# EE577B Homework #4

# Asynchronous Multiplier

# Due: 10/25/2001

## 1. Comparison test bench and 8-bit PCHB full adder behavior view

The objective of this part is to design a test bench that scans all combinations of inputs and compare the results of two different cells views of the same circuit, in this case, the 8-bit PCHB full adder implemented in the previous homework.

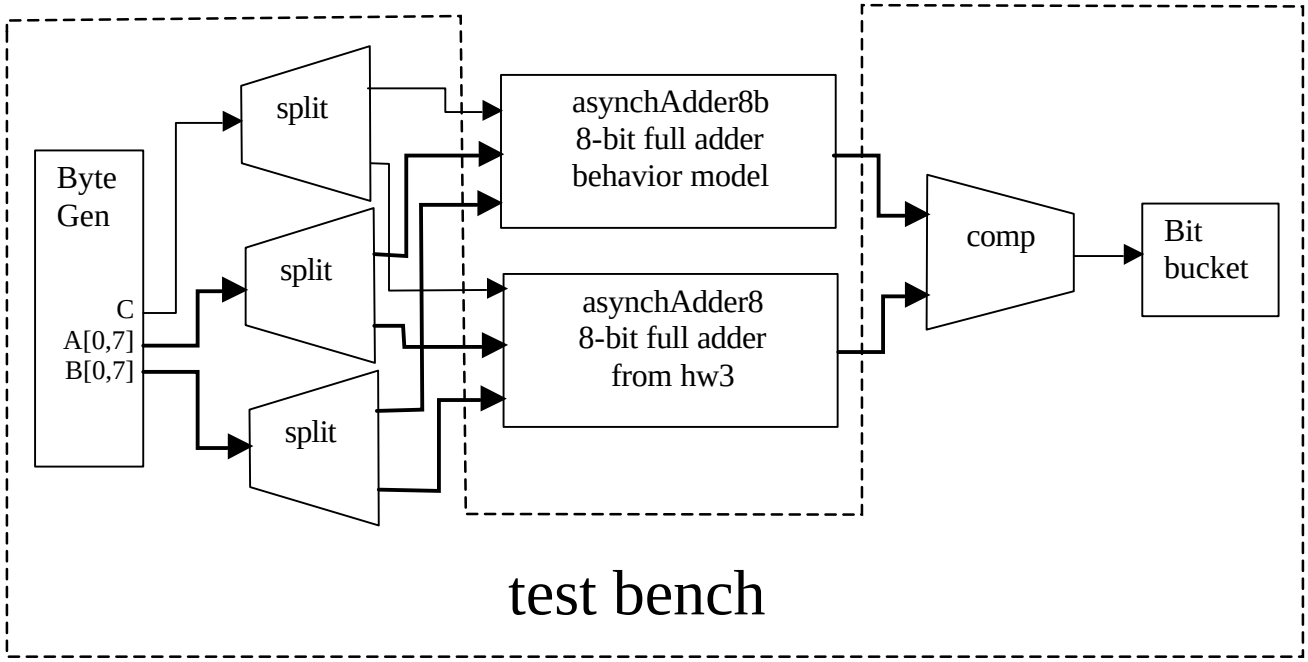
a) Create a "**splitWCHB"** element with one dual rail input (L) and two dual rail outputs (RA and RB). Create the symbol and functional views **only**. In the verilog code use some reasonable delay in order to make the WCHB Split not too fast (don't use zero delay) and not too slow (it must not be the bottleneck of the test). The splitWCHB cell waits until RA and RB are ready to receive data, then, if L is available, it copies L to RA and RB. When both RA and RB outputs acknowledge, the splitWCHB acknowledges L.

b) Use 8 splitWCHB's to create a **split8WCHB** (symbol and schematic views).

c) Create a "**compWCHB**" (comparator) element with two 9-bit dual rail input channels (LA[0,8] and LB[0,8]) and one 1-bit dual-rail output channel (R), which returns high if LA is equal to LB. The compWCHB waits for both 9-bit dual-rail numbers to arrive before generate the 1-bit dual-rail output.

d) Copy the symbol cell view of the cell 'asynchAdder8' of the previous homework to a new cell '**asynchAdder8b**'. In this new cell, create a functional view with the **behavioral** description of this 8-bit adder (We are looking for a better way to do this instead of creating a new cell).

e) Create a byte generator that scans all the combinations of the inputs as used in the previous homework and the test bench schematic as shown below.



Submission: Top level schematic of the test bench, the verilog code of the asynchAdder8b, the split and the comparator. Plot waveforms of some of the scanned inputs and their results. Group the signals by byte. For example: S1[0,7] and S0[0,7] Se[0,7] representing one of the outputs.

## 2. 8x8 Multiplier.

The objective of this part is to implement an 8-bit multiplier ('**asynchMult8**'). The suggested architecture is the carry-save multiplier with a vector-merging adder at the bottom.

