UNIVERSITY OF SOUTH CAROLINA

CSCE 611
Digital Systems Design I

2005/3/27

Week 13 Supplemental Notes
Multiplication, Testing, and Memory-driven Test Pattern Generation

© 2004 Dr. James P. Davis

Some figures: Tanenbaum, 4th ed. © 1999, Prentice-Hall,
Lewin & Protheroe © 1992 Chapman Hall Publishers, Ltd.

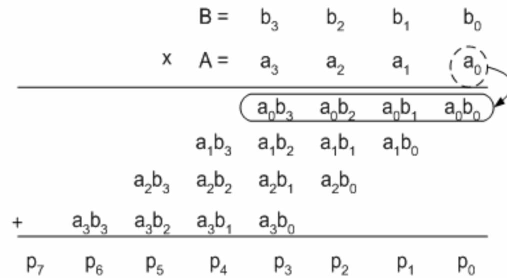
Outline

- Key points:
  - We discuss the Shift-Add multiplier model. We also discuss the Serial-Parallel model presented in the Wolf, 2003 text.
  - This is part of the set-up for the Lab Assignments (HWs #5&6).
  - In order to subject your model to rigorous testing, we'll discuss how to drive your ALU and MUL lab models using ASM-based test harness threads created in Nimbus.
  - Look at 2 different examples using ASM modeling:
    - Memory array (memory modeling in Nimbus).
    - Multiplier circuits – we'll explore testing the architectures of multipliers using memory-based test operand generators.

Multiplication Example – pen and paper

Source: Protheroe & Lewin © 1992, Chapman Hall

- Shift-add multiplication:
  - ✓ Take each digit of *multiplicand* operand, and multiply it by the first digit of the *multiplier* operand.
  - ✓ Bring each *partial product* term down into its column.
  - ✓ Repeat, multiplying each digit of *multiplicand* with each subsequent digit of *multiplier* operand, bring the *partial product* terms down onto the column and row.
  - ✓ Add the *partial product* terms, column by column, to generate the *product*.



$p_0 = a_0b_0$

$p_1 = a_0b_1 + a_1b_0$

$p_2 = a_0b_2 + a_1b_1 + a_2b_0$

$p_3 = a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0$

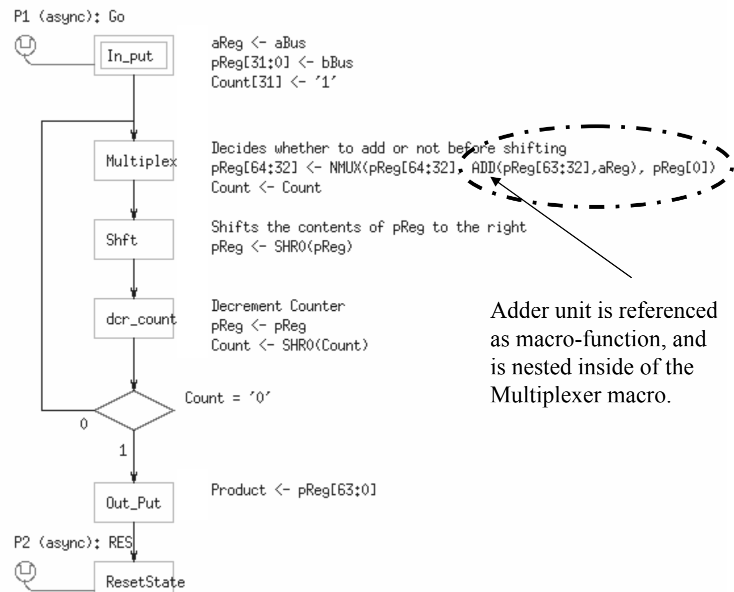
...

Using Add Macro in a Multiplier Model

This is a reference to a behavioral Adder macro model, maintained by Nimbus™, with all bits operated on in parallel by a 32-bit unit. It is a higher-level model.

