Testbenching Report for restoring_divider

Table of Contents

Testbench Summary

Component	Total Tests	Passed	Failed
restoring_divider_N2	16	7	9
restoring_divider_N3	64	37	27
restoring_divider_N4	256	136	120
restoring_divider_N5	1024	484	540
restoring_divider_N6	2766	1373	1393
restoring_divider_N7	2766	1382	1384
restoring_divider_N8	2766	1406	1360

Testbench for restoring_divider with parameter(s) N2

Total tests: 16

Passed tests: 7

Failed tests: 9

est Case	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected valid	S
1	10 (bin) / 2 (dec)	01 (bin) / 1 (dec)	10 (bin) / 2 (dec)	-2 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
8	00 (bin) / 0 (dec)	11 (bin) / 3 (dec)	10 (bin) / 2 (dec)	0 (dec)	10 (bin) / 2 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
0	11 (bin) / 3 (dec)	00 (bin) / 0 (dec)	10 (bin) / 2 (dec)	-1 (dec)	11 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
6	00 (bin) / 0 (dec)	11 (bin) / 3 (dec)	10 (bin) / 2 (dec)	0 (dec)	10 (bin) / 2 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
9	01 (bin) / 1 (dec)	11 (bin) / 3 (dec)	11 (bin) / 3 (dec)	0 (dec)	00 (bin) / 0 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
12	01 (bin) / 1 (dec)	00 (bin) / 0 (dec)	11 (bin) / 3 (dec)	-1 (dec)	01 (bin) / 1 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
10	11 (bin) / 3 (dec)	00 (bin) / 0 (dec)	10 (bin) / 2 (dec)	-1 (dec)	11 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
4	00 (bin) / 0 (dec)	11 (bin) / 3 (dec)	10 (bin) / 2 (dec)	0 (dec)	10 (bin) / 2 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
14	10 (bin) / 2 (dec)	01 (bin) / 1 (dec)	10 (bin) / 2 (dec)	-2 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	I
3	11 (bin) / 3 (dec)	11 (bin) / 3 (dec)	01 (bin) / 1 (dec)	1 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P
5	10 (bin) / 2 (dec)	10 (bin) / 2 (dec)	01 (bin) / 1 (dec)	1 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P
15	11 (bin) / 3 (dec)	11 (bin) / 3 (dec)	01 (bin) / 1 (dec)	1 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P
2	11 (bin) / 3 (dec)	10 (bin) / 2 (dec)	01 (bin) / 1 (dec)	1 (dec)	01 (bin) / 1 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P
11	00 (bin) / 0 (dec)	01 (bin) / 1 (dec)	00 (bin) / 0 (dec)	0 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P
7	10 (bin) / 2 (dec)	11 (bin) / 3 (dec)	00 (bin) / 0 (dec)	0 (dec)	10 (bin) / 2 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P
13	00 (bin) / 0 (dec)	10 (bin) / 2 (dec)	00 (bin) / 0 (dec)	0 (dec)	00 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)	P

