Lecture 20: High-level Synthesis (1)

Deming Chen

Some slides are from Prof. S. Levitan of U. of Pittsburgh
Outline

- High-level synthesis introduction
- High-level synthesis operations
  - Scheduling
    - ASAP and ALAP
    - List scheduling
  - Allocation
  - Binding
- Loop unrolling and pipelining
High-level Synthesis: Silicon Compilation?

- In the 1980’s silicon compilation was the “holy grail” of EDA/CAD researchers – and industry
- The idea was oversold – the technology, techniques, algorithms and compute platforms available at the time could not (in general) do the job.
- New languages, new algorithms, faster computers and lower expectations have re-invented the idea as “high-level synthesis”
- Ideally we can now go from an algorithm to silicon
What is High-level Synthesis?

- Input is a high level, algorithmic description
  - Control structures (if/then, loop, subroutines)
  - Concurrent and sequential semantics
  - Abstract data types
  - Logical and arithmetic operators

- A set of constraints
  - Speed, power, area, interconnect style, coding style
  - A library of pre-specified components

- Output is an RTL description for further synthesis and optimization phases
High-level Synthesis vs. RTL Coding

- Ideally want to move from behavioral model directly to synthesized design
- Automatically select “architecture” and bypass RTL coding loop
HLS Output

Data Path

Register
Register
Register
Adder
Multiplier
Shifter
Logic Unit
misc
misc
misc

Control (Finite State Machine)
Behavior in – RTL out

Figure 2-5: Architectural alternatives from a behavioral description
Design Space Exploration

**VLSI Postulate:** For any realization of a specific algorithm: $AT^2 = \text{constant}$
(C.D. Thompson & H.T. Kung)

Ideal: Area = const/sqrt(Time)

Real Designs
High-level Synthesis Process

- **Scheduling**
  - Scheduling the operations in the CDFG to minimize area, time and/or power

- **Allocation**
  - Allocating resources (library components) to each of the operations, buses, muxes, and registers for storage

- **Binding**
  - Determining the time of use of each component
  - e.g. which register used when
Synthesis Steps

- Behavioral Description
- CDFG Generation
- Resource Allocation
- Scheduling
- Netlisting (scheduled)

- Register Allocation
- Binding
- Data Path and State Machine Extraction
- Netlisting (final)
Resource Allocation

- Deciding how many and which kinds of resources will be used in a given implementation
- This has a major impact on final design
  - Number of operation units (multiple adders?) set the maximum parallelism that the architecture can provide
  - Reuse of overloaded operators (e.g. an adder/subtractor unit) provides smallest designs
  - Choice of buses or muxes provides parallelism vs. size
  - Choice of registers, multi-ported register files or RAM also limits parallelism in data movement
Mapping of Operators to Components

Operators | Possible mappings | Components
---|---|---
ADD | | add_rpl (ripple)
| | add_csa (carry save)
| | add16 (tech spec.)
| Width<=16 | | addsub_rpl (ripple)
| | sub_rpl (ripple)
| | sub_csa (carry save)
SUB | | multiply
MUL
Scheduling

- As soon as possible (ASAP)
- As late as possible (ALAP)
- List scheduling
ASAP Schedule (unconstrained)

Y = ((a*b)+c)+(d*e)-(f+g)

The start time for each operation is the least one allowed by the dependencies.

Clock cycle 1

ECE 425  Intro. VLSI System Design
ASAP Algorithm

\[
\textit{ASAP} \ (G(V,E)) \ \{ \\
\text{Schedule all the nodes driven only by PIs to cycle 1,} \\
\text{for all such } v_i \text{ nodes, } t_i \text{ (starting time)} = 1; \\
\text{Repeat} \ \{ \\
\text{Select a vertex } v_i \text{ whose predecessors are all scheduled;} \\
\text{Schedule } v_i \text{ by setting } t_i = \text{MAX}(t_j + d_j); \\
\quad (v_j, v_i) \in E \\
\} \\
\text{Until all the nodes are scheduled;} \\
\text{Return the schedule in a vector;}
\}
\]
The end time of each operation is the latest one allowed by the dependencies and the latency constraint.
Mobility (or Slack)

Y = ((a*b)+c)+(d*e)-(f+g)

Mobility is the difference of the start times computed by the ALAP and ASAP.

