Logic Synthesis and Verification

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University

Fall 2014

Course Info

Instructor

Jie-Hong Roland Jiang
email: jhjiang@ntu.edu.tw
office: 242, EEII
phone: (02)3366-3685
office hour: 16:00-18:00 Thu

Course webpage

http://cc.ee.ntu.edu.tw/~jhjiang/instruction/courses/fall14-lsv/lsv.html

Email contact

Your official NTU email addresses will be used for future contact

Grading Policy

Grading rules

- Homework 30%
- Programming assignments 10%
- Midterm exam 25%
- Final quiz 5%
- Project 30%
  - Presentation + report

Homework
- discussions encouraged, but write down solutions individually and separately
- due one week from the problem set is out except for programming assignments (due date will be specified)
- 20% off per day for late homework
- 6 homework assignments (peer-review grading)

Midterm
- in-class exam (schedule may/may not differ from the academic calendar)

Project
- oral presentation, final report

Report grading errors within one week after receiving notice.
Plagiarism and cheating are strictly prohibited (no credits for plagiarism).

References

- J.-H. R. Jiang and S. Devadas. *Logic Synthesis in a Nutshell*. (Chapter 6 of *Electronic Design Automation: Synthesis, Verification, and Test*), Elsevier 2009.
  - Downloadable handout
- F. M. Brown. *Boolean Reasoning: The Logic of Boolean Equations.* Dover, 2003.
  - Used in the introduction to Boolean algebra
- S. Hassoun and T. Sasao. *Logic Synthesis and Verification*. Springer, 2001.
- G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Springer, 2006.
- W. Kunz and D. Stoffel. *Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques*. Springer, 1997.

References (cont'd)

- Papers on course webpage
- Conference Proceedings
  - ACM/IEEE Design Automation Conference (DAC)
  - IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD)
  - DATE, ASP-DAC
  - Computer-Aided Verification (CAV)
  - TACAS, FMCAD
- Journals
  - IEEE Trans. on Computer-Aided Design
  - IEEE Trans. on Computers

Introduction

Reading:

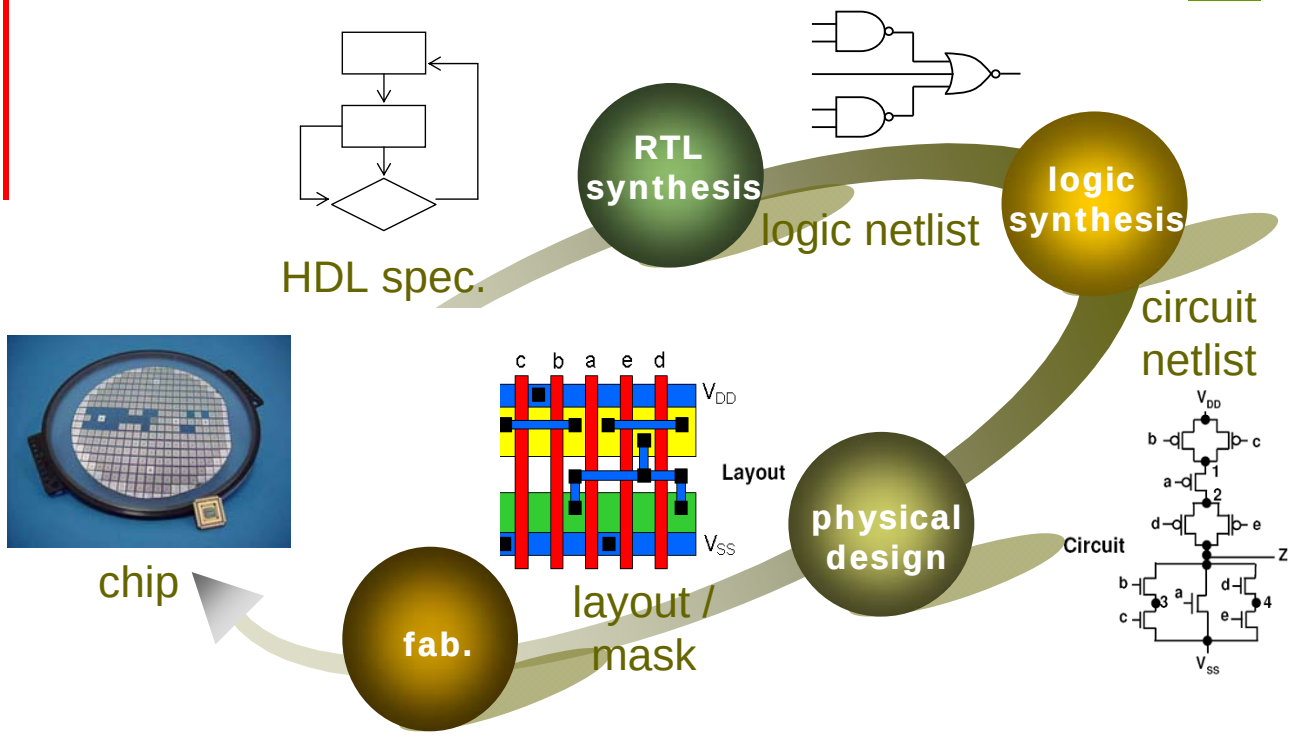
Logic Synthesis in a Nutshell

Section 1

Evolving Information Technology

- The Industrial Revolution
  - Application of power-driven machinery to manufacturing (1750 – 1830)
- IT Revolution
  - Application of electronic devices to information processing
    (1950 – present)
- Electronic systems evolve in a fascinating speed
  - Design challenges emerge and design paradigms shift in this evolution
  - EDA tools change along the evolution

IC Design Flow

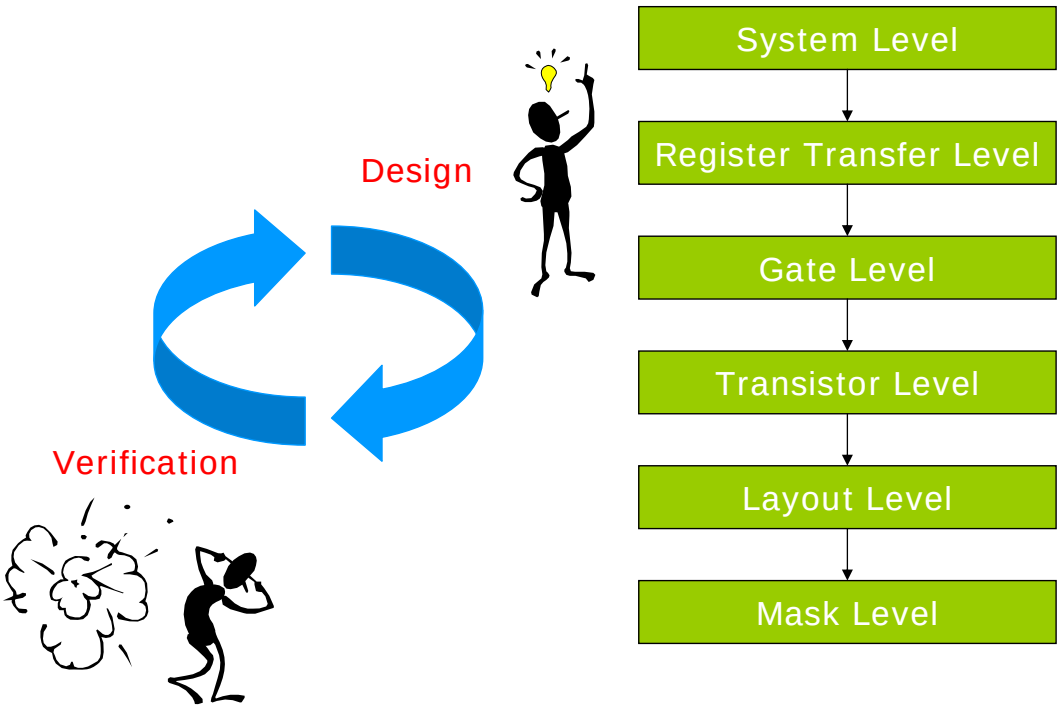


Electronic Design Automation

- EDA tools aim at automating electronic system design and optimizing **general** design instances (not just some specific design)

