Homework 1

Thomas Murphy (*trm70*)
September 29, 2015

Problem 1. Ripple-Carry Adder

(a) Design the 8-bit ripple-carry adder with 10-unit gate delay.

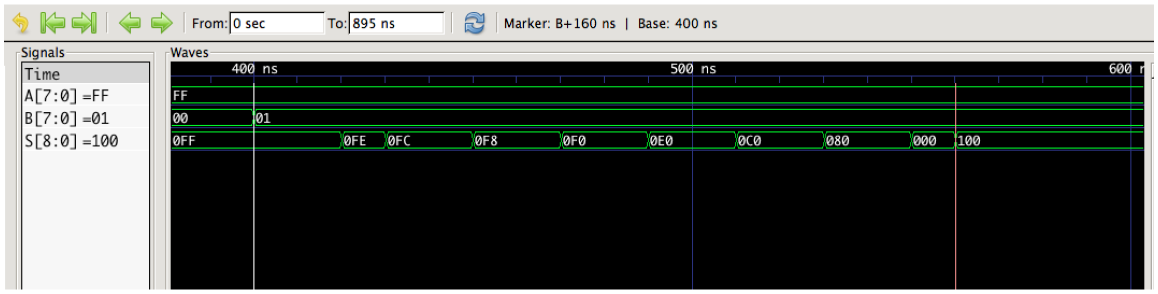
The adder is implemented in `q1/adder.v`, the full-adder and `q1/adder_rc_8.v`, the instantiation and connection of 8 full-adders in the ripple-carry configuration.

(b) Input with largest delay on most significant bit

The path with the largest delay to propagate to to the MSB of the sum is the one with the most carries created. The 4-bit case is illustrated in the assignment: taking the sum of `4'b1111` and `4'b0001`. This path can be demonstrated by transitioning the B input to the adder from `0` to `1`. Disregarding interconnect delays, the combinatorial path for this transition takes 2 gate delays to propagate through each full adder along the n-bit adder. Thus, with a 10-unit gate delay and an 8-bit adder, the expected delay on this path is 160 units.

(c) Verification of timing results

The testbench to produce the largest delay is implemented in `q1/tb_adder_rc_8.v`, the 8-bit testbench. The waveform produced from the testbench around the largest propagation delay transition is visualized below.



The expected gate delay is seen between the B marker and the primary marker with a difference of 160 ns in a 1 ns delay unit simulation.

Problem 2. Carry-Lookahead Adder

(a) Design the 4-bit CLA adder with 10-unit gate delay

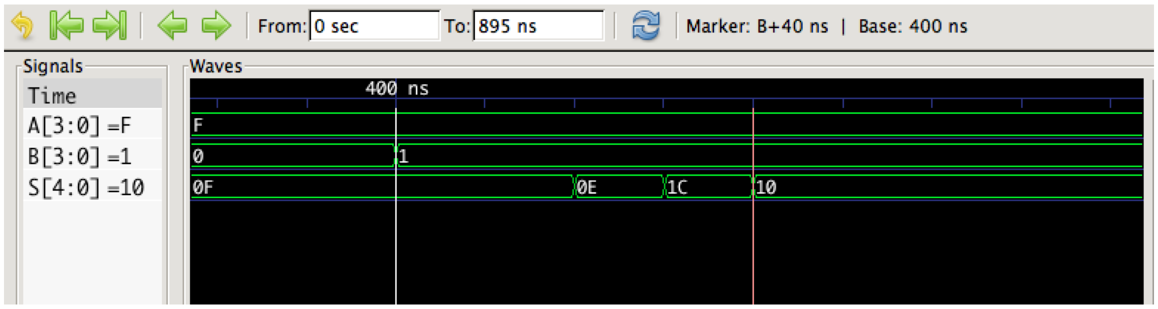
The adder is implemented in `q2/adder_cla.v`, the partial full-adder and `q2/adder_cla_4.v`, the instantiation and connection of 4 partial full-adders and the 4 carry-look-ahead logic modules (`q2/cla0.v`,`q2/cla1.v`,`q2/cla2.v`, and `q2/cla3.v`).

(b) Input with largest delay on most significant bit

The path with the largest delay to propagate to the MSB of the sum is any update to the input that produces the final carry: either by generation or by propagation. Since all of the propagation and generation signals are produced within two gate delays and the wide-AND/wide-OR implementation of the final CLA produces a carry out in two gate delays, the maximum delay is four gate delays. With a 10-unit gate delay, the expected delay on this path is 40 units.

(c) Verification of timing results

The testbench to produce the largest delay is implemented in `q2/tb_adder_cla_4.v`. It implements the narrower case as in Question 1, adding `0` and then `1` to `4'b1111`. The waveform produced from the testbench around the largest propagation delay transition is visualized below.



The expected gate delay is seen between the B marker and the primary marker with a difference of 40 ns in a 1 ns delay unit simulation.

