



CSCE 491 Computer Engr. Design Project

2005/9/1

Week 3 Digital VLSI Design Methods-2 ASM Modeling

© 2004 Dr. James P. Davis

Week 3 - Outline

- We have briefly discussed System-level specification, analysis and architecture design.
 - ✓ Unified Modeling Language (UML) as the means for expressing the specifications and exploring design architecture.
 - ✓ We'll come back to this when we start discussing details of 802.11 analysis model and architecture.
- Now, we focus on Systems-level architecture and Register Transfer Level (RTL) design using the Algorithmic State Machine.
 - ✓ Discussion of basic drivers behind VLSI design
 - ✓ Discussion of the process, methods and tools we'll be using for executing the VLSI-based hardware design activities for the 802.11 WLAN application.
 - ✓ Focus on Nimbus and the use of FSM (finite state machine, CSCE 211) and RTN (register transfer notation, CSCE 212) for control path and data path architecture and design, using extended ASM notation.
 - ✓ Discussion of methods for design evaluation and tradeoff analysis of designed micro-architectures.



© 2002 Dr. James P. Davis Page 2

I. Design Heuristics:

Questions to Ask as You Start Modeling a Digital Circuit or System



© 2002 Dr. James P. Davis Page 3

Five Heuristic “Principles” of Digital Systems Modeling

- We express the inquiry of design as “probing” questions capturing a set of heuristic principles we want to apply in algorithm transformation.
- Each principle concerns a different aspect of the architecture analysis and design problem-solving activities.
- Each is used to create a set of candidate architectures that may differ along one or more of the principle dimensions.
- Exploring the design space involves creating candidate models, verifying their correct behavior, then conducting logic synthesis and layout in target technology platform.



© 2002 Dr. James P. Davis Page 4

The Five “Principles” – Question 1

- Question 1: What control and data path functionality should be included in each partitioned block?
 - ✓ Partitioning can be carried out according to “function”, or by “responsibility”.
 - ✓ “Responsibility-driven design” is used in Object-Oriented Analysis & Design practice.
 - ✓ Partitioning according to clear responsibilities, and clear collaboration between modules, minimizes “coupling” and increases “cohesion” in partitioned modules.
 - ✓ Each partitioned block has a range of control and data path capabilities.
 - ✓ Some blocks are only controllers, sending out control signals.
 - ✓ Some blocks are only datapath, providing single or multistaged (pipelined) computation.
 - ✓ Most blocks of computation are a mixture of both control and datapath.



© 2002 Dr. James P. Davis Page 5

The Five “Principles” – Question 2

- Question 2: What amount of computation can be carried out within a given clock cycle? (i.e., “computing step”).
 - ✓ We assume the design of synchronous sequential systems under the control of one or more “synchronizing” signals.
 - ✓ These signals can be the system clock, or other internally generated signals.
 - ✓ Clocking can be “periodic” or “aperiodic” (to approximate “asynchronous” circuits and “isochronous” coordination in distributed systems).
 - ✓ We will decompose application-specific computation across some number of computing “cycles”, generally synchronized to one or more clocking signals.
- Alternate question: What should length of the clock cycle be in order to accommodate (“service”) our computation?



© 2002 Dr. James P. Davis Page 6

The Five "Principles" – Question 3

- Question 3: For a specific unit of computation, when do we need to have the data available on the computing unit's output?
 - ✓ Units of computation involve some datapath function followed by an assignment operation.
 - ✓ For example: **Out <- ADD (A, B)**
 - ✓ Signals are registered, latched or wired on completion of computation.
 - ✓ Concerns the "cycle delay" or "latency" of a specific computation, and when the value is available for use in downstream computations.
 - ✓ Assigning the result of computation to a wire is available immediately. In general, $t_{prop} \ll t_{clk}$, so we ignore wire delay for now.
 - ✓ Assigning computation to a register incurs a one-cycle delay from when the computation is "scheduled" by the state machine to when the result is available on the registered output for the next computation.



The Five "Principles" – Question 4

- Question 4: For a given stream of computation, what must be computed serially, and what can be computed in parallel?
 - ✓ Algorithmic computation may be *strictly ordered*, *partially ordered*, or *unordered*.
 - ✓ The ordering of computation is based on the data dependencies between each stage or "unit" of computation.
 - ✓ What forms of parallelism? We use basic "patterns".
 - ✓ Pipelining of the datapath.
 - ✓ Coordinated control via "handshaking" or "arbitration" in the control path.
 - ✓ Concurrency in either the control or datapath, or both.
 - ✓ The form of parallelism governs computational "latency" and "throughput".
 - ✓ The "granularity" of computation of each parallel unit, and topology of interconnect between units, affects circuit complexity.