- EDA is a field with rich applications from electrical engineering, computer science, and mathematics
  - Electronics, circuit theory, communication, DSP, device physics, ...
  - Algorithms, complexity theory, automata theory, logics, games, ...
  - Probability, statistics, algebra, numerical analysis, matrix computation, ...
- EDA is one of the most advanced areas in practical computer science
  - Many problems require sophisticated mathematical modeling
  - Many algorithms are computationally hard, and require advanced heuristics to work on realistic problem sizes
- EDA is a very good workplace for software engineers
  - E.g., modern SAT solvers (GRASP, Chaff, BerkMin, MiniSAT) are developed in the field of EDA

VLSI Design Flow & Abstraction Levels



System Level

- Abstract algorithmic description of high-level behavior
  - E.g., C-programming language

```
Port*
compute_optimal_route_for_packet(Packet_t *packet,
                                 Channel_t *channel)
{
  static Queue_t *packet_queue;

  packet_queue = add_packet(packet_queue, packet);
  ...
}
```

  - abstract because it does not contain any implementation details for timing or data
  - efficient to get a compact execution model as first design draft
  - difficult to maintain throughout project because no link to implementation

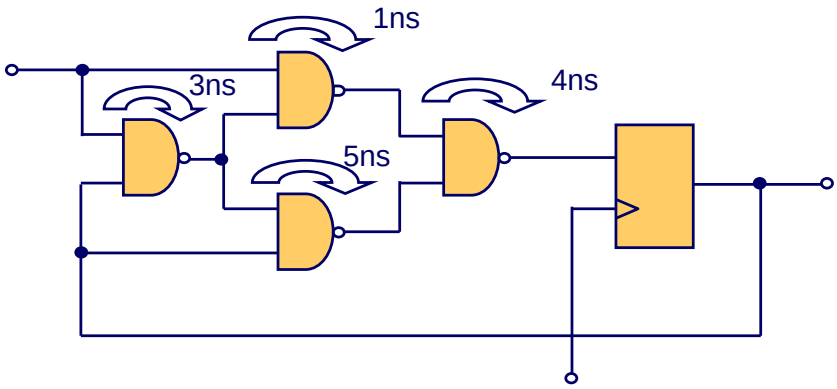
Register Transfer Level

- Cycle accurate model “close” to the hardware implementation
  - bit-vector data types and operations as abstraction from bit-level implementation
  - sequential constructs (e.g. if - then - else, while loops) to support modeling of complex control flow

```
module mark1;
reg [31:0] m[0:8192];
reg [12:0] pc;
reg [31:0] acc;
reg[15:0] ir;
always
   begin
     ir = m[pc];
     if(ir[15:13] == 3b’000)
        pc = m[ir[12:0]];
     else if (ir[15:13] == 3’b010)
        acc = -m[ir[12:0]];
     ...
   end
endmodule
```

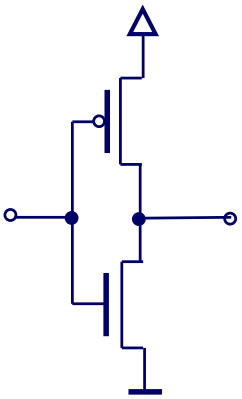
Gate Level

- Model on finite-state machine level
  - Functions represented in Boolean logic using registers and gates
  - Various delay models for gates and wires



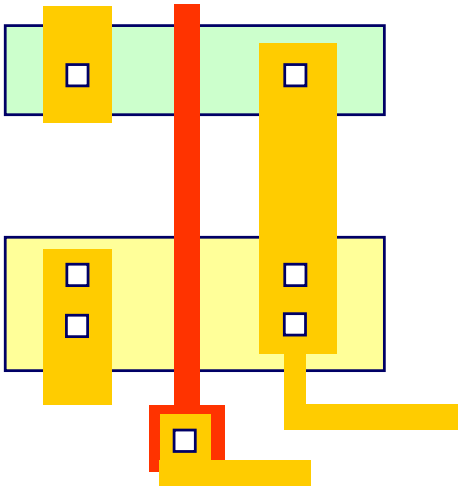
Transistor Level

- Model on CMOS transistor level
  - Binary switches used for function modeling
    - E.g., in functional equivalence checking
  - Differential equations used for circuit simulation
    - E.g., in timing/waveform analysis

