RTL and Behavioral Synthesis

Srinivas Devadas
MIT
Kurt Keutzer
Richard Newton
UC Berkeley

RTL Synthesis Flow

244 Lecture Notes 11-1
**RTL level**

An RTL description is always implicitly structural -
- the registers and their interconnectivity are defined
- Thus the clock-to-clock behavior is defined
- Only the control logic for the transfers is synthesized.

This approach can be enhanced:
- Register inferencing
- Automating resource allocation

**RTL level**

\[
\begin{align*}
\text{RTL implies} & \\
\begin{align*}
 b & \rightarrow + \\
 c & \rightarrow + \\
 d & \rightarrow + \\
 e & \rightarrow +
\end{align*}
\end{align*}
\]

**Behavioral implies**

\[
\begin{align*}
a &= b + c; \\
d &= a + 1; \\
e &= d \times 2; \\
e &= 2 \times (b + c + 1)
\end{align*}
\]

---

**Behavioral Synthesis Flow**

[Diagram showing the flow from Behavioral Synthesis to HDL, passing through RTL Synthesis, netlist, logic optimization, netlist, physical design, and layout.]
A behavioral description is always functional
- Temporal relationships are only expressed as precedences
- An entire micro-architecture is synthesized from the behavioral description

The two key elements of behavioral transformation:
- Automating resource allocation (could be RTL also)
- Scheduling

RTL implies
\[ a = b + c; \]
\[ d = a + 1; \]
\[ e = d \times 2; \]

Behavioral implies
\[ e = 2 \times (b + c + 1) \]

RTL Synthesis

Two Steps:
1. Translation from RTL description into gates
2. Optimization of logic

Ideally, all of the intelligence should be in the optimization step
Unfortunately, starting point for optimization affects results
⇒ Need to write a “good” RTL description

244 Lecture Notes 11-1
entity VHDL is
   port C (A, B, C : in BIT ; Z : out BIT ; ) ;
end VHDL ;

architecture VHDL_1 of VHDL is
begin
   Z <= (A and B) or C ;
end VHDL_1 ;

entity VHDL is
   port C (A, B, C : in BIT ; Z : out BIT ; ) ;
end VHDL ;

architecture VHDL_1 of VHDL is
begin
   Z <= (A and B) or C ;
end VHDL_1 ;

Wait Statements

process begin
   wait until CLOCK' event and CLOCK = '1' ;
   if (ENABLE = '1') then
      TOGGLE = not TOGGLE ;
   end if ;
end process ;

wait infers a flip-flop
Writing RTL Descriptions

Current scenario: Designer iterates over RTL descriptions using simulation and synthesis tools to verify and evaluate design.

PROBLEM: Equivalent RTL descriptions may result in dramatically different logic circuits in terms of area/performance EVEN AFTER OPTIMIZATION.

Equivalent RTL Descriptions

if (SUB)
    result = a - b;
else
    result = a + b;

if (SUB)
    b = -b;
result = a + b;

push multiplexors to inputs
Practical Solution

1. Increase power of optimization.
2. Define “good” input and restrict oneself to writing “good” input descriptions.
3. Manipulate input descriptions to make them “good”.

Rules for writing descriptions based on knowledge that a library of descriptions are good starting points for synthesis.

Behavioral Synthesis Features

Scheduling of operations
- Operation Chaining and Multi-cycling
- Different Scheduling Modes, and constraints

Memory Inferencing
- Array’s can be mapped directly to RAM
- Automatic control/data/address generation
- RAM tradeoffs (1 port vs. 2 port) are easy

Allocation of resources
- Building your own design components

Using Pipelined Components
- Multipliers, Adders, RAMs, others

Pipelining Loop with various Throughput & Latency

Automatic generation of FSM Controller and integration of Datapath
Level of Acceptance of Synthesis Techniques

Source: A. DeGeus

Tradeoff Example - Echo Canceler

Complex Echo Canceler

244 Lecture Notes 11-1
Echo Canceler - Tap

Total: 6 Taps

Tap - Complex Multiplication

Total: 6 complex multiplications

244 Lecture Notes 11-1
Echo Canceler Complexity

Clock cycle : 30ns
Complexity : around 10K to 20K gates
Modules : 16x2 multiplier : 22ns 713 gates
17x17 add/sub : 20ns 193 gates
Source : ~2000 VHDL excl. comments

operations handled by BC operations handled by DC

addition : 30 constant : 11
subtraction : 20 logic oper. : 66
multiplication : 24 word expand : 100
compare oper. : 2 truncation : 124
data select : 29 shift oper. : 62
delay elements : 24