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```

Testbench for restoring_divider with parameter(s) N3

Total tests: 64

Passed tests: 37

Failed tests: 27

st Case	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected valid
25	111 (bin) / 7 (dec)	000 (bin) / 0 (dec)	110 (bin) / 6 (dec)	-1 (dec)	111 (bin) / 7 (dec)	7 (dec)	1 (bin) / 1 (dec)	N/A (dec)
13	001 (bin) / 1 (dec)	110 (bin) / 6 (dec)	101 (bin) / 5 (dec)	0 (dec)	011 (bin) / 3 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
59	001 (bin) / 1 (dec)	110 (bin) / 6 (dec)	101 (bin) / 5 (dec)	0 (dec)	011 (bin) / 3 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
39	011 (bin) / 3 (dec)	000 (bin) / 0 (dec)	111 (bin) / 7 (dec)	-1 (dec)	011 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
5	011 (bin) / 3 (dec)	000 (bin) / 0 (dec)	111 (bin) / 7 (dec)	-1 (dec)	011 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
52	101 (bin) / 5 (dec)	111 (bin) / 7 (dec)	101 (bin) / 5 (dec)	0 (dec)	010 (bin) / 2 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
30	000 (bin) / 0 (dec)	111 (bin) / 7 (dec)	110 (bin) / 6 (dec)	0 (dec)	110 (bin) / 6 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
44	000 (bin) / 0 (dec)	110 (bin) / 6 (dec)	101 (bin) / 5 (dec)	0 (dec)	010 (bin) / 2 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
0	101 (bin) / 5 (dec)	110 (bin) / 6 (dec)	111 (bin) / 7 (dec)	0 (dec)	011 (bin) / 3 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
6	000 (bin) / 0 (dec)	000 (bin) / 0 (dec)	111 (bin) / 7 (dec)	-1 (dec)	000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
24	001 (bin) / 1 (dec)	111 (bin) / 7 (dec)	111 (bin) / 7 (dec)	0 (dec)	000 (bin) / 0 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
15	001 (bin) / 1 (dec)	101 (bin) / 5 (dec)	110 (bin) / 6 (dec)	0 (dec)	011 (bin) / 3 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
49	000 (bin) / 0 (dec)	000 (bin) / 0 (dec)	111 (bin) / 7 (dec)	-1 (dec)	000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
17	010 (bin) / 2 (dec)	111 (bin) / 7 (dec)	100 (bin) / 4 (dec)	0 (dec)	110 (bin) / 6 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
9	011 (bin) / 3 (dec)	101 (bin) / 5 (dec)	111 (bin) / 7 (dec)	0 (dec)	000 (bin) / 0 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
51	011 (bin) / 3 (dec)	110 (bin) / 6 (dec)	100 (bin) / 4 (dec)	0 (dec)	011 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
4	010 (bin) / 2 (dec)	101 (bin) / 5 (dec)	110 (bin) / 6 (dec)	0 (dec)	100 (bin) / 4 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
2	010 (bin) / 2 (dec)	111 (bin) / 7 (dec)	100 (bin) / 4 (dec)	0 (dec)	110 (bin) / 6 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
22	101 (bin) / 5 (dec)	111 (bin) / 7 (dec)	101 (bin) / 5 (dec)	0 (dec)	010 (bin) / 2 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
57	001 (bin) / 1 (dec)	111 (bin) / 7 (dec)	111 (bin) / 7 (dec)	0 (dec)	000 (bin) / 0 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
41	101 (bin) / 5 (dec)	011 (bin) / 3 (dec)	001 (bin) / 1 (dec)	1 (dec)	010 (bin) / 2 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
38	111 (bin) / 7 (dec)	100 (bin) / 4 (dec)	001 (bin) / 1 (dec)	1 (dec)	011 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
56	100 (bin) / 4 (dec)	011 (bin) / 3 (dec)	001 (bin) / 1 (dec)	1 (dec)	001 (bin) / 1 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)

st Case	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected valid
18	000 (bin) / 0 (dec)	001 (bin) / 1 (dec)	000 (bin) / 0 (dec)	0 (dec)	000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
7	001 (bin) / 1 (dec)	010 (bin) / 2 (dec)	000 (bin) / 0 (dec)	0 (dec)	001 (bin) / 1 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```

