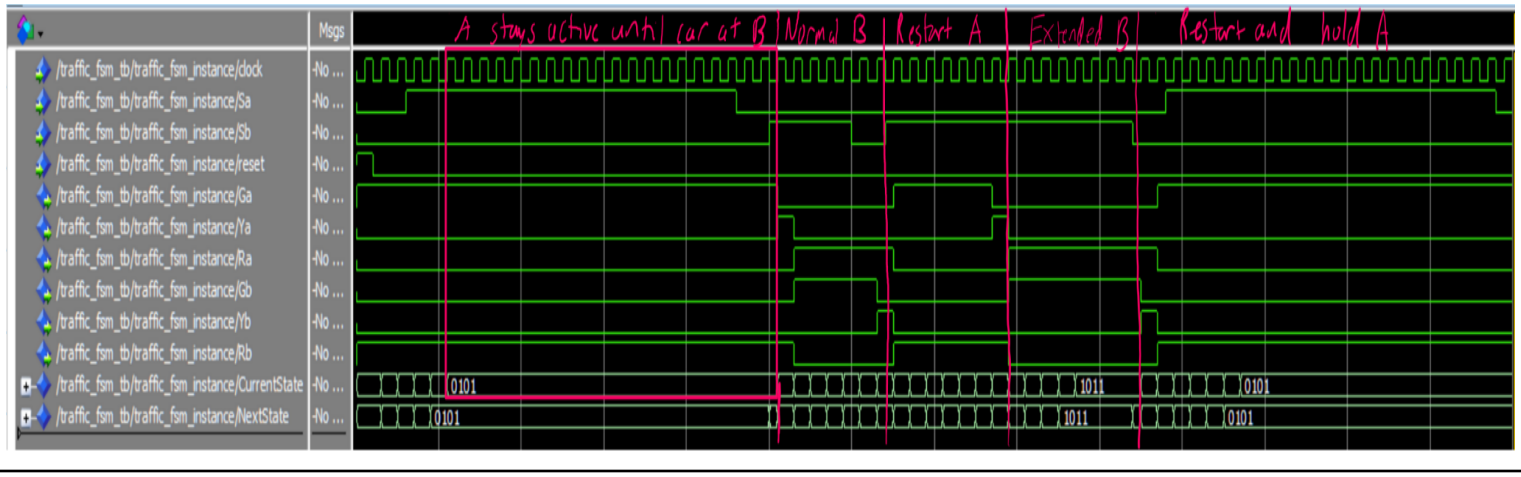
ECSE 318 Lab 5 Report

Group 5

Nabeeh Daouk and Kyle Heston

Thursday, November 30, 2023

Problem 1:



Testbench results for traffic controller FSM. Annotated to show key transitions. Order of testing: normal operation where A street is in green light mode and B street is in red light. Once Sb indicates a car on B street, the A light goes yellow then red. Then there is normal B operation where the B light is green for 5 clock cycles, then the machine restarts. Normal A operation occurs, then B light is green. B light is then extended while Sa is low and Sb is high. The system then goes back to A light green when Sa goes high and holds that state.

Problem 2:

```
# run -all
# Test Case 1: A=abcd, B=1234, Cin=0, Sum=be01, Cout=0
# Test Case 1: A=43981, B= 4660, Cin=0, Sum=48641, Cout=0
#
# Test Case 2: A=ffff, B=0001, Cin=1, Sum=0001, Cout=1
# Test Case 2: A=65535, B=    1, Cin=1, Sum=    1, Cout=1
# ** Note: $stop    : C:/Users/nabeehdaouk/source/repos/ECSE318/Lab5/Problem2/16bitRCA_tb.v(37)
```

Results from 16-bit Ripple Carry Adder TB

```
# run -all
# Test Case 1: A=abcd, B=1234, Cin=0, Sum=be01, Cout=0
# Test Case 1: A=43981, B= 4660, Cin=0, Sum=48641, Cout=0
#
# Test Case 2: A=ffff, B=0001, Cin=1, Sum=0001, Cout=1
# Test Case 2: A=65535, B=    1, Cin=1, Sum=    1, Cout=1
# ** Note: $stop    : C:/Users/nabeehdaouk/source/repos/ECSE318/Lab5/Problem2/16bitCSA_tb.v(36)
```

Results from 16-bit Carry Save Adder TB

```
# run -all
# Test Case 1: A=8abcd, B=41234, Cin=0, Sum=cbe01, Cout=0
# Test Case 1: A= 568269, B= 266804, Cin=0, Sum= 835073, Cout=0
#
# Test Case 2: A=fffff, B=00001, Cin=1, Sum=00001, Cout=1
# Test Case 2: A=1048575, B=      1, Cin=1, Sum=      1, Cout=1
# ** Note: $stop    : C:/Users/nabeehdaouk/source/repos/ECSE318/Lab5/Problem2/NbitCSA_tb.v(37)
```

Results from N-bit Carry Save Adder TB (N=20-bits in testing example)

Comparing the adders:

Assuming each gate has a 1-unit delay, the 16-bit RCA has a delay of 16 units, while the 16-bit size-4 uniform CSA has a delay of 4 units. Therefore, the uniform carry select adder is significantly faster in this case due to its reduced delay, which is achieved by dividing the computation into smaller blocks and selecting the correct result using a multiplexer.