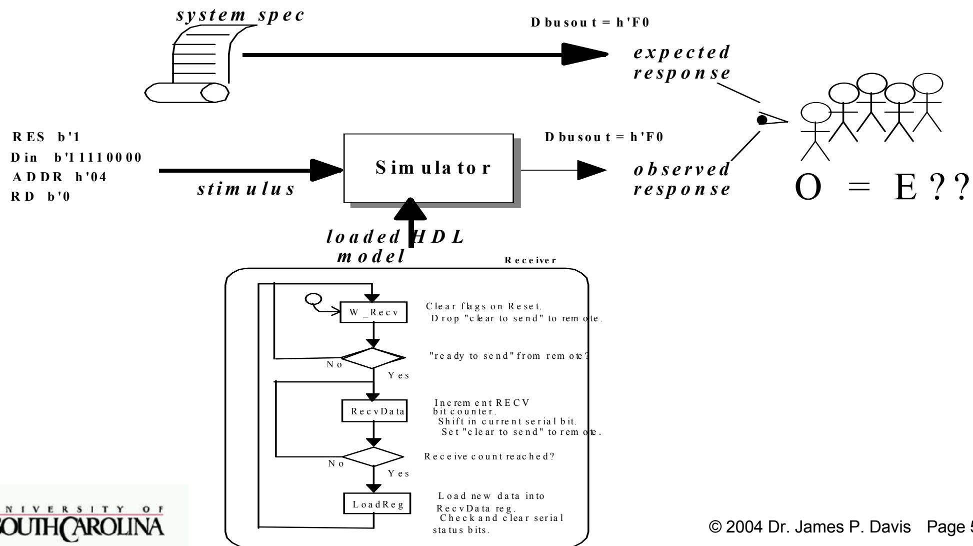
The shift-add Multiplier scheme is the most basic of unsigned Integer multiplication algorithms.

Note the algorithmic nature of the model: (1) loop control using a "count" register, (2) concurrent operations scheduled on each state, (3) bus slicing, shifting and register-register assignment.



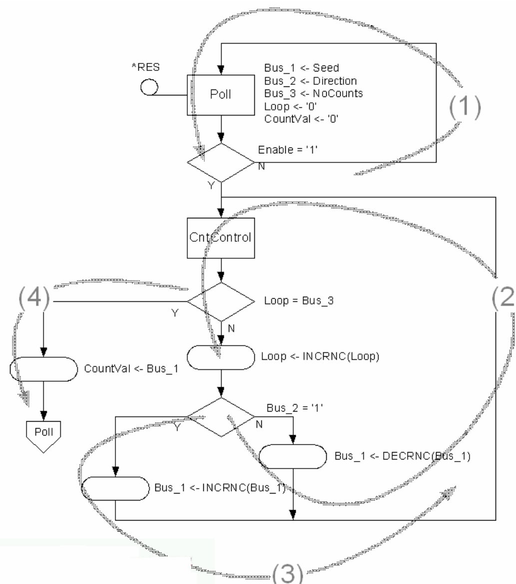
Systems Design Method – Test Process

- 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.



Creating a Test Plan – ASM Design Model

- In examining an ASM "thread", we want to evaluate which logic paths we can test by defining a set of input stimuli to excite the design, so that each run allows different logic to be tested.



- **<u>Test points</u>:**
  - ✓ We start by looking at specific points allowing us to check different logic paths through the design.
  - ✓ We look at the signals and the value ranges that will cause our design to choose a different logic path.
  - ✓ We then create a set of stimulus "steps" that will allow us to trace the design in simulation.

