ECE 5775 (Fall’17)
High-Level Digital Design Automation

More Scheduling
Resource Sharing
Logistics

- Lab 1 graded
  - Exploring integer bitwidth?

- Lab 3 due Friday 10/6
  - No late penalty for this assignment (up to 3 days late)

- First batch of *raw* quiz scores released on CMS
Agenda

- 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

**minimize** $c^Ty$

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

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

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

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

...

What is the $y$ vector?
SDC-Based Scheduling

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

\( s_i \) : 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
Handling Resource Constraints (NP-Hard in General)

- Target cycle time: 5ns
- Delay estimates
  - Add (+) 1ns
  - Load (ld) 3ns
  - Store (st) 1ns
- Two read ports available

$s_i$ : 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
  - $v_2 \rightarrow v_5 : s_2 - s_5 \leq -1$
  - $v_1 \rightarrow v_5 : s_1 - s_5 \leq -1$

- Resource constraints:
  - heuristic order
  - $v_0 \rightarrow v_2$ : 3 cycle latency
  - $v_2 \rightarrow v_0$ : 2 cycle latency
SDC Constraint Matrix

- The constraint matrix of SDC($X, C$) is a totally unimodular matrix (TUM):
  - Every nonsingular square submatrix has a determinant of $-1/+1$.

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & -1 \\
-1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & -1 & 0 \\
0 & 0 & -1 & 1 & 0
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ -4 \\ 3 \\ -1 \end{pmatrix}
\]

- Theorem (Hoffman & Kruskal, 1956): If $A$ is totally unimodular and $b$ is a vector of integers, every extreme point of polyhedron $\{x : Ax \leq b\}$ is integral.
  - Solving linear programming (LP) relaxation leads to integral solutions
Optimizing Linear Objectives

- **Objective function**
  - Represent the design objective in a linear function

- **Examples**
  - ASAP and ALAP
  - Minimum latency: \( \min \{\max[s(oi)]\} \)
  - Minimum average case latency (control-intensive design)
  - Minimum total variable lifetimes
  - Maximize node slacks
  ...

Example: ASAP and ALAP Scheduling

- ASAP objective: \( \min \sum_{v \in V_{op}} s(v) \)
- ALAP objective: \( \max \sum_{v \in V_{op}} s(v) \)

\[
\begin{align*}
\text{min } \{s(v_1) + \ldots + s(v_5)\} \\
\text{max } \{s(v_1) + \ldots + s(v_5)\}
\end{align*}
\]
Handling Control Dependencies

- Control dependencies can also be honored (i.e., no speculative executions)
  - If $bb_2$ is control dependent on $bb_1$, the operation nodes of $bb_2$ are not allowed to be scheduled before those of $bb_1$
  - Polarize each basic block $bb_i$ with two scheduling variables (head and tail)
    - $\forall v \in bb_i$, $s_h(bb_i) - s(v) \leq 0$
    - $\forall v \in bb_i$, $s(v) - s_t(bb_i) \leq 0$
  - If $e_c(bb_i, bb_j) \in E_c$ and $e_c$ is not a back edge
    - $s_t(bb_i) - s_h(bb_j) \leq 0$
Example: Greatest Common Divisor (GCD)

- GCD C description

```c
x = in1;
y = in2;
while (x != y) {
    if (x > y)
        x = x - y;
    else
        y = y - x;
}
*out = x;
```

```
x0 = in1
y0 = in2
cond1 = (x0 != y0)
```

```
x1 = Φ(x0, x1, x2)
y1 = Φ(y0, y1, y2)
cond2 = (x1 > y1)
```

```
x2 = x1 - y1
cond3 = (x2 != y1)
```

```
y2 = y1 - x1
cond4 = (x1 != y2)
```

```
x3 = Φ(x0, x1, x2)
*out = x3
```
Interpreting Solution from Linear Programming

$x_0 = \text{in1}$
$y_0 = \text{in2}$
$\text{cond1} = (x_0 \neq y_0)$

$x_1 = \Phi(x_0, x_1, x_2)$
$y_1 = \Phi(y_0, y_1, y_2)$
$\text{cond2} = (x_1 > y_1)$

$x_2 = x_1 - y_1$
$\text{cond3} = (x_2 \neq y_1)$

$y_2 = y_1 - x_1$
$\text{cond4} = (x_1 \neq y_2)$

$x_3 = \Phi(x_0, x_1, x_2)$
*out = $x_3$

All operations in the blue region are assigned a schedule label of 0.

All operations in the green region get assigned the same schedule label of 1.

