Logic optimization
- Netlist
- Physical design
- Layout

Manual design

Specifyation

Is the design consistent with the original specification?

Is what I think I want what I really want?
Implementation Verification

Is the implementation consistent with the original design intent?
Is what I implemented what I wanted?

Manufacture Verification (Test)

Is the manufactured circuit consistent with the implemented design?
Did they build what I wanted?
Approaches to Design Verification

Formal verification
- Model checking - prove properties relative to model
- Theorem proving - prove properties of a circuit

Simulation
- Application of simulation stimulus to model of circuit

Emulation
- Implement a version of the circuit on emulator

Rapid prototyping
- Create a prototype of actual hardware

Verification using Emulation

System Hardware
- Customized parallel processor system for emulating logic
- In-circuit target interface

Software Compiler
- Mapping RTL & Gate designs to emulator

Runtime Software
- C-API
- Open SW architecture for tight integration
- Flexible modes of stimulus
**Rapid System Prototyping Environment**

- Need low-cost, instrument-like system prototyping environment
- Must be well-integrated into overall component-based flow

**Software Simulation**

- Simulation driver (vectors)
- Simulation model (HDL)
- Simulation monitor (yes/no)
Types of software simulators

Circuit simulation
- Spice, Advice, Hspice
- Timemill + Ace, ADM

Event-driven gate/RTL/Behavioral simulation
- Verilog - VCS, NC-Verilog, Turbo-Verilog, Verilog-XL
- VHDL - VSS, MTI, Leapfrog

Cycle-based gate/RTL/Behavioral simulation
- Verilog - Frontline, Speedsim
- VHDL - Cyclone

Domain-specific simulation
- SPW, COSSAP

Architecture-specific simulation

Implementation

**Design**: specify and enter the design intent

**Verify**: verify the correctness of design and implementation

**Implement**: refine the design through all phases
RTL Design Flow

Manual Design

Performed at
- Gate level (100 gates/week) /gate-level editor
- Transistor level (10 - 20 gates week)/tr level editor

Very expensive in design cost and design time -

Used for:
- Analog
- Leaf cells - libraries, memory cells
- Datapaths in high performance designs - DSP, microprocessor etc.
Module Generators

Parameterized generators of actual physical layout

Typically used for:
- Memories (word length, #words, # ports)
- Programmable logic arrays (PLA)
- Register files

Occasionally used for:
- Multipliers
- General-purpose datapath
- Datapaths in high performance designs - DSP, microprocessors etc.

RTL Synthesis Flow

Library → HDL
RTL Synthesis → netlist
Logic optimization → netlist
Physical design → layout
**Library**

Contains for each cell:
- Functional information: cell = a \( \times \) b \( \times \) c
- Timing information: function of
  - input slew
  - intrinsic delay
  - output capacitance
  - non-linear models used in tabular approach
- Physical footprint (area)
- Power characteristics

Wire-load models - function of
- Block size
- Wiring

---

**RTL Synthesis**

module fooobar (q, clk, s, a, b);
input clk, s, a, b;
output q;
reg q;
reg d;

always @(a or b or s) // mux
begin
  if(~s)
    d = a;
  else if(s)
    d = b;
  else
    d = 'bx;
end // always @(a or b or s)

translate HDL source into netlist
Logic Optimization

Perform a variety of transformations and optimizations
- Structural graph transformations
- Boolean transformations
- Mapping into a physical library

Technologies

“Closed book”: gate-array, standard-cell
“Open book”: CMOS Domino, complex gate static CMOS
Physical Design

Transform sequential circuit netlist into a physical circuit
- \textit{place} circuit components
- \textit{route} wires
- transform into a mask

Or for FPGA’s
- \textit{place} look-up tables
- \textit{route} wires

Channeled Gate Array
**Standard Cell Layout**

**Gordian Placement Flow**

J. Kleinhaus, G. Sigl, F. Johannes, K. Antreich,
GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,

Fig. 1. Data flow in the placement procedure GORDIAN.
Library, Netlist, and Aspect Ratio

Netlist - >100K cells from library

Size and aspect ratio of core die

Setting up Global Optimization

Fig. 1. Data flow in the placement procedure GORDIAN.
Resulting Layout

Partitioning

Fig. 1. Data flow in the placement procedure GORDIAN.
**Layout after Min-cut**

![Diagram of layout after Min-cut]

---

**Final Placement**

Fig. 1. Data flow in the placement procedure GORDIAN.
**Slicing Tree**


---

**Other details - slotting constraints**


Slotting constraints
Generating Final Placement

Fig. 1. Data flow in the placement procedure GORDIAN.

