Unit 2: High-Level Synthesis

- Course contents
 - Hardware modeling
 - Data flow
 - Scheduling/allocation/assignment
- Reading
 - Chapter 11



1

High-Level Synthesis (HLS)

- Hardware-description language (HDL) synthesis
 - Starts from a register-transfer level (RTL) description; circuit behavior in each clock cycle is fixed.
 - Uses logic synthesis techniques to optimize the design.
 - Generates a netlist.
- High-level synthesis (also called architectural or behavioral synthesis)
 - Starts from an abstract behavioral description.
 - Generates an RTL description.
 - It normally has to perform the trade-off between the number of cycles and the hardware resources to fulfill a task.

HLS Versus RTL Synthesis

- RTL synthesis implements all functionality within a single clock cycle.
- Behavioral synthesis automatically allocates the functionality across multiple clock cycles.



What Is Generated by HLS

 Behavioral Compiler creates a design that consists of a datapath, memory I/O and a control FSM



Benefits of HLS (1)

- Quick specification and verification.
 - Specify behavioral HDL easily, since it's intuitive and natural to write.
 - Save time -- behavioral HDL code is up to 10 times shorter than equivalent RTL.
 - Simulate orders of magnitude faster because of the higher level of abstraction.
 - Reuse designs more readily by starting with a more abstract description.
- Automatically infer memory and generate finite state machine (FSM).
 - Specify memory reads and writes.
 - Schedule memory I/O, resolve contention and build control FSM.
 - Trade-off single vs. multiported memories.
 - Generate a new FSM.

Benefits of HLS (2)

- Explore architectural trade-offs.
 - Create multiple architectures from a single specification.
 - Trade-off throughput and latency using high-level constraints.
 - Analyze various combinations of technology-specific datapath and memory resources.
 - Evaluate cost/performance of various implementations rapidly.
- Reduce design time
 - Model hardware and software components of system concurrently.
 - Easily implement algorithms in behavioral HDL and generate RTL code with a behavioral compiler.
 - Verify hardware in system context at various levels of abstraction.

Hardware Models for High-level Synthesis

- All HLS systems need to restrict the target hardware.
 The search space is too large, otherwise.
- All synthesis systems have their own peculiarities, but most systems generate synchronous hardware and build it with functional units:
 - A functional unit can perform one or more computations, e.g. addition, multiplication, comparison, ALU.



Hardware Models

- Registers: they store inputs, intermediate results and outputs; sometimes several registers are taken together to form a register file.
- Multiplexers: from several inputs, one is passed to the output.





Hardware Models (cont'd)

- Buses: a connection shared between several hardware elements, such that only one element can write data at a specific time.
- Three-state (tri-state) drivers control the exclusive writing on the bus.





Hardware Models (cont'd)

- Parameters defining the hardware model for the synthesis problem:
 - Clocking strategy: e.g. single or multiple phase clocks.
 - Interconnect: e.g. allowing or disallowing buses.
 - Clocking of functional units: allowing or disallowing of
 - multicycle operations
 - chaining
 - pipelined units.

Chaining, Multicycle Operation, Pipelining

multicycle twooperation stage 🖉 pipeline \mathcal{R} < chaining Cycle boundary

Example of a HLS Hardware Model



Hardware Concepts: Data Path + Control

- Hardware is normally partitioned into two parts:
 - Data path: a network of functional units, registers, multiplexers and buses.
 - The actual "computation" takes place in the data path.
 - Control: the part of the hardware that takes care of having the data present at the right place at a specific time, of presenting the right instructions to a programmable unit, etc.
- High-level synthesis often concentrates on data-path synthesis.
 - The control part is then realized as a finite state machine or in microcode.

Steps of High Level Synthesis

- Preprocess the design with high-level optimization
 - Code motion
 - Common subexpression elimination
 - Loop unrolling
 - Constant Propagation
 - Modifications taking advantage of associativity and distributivity, etc.
- **Transform** the optimized design into intermediate format (internal representation) which reveals more structural characteristics of the design.
- **Optimize** the intermediate format
 - Tree height reduction
 - Behavior retiming
- Allocate the required resources to implement the design
 - Also called module selection
- Schedule each operation to be performed at certain time such that no precedence constraint is violated.
- Assign (Bind) each operation to a specific functional unit and each variable to a register.

HLS Optimization Criteria

- Typically, speed, area, and power consumption.
- Optimization is often constrained
 - Optimize area when the minimum speed is given \Rightarrow timeconstrained synthesis.
 - Optimize speed when a maximum for each resource type is given \Rightarrow resource-constrained synthesis.
 - Minimize power dissipation for a given speed and area requirement => Power-constrained synthesis

Input Format

- The algorithm, that is the input to a high-level synthesis system, is often provided in textual form either
 - in a conventional programming language, such as C, C++, SystemC, or
 - in a hardware description language (HDL), which is more suitable to express the parallelism present in hardware.
- The description has to be parsed and transformed into an internal representation and thus conventional compiler techniques can be used.

Example of High-Level Optimization

• Applying the distributivity law to reduce resource requirement.