Testbench for restoring_divider with parameter(s) N4

Total tests: 256

Passed tests: 136

Failed tests: 120

Case	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected valid
07	1110 (bin) / 14 (dec)	1110 (bin) / 14 (dec)	1011 (bin) / 11 (dec)	1 (dec)	0100 (bin) / 4 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
25	0101 (bin) / 5 (dec)	0000 (bin) / 0 (dec)	1111 (bin) / 15 (dec)	-1 (dec)	0101 (bin) / 5 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
38	1100 (bin) / 12 (dec)	1010 (bin) / 10 (dec)	1111 (bin) / 15 (dec)	1 (dec)	0110 (bin) / 6 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
21	1111 (bin) / 15 (dec)	1010 (bin) / 10 (dec)	1110 (bin) / 14 (dec)	1 (dec)	0011 (bin) / 3 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
52	1111 (bin) / 15 (dec)	1110 (bin) / 14 (dec)	1011 (bin) / 11 (dec)	1 (dec)	0101 (bin) / 5 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
8	1110 (bin) / 14 (dec)	1011 (bin) / 11 (dec)	1100 (bin) / 12 (dec)	1 (dec)	1010 (bin) / 10 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
13	0000 (bin) / 0 (dec)	1111 (bin) / 15 (dec)	1110 (bin) / 14 (dec)	0 (dec)	1110 (bin) / 14 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
13	0101 (bin) / 5 (dec)	1111 (bin) / 15 (dec)	1101 (bin) / 13 (dec)	0 (dec)	0010 (bin) / 2 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
3	1011 (bin) / 11 (dec)	1111 (bin) / 15 (dec)	1100 (bin) / 12 (dec)	0 (dec)	0111 (bin) / 7 (dec)	11 (dec)	1 (bin) / 1 (dec)	N/A (dec)
1	0100 (bin) / 4 (dec)	0000 (bin) / 0 (dec)	1111 (bin) / 15 (dec)	-1 (dec)	0100 (bin) / 4 (dec)	4 (dec)	1 (bin) / 1 (dec)	N/A (dec)
96	1011 (bin) / 11 (dec)	1100 (bin) / 12 (dec)	1000 (bin) / 8 (dec)	0 (dec)	1011 (bin) / 11 (dec)	11 (dec)	1 (bin) / 1 (dec)	N/A (dec)
0	0111 (bin) / 7 (dec)	1010 (bin) / 10 (dec)	1101 (bin) / 13 (dec)	0 (dec)	0101 (bin) / 5 (dec)	7 (dec)	1 (bin) / 1 (dec)	N/A (dec)
9	1001 (bin) / 9 (dec)	1100 (bin) / 12 (dec)	1000 (bin) / 8 (dec)	0 (dec)	1001 (bin) / 9 (dec)	9 (dec)	1 (bin) / 1 (dec)	N/A (dec)
8	1110 (bin) / 14 (dec)	0000 (bin) / 0 (dec)	1110 (bin) / 14 (dec)	-1 (dec)	1110 (bin) / 14 (dec)	14 (dec)	1 (bin) / 1 (dec)	N/A (dec)
1	1101 (bin) / 13 (dec)	0001 (bin) / 1 (dec)	1101 (bin) / 13 (dec)	-3 (dec)	0000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
4	1111 (bin) / 15 (dec)	1100 (bin) / 12 (dec)	1001 (bin) / 9 (dec)	1 (dec)	0011 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
0	0110 (bin) / 6 (dec)	1010 (bin) / 10 (dec)	1101 (bin) / 13 (dec)	0 (dec)	0100 (bin) / 4 (dec)	6 (dec)	1 (bin) / 1 (dec)	N/A (dec)
5	1010 (bin) / 10 (dec)	0000 (bin) / 0 (dec)	1110 (bin) / 14 (dec)	-1 (dec)	1010 (bin) / 10 (dec)	10 (dec)	1 (bin) / 1 (dec)	N/A (dec)
00	1101 (bin) / 13 (dec)	1011 (bin) / 11 (dec)	1100 (bin) / 12 (dec)	1 (dec)	1001 (bin) / 9 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
04	0011 (bin) / 3 (dec)	1001 (bin) / 9 (dec)	1110 (bin) / 14 (dec)	0 (dec)	0101 (bin) / 5 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
8	0110 (bin) / 6 (dec)	0101 (bin) / 5 (dec)	0001 (bin) / 1 (dec)	1 (dec)	0001 (bin) / 1 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
22	0011 (bin) / 3 (dec)	0100 (bin) / 4 (dec)	0000 (bin) / 0 (dec)	0 (dec)	0011 (bin) / 3 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec)
5	0000 (bin) / 0 (dec)	0001 (bin) / 1 (dec)	0000 (bin) / 0 (dec)	0 (dec)	0000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)

Case	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected valid
54	1110 (bin) / 14 (dec)	1001 (bin) / 9 (dec)	0001 (bin) / 1 (dec)	1 (dec)	0101 (bin) / 5 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
2	1111 (bin) / 15 (dec)	0101 (bin) / 5 (dec)	0011 (bin) / 3 (dec)	3 (dec)	0000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```

