Previous Lecture
Guest Presentation

- The level of abstraction as well as the implementation framework depends on the metrics of interest.
- For NN with quantized multiplication operations, the accuracy of the NN can be assessed by modeling the operations in really high-level language such as Python leveraging the existing frameworks.
SystemC Design Methodology

Benefits:
- Allows gradual refinement or behavior synthesis
- System model and synthesis models are written in a single language.
- Test benches can be reused at different abstraction levels.
SystemC Design Flow

1. Write source files and test benches for the system.
2. Compile codes and link with SystemC class library.
3. Execute the compiled binary and check simulation results.
4. Iterate to step 1 if performance not satisfied
Module, Ports, Processes and Signals
Example – Half-Adder in SystemC and VHDL

//File: half_adder.h
#include “systemc.h”
SC_MODULE(half_adder) {
    sc_in<bool>a, b;
    sc_out<bool>sum, carry;
    void proc_half_adder();
    SC_CTOR(half_adder) {
        SC_METHOD (proc_half_adder);
        sensitive << a << b;
    }
}

//File: half_adder.cpp
#include “half_adder.h”
void half_adder::proc_half_adder() {
    sum = a ^ b;
    carry = a & b;
}

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity half_adder is
    port ( 
        a : in std_logic;
        b : in std_logic;
        
        sum : out std_logic;
        carry : out std_logic
    );
end half_adder;

architecture rtl of half_adder is
begin
    process (a, b)
    begin
        sum <= a xor b;
        carry <= a and b;
    end process;
end rtl;
Network-on-Chip

- A network on a chip or network-on-chip (NoC) is a network-based communications subsystem on an integrated circuit ("microchip"), most typically between modules in a system on a chip (SoC)

[Diagram of a network-on-chip (NoC) network]

[Link to Wikipedia article on Network on a Chip]

https://en.wikipedia.org/wiki/Network_on_a_chip
Types of NoC

- Circuit-switched (CSw) vs. Packet-switched
  - Better predictability in real-time embedded systems [1,2]
- TDM vs. SDM
  - Lower data latency, less complex switch and cheaper network interfaces/switches [3]

Agenda

1. Guest Presentation
2. Behavioral Synthesis
3. Behavioral Synthesis with High-level Language
Agenda

1. Guest Presentation
2. Behavioral Synthesis
3. Behavioral Synthesis with High-level Language
Agenda

1. Guest Presentation
2. Behavioral Synthesis
3. Behavioral Synthesis with High-level Language
Behavioral Synthesis VS RTL Synthesis

**Behavioral Synthesis**

- Behavioral code
- Behavioral synthesis
- Logic synthesis
- Gate level netlist

- Write optional RTL code

**RTL Synthesis**

- RTL code
- Logic synthesis
- Gate level netlist
Behavioral Synthesis Problem Definition

- We assume that a circuit is specified by
  - A sequencing graph
  - A set of functional resources, fully characterized in terms of area and execution delay
  - A set of timing and area constraints

- Behavioral synthesis consists of two stages
  - Placing the operations in time (scheduling) and in space (binding)
    - This step is equivalent to annotating the sequencing graph with additional information about the schedule and resource binding
  - Determining the detailed interconnections of the data path and the logic of the control unit
Data-Flow Graph

- Data-flow graph: Operations + data dependency

\[
\begin{align*}
    xl &= x + dx; \\
    ul &= u - (3*x*u*dx) - (3*y*dx); \\
    yl &= y + (u*dx); \\
    c &= xl < a;
\end{align*}
\]
Question 1

- Consider the data-flow graph in the previous slide. Apply the following optimization techniques:
  - Operation-strength reduction
  - Common sub-expression elimination
- Draw the optimized data-flow graph
Data Flow Transformation

- **Common subexpression elimination**
  
  
  ```
  a = x + y;     \quad a = x + y;
  b = a + 1;     \quad \rightarrow \quad b = a + 1;
  c = x + y;     \quad c = a;
  ```

- **Operator strength reduction**
  
  ```
  a = x^2;       \quad a = x \times x;
  b = 3 \times x; \quad \rightarrow \quad t = x \ll 1;
  b = x + t;     \quad b = x + t;
  ```
Solution to Question 1
Operator Strength Reduction

- Use the node numbers of data-flow graph to assign temporary variables with the same numbers to the operations’ outputs

