Testbenching Report for square

Table of Contents

Testbench Summary

Component	Total Tests	Passed	Failed
square_	4	4	0

Testbench for square with parameter(s)

Total tests: 4

Passed tests: 4

Failed tests: 0

Test Case	Input Ai	Input Bi	Output G (Actual)	Expected G	Output P (Actual)	Expected P	Status
3	0 (bin) / 0 (dec)	0 (bin) / 0 (dec)	0 (bin) / 0 (dec)	0 (dec)	0 (bin) / 0 (dec)	0 (dec)	Passed
2	0 (bin) / 0 (dec)	1 (bin) / 1 (dec)	0 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	1 (dec)	Passed
1	1 (bin) / 1 (dec)	0 (bin) / 0 (dec)	0 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	1 (dec)	Passed
0	1 (bin) / 1 (dec)	1 (bin) / 1 (dec)	1 (bin) / 1 (dec)	1 (dec)	0 (bin) / 0 (dec)	0 (dec)	Passed

Rule: SquareRule

Input Variables: Ai, Bi

Output Variables: G, P

Bit Width: 4

Pattern: StringMatchPattern

```
            def matches(self, filename):
                #print(self.pattern, filename)
                return self.pattern == filename
```

Generate expected values function:

```
def generate_expected(self, test_case):
    a_bits = [(test_case["Ai"] >> i) & 1 for i in range(self.bit_width)]
    b_bits = [(test_case["Bi"] >> i) & 1 for i in range(self.bit_width)]

    G = [a_bits[i] & b_bits[i] for i in range(self.bit_width)]
    P = [a_bits[i] ^ b_bits[i] for i in range(self.bit_width)]

    G_int = sum(G[i] << i for i in range(self.bit_width))
    P_int = sum(P[i] << i for i in range(self.bit_width))

    return {
        "G": G_int,
        "P": P_int
    }
```