Testbench for restoring_divider with parameter(s) N5

Total tests: 1024

Passed tests: 484

Failed tests: 540

ase	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected vali
	11011 (bin) / 27 (dec)	11111 (bin) / 31 (dec)	11100 (bin) / 28 (dec)	0 (dec)	10111 (bin) / 23 (dec)	27 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01100 (bin) / 12 (dec)	10101 (bin) / 21 (dec)	11000 (bin) / 24 (dec)	0 (dec)	10100 (bin) / 20 (dec)	12 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	10111 (bin) / 23 (dec)	10110 (bin) / 22 (dec)	11111 (bin) / 31 (dec)	1 (dec)	01101 (bin) / 13 (dec)	1 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01010 (bin) / 10 (dec)	11100 (bin) / 28 (dec)	11001 (bin) / 25 (dec)	0 (dec)	01110 (bin) / 14 (dec)	10 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	00110 (bin) / 6 (dec)	11101 (bin) / 29 (dec)	11010 (bin) / 26 (dec)	0 (dec)	10100 (bin) / 20 (dec)	6 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	11101 (bin) / 29 (dec)	11000 (bin) / 24 (dec)	10110 (bin) / 22 (dec)	1 (dec)	01101 (bin) / 13 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	10010 (bin) / 18 (dec)	11101 (bin) / 29 (dec)	11000 (bin) / 24 (dec)	0 (dec)	11010 (bin) / 26 (dec)	18 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	10011 (bin) / 19 (dec)	10100 (bin) / 20 (dec)	11010 (bin) / 26 (dec)	0 (dec)	01011 (bin) / 11 (dec)	19 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	11111 (bin) / 31 (dec)	11101 (bin) / 29 (dec)	11001 (bin) / 25 (dec)	1 (dec)	01010 (bin) / 10 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	11111 (bin) / 31 (dec)	10111 (bin) / 23 (dec)	10011 (bin) / 19 (dec)	1 (dec)	01010 (bin) / 10 (dec)	8 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01101 (bin) / 13 (dec)	10100 (bin) / 20 (dec)	11010 (bin) / 26 (dec)	0 (dec)	00101 (bin) / 5 (dec)	13 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01010 (bin) / 10 (dec)	10001 (bin) / 17 (dec)	11110 (bin) / 30 (dec)	0 (dec)	01100 (bin) / 12 (dec)	10 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	11000 (bin) / 24 (dec)	00001 (bin) / 1 (dec)	11000 (bin) / 24 (dec)	-8 (dec)	00000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	00111 (bin) / 7 (dec)	10001 (bin) / 17 (dec)	11110 (bin) / 30 (dec)	0 (dec)	01001 (bin) / 9 (dec)	7 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	11010 (bin) / 26 (dec)	10110 (bin) / 22 (dec)	11110 (bin) / 30 (dec)	1 (dec)	00110 (bin) / 6 (dec)	4 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	10101 (bin) / 21 (dec)	11001 (bin) / 25 (dec)	10101 (bin) / 21 (dec)	0 (dec)	01000 (bin) / 8 (dec)	21 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	10001 (bin) / 17 (dec)	10100 (bin) / 20 (dec)	11010 (bin) / 26 (dec)	0 (dec)	01001 (bin) / 9 (dec)	17 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	11110 (bin) / 30 (dec)	11010 (bin) / 26 (dec)	10100 (bin) / 20 (dec)	1 (dec)	10110 (bin) / 22 (dec)	4 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01000 (bin) / 8 (dec)	10111 (bin) / 23 (dec)	10110 (bin) / 22 (dec)	0 (dec)	01110 (bin) / 14 (dec)	8 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01001 (bin) / 9 (dec)	11001 (bin) / 25 (dec)	10100 (bin) / 20 (dec)	0 (dec)	10101 (bin) / 21 (dec)	9 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	10001 (bin) / 17 (dec)	01111 (bin) / 15 (dec)	00001 (bin) / 1 (dec)	1 (dec)	00010 (bin) / 2 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
5	00010 (bin) / 2 (dec)	01001 (bin) / 9 (dec)	00000 (bin) / 0 (dec)	0 (dec)	00010 (bin) / 2 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01111 (bin) / 15 (dec)	00001 (bin) / 1 (dec)	01111 (bin) / 15 (dec)	15 (dec)	00000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec)

ase	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected vali
	11011 (bin) / 27 (dec)	01011 (bin) / 11 (dec)	00010 (bin) / 2 (dec)	2 (dec)	00101 (bin) / 5 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A (dec)
	01110 (bin) / 14 (dec)	10000 (bin) / 16 (dec)	00000 (bin) / 0 (dec)	0 (dec)	01110 (bin) / 14 (dec)	14 (dec)	1 (bin) / 1 (dec)	N/A (dec)

