4.1 Adders

Many digital circuits, like in a calculator or computer, must add binary numbers. For decimal numbers, adding by hand starts at the right and adds each digit, possibly carrying a 1 to the next digit. Adding in binary is identical.

PARTICIPATION ACTIVITY 4.1.1: Adding in binary.

Animation captions:

1. Add rightmost digit: 1 + 0 = 1.
2. Add next digit: 1 + 1 = 10, so carry 1.
3. Continue for next digit: 1 + 1 + 1 = 11. Carry 1.
4. Continue for the last digit: 1 + 0 + 0 = 1.

PARTICIPATION ACTIVITY 4.1.2: Adding binary numbers.

1)
```
  0010
+ 0010
----------
```

Check Show answer

2)
```
  0110
+ 0010
----------
```

Check Show answer

3)
```
  0101
+ 0111
----------
```

Check Show answer

4) 1111
+ 0001

Check Show answer

An ***adder*** computes A + B, where A and B are N-bit numbers, such as 8-bit numbers. A ***carry-ripple adder*** mimics adding by hand, adding a digit's pair of bits and carry-in bit, and generating a sum and carry-out bit.

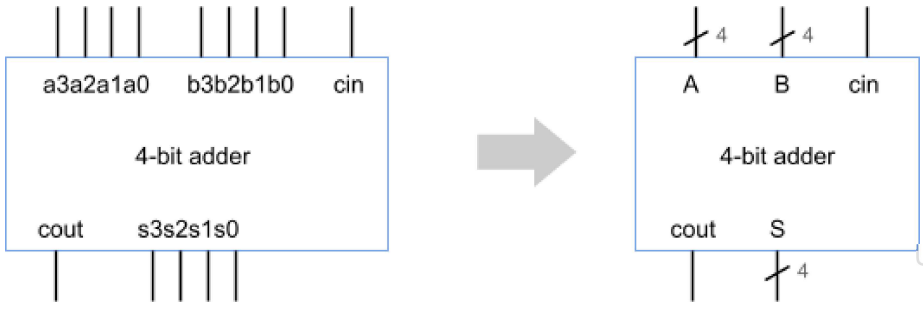
PARTICIPATION ACTIVITY 4.1.3: A carry-ripple adder.

Animation captions:

1. Each digit's pair of bits and carry-in bit is added. A sum and carry-out bit is generated.
2. The carry-out (cout) from each addition propagates to the left.
3. Each digit pair and carry-in are added at the same time. Notice that the initial result is not correct.
4. Once carry bits propagate left, the correct result is obtained: 1101 + 1001 = 10110.
5. An adder is commonly represented as a block symbol.

A datapath component is commonly represented as a block symbol. This material generally uses uppercase letters to represent multi-bit data and lowercase letters to represent single-bit data. A multi-bit wire is drawn as a single wire with a slash (/), as shown below.

Figure 4.1.1: Block symbol: 4-bit adder.



PARTICIPATION ACTIVITY 4.1.4: Carry-ripple adders.

1) A 12-bit carry-ripple adder adds two ?-bit numbers.
   - 6
   - 12
   - 13

2) Each digit's pair of bits and carry-in bit are added simultaneously.
   - False
   - True

3) A = 1111, B = 0001
   - cout = 0
     s3s2s1s0 = 0000
   - cout = 1
     s3s2s1s0 = 1111
   - cout = 1
     s3s2s1s0 = 0000

A ***full adder*** is a circuit that adds three bits and generates a sum and carry-out. Four full adders were used above. The strange name "full adder" is historical, intended to contrast with a half adder. A ***half adder*** is a circuit that adds two bits and generates a sum and carry-out bit. A full adder can be designed starting from a truth table.

PARTICIPATION ACTIVITY 4.1.5: Full adder.

Animation content:

undefined

Animation captions:

1. Fill in truth table.
2. Continue filling in truth table.
3. Derive circuit equations.
4. Draw full adder circuit.

PARTICIPATION ACTIVITY 4.1.6: Full adders.

1) A circuit that adds three bits and generates a sum and carry-out is know as a _____.