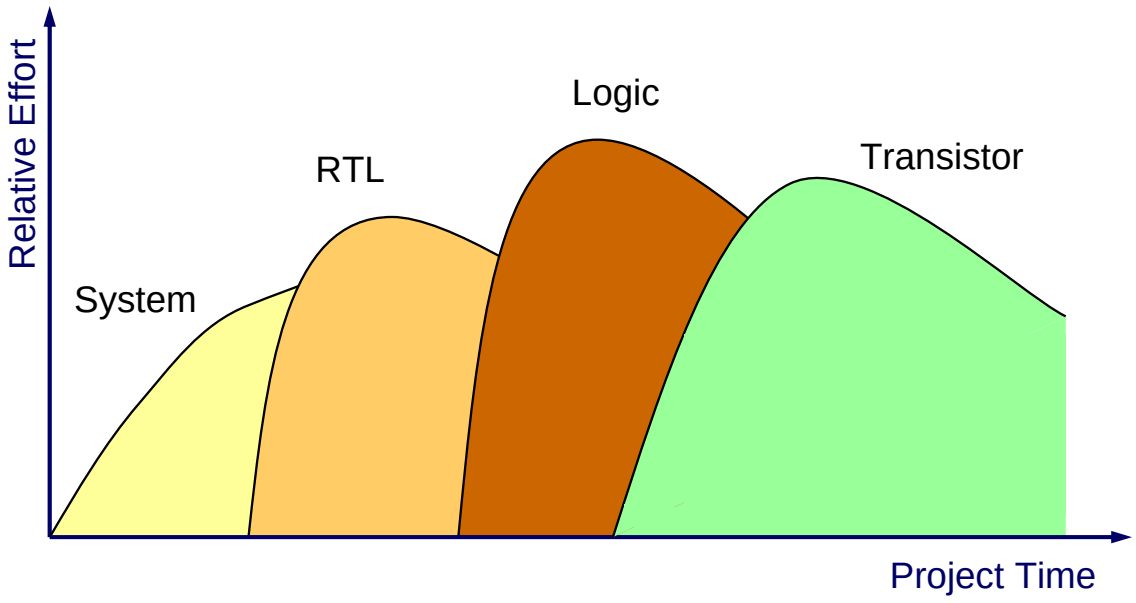


Layout Level

- Transistors and wires are laid out as polygons in different technology layers such as diffusion, poly-silicon, metal, etc.



Integrated System Design



General Design Approaches

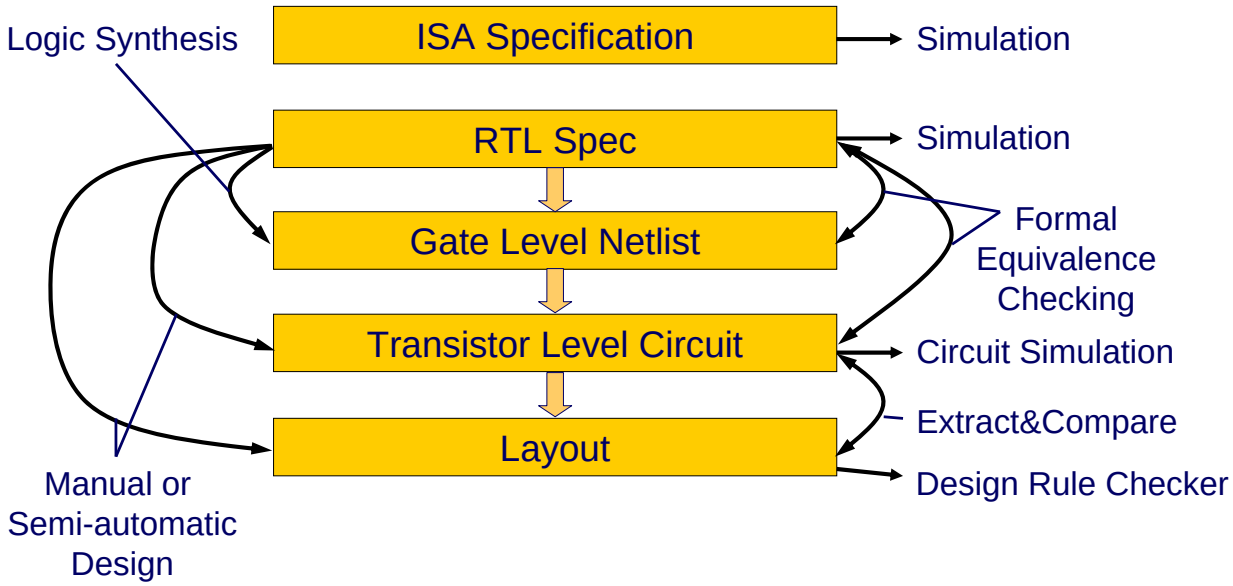
- Divide and conquer !
  - partition design problem into many sub-problems which are manageable
  - define mathematical model for sub-problem and find an algorithmic solution
    - beware of model limitations and check them !
  - implement algorithm in individual design tools, define and implement general interfaces among the tools
  - implement checking tools for boundary conditions
  - concatenate design tools to general design flows which can be managed
  - see what doesn't work and start over