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```

Testbench for restoring_divider with parameter(s) N6

Total tests: 2766

Passed tests: 1373

Failed tests: 1393

e	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected v
	100001 (bin) / 33 (dec)	110000 (bin) / 48 (dec)	101000 (bin) / 40 (dec)	0 (dec)	100001 (bin) / 33 (dec)	33 (dec)	1 (bin) / 1 (dec)	N/A (dec
	101110 (bin) / 46 (dec)	111010 (bin) / 58 (dec)	110100 (bin) / 52 (dec)	0 (dec)	100110 (bin) / 38 (dec)	46 (dec)	1 (bin) / 1 (dec)	N/A (dec
	011011 (bin) / 27 (dec)	111101 (bin) / 61 (dec)	111001 (bin) / 57 (dec)	0 (dec)	000110 (bin) / 6 (dec)	27 (dec)	1 (bin) / 1 (dec)	N/A (dec
	111110 (bin) / 62 (dec)	000001 (bin) / 1 (dec)	111110 (bin) / 62 (dec)	-2 (dec)	000000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec
	101011 (bin) / 43 (dec)	101111 (bin) / 47 (dec)	101111 (bin) / 47 (dec)	0 (dec)	001010 (bin) / 10 (dec)	43 (dec)	1 (bin) / 1 (dec)	N/A (dec
	100000 (bin) / 32 (dec)	000001 (bin) / 1 (dec)	100000 (bin) / 32 (dec)	-32 (dec)	000000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec
	110001 (bin) / 49 (dec)	111011 (bin) / 59 (dec)	110111 (bin) / 55 (dec)	0 (dec)	000100 (bin) / 4 (dec)	49 (dec)	1 (bin) / 1 (dec)	N/A (dec
	100011 (bin) / 35 (dec)	110101 (bin) / 53 (dec)	100010 (bin) / 34 (dec)	0 (dec)	011001 (bin) / 25 (dec)	35 (dec)	1 (bin) / 1 (dec)	N/A (dec
	000100 (bin) / 4 (dec)	100001 (bin) / 33 (dec)	111110 (bin) / 62 (dec)	0 (dec)	000110 (bin) / 6 (dec)	4 (dec)	1 (bin) / 1 (dec)	N/A (dec
	111110 (bin) / 62 (dec)	101011 (bin) / 43 (dec)	111100 (bin) / 60 (dec)	1 (dec)	101010 (bin) / 42 (dec)	19 (dec)	1 (bin) / 1 (dec)	N/A (dec
	111010 (bin) / 58 (dec)	000001 (bin) / 1 (dec)	111010 (bin) / 58 (dec)	-6 (dec)	000000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec
	001100 (bin) / 12 (dec)	111011 (bin) / 59 (dec)	110111 (bin) / 55 (dec)	0 (dec)	011111 (bin) / 31 (dec)	12 (dec)	1 (bin) / 1 (dec)	N/A (dec
	011010 (bin) / 26 (dec)	100100 (bin) / 36 (dec)	111001 (bin) / 57 (dec)	0 (dec)	010110 (bin) / 22 (dec)	26 (dec)	1 (bin) / 1 (dec)	N/A (dec
	000000 (bin) / 0 (dec)	111101 (bin) / 61 (dec)	111010 (bin) / 58 (dec)	0 (dec)	101110 (bin) / 46 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec
	000011 (bin) / 3 (dec)	100101 (bin) / 37 (dec)	110111 (bin) / 55 (dec)	0 (dec)	010000 (bin) / 16 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A (dec
	010110 (bin) / 22 (dec)	111001 (bin) / 57 (dec)	110000 (bin) / 48 (dec)	0 (dec)	100110 (bin) / 38 (dec)	22 (dec)	1 (bin) / 1 (dec)	N/A (dec
	001001 (bin) / 9 (dec)	111011 (bin) / 59 (dec)	110111 (bin) / 55 (dec)	0 (dec)	011100 (bin) / 28 (dec)	9 (dec)	1 (bin) / 1 (dec)	N/A (dec
	101101 (bin) / 45 (dec)	101111 (bin) / 47 (dec)	101111 (bin) / 47 (dec)	0 (dec)	001100 (bin) / 12 (dec)	45 (dec)	1 (bin) / 1 (dec)	N/A (dec
	101100 (bin) / 44 (dec)	101111 (bin) / 47 (dec)	101111 (bin) / 47 (dec)	0 (dec)	001011 (bin) / 11 (dec)	44 (dec)	1 (bin) / 1 (dec)	N/A (dec
	100100 (bin) / 36 (dec)	101101 (bin) / 45 (dec)	100010 (bin) / 34 (dec)	0 (dec)	101010 (bin) / 42 (dec)	36 (dec)	1 (bin) / 1 (dec)	N/A (dec
	000010 (bin) / 2 (dec)	010000 (bin) / 16 (dec)	000000 (bin) / 0 (dec)	0 (dec)	000010 (bin) / 2 (dec)	2 (dec)	1 (bin) / 1 (dec)	N/A (dec
	100011 (bin) / 35 (dec)	011101 (bin) / 29 (dec)	000001 (bin) / 1 (dec)	1 (dec)	000110 (bin) / 6 (dec)	6 (dec)	1 (bin) / 1 (dec)	N/A (dec
	110000 (bin) / 48 (dec)	001011 (bin) / 11 (dec)	000100 (bin) / 4 (dec)	4 (dec)	000100 (bin) / 4 (dec)	4 (dec)	1 (bin) / 1 (dec)	N/A (dec

e	Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expected v
	100000 (bin) / 32 (dec)	000010 (bin) / 2 (dec)	010000 (bin) / 16 (dec)	16 (dec)	000000 (bin) / 0 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A (dec
	001101 (bin) / 13 (dec)	010001 (bin) / 17 (dec)	000000 (bin) / 0 (dec)	0 (dec)	001101 (bin) / 13 (dec)	13 (dec)	1 (bin) / 1 (dec)	N/A (dec