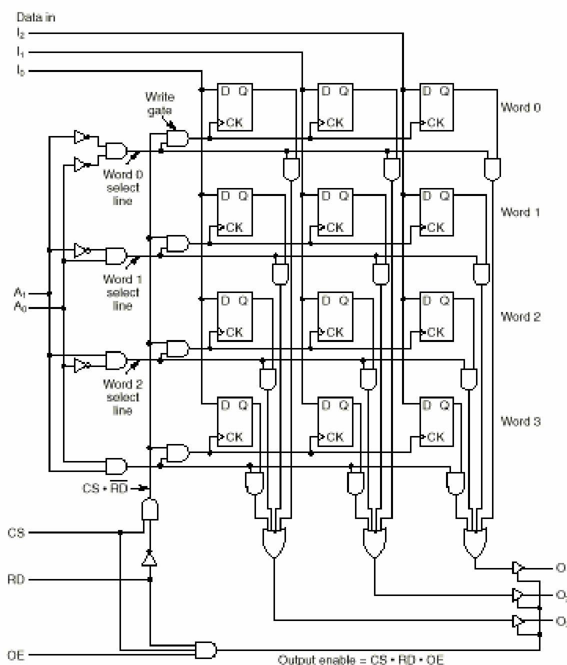
The ASM Test Case Planning Worksheet

Instructions: For each test case scenario, fill in one row in the table. Number each test case. Give each one a meaningful test name. For each case, indicate as precisely as you can, (1) the input bus set-up to trigger your test, and (2) the expected results of your test run, in terms of bus values you should see, and on what simulation clock cycles you expect to see these values. The simulation clock cycle should be indicated for intermediate values, or when you are watching an output bus change value multiple times during a single test run. These fields get filled out **before** you execute the simulation run. The field for observed description of the simulation run results is filled in **after** you simulate. This should describe the overall results. This would provide a narrative that goes along with the simulation waveform output.

Test No.	Test Name	Simulation Input			Expected Results			Observed Outcome (in text)
		Elapsed Cycle #	Input Bus Name	Value	Elapsed Cycle #	Bus Name	Value	

Name:____________________Sheet _____ of _____

The Gate-level Structure of a Memory Array

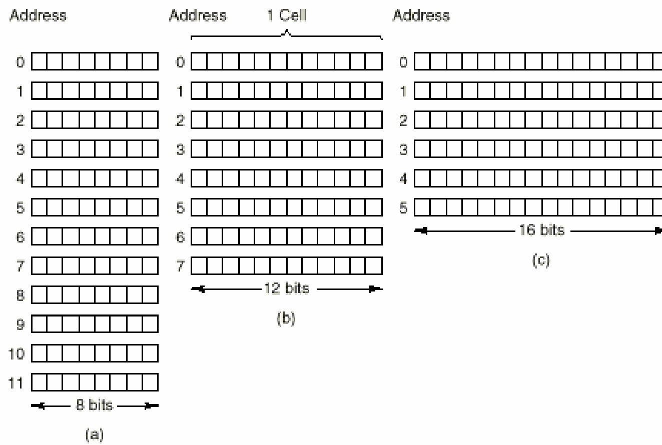


Source: Tanenbaum, 4th ed. © 1999, Prentice-Hall

- 4 x 3 Memory Array
  - The Memory array is built up from gates and flip flops, to take advantage of certain properties of the devices. Each row is one of four 3-bit words.
  - Data Lines $I_0 - I_2$ feed all of the D FF's in a column. The address lines $A_0 - A_1$ act as select lines for a given bank.
  - The control signals CS (control select), RD (read enable), OE (output enable) are used to route data to and from memory (allowing writes and reads) based on the combinational logic gates.
  - The bus "drivers" are enabled by the AND of the 3 control signals.

