ECE 695R: System-on-Chip Design

Module 1: Introduction and Background
Lecture 1.6: SoC Design - Behavioral Synthesis

Anand Raghunathan
raghunathan@purdue.edu

Fall 2014, ME 1052, T Th 12:00PM-1:15PM
Outline

• Background
  – Taxonomy of ICs
  – Levels of design abstraction

• Key topics in SoC design
  – HW/SW Partitioning
  – Behavioral synthesis
  – On-chip communication architecture
  – SoC modeling and co-simulation
Abstraction levels for specifying hardware and software

- Designers are less likely to explore HW/SW partitioning if significant manual effort is involved.
- Key issue: Traditionally, level of abstraction in HW design has lagged SW.
- Recent developments allow hardware to be specified using software-like descriptions.
  - Need to automatically translate to lower levels of abstraction.

The HW/SW co-design ladder:

- High-level programming language (e.g., C, C++, SystemC)
  - Compilers (1960's, 1970's)
  - Assembly
    - Assemblers, linkers (1950's, 1960's)
  - Machine instructions
  - Microprocessor plus program: “software”

- Implementation
  - Transistors
  - Logic equations / FSM's
    - Logic synthesis (1970's, 1980's)
  - RT synthesis (1980's, 1990's)
  - Register transfers
  - Behavioral synthesis (1990's, 2000's)
  - ASIC, or FPGA: “hardware”
Behavioral Synthesis

• Input is a high level, algorithmic description
  – Concurrent and sequential program regions
  – Control structures (if/then, loop, subroutines)
  – Abstract data types
  – Logical and arithmetic operators

• A set of constraints / synthesis directives
  – Speed, power, area, interconnect style
  – A library of pre-specified components

• Output is an RTL description for further synthesis
Behavioral Synthesis: Input

• Behavioral description
  – No notion of time (or loosely timed)
  – No structure

ARCHITECTURE algorithm OF barcode IS
BEGIN barcode: PROCESS
  BEGIN
    eoc <= false; memw <= false;
data <= 0; addr <= 0;

  RESET_LOOP: LOOP
  LOOP EXIT WHEN start; END LOOP;
  LOOP
    LOOP EXIT WHEN scan; END LOOP;
    flag := wh; actnum := 0; white := 0; black := 0;
eoc <= false;
  LOOP
    IF video = wh THEN
      white := white + 1;
    IF flag = bl THEN
      actnum := actnum + 1; memw <= false;
    ELSE
      memw <= true;
    END IF;
  END LOOP
  END LOOP;
  black := 0; flag := wh; data <= white;
  ELSE
    black := black + 1;
    IF flag = wh THEN
      actnum := actnum + 1; memw <= false;
    ELSE
      memw <= true;
    END IF;
  END IF;
  addr <= actnum;
  EXIT WHEN (white = 255) OR (black = 255); END LOOP;
  EXIT WHEN (actnum = num) AND (white = 255); END LOOP;
  memw <= false; eoc <= true;
  LOOP EXIT WHEN start = false; END LOOP;
END LOOP RESET_LOOP;
END PROCESS;
END algorithm;

Behavioral description of Barcode Reader in VHDL
Behavioral Synthesis: Output

- Structural RTL that can be further synthesized using logic synthesis tools
Why Behavioral Synthesis?

• Reduced design effort and cost
• Better architectural exploration
• Bridging abstraction gap between HW and SW enables tradeoffs and co-design
Behavioral Synthesis vs. RTL Coding

- Automatically select “architecture” and bypass RTL coding / verification /debugging loop
Behavioral Synthesis: Automatic Design
Space Exploration

C.D. Thompson & H.T. Kung: Any realization of a specific algorithm in hardware follows a fundamental area-speed tradeoff curve (e.g., $AT^2 = \text{constant}$)

Similar tradeoff exists for power vs. performance
Behavioral Synthesis: Automatic Design Space Exploration

Designs with a fixed RTL (no. of FUs, registers, buses, etc.)

Beh. Synth.: Different architectures
Behavioral Synthesis Process

- **Scheduling**
  - Determining the cycle-by-cycle timing behavior

- **Resource Allocation**
  - Allocating resources (hardware components) to realize computation and storage

- **Binding / Assignment**
  - Mapping each operation (variable) in the behavior to a specific component instance, each variable to a specific register, and instantiating buses / muxes for data transfer
Behavioral Synthesis Steps

1. Behavioral Description
2. CDFG Generation
3. Resource Allocation
4. Scheduling
5. Simulation model (cycle-accurate)
6. Binding
7. Data Path and State Machine Extraction
8. RTL generation
Behavioral Synthesis: Historical Perspective

• In the 1980’s Silicon compilation was the “holy grail” of EDA researchers – and industry
  – Idea: Go from algorithm to Silicon
• The idea was oversold and/or ahead of its time
  – The technology and compute platforms available at the time could not (in general) do the job
  – No burning need
• The emergence of SoCs has created a new need
• New languages, new algorithms, faster computers and lower expectations have led to a revival of this area as “behavioral synthesis”