Testing the ALU

Bookmark this page

Lab due Nov 7, 2016 21:59 -02 Past due

Testing the ALU

0.0/1.0 point (graded)

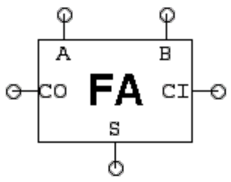
In this lab's design problem, you were asked to build a 32-bit arithmetic and logic unit (ALU) that performs arithmetic and logic operations on 32-bit operands, producing a 32-bit result. You may want to work on the design problem first and come back to this problem when you've completed the design.

The test for the design problem verifies your ALU circuitry by applying 186 different sets of input values. This question explores how those values were chosen.

No designer I know thinks testing is fun -- designing the circuit seems so much more interesting than making sure it works. But a buggy design isn't much fun either! Remember that a good engineer not only *knows* how to build good designs but also *actually* builds good designs, and that means testing the design to make sure it does what you say it does.

An obvious way to test a combinational circuit is to try all possible combinations of inputs, checking for the correct output values after applying each input combination. This type of *exhaustive test* proves correct operation by enumerating the truth table of the combinational device. This is a workable strategy for circuits with a few inputs but quickly becomes impractical for circuits with many inputs. By taking advantage of information about how the circuit is constructed we can greatly reduce the number of input combinations needed to test the circuit.

The ripple-carry adder architecture suggested in the Design Problem uses 32 copies of the **full adder** module to create a 32-bit adder. Each full adder has 3 inputs (A, B, CI) and two outputs (S, CO):



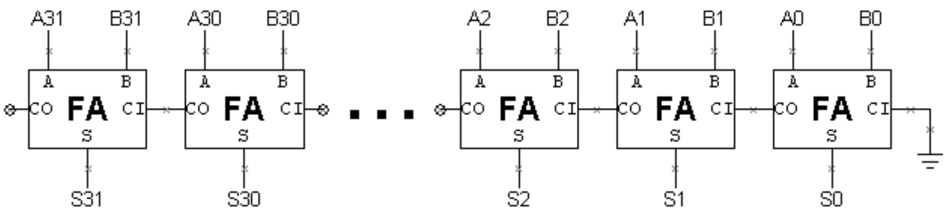
A single *test vector* for the full adder consists of 3 input values (one each for A, B and CI) and 2 output values (S and CO). To run a test the input values from the current test vector are applied to the device under test and then the actual output values are compared against the expected values listed by the test vector. This process is repeated until all the test vectors have been used. Assuming we know nothing about the internal circuitry of the full adder, how many test vectors would we need to exhaustively test its functionality?

Number of test vectors to exhaustively test full adder: 8 Answer: 8

Consider a 32-bit adder with 64 inputs (two 32-bit input operands, assume CIN is tied to ground as shown in the diagram below) and 32 outputs (the 32-bit result). Assume we don't know anything about the internal circuitry and so can't rule out the possibility that it might get the wrong answer for any particular combination of inputs. In other words, just because the adder got the correct answer for 2 + 3 doesn't allow us to draw any conclusions about what answer it would get for 2 + 7. If we could apply one test vector every 100ns, how long would it take to exhaustively test the adder?

Time to exhaustively test 32-bit adder (in years): 58561 Answer: 58454

Clearly, testing a 32-bit adder by trying all combinations of input values isn't a good plan! Shown below is a schematic for a 32-bit ripple-carry adder.



Except for the carry-in from the bit to the right, each bit of the adder operates independently. We can use this observation to test the adder bit-by-bit and with a bit of thought we can actually run many of these tests in parallel. In this case the fact that the adder got the correct answer for 2 + 3 actually tells us a lot about the answer it will get for 2 + 7. Since the computation done by adder bits 0 and 1 is same in both cases, if the answer for 2 + 3 is correct, the low-order two bits of the answer for 2 + 7 will also be correct.

So our plan for testing the ripple-carry adder is to test each full adder independently. When testing bit N we can set A[N] and B[N] directly from the test vector. It takes a bit more work to set CI[N] to a particular value, but we can do it with the correct choices for A[N-1] and B[N-1].

If we want to set CI[N] to 0, what values should A[N-1] and B[N-1] be set to? If we want to set CI[N] to 1? Assume that we can't assume anything about the value of CI[N-1].

Values of A[N-1] and B[N-1] to make C[N]=0?

- [✓ A[N-1]=0, B[N-1]=0] ✓
- A[N-1]=1, B[N-1]=0
- A[N-1]=0, B[N-1]=1
- A[N-1]=1, B[N-1]=1

Values of A[N-1] and B[N-1] to make C[N]=1?

