Lecture 20: High-level Synthesis (1)

Slides courtesy of 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
- Reading: Synthesis and optimization of digital circuits, G. De Micheli, 1994, Section 5
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
  - AND

- 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

Diagram:
- RTL Design: Simulate RTL Code, Perform RTL Synthesis
- Expensive Iteration: Verify / Analyze Gate-level Design
HLS Output

Data Path

Control (Finite State Machine)
Behavior in – RTL out

Figure 2-5: Architectural alternatives from a behavioral description
**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 control and data-flow graph (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             Width\leq 16           add_rpl (ripple)

SUB             add_csa (carry save)

MUL             multiply

add16 (tech spec.)

addsub_rpl (ripple)

sub_rpl (ripple)

sub_csa (carry save)
Scheduling

- As soon as possible (ASAP)
- As late as possible (ALAP)
- List scheduling
Y = ((a*b)+c)+(d*e)-(f+g)

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

\[ \text{ASAP} \left( G(V, E) \right) \{ \]
\[ \quad \text{Schedule all the nodes driven only by PIs to cycle 1,} \]
\[ \quad \text{for all such } v_i \text{ nodes, } t_i \text{ (starting time)} = 1; \]
\[ \text{Repeat} \{ \]
\[ \quad \text{Select a vertex } v_i \text{ whose predecessors are all scheduled;} \]
\[ \quad \text{Schedule } v_i \text{ by setting } t_i = \text{MAX}(t_j + d_j); \]
\[ \quad \quad (v_j, v_i) \in E \]
\[ \} \]
\[ \text{Until all the nodes are scheduled;} \]
\[ \text{Return the schedule in a vector;} \]
\[ \} \]
ALAP Schedule

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

The end time of each operation is the latest one allowed by the dependencies and the latency constraint.
Mobility (or Slack)

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

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

Clock cycle 1

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

Clock cycle 1

- Schedule op1 and op3
List Scheduling (2)

Clock cycle 1

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

Schedule op4 and op2
List Scheduling (3)

Clock cycle 1

Prioritized Ready List

op5(add) 0
List Scheduling (4)

Clock cycle 1 2 3 4

Prioritized Ready List

op6(add) 0

Formulation can be extended to handle multi-cycle operations

Intro. VLSI System Design
LIST Scheduling Algorithm

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

An example CDFG: schedule it using list scheduling

[Diagram of a CDFG with nodes labeled 1 to 11 and operations indicated by arrows and symbols.]
Register Allocation

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

Clock cycle 1
Clock cycle 2
Clock cycle 3

Registers
Implied

\[ + \]
\[ + \]
\[ + \]
\[ + \]

ECE 425
Intro. VLSI System Design
Lifetime Analysis

Registers 1 & 2 can be shared

Clock cycle

1

2

3

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

Clock cycle

1  2  3

a  b  op1

+  op2

d  e

+  op3

c

*  y
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
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

```c
void vectoradd(float a[1000], float b[1000],
    float c[1000])
{
    int i=0;
    for(i=0; i<1000; i++)
        #pragma HLS UNROLL 4
        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

Combinationlal → latch → combinationlal → latch → combinationlal → latch → combinationlal → latch

Intro. VLSI System Design

ECE 425
Multi-Cycle: skip clock edges and use faster clocks

combinational

enable

latch
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