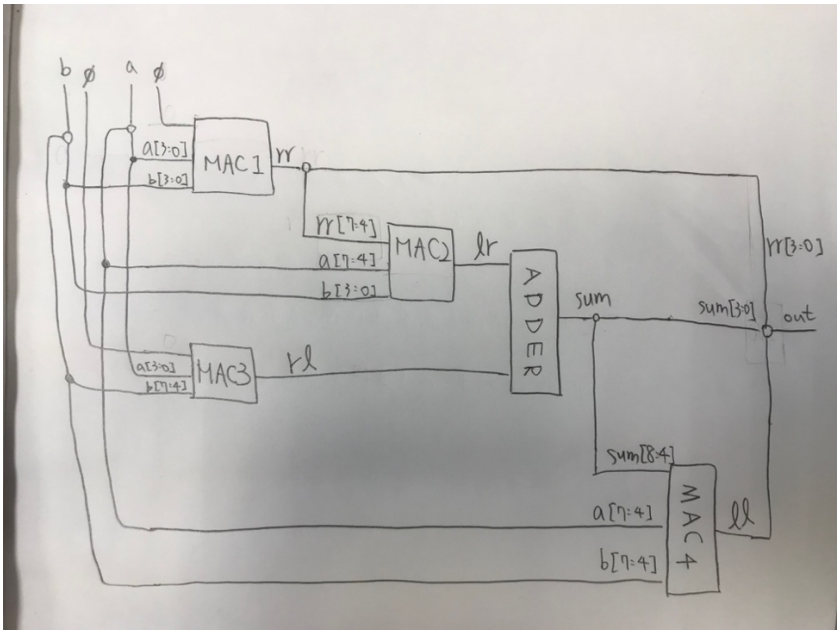
hw04_report_106062202

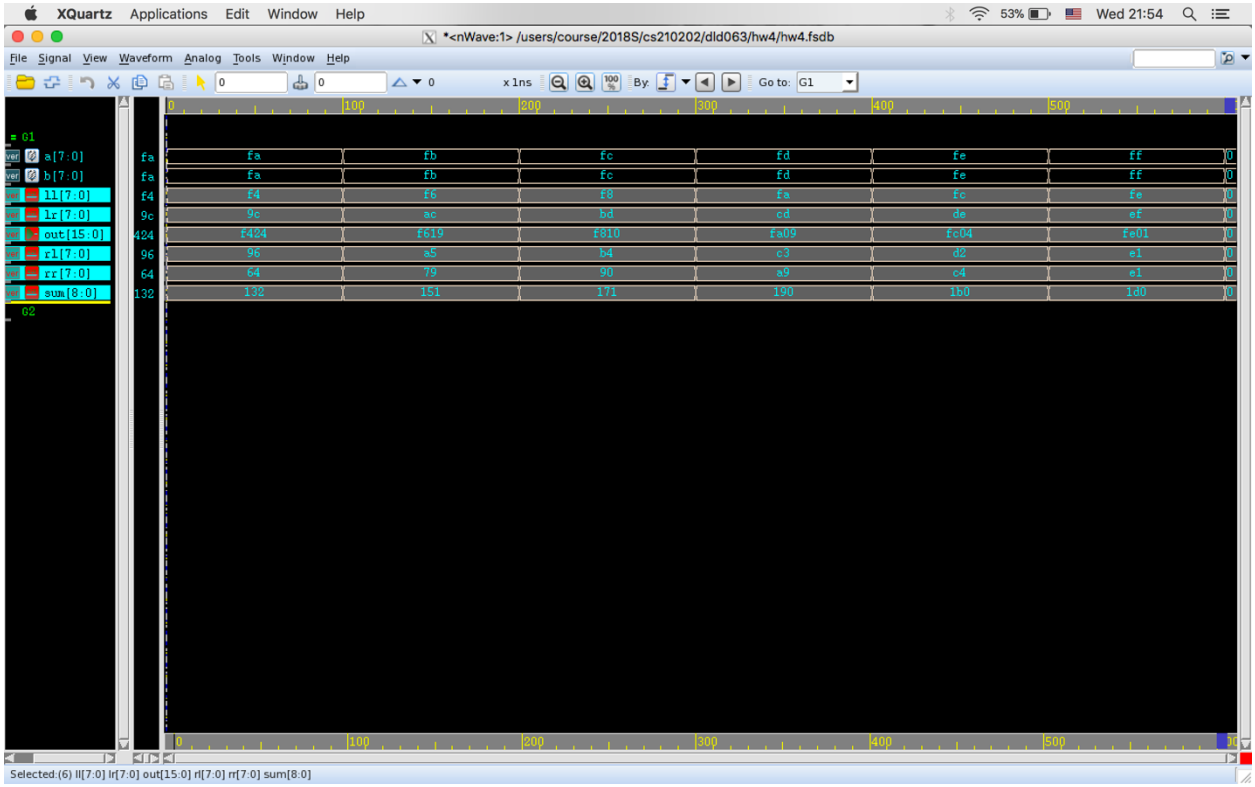
1. I verified the design by writing a simple test bench:

```
1 `timescale 1ns / 100ps
2 module test_mac;
3
4 reg [3:0] a, b;
5 reg [4:0] c;
6 wire [7:0] out;
7
8 mac mac1(.out(out), .a(a), .b(b), .c(c));
9
10 initial begin
11     a = 15;
12     b = 15;
13     c = 10;
14     #100
15     $display("%d\n", out);
16 end
17
18 endmodule
~
~
~
~
~
~
~
~
~
~
                                                    13,2-5        All
```