The Five "Principles" – Question 5

- Question 5: For a given stream of computation, what is the nature of "looping" in the computation?
 - ✓ Algorithmic computation may require iteration (FOR loop).
 - ✓ Iteration requires control of the looping process using feedback (using a Counter and/or Comparator circuit).
 - ✓ Feedback may exist in the datapath, where an internal bus may recycle a newly computed value back through a computing element.
 - ✓ In hardware, feedback without sufficient "delay" unit causes oscillation, metastable, bistable or other non-converging behavior. A register or latch is needed to store values prior to feedback through the circuit.
 - ✓ Looping for iteration can be "unrolled" to minimize logic depth, cost of comparator logic.

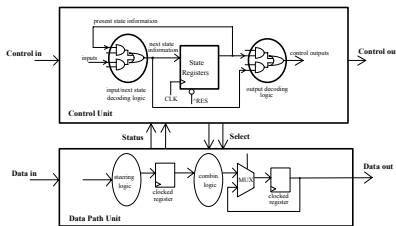


II. ASM Design Notation:

Specifying Signal Assertions and Register Assignments



Functional Partitioning of Control and Datapath



Control Unit	Data Path Unit
<ul style="list-style-type: none"> <input type="checkbox"/> modeled using FSM model <input type="checkbox"/> defines the clock-based sequencing of actions in the data path or external to the block 	<ul style="list-style-type: none"> <input type="checkbox"/> modeled using RTL model <input type="checkbox"/> defines both synchronous and asynchronous transformations of data as it moves through the block

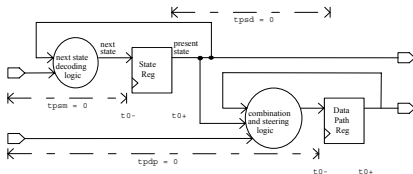


Delay Assumption for Register-Transfer Level

In ASM, we model the behavior of registers using the "limit" assumption. First, at some time t_a corresponding to the active edge of a clock, there is a different value on the input of a register than on its output.

The time between when the register input is "sampled" and when the value appears on the register output cannot be zero. We need to consider the change in "state" of the design on the clock edge, where we are assigning values to the input and wanting to see the results that appear on the output.

We assume the mathematical limit of t_a from both sides of the clock transition, which we refer to as times t_a^- and t_a^+ . The time t_a^- is when the register input is being "sampled", and the time t_a^+ is when the sampled value appears on the register output.

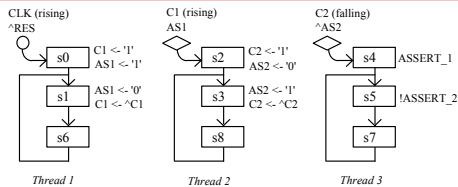


III. ASM Design Notation:

Modeling Concurrent Components in Digital Systems



ASM - Modeling Thread-level Concurrency



Modeling Concurrency:

- Multiple model ASM "threads" having shared buses.
- Independent clocking schemes and enabling events (e.g., ^RES).
- Types of concurrent interaction:

I. Synchronization

- coordinated activities (e.g., handshaking, pipelining).
- implicit references to shared buses.

II. Competition

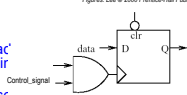
- shared resources (for example, bus arbitration).
- explicit use of other concurrent processes, components, or entities to model the arbitration protocol.



ASM- Control Exerted Across Threads

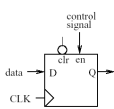
• Gated/User-defined clocking:

- ✓ We use some combination of signals to derive the clock signal for an ASM thread governing its state transitions and clock of its data path registers.
- ✓ Nimbus only lets you specify a single use defined signal on the clock.



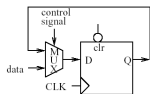
• Gated/User-defined resets:

- ✓ We use some control signal to reset the registers of a control block and its data path, either from the data path or from another controller.

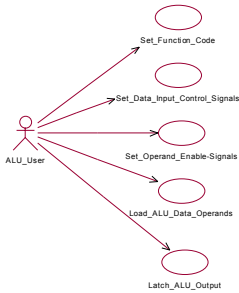


• MUXed data path inputs:

- ✓ We use control signals to "gate" the data into a register or data path stage by "implying" a MUX by the specification of an expression assignment to a particular bus in one or more states.



The ALU Analysis Model – Use Cases



- Abstract ALU:
 - ✓ This circuit takes two inputs and operates on them to produce an output.
 - ✓ The output is determined by the selection of function codes, data input control, and operand enable signals.
 - ✓ There is an implies ordering of tasks for setting up the ALU (not shown in the Use Case diagram): all the control signals must be stable before applying the data inputs; then the device is enabled.
 - ✓ The ALU operates as a combinational logic device; so its output is latched into a register when the operation is complete.



