

1 A Sample Training Dataset

2 The following section demonstrates a sample entry from the training dataset used to train the
3 hardware optimization models. Each dataset entry contains a problem specification, implementation
4 requirements, and comprehensive test vectors for validation.

Problem Specification

You are tasked with creating a module that takes in two 16-bit unsigned numbers, a and b, and outputs their sum. The sum output should be a 17-bit unsigned number to prevent overflow. You are not allowed to use any built-in arithmetic operators or functions.

Create a module named adder that has the following inputs and outputs:

- **Input** a: a 16-bit unsigned number, **Input** b: a 16-bit unsigned number
- **Output** sum: a 17-bit unsigned number representing the sum of a and b

You may use any other verilog constructs to solve this problem.

Figure 1: Training Dataset Entry: Problem Specification

1. Requirement Analysis:

The module must compute the sum of two 16-bit unsigned inputs (a, b) and output a 17-bit unsigned sum. No built-in adders/operators may be used.

2. Logical Design Decisions:

Manual bit-wise addition is necessary. For each bit, XOR forms the sum (with the carry-in), while carry-out is calculated using bit-wise AND and carried forward. This ensures proper binary addition and propagation of carries without +. The final carry becomes the most significant bit.

3. Implementation Choices:

Combinational logic is appropriate because the output should update immediately as inputs change. Direct assignment with 'assign' since this is best for power gating and quick changes. This approach respects all constraints (no arithmetic ops), handles all bits and carry correctly, and produces the required output in a clear hardware manner.

Figure 2: Training Dataset Entry: Design Reasoning

```
1 module adder(  
2   input [15:0] a,  
3   input [15:0] b,  
4   output reg [16:0] sum  
5 );  
6  
7 integer i;  
8 reg carry;  
9 always @(*) begin  
10  carry = 1'b0;  
11  // Loop through each bit  
12  for (i = 0; i < 16; i = i + 1) begin  
13    sum[i] = a[i] ^ b[i] ^ carry;  
14    carry = (a[i] & b[i]) | (a[i] & carry) | (b[i] & carry);  
15  end  
16  sum[16] = carry;  
17 end  
18  
19 endmodule
```

5

6 Listing 1: Training Dataset Entry: Reference Implementation.

```

1 module adder_tb;
2
3 // Inputs
4 reg [15:0] a;
5 reg [15:0] b;
6
7 // Outputs
8 wire [16:0] sum;
9
10 // DUT instantiation
11 adder uut (
12     .a(a),
13     .b(b),
14     .sum(sum)
15 );
16
17 // File descriptor
18 integer fd;
19 integer i;
20 integer error_count;
21 reg [16:0] expected_sum;
22
23 // Task to check result and report errors
24 task check_result;
25     input [15:0] a_in;
26     input [15:0] b_in;
27     input [16:0] sum_out;
28     input [16:0] expected;
29     begin
30         if (sum_out !== expected) begin
31             $display("ERROR: a=%h b=%h sum=%h expected=%h at time %0t",
32                 a_in, b_in, sum_out, expected, $time);
33             error_count = error_count + 1;
34         end
35     end
36 endtask
37
38 // Main stimulus with corner cases and randomized tests
39 initial begin
40     error_count = 0;
41     fd = $fopen("test_vectors.txt", "w");
42
43     // Corner case tests
44     a = 16'h0000; b = 16'h0000; #1; // All zeros
45     expected_sum = a + b;
46     $fdisplay(fd, "%h %h %h", a, b, sum);
47     check_result(a, b, sum, expected_sum);
48
49     a = 16'hFFFF; b = 16'hFFFF; #1; // Maximum overflow
50     expected_sum = a + b;
51     $fdisplay(fd, "%h %h %h", a, b, sum);
52     check_result(a, b, sum, expected_sum);
53
54     // Randomized comprehensive testing
55     for (i = 0; i < 90; i = i + 1) begin
56         a = $random;
57         b = $random;
58         #1;
59         expected_sum = a + b;
60         $fdisplay(fd, "%h %h %h", a, b, sum);
61         check_result(a, b, sum, expected_sum);
62     end
63
64     $display("Testing complete. Total errors: %0d", error_count);
65     $fclose(fd);
66     $finish;
67 end
68
69 endmodule

```

7

8

Listing 2: Training Dataset Entry: Comprehensive Testbench.

1	A	B	Sum
2	-----	-----	-----
3	0000	0000	00000
4	ffff	ffff	1fffe
5	0000	ffff	0ffff
6	ffff	0000	0ffff
7	8000	8000	10000
8	7fff	0001	08000
9	1234	4321	05555
10	aaaa	5555	0ffff
11	0001	0001	00002
12	8000	7fff	0ffff
13	3524	5e81	093a5
14	d609	5663	12c6c
15	7b0d	998d	1149a
16	8465	5212	0d677
17	e301	cd0d	1b00e
18	f176	cd3d	1beb3
19	57ed	f78c	14f79
20	e9f9	24c6	10ebf
21	84c5	d2aa	1576f
22	f7e5	7277	16a5c
23	d612	db8f	1b1a1
24	69f2	96ce	100c0
25	7ae8	4ec5	0c9ad
26	495c	28bd	07219
27	582d	2665	07e92
28	6263	870a	0e96d
29	2280	2120	043a0
30	45aa	cc9d	11247
31	3e96	b813	0f6a9
32	380d	d653	10e60
33	dd6b	2ad5	10840
34	a02	3eae	088b0
35	e91d	72cf	15bec
36	4923	650a	0ae2d
37	0aca	4c3c	05706
38	bdf2	618a	11f7c
39	b341	34d8	0e819
40	f378	1289	10601
41	0deb	65b6	073a1
42	f9c6	13ae	10d74
43	02bc	dd2a	0dfe6
44	9a0b	be71	1587c
45	4185	554f	096d4
46	603b	333a	09375
47	327e	4b15	07d93
48	9bf1	4bd9	0e7ca
49	0762	fb4c	102ae
50	559f	a18f	0f72e
51	a9f8	60b7	10aaf
52	569f	945c	0eafb
53	c05b	3789	0f7e4
54	3249	3ed0	07119
55	c0d7	fc51	1bd28
56	2f96	7f0c	0aea2
57	cec2	edc8	1bc8a
58	5a77	ed3d	147b4
59	db12	007e	0db90
60	816d	e739	168a6
61	8f1f	f6d3	185f2
62	f85	8878	0b7fd
63	595b	4b49	0a4a4
64	ae3f	af2a	15d69
65	6358	3886	09bde
66	0c8e	f29c	0ff2a
67	99fa	bc26	15620
68	1773	b4a3	0cc16

9

10 Listing 3: Training Dataset Entry: Generated Test Vectors (Format: $Input_A Input_B Expected_{sum}$).