Clock cycle

Clock cycle 1

1

2

3

4

ECE 425

Intro. VLSI System Design
List Scheduling (1)

- Priority based on mobility
  (other metrics possible)
- Resource constraints: one adder, one multiplier
- Schedule ready nodes

Prioritized Ready List

<table>
<thead>
<tr>
<th>Operation</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1(mul)</td>
<td>0</td>
</tr>
<tr>
<td>op2(mul)</td>
<td>1</td>
</tr>
<tr>
<td>op3(add)</td>
<td>2</td>
</tr>
</tbody>
</table>

Clock cycle 1

2 3 4
List Scheduling (2)

Schedule op4 and op2

Prioritized Ready List

<table>
<thead>
<tr>
<th>op2(mul)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>op4(add)</td>
<td>0</td>
</tr>
</tbody>
</table>

Clock cycle 1  2  3  4

ECE 425  Intro. VLSI System Design
List Scheduling (3)

Prioritized Ready List

op5(add)0

Clock cycle 1 2 3 4

ECE 425 Intro. VLSI System Design
List Scheduling (4)

Clock cycle 1 2 3 4

Prioritized Ready List

op6(add)0

Formulation can be extended to handle multi-cycle operations
List Scheduling Algorithm

\[ \text{LIST}_L(G(V, E), a) \} \{ \]
\[ l = 1; \]
\[ \text{repeat} \} \{ \]
\[ \text{for each resource type } k = 1,2,...,n_{\text{res}} \} \{ \]
\[ \quad \text{Determine candidate operations } U_{l,k}; \]
\[ \quad \text{Determine unfinished operations } T_{l,k}; \]
\[ \quad \text{Select } S_k \subseteq U_{l,k} \text{ vertices, such that } |S_k| + |T_{l,k}| \leq a_k; \]
\[ \quad \text{Schedule the } S_k \text{ operations at step } l \text{ by setting } \]
\[ \quad t_i = l \text{ for all } i : v_i \in S_k; \]
\[ \} \]
\[ l = l + 1; \]
\[ \} \]
\[ \text{until (all nodes are scheduled);} \]
\[ \text{return } (t) \]
\}
Example

An example CDFG: schedule it using list scheduling
Register Allocation

\[ y = a + b + c + d \]

Clock cycle

1  2  3

Registers Implied
Lifetime Analysis

Registers 1 & 2 can be shared

R1
R2
R3

Clock cycle
1
2
3

(R3 needed for output latch)
y = (a + b + c) \times (d + e)
Binding 1
Results

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op1</td>
<td>Add1</td>
</tr>
<tr>
<td>Op2</td>
<td>Add2</td>
</tr>
<tr>
<td>Op3</td>
<td>Add1</td>
</tr>
</tbody>
</table>
Binding 2
Results

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op1</td>
<td>Add1</td>
</tr>
<tr>
<td>Op2</td>
<td>Add2</td>
</tr>
<tr>
<td>Op3</td>
<td>Add2</td>
</tr>
</tbody>
</table>
Rolled vs. Unrolled Loops

- Unrolled loops
  - Perform everything in parallel
  - Not as much chance for scheduler optimization
  - Generally lead to faster designs

- Rolled loops
  - Schedule faster
  - May improve resource sharing
  - Generally extend latency
 Rolled For Loop

```c
void vectoradd(float a[1000], float b[1000],
               float c[1000])
{
    int i=0;
    for(i=0; i<1000; i++)
        c[i] = a[i] + b[i];
}
```
Unrolling a For Loop

- **void** vectoradd(float a[1000], float b[1000], float c[1000])

  
  ```
  int i=0;
  for(i=0; i<1000; i++)
    c[i] = a[i] + b[i];
  ```

If the loop is unrolled, 4 adders are required. If not, then one adder is needed.
Pipelining and Multi-Cycle

- Given a synchronous system, clock must run at the speed of slowest functional unit and longest wire delay
- Complex operations (like multiply) can slow down whole design
  - Need to Pipeline or Multi-Cycle
- If you have many slow paths – rethink the architecture
Pipelining: split up logic and use faster clocks

- Combinational logic
- Latches

Diagram:

- Top:
  - Combinational
  - Latch

- Bottom:
  - Combinational
  - Latch
  - Combinational
  - Latch
  - Combinational
  - Latch

ECE 425 Intro. VLSI System Design
Multi-Cycle: skip clock edges and use faster clocks
Summary

- Overview of the main operations of HLS
  - Scheduling
  - Allocation
  - Binding
- Some popular high-level HLS optimization options

- Next lecture
  - HLS (2) with more depth