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```

Testbench for restoring_divider with parameter(s) N7

Total tests: 2766

Passed tests: 1382

Failed tests: 1384

Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expect
0010011 (bin) / 19 (dec)	1001110 (bin) / 78 (dec)	1101001 (bin) / 105 (dec)	0 (dec)	0010101 (bin) / 21 (dec)	19 (dec)	1 (bin) / 1 (dec)	N/A
110011 (bin) / 115 (dec)	1101110 (bin) / 110 (dec)	1111011 (bin) / 123 (dec)	1 (dec)	0011001 (bin) / 25 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A
0010001 (bin) / 17 (dec)	1000101 (bin) / 69 (dec)	1110110 (bin) / 118 (dec)	0 (dec)	1000011 (bin) / 67 (dec)	17 (dec)	1 (bin) / 1 (dec)	N/A
1011100 (bin) / 92 (dec)	1111001 (bin) / 121 (dec)	1110011 (bin) / 115 (dec)	0 (dec)	0000001 (bin) / 1 (dec)	92 (dec)	1 (bin) / 1 (dec)	N/A
0011111 (bin) / 31 (dec)	1001001 (bin) / 73 (dec)	1111110 (bin) / 126 (dec)	0 (dec)	0110001 (bin) / 49 (dec)	31 (dec)	1 (bin) / 1 (dec)	N/A
0101010 (bin) / 42 (dec)	1111010 (bin) / 122 (dec)	1110100 (bin) / 116 (dec)	0 (dec)	1100010 (bin) / 98 (dec)	42 (dec)	1 (bin) / 1 (dec)	N/A
0001010 (bin) / 10 (dec)	1111110 (bin) / 126 (dec)	1111000 (bin) / 120 (dec)	0 (dec)	1111010 (bin) / 122 (dec)	10 (dec)	1 (bin) / 1 (dec)	N/A
1010100 (bin) / 84 (dec)	1110001 (bin) / 113 (dec)	1100110 (bin) / 102 (dec)	0 (dec)	1001110 (bin) / 78 (dec)	84 (dec)	1 (bin) / 1 (dec)	N/A
0011111 (bin) / 31 (dec)	1011100 (bin) / 92 (dec)	1011001 (bin) / 89 (dec)	0 (dec)	0100011 (bin) / 35 (dec)	31 (dec)	1 (bin) / 1 (dec)	N/A
0010001 (bin) / 17 (dec)	1101000 (bin) / 104 (dec)	1000101 (bin) / 69 (dec)	0 (dec)	0001001 (bin) / 9 (dec)	17 (dec)	1 (bin) / 1 (dec)	N/A
100101 (bin) / 101 (dec)	1111111 (bin) / 127 (dec)	1111100 (bin) / 124 (dec)	0 (dec)	1100001 (bin) / 97 (dec)	101 (dec)	1 (bin) / 1 (dec)	N/A
101001 (bin) / 105 (dec)	1011001 (bin) / 89 (dec)	1000111 (bin) / 71 (dec)	1 (dec)	0111010 (bin) / 58 (dec)	16 (dec)	1 (bin) / 1 (dec)	N/A
0000000 (bin) / 0 (dec)	1001011 (bin) / 75 (dec)	1101101 (bin) / 109 (dec)	0 (dec)	0010001 (bin) / 17 (dec)	0 (dec)	1 (bin) / 1 (dec)	N/A
0111001 (bin) / 57 (dec)	1111001 (bin) / 121 (dec)	1110010 (bin) / 114 (dec)	0 (dec)	1010111 (bin) / 87 (dec)	57 (dec)	1 (bin) / 1 (dec)	N/A
0110100 (bin) / 52 (dec)	1101101 (bin) / 109 (dec)	1111010 (bin) / 122 (dec)	0 (dec)	1000010 (bin) / 66 (dec)	52 (dec)	1 (bin) / 1 (dec)	N/A
0000011 (bin) / 3 (dec)	1111111 (bin) / 127 (dec)	1111101 (bin) / 125 (dec)	0 (dec)	0000000 (bin) / 0 (dec)	3 (dec)	1 (bin) / 1 (dec)	N/A
1100000 (bin) / 96 (dec)	1010111 (bin) / 87 (dec)	1000000 (bin) / 64 (dec)	1 (dec)	0100000 (bin) / 32 (dec)	9 (dec)	1 (bin) / 1 (dec)	N/A
0011000 (bin) / 24 (dec)	1111001 (bin) / 121 (dec)	1110011 (bin) / 115 (dec)	0 (dec)	0111101 (bin) / 61 (dec)	24 (dec)	1 (bin) / 1 (dec)	N/A
0101001 (bin) / 41 (dec)	1011000 (bin) / 88 (dec)	1000110 (bin) / 70 (dec)	0 (dec)	0011001 (bin) / 25 (dec)	41 (dec)	1 (bin) / 1 (dec)	N/A
1010111 (bin) / 87 (dec)	1101110 (bin) / 110 (dec)	1111000 (bin) / 120 (dec)	0 (dec)	1000111 (bin) / 71 (dec)	87 (dec)	1 (bin) / 1 (dec)	N/A
110011 (bin) / 115 (dec)	0111100 (bin) / 60 (dec)	0000001 (bin) / 1 (dec)	1 (dec)	0110111 (bin) / 55 (dec)	55 (dec)	1 (bin) / 1 (dec)	N/A
0111110 (bin) / 62 (dec)	0101011 (bin) / 43 (dec)	0000001 (bin) / 1 (dec)	1 (dec)	0010011 (bin) / 19 (dec)	19 (dec)	1 (bin) / 1 (dec)	N/A
0011101 (bin) / 29 (dec)	0011000 (bin) / 24 (dec)	0000001 (bin) / 1 (dec)	1 (dec)	0000101 (bin) / 5 (dec)	5 (dec)	1 (bin) / 1 (dec)	N/A

Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expect
101011 (bin) / 107 (dec)	0111011 (bin) / 59 (dec)	0000001 (bin) / 1 (dec)	1 (dec)	0110000 (bin) / 48 (dec)	48 (dec)	1 (bin) / 1 (dec)	N/A
0010101 (bin) / 21 (dec)	0100100 (bin) / 36 (dec)	0000000 (bin) / 0 (dec)	0 (dec)	0010101 (bin) / 21 (dec)	21 (dec)	1 (bin) / 1 (dec)	N/A

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```

Testbench for restoring_divider with parameter(s) N8

Total tests: 2766

Passed tests: 1406

Failed tests: 1360

Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expe
001101 (bin) / 13 (dec)	10111110 (bin) / 190 (dec)	10101111 (bin) / 175 (dec)	0 (dec)	00101011 (bin) / 43 (dec)	13 (dec)	1 (bin) / 1 (dec)	N
01001 (bin) / 105 (dec)	10100011 (bin) / 163 (dec)	11001001 (bin) / 201 (dec)	0 (dec)	01101110 (bin) / 110 (dec)	105 (dec)	1 (bin) / 1 (dec)	N
00111 (bin) / 167 (dec)	11000111 (bin) / 199 (dec)	10111011 (bin) / 187 (dec)	0 (dec)	01001010 (bin) / 74 (dec)	167 (dec)	1 (bin) / 1 (dec)	N
10000 (bin) / 112 (dec)	10100100 (bin) / 164 (dec)	11001110 (bin) / 206 (dec)	0 (dec)	01111000 (bin) / 120 (dec)	112 (dec)	1 (bin) / 1 (dec)	N
01000 (bin) / 136 (dec)	11100100 (bin) / 228 (dec)	11011110 (bin) / 222 (dec)	0 (dec)	11010000 (bin) / 208 (dec)	136 (dec)	1 (bin) / 1 (dec)	N
00110 (bin) / 230 (dec)	11101110 (bin) / 238 (dec)	11000010 (bin) / 194 (dec)	0 (dec)	10001010 (bin) / 138 (dec)	230 (dec)	1 (bin) / 1 (dec)	N
01111 (bin) / 239 (dec)	10100000 (bin) / 160 (dec)	11001110 (bin) / 206 (dec)	1 (dec)	00101111 (bin) / 47 (dec)	79 (dec)	1 (bin) / 1 (dec)	N
01001 (bin) / 233 (dec)	11010001 (bin) / 209 (dec)	10001011 (bin) / 139 (dec)	1 (dec)	01101110 (bin) / 110 (dec)	24 (dec)	1 (bin) / 1 (dec)	N
000101 (bin) / 69 (dec)	11100111 (bin) / 231 (dec)	11010101 (bin) / 213 (dec)	0 (dec)	00010010 (bin) / 18 (dec)	69 (dec)	1 (bin) / 1 (dec)	N
11000 (bin) / 248 (dec)	11101100 (bin) / 236 (dec)	11011011 (bin) / 219 (dec)	1 (dec)	00010100 (bin) / 20 (dec)	12 (dec)	1 (bin) / 1 (dec)	N
10000 (bin) / 144 (dec)	11000101 (bin) / 197 (dec)	10100000 (bin) / 160 (dec)	0 (dec)	01110000 (bin) / 112 (dec)	144 (dec)	1 (bin) / 1 (dec)	N
10100 (bin) / 180 (dec)	11110110 (bin) / 246 (dec)	11101111 (bin) / 239 (dec)	0 (dec)	00001010 (bin) / 10 (dec)	180 (dec)	1 (bin) / 1 (dec)	N
000111 (bin) / 7 (dec)	11111110 (bin) / 254 (dec)	11111111 (bin) / 255 (dec)	0 (dec)	00000101 (bin) / 5 (dec)	7 (dec)	1 (bin) / 1 (dec)	N
10110 (bin) / 246 (dec)	10000011 (bin) / 131 (dec)	11111111 (bin) / 255 (dec)	1 (dec)	01111001 (bin) / 121 (dec)	115 (dec)	1 (bin) / 1 (dec)	N
