



High-level VLSI Digital Systems Design and Analysis: Hands-on Workshop

Dr. James P. Davis, Associate Professor
University of South Carolina
Columbia, S.C., U.S.A.

Ms. Yasmin Othman, R&D Engineer
Exsedia Sdn. Bhd.



© 2004 Exsedia Sdn. Bhd. & University of South Carolina



exsedia

Seminar - Session 2 Outline

Teaching and Practicing ASM-based Analysis & Design – Hands-on Workshop

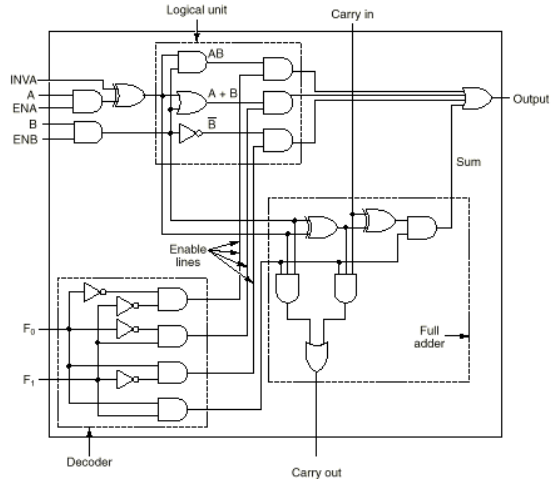
1. Six-function ALU block
 - relationship of ASM to gate-level schematic and truth table.
 - combinational logic design using macro-functions or algorithmic constructs and conditional outputs.
2. Integer Multiplier/Divider
 - algorithm design: control structures, branching, looping.
 - synchronous sequential finite state machine modeling.
 - register delay, signal “set and test”, signals as control variables.
 - verification testing: test planning, cycle-based simulation.



© 2004 Exsedia Sdn. Bhd. & University of South Carolina

Workshop #1 – Arithmetic Logic Unit

- ALU structure
 - A single-bit ALU consists of logic for (1) function decoding, (2) performing 3 logic functions AND, OR, NOT (B), (3) Full Adder unit, (4) Input enable logic, (5) NOT (A) function from separate input.
 - Creating a multi-bit ALU unit involves scaling the bit-widths of the input operands A, B and the Output.
 - We will start out by creating a single-bit ASM model for this circuit, then scaling it to 4 bits.
 - There are two styles of ASM model that can be created: (1) a structural model mimicking the gate schematic, and (2) a functional model independent of the structure.



Source: Tanenbaum, 4th Edition, ©1999 Prentice Hall Publishers, Inc.

Workshop #1 - ALU Model (Continued)

- ALU Function:
 - F₁, F₀ define selection of 4 functions: '00' AND, '01' OR, '10' NOT (input B), '11' ADD.
 - These functions are “qualified” by other control signals input to the ALU: (1) A,B input Enable lines, (2) INVA – NOT (input A) function, (3) INC increment function (which has special meaning, depending on the status of the other control inputs).
 - The outputs are defined in terms of the two inputs, but some outputs are dependent on a single input (if the control signals are configured appropriately).

F ₀	F ₁	ENA	ENB	INVA	INC	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

Workshop #1 - ALU Signal Declarations

- Thinking about the interface and internal signaling:
 - You need to define your signal interface as a collection of buses in Nimbus.
 - You'll start with the Inputs and Outputs (as labeled in the gate-level schematic) then consider the Internal signal interconnect.
 - You can enter these first, or wait until after you create the ASM diagram.

You won't declare these, as they are pre-defined.