Total : 492

Echo Canceler Source Code
Scheduling Tradeoffs I

Design Tradeoff with Scheduling

Design #1
- Clock: 50ns
- 2 fast multipliers
- 3 cycles

Design #2
- Clock: 50ns
- 1 fast multiplier
- 4 cycles

Design #3
- Clock: 50ns
- 2 slow multipliers
- 4 cycles

Design Tradeoff with Scheduling

Scheduling Tradeoffs II

Design #4
- Clock: 100ns
- 2 slow multipliers
- 2 cycles

Design #5
- Clock: 50ns
- 1 pipeline multiplier
- 5 cycles

Tradeoff:
Clock speed vs Latency vs Resources
Scheduling Tradeoff Results

Tradeoff: Resources, Area & Number of cycles

<table>
<thead>
<tr>
<th># multiplier</th>
<th># adder</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Scheduling Modes

Cycle-Fixed Mode
- In this mode, the exact cycle times of read and write operations on signals are preserved. Operations on variables are free to move around.

Superstate-Fixed Mode
- In this mode, the relative times of read and write operations are preserved, but superstates (delimited by wait statements) may be stretched. Operations on variables are free to move around.

Free-floating Mode
- In this mode, computations, read and write operations may flow freely past wait statements

In all cases, user can provide constraints to control the scheduling process.
Approaches to Scheduling

(1) Exhaustive Approaches
   e.g. EXPL (Barbacci, 1962): exhaustive search, Hafer & Parker: Branch & bound

(2) As-Soon-As-Possible (ASAP)
   Used by many pioneering systems (e.g. early CMUDA, MIMOLA, and Flamel)

As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) Scheduling
Approaches to Scheduling

(3) List Scheduling
- Keep list of operations available to be scheduled (i.e., whose predecessors have already been scheduled) and order them for scheduling using some priority function
  - BUD - length of path from operation to end of enclosing block
  - Elf, ISYN - "urgency": length of shortest path from operation to nearest local constraint
- O(C N log N) time complexity

(4) Force-Directed Approaches
- "Force" between operation and control-step proportional to the number of operations of the same type that could go in that control step
- Scheduling to minimize force tends to minimize the number of resources needed (e.g. HAL)
- O(C N^2) time complexity

Minimizing Resources Using Force-Directed Techniques

244 Lecture Notes 11-1
**Basic Force-Directed Algorithm**

repeat {
(1) evaluate ASAP and ALAP limits for each operation;
(2) create or update distribution graphs;
(3) calculate the self-force for every unscheduled operation and feasible control step;
(4) add predecessor and successor forces to self-force;
(5) schedule the operation with lowest force;
} until (all operations are scheduled;)

---

**5th-Order Wave-Digital Filter**

26 additions and 8 multiplies, as specified

---

244 Lecture Notes 11-1
5th Order Wave-Digital Filter Dataflow Graph

Results from List Scheduling

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>+ +</td>
</tr>
<tr>
<td>13</td>
<td>+ +</td>
</tr>
<tr>
<td>14</td>
<td>× × +</td>
</tr>
<tr>
<td>15</td>
<td>+ + +</td>
</tr>
<tr>
<td>16</td>
<td>+ + +</td>
</tr>
<tr>
<td>17</td>
<td>+ + +</td>
</tr>
<tr>
<td>18</td>
<td>+ + +</td>
</tr>
<tr>
<td>19</td>
<td>+ + +</td>
</tr>
</tbody>
</table>

244 Lecture Notes 11-1
ASAP / ALAP Schedules for Case 1

ASAP / ALAP Schedules for Case 2

244 Lecture Notes 11-1
## Representative Results for 5th Order WDF

<table>
<thead>
<tr>
<th>Synthesis System</th>
<th>Adders</th>
<th>Multipls</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDS(HAL)</td>
<td>2</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>APARTY(SAW), CMU</td>
<td>2</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>CADDY</td>
<td>2</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>FDLS(HAL)</td>
<td>2</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>&lt;ESC&gt;</td>
<td>2</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>HAL, CADDY, Nische</td>
<td>2</td>
<td>1p</td>
<td>19</td>
</tr>
<tr>
<td>SPAID, &lt;ESC&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Memory Types, Embedded

Memory Types:

- **Synchronous**: Inputs & outputs occur at clock edge
- **Asynchronous**: Unclocked signals
- **Single Port**: Either read or write to one address location
- **Dual/Multi Port**: Two fully independent read/write ports