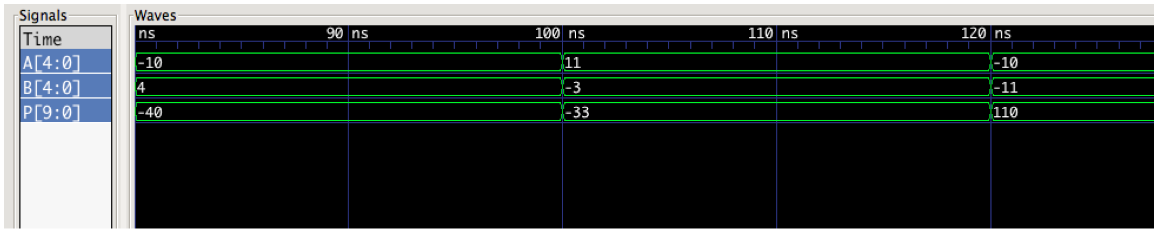
Problem 3. Signed Multiplier

(a) Write a Verilog design for this multiplier

The top-level multiplier is implemented in `q3/multiplier_signed_5.v` along with the supporting modules and testbench.

(b) Simulate multiplier with three input pairs: $(-10, 4)$, $(11, -3)$, $(-10, -11)$.

Simulating with the required three input pairs gives the result:



Problem 4. Carry-Save Adder

(a) Design a CSA to add a sequence of 10 8-bit numbers. How many CSA stages are needed?

The adder needs five CSA stages. They operate, for the inputs `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, `j`, and an output `z` as follows:

(i) Stage 1 implements 3 8-bit input CSAs. It consumes the values `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`. It produces the values `s_abc`, `c_abc`, `s_def`, `c_def`, `s_ghi`, `c_ghi`.

(ii) Stage 2 implements 2 9-bit input CSAs. It consumes the values `s_abc`, `c_abc`, `s_def`, `c_def`, `s_ghi`, `c_ghi`. It produces the values `s_csa2_A`, `c_csa2_A`, `s_csa2_B`, `c_csa2_B`.

(iii) Stage 3 implements 2 10-bit input CSAs. It consumes the values `s_csa2_A`, `c_csa2_A`, `s_csa2_B`, `c_csa2_B`, `j`, `0`. It produces the values .

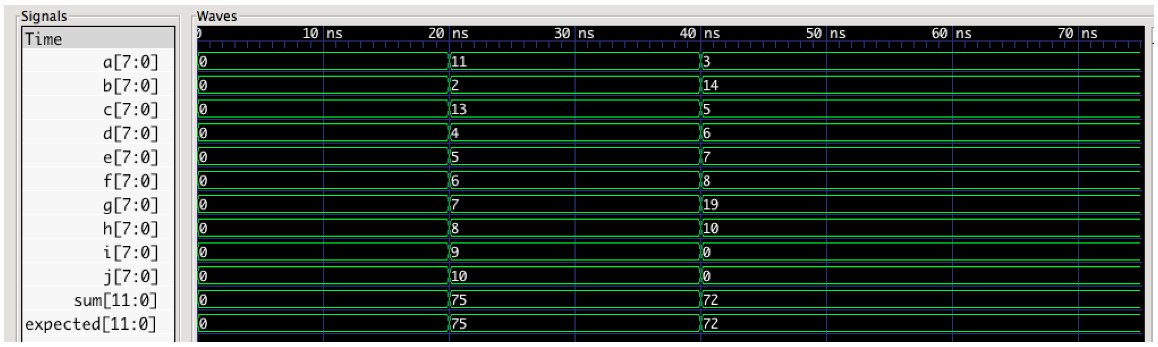
(iv) Stage 4 implements 1 11-bit input CSA. It consumes the values `s_csa3_A`, `c_csa3_A`, `s_csa3_B`. It produces the values `s_csa4`, `c_csa4`.

(v) Stage 5 implements 1 12-bit input CSA. It consumes the values `s_csa4`, `c_csa4`, `c_csa3_B`. It produces the values `s_csa5`, `c_csa5`.

(vi) The final stage of the adder is a normal ripple-carry adder for two 13-bit numbers. It consumes the values `s_csa5`, `c_csa5`. It produces the result value `z`.

(b) Simulate adder with two input tuples: $(11, 2, 13, 4, 5, 6, 7, 8, 9, 10)$, $(3, 14, 5, 6, 7, 8, 19, 10)$.

The simulation result with the 10-tuple and the 8-tuple, in comparison with the Verilog-computed result are:



```
Signals           Waves
Time              0        10 ns     20 ns     30 ns     40 ns     50 ns     60 ns     70 ns
a[7:0]            0                  11                  3
b[7:0]            0                  2                   14
c[7:0]            0                  13                  5
d[7:0]            0                  4                   6
e[7:0]            0                  5                   7
f[7:0]            0                  6                   8
g[7:0]            0                  7                   19
h[7:0]            0                  8                   10
i[7:0]            0                  9                   0
j[7:0]            0                  10                  0
sum[11:0]         0                  75                  72
expected[11:0]    0                  75                  72
```

Problem 5. State Machine

(a) Derive a state diagram

Since a state table is easier to textually represent, the four (Y_2, Y_1) pairs are states that have the following next-states based on the value of X

(Y_2, Y_1)	S	$S^*@X = 0$	$S^*@X = 1$
$(0, 0)$	A	D	C
$(0, 1)$	B	A	A
$(1, 1)$	C	A	B
$(1, 0)$	D	D	C

(b) Write a structural Verilog model and simulate it

Implemented in `q5/fsm_structural.v`.

(c) Write a behavioral Verilog model from the state-diagram and simulate it

Implemented in `q5/fsm_behavioral.v`.

(d) How to verify that the models in (a) and (c) are equivalent?

During simulation, the state transitions in the table can be compared with the state transitions displayed by the behavioral model given a certain `clk` and `x` stimulus.