Lecture 10

Design Methodologies and Tools

Konstantinos Masselos
Department of Electrical & Electronic Engineering
Imperial College London

URL: http://cas.ee.ic.ac.uk/~kostas
E-mail: k.masselos@ic.ac.uk
Based on slides/material by...

- J. Rabaey http://bwrc.eecs.berkeley.edu/Classes/IcBook/instructors.html

Recommended Reading


- W. Wolf, “Modern VLSI Design: System-on-Chip Design”: Chapters 7, 8, 10
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Digital IC Implementation Approaches

- Custom
  - Cell-Based
    - Standard Cells
    - Compiled Cells
  - Array-Based
    - Macro Cells
    - Pre-diffused (Gate Arrays)
    - Pre-wired (FPGA)

- Semi-custom
Design Methodology

- Design process traverses iteratively between three abstractions: behavior, structure, and geometry
- More and more automation for each of these steps
The Design Problem

A growing gap between design complexity and design productivity

Source: sematech97
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Every company has its own design methodology.
Methodology depends on:
- type of chip
- size of chip
- design time constraints
- cost/performance
- available tools
Generic Design Flow

1. Detailed specs
   - architectural design
   - floorplan
   - functional/performance verification
   - testability

2. Register-transfer design

3. Logic design

4. Layout

5. Circuit design

6. Tapeout
IBM ASIC design flow

- Design entry: Usually VHDL or Verilog. Schematic is also supported.
- Logic synthesis with IBM tools targeting IBM cell library
- Simulation: either at the functional or gate level. Gate level netlists can be back annotated with timing information for delay simulation
- Floorplanning: used to estimate wiring capacitance, area and wiring congestion
- Test structure verification ensures that the design satisfies a set of IBM-defined rules that ensure the design – including RAM – is in compliance with the requirements for LSSD (level sensitive scan design)
- Static timing analysis analyzes the worst clock speed for the implementation
- Formal verification: the design is checked for equivalence with a boolean specification using efficient algorithms for solving the boolean equivalence problem
- CMOS checks: Checks fan out, I/O boundary scan and other low level issues
IBM ASIC design flow

- Design hand off: when the design is ready for physical design, a netlist, timing assertion data, pad placement, and floorplanning information are given to the IBM design center.

At this point physical design is handled by the manufacturing center:

- Front end processing. Clock trees and test logic are generated at this step and static timing analysis is used to check performance.
- Prelayout sign off: ensures that no errors have been introduced by front end processing.
- Layout is performed by automatic tools guided by professional designers. Layout can be performed either on a flat or hierarchical design.
- Post layout sign off: verifying the logic and timing at this step ensures that no errors were introduced during layout.
- Tape out to manufacturing. ATPG is used to generate test vectors and the mask data is generated and sent to the manufacturing line.
Tools aren’t very useful if they don’t talk to each other.

Design interchange languages:
- VHDL (TM), Verilog (TM) (function and structure);
- EDIF (netlists);
- GDS, CIF (masks).
CAD Tool Interactions

database (hub-and-spoke)
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Design Analysis and Verification

- Accounts for largest fraction of design time
- More efficient when done at higher levels of abstraction - selection of correct analysis level can account for multiple orders of magnitude in verification time
- Two major approaches:
  - Simulation
  - Verification
Digital Data treated as Analog Signal

Circuit Simulation

Both Time and Data treated as Analog Quantities
Also complicated by presence of non-linear elements
(relaxed in timing simulation)
Representing Data as Discrete Entity

Discretizing the data using switching threshold

The linear switch model of the inverter
Circuit versus Switch-Level Simulation

The figure illustrates the comparison between circuit-level and switch-level simulation. The top graph shows the time-domain behavior of signals, with labels for inputs and outputs. The bottom graph provides a detailed view of the switch-level simulation, highlighting the state transitions over time.
Delay Models

- **Unit-delay simulators** assume that each component has a one-unit delay. Model function but not performance.
- **Variable-delay simulators** allow each component to have its own delay. Accuracy of performance estimates from variable-delay simulators depends on how well circuits can be extracted to digital model.
Structural Description of Accumulator

entity accumulator is
  port ( -- definition of input and output terminals
    DI: in bit_vector(15 downto 0) -- a vector of 16 bit wide
    DO: inout bit_vector(15 downto 0);
    CLK: in bit
  );
end accumulator;

architecture structure of accumulator is
  component reg -- definition of register ports
    port(
      DI : in bit_vector(15 downto 0);
      DO : out bit_vector(15 downto 0);
      CLK : in bit
    );
  end component;
  component add -- definition of adder ports
    port(
      IN0 : in bit_vector(15 downto 0);
      IN1 : in bit_vector(15 downto 0);
      OUT0 : out bit_vector(15 downto 0)
    );
  end component;
  -- definition of accumulator structure
  signal X : bit_vector(15 downto 0);
  begin
    add1 : add
      port map (DI, DO, X); -- defines port connectivity
    reg1 : reg
      port map (X, DO, CLK);
  end structure;