Full Custom Design Flow

- Application: ultra-high performance designs
  - general-purpose processors, DSPs, graphic chips, internet routers, game processors, etc.
- Target: very large markets with high profit margins
  - e.g. PC business
- Complexity: very complex and labor intense
  - involving large teams
  - high up-front investments and relatively high risks
- Role of logic synthesis:
  - limited to components that are not performance critical or that might change late in design cycle (due to design bugs found late)
    - control logic
    - non-critical data-path logic
  - bulk of data-path components and fast control logic are manually crafted for optimal performance

Full Custom Design Flow

(incomplete picture)

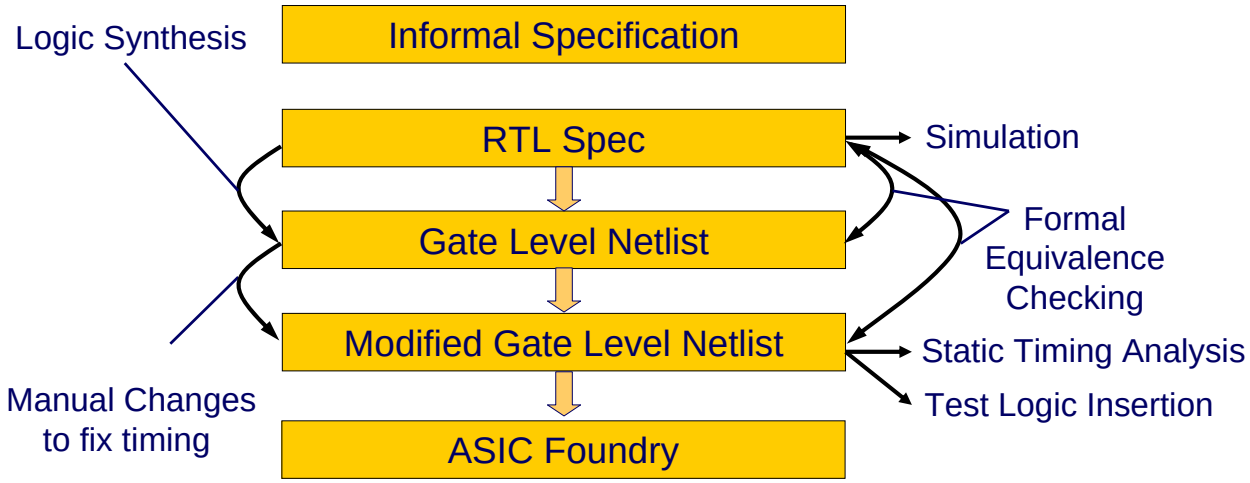


ASIC Design Flow

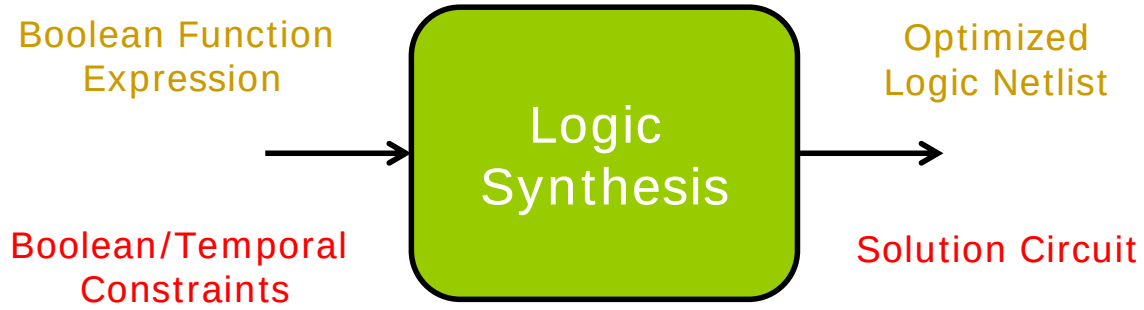
- Application: general IC market
  - peripheral chips in PCs, toys, handheld devices, etc.
- Target: small to medium markets, tight design schedules
  - e.g. consumer electronics
- Complexity of design: standard design style, quite predictable
  - standard flows, standard off-the-shelf tools
- Role of logic synthesis:
  - used on large fraction of design except for special blocks such as RAM's, ROM's, analog components

ASIC Design Flow

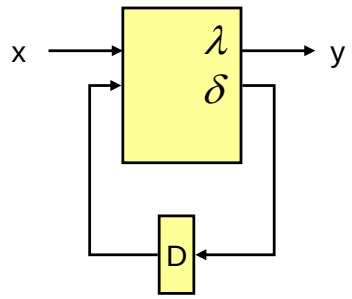
(incomplete picture)



What Is Logic Synthesis About?

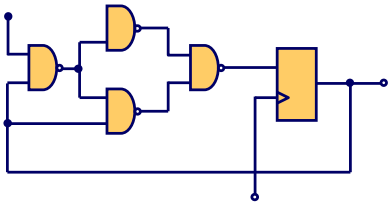


What Is Logic Synthesis About?



Given: Finite-State Machine F(Q,X,Y,δ,λ) where:

Q: Set of states
X: Input alphabet
Y: Output alphabet
δ: $X \times Q \rightarrow Q$ (next-state *function*)
λ: $X \times Q \rightarrow Y$ (output *function*)



Target: Circuit C(G, W) where:

G: set of circuit components $g \in$ {*Boolean* gates, flip-flops, etc.}

W: set of wires connecting G

by courtesy of A. Kuehlmann

Why Is Logic Synthesis Useful?

- Core logic optimization technique in today's EDA flows for IC and system design