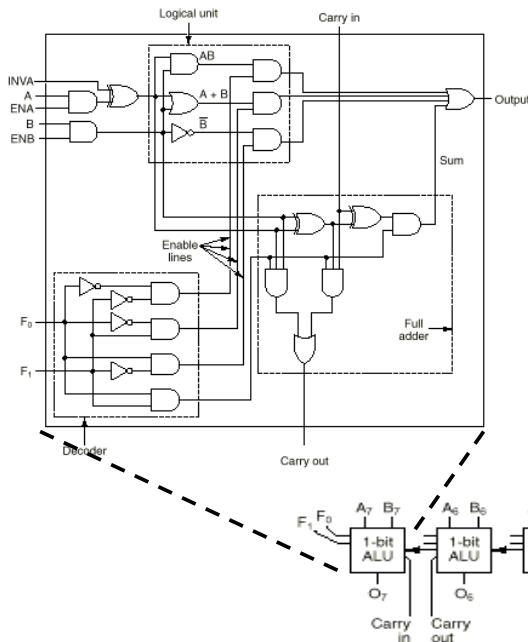
You'll declare these as "wires".

Design Name:	ALU_design1					
Designer:	Dinakar Kanuganti					
Version:	csce 491 - v1					
Date:	30 August 2003					
Value Format:	Hexadecimal					
Bus Name	Type	Mode	Element	Default Value	Value	Width
A	Binary	INPUT	Context	0	0	1
B	Binary	INPUT	Context	0	0	1
Carry_in	Binary	INPUT	Context	0	0	1
Carry_out	Binary	OUTPUT	Context	0	0	1
ENA	Binary	INPUT	Context	0	0	1
ENB	Binary	INPUT	Context	0	0	1
F0	Binary	INPUT	Context	0	0	1
F1	Binary	INPUT	Context	0	0	1
INVA	Binary	INPUT	Context	0	0	1
out_put	Binary	OUTPUT	Context	0	0	1
Sum	Binary	INTERNAL	Context	0	0	1
log_out1	Binary	INTERNAL	Context	0	0	1
log_out2	Binary	INTERNAL	Context	0	0	1
log_out3	Binary	INTERNAL	Context	0	0	1
RES	Binary	INTERNAL	Context	0	0	1
CLK	Binary	INPUT	Context	0	0	1

Workshop #1 – Single-bit ALU

- You will use Nimbus™ to create an ASM model of the design as described. The design entry consists of the following parts:
 - Declare the signals and buses in the Bus tree view dialog box. You'll need to declare all buses to have Element type = "wire".
 - Note we are starting with a single-bit ALU unit in this part of the workshop.
 - Create a model for the ALU block, as discussed. A combinational circuit will consist of an ASM "thread" with a single state. Thus, the design consists of modeling the operations using the Nimbus macro-functions, nested together, or of modeling the decision logic as conditional output operations using the ASM control constructs (Case, Condition), all looping on the single state of the ALU thread.
 - Note: you are also including declarations for CLK and RES signals, although this is a combinational logic design unit, so they will not be referenced in the actual design.
 - Compile your model: If you get errors during model compilation, you should go to the state where errors are flagged (indicated by an error code), and click the object while holding down the 'CTRL' key (CTRL + Left Mouse). This will bring up and explanation of the error in a dialog box.
 - Usually, the cause of errors is that you are using signals in an assignment expression that have different bus widths, different signal types, or signal modes (input, internal, internal_out, versus output). So, you'll have to debug the design a bit. Nimbus uses strongly typed signal definitions.