![Diagram of Memory Types](image.png)
Memory Inferencing

--- RAM Declaration Section ---

```plaintext
type word is integer range 0 to 255;
variable a : array N downto 1 of word;
variable b : array M downto N + 1 of word;
constant MY_RAM : resource := 0;
attribute variables of MY_RAM is "a b";
attribute map_to_module of MY_RAM is "DW03_ram1_s_d";

--- RAM Read and Write Example ---
a(i) := x * y;  -- write
SUM := a(i) + b(N+i);  -- read
```

--- RAM Read and Write Example ---

```plaintext
/a[i] = x * y;
SUM = a[i] + b[N+i];
```

Support for Single Port, Dual Port, Etc.
Several arrays can be mapped to the same RAM

Memory Tradeoff Example

```plaintext
/***** RAM Declaration Section ****/
reg [7:0] a[N:0], b[M:N+1];
/* synopsys resource MY_RAM: variables = "a b",
map_to_module = "DW03_ram1_s_d"; */
/***** RAM Read and Write Example ****/
a[i] = x * y;
SUM = a[i] + b[N+i];
```

Support for Single Port, Dual Port, Etc.
Several arrays can be mapped to the same RAM
### Memory Tradeoff Results

#### Tradeoff: RAM read/write ports, Area & Latency

<table>
<thead>
<tr>
<th># port in RAM</th>
<th>Latency 1c</th>
<th>Latency 1c</th>
<th>Latency 1c</th>
<th>Latency 1c</th>
</tr>
</thead>
<tbody>
<tr>
<td>read 1c write 2c</td>
<td>132</td>
<td>131</td>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>read 1c write 1c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read 1c write 1c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Loop Pipelining

**For high-throughput designs**

Without Loop Pipelining:

![Diagram](Without Loop Pipelining.png)

With Loop Pipelining...

![Diagram](With Loop Pipelining.png)

*Pipelined to start every two cycles, Operations overlapped.*

*Latency remains the same. Throughput increases and input/output streams become more regular.*
Loop Pipelining Tradeoffs

Without Loop pipelining:
- Latency 4 cycles
- Throughput 4 cycles
- 1 fast multiplier

Latency 4 cycles
Throughput 1 cycles
2 fast multipliers

Latency 4 cycles
Throughput 2 cycles
1 fast multiplier

Loop Pipelining Tradeoffs Results

Tradeoff: Resources, Initiation, Throughput & Area

<table>
<thead>
<tr>
<th># cycles</th>
<th># multiplier</th>
<th># adder</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 cycles</td>
<td></td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>8 cycles</td>
<td></td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>16 cycles</td>
<td>7</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>24 cycles</td>
<td>6</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>32 cycles</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Area

4 cycles
8 cycles
16 cycles
24 cycles
32 cycles

# cycles
throughput
latency

Total Area
# adder
# multiplier

244 Lecture Notes 11-1
Pipelined Components

- If faster multiplier required, use pipelined multipliers from design library
  - Two-stage
  - Three-stage

- Benefit of using design library
  - Behavioral synthesis automatically generates control logic
  - Better area efficiency
  - Resource sharing

Behavioral Synthesis

- Write Source Code
- Check Timing (DC Output)
- If Multiplier on Critical Path
- Change $y = a \cdot b$ to $y = \text{mult}_2s(a, b, \text{clk})$

Pipelined Tradeoff Results

Tradeoff: Resources, Area & Number of cycles

<table>
<thead>
<tr>
<th>Area</th>
<th># cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>48</td>
</tr>
</tbody>
</table>

Legend:
- Total Area
- # adder
- # pipeline multiplier

244 Lecture Notes 11-1
Creating customized design library parts

Incremental Step for defining design library components for Behavioral Synthesis

Identify the common expression

Implemented as design library component

function [15:0] sum_of_product;
    // map_to_operator dw_sop
    // return_port_name sz
    input [7:0] a, b, c, d;
    begin
        sum_of_product = a * b + c * d;
    end
Creating the implementation

Function Definition

```vhdl
function sum_of_product (a, ...: input UNSIGNED) return UNSIGNED is
    -- pragma map_to_operator dw_sop
    -- pragma return_port_name sz
    begin
        .......... end;
```

Synthetic Operator

```
dw_sop
```

Synthetic Module

```
dw_sop_mod .sl file
```

Implementations

```
Small Fast
```

Other Features included

Bit-level timing of design library components
  – Does the operation fit within a clock?
  – Do multiple operations fit within a clock?

