ECE 5775
High-Level Digital Design Automation
Fall 2018

More Scheduling
Resource Sharing
Announcements

- Lab 3 is released (due Friday 10/5)
  - 10 FPGA boards available
  - Go through the CORDIC tutorial asap
Outline

- More SDC scheduling

- Resource sharing overview
  - Sub-problems: functional unit, register, and connectivity binding problems
  - Key concepts: compatibility and conflict graphs
Review: ILP Formulation for TCS

- ILP for time-constrained scheduling

\[
\text{minimize } c^T y
\]

\[
x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} - y_1 \leq 0
\]

\[
x_{6,2} + x_{7,2} + x_{8,2} - y_1 \leq 0
\]

\[
x_{7,3} + x_{8,3} - y_1 \leq 0
\]

\[
x_{5,4} + x_{9,4} + x_{11,4} - y_2 \leq 0
\]

... 

What is the \( y \) vector?

\[y_1: \text{# of multis required}\]

\[y_2: \text{# of AVUs required}\]
Review: SDC-Based Scheduling

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

\[ s_i : \text{schedule variable for operation } i \]

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

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

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

\[
\begin{align*}
    s_0 - s_4 &\leq 0 \\
    s_1 - s_3 &\leq 0 \\
    s_2 - s_3 &\leq 0 \\
    s_3 - s_4 &\leq 0 \\
    s_4 - s_5 &\leq 0 \\
    s_2 - s_4 &\leq -1 \\
    s_1 - s_4 &\leq -1 \\
    s_4 - s_2 &\leq 0
\end{align*}
\]
Handling Resource Constraints (NP-Hard in General)

- Resource constraints cannot be represented exactly in integer difference form

- Resource constraints
  - Heuristic partial orderings
  \[ v_0 \rightarrow v_2 : s_0 - s_2 \leq -1 \]  
  \[ v_1 \rightarrow v_0 : s_1 - s_0 \leq -1 \]  
  \[ v_2 \rightarrow v_0 : s_2 - s_0 \leq -1 \]

- 3 cycle latency
- 2 cycle latency

- Resource constraint
  - Two read ports

[J. Cong & Z. Zhang, DAC, 2006] [Z. Zhang & B. Liu, ICCAD, 2013]
Exact and Practically Scalable Scheduling with SDC and SAT (SDS)

Partial orderings

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

Difference constraints

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

Resource Constraints

SAT

Conflict based search

~1M variables

>1M clauses

Graph based feasibility checking

Polynomial time

SDC Timing Constraints

Conflict clauses

Conflict-driven learning

Infeasibility

[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

\( R_{uv} \) : whether operation \( u \) is sharing the same resource with operation \( v \)

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

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

\[
\begin{align*}
R_{01} & \rightarrow ( O_{0 \rightarrow 1} \vee O_{1 \rightarrow 0} ) \\
\neg( O_{0 \rightarrow 1} \land O_{1 \rightarrow 0} ) \\
R_{02} & \rightarrow ( O_{0 \rightarrow 2} \vee O_{2 \rightarrow 0} ) \\
\neg( O_{0 \rightarrow 2} \land O_{2 \rightarrow 0} ) \\
R_{12} & \rightarrow ( O_{1 \rightarrow 2} \vee O_{2 \rightarrow 1} ) \\
\neg( O_{1 \rightarrow 2} \land O_{2 \rightarrow 1} )
\end{align*}
\]

Note: \( R_{01} \rightarrow ( O_{0 \rightarrow 1} \vee O_{1 \rightarrow 0} ) \) means \( R_{01} \) implies \( ( O_{0 \rightarrow 1} \vee O_{1 \rightarrow 0} ) \)

Two read ports available
Conflict-Driven Learning

- Is 2-cycle schedule feasible?

\[ R_{01} \rightarrow (O_{0\rightarrow1} \lor O_{1\rightarrow0}) \]
\[ \neg( O_{0\rightarrow1} \land O_{1\rightarrow0} ) \]
\[ R_{02} \rightarrow (O_{0\rightarrow2} \lor O_{2\rightarrow0}) \]
\[ \neg( O_{0\rightarrow2} \land O_{2\rightarrow0} ) \]
\[ R_{12} \rightarrow (O_{1\rightarrow2} \lor O_{2\rightarrow1}) \]
\[ \neg( O_{1\rightarrow2} \land O_{2\rightarrow1} ) \]

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

- Is a 2-cycle schedule feasible?

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

\[ \text{Negative cycle sum} = -2 \]

 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

Is a 2-cycle schedule feasible?

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

Feasible! Returns schedule.
Fast Conflict-Driven Learning

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

$$\neg(0 \rightarrow_1 \land 0 \rightarrow_2 \land 0 \rightarrow_2)$$

- 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 specific to HLS
  - Applies to constrained scheduling problems in other fields
Recap: High-Level Synthesis Flow

High-level Programming Languages
(C/C++, SystemC, Matlab, ...)

Compilation

Transformations

Allocation

Scheduling

Binding

RTL generation

if (condition) {
    ...
} else {
    \texttt{t}_1 = a + b;
    \texttt{t}_2 = c \times d;
    \texttt{t}_3 = e + f;
    \texttt{t}_4 = \texttt{t}_1 \times \texttt{t}_2;
    \texttt{z} = \texttt{t}_4 - \texttt{t}_3;
}

Control data flow graph (CDFG)

Finite state machines with datapath

3 cycles
Resource Sharing and Binding

- Resource sharing: shares resources to minimize cost, in resource usage/area/power
  - Typically carried out by binding in high-level synthesis
  - 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

clock edge 1 2 3
Register Binding

Lifetime crossing clock edge;
Register Implied
Variable Lifetime Analysis

Variables $v_1$, $v_2$, and $v_3$ can share the same register.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Lifetimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>[1, 2)</td>
</tr>
<tr>
<td>$v_2$</td>
<td>[2, 3)</td>
</tr>
<tr>
<td>$v_3$</td>
<td>[3, 4)</td>
</tr>
</tbody>
</table>

Variable lifetimes

Clock edge

1  2  3  4
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: The 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
Before Next Class

- Next lecture: More Binding, Pipelining
Acknowledgements

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