After operator strength reduction

\[
\begin{align*}
  t1 &= 3 \times x \\
  t2 &= u \times dx \\
  t6 &= 3 \times y \\
  t8 &= u \times dx \\
  x1 &= x \times dx \\
  t3 &= t1 \times t2 \\
  t7 &= t6 \times dx \\
  y1 &= y + t8 \\
  c &= x1 < a \\
  t4 &= u - t3 \\
  u1 &= t4 - t7 \\
  t5 &= x << 1 \\
  t1 &= t5 + x \\
  t2 &= u \times dx \\
  t9 &= y << 1 \\
  t6 &= t9 + y \\
  t8 &= u \times dx \\
  x1 &= x \times dx \\
  t3 &= t1 \times t2 \\
  t7 &= t6 \times dx \\
  y1 &= y + t8 \\
  c &= x1 < a \\
  t4 &= u - t3 \\
  u1 &= t4 - t7 
\end{align*}
\]
Solution to Question 1
Common Sub-expression Elimination

\[
\begin{align*}
    t5 &= x \ll 1 \\
    t1 &= t5 + x \\
    t2 &= u \times dx \\
    t9 &= y \ll 1 \\
    t6 &= t9 + y \\
    t8 &= u \times dx \\
    x1 &= x \times dx \\
    t3 &= t1 \times t2 \\
    t7 &= t6 \times dx \\
    y1 &= y + t8 \\
    c &= x1 < a \\
    t4 &= u - t3 \\
    u1 &= t4 - t7
\end{align*}
\]

After common sub-expression elimination

\[
\begin{align*}
    t12 &= x \ll 1 \\
    t1 &= t12 + x \\
    t2 &= u \times dx \\
    t13 &= y \ll 1 \\
    t6 &= t13 + y \\
    x1 &= x \times dx \\
    t3 &= t1 \times t2 \\
    t7 &= t6 \times dx \\
    y1 &= y + t2 \\
    c &= x1 < a \\
    t4 &= u - t3 \\
    u1 &= t4 - t7
\end{align*}
\]
Solution to Question 1
The New Data-flow Graph

\[ t_{12} = x \ll 1 \]
\[ t_1 = t_{12} + x \]
\[ t_2 = u \cdot dx \]
\[ t_{13} = y \ll 1 \]
\[ t_6 = t_{13} + y \]
\[ x_1 = x \cdot dx \]
\[ t_3 = t_1 \cdot t_2 \]
\[ t_7 = t_6 \cdot dx \]
\[ y_1 = y + t_2 \]
\[ c = x_1 < a \]
\[ t_4 = u - t_3 \]
\[ u_1 = t_4 - t_7 \]
Question 2
Scheduling without Resource Constraints

- Assume all operations have unit execution delay and there is no resource constraint, determine the schedule of the sequencing graph.

\[ x_l = x + dx; \]
\[ u_l = u - (3*x*u*dx) - (3*y*dx); \]
\[ y_l = y + (u*dx); \]
\[ c = x_l < a; \]
Answer to Question 2
Determining the Start Time for Each Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1, v2, v6, v8, v10</td>
<td>1</td>
</tr>
<tr>
<td>v3, v7, v9, v11</td>
<td>2</td>
</tr>
<tr>
<td>v4</td>
<td>3</td>
</tr>
<tr>
<td>v5</td>
<td>4</td>
</tr>
</tbody>
</table>
Answer to Question 2
Scheduled Sequencing Graph

Time 1

Time 2

Time 3

Time 4

© Akash Kumar
Question 3
Scheduling with Resource Constraints

- Consider the previous sequencing graph again
  - Assume all operations have unit execution delay.
  - Determine the schedule of the sequencing graph. Assume that in any one clock cycle, only one resource per type can be used.
    - The available resource types are \{multiplier, ALU\}, where ALU can perform addition, subtraction and comparison
  - Draw the scheduled sequencing graph.
Answer to Question 3
Determining the Start Time for Each Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier ALU</td>
<td></td>
</tr>
<tr>
<td>V1 V10</td>
<td>1</td>
</tr>
<tr>
<td>V2 V11</td>
<td>2</td>
</tr>
<tr>
<td>V3 -</td>
<td>3</td>
</tr>
<tr>
<td>V6 V4</td>
<td>4</td>
</tr>
<tr>
<td>V7 -</td>
<td>5</td>
</tr>
<tr>
<td>V8 V5</td>
<td>6</td>
</tr>
<tr>
<td>- V9</td>
<td>7</td>
</tr>
</tbody>
</table>
Answer to Question 3
Scheduled Sequencing Graph
Consider the previous scheduled sequencing graph in question 2.