When should the operations be performed
  – 3rd, 4th cycle?
  – How many operations in each clock cycle?

Register assignment
  – Where should temporary values be stored?
  – How long are the inputs, temporary values needed?

Is the mux cost more expensive than sharing a resource?
Summary of all Tradeoff Results

18 different design tradeoffs

in 3 days

Behavioral Design Benefits

Benefits:
- more abstraction
  - specifying functionality instead of implementation
- faster simulation
- system level design
  - latency, #cycles, #resources, clock period
  - throughput and area correlation
- better quality of result
  - multi-cycle optimization
  - optimization not limited by register boundaries
- automatic generation of FSM

244 Lecture Notes 11-1
Status of Synthesis Approaches

RTL synthesis is mature and synthesis from RTL descriptions is the most popular means of integrated circuit design.

Behavioral synthesis is much less mature
- Initial use in DSP design (constrained problems: such as register allocation given a schedule)
- Growing use in video (lots of arithmetic), networking (complex schedules), and gradually, general ASIC design
- Has a cult of designers that use it but it has not (as some have predicted) grown to overtake RTL synthesis in popularity

SYNTHESIS REFERENCES

Overviews:

Particularly for DSP:
Extras

Behavioral Synthesis

Manipulating and optimizing RTL descriptions while maintaining required functionality but changing FSM behavior

Behavioral transformations

- Parallelize, serialize or pipeline a behavioral description
- Allocate hardware; registers buses, ALUs to obtain a RTL description with a definite structure
Behavioral Specification

Elliptic-filter (In, Out)
{
    i = In
    a = i + t2
    b = a + t13
    g = t33 + t39
    e = g + t26 + b
    d = (m21 * e) + b
    f = (m24 * e) + g
    ...
    ...
    Out = o
}

Hardware Allocation

Treat the given description as a “dataflow” type specification.

Synthesize a datapath by first constructing a schedule of operations, that will require a certain amount of hardware, and then synthesize a controller given the datapath and associated schedule.

Control is viewed totally as a by-product of datapath synthesis.
Example Behavior

\[ V_3 = V_1 + V_2 \]
\[ V_5 = V_3 - V_4 \]
\[ V_7 = V_3 \times V_6 \]
\[ V_8 = V_3 + V_5 \]
\[ V_9 = V_1 + V_7 \]
\[ V_{11} = V_{10} / V_5 \]
\[ V_{13} = V_3 \]
\[ V_{12} = V_1 \]
\[ V_{14} = V_{11} \text{ and } V_8 \]
\[ V_{15} = V_{12} \text{ or } V_9 \]
\[ V_1 = V_{14} \]
\[ V_2 = V_{15} \]

Schedule - I

A schedule:

\[ r_3 = r_1 + r_2 \]
\[ r_7 = r_3 - r_4 \]
\[ r_2 = r_3 \times r_5 \]
\[ r_3 = r_3 + r_7 \]
\[ r_2 = r_1 + r_2 \]
\[ r_7 = r_6 / r_7 \]
\[ r_1 = r_3 \text{ and } r_7 \]
\[ r_2 = r_2 \text{ or } r_8 \]

\[ V_1 \text{ to } V_{15} \text{ merged into } r_1 \text{ to } r_8 \]
**Datapath - I**

ALU performs +, -, *, /, and, or

\[
\begin{align*}
r_3 &= r_1 + r_2 \\
r_7 &= r_3 - r_4 \\
r_2 &= r_3 \times r_5 \\
r_3 &= r_3 + r_7 \\
r_2 &= r_1 + r_2 \\
r_7 &= r_6 / r_7 \\
r_1 &= r_3 \text{ and } r_7 \\
r_2 &= r_2 \text{ or } r_8
\end{align*}
\]

**Controller - I**

Controller:

<table>
<thead>
<tr>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>Opcode</th>
<th>OpCode1</th>
<th>OpCode2</th>
<th>OpCode3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S0</td>
<td>NOP</td>
<td>0000000000000000</td>
<td>1110100100000010</td>
<td>1110000101000010</td>
<td>1110000000000000</td>
<td>1110000000000000</td>
<td>1110000000000000</td>
<td>1110000000000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>S0</td>
<td>S1</td>
<td>NOP</td>
<td>0000000000000000</td>
<td>1110100100000010</td>
<td>1110000101000010</td>
<td>1110000000000000</td>
<td>1110000000000000</td>
<td>1110000000000000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

