More Scheduling
Resource Sharing
Announcements

▪ Lab 3 is released (due Monday 10/3)
  – Lower penalty on late submission
  – Go through the CORDIC tutorial first

▪ Second reading assignment
  – Complete reading the first 3 sections before Tuesday 10/4
Agenda

- More SDC scheduling
  - An exact formulation combining SDC and SAT

- Resource sharing overview
  - Sub-problems: functional unit, register, and connectivity binding problems
  - Key concepts: compatibility and conflict graphs
Review: SDC-Based Scheduling

- A linear programming formulation based on system of integer difference constraints (SDC)

- Target cycle time: 5ns
- Delay estimates
  - Mul (x): 3ns
  - Add (+): 1ns
  - Load/Store (ld/st): 1ns

\( s_i \): schedule variable for operation \( i \)

- Dependence constraints
  \[
  \langle v_0, v_4 \rangle : s_0 - s_4 \leq 0 \\
  \langle v_1, v_3 \rangle : s_1 - s_3 \leq 0 \\
  \langle v_2, v_3 \rangle : s_2 - s_3 \leq 0 \\
  \langle v_3, v_4 \rangle : s_3 - s_4 \leq 0 \\
  \langle v_4, v_5 \rangle : s_4 - s_5 \leq 0
  \]

- Cycle time constraints
  \[
  v_2 \rightarrow v_5 : s_2 - s_5 \leq -1 \\
  v_1 \rightarrow v_5 : s_1 - s_5 \leq -1
  \]

Operation chaining is naturally supported

[J. Cong & Z. Zhang, DAC, 2006] [Z. Zhang & B. Liu, ICCAD, 2013]
Recap: SDC Constraint Graph

- Difference constraints can be conveniently represented using constraint graph
  - Each vertex represents a variable, and each weighted edge corresponds to a different constraint
  - Detect infeasibility by the presence of negative cycle (by solving single-source shortest path)
Encoding Resource Constraints

Two read ports only!

Load operations must be serialized
(NP-Hard in general)

$s_0 - s_1 \neq 0$

Difficult to exactly encode resource constraints in the strict SDC form

Using Boolean formulas instead

Resource sharing variable
$v_0$ and $v_1$ share the same port?

Ordering variable
$v_0$ scheduled before $v_1$?

$v_1$ scheduled before $v_0$?

Note: $R_{0,1} \rightarrow (O_{0\rightarrow 1} \lor O_{1\rightarrow 0})$ reads
“$R_{0,1}$ implies $O_{0\rightarrow 1}$ or $O_{1\rightarrow 0}$”
SDS: Exact and Practically Scalable Scheduling with SDC and SAT

Partial orderings

Difference constraints

Conflict clauses

SDC
Timing Constraints

\[ R_{01} \rightarrow (O_{0 \rightarrow 1} \lor O_{1 \rightarrow 0}) \]
\[ \neg(O_{0 \rightarrow 1} \land O_{1 \rightarrow 0}) \]
\[ R_{02} \rightarrow (O_{0 \rightarrow 2} \lor O_{2 \rightarrow 0}) \]
\[ \neg(O_{0 \rightarrow 2} \land O_{2 \rightarrow 0}) \]
\[ R_{12} \rightarrow (O_{1 \rightarrow 2} \lor O_{2 \rightarrow 1}) \]
\[ \neg(O_{1 \rightarrow 2} \land O_{2 \rightarrow 1}) \]

\[ s_0 - s_4 \leq 0 \]
\[ s_1 - s_3 \leq 0 \]
\[ s_2 - s_3 \leq 0 \]
\[ s_3 - s_4 \leq 0 \]
\[ s_4 - s_5 \leq 0 \]
\[ s_2 - s_5 \leq -1 \]
\[ s_1 - s_5 \leq -1 \]

Conflict based search
~1M variables
>1M clauses

[ S. Dai, G. Liu, and Z. Zhang, FPGA 2018 ]
Given a Boolean function $F(x_1, x_2, \ldots x_n)$, find an assignment to $x_i$’s to make $F$ evaluate to 1
- If such assignment exists, $F$ is satisfiable
- Otherwise, $F$ is unsatisfiable