2. 16 bits. Yes, an adder will help. I designed an adder which adds two 8-bit number. I used one adder to minimize the number of MACs used.
3. 

Discussion: I believe to implement an 8-bit multiplier, a the I/O bit widths of the given MAC block is sufficient. Two 8-bit multiplication can be broken down into four 4-bit multiplication, by the distribution law. The max value of a 4-bit multiplication can be stored in an 8-bit register, and the max value of ‘c’ is the first five bits of 15 * 15 + 15 * 15, which equals to 28. 15 * 15 * 2 + 28 = 253 < 256, thus can be stored in an 8-bit register.

4.



The six test patterns I tested are numbers starting from 250 * 250 to 255 * 255. Inputs are ‘a’ and ‘b’. ‘ll’, ‘lr’, ‘rl’, and ’rr’ are the partial products. The wire ‘sum’ holds an intermediate sum of ‘rl’ and ‘rl’. The output is stored in ‘out’.

5.

```
 1 `timescale 1ns / 100ps
 2 module test;
 3
 4  parameter pattern = (1 << 16); // 2^16
 5  reg [7:0] a, b;
 6  integer i, error;
 7
 8  wire [15:0] out;
 9
10  multiplier multi(.out(out), .a(a), .b(b));
11
12  integer golden, count;
13
14  initial begin
15     $fsdbDumpfile("hw4.fsdb");
16     $fsdbDumpvars;
17  end
18
19
20  initial begin
21     a = 0;
22     b = 0;
23     error = 0;
24     count = 0;
25     for (i = 0; i < pattern; i = i + 1) begin
26         a = i[15:8];
27         b = i[7:0];
28         golden = a * b;
                                                                                  3,1-4         Top
29         #10
30         $display("<%5d> a=%d | b=%d | out=%d (%b) [%d (%b)]", i, a, b, out, out, golden, golden);
31
32         if (golden != out) begin
33             count = count + 1;
34             $display("Mismatched!");
35         end
36     end
37
38     if (count ==  0) $display("\n<<< PERFECT!! >>>\n");
39     else $display("%d ERRORS\n", count);
40
41     #100 $finish;
42  end
43 endmodule
44
```