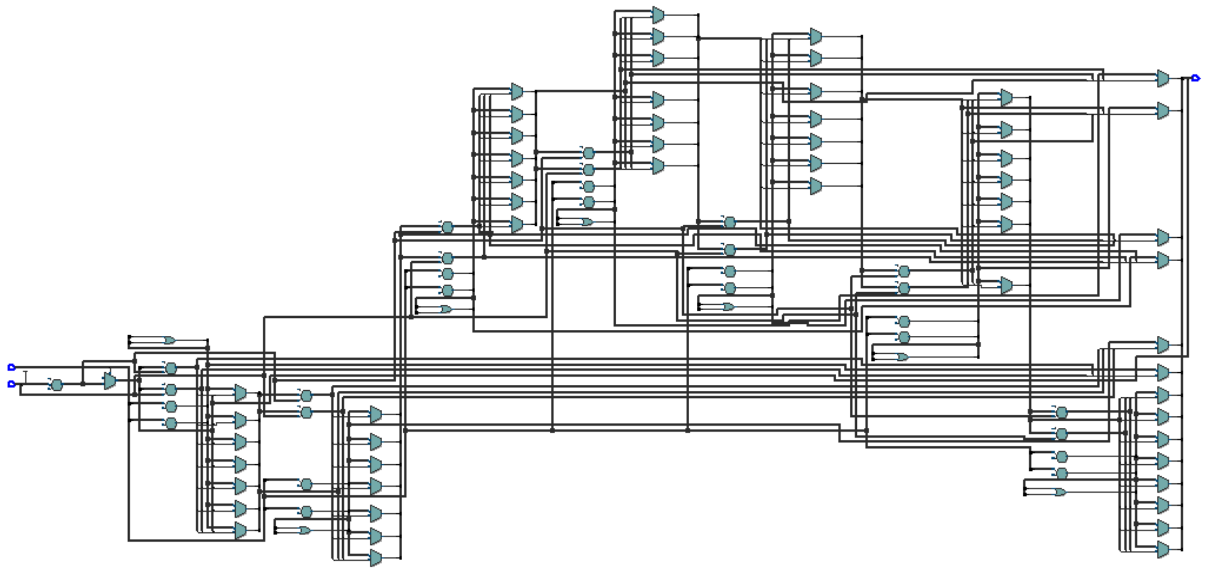
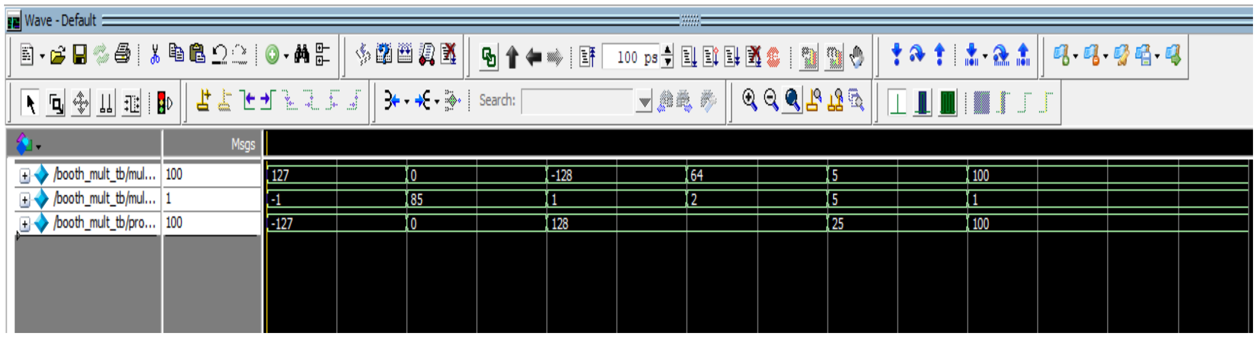
5. 8-bits Booth Multiplier Circuit

Design:



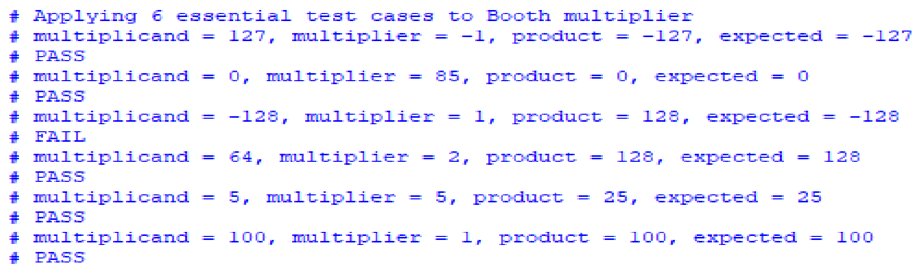
- Designed a “booth_mult” module that implements an 8×8 signed Booth multiplication algo using a combinational loop to simulate arithmetic shifts.
- Actually, the multiplier design is entirely combinational logic (without FSM) having no clock signal.
- First, It initializes partial products and conditionally adds or subtracts based on the multiplier bits using Booth’s encoding.
- Finally, the result is extracted from the shifted partial product and assigned to the 16-bit output.

Verificiation:



Wave - Default

```
                     Msgs
/booth_mult_tb/mul...  100   127   0    -128  64  5   100
/booth_mult_tb/mul...  1     -1    85   1     2   5   1
/booth_mult_tb/pro...  100   -127  0    128       25  100
```



```
# Applying 6 essential test cases to Booth multiplier
# multiplicand = 127, multiplier = -1, product = -127, expected = -127
# PASS
# multiplicand = 0, multiplier = 85, product = 0, expected = 0
# PASS
# multiplicand = -128, multiplier = 1, product = 128, expected = -128
# FAIL
# multiplicand = 64, multiplier = 2, product = 128, expected = 128
# PASS
# multiplicand = 5, multiplier = 5, product = 25, expected = 25
# PASS
# multiplicand = 100, multiplier = 1, product = 100, expected = 100
# PASS
```

- ❖ Designed the testbench that verified the ‘booth_mult’ module by applying six signed input combinations to test essential case scenarios.
- ❖ Inside the test, there is a custom checker function block to compare the module’s output against the expected result from built-in multiplication operation (*).
- ❖ Each case is delayed and checked sequentially, with pass/fail feedback message printed for validation for resulr.