Lecture 7

- Allocation and binding
- Control unit synthesis
- Advanced HLS issues
Allocation and Binding

- Allocation (unit selection) — To determine the type and number of hardware resources required, including:
  - Functional units
  - Storage elements
  - Buses

- Binding — Assignment to resource instances:
  - Operations to functional unit instances
  - Values to be stored to instances of storage elements
  - Data transfers to bus instances

- Allocation and binding generate the datapath of the design.

Allocation and Binding Principle

- Resource sharing: Allow multiple non-concurrent operations to share the same hardware as much as possible.
- Optimization goal:
  - Minimize total cost of functional units, registers, bus drivers, and multiplexers.
  - Minimize total interconnection length (placement info needed).
  - Constraint on critical path delay.
Allocation/Binding — Approach 1

- **Constructive** — start with an empty datapath and add functional, storage and interconnection components as needed.
  - Greedy algorithms — perform allocation/binding for one control step at a time.

- **Greedy algorithms**

- **Rule-based** — used to select type and numbers of function units, especially prior to scheduling.

Allocation/Binding — Approach 2

- **Graph-theoretical formulations** — Sub-tasks are mapped into well-defined problems in graph theory.
  - Clique partitioning.
  - Left-edge algorithm.
  - Graph coloring.
Clique Partitioning

- $G = (V, E)$, an undirected graph with a set $V$ of vertices and a set $E$ of edges.
- A clique is a set of vertices that form a complete subgraph of $G$.
- The Clique Partitioning Problem: To partition $G$ into a minimal number of cliques such that each vertex belongs to exactly one clique.

Allocation as Clique Partitioning

Functional unit allocation:
- Each vertex represents an operation.
- An edge connects two vertices iff:
  - The two operations are scheduled into different control steps, and
  - There exists a functional unit that is capable of carrying out both operations.
S. Allocation as Clique Partitioning

- Storage allocation as a clique partitioning problem:
  - Each value needed to be stored is mapped to a vertex.
  - Two vertices are connected, iff the life-times of the two values do not intersect.

- The clique partitioning problem is NP-complete.

- Efficient heuristics must be developed.
  - Ex. Tseng developed a polynomial time algorithm, based on step-wise grouping, which generates very good results.

Tseng’s Algorithm

- A super-graph is derived from the original graph.
- Find two connected super-nodes such that they have the maximum number of common neighbors.
- Merge the two nodes and repeated from the first step, until no more merger can be carried out.

Edge | Common neighbors
--- | ---
(V1,V3) | 1
(V1,V4) | 1
(V2,V3) | 0
(V2,V5) | 0
(V3,V4) | 1
(V4,V5) | 0
Tseng’s Algorithm (Cont’d)

Left-Edge (LE) Algorithm

- Used in channel routing to minimize the number of tracks used to connect points (layout design).

  - To minimize the number of needed tracks.
  - To reduce wire lengths.
  - To avoid wire crossings.
**LE Algorithm for Reg. Allocation**

- Map birth time of a value to the left (top) edge, and its death time to the right (down) edge of a wire.

---

**The Left-Edge Algorithm**

1. The values are sorted in increasing order of their birth times.
2. The first value is assigned to the first register.
3. The list is then scanned for the next value whose birth time is equal to or larger than the death time of the previous value.
4. This value is assigned to the current register.
5. The list is scanned until no more value can share the same register.
6. A new register is then introduced to hold the next value in the sorted list, and the algorithm iterates from step 3.
LE Algorithm Example

Original life-times

Sorted list based on birth times

Allocated registers

LE Algorithm Discussions

- The algorithm guarantees to allocate the minimum number of registers.

- However, it has two disadvantages:
  - Not all life-time table can be interpreted as intersecting intervals on a line.
    - Loop
    - Conditional branches
  - The assignment is neither unique, nor necessarily optimal, in terms of minimal number of multiplexers, for example.
**Allocation/Binding — Approach 3**

- **Transformational allocation** — starting from an initial allocation and binding, a final design is obtained by successive transformations.
  - Usually it starts with a maximal allocation (each operation has its dedicated physical unit).
  - The design is then improved by merging, step-by-step, physical units so that hardware resources are shared as much as possible.