244 Lecture Notes 11-1
Schedule and Datapath - II

\[ r3 = r1 + r2 \quad \text{r8} = r1 \]
\[ r5 = r3 - r4 \quad \text{r2} = r3 * r6 \]
\[ r3 = r3 + r5 \quad \text{r5} = r5 * r7 \]
\[ r2 = r1 \text{ and } r2 \quad \text{r1} = r3 \text{ or } r5 \]
\[ r2 = r8 + r2 \]

Controller - II

Controller

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S0</td>
<td>NOP</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>S0</td>
<td>NOP</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>S1</td>
<td>ADD</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>S2</td>
<td>SUB</td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>S3</td>
<td>ADD</td>
<td>DIV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>S4</td>
<td>AND</td>
<td>OR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>S5</td>
<td>ADD</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

244 Lecture Notes 11-1
Allocation Problems

Arithmetic unit allocation: Decide the number of arithmetic units, as well as the grouping of arithmetic operators

e.g., ALU1: +, -
ALU2: +, -, incr
ALU3: *, shift

or ALU1: +, -, incr, shift
ALU2: *

Register allocation: Coalesce the variables into a minimum number of registers

Interconnect allocation: Allocate buses and links, to implement all the required data transfers between the registers and ALUs.

Placement Formulation

Call the X dimension (hardware) space and the Y dimension (execution) time

\[
\begin{align*}
Z_1 &= X_1 + Y_1 \\
K_1 &= Z_1 - X_2 \\
L_1 &= Z_2 \text{ or } Z_3 \\
Z_2 &= X_1 \times Y_2 \\
K_2 &= Z_2 \times X_1 \\
Z_3 &= X_2 \\
K_3 &= K_2 + 1
\end{align*}
\]

Minimum number of ALUs:
- Maximum number of occupied space slots. Each ALU performs operations in its slot.

Minimum number of registers:
- Maximum density of variable lifetimes across all the time slots.

Interconnections related to the number of parallel data transfers in each time slot.
Register Allocation

Given a code sequence, coalesce the variables into a minimum number of registers.

\[ V_1 = V_2 + V_3 \]
\[ V_4 = V_2 - V_3 \]
\[ V_5 = V_1 \times V_2 \]
\[ V_6 = V_4 \text{ and } V_3 \]
\[ V_7 = V_5 \text{ or } V_6 \]

Define the lifetime of a variable as the interval between its first assignment and last use.

Merge variables with non-overlapping lifetimes

Lifetimes are open intervals, at bottom end.

Register Allocation - II

Maximum density of variable lifetimes across all timeslots = minimum # registers required

\[ V_1 = V_2 + V_3 \]
\[ V_4 = V_2 - V_3 \]
\[ V_5 = V_1 \times V_2 \]
\[ V_6 = V_4 \text{ and } V_3 \]
\[ V_7 = V_5 \text{ or } V_6 \]

Merge variables into # registers = maximum density

\[ R_1 = R_2 + R_3 \]
\[ R_4 = R_2 - R_3 \]
\[ R_1 = R_1 \times R_2 \]
\[ R_4 = R_4 \text{ and } R_3 \]
\[ R_4 = R_1 \text{ or } R_4 \]

Different schedules have different max. densities.
Languages versus Models

Esterel

Language

Model of Computation

Synchronous
Reactive

FPGA-Based

Implementation Media

UML

C for RTOS

RTOS

Languages versus Models

Esterel

“C, C++”

VHDL

Matlab

UML

Synchronous
Reactive

Discrete Event

“FSMs”

“Von Neumann” Processor

Hard-Wired ASIC

C for RTOS

RTOS
Languages versus Models

Languages versus Models: A Software Analogy

244 Lecture Notes 11-1
What is an Architecture?

- Amdahl, Blaauw, and Brooks, 1964, defined three interfaces:
  - **Computer Architecture:** "The attributes of a computer as seen by a machine language programmer."
  - **Implementation:** "Actual hardware structure, logic design, and datapath organization."
  - **Realization:** "Encompasses the logic technologies, packaging, and interconnection."

What is an Architecture?

- A description of the behavior of a system that is independent of its implementation.
  - Isomorphic to its "interface specification" (Siewiorek, Bell, Newell, 1971)

- For example:
  - Instruction set definition of a computer
  - Z-domain description of a filter
  - Handshaking protocol for a bus

- May guide implementation (contain 'hints' or 'pragmas')
  - e.g., a particular specification may lead naturally to a serial or parallel implementation.
What is an Architecture?

- Example: Instruction set

  Inst_A
  Inst_B
  Inst_C
  ...
  Inst_C may not follow Inst_A

  Architecture

  Not architecture