Example: $(x + y + z) (x' + y' + z) (x' + y' + z')$
- A satisfying assignment: $x=1$, $y=0$, $z=1$

First NP-complete problem (Cook-Levin theorem)

Numerous practical applications
- Hardware/software verification (e.g., equivalence checking, model checking)
- Artificial intelligence (e.g., planning, automated reasoning)
- Automated theorem proving
- Combinatorial design
...
Scalability of SAT Solvers

- SAT solvers have made significant progress in scalability
  - From toy problems with 100-200 variables (early 90s)
  - To industrial applications with 1M+ variables, 5M+ constraints (2010s)

- Modern SAT solvers typically employ a backtracking-based search algorithm where conflict-driven clause learning is a key to efficiency

[source: A. Sabharwal, Modern SAT Solvers: Key Advances and Applications, 2011]
Encoding Resource Constraints in SAT

\( R_{u,v} \) denotes whether \( u \) shares the same resource with \( v \)

\( O_{u \rightarrow v} \) denotes whether \( u \) is scheduled earlier than \( v \)

**Ordering constraints:** Operations sharing the same resources must be scheduled apart

\[ R_{0,1} \rightarrow ( O_{0 \rightarrow 1} \lor O_{1 \rightarrow 0} ) \]
\[ R_{0,2} \rightarrow ( O_{0 \rightarrow 2} \lor O_{2 \rightarrow 0} ) \]
\[ R_{1,2} \rightarrow ( O_{1 \rightarrow 2} \lor O_{2 \rightarrow 1} ) \]
\[ \neg ( O_{0 \rightarrow 1} \land O_{1 \rightarrow 0} ) \]
\[ \neg ( O_{0 \rightarrow 2} \land O_{2 \rightarrow 0} ) \]
\[ \neg ( O_{1 \rightarrow 2} \land O_{2 \rightarrow 1} ) \]

**Note 1:** \( R_{0,1} \rightarrow ( O_{0 \rightarrow 1} \lor O_{1 \rightarrow 0} ) \) reads “\( R_{0,1} \) implies \( O_{0 \rightarrow 1} \) or \( O_{1 \rightarrow 0} \)”

**Note 2:** \( \neg ( O_{0 \rightarrow 1} \land O_{1 \rightarrow 0} ) \) reads “\( O_{0 \rightarrow 1} \) and \( O_{1 \rightarrow 0} \) cannot be both true”
Conflict-Driven Learning

Resource constraints encoded in SAT

Partial orderings

Difference constraints

Timming constraints encoded in SDC graph

Conflict clauses

Infeasibility

What SAT learns from SDC:
Any ordering involving operation 0 before 2 should no longer be attempted

\[ R_{01} \rightarrow (O_{0 \rightarrow 1} \lor O_{1 \rightarrow 0}) \]
\[ \neg (O_{0 \rightarrow 1} \land O_{1 \rightarrow 0}) \]
\[ R_{02} \rightarrow (O_{0 \rightarrow 2} \lor O_{2 \rightarrow 0}) \]
\[ \neg (O_{0 \rightarrow 2} \land O_{2 \rightarrow 0}) \]
\[ R_{12} \rightarrow (O_{1 \rightarrow 2} \lor O_{2 \rightarrow 1}) \]
\[ \neg (O_{1 \rightarrow 2} \land O_{2 \rightarrow 1}) \]

Note: \( O_{0 \rightarrow 2} = \text{True} \)
\( s_0 - s_2 \leq -1 \)
Conflict-Driven Learning

Resource constraints encoded in SAT

\[
R_{01} \rightarrow (O_{0 \rightarrow 1} \lor O_{1 \rightarrow 0}) \\
\neg(O_{0 \rightarrow 1} \land O_{1 \rightarrow 0})
\]

\[
R_{02} \rightarrow (O_{0 \rightarrow 2} \lor O_{2 \rightarrow 0}) \\
\neg(O_{0 \rightarrow 2} \land O_{2 \rightarrow 0})
\]

\[
R_{12} \rightarrow (O_{1 \rightarrow 2} \lor O_{2 \rightarrow 1}) \\
\neg(O_{1 \rightarrow 2} \land O_{2 \rightarrow 1}) \\
\neg O_{0 \rightarrow 2}
\]

Timing constraints encoded in SDC graph

Negative cycle (sum of edge weights = -2)