Workshop #2 - The Multi-bit Arithmetic Logic Unit



• ALU description

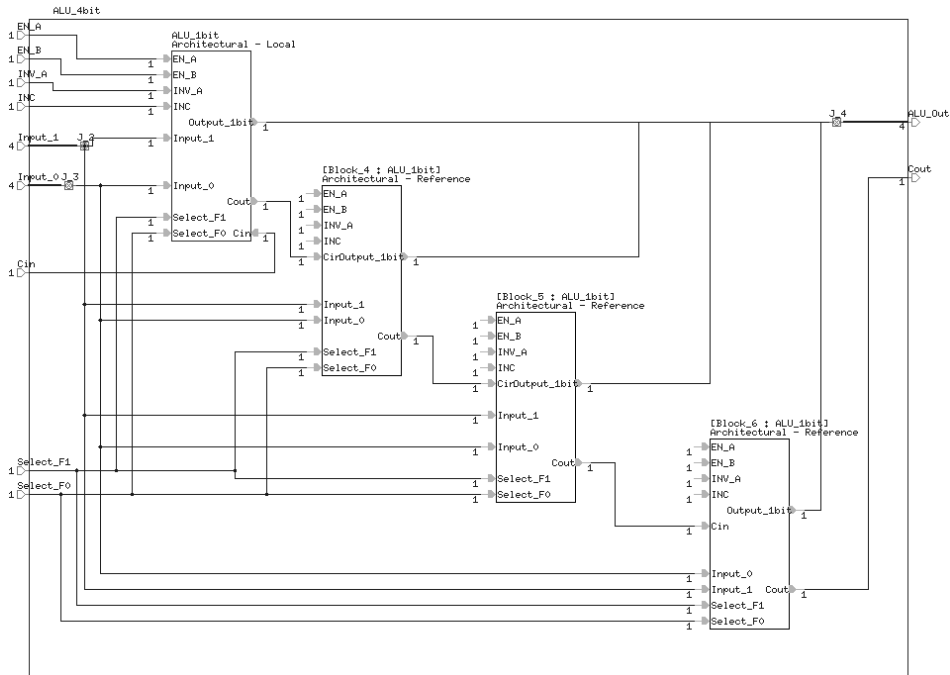
- ✓ Provides the arithmetic, logic and other data functions in a single package.
- ✓ ALU functions are selectable using control signals.
- ✓ Individual gate-level elements are cascaded together to form an ALU of the computer's word length.
- ✓ NOTE: The ALU incorporates the Full Adder model.

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

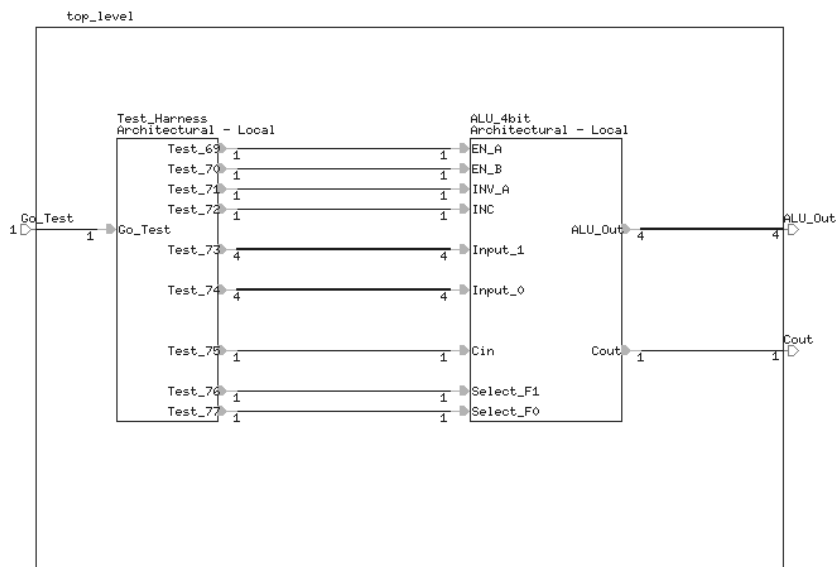
Workshop #2 – Scaling the ALU Model

- 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.