Internal Representation

- Most systems use some form of a data-flow graph (DFG).
 - A DFG may or may not contain information on control flow.
- A data-flow graph is built from
 - Vertices (nodes): representing computation, and
 - Edges: representing precedence relations.

x := a * b; y := c + d; z := x + y;



Token Flow in a DFG

- A node in a DFG fires when all tokens are present at its inputs.
- The input tokens are consumed and an output token is produced.



• By means of two special nodes:



Explicit Iterative Data Flow

 Selector and distributor nodes can be used to describe iteration.

Example: while (a > b) $a \leftarrow a - b;$

 Loops require careful placement of initial tokens on edges



Implicit Iterative Data Flow

- Iteration implied by regular input stream of tokens.
- Initial tokens act as buffers.
- Delay elements instead of initial tokens.



Iterative DFG Example



A second-order filter section.

Optimization of Internal Representation

- Restructuring data and control flow graphs prior to the actual mapping onto hardware. Examples:
 - Replacing chain of adders by a tree.
 - Behavior retiming



Tree height reduction

Behavior Retiming (BRT)

 By moving registers through logic and hierarchical boundaries, BRT reduces the clock period with minimum area impact.





Effectiveness of BRT

psys exp:	RTLD	ESIGN-	BEHAVIORAL RETIMING		SUMMARY
DESIGN TYPE	SPEED (CLOCE/PEROD)	Gates	SPEED (CLOCE PERIOD) GATES		
Control	44 ns	10,913 gates	30.6 ns	11,313 gates	30% faster, 4% more area
Control	23.1 ns	3,598-gates	19.6-ns	4,675 gates	15% faster, 27% more area
Control	28.6 ns	3,585-gates	28.6 ns	3,369 gates	same-speed, 6%-less-area
Dataflow& Control	17 ns	28,900 gates	12.5.ns	30,100 gates	26%-faster, 4%-more-area
Dataflow& Control	16 ns	7,620-gates	13.ns	8,019 gates	20% faster, 5% more area
Dataflow	22 ns	4,990 gates	18.5 ns	5,109 gates	16% faster, 2% more area
Dataflow	28 ns	31,226-gates	26.ns	32,032-gates	8% faster, 2% more area
Dataflow	26.2-ns	14,351 gates	23.6-ns	13,847 gates	10% faster, 4% less area
Dataflow	25.9.ns	16,798-gates	20.8-ns	15,550 gates	20% faster, 7% less area
Dataflow	45 ns	28,705-gates	26·ns	30,987 gates	42% faster, 8% more area

- RTL designs have a single clock net and were synthesized into gates using Synopsys Design Compiler.
- Design type: dataflow implies significant number of operators; control implies state machine dominated.

HLS Subtasks: Allocation, Scheduling, Assignment

- Subtasks in high-level synthesis
 - Allocation (Module selection): specify the hardware resources that will be necessary.
 - Scheduling: determine for each operation the time at which it should be performed such that no precedence constraint is violated.
 - Assignment (Binding): map each operation to a specific functional unit and each variable to a register.
- Remarks:
 - Though the subproblems are strongly interrelated, they are often solved separately. However, to attain a better solution, an iterative process executing these three subtasks must be performed.
 - Most scheduling problems are NP-complete \Rightarrow heuristics are used.

Example of High Level Synthesis

- The following couple of slides shows an example of scheduling and binding of a design with given resource allocation.
- Given the second-order filter which is first made acyclic:



Example of Scheduling

• The schedule and operation assignment with an allocation of one adder and one multiplier:



Perform Binding to Generate Data Path

- The resulting data path after register assignment.
 - The specification of a controller would complete the design.



Resource Allocation Problem

- This problem is relatively simple. It simply decides the kinds of hardware resources (hardware implementation for certain functional units such as adder, multiplier, etc.) and the quantity of these resources.
 - For example two adders, one multiplier, 4 32-bit registers, etc. for a certain application.
- The decision made in this step has a profound influence on the scheduling which under the given resource constraints decides the time when an operation should be executed by a functional unit.
- This step set an upper bound on the attainable performance.

Problem Formulation of Scheduling

- Input consists of a DFG G(V, E) and a library \Re of resource types.
- There is a fixed mapping from each v ∈ V to some r ∈ ℜ;
 the execution delay δ(v) for each operation is therefore known.
- The problem is time-constrained; the available execution times are in the set $\mathcal{T} = \{0, 1, \dots, T_0 1\}$.
- A schedule maps each operation to its starting time; for each edge (v_i, v_j) ∈ E, a schedule should respect: σ(v_j) ≥ σ(v_i) + δ(v_i).
- Given the resource type cost $\omega(r)$ and the requirement function $N_r(\sigma)$, the cost of a schedule σ is given by:

$$\sum_{r\in\mathfrak{R}}\omega(r)N_r(\sigma).$$

- As soon as possible (ASAP) scheduling maps an operation to the earliest possible starting time not violating the precedence constraints.
- Properties:
 - It is easy to compute by finding the longest paths in a directed acyclic graph.
 - It does not make any attempt to optimize the resource cost.

