A Gentle Introduction to High Level Synthesis

Jianwen Zhu

Electrical and Computer Engineering
University of Toronto

jzhu@eecg.toronto.edu
http://www.eecg.toronto.edu/~jzhu

Outline

- Overview
- Scheduling
- Resource Sharing
- Summary
**Y-chart**

**Behavioral Domain**
- System level
- Architecture level
- Logic level
- Geometric level

**Structural Domain**
- System level
- Architecture level
- Logic level
- Geometric level

**Geometric Domain**
- Logic level
- Architecture level
- System level

---

**High-level Synthesis**

- **What is Synthesis**
  - Given a functional model of a design
  - Find a structural model of a design
  - Such that some figure of merit is optimized
    - Speed
    - Area
    - Power
    - Noise
  - Subject to some constraints

- **What is High-level Synthesis**
  - Given an algorithm model of a design
  - Find a micro-architecture
    - Controller: sequential random logic, ROM
    - Datapath: adder, ALU, mux, register, register file
  - Such that speed/area/power is optimized
  - Subject to some constraints
High-level Function Model

- Can be captured by an imperative program
  - C/C++, Java, ...
  - Behavioral VHDL/Verilog
- Untimed state machine

- Can be transformed into control-dataflow graph (CDFG) by synthesis front-end
  - Ease for machine manipulation
  - Control flow: statement
  - Data flow: expression

Example of Dataflow Graph

```
... while( x > a ) {
  x1 = x + dx;
  u1 = u - (3 * x + u * dx);
  y1 = y + u * dx;
  x = x1;
  u = u1;
  y = y1;
}
...`
Micro-architecture

- Controller is responsible for *when* and *what* register transfer operations (assignments) to perform
- Datapath is responsible for *how* to perform the register transfer operations

![Diagram of Micro-architecture]

Micro-architecture: Controller

- Controller is a sequential network
- Can be implemented using logic synthesis and layout tool

![Diagram of Micro-architecture: Controller]
Micro-architecture: Datapath

- Datapath is a network of sequential and combinational components:
  - Registers, register files
  - Adders, Subtracters, ...
  - Steering components: buses, selectors, ...

Quality Metrics/Constraints

- Latency: the time it takes to process the data
- Throughput: the rate to process the data
- Cycle time
- Area
- Power
Outline

- Overview
- Scheduling
- Resource Sharing
- Summary

Scheduling

- Have to decide *when* each operation is performed
  - untimed state machine $\rightarrow$ timed state machine
  - flow graph $\rightarrow$ ASM chart
- Pretty much like the determine the class schedule
  - Dependency constraint: ECE241 is a prerequisite of ECE451
  - Resource constraint: Do not have enough #rooms to hold all classes simultaneously
- Here
  - Dependency constraint: A depends on the result of B
  - Resource constraint: there are at most 3 adders
Unconstrained Scheduling: List Scheduling

- As Soon As Possible (ASAP) schedule
- As Late As Possible (ALAP) schedule

Resource-constrained List Scheduling

- Keep a list of ready operations whose predecessors are all scheduled
- Schedule an operation from the ready list
  - Has no resource conflict
  - Has higher priority
- Heuristics to determine the priority
  - Mobility
  - Out degree
  - Distance to the sink
List Scheduling Example

ASAP Schedule

Operator Mobility

ALAP Schedule

Mobility(op) = ASAP − ALAP

Outline

- Overview
- Scheduling
- Resource Sharing
- Summary
Square Root Approximation (SRA) Example

- Computes $O = \sqrt{a^2 + b^2}$
- Approximation:
  
  $O = \max((0.875x + 0.5y), x)$ where
  
  $x = \max(|a|, |b|), y = \min(|a|, |b|)$

- A scheduled design
- ASM chart

A Straight-forward Approach

- Map each variable to a distinct register
- Map each operation to a distinct combinational component
**Solving Resource Sharing Problem**

- Resources (registers, functional units) can be shared
- Cast resource sharing problem into a graph problem
  - Graph nodes: subject objects to be shared
  - Graph edges: sharing relation
  - Dual problem:
    - Coloring of conflict graph
    - Clique-partitioning of compatibility graph

**Resource Sharing Algorithm**

- Graph coloring
  - Color one node at a time
  - Select the color different from its neighbors
- Graph partitioning
  - Select two compatible nodes at a time
  - Merge them into a supernode
  - Update edges accordingly
Register Sharing

- Two variables can share the same register if and only if they have non-overlapping lifetime.
- Lifetime = [first time write, last time read]
- Casting register sharing into graph partitioning problem
  - Each vertex represents a variable.
  - There is a dashed edge between two vertices if the corresponding vertices have overlapping lifetime.
  - There is a solid edge between two vertices if the corresponding vertices have non-overlapping lifetime.
  - The solid edge is annotated with #common src/#common dest.

Variable Compatibility Graph

(a) Initial compatibility graph
Register Sharing by Graph Partitioning

(a) Initial compatibility graph

(b) Compatibility graph after merging $t_1$, $t_2$, and $t_3$

(c) Compatibility graph after merging $t_1$ and $y$

(d) Compatibility graph after merging $t_5$ and $y$

(e) Final compatibility graph

SRA Implementation after Register Sharing

(a) Register assignments

$$R_1 = \{a, t_1, x, t_2\}$$
$$R_2 = \{b, t_1, y, t_3, t_5, t_6\}$$
$$R_7 = \{t_5\}$$

(b) Datapath
Functional Unit Sharing

- Two Operations can share the same functional unit if they are non-concurrent and has “similar” functionality
- Sharing priority: #common source, #common destinations

![Partial ASM Chart](image1)

![Non-shared design](image2)

![Shared design](image3)

More Components in the Library

(a) Unit for computing minimum, maximum and absolute value

(b) Unit for computing addition, subtraction, minimum and maximum

(c) Unit for computing addition, subtraction, and absolute value

(d) Unit for computing addition, subtraction, minimum, maximum and absolute value
Functional Unit Sharing by Graph Partitioning

(a) Compatibility graph
(b) Cost table
(c) Merging alternative
(d) Cost table

SRA Implementation after Functional Unit Sharing

(a) Datapath schematic for unit allocation from Figure 8.22(c)
(b) Datapath schematic for unit allocation from Figure 8.22(e)
Bus Sharing

- Wires can be expensive
- Different interconnections can be shared if they are not used at the same cycle
- Pitfall: may end up longer wires

Bus Sharing by Graph Partitioning

(a) Datapath for SRA
(b) Connectivity usage table
(c) Compatibility graph for input buses
(d) Compatibility graph for output buses
(e) Bus assignment

Bus 1 = \{A, C, D, E, H\}
Bus 2 = \{B, F, G\}
Bus 3 = \{I, K, M\}
Bus 4 = \{J, L, N\}
SRA Implementation after Bus Sharing

Bus 1 = [A, C, D, E, H]
Bus 2 = [B, F, G]
Bus 3 = [I, K, M]
Bus 4 = [J, L, N]

Datapath for SRA

Summary

- High-level synthesis maps an imperative program into a micro-architecture, which can be further synthesized by lower-level tools.
- Scheduling determines the control step at which each operation is performed.
- Binding determines how variables, operations, data transferred are mapped into shared registers, functional units and buses.