- There are 11 operations.
- Assume that the available resource types are \{multiplier, ALU\}, where ALU can perform addition, subtraction and comparison.
- In addition, assume that there are 6 multipliers and 6 ALUs are available.

Without using any resource sharing, decide the number and type of resources that we need for the resource binding solution of this problem.
Answer to Question 4

Answer: Need 6 instances of the multiplier type (type=1) and 5 instances of the ALU type (type=2).

<table>
<thead>
<tr>
<th>Binding</th>
<th>Resource (type, #instance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(v1)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>B(v2)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>B(v3)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>B(v4)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>B(v5)</td>
<td>(2,2)</td>
</tr>
<tr>
<td>B(v6)</td>
<td>(1,4)</td>
</tr>
<tr>
<td>B(v7)</td>
<td>(1,5)</td>
</tr>
<tr>
<td>B(v8)</td>
<td>(1,6)</td>
</tr>
<tr>
<td>B(v9)</td>
<td>(2,3)</td>
</tr>
<tr>
<td>B(v10)</td>
<td>(2,4)</td>
</tr>
<tr>
<td>B(v11)</td>
<td>(2,5)</td>
</tr>
</tbody>
</table>
Consider the previous binding answer to question 4, which is not efficient. Determine a more optimized binding which uses only four multipliers and two ALUs.
Answer to Question 5

Answer: Need 4 instances of the multiplier type (type=1) and 2 instances of the ALU type (type=2). 

<table>
<thead>
<tr>
<th>Binding</th>
<th>Resource (type, #instance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(v1)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>B(v2)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>B(v3)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>B(v4)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>B(v5)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>B(v6)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>B(v7)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>B(v8)</td>
<td>(1,4)</td>
</tr>
<tr>
<td>B(v9)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>B(v10)</td>
<td>(2,2)</td>
</tr>
<tr>
<td>B(v11)</td>
<td>(2,2)</td>
</tr>
</tbody>
</table>
Agenda

1. Guest Presentation
2. Behavioral Synthesis
3. Behavioral Synthesis with High-level Language
The compiler synthesizes the hardware from a behavioral description by:

- Timing all operations, based on a technology library
- Scheduling operations, I/O, and memory accesses into clock cycles.
- Allocating hardware by assigning variables and signals to registers and assigning operations to synthetic components.
- Creating a finite state machine (FSM) and memory interface control logic.
- Pipelining loops for higher throughput, which typically increases the size of the hardware.
- Inferring memory for arrays
- Chaining and multicycling operations
Generated Structure

- **External input**
- **FSM**
- **Status**
- **Control**
- **Data Path**
  - **MULT**
  - **ADD**
- **Memory**

Diagram illustrating the flow of data and control signals through the structure.
During timing estimation process, the compiler determines the delay through each component. It uses

- Vendor libraries (CLB, DSP, or ASIC gates, etc.)
- Wire load models
- Operating conditions

The estimated timings are used during scheduling and allocation to determine an appropriate architecture.
Scheduling

Behavioral code

```c
wait_until(start.delayed() == true);
A = port1.read() * port2.read();
B = port3.read() * port4.read();
C = A + B;
If (C<0) {...}
```

Scheduled Operations

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>read inputs</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
Scheduling Objectives

- Satisfy the data and control dependencies between operations
- Ensure that the scheduling constraints of latency, throughput and clock period are met
- Facilitate maximum resource sharing by distributing operations over the allowed number of cycles.
- Allow for maximum register sharing by producing and consuming variables intelligently.
Allocating Hardware

### Scheduled Operations

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>read inputs</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

### Hardware Resources

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Adder</th>
<th>Comparator</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>+</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Allocation
Creating FSM and Data Path

Scheduled Operations

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>read inputs</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Hardware Resources

<table>
<thead>
<tr>
<th></th>
<th>Multiplier</th>
<th>Adder</th>
<th>Comparator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>+</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Before scheduling and binding

After scheduling and binding

© Akash Kumar
Homework

□ What kinds of algorithm that are used in scheduling/binding?

□ Use SDSoC and Vivado HLS for different scheduling/binding with different number of resources
Thank You!