01000 (bin) / 168 (dec)	10010100 (bin) / 148 (dec)	11000010 (bin) / 194 (dec)	1 (dec)	10000000 (bin) / 128 (dec)	20 (dec)	1 (bin) / 1 (dec)	N
010101 (bin) / 85 (dec)	11001001 (bin) / 201 (dec)	10110010 (bin) / 178 (dec)	0 (dec)	10010011 (bin) / 147 (dec)	85 (dec)	1 (bin) / 1 (dec)	N
00011 (bin) / 163 (dec)	10110100 (bin) / 180 (dec)	10001001 (bin) / 137 (dec)	0 (dec)	01001111 (bin) / 79 (dec)	163 (dec)	1 (bin) / 1 (dec)	N
10001 (bin) / 113 (dec)	10111010 (bin) / 186 (dec)	10111001 (bin) / 185 (dec)	0 (dec)	00000111 (bin) / 7 (dec)	113 (dec)	1 (bin) / 1 (dec)	N
00100 (bin) / 164 (dec)	10101000 (bin) / 168 (dec)	11110100 (bin) / 244 (dec)	0 (dec)	10000100 (bin) / 132 (dec)	164 (dec)	1 (bin) / 1 (dec)	N
100100 (bin) / 36 (dec)	11010111 (bin) / 215 (dec)	11110011 (bin) / 243 (dec)	0 (dec)	00001111 (bin) / 15 (dec)	36 (dec)	1 (bin) / 1 (dec)	N
00100 (bin) / 164 (dec)	01000100 (bin) / 68 (dec)	00000010 (bin) / 2 (dec)	2 (dec)	00011100 (bin) / 28 (dec)	28 (dec)	1 (bin) / 1 (dec)	N
00010 (bin) / 130 (dec)	01001000 (bin) / 72 (dec)	00000001 (bin) / 1 (dec)	1 (dec)	00111010 (bin) / 58 (dec)	58 (dec)	1 (bin) / 1 (dec)	N
01010 (bin) / 234 (dec)	10000000 (bin) / 128 (dec)	00000001 (bin) / 1 (dec)	1 (dec)	01101010 (bin) / 106 (dec)	106 (dec)	1 (bin) / 1 (dec)	N

Input X	Input Y	Output quot (Actual)	Expected quot	Output rem (Actual)	Expected rem	Output valid (Actual)	Expe
001111 (bin) / 79 (dec)	01100000 (bin) / 96 (dec)	00000000 (bin) / 0 (dec)	0 (dec)	01001111 (bin) / 79 (dec)	79 (dec)	1 (bin) / 1 (dec)	N
011100 (bin) / 92 (dec)	00000110 (bin) / 6 (dec)	00001111 (bin) / 15 (dec)	15 (dec)	00000010 (bin) / 2 (dec)	2 (dec)	1 (bin) / 1 (dec)	N

Rule: RestoringDividerRule

Input Variables: X, Y

Output Variables: quot, rem

Bit Width: 8

Pattern: StringMatchPattern

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

Generate expected values function:

```
def generate_expected(self, test_case):
    divisor = test_case["Y"]
    dividend = test_case["X"]

    # Handle division by zero
    if divisor == 0:
        quotient = (1 << self.bit_width) - 1  # Set quotient to maximum value for divide by zero case
        remainder = dividend  # Set remainder as the dividend
    else:
        quotient = dividend // divisor
        remainder = dividend % divisor

    # Handle signed division if required for 2's complement numbers
    if quotient >= (1 << (self.bit_width - 1)):
        quotient -= (1 << self.bit_width)
    elif quotient < -(1 << (self.bit_width - 1)):
        quotient += (1 << self.bit_width)

    outs = {
        "quot": quotient,
        "rem": remainder
    }
    return outs
```