![Diagram showing the transformation of allocation and binding]

---

**Lecture 7**

- Allocation and binding
- Control unit synthesis
- Advanced HLS issues
Control-Unit Synthesis

- Two basic approaches are widely used:
  - Microcode.
  - Hard-wired.
- The basic assumptions:
  - A synchronous controller is used.
  - A schedule is given with the set of activation signals
    • for enabling, multiplexer input selection, bus control, etc.
  - The controller is modeled as a finite-state machine.

Microcoded Control Synthesis

- To store the control information in an organized fashion.
- A microcode ROM of size $\lambda$ is used, where $\lambda$ is the number of schedule steps.
- The ROM must have $\lceil \log_2 \lambda \rceil$ address bits (note: $\lceil x \rceil$ denotes the ceiling function).
- A synchronous counter with a reset signal is used to address the ROM.
- The counter is controlled by the system clock.
- The ROM contents can be implemented as horizontal or vertical microcode.
**Horizontal Microcode**

- Each activation signal is associated to one bit of the word in the microcode.

<table>
<thead>
<tr>
<th>Address</th>
<th>Microwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>11000101010</td>
</tr>
<tr>
<td>01</td>
<td>00100010101</td>
</tr>
<tr>
<td>10</td>
<td>00010000000</td>
</tr>
<tr>
<td>11</td>
<td>00001000000</td>
</tr>
</tbody>
</table>

- The word length is usually much larger than $\lambda$, and the ROM has therefore a width larger than its height.
- Each bit is connected directly to an activation signal — high performance.
- There are many zeros — wasted storage resource.

**Vertical Microcode**

- A fully vertical microcode encodes the $n$ activation signals with $\lceil \log_2 n \rceil$ bits to reduce the width of the ROM.
  - Several words may be needed for a schedule step.

<table>
<thead>
<tr>
<th>Activation signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 (n = 11)</td>
</tr>
<tr>
<td>11000101010</td>
</tr>
<tr>
<td>00100010101</td>
</tr>
<tr>
<td>00010000000</td>
</tr>
<tr>
<td>00001000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
</tr>
<tr>
<td>0111</td>
</tr>
<tr>
<td>1001</td>
</tr>
<tr>
<td>1011</td>
</tr>
<tr>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
</tr>
</tbody>
</table>
Vertical Microcode Issues

- A decoder is needed, which can be implemented by another ROM to form a two-stage control store.

- Operation concurrency may not be fully supported.
  - Reserve code-words for concurrent operations.
    - e.g., using “1100” to denote activation of the first group of activation signals.

- Vertical control schemes can be implemented by:
  - Lengthening the schedule, or
  - Reading multiple ROM words in each step.
  - Both have, however, some disadvantages.

Microcode Optimization

- To find the shortest encoding of the words such that full concurrency is preserved — the microcode compaction problem (an intractable problem).

- MC can be approached by partitioning the operations into groups such that only one operation is active in each group and therefore vertical encoding can be used in it.
Microcode Compaction

- To minimize the number of groups.
- Construct a conflict graph, where the vertices correspond to the operations and the edges represent concurrency.
- A minimum coloring of this graph gives the minimum number of groups needed.
- Note: this does not necessarily lead to the minimum number of word bits (e.g., 10 can be divided as 5+5, or 7+3).

Hard-Wired Control Synthesis

- Generate a Moore-type finite-state machine from a schedule.
- Synthesize the FSM model.
Advanced HLS issues

- Many-to-many mapping between operations and physical components.
- Re-use of previous designs (partial structure).
- Synthesis with commercially available sub-systems, IP-based synthesis.
- HLS with testability consideration.
Summary

- High-level synthesis is one of the most important design steps in the design process of electronic systems.
- The use of efficient HLS tools has led to the great improvement of design productivity.
- The two most important tasks are scheduling and allocation/binding, which are interdependent.
- Controller design is also an important task, and its interaction with datapath design should be considered.
- The HLS tasks are usually formulated as optimization problems and heuristic algorithms are used.