Design defined as composition of register and full-adder cells ("netlist")

Data represented as \{0,1,Z\}

Time discretized and progresses with unit steps

Description language: VHDL
Other options: schematics, Verilog
Design described as set of input-output relations, regardless of chosen implementation

Data described at higher abstraction level ("integer")
Behavioral simulation of accumulator

Discrete time

Integer data

(Synopsys Waves display tool)
Timing Verification

Critical path

Enumerates and rank orders critical timing paths

No simulation needed!

(Synopsys-Epic Pathmill)
Issues in Timing Verification

![Diagram of a 4-bit adder and MUX with bypass connections.]

False Timing Paths
Timing Analysis

- Unlike simulation, timing analysis is value-independent—doesn’t require specifying inputs.
- Simulation can be optimistic—you may not apply worst-case input vector.
- Timing analysis can be pessimistic, but that is safer than optimistic.

Timing analysis procedure
- Two major steps:
  - build graph with elemental delays
  - traverse graph to find longest path
- Must model 0-1 and 1-0 delays independently for more accurate total delay.
- Use value analysis to prune impossible paths.
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- **Design creation**
  - Register Transfer design
  - Hardware Description Languages
  - Floorplanning
  - Layout design
  - Logic synthesis and state machine optimization
  - High level synthesis and hardware/software codesign
Automatic Cell Generation

Random-logic layout generated by CLEO cell compiler (Digital)
Macrocell Design Methodology

Floorplan:
Defines overall topology of design, relative placement of modules, and global routes of busses, supplies, and clocks
Module Generators — Compiled Datapath

Advantages: One-dimensional placement/routing problem
### Taxonomy of Synthesis Tasks

<table>
<thead>
<tr>
<th>Behavioral View</th>
<th>Architectural Level</th>
<th>Logic Level</th>
<th>Circuit Level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(i: 1..16) ::</strong> sum = sum*(z^{-1}) + coeff[i]*(In)(z^{-1})</td>
<td>Architecture Synthesis</td>
<td>Logic Synthesis</td>
<td>Circuit Synthesis</td>
</tr>
</tbody>
</table>

```
state
```

\[ a \quad b \quad c \quad x \]

\[ \text{mem} \quad \text{fsm} \quad D> \quad \text{}\quad t_p \quad \text{c} \quad \text{2} \quad \text{2} \quad \text{2} \quad \text{2} \quad \text{4} \]

---

**Design Methodologies and Tools**

**Introduction to Digital Integrated Circuit Design**

**Lecture 10 - 31**
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- **Register Transfer design**
- Hardware Description Languages
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Register-Transfer Design

- A register-transfer system is a sequential machine.
- Register-transfer design is structural—complex combinations of state machines may not be easily described solely by a large state transition graph.
- Register-transfer design concentrates on functionality, not details of logic design.
- A register-transfer machine has combinational logic connecting registers:
Register-Transfer Simulation

- Simulates to clock-cycle accuracy. Doesn’t guarantee timing.
- Important to get proper function of machine before jumping into detailed logic design. (But be sure to take into account critical delays when choosing register-transfer organization.)

- Hardware description languages are typically supported by a simulation system: VHDL, Verilog, etc.
  - Simulation engine takes care of scheduling events during simulation.
- Can hand-code a simulation in a programming language.
  - Must be sure that register-transfer events happen in proper order.
Data Path-Controller Systems

- One good way to structure a system is as a data path and a controller:
  - data path executes regular operations (arithmetic, etc.), holds registers with data-oriented state
  - controller evaluates irregular functions, sets control signals for data path

- Data and control are equivalent
- We can rewrite control into data and visa versa:
  - control: if $i_1 = '0'$ then $o_1 <= a$; else $o_1 <= b$; end if;
  - data: $o_1 <= ((i_1 = '0') \text{ and } a) \text{ or } ((i_1 = '1') \text{ and } b)$;
- Data/control distinction is useful but not fundamental
Data Operators

- Arithmetic operations are easy to spot in hardware description languages:
  - \( x <= a + b; \)
- Multiplexers are implied by conditionals. Must evaluate entire program to determine which sources of data for registers.
- Multiplexers also come from sharing adders, etc.
if x = '0' then
    reg1 <= a;
else
    reg1 <= b;
end if;

register-transfer
Alternate Data Path - Controller Systems

one controller, one data path

two communicating data path-controller systems
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- **Hardware Description Languages**
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Hardware Description Languages

