EE577B Homework #3

Asynchronous Basic Blocks

Due: 10/11/2001

As a good reference, see the circuits in Lines thesis.

1. Dual-Rail Weak-Condition Half-Buffer Buffer (bufferWCHB)

Initially you need to create the cells to run an hspice simulation, check the timing of your circuits and, then, add the appropriate delays in the bufferWCHB, bitGen and bitBucket functional views.

a) Create a parameterized 2-input C-element (symbol and schematic views). The C-element should have an active low rst (reset) input.
b) Create a staticizer (symbol and schematic views). The staticizer uses the smallest transistor size inverter with weak transistors (long L) in series with Vdd and Gnd (10x weaker than the minimum size transistor).
c) Use the C-element, the staticizer and other basic gates to create a cell 'bufferWCHB' (symbol, schematic and functional views).
d) Create a bit generator cell ('bitGen') (symbol, schematic and functional views) with an active high enable input (just use one nand2 and an inverter).
e) Create a bit bucket cell ('bitBucket') (symbol, schematic and functional views) with an active high enable input (just use one nor2, one nand2 and an inverter).

Now, create a new cell ('pipeline') with a schematic view where you connect 40 buffers in series with the bit generator in one side and the bit bucket in the other. Label the wires in such way that you may probe for specific signals after your simulation.

Generate the hspice file and include the necessary lines to perform the following simulations:

- Apply Vdd=2.5V, T=25C and perform a transient analysis for 500 ns.
- Reset all the buffers and hold the bit generator and bit bucket disabled for 10 ns.
- Release the reset and the bit generator for 200 ns.
    - o Check how long does it take for the first token to reach the end of the pipeline. This number divided by 40 is the forward latency of the buffer.
- Release the bit bucket for the remaining 300 ns.
    - o Check how long does it take for the first acknowledge signal to reach the beginning of the pipeline. This number divided by 40 is the backward latency of the buffer.
    - o Check the time between two tokens entering in the pipeline while there are tokens leaving the pipeline (at full speed). The inverse of this timing is the throughput of the pipeline (in Hz).

Adjust your verilog code in the functional views of the bufferWCHB, bitGen and bitBucket cells to match as close you can the latencies measured on the hspice simulation. Use 10 time units equal to 1ns (1 time unit per 100 ps).

Generate your verilog simulation similar to the hspice above. Compare the results.

Submission: C-element and pipeline schematic plot, bitGen, bitBucket and bufferWCHB schematic, symbol and functional view plots. Plot the mwaves waveforms with your measures of the forward latency, backward latency and throughput. Plot similar waveforms from your verilog simulation.

1.1. Generate the throughput vs. tokens in the pipeline graph.

This is the "triangular graph" presented in class. To generate this graph you have to run your hspice simulation several times with some A/D sampler circuits added to it and then, for each simulation A/D output file, apply the script tokenCount.prl to calculate the average number of tokens inside the pipeline. After reset all the circuits, allows it runs in regime for a while (~ 500 ns).

- Copy tokenCount.prl using: cp ~ee577/ee577bb/hw/tokenCount.prl
- To execute it type: tokenCount.prl < filename.A2D

Start with the fastest bitBucket and bitGen cells. Then, generate at least 3 simulations with slower bitGen (adding pairs of inverters to slow it down) and full speed bit Bicket. Repeat the simulations using the fastest bitGen and slowing the bitBucket. For each simulation calculate the average number of tokens in the pipeline.

The addition to your spice file is:

```
UA A VREF A2D SIGNAME=A
UZ Z40 VREF A2D SIGNAME=Z40

VREF VREF gnd! DC 0.0V
.MODEL A2D U LEVEL=4 TIMESTEP=0.01NS TIMESCALE=1
+ S0NAME=0 S0VLO=-1 S0VHI=2.0
+ S4NAME=1 S4VLO=0.5 S4VHI=9.0
```

where, A is at the input of the pipeline and Z40 is at the output.

Notice that Recep Ozdag created the script and the addition to the spice file above, and he used debug functions in his script. You can use and/or modify these functions and the parameters in the spice file addition to understand how it works. Also check the hspice manual for more information.

Submission: Graph plot with at least 7 points.

2. Dual-Rail Pre-Charge Half-Buffer Full-Adder (fullAdderPCHB)

Initially you need to create a 3-input C-element, run an hspice simulation, check the timing of your circuits and, then, add the appropriate delays in the fullAdderPCHB functional views.

a) Create a parameterized 3-input dynamic C-element (symbol and schematic views). The C-element should have an active low rst (reset) input.
b) Use the C-element, the staticizer and other basic gates to create a cell 'fullAdderWCHB' (symbol, schematic and functional views). Add the necessary transistors to reset the outputs (S and Cout) considering that the inputs will also be reset.

Now, create a new cell ('fullAdderTest') with a schematic view where you connect bit generator to the three inputs and two bit buckets to the outputs fullAdderPCHB cell. Label all the wires in such way that you may probe for specific signals after your simulation

Repeat the simulation below by connecting the bit generator to the full adder in order to test all possible combinations of A, B and Cin (000, 001, 010, 011, 100, 101, 111)

Generate the hspice file and include the necessary lines to perform the following simulations:

- Apply Vdd=2.5V, T=25C and perform a transient analysis for 20 ns.
- Reset the fullAdder and hold the bit generator and bit bucket disabled for 5 ns.
- Release the reset and the bit generator for 5 ns.
    - Check how long does it take for the first token to reach the S and the Cout outputs. This numbers are the forward latency of the full Adder.
- Release the bit bucket for more 10 ns.
    - Check how long does it take for the first acknowledge signal to reach the bit generator. This number is the backward latency of the buffer.
    - Check the time between two tokens entering in the full adder while there are tokens consumed by the bit bucket. The inverse of this timing is the throughput of the full adder (in Hz).

Adjust your verilog code in the functional views of the fullAdderPCHB cell to match as close you can the latencies measured on the hspice simulation. Use 10 time units equal to 1ns.

Generate your verilog simulation similar to the hspice above. Compare the results

Submission: 3-input C-element schematic plot, fullAdderPCHB schematic, symbol and functional view plots. Plot the mwaves waveforms with your measures of the forward latency, backward latency and throughput. Plot similar waveforms from your verilog simulation.

3. 8-bit Asynchronous Ripple-Carry Adder (asynchAdder8)

Create a new cell ('asynchAdder8') with a schematic view where you connect 8 fullAdderPCHB cells in such a way that we will have an 8-bit Asynchronous Ripple-Carry Adder. Since the carry signal has to travel through all the full adder cells, allocate **tens of bufferWCHB** cells around the full adders in order to match the slacks and optimize the throughput.

The asyncAdder8 symbol view should have A0...A7, B0...B7 and Cin as inputs and S0...S7 and Cout as outputs (remember that these inputs and outputs are dual-rail channels each one with its respective enable signal; for example: At0, Af0 and Ae0 are the true, false and enable signals of the channel A0).

Generate your verilog simulation similar to the 8-bit adder tutorial test. You may wish to wait for all the input enable signals of each input channel (A and B) to assert before you write a new data and to all the output channels to be valid before you consume the data.

Submission: asynchAdder8 schematic plot. Plot the waveforms with your verilog simulations showing the asynchAdder8 correctness operation, forward latency, backward latency and throughput.