Sharing vs. Binding

- **Resource Sharing**
  - Assignment of a resource to more than one operation
  - Goal – reduce area by allowing multiple non-concurrent operations to share the same hardware operator

- **Resource Binding**
  - Explicit mapping between operations and resources

**Example 1**

![Diagram](image1)

- Resource Sharing
  - We have 3 add operations and 2 adder units

- Resource Binding
  - Add op1 and op2 executes on adder unit 1
  - Add op3 executes on adder unit 2

**Example 2**

![Diagram](image2)

- Adders available = 1
- Adders available = 2

**Example 3**

![Diagram](image3)

- At most 2 add operations can be executed in a time slice, latency = 2

**Example 4**

![Diagram](image4)

- At most 1 add operation can be executed in a time slice, latency = 4
Sharing and Binding for Resource Dominated Circuits

- We are interested in the set of vertices of the sequencing graph (omit source/sink nodes)
- How much sharing is possible?
- Two or more operations can be bound to the same resource if they are compatible
  - Not concurrent
  - Can be implemented with the same resource type

\[
a = b + c \\
e = a + 5
\]

Two operations are NOT concurrent if
- Either one starts after the other has finished
- Alternative choices (mutually exclusive) of a branching decision

\[
\begin{align*}
a &= b + c \\
e &= a + 5 \\
f &= c + 1 \\
g &= c + 3 \\
\end{align*}
\]

Clique Partitioning

```plaintext
CLIQUE_PARTITION( G(x, e) )
\{Π = ⊖ \}
while G(x, e) not empty \{ do( \)
   C = MAX_CLIQUE( G(x, e) ) \}
   Π = Π U C \}
delete C from G(x, e) \}
while( G(x, e) not empty ) do{ \}

MAX_CLIQUE( G(x, e) )
\{C = vertex with largest degree \}
repeat \{ repeat \{ \}
   U = { v ∈ V : v ∈ C and adjacent to all vertices of C } \}
   if U ≠ Φ \{ \}
      delete C from G(x, e) \}
   else{ \}
      select v ∈ U \}
      C = C U v \}
\}
\}
```

Resource Compatibility Graph

- Graph whose set of vertices is a one-to-one correspondence with operations in the sequencing graph and whose edges denotes the compatible operations pairs

Compatibility Graph Shows Resource Sharing

- As many disjoint (no common elements) components as resource types
  - A multiply operations is not compatible with an add operation
  - Each vertices connected to every other vertices
- Clique - group of mutually compatible operations correspond to subset of vertices that are mutually connected
- Maximal set of mutually compatible operations are represented by maximal clique

The optimum resource sharing is one that minimizes the number of required resource instances
- Resource instance relates to cliques
- Partitioning graph into minimum number of cliques yields optimal sharing

Clique Partitioning

- Initial set of partitions to empty
- While the graph is not empty, keep iterating
- Compute a maximal clique in graph
- Add max clique to set of partitions
- Remove max clique from graph

Maximize size of cliques, must ensure all vertices included
- (1, 3, 7)
- (2, 6, 8)
- (1, 8)
- (9, 5, 10, 11)
- (9)

# Resources = # cliques
We need 2 adders, 2 multipliers
### Clique Partitioning

**Example 1**

**Π = Φ**  // set of partitions is initially empty

**Is G empty? No.**

**Find max clique**

- \[ C = 1 \]  // vertex with largest degree, anything with 4 will do
- \[ U = \{3, 7, 6, 8\} \]  // these vertices are connected to 1
- \[ V = 3 \]
- \[ C = (1) \cup (3) = (1, 3) \]
- \[ U = \{2, 8\} \]  // these vertices are connected to 1 and 3
- \[ C = (1, 3) \cup (7) = (1, 3, 7) \]
- \[ U = \emptyset \]  // no others vertices connect to 1, 3, and 7

Return \(\{1, 3, 7\}\)