If we consider the individual logic gates required to achieve each carry out on a RCA, we see that there are 2 gates per full adder, meaning that a 16 bit RCA would actually take 32 units, while RCA would take 8.

Either way, we can see that with 16 bits, it is 4x faster (32/8=4), or SQRT(N) faster.

Problem 3:

```
mem[{val_10[13:0], 2'b11}]= 8'haa;
mem[{val_10[13:0], 2'b10}]= 8'hbb;
mem[{val_10[13:0], 2'b01}]= 8'hcc;
mem[{val_10[13:0], 2'b00}]= 8'hdd;
        //10: aabbccdd

mem[{val_20[13:0], 2'b11}]= 8'h11;
mem[{val_20[13:0], 2'b10}]= 8'h22;
mem[{val_20[13:0], 2'b01}]= 8'h33;
mem[{val_20[13:0], 2'b00}]= 8'h44;
        // 20: 11223344

mem[{val_31[13:0], 2'b11}]= 8'h99;
mem[{val_31[13:0], 2'b10}]= 8'h88;
mem[{val_31[13:0], 2'b01}]= 8'h11;
mem[{val_31[13:0], 2'b00}]= 8'h33;
        //31: 99881133

mem[{val_80[13:0], 2'b11}]= 8'hab;
mem[{val_80[13:0], 2'b10}]= 8'hcd;
mem[{val_80[13:0], 2'b01}]= 8'hef;
mem[{val_80[13:0], 2'b00}]= 8'h01;
        //80: abcdef01
```

```
mem[{val_51[13:0], 2'b11}]= 8'h12;
mem[{val_51[13:0], 2'b10}]= 8'h34;
mem[{val_51[13:0], 2'b01}]= 8'h56;
mem[{val_51[13:0], 2'b00}]= 8'h78;
          //51: 12345678

mem[{val_57[13:0], 2'b11}]= 8'h67;
mem[{val_57[13:0], 2'b10}]= 8'h89;
mem[{val_57[13:0], 2'b01}]= 8'hab;
mem[{val_57[13:0], 2'b00}]= 8'hcd;
          //57: 6789abcd

mem[{val_58[13:0], 2'b11}]= 8'ha1;
mem[{val_58[13:0], 2'b10}]= 8'hb2;
mem[{val_58[13:0], 2'b01}]= 8'hc3;
mem[{val_58[13:0], 2'b00}]= 8'hd4;
        //58: a1b2c3d4

mem[{val_59[13:0], 2'b11}]= 8'h9a;
mem[{val_59[13:0], 2'b10}]= 8'h8b;
mem[{val_59[13:0], 2'b01}]= 8'h1c;
mem[{val_59[13:0], 2'b00}]= 8'h3d;
       //59: 9a8b1c3d
```

Data stored in mem for reference, this is what will be pulled by cache and stored in cache. This is what we expect to see for each address when requesting data from cache/mem. Memory is initialized with values for testing.

```
TESTING CACHE DESIGN...
Memory has been innitialized for this test via initial begin for testing purposes
------------------------------------------------------------
Address Request: 10          Timestamp:                1100
Data Out: aabbccdd          Timestamp:                1700
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 20          Timestamp:                1800
Data Out: 11223344          Timestamp:                2400
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 31          Timestamp:                2500
Data Out: 99881133          Timestamp:                3100
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 80          Timestamp:                3200
Data Out: abcdef01          Timestamp:                3800
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 51          Timestamp:                3900
Data Out: 12345678          Timestamp:                4500
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 57          Timestamp:                4600
Data Out: 6789abcd          Timestamp:                5200
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 58          Timestamp:                5300
Data Out: a1b2c3d4          Timestamp:                5900
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 59          Timestamp:                6000
Data Out: a1b2c3d4,          Timestamp:                6600
CACHE MISS: note, delay of 600 as data had to be fetched from memmory

Address Request: 57          Timestamp:                6700
Data Out: 6789abcd          Timestamp:                6800
NOTE: DELAY OF ONLY 100 AS CACHE HIT!!!!!!!!!

Address Request: 51          Timestamp:                6900
Data Out: 12345678,          Timestamp:                7000
NOTE: DELAY OF ONLY 100 AS CACHE HIT!!!!!!!!!
```

Testbench results of cache read operation. Shows cache hits and cache misses with timing highlighted. Timing shows that access is MUCH faster when there is a cache hit as opposed to a cache miss.

```
------------------------------------------------------------
TESTING WRITE
------------------------------------------------------------
Address WRITE: 99          Timestamp:                7100
Data In: a123b456,          Timestamp:                7900

Address Request: 99          Timestamp:                8000
Data Out: a123b456,          Timestamp:                8100
NOTE: DELAY OF ONLY 100 AS CACHE HIT!!!!!!!!! as we just wrote there
SINCE WRRITE OPPERARTION WRITES TO CACHE AS WELL, also stored in memory
```

Testing write operation to memory through cache. Results written to both memory and cache.

Note: Cache is designed in VHDL, but testing is in verilog (aka, TB and Memory for testing). This was approved by Professor Saab.