Workshop #2 – Structural ALU Model

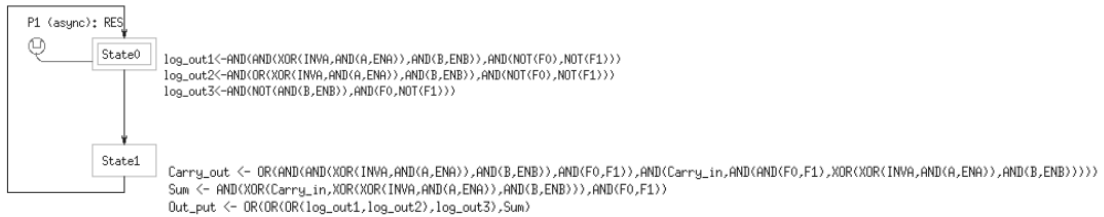


Workshop #2 – Functional ALU Model



Workshop #3 – Alternate ALU Model

- Some questions about the ALU model shown below:
 - This model has two states, and has operations scheduled on each state. Is there any problem with this design, in regards to its specification in the truth table shown earlier?
 - If you see a problem with how the design block has been modeled, how would you fix the problem (i.e., how would you model the ALU differently so that the problem goes away)?



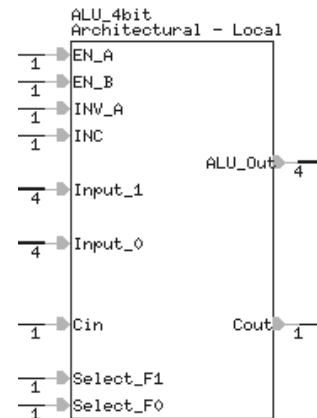
Workshop #4 – Integer Multiplication

- Basic Approach:
 - Pencil & paper method: we take one operand's digits (or "terms") and multiply it through the digits of the other operand. The resultant product term is brought down, aligned with the appropriate operand terms.
 - Each row is a partial product term, representing the result of multiplying each pair of operand digits. After the first partial product row is formed, subsequent rows are *shifted left* by the number of places of the operand term (either a single digit, a single bit, or multiple bits, depending on the size of the operand term being multiplied).

		Each pair of operand terms...							
Multiplicand	>				X3	X2	X1	X0	
Multiplier	>			x	Y3	Y2	Y1	Y0	
1st partial product	>				Y0X3	Y0X2	Y0X1	Y0X0	
2nd partial product	>			Y1X3	Y1X2	Y1X1	Y1X0		
3rd partial product	>		Y2X3	Y2X2	Y2X1	Y2X0			
4th partial product	>	+	Y3X3	Y3X2	Y3X1	Y3X0is multiplied to obtain a partial product term.	
Final product	>	P7	P6	P5	P4	P3	P2	P1	P0

Extending the ALU for Multiplier Function

- Question: how would you extend the ALU model to include a multiplier function?
 - Add a Function Select line, so that a new function could be added.
 - Modify the bus width of the ALU_Out bus, so that the 8-bit product of a 4x4 multiplication can be output from the ALU.
 - What about the relationship with other ALU control lines?
- Nimbus supports a MUL macro-function in its library, so this could be used.
 - How would you implement the “shift add” function for multiplication in a separate ASM thread?
 - How would you control its selection and execution?
 - How many control cycles would it take to multiply 2 4-bit Integers?



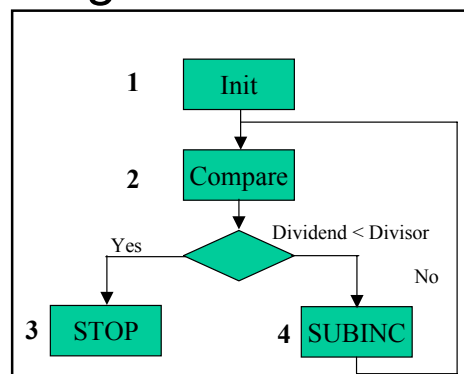
Workshop #4 – Integer Division

1. Initialization

- Count = 0
- Dividend = Input1
- Divisor = Input2

2. Compare Dividend with Divisor

- Dividend \geq Divisor, Go to 3
- Dividend < Divisor, Go to 4



3. Dividend bigger than Divisor: SUBTRACT & INCREMENT COUNT

- Dividend = Dividend – Divisor
- Count = Count + 1
- Go to 2

4. Dividend smaller than Divisor: STOP ALGO

- Quotient = Count
- Remainder = Dividend