**Π = \{1, 3, 7\}**

Remove \(\{1, 3, 7\}\) from G

---

**Vertices** | **Degree**
--- | ---
1 | 4
2 | 4
3 | 4
7 | 4
8 | 4
10 | 4
11 | 4

---

---

### Clique Partitioning

**Example 1**

**Π = \{1, 3, 7\}, \{4, 5, 10, 11\}**

**Is G empty? No.**

**Find max clique**

- \[ C = 4 \]  // vertex with largest degree, anything with 4 will do
- \[ U = \{5, 9, 10, 11\} \]  // these vertices are connected to 4
- \[ V = 5 \]
- \[ C = (4) \cup (5) = (4, 5) \]
- \[ U = \{10, 11\} \]  // these vertices are connected to 4 and 5
- \[ C = (4, 5) \cup (10) = (4, 5, 10) \]
- \[ U = \{11\} \]  // these vertices are connected to 4, 5, and 10
- \[ C = (4, 5, 10) \cup (11) = (4, 5, 10, 11) \]
- \[ U = \emptyset \]  // no others vertices connect to 4, 5, 10, and 11

Return \(\{4, 5, 10, 11\}\)

**Π = \{1, 3, 7\}, \{4, 5, 10, 11\}\**

Remove \(\{4, 5, 10, 11\}\) from G

---

---

### Clique Partitioning

**Example 1**

**Π = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}**

**Is G empty? No.**

**Find max clique**

- \[ C = 2 \]  // vertex with largest degree, anything with 2 will do
- \[ U = \{6, 8\} \]  // these vertices are connected to 2
- \[ V = 6 \]
- \[ C = (2) \cup (6) = (2, 6) \]
- \[ U = \{8\} \]  // these vertices are connected to 2 and 6
- \[ C = (2, 6) \cup (8) = (2, 6, 8) \]
- \[ U = \emptyset \]  // no others vertices connect to 2, 6, and 8

Return \(\{2, 6, 8\}\)

**Π = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}\**

Remove \(\{2, 6, 8\}\) from G

---

---

### Clique Partitioning

**Example 1**

**Π = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}, \{9\}**

**Is G empty? No.**

**Find max clique**

- \[ C = 9 \]  // vertex with largest degree, anything with 9 will do
- \[ U = \{9\} \]
- \[ U = \emptyset \]  // no others vertices connect to 9

Return \(\{9\}\)

**Π = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}, \{9\}\**

Remove \(\{9\}\) from G

---

---
Clique Partitioning
Example 1

\[ \Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}, \{9\} \]

Is \( G \) empty? Yes!

- What does clique partition tell us?
  - \( \{1, 3, 7\} \) – multiplier
  - \( \{4, 5, 10, 11\} \) – alu
  - \( \{2, 6, 8\} \) – multiplier
  - \( \{9\} \) – alu

- We know how much sharing AND binding

Clique Partitioning
Example 2

\[ \Pi = \phi \]

Is \( G \) empty? No.

Find max clique

- \( C = 4 \)
- \( U = \{1, 2, 3, 5, 6\} \)
- \( V = \{4\} \)
- \( C = (U \cap V) = \{4\} \)
- \( U = \{5, 6\} \)
- \( C = (U \cap V) = \{5, 6\} \)
- \( U = \{6\} \)
- \( C = (U \cap V) = \{6\} \)
- \( U = \phi \)

Return \( \{1, 4, 5, 6\} \)

\[ \Pi = \{1, 4, 5, 6\} \]

Remove \( \{1, 4, 5, 6\} \) from \( G \)

Clique Partitioning
Example 2

\[ \Pi = \{1, 4, 5, 6\}, \{2\} \]

Is \( G \) empty? Yes

- Need 3 ALUs
  - ALU 1 executes ops 1, 4, 5, 6
  - ALU 2 executes op 2
  - ALU 3 executes op 3
Resource Conflict Graph

- Instead of compatibility we can instead look at conflicts
  - May simplify the graph
- Resource conflict graph
  - Graph whose set of vertices is a one-to-one correspondence with operations in the sequencing graph and whose edges denote the conflicting operations pairs
  - To simplify graph, we consider conflicts between each resource type independently

Building Resource Conflict Graph

- To simplify graph, we consider conflicts between each resource type independently

Graph Coloring

```c
VERTEX_COLOR ( G(v, e) )
for i=1 to |V|
  C = 1
  // use number to represent color
  while there exists a vertex adjacent to v, with color c do(
    C = C + 1
  )
label v, with C
```