Propose:
- \(O_{0 \rightarrow 1} = \text{True}\)
- \(O_{2 \rightarrow 0} = \text{True}\)
- \(O_{1 \rightarrow 2} = \text{True}\)

Conflict:
- \(\neg(O_{0 \rightarrow 1} \land O_{1 \rightarrow 2})\)
Conflict-Driven Learning

Resource constraints encoded in SAT

\[ R_{01} \rightarrow (O_{0\rightarrow 1} \lor O_{1\rightarrow 0}) \]
\[ \neg (O_{0\rightarrow 1} \land O_{1\rightarrow 0}) \]
\[ R_{02} \rightarrow (O_{0\rightarrow 2} \lor O_{2\rightarrow 0}) \]
\[ \neg (O_{0\rightarrow 2} \land O_{2\rightarrow 0}) \]
\[ R_{12} \rightarrow (O_{1\rightarrow 2} \lor O_{2\rightarrow 1}) \]
\[ \neg (O_{1\rightarrow 2} \land O_{2\rightarrow 1}) \]
\[ \neg O_{0\rightarrow 2} \]
\[ \neg (O_{0\rightarrow 1} \land O_{1\rightarrow 2}) \]

Feasible!
Returns schedule

Timing constraints encoded in SDC graph

No conflict

No negative cycles
Fast Conflict-Driven Learning

- Generate short conflicts
  - Shorter conflict $\Rightarrow$ more pruning $\Rightarrow$ faster convergence

  \[ \neg (O_{0\rightarrow 1} \land O_{0\rightarrow 2} \land O_{1\rightarrow 2}) \]

- Negative cycle = irreducibly inconsistent set of constraints
  - Keeps conflicts short
  - Becomes consistent if any constraint is removed from the set
Take-Away Points on SDS Scheduling

▶ Combining SDC and SAT with conflict-driven learning enables fast yet exact resource-constrained scheduling
  – Up to 1000X faster than ILP

▶ Broader applications
  – Not just limited to HLS
  – Applicable to constrained scheduling problems in other fields
Recap: A Typical HLS Flow

- High-level Programming Languages (C/C++, OpenCL, SystemC, ...)
- Parsing
- Transformations
- Allocation
- Scheduling
- Binding
- RTL generation

if (condition) {
  ...
} else {
  t_1 = a + b;
  t_2 = c * d;
  t_3 = e + f;
  t_4 = t_1 * t_2;
  z = t_4 - t_3;
}

Control data flow graph (CDFG)

Finite state machines with datapath

3 cycles
Resource Sharing and Binding

- **Resource sharing** enables reuse of hardware resources to minimize cost, in resource usage/area/power
  - Typically carried out by binding in HLS
  - Other subtasks such allocation and scheduling greatly impact the resource sharing opportunities

- **Binding** maps operations, variables, and/or data transfers to the available resources
  - After scheduling: decide resource usage and detailed architecture (**focus of this lecture**)
  - Before scheduling: affect both area and delay
  - Simultaneous scheduling and binding: better result but more expensive
Binding Sub-problems

- Functional unit (FU) binding
  - Primary objective is to minimize the number of FUs
  - Considers connection cost

- Register binding
  - Primary objective is to minimize the number of registers
  - Considers connection cost

- Connectivity binding
  - Minimize connections by exploiting the commutative property of some operations / FUs
  - NP-hard
Sharing Conditions

- Functional units (registers) are shared by operations (variables) of same type whose *lifetimes* do not overlap.

- **Lifetime**: [birth-time, death-time)
  - Operation: The whole execution time (if unpipelined)
  - Variable: From the time this variable is defined to the time it is last used
### Operation Binding

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul1</td>
<td>op1, op3</td>
</tr>
<tr>
<td>AddSub1</td>
<td>op2, op4</td>
</tr>
<tr>
<td>AddSub2</td>
<td>op5, op6</td>
</tr>
</tbody>
</table>

**Binding 1**

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul1</td>
<td>op1, op3</td>
</tr>
<tr>
<td>AddSub1</td>
<td>op2, op4, op6</td>
</tr>
<tr>
<td>AddSub2</td>
<td>op5</td>
</tr>
</tbody>
</table>