Scheduling are performed across the basic block boundaries.
Operations and Predicates

Create additional predicates for conditionally executed operations in each state

\[
\begin{align*}
  x_0 &= \text{in1} \\
  y_0 &= \text{in2} \\
  \text{cond1} &= (x_0 \neq y_0)
\end{align*}
\]

\[
\begin{align*}
  x_1 &= \Phi (x_0, x_1, x_2) \\
  y_1 &= \Phi (y_0, y_1, y_2) \\
  \text{cond2} &= (x_1 > y_1) \\
  x_2 &= x_1 - y_1 \\
  \text{cond3} &= (x_2 \neq y_1) \\
  y_2 &= y_1 - x_1 \\
  \text{cond4} &= (x_1 \neq y_2) \\
  x_3 &= \Phi (x_0, x_1, x_2) \\
  \text{*out} &= x_3
\end{align*}
\]

\[
\begin{align*}
  x_0 &= \text{in1} \\
  y_0 &= \text{in2} \\
  \text{cond1} &= (x_0 \neq y_0)
\end{align*}
\]

\[
\begin{align*}
  x_1 &= \Phi (x_0, x_1, x_2) \\
  y_1 &= \Phi (y_0, y_1, y_2) \\
  \text{cond2} &= (x_1 > y_1) \\
  \text{if (cond2)} \{ \\
    &\quad x_2 = x_1 - y_1 \\
    &\quad \text{cond3} = (x_2 \neq y_1) \\
  \} \text{ else } \{ \\
    &\quad y_2 = y_1 - x_1 \\
    &\quad \text{cond4} = (x_1 \neq y_2)
\}
\]

\[
\begin{align*}
  \text{if(!cond1 || (!cond3 && !cond4))} \{ \\
    &\quad x_3 = \Phi (x_0, x_1, x_2) \\
    &\quad \text{*out} = x_3
  \}
\]
Deriving State Transition Graph (STG)

Final STG for GCD
- Predicates for operations and transitions can be derived from original control flow + dominator analysis

\[
\begin{align*}
    x_0 &= \text{in1} \\
    y_0 &= \text{in2} \\
    \text{cond1} &= (x_0 \neq y_0)
\end{align*}
\]

if (\text{cond1}) {
    \begin{align*}
        x_1 &= \Phi (x_0, x_1, x_2) \\
        y_1 &= \Phi (y_0, y_1, y_2) \\
        \text{cond2} &= (x_1 > y_1)
    \end{align*}
\}

if (\text{cond2}) {
    \begin{align*}
        x_2 &= x_1 - y_1 \\
        \text{cond3} &= (x_2 \neq y_1)
    \end{align*}
\}
else {
    \begin{align*}
        y_2 &= y_1 - x_1 \\
        \text{cond4} &= (x_1 \neq y_2)
    \end{align*}
\}

if (!\text{cond1} || (!\text{cond3} && !\text{cond4})) {
    \begin{align*}
        x_3 &= \Phi (x_0, x_1, x_2) \\
        \text{*out} &= x_3
    \end{align*}
\}

\text{cond1} && (\text{cond3} || \text{cond4})
Constrained Scheduling Summary

- **ILP**
  - Exact, but exponential worst case runtime

- **Hu’s algorithm**
  - Optimal and polynomial
  - Only works in restricted cases

- **List scheduling**
  - Extension to Hu’s for general case
  - Greedy (fast) but suboptimal

- **SDC-based scheduling**
  - A versatile heuristic based on LP formulation with different constraints
  - Amenable to global optimization
Case Study: Memcached on FPGAs

Module implementations with HLS:
- Hash Table
- Value Store

Source: K, Karras, M. Blott, and K. Vissers, FSP’2014
Memcached: HLS vs. Manual RTL

Resource Usage

Variable but clearly downward trend when using HLS

Performance

HLS superior in all scenarios

Source: K. Karras, M. Blott, and K. Vissers, FSP’2014
Effectiveness of Human Scheduling?

**Given:** A chain of n operations. Without any registers, the cycle time equals the total combinational delay, which is \( D = \text{sum}(d_i) \).

\[ \rightarrow d_1 \rightarrow d_2 \rightarrow d_3 \rightarrow \ldots \rightarrow d_N \]

**Question:** How to place 3 registers on the chain to achieve the minimum cycle time?

**Example:**

\[ \rightarrow 5 \rightarrow 3 \rightarrow 8 \rightarrow 2 \rightarrow 7 \]

Manual trial-and-error is not effective for large complex graphs.
Recap: High-Level Synthesis Flow

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

Compilation

Transformations

Allocation

Scheduling

Binding

RTL generation

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

Control data flow graph (CDFG)

Finite state machines with datapath

3 cycles
Resource Sharing and Binding

- Resource sharing: 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

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

- Functional unit (FU) binding
  - Primary objective is to minimize the number of FUs
  - 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**: \([\text{birth-time, death-time}]\)
    - Operation: The whole execution time (if unpipelined)
    - Variable: From the time this variable is defined to the time it is last used
**Operation Binding**

![Diagram of 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

Lifetime crossing clock edge; Register Implied
Variable Lifetime Analysis

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

- v1: [1, 2)
- v2: [2, 3)
- v3: [3, 4)

Variable lifetimes

<table>
<thead>
<tr>
<th>Variable</th>
<th>Lifetimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>[1, 2)</td>
</tr>
<tr>
<td>v2</td>
<td>[2, 3)</td>
</tr>
<tr>
<td>v3</td>
<td>[3, 4)</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: 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)