- Use graph coloring to find independent sets
  - Each color represents a resource instance (two adders will be represented by two different colors)
  - Optimal resource sharing corresponds to vertex coloring with minimal amount of colors
Graph Coloring
Example 1

\[ \text{i} = 1 \quad \text{// look at vertex 1} \]
\[ C = c_1 \quad \text{// represents first color} \]
\[ \text{Is there any adjacent vertices with color = 1? No.} \]
\[ v_1 = c_1 \]

\[ \text{i} = 2 \quad \text{// look at vertex 2} \]
\[ C = c_1 \]
\[ \text{Is there any adjacent vertices with color = 1? Yes.} \]
\[ C = c_2 \]
\[ \text{Is there any adjacent vertices with color = 2? No.} \]
\[ v_2 = c_2 \]

\[ \text{i} = 3 \quad \text{// look at vertex 3} \]
\[ C = c_1 \quad \text{// represents first color} \]
\[ \text{Is there any adjacent vertices with color = 1? No.} \]
\[ v_3 = c_1 \]

\[ \text{i} = 4 \quad \text{// look at vertex 4} \]
\[ C = c_1 \quad \text{// represents first color} \]
\[ \text{Is there any adjacent vertices with color = 1? Yes.} \]
\[ C = c_2 \]
\[ \text{Is there any adjacent vertices with color = 2? Yes.} \]
\[ C = c_3 \]
\[ \text{Is there any adjacent vertices with color = 3? No.} \]
\[ v_4 = c_3 \]

\[ \text{Similarly repeat for remaining} \]

\[ \text{Four colors required – need four resources} \]
\[ \bullet \ c_1 \text{ is used for multiply} \]
\[ \bullet \ c_2 \text{ is used for multiply} \]
\[ \bullet \ c_3 \text{ is used for alu} \]
\[ \bullet \ c_4 \text{ is used for alu} \]

Graph Coloring
Example 2

\[ \text{i} = 1 \quad \text{// look at vertex 1} \]
\[ C = c_1 \]
\[ \text{Adjacent vertices with color = 1? No.} \]
\[ v_1 = c_1 \]

\[ \text{i} = 2 \quad \text{// look at vertex 2} \]
\[ C = c_1 \]
\[ \text{Adjacent vertices with color = 1? Yes.} \]
\[ C = c_2 \]
\[ \text{Adjacent vertices with color = 2? No.} \]
\[ v_2 = c_2 \]

\[ \text{i} = 3 \quad \text{// look at vertex 3} \]
\[ C = c_1 \]
\[ \text{Adjacent vertices with color = 1? No.} \]
\[ v_3 = c_1 \]

\[ \text{i} = 4 \quad \text{// look at vertex 4} \]
\[ C = c_2 \]
\[ \text{Adjacent vertices with color = 2? Yes.} \]
\[ C = c_3 \]
\[ \text{Adjacent vertices with color = 3? Yes.} \]
\[ C = c_4 \]

\[ \text{Four colors required – need four resources} \]
\[ \bullet \ c_1 \text{ for node 1, 3 op} \]
\[ \bullet \ c_2 \text{ for node 2, 4 op} \]
\[ \bullet \ c_3 \text{ for node 5 op} \]
\[ \bullet \ c_4 \text{ for node 6 op} \]

\[ \text{VERTEX_COLOR algorithm sensitive to ordering of vertices explored - variety of} \]
\[ \text{modifications available} \]
\[ \bullet \text{ Switching pair assignment of colors} \]
\[ \bullet \text{ Backtracking to switching larger number of vertices} \]

\[ \text{Node ordering 1, 2, 3, 4, 5} \]
\[ \text{Requires 2 colors} \]

\[ \text{Node ordering 1, 5, 2, 3, 4} \]
\[ \text{Requires 3 colors} \]
Conclusion

- Considered several types ways to find resource sharing and binding
  - Compatibility Graph / Max Clique
  - Conflict Graph / Vertex color

- Again, many other methods available
  - Golumbic’s algorithm
  - Left-edge algorithm
  - ILP formulation

- Idea of sharing and binding not limited to adders and multipliers
  - Registers
  - Determining minimal number of memory ports
  - Bus sharing