**Binding 2**
Register Binding

Lifetimes crossing clock edge
=> register(s) inferred

clock edge

a × v1
b + v2
c + v3
d ×
e +
f +
g -
Variable Lifetime Analysis

Variables v1, v2, and v3 can share the same register

Variable lifetimes [birth-time, death-time]

<table>
<thead>
<tr>
<th></th>
<th>[1, 2)</th>
<th>[2, 3)</th>
<th>[3, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compatibility and Conflict Graphs

- **Operation/variables compatibility**
  - Same type, non-overlapping lifetimes

- **Compatibility graph**
  - Vertices: operations/variables
  - Edges: compatibility relation

- **Conflict graph**: Complement of compatibility graph

A scheduled DFG (operations have the same type)

Compatibility graph

Conflict graph

Note: A compatibility/conflict graphs for variables/registers can be constructed in a similar way
Clique Cover Number and Chromatic Number

▸ Compatibility graph
  – Partition the graph into a **minimum number of cliques**
    • Clique in an undirected graph is a subset of its vertices such that
every two vertices in the subset are connected by an edge

▸ Conflict graph
  – Color the vertices by a **minimum number of colors** (chromatic number), where adjacent vertices cannot use the same color

A scheduled DFG

Clique partitioning on compatibility graph

Coloring on conflict graph
Example: Meeting Assignment Problem

<table>
<thead>
<tr>
<th>Meeting</th>
<th>Schedule (am)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9:00~11:00</td>
</tr>
<tr>
<td>B</td>
<td>9:30~10:00</td>
</tr>
<tr>
<td>C</td>
<td>10:00~11:00</td>
</tr>
<tr>
<td>D</td>
<td>11:00~11:30</td>
</tr>
</tbody>
</table>

Conflict graph
(chromatic number?)

Compatibility graph
(clique cover?)

Gantt chart
Perfect Graphs

- Clique partitioning and graph coloring problems are NP-hard on general graphs, with the exception of perfect graphs

- Definition of perfect graphs
  - For every induced subgraph, the size of the maximum (largest) clique equals the chromatic number of the subgraph
  - Examples: bipartite graphs, chordal graphs, etc.
    - Chordal graphs: every cycle of four or more vertices has a chord, i.e., an edge between two vertices that are not consecutive in the cycle.
Interval Graph

- Intersection graphs of a (multi)set of intervals on a line
  - Vertices correspond to intervals
  - Edges correspond to interval intersection
  - A special class of chordal graphs

[Figure source: en.wikipedia.org/wiki/Interval_graph]
Left Edge Algorithm

- Problem statement
  - Given: Input is a group of intervals with starting and ending time
  - Goal: Minimize the number of colors of the corresponding interval graph

```
Repeat
  create a new color group c
  Repeat
    assign leftmost feasible interval to c
  until no more feasible interval
until no more interval

Interval are sorted according to their left endpoints
```

Greedy algorithm, O(nlogn) time
Left Edge Demonstration

Lifetime intervals with a given schedule

Assign colors (or tracks) using left edge algorithm

Corresponding colored conflict graph
Binding Impact on Multiplexer Network

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul1</td>
<td>op1, op3</td>
</tr>
<tr>
<td>AddSub1</td>
<td>op2, op4</td>
</tr>
<tr>
<td>AddSub2</td>
<td>op5, op6</td>
</tr>
</tbody>
</table>

Binding 1

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul1</td>
<td>op1, op3</td>
</tr>
<tr>
<td>AddSub1</td>
<td>op2, op4, op6</td>
</tr>
<tr>
<td>AddSub2</td>
<td>op5</td>
</tr>
</tbody>
</table>

Binding 2
Binding Summary

- Resource sharing directly impacts the complexity of the resulting datapath
  - # of functional units and registers, multiplexer networks, etc.

- Binding for resource usage minimization
  - Left edge algorithm: greedy but optimal for DFGs
  - **NP-hard problem with the general form of CDFG**
    - Polynomial-time algorithm exists for SSA-based register binding, although more registers are required

- Connectivity binding problem (e.g., multiplexer minimization) is NP-Hard
Next Lecture

- Pipelining
Acknowledgements

- These slides contain/adapt materials developed by
  - Prof. Deming Chen (UIUC)