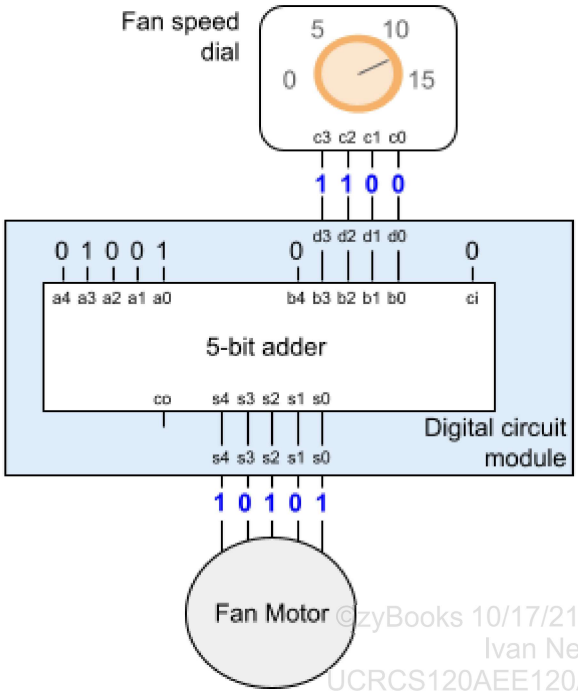
- Half adder
- Full adder

2) How many full adders are needed to create a 7-bit carry-ripple adder?

- 7
- 14

Example 4.1.1: Fan speed adjustment using an adder.

An electronic room fan has a digital circuit module. The module's 5-bit output S controls the fan's motor rotation speed: 0 means no rotation, 11111 means fastest rotation. The module's 4-bit input D comes from a dial. When on, the fan's slowest speed is S = 9 (01001). Turning the dial increases D anywhere from 0(0000) to 15 (1111). The output S should be set as: S = 9 + D.



PARTICIPATION ACTIVITY 4.1.7: Fan speed adjustment.

Consider the fan speed adjustment example.

1) What is the greatest possible value of S? Give answer in decimal.

Check Show answer

2) What is the greatest possible value of S? Give answer in binary.

Check Show answer

3) Suppose the fan had an electronic on-off switch. When off, S should output what value? Give answer in decimal.

Check Show answer

An ***incrementer*** adds 1 to a number, which is a common operation. An incrementer can be built by inputting 00..00 to an adder's B input, and 1 to the carry-in. Such a circuit is unnecessarily large, because each digit's full adder can add 3 bits (a, b, ci), but b is always 0. A ***half adder*** adds two bits (a, b), and is sufficient for an incrementer.

Figure 4.1.2: Half adder truth table equations, and circuit.

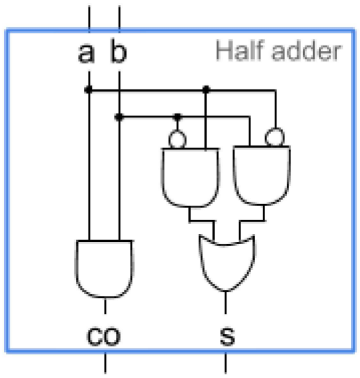
©zyBooks 10/17/21 22:04 829162
Ivan Neto
UCRCS120AEE120AChenFall2021

a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

co = ab

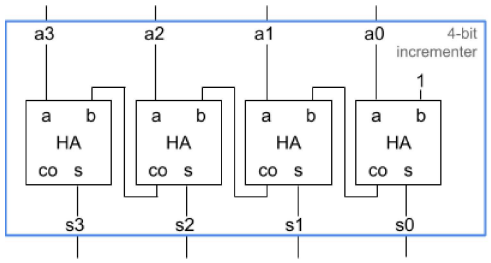
s = a'b + ab'

(a) Half adder truth table

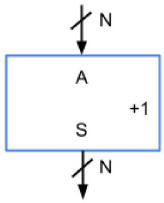


(b) Half adder (HA) circuit

Figure 4.1.3: Incrementer circuit and block diagram.



(a) 4-bit incrementer constructed with half adders



(b) Incrementer block diagram

PARTICIPATION ACTIVITY 4.1.8: Half adders.

1) A half adder adds two bits.
   - True
   - False

2) A half adder generates a carry-out if any input is 1.
   - True
   - False

3) A half adder circuit requires fewer gates than a full adder circuit.

- True
- False

4) An incrementer adds 1 to input A.

- True
- False

5) Full adders can be used to construct an incrementer.

- True
- False

6) A 4-bit carry-ripple adder can be constructed using 4 half adders.

- True
- False

CHALLENGE ACTIVITY 4.1.1: Add the two 4-bit numbers.