Graph for ASAP Scheduling



Mobility-Based Scheduling

- Compute both the ASAP and ALAP (as late as possible) schedules σ_S and σ_L .
- For each $v \in V$, determine the scheduling range $[\sigma_{S}(v), \sigma_{L}(v)]$.
- $\sigma_L(v)$ $\sigma_S(v)$ is called the mobility of *v*.
- Mobility-based scheduling tries to find the best position within its scheduling range for each operation.

- A partial schedule $\tilde{\sigma}: V \rightarrow [\mathcal{T}, \mathcal{T}]$ assigns a scheduling range to each $v \in V$, $\tilde{\sigma}(v) = [\tilde{\sigma}_{\min}(v), \tilde{\sigma}_{\max}(v)].$
- Finding a schedule can be seen as the generation of a sequence of partial schedules: $\tilde{\sigma}^{(0)}, ..., \tilde{\sigma}^{(n)}$.

Simple Mobility-Based Scheduling

- A partial schedule $\tilde{\sigma}: V \rightarrow [\mathcal{T}, \mathcal{T}]$ assigns a scheduling range to each
- Finding a schedule can be seen as the generation of a sequence of partial schedules

```
"determine σ̃<sup>(0)</sup> by computing σ<sub>S</sub> and σ<sub>L</sub>";
k ← 0;
while ("there are unscheduled operations") {
v ← "one of the nodes with lowest mobility";
"schedule v at some time that optimizes the current resource utilization";
"determine σ̃<sup>(k+1)</sup> by updating the scheduling ranges
of the unscheduled nodes";
k ← k + 1
```

- A resource-constrained scheduling method.
- Start at time zero and increase time until all operations have been scheduled.

– Consider the precedence constraint.

- The ready list L_t contains all operations that can start their execution at time *t* or later.
- If more operations are ready than there are resources available, use some priority function to choose, e.g. the longest-path to the output node ⇒ critical-path list scheduling.

List Scheduling Example



The Assignment Problem

- Subtasks in assignment:
 - operation-to-FU assignment
 - value grouping
 - value-to-register assignment
 - transfer-to-wire assignment
 - wire to FU-port assignment
- In general: task-to-agent assignment

Compatibility and Conflict Graphs

- Clique partitioning gives an assignment in a compatibility graph.
- Graph coloring gives an assignment in the complementary conflict graph.



The Assignment Problem

- Assumption: assignment follows scheduling.
- The claim of a task on an agent is an interval ⇒ minimum resource utilization can be found by left-edge algorithm.
- In case of iterative algorithm, interval graph becomes circular-arc graph ⇒ optimization is NP-complete.



Tseng and Sieworek's Algorithm

```
k \leftarrow 0:
G_c^k(V_c^k, E_c^k) \leftarrow G_c(V_c, E_c);
while (E_c^k \neq \emptyset) {
    "find (v_i, v_j) \in E_c^k with largest set of common neighbors";
     N \leftarrow "set of common neighbors of v_i and v_j";
    s \leftarrow i \cup j:
    V_c^{k+1} \leftarrow V_c^k \cup \{v_s\} \setminus \{v_i, v_j\};
    E_{c}^{k+1} \leftarrow \emptyset:
    for each (v_m, v_n) \in E_c^k
        if (v_m \neq v_i \land v_m \neq v_j \land v_n \neq v_i \land v_n \neq v_j)
           E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_m, v_n)\};
    for each v_n \in N
       E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_n, v_s)\};
    k \leftarrow k + 1:
}
```

Clique-Partitioning Example



Example of Behavior Optimization



Effectiveness of BOA

Supopeus	DESIGN TYPE	RTL DESIGN	воа	SUMMARY
example	Motion estimation	23.6 пs 12,793 gates	20.2 пs 12,215 gates	14% faster, 5% less area
	Graphics interpolation	19.2 ns 3,507 gates	17.5 ns 2,952 gates	9% faster, 16% less area
	Color space conversion and scaling	16 ns 35,866 gates (manual CSA implementation)	14.9 пs 34,397 gates	7% faster, 4% less area
	Sum of 9 operands	7.7 ns 1,418 gates	5.3 ns 1,307 gates	31 % faster, 8 % less area
	a * b + 1	11.6 ns 2,577 gates	9.3 ns 2,524 gates	20 % faster, 2 % less area
	a * 4104 (0100000 100000100)	4.4 ns 759 gates	3.1 ns 449 gates	30 % faster, 40 % less area
	a * 3 E3 E (0011111000 111110)	5.7 ns 927 gates	4.6 ns 709 gates	19% faster, 23% less area
	a * b + c	11.0 ns 2,707 gates	10.0 ns 2,689 gates	9% faster, same area
	a*b+c*d+e*f	14.2 ns 7,435 gates	12.8 пs 7,110 gates	10% faster, 4% less area
	Sum of 16 operands	8.1 ns 2,836 gates	6.7 пs 2,123 gates	17% faster, 25% less area

Summary

- Concepts of High Level Synthesis (HLS)
- Benefits of HLS
- Steps of High Level Synthesis
- Internal Representation and Optimization
- Resource Allocation
- Algorithms for Scheduling
- Algorithms for Binding (Assignment)