- Textual languages for describing hardware:
  - structure
  - function
- Two major HDLs designed for simulation:
  - VHDL
  - Verilog
    - Similar capabilities but somewhat different language philosophies.
- EDIF is a standard netlist format
Simulation vs. Programming

- Simulation tags computations with times.
  - Must know when signals change to properly simulate hardware.
- Simulation is parallel.
  - Many statements can execute at the same (simulation) time.
  - Just like hardware.
Types of Simulation

- Compiled code simulation.
  - Generate program that evaluates a hardware block.
  - Operational details within the hardware block are lost.

- Event-driven simulation.
  - An event is a change in a net's value.
  - An event has two components:
    - Value
    - Time
  - Propagate events through simulation.
  - Don’t simulate a block until its inputs change.
Timewheel

- The timewheel is a data structure in the simulator that efficiently determines the order of events processed.
- Events are placed on the timewheel in time order.
- Events are taken out of the head of the timewheel to process them in order.
- Order of evaluation is important.
  - Causality must be obeyed.
- Evaluating events in the wrong order can cause inaccurate results.
Modeling

- Structural modeling describes the connections between components.
  - Netlists are structural models.
- Behavioral models describes the functional relationship between inputs and outputs.
  - Similar to programming but values are events.
Testbenches

- A testbench is a model used to exercise a simulation.
  - Provides stimulus.
  - Checks outputs.
- Testbenches help automate design verification.
  - Rerun edited module against testbench.
  - Run models at behavioral, RTL levels against the same testbench.
Synthesis Subsets

- VHDL and Verilog were designed for simulation.
- A synthesis subset is:
  - Synthesizable.
  - produces consistent simulation results.
- Different tools may use different synthesis subsets.
Register-Transfer Synthesis

- Most common type of synthesis.
- Synthesizes gates from abstract RT model.
  - Registers are explicit.
  - Some tools will infer storage elements---be careful.
- Optimized for performance, area, power.
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- **Floorplanning**
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Floorplanning Strategies

- Floorplanning must take into account blocks of varying function, size, shape.
- Must design:
  - space allocation
  - signal routing
  - power supply routing
  - clock distribution
Purposes of Floorplanning

◆ Early in design:
  • Prepare a floorplan to budget area, wire area/delay. Tradeoffs between blocks can be negotiated.

◆ Late in design:
  • Make sure the pieces fit together as planned.
  • Implement the global layout.
Block Placement

- Blocks have:
  - area
  - aspect ratio
- Blocks may be placed at different rotations and reflections.
- Uniform size blocks are easier to interchange.
Blocks and Wiring

- Cannot ignore wiring during block placement—large wiring areas may force rearrangement of blocks.
- Wiring plan must consider area and delay of critical signals.
- Blocks divide wiring area into routing channels.
Channels and Switchboxes

channel

switchbox

switchbox pins
Channel Definition

- Channels end at block boundaries.
- Several alternate channel definitions are possible:
Global Routing

- Goal: assign wires to paths through channels.
- Don’t worry about exact routing of wires within channel.
- Can estimate channel height from global routing using congestion.
Channel Utilization

- Want to keep all channels about equally full to minimize wasted area.
- Important to route time-critical signals first.
- Shortest path may not be best for global wiring.
- In general, may need to rip-up wires and reroute to improve the global routing.
Switchbox Routing

- Can’t expand a switchbox to make room for more wiring.
- Switchbox may be defined by intersection of channels.
- Switchboxes frequently need more experimentation with wiring order because nets may block other nets.
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- Floorplanning
- **Layout design**
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Transistor Sizing

- Once transistor-level critical path has been identified, transistors can be sized to optimize delay.
- Transistor sizing is cast as optimization problem to meet performance goal while minimizing total active area.
Layout Synthesis

- Two critical phases of layout design:
  - placement of components on the chip;
  - routing of wires between components.
- Placement and routing interact, but separating layout design into phases helps us understand the problem and find good solutions.
Placement Metrics

- Quality metrics for layout:
  - area
  - delay
- Area and delay determined in part by wiring.
- How do we judge a placement without wiring? Estimate wire length without actually performing routing.
Wire Length as a Quality Metric

bad placement

good placement
Placement Techniques

- Can construct an initial solution, improve an existing solution.

- Pairwise interchange placement
  - simple improvement metric:
  - Interchange a pair, keep the swap if it helps wire length.
  - Heuristic determines which two components to swap.