347136.1658324.qx3zqy7

Start

Add 8 and 4

	0	0	0	0	Carry bits
	1	0	0	0	4-bit number
+	0	1	0	0	4-bit number
	0	0	0	0	Result number

1 2 3 4 5 6

Check Next

4.2 Signed numbers in binary

Unsigned numbers involve only non-negative numbers, like 0 and 3. ***Signed numbers*** involve both positive and negative numbers, like 3 and -3.

In binary, a ***signed-magnitude representation*** uses the left bit for the sign: 0 means positive, 1 means negative. Ex: For 4-bit numbers, 0011 is 3, and 1011 is -3. Signed-magnitude representation is rarely used, because calculations involving negative numbers, such as 5 - 3, would require special circuits beyond an adder.

A more clever negative number representation exists that can use an adder for both positive and negative numbers. A ***complement*** of an N-digit number is another number that yields a sum of 100..00 (with N 0's), and can be used to represent the negative of that number.

PARTICIPATION ACTIVITY 4.2.1: Two's complement signed number representation.

Animation captions:

1. Intuition in base 10.
2. In base two, a complement is obtained by inverting each bit and adding 1.
3. Subtraction is performed by adding the complement.

The above is called the ***two's-complement representation***, which inverts every bit and adds 1. One's complement also exists, but is rarely used, so not discussed further. This material uses "complement" to mean two's complement.

The left bit indicates the sign. 0011 is +3. 1101 is a negative; complementing yields the positive version: 0010 + 1 = 0011, which is 3. So 1101 is -3.

This section uses 4-bit numbers for ease of example; wider numbers like 8 or 32 bits are more typical.

PARTICIPATION ACTIVITY 4.2.2: Two's-complement signed number representation.

1) In base ten, for two digits, what is the complement of 33?

Check Show answer

2) In base two, for four bits, what is

the complement of 0010?

Check Show answer

3) What is -2 in four-bit two's-complement representation?

Check Show answer

4) What is -7 in four-bit two's-complement representation?

Check Show answer

5) Assuming four-bit two's-complement representation, is 1011 positive or negative?

Check Show answer

6) Assuming two's-complement representation, what base ten number does 1001 represent?

Check Show answer

7) Assuming two's-complement representation, what base ten number does 1111 represent?

Check Show answer

8) In base two, for four bits, what is the complement of 0000?

Check Show answer

9) What is -3 in eight-bit two's-complement representation?

Check Show answer

Two's-complement representation has the benefit of allowing an adder to be used even when dealing with negative numbers. Ex: 5 + -3 is just 0101(5) + 1101(-3) = 10010, or 0010(2) after ignoring the carry. No extensive special circuitry for negative numbers is needed.

PARTICIPATION ACTIVITY 4.2.3: Two's-complement arithmetic.

Assume four-bit two's-complement representation.

1) 6 + 2 is 0110 + ?

Check Show answer

2) 6 + -2 is 0110 + ?

Check Show answer

3) 3 + -4 is 0011 + 1100 = ?

Check Show answer

4) 2 - 3 is 0010 + ?

Check Show answer

5) -3 + 2 is ? + 0010

Check Show answer

The largest positive four-bit two's-complement number is 0111, or 7. The smallest negative is 1000, or -8 (0111 + 1 = 1000, so magnitude is 8). Note that two's complement represents one more negative than positive. Positive 8, 9, ... cannot be represented in four-bit two's complement. Likewise, -9, -10, ... cannot either. Adding two positives, or adding two negatives, may yield a value that can't be represented in the given number of bits, a situation known as ***overflow***. Ex: 0101 (5) + 0011 (3) incorrectly yields 1000, which is -8 in two's complement.

PARTICIPATION ACTIVITY 4.2.4: Overflow.

Animation captions:

1. Adding two positives may overflow.
2. Adding two negatives may overflow. Note: Ignore carry-out bit.
3. Adding a positive and negative cannot overflow.
4. Overflow if numbers' sign bits were 0's but sum's is 1, OR numbers' sign bits were 1's but sum's is 0.

PARTICIPATION ACTIVITY 4.2.5: Overflow.

All numbers are in four-bit two's-complement representation.

1) 0011 + 0010 results in overflow.
   - True
   - False

2) 0111 + 0110 results in overflow.
   - True
   - False

3) 0001 + 1111 results in overflow.
   - True
   - False

4) 1011 + 1110 results in overflow.
   - True
   - False

5) Number A's sign bit is 0. Number B's sign bit is 0. The sum's sign bit is 1. The addition resulted in overflow.

- True
- False

6) Number A's sign bit is 1. Number B's sign bit is 0. The sum's sign bit is 0. The addition resulted in overflow.

