**Introduction: Binding**

- Part of a 4-lecture introduction
  - Scheduling
  - Resource binding
  - Area and performance estimation
  - Control unit synthesis
- This lecture covers
  - Resources and resource types
  - Resource sharing and binding
  - Graph models of resource binding
  - Conflict graphs
  - Templates for architectural synthesis
  - A complete worked problem

**Resources**

- We refer to a piece of hardware that can perform a specific function as a “resource”
  - e.g. a 16x16-bit multiplier, a PCI interface
- An operation could be performed on one of several resources
  - e.g. a multiplication could be performed on one of two physically distinct multipliers
  - e.g. an addition could be performed by a special-purpose adder, or an ALU.
- We are distinguishing here between the operation, and the resource that will execute that operation

**Resource Types**

- The “type” of a resource denotes its ability to perform different operations
  - A multiplier can do multiplications
  - An adder can do additions
  - An ALU can do comparisons and additions
- The resource type set $R$ consists of all the different resource types we have available
  - $R = \{\text{multiplier, adder, ALU}\}$

**Resource Sharing**

- Just because we have $n$ additions in an algorithm, we don’t need $n$ adders
  - In traditional sequential processors, we use just a single adder to do all the additions in our program
  - This is possible because we have scheduled them – an adder is only used for one addition at one time
- Using the same resource to perform several different operations is “resource sharing”
- Advantage: can save area and peak power.
- Disadvantage: can make things slower and use more energy.
Resource Sharing

- Consider the code below and its scheduled DFG

\[
\begin{align*}
 a &= b + 2; \\
 c &= a + 7;
\end{align*}
\]

- We could use two adders or one shared adder

\[
\begin{align*}
 b &\rightarrow + &\rightarrow D &\rightarrow + &\rightarrow c \\
 2 &\rightarrow & & &\rightarrow a \\
 b &\rightarrow + &\rightarrow D &\rightarrow + &\rightarrow a/c \\
 7 &\rightarrow & & &\rightarrow 2
\end{align*}
\]

One fewer adder but 2 more MUXs, possibly worse max clock rate

Need to generate select signals

Resource Binding

- Resource binding is the process of deciding which resource should perform which operation

- Cartesian product
  - \( \times \) denotes the Cartesian product of two sets
  - \( A \times B = \{ (a,b) | a \in A, b \in B \} \)
  - e.g. \( \{a,b\} \times \{1,2\} = \{(a,1),(a,2),(b,1),(b,2)\} \)

- A resource binding is a function \( Y: V \rightarrow R \times N \)

Resource Binding

- Revisiting our example…

\[
\begin{align*}
 v_1 &\rightarrow v_2 \\
 b &\rightarrow +,1 &\rightarrow D &\rightarrow +,2 &\rightarrow c \\
 2 &\rightarrow & & &\rightarrow a \\
 v_1 &\rightarrow +,1 \\
 b &\rightarrow +,1 &\rightarrow D &\rightarrow +,2 &\rightarrow a/c \\
 7 &\rightarrow & & &\rightarrow 2
\end{align*}
\]

\( Y(v_1) = (+,1) \)
\( Y(v_2) = (+,2) \)

Binding Graphs

- A hypergraph extends the notion of a graph by allowing edges to be incident to any number of nodes

- We can represent a bound CDFG or DFG by a hypergraph \( G'(V,E \cup E_B) \)

\[
\begin{align*}
 v_1 &\rightarrow v_2 \\
 v_1 &\rightarrow v_2 \\
 v_1 &\rightarrow v_2
\end{align*}
\]

\[
\begin{align*}
 E &= \{ (v_1,v_2) \} \\
 E_B &= \{ \{v_1\}, \{v_2\} \}
\end{align*}
\]

\[
\begin{align*}
 v_1 &\rightarrow v_2 \\
 v_1 &\rightarrow v_2 \\
 v_1 &\rightarrow v_2
\end{align*}
\]

\[
\begin{align*}
 E &= \{ (v_1,v_2) \} \\
 E_B &= \{ \{v_1,v_2\} \}
\end{align*}
\]
Conflict Graphs

- Sometimes we must bind operations to different resources
  - e.g. if they execute at the same time
- Such information can be represented using conflict graphs
- These have the same nodes as the corresponding DFG or CDFG.
- An edge corresponds to a conflict
  - two nodes connected by an edge cannot be bound to the same resource

Conflict Graphs

- In this example, the structure of the conflict graph is very simple
  - two disjoint sets of nodes, each one fully connected within itself
- This is because all operations took a single cycle – with multicycle operations, conflict graphs become more interesting and important (a later lecture…)

Architectural Templates

- Once we have a schedule S and a resource binding Y, we know all we need to construct our circuit
- In order to do this, the synthesis tool needs to have a “template” in mind
- We will be working with register bus-based architectures: in one clock cycle
  - values are read from registers, pass through multiplexers, and get steered to the right resource
  - the operations are performed
  - the results are written back into the registers
Architectural Templates

A register is enabled when it should be written to in that clock cycle. The select-lines decide which register to send to each resource. Some resources may require additional control.

Worked Problem

- Consider the following code:
  \[ a = i + j; \ b = 2 \times a + j; \ c = a \times b; \ d = a^2; \]

  - (a) construct a CDFG for the code
  - (b) schedule the graph so that each operation starts as soon as it can, assuming each multiplication takes two cycles and each addition takes one cycle
  - (c) if you have the resource type set \( R = \{ \text{adder}, \text{multiplier} \} \), construct a resource binding for this example
  - (d) draw the completed data-path
  - (e) suggest a way you could save area

Worked Problem (a-c)

Y(v1) = (adder, 1)
Y(v2) = (multiplier, 1)
Y(v3) = (adder, 1)
Y(v4) = (multiplier, 1)
Y(v5) = (multiplier, 2)

Cycle: 1 2 3 4 5 6

Worked Problem (d)

From control unit

(a) (adder, 1) ( multiplier, 1) ( multiplier, 2)
Worked Problem (e)

• Area could be saved by scheduling v5 in cycles 4-5, and v4 in 6-7, at the penalty of one clock cycle

• (actually if we pipelined one of the multipliers, we wouldn’t have to pay any penalty…)

Summary

• This lecture has covered
  – Resources and resource types
  – Resource sharing and binding
  – Graph models of resource binding
  – Conflict graphs
  – Templates for architectural synthesis
  – A complete worked problem

• Later in the course, we will be examining algorithms to perform automatic binding

Suggested Problems

• De Micheli, Problems 4.11, Q5 (assume all additions take one cycle) (**)

• For the binding hypergraph shown in De Micheli, Fig. 4.5, construct a datapath design (you may label your registers in any way) (*)