- Placement by Partitioning
  - Works well for components of fairly uniform size.
  - Partition netlist to minimize total wire length using min-cut criterion.
  - Partitioning may be interpreted as 1-D or 2-D layout.
Min-Cut Bisecting Partitioning

Partition 1

Partition 2

1 net
3 nets
More Partitioning Algorithms

- Kernighan-Lin Algorithm
  - Compute min cut criterion:
    - \( \text{count total net cut change} \)
  - Algorithm exchanges sets of nodes to perform hill-climbing—finding improvements where no single swap will improve the cut
  - Recursively subdivide to determine placement detail

- Simulated Annealing
  - Powerful but CPU-intensive optimization technique.
  - Analogy to annealing of metals:
    - temperature determines probability of a component jumping position;
    - probabilistically accept moves.
    - start at high temperature, cool to lower temperature to try to reach good placement.
Routing

- Major phases in routing:
  - global routing assigns nets to routing areas.
  - detailed routing designs the routing areas.
- Net ordering is a major problem. Order in which nets are routed determines quality for result. Net ordering is a heuristic.
Maze Routing

- Will find shortest path for a single wire, if such a path exists.
- Two phases:
  - Label nodes with distance, radiating from source.
  - Use distances to trace from sink to source, choosing a path that always decreases distance to source.
Detailed Routing

- **Dogleg router** breaks net into multiple segments as needed.
- Try to minimize number of dogleg segments per net to minimize congestion for future nets.
- One good heuristic—use left-edge criterion on each dogleg segment to fill up the channel.
More Routing Algorithms

◆ Rivest-Fiduccia Channel Router
  • Routes from left to right. Assigns all nets that cross the current column to tracks.
  • Heuristics:
    ➫ Make connections to pins.
    ➫ Add jogs to put multi-track net into one track.
    ➫ Add jogs to reduce distance in multi-track nets.
    ➫ Add jogs to move net toward next pin.
    ➫ Add tracks when necessary.

◆ YACR2
  • Tries to minimize number of vias as well as number of tracks.
  • Temporarily satisfies vertical constraints by adding blank space between pins.
  • Eliminates blank space after by adding jobs.
  • May route in both directions on same layer.
Layout Analysis

- Test design rules using Boolean combinations of masks, grow/shrink.

![Diagram showing M1 and M2 masks with notation not (M1 or M2)]
Scan Line Algorithm

- Mark each edge of polygon with direction.
- Sweep scan line across layout.
- At each point on scan line, count number of left-hand and right-hand edges to determine what rectangle that point is in.
Back Annotation

- Often want to iteratively improve design.
- Back annotation updates a more-abstract design with information from later design stages.
  - Example: annotate logic schematic with extracted parasitic Rs and Cs.
- Back annotation requires tools to know more about each other.
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- High level synthesis and hardware/software codesign
Logic Synthesis

- Goal: create a logic gate network which performs a given set of functions.
- Input is Boolean formulas; gates also implement Boolean functions.
- Logic synthesis:
  - maps onto available gates
  - restructures for delay, area, testability, power, etc.

- Logic Synthesis Phases
  - Technology-independent optimizations work on logic representations that do not directly model logic gates.
  - Technology-dependent optimizations work in the available set of logic gates.
  - Transformation from technology-independent to technology-dependent is called library binding.
A Boolean network is the main representation of the logic functions for technology independent optimizations.

Each node can be represented as sum-of-products (or product-of-sums).

Provides multi-level structure, but functions in the network need not correspond to logic gates.
Terms

- **Support**: set of variables used by a function.
- **Transitive fanout**: all the primary outputs and intermediate variables of a function.
- **Transitive fanin**: all the primary inputs and intermediate variables used by a function. Transistive fanin determines a cone of logic.
Technology-Independent Logic Optimization

- **Simplification** rewrites node to simplify its form.
- **Network restructuring** introduces new nodes for common factors, collapses several nodes into one new node.
- **Delay restructuring** changes factorization to reduce path length.

**Cost in boolean network:** How to judge whether an operation has improved the network?

- Don’t know exact gate structure, but can estimate final network cost:
  - area estimated by number of literals (true or complement forms of variables)
  - delay estimated by path length
Simplification

- Rewrites a node to reduce the number of literals in the node.
- Function defined by:
  - on-set: set of inputs for which output is 1
  - off-set: set of inputs for which output is 0
  - don’t-care-set: set of inputs for which output is don’t-care.

- Each way to write a function as a sum-of-products is a cover since it covers the on-set.
- A cover is composed of cubes—product terms which define a subspace cube in the function space.