- True
- False

CHALLENGE ACTIVITY 4.2.1: Signed numbers in binary.

347136.1658324.qx3zqy7

Start

What is -6 in four-bit two's-complement representation?

Ex: 0101

1 2 3 4 5

Check Next

4.3 Subtractors

A ***subtractor*** computes A - B, where A and B are N-bit numbers, such as 8-bit numbers. If numbers are represented using two's-complement representation, a subtractor can be built using an adder. Inverting B's bits and setting the adder's carry-in to 1 adds B's complement to A.

PARTICIPATION ACTIVITY 4.3.1: A 4-bit subtractor.

Animation content:

undefined

Animation captions:

1. Subtractor can be built using an adder.
2. Complementing B requires inverting B's bits...
3. ...and adding 1, achieved by setting carry-in to 1.
4. Example: 5 - 3 = 5 + (-3).
5. Example: 5 - 3 = 2.
6. A subtractor is commonly represented as a block symbol.

PARTICIPATION ACTIVITY 4.3.2: Subtractor.

1) An adder can be used to perform subtraction if input B is represented in two's complement.
   - True
   - False

2) For a subtractor built from an adder, the adder is configured to subtract by setting the adder's cin bit to ___.
   - 0
   - 1

3) To perform 7 - 2 using a 4-bit subtractor, A is 0111 and B is _____.
   - 0010
   - 1110

Because two's-complement representation performs subtraction by complementing and adding, a single adder circuit can perform either addition or subtraction, thus saving circuit size.

PARTICIPATION ACTIVITY 4.3.3: A 4-bit adder/subtractor.

Animation captions:

1. 2x1 muxes can choose between B and inverted B.
2. Input sub = 0 does normal addition.
3. Input sub = 1 does subtraction by complementing B (inverting B, carry-in = 1).
4. Input sub = 1 does subtraction by complementing B (inverting B, carry-in = 1).
5. An adder/subtractor is commonly represented as a block symbol.

PARTICIPATION ACTIVITY 4.3.4: Adder/subtractor.

Consider a 4-bit adder/subtractor (seen in the animation above).

1) If sub is 0, what is the adder/subtractor component configured to do?
   - Addition
   - Subtraction
   - sub does not determine the circuit's operation.

2) Which of the following operations are not valid?
   - 5 - 3
   - (-5) - (-3)
   - -5 + 3
   - None of the above.

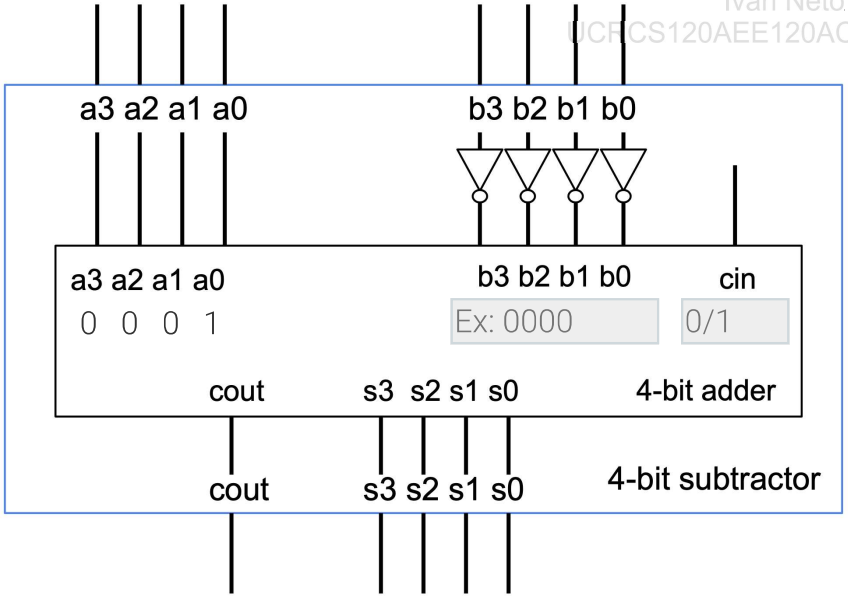
3) Configure the adder/subtractor to perform the following operation: 7 - 2
   - a3a2a1a0 = 0111
b3b2b1b0 = 0010
sub = 0
   - a3a2a1a0 = 0111
b3b2b1b0 = 0010
sub = 1
   - a3a2a1a0 = 0010
b3b2b1b0 = 0111
sub = 1

CHALLENGE ACTIVITY 4.3.1: Subtractors.