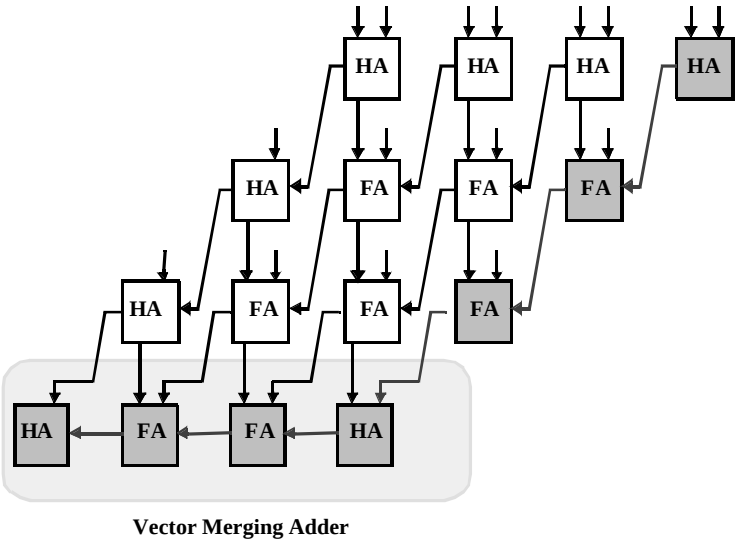



**Figure 1. A 4 x 4 carry-save multiplier.**

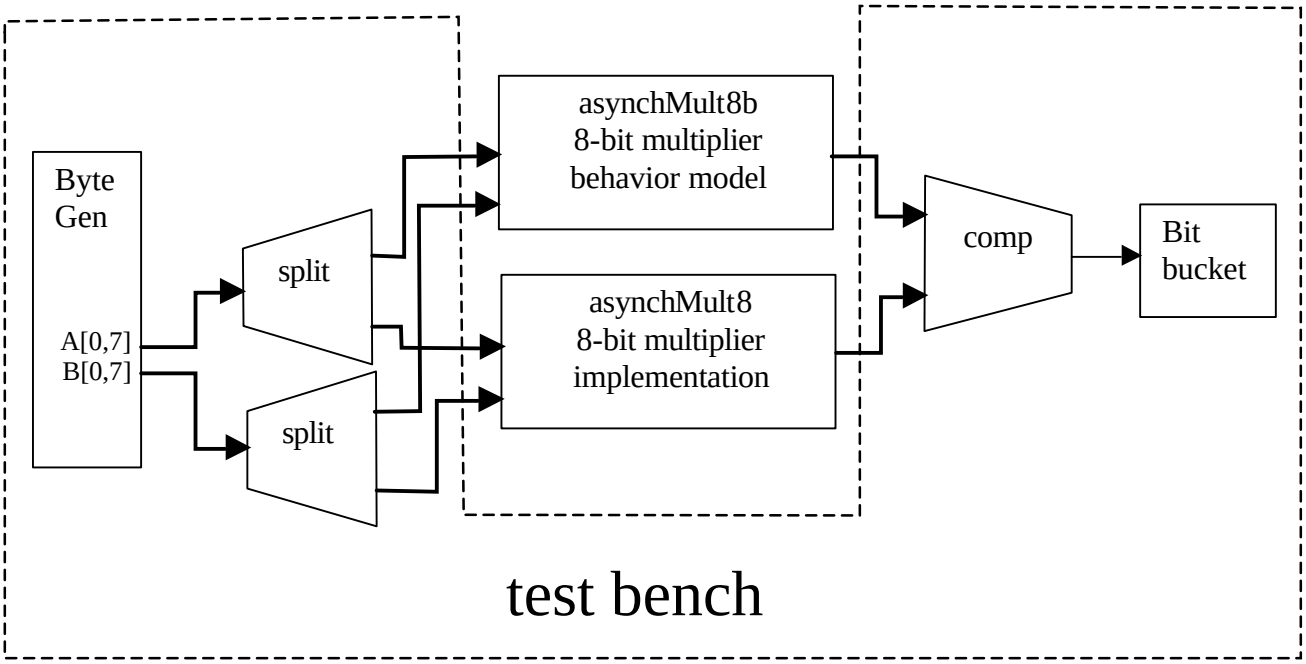
**(from J. M. Rabaey, *Digital Integrated Circuits*, Prentice Hall, NJ 1996)**

Figure 1 illustrates a synchronous 4 x 4 carry-save multiplier. We want to build an asynchronous 8 x 8 with a very high throughput (fine grain pipelined).

a) Since the multiplication of each two bits is an AND operation (see J. M. Rabaey, *Digital Integrated Circuits*, chapter 7.4), we need to implement some sort of "**andWCHB"** stage and connect it properly to the multiplier. There are many different ways to implement this. Choose one that you think is the best for throughput. Create just the symbol and functional views.
b) Based on the "fullAdderPCHB" cell of the hw3, create the "**halfAdderPCHB**" with just symbol and functional views.
c) Create the "**asynchMult8**" cell with schematic and symbol views. Be sure that the vector-merging adder is not a bottleneck in your design and use enough buffers to maximize the multiplier throughput.
d) Copy the symbol cell view of the cell 'asynchMult8' to a new cell '**asynchMult8b**'. In this new cell, create a functional view with the **behavioral**

description of this 8-bit multiplier (We are looking for a better way to do this instead of creating a new cell).

e) Using a test bench similar with part 1 of this hw, demonstrate that your multiplier is working properly. Notice that the comparator now has two 16-bit input channels.




Submission: Top level schematic of the test bench, of the asynchMult8 stage and the verilog code of the asynchMult8b. Also submit symbol, schematic and verilog code of any new cell with some explanation of its function. Plot waveforms of some of the scanned inputs and their results. Group the signals by byte or word. For example: M1[0,15] and M0[0,15] Me[0,15] representing one of the outputs.