- A[N-1]=0, B[N-1]=0
- A[N-1]=1, B[N-1]=0
- A[N-1]=0, B[N-1]=1
- [✓ A[N-1]=1, B[N-1]=1] ✓

With this strategy we can test the even bits of the adder in parallel with one set of test vectors and test the odd bits of the adder in parallel with another set of test vectors. Here's a set of 10 test vectors that should test all combinations of input values for each FA in a 32-bit ripple-carry adder:

bits 0, 2, ...	**bits 1, 3, ...**	**A[31:0]**	**B[31:0]**
A=0, B=0, CI=0	A=0, B=0, CI=0	0x00000000	0x00000000
A=1, B=0, CI=0	A=0, B=0, CI=0	0x55555555	0x00000000
A=0, B=1, CI=0	A=0, B=0, CI=0	0x00000000	0x55555555
A=1, B=1, CI=0	A=0, B=0, CI=1	0x55555555	0x55555555
A=0, B=0, CI=0	A=1, B=0, CI=0	0xAAAAAAAA	0x00000000
A=0, B=0, CI=0	A=0, B=1, CI=0	0x00000000	0xAAAAAAAA
A=0, B=0, CI=1	A=1, B=1, CI=0	0xAAAAAAAA	0xAAAAAAAA
A=1, B=1, CI=1	A=1, B=1, CI=1	0xFFFFFFFF	0xFFFFFFFF
A=0, B=1, CI=1	A=0, B=1, CI=1	0x00000001	0xFFFFFFFF
A=1, B=0, CI=1	A=1, B=0, CI=1	0xFFFFFFFF	0x00000001

Three of the compare unit's inputs (Z, V and N) come from the adder/subtractor running in subtract mode computing A-B:

$Z = 1$ if $A - B$ is 0

$N = 1$ if $A - B$ is negative (OUT[31] = 1)

$V = 1$ if there's been an overflow. The ALU, which only has an adder, computes $A - B$ as $A + (-B) = A + (\sim B) + 1$. Let XB = ~B, the bit-wise complement of B. An overflow occurs if the sign of the result (OUT[31]) differs from the signs of the adder's operands (A[31], XB[31]). Note that if the signs of A and XB differ, the addition cannot produce an overflow.

To test the compare unit, we'll need to pick operands for the adder/subtractor that generate all possible combinations of Z, V and N. It's easy to see that any combination with Z = 1 and N = 1 is not possible (the output of the adder cannot be negative and zero at the same time!). It also turns out that combinations with Z = 1 and V = 1 cannot be produced by a subtract operation.

For each of the combinations of Z, V and N shown below, choose the subtraction operation that will produce the specified combination of condition codes.

Subtraction that produces Z=0, V=0, N=0?

- 0×12345678 - 0×12345678
- 0×7FFFFFFF - 0xFFFFFFFF
- [✓ 0×00000005 - 0xDEADBEEF] ✓
- 0xDEADBEEF - 0×00000005
- 0×80000000 - 0×00000001

Subtraction that produces Z=1, V=0, N=0?

- [✓ 0×12345678 - 0×12345678] ✓
- 0×7FFFFFFF - 0xFFFFFFFF
- 0×00000005 - 0xDEADBEEF
- 0xDEADBEEF - 0×00000005
- 0×80000000 - 0×00000001

Subtraction that produces Z=0, V=1, N=0?

- 0×12345678 - 0×12345678
- 0×7FFFFFFF - 0xFFFFFFFF
- 0×00000005 - 0xDEADBEEF
- 0xDEADBEEF - 0×00000005
- [✓ 0×80000000 - 0×00000001] ✓

Subtraction that produces Z=0, V=0, N=1?

- 0×12345678 - 0×12345678

- 0×7FFFFFFF - 0xFFFFFFFF
- 0×00000005 - 0xDEADBEEF
- [✓ 0xDEADBEEF - 0×00000005] ✓
- 0×80000000 - 0×00000001

Subtraction that produces Z=0, V=1, N=1?

- 0×12345678 - 0×12345678
- [✓ 0×7FFFFFFF - 0xFFFFFFFF] ✓
- 0×00000005 - 0xDEADBEEF
- 0xDEADBEEF - 0×00000005
- 0×80000000 - 0×00000001

Submit

Answers are displayed within the problem

Previous Next

edX

About
Affiliates
edX for Business
Open edX
Careers
News

Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap
Cookie Policy
Your Privacy Choices

Connect

Idea Hub
Contact Us
Help Center
Security
Media Kit

Calculator

GET IT ON Google play
Download on the App Store