- Broad applications in hardware model checking, software verification, program synthesis, and other areas besides circuit optimization
  - Synthesis and verification are two sides of the same coin
- Good subject to get acquainted to Boolean reasoning

Brief History

- 1847: Boole's "algebra of logic"
- 1937: Shannon's M.S. thesis, *A Symbolic Analysis of Relay and Switching Circuits*
- 1950s: Quine's minimization theory of Boolean formulas
- 1958: Kilby's invention of IC
- 1960s: ATPG D-Algorithm for Boolean reasoning
- 1970s: two-level logic minimization for PLA,
  - IBM introduced formal equivalence checking in computer design in 1978 and logic synthesis for gate array based design in 1979
- 1980s: multi-level logic minimization, FSM optimization, technology mapping, BDD, symbolic equivalence checking
  - Synopsys founded in 1986
    - first product "remapper" between standard cell libraries
- 1990s: sequential circuit optimization, don't care computation, FPGA synthesis, SAT, low-power synthesis, physical-aware logic synthesis, hardware property checking
  - More companies founded including Ambit, Compass, Synplicity. Magma, Monterey, ...
- 2000s: large-scale logic synthesis, synthesis for reliability, synthesis for emerging technologies, statistical analysis and optimization

Course Outline

- Representation of Boolean functions and basic algorithms
  - Boolean functions, formulas, circuits, SOP and POS representations, BDDs
  - Efficient data structures and algorithms for Boolean reasoning
- Combinational circuit optimization
  - Technology-independent two-level/multi-level logic optimization
  - Technology mapping
- Timing analysis and optimization
- Sequential circuit optimization
  - Clock skewing, retiming and resynthesis
- Formal verification
  - Reachability analysis
  - Formal equivalence checking
  - Safety property checking
- Logic synthesis and verification tool
  - ABC

(Detailed schedule is on the course webpage)