347136.1658324.qx3zqy7

Start

Trace the subtractor's logic when performing 1 - 7.



1 2 3

Check Next

4.4 Comparators

A ***comparator*** compares two numbers, indicating whether the numbers are equal, or which number is greater. Same-length unsigned binary numbers can be compared by hand just like base ten numbers: Starting from the left, digits are compared until a difference is found.

PARTICIPATION ACTIVITY 4.4.1: Comparator.

Animation captions:

1. Comparator outputs whether A is greater than, A is less than, or A is equal to B.
2. Achieved by starting from left, comparing each digit until finding 1 and 0.
3. If A's bit is 0, A < B. If A's bit is 1, A > B.
4. This time A > B.
5. If reach rightmost bit and still no difference, A = B.

PARTICIPATION ACTIVITY 4.4.2: Comparator.

Indicate which comparator output will be 1.

1) A: 1100
B: 1101
   - gt
   - lt
   - eq

2) A: 0100
B: 1000
   - gt
   - lt
   - eq

3) A: 1111
B: 1111
   - gt
   - lt
   - eq

A ***carry-ripple comparator*** compares two N-bit numbers from left to right, with the result of each digit's comparison "rippling" to the next digit. For each digit, a ***one-bit comparator*** compares two bits a and b only if the eq input was 1 from the higher digit, else just passing along a gt 1 or an lt 1. The rightmost digit's output becomes the N-bit comparator's output. The name "carry-ripple" refers to the similarity of the comparator's implementation to a carry-ripple adder's implementation.

PARTICIPATION ACTIVITY 4.4.3: Carry-ripple comparator.

Animation captions:

1. N-bit comparator can be built using N components whose result ripples to next digit.
2. Leftmost digits are compared first. eqi is 1 so the comparison between a and b will determine gto, gti, and eqo
3. Result ripples to next digit. While eqi is 1, the comparison between a and b will determine gto, gti, and eqo
4. Once lti is set, there is no need to continue to compare. lti is passed to lto.

The correct output of leftmost digit ripples to next digit, and so on. Eventually the external outputs become correct.

PARTICIPATION ACTIVITY 4.4.4: Carry-ripple comparator.

Consider a 4-bit carry-ripple comparator (seen in the above animation). Indicate which output will be 1 for each digit.

Assume: a3a2a1a0 = 0100 (4), b3b2b1b0 = 0010 (2)

1) Digit 3
   - gto
   - lto
   - eqo

2) Digit 2
   - gto
   - lto
   - eqo

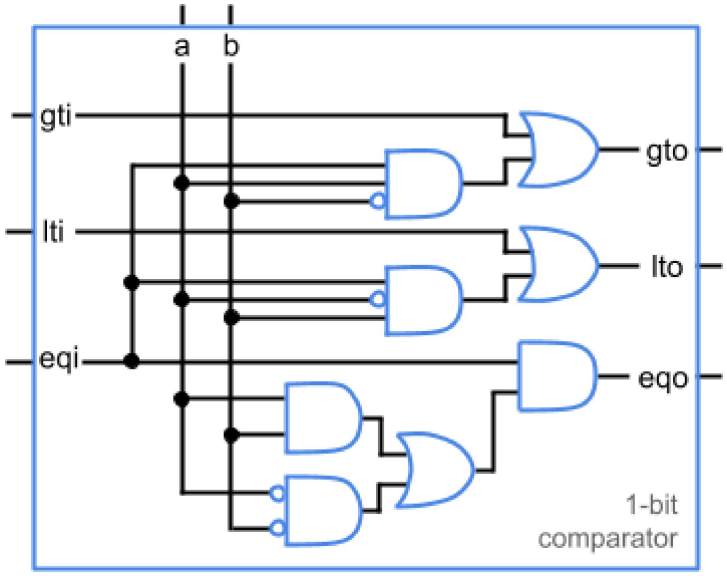
3) Digit 1
   - gto
   - lto
   - eqo

4) Digit 0
   - gto
   - lto
   - eqo

A one-bit comparator can be implemented using combinational logic for each output. A designer could start by filling in a truth table with 5 inputs gti, lti, eqi, a, and b, and 3 outputs gto, lto, eqo. The truth table will have 2^5 = 32 rows. Alternatively, the designer can start directly with equations.

- eqo should be 1 if eqi is 1 AND a, b are the same. Thus: eqo = eqi(