© 2002 Dr. James P. Davis Page 43

The ALU Function Model – Truth Table

F ₀	F ₁	ENA	ENB	INVA	INCB	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	\bar{A}
1	0	1	1	0	0	\bar{B}
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B - A
1	1	0	1	1	1	B - 1
1	1	1	0	1	1	-A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	-1

Functional specification:

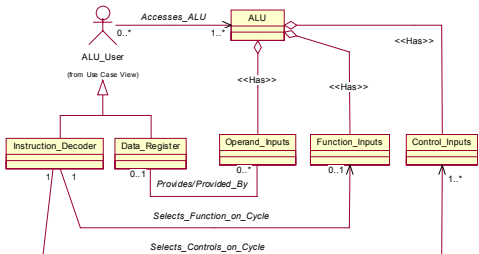
- ✓ This is a technique to represent a purely functional result.
- ✓ You list a column in the table for each input, and a column for each possible output combination.
- ✓ The table is used as a "look-up", given some input combination, to determine the output.
- ✓ The outputs are a function of the inputs.
- ✓ Truth table results are derived based on rules of Boolean Logic, and complex functions can be built up from understanding more primitive ones.

Source: A. Tanenbaum, 4th ed., 1999.



© 2002 Dr. James P. Davis Page 44

The ALU Analysis Model – Class Diagram-1



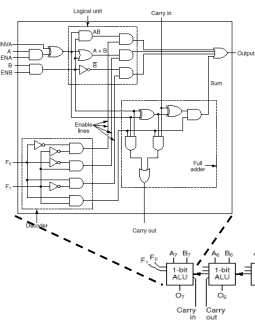
© 2002 Dr. James P. Davis Page 45

Lab – Scaling the ALU Model

- For this part of the workshop, you will take the specification for the single-bit ALU (using the original truth table only), and devise an ASM model that meets the requirements of a combinational logic model, namely that the operation can be completed in a single cycle.
- In addition, the input buses A and B will be 4-bits rather than 1-bit in width. This means the output bus will also be 4-bits. (Note that the control inputs and the carry-in and carry-out signals will be similar to those in the first version of the ALU model.)
- You can use nested conditionals, case constructs and macro-functions to implement the arithmetic and logical functions required for generating the outputs in the presence of the inputs and controlling signals.



Lab - The Multi-bit Arithmetic Logic Unit

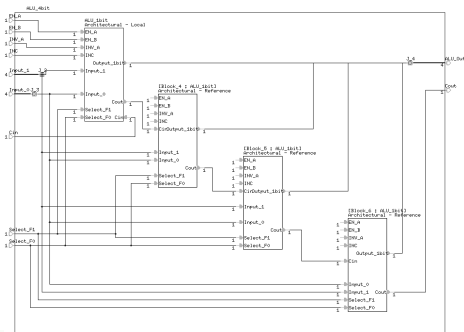


- ALU description
 - ✓ Provides the arithmetic, logic and other data functions in a single package.
 - ✓ ALU functions are selectable using control signals.
 - ✓ Individual gate-level elements are cascaded together to form an ALU of the computer's word length.
 - ✓ NOTE: The ALU incorporates the Full Adder model.

Source: Tanenbaum, 4th ed © 1999, Prentice-Hall



Lab – Structural 4-bit ALU



Laboratory ASM Worksheet

ALU_version_1

- Use this worksheet to hand-draw your initial ASM design.
- The single-thread ASM model should only have one state, as the circuit to be modeled is a combinational logic circuit.
- Can you model this version using a set of nested ASM control structures (if-then and case constructs)?



© 2002 Dr. James P. Davis Page 61

HW #3 – Testing Computation of the ALU

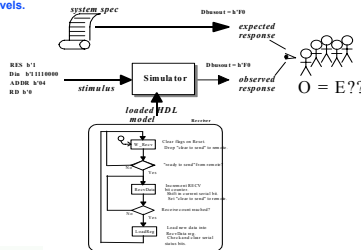
- For this assignment, you will take your 4-bit ALU and create a second ASM “test thread” that uses the ALU thread to implement a Subtraction function for signed integers.
- Subtraction is carried out as follows: (1) convert the second operand into two’s complement (invert and add 1), (2) perform addition of the operands, and (3) convert sum back from two’s complement (depending on the value of the “sign” bit).
- This will require consecutive operations using the ALU: (1) NOT(B), (2) B<-INCRNC(B), (3) ADD(A, B), (4) test of MSB (sign bit).



© 2002 Dr. James P. Davis Page 62

Debug Test & Verification Process

- **Approach to Verification**
 - Create a set of stimuli that can be used to verify observed behavior against expected behavior.
 - We'll use different levels of design and test specification, to take advantage of fast turnaround in gaining functional and cycle-level timing verification, then obtaining gate-level timing and load analysis, post-layout. The same test harness will be used at both levels.



© 2002 Dr. James P. Davis Page 63