- Partially-Specified Functions
  - Don’t-cares can be implemented in either the on-set or off-set.
  - Don’t-cares provide the greatest opportunities for minimization in many cases.
Lecture 10 - 79
Introduction to Digital Integrated Circuit Design

**Espresso**

- Well-known two-level logic optimizer.
- Espresso optimization loop:
  - expand
  - make irredundant
  - reduce.
- Optimization loop is designed to refine cover to reduce its size.
Don’t-Cares in Boolean Networks

- In two-level function, don’t-cares are defined at primary output.
- In Boolean network, structure of network itself introduces don’t-cares.
- Types of structural don’t-cares:
  - **Satisfiability**: Occur when an intermediate variable value is inconsistent with its function inputs. Since this can’t happen, we don’t care.
  - **Observability**: Occur when an intermediate variable’s value doesn’t affect the network primary outputs.
Partial Collapsing

before

after
Factorization

Based on division:
- formulate candidate divisor
- test how it divides into the function
- if \( g = f/c \), we can use \( c \) as an intermediate function for \( f \)

Algebraic division: don’t take into account Boolean simplification. Less expensive then Boolean division.

Three steps of factorization based on algebraic or Boolean division:
- generate potential common factors and compute literal savings if used
- choose factors to substitute into network
- restructure the network to use the new factors

Algebraic/Boolean division can be used to implement first step.
Factorization for Delay

- Remove factors from critical path, add them off critical path:
Library Binding

- Also known as technology mapping.
- Rewrites Boolean network in terms of available logic functions.
- Can optimize for both area and delay.
- Can be viewed as a pattern matching problem. Tries to find pattern match which minimizes area/delay cost.

Technology mapping procedure
- Write Boolean network in canonical NAND form.
- Write each library gate in canonical NAND form. Assign cost to each library gate.
- If network is a tree, can use dynamic programming to select minimum-cost cover of network by library gates.
Breaking into Trees

- Not optimal, but reasonable cuts usually work OK.
Technology Mapping Example

- Try to cover tree from primary inputs to primary outputs.
- Proceed one gate at a time. At the next level, select minimum-cost cover at that point.
- Latest selection may require removing some earlier matches.
Sequential Machine Optimizations

- State assignment: choose codes for states.
- Create common factors in states by assigning them close codes:
  - \( s_0 = 000 \), \( s_1 = 001 \) (\( x_0 x_1 x_2 \));
  - \( s_0 + s_1 = x_0 x_1 \).

- Kiss: minimizes symbolic state machine to find states which should be coded close together.

- Coding procedure
  - Symbolic minimization creates constraints on coding—sets of states which should be coded closely together.
  - If all constraints cannot be satisfied in minimum number of bits, can add bits to code to allow constraints to be satisfied.
Outline

- Introduction
- Design flows and CAD tools
- Design analysis and verification
- Design creation
- Register Transfer design
- Hardware Description Languages
- Floorplanning
- Layout design
- Logic synthesis and state machine optimization
- **High level synthesis and hardware/software codesign**
High Level Synthesis - Scheduling and Binding

- High-level synthesis systems translate behavioral descriptions into register-transfer implementations by scheduling and binding.
- Most high-level synthesis systems concentrate on data paths.
Force-Directed Scheduling

- Compute schedulable range of each operator by computing ASAP, ALAP schedules.
- **Distribution cost** = cost of operator \( \times \) probability it will be scheduled in that step.
- **Force** = difference between distribution at a control step and average distribution.
Scheduling Extensions

- **Chaining**: executing several operators in one cycle to balance path lengths.
- **Multicycling**: executing a pipelined operator over several cycles.
Binding

- Many resources must be bound: operators, variables, etc.
- Many binding problems can be formulated as clique partitioning.
- Good heuristic algorithms exist to solve clique partitioning problems.
Clique Partitioning

- Node in graph is an element to be allocated.
- Edge is added to graph between two nodes if those elements can be allocated to the same element.
- Clique: fully-connected set of nodes.
- Example: two + scheduled to different control steps can share a single adder since they do not execute at the same time.
Hardware/Software Co-Design

- Use programmable CPUs along with specialized logic blocks to implement a particular application.
- CPUs can implement background functions much more efficiently than dedicated logic: higher utilization of logic.
- Important in systems-on-silicon.
Co-Synthesis Styles

- **Hardware/software partitioning:**
  - Architectural template consists of 1 CPU + n ASICs.
  - Put operations on CPU or ASIC.

- **Distributed system synthesis:**
  - No fixed architectural template.
  - Allocate function units and communication, schedule operations.