University of South Carolina

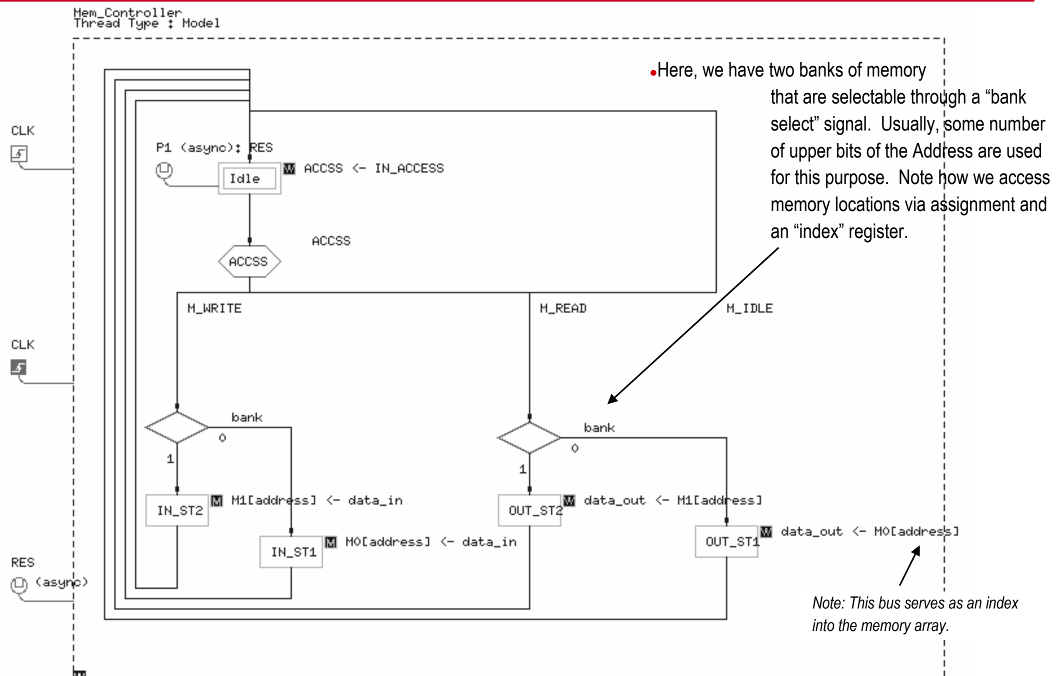
Memory Structure and Organization



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

- *m x n* Memory Array
  - Memory arrays allow multi-bit values to be stored and retrieved by address.
  - A memory of n bits can be organized in different ways.
  - Memory array as a *word length*: the number of bits of each word stored in memory (e.g., 32 bits).
  - Memory array has *number of words* (e.g., 128MB).
  - Each memory location is uniquely addressable (called *content addressable memory*).

Memory Use Example – RAM Control Thread



Memory Use Example – Bus Table

Bus Table – Memory_Controller–021105.flo

Code	Bus Name	Type	Mode	Element	Width	Comment	Values:	Default Value:
	CLK	Binary	Input	Wire	1	N	0	0
	RES	Binary	Input	Wire	1	N	0	0
	ACCSS	CONTROL	Internal	Wire	0	N	M_IDLE	M_IDLE
	IN_ACCESS	CONTROL	Input	Wire	0	N	M_IDLE	M_IDLE
	data_in	MVL3	Input	Wire	8	N	00	00
	data_out	MVL3	Internal Out	Wire	8	N	00	00
	bank	MVL3	Input	Wire	1	N	0	0
	address	MVL3	Input	Wire	4	N	0	0

undo, define, modify, delete, error, set, format, view, sort, quit

Note: This bus serves as an index into the memory array.

Note: Buses defined that will either access memory or be used as a memory index register need to be declared as MVL3 type.

Note: the enumerated values set – M_READ, M_WRITE, M_IDLE, M_REFRESH, defined in the Enumeration Table in flowHDL.

Memory Use Example – Memory Table

- Memory creation consists of several steps:
- (1) Create a RAM array in the Memory Table (with Addr. Width, Data Width and Type).
- (2) Edit values in the array using the Matrix Table.
- (3) Reference the memory locations using an index.

Memory Table – Memory_Controller–021105.flo

Name	Address Width	Data Width	Access	Type
M0	4	8	RAM	MVL3
M1	4	8	RAM	MVL3

undo, define, modify, delete, matrix, quit

Memory Modification

Name: M0

Address Width: 2, 16, 4, Other, 8

Data Width: 1, 16, 2, 32, 4, 64, 8, Other

Access: RAM, ROM

Type: MVL3, MVL7, MVL4, MVL9

Modify, Cancel, Help

M0 – Matrix Table

format, clear, set, open, save, quit

ADDRESS	DATA (Hexadecimal)			
0000 : 0003	F9	E7	13	6
0004 : 0007	FF	74	66	0
0008 : 000B	4F	23	10	58
000C : 000F	19	6E	22	F

Set Data Value

Start Address: 0001

End Address: 0001

Format : Hexadecimal Width : 8

Value: E7

OK, Cancel, Help

Arbitrating Access to Memory Arrays

- Treating memory as a shared resource:
  - ✓ If we define a memory array, and only reference it from a single ASM thread, then we can access it without having to define arbitration.
  - ✓ If we will access (read or write) to the memory array from more than one ASM thread, then we need to define an arbitration scheme.
  - ✓ Arbitration is the process of “clients” requesting service, and a “server” granting access to the shared resource according to some policy.
- Ways to get around this need to arbitrate:
  - ✓ “Hide” memory from other threads by defining one thread to control access to it; this is the only thread that sees it.
  - ✓ In both flowHDL® and Nimbus™, if you have memory access in assignment expressions, the Checker will flag a “multi-drive” error, because more than one entity is trying to control the memory, which is a shared resource.

Using A Memory-based Test Pattern Generator