Standard Cell Layout
Routing

To simplify routing problem, divide it into two phases

- Global
- Detailed

Global routing

- Define routing regions
- Assign nets to regions

Detailed routing

- Route nets within each region
- Assign nets to pins

Global Routing

Provide guidance to detailed routing

Objective function application-dependent
**Global Routing**

Grid-Graph Model

Checker-Board Graph (also use slicing structure)

---

**Channel Routing**

Basic Terminology:

- tracks
- branch
- via
- trunk
- end
- net

Fixed pin positions on top and bottom edges

Classical channel: no nets leave channel

Three-sided channel possible
Two-Dimensional Compaction

X then Y 1D Compaction  Y then X 1D Compaction

Final Placement

Fig. 10. Macrocell design with standard cell blocks scb8 and scb9.
Re-visiting Verification

Design: specify and enter the design intent

Verify:
verify the correctness of design and implementation

Implement:
refine the design through all phases

Implementation Verification

Is the implementation consistent with the original design intent?

Is what I implemented what I wanted?
Implementation verification for ASIC’s

Apply gate-level simulation ("the golden simulator") at each step to verify functionality:

- 0-1 behavior on regression test set

and timing:

- maximum delay of circuit across critical paths

Advantages of gate-level simulation

- verifies timing and functionality simultaneously
- approach well understood by designers

Disadvantages of gate-level simulation

- computationally intensive - only 1 - 10 clock cycles of 100K gate design per 1 CPU second
- incomplete - results only as good as your vector set - easy to overlook incorrect timing/behavior
Alternative - Static Sign-off

Use static analysis techniques to verify:

- functionality: • formal equivalence-checking techniques
- and timing: • use static timing analysis

Problem: RTL to RTL Verification

After verification RTL may still be modified
- RTL level improvements for:
  - performance
  - power
  - area
  - testability

Need to verify that new RTL is correct
**Problem: RTL to Gates Verification**

Verify the gate level implementation is consistent with the RTL level design

Errors may have occurred due to
- RTL synthesis (heaven forbid!!)
- manual intervention

![Diagram](image1)

---

**Problem: Gates to Gates Verification**

Verify the modified gate level implementation is consistent with the RTL level design

Errors may have occurred due to
- Incorrect logic synthesis or module generation (heaven forbid!!)
- Test insertion
- Scan chain reordering
- Clock tree synthesis

![Diagram](image2)
**Physical Design**

Transform sequential circuit netlist into a physical circuit
- place circuit components
- route wires
- transform into a mask

**Problem: Layout to Gates Verification (LVS)**

Verify the layout level implementation is consistent with the gate level design
Errors may have occurred due to
- Errors in physical design tools
- Manual changes in layout
Verification is primarily graphical or "topological"
Solving Layout to Gates Verification (LVS)

Extract gate level models from physical level
Graphically compare extracted model against gate-level schematic (layout versus schematic)
Flag any discrepancies

Solving Gates to Gates Verification

``specification''

implementation
Register Extraction

``specification''

implementation

Combinational Comparison

``specification''

implementation
Combinational Equivalence Checking

Presumes equivalence-relation given (or discovered) between sequential circuits

Approaches
- Canonical forms - bdd’s and variants
- Test-oriented methods
  - static, dynamic learning
- symbolic manipulation
  - graph isomorphism
  - structural reductions

These techniques form the foundation of implementation verification

Solving RTL-to-Gates Verification

Step 1: (formally) translate HDL source into netlist
Step 2: Perform gates-to-gates verification
Solving RTL-to-RTL Verification

Step 1:
(formally)
translate both HDL sources into netlists

Step 2:
perform gate-to-gate verification on netlists

Alternative - Static Sign-off

Use static analysis techniques to verify:
functionality:
• formal equivalence-checking techniques

and timing:
• use static timing analysis
Purpose of Static Timing Verification

• determine fastest permissable clock speed (e.g. 100MHz) by determining delay (including set-up and hold time) of longest path from register to register (e.g. 10ns.)

• largely eliminates need for gate-level simulation to verify the delay of the circuit

Elements of Timing Verification - 1

Gate and interconnect delay-modeling - a function of
- Intrinsic gate delay
- Input slew rate
- Output capacitance -
  - Estimated
  - Extracted
- Process, voltage, temperature - including variance
Clocking issues
- Regimes: single-phase, two-phase, multi-phase
- overlapping, non-overlapping
- qualified clocks, clock skew

Delay calculation procedure
- Longest graphical path
- Longest true delay
Method of calculation
- “Batch mode”
- Incremental
**Manufacture Verification (Test)**

Is the manufactured circuit consistent with the implemented design?

Did they build what I wanted?

---

**Test Synthesis**

Full-chip test requires:

- BIST
- JTAG 1149.1
- Fault simulation for asynchronous interfaces
- Partial scan

---
Current Status of RTL Design Flow

Current RTL design flow is able to produce

- High speed microprocessors - e.g. Alpha, Pentium Pro > 1M gate-equivalents > 1GHz. (with intervention)
- System-on-a-chip/Systems on silicon
  - Integration of micro-p, DSP, memory and ASIC on a single die > 1-10M gate-equivalents >200MHz.
- Rapid turnaround ASIC
  - 200K gate ASICs design in:
    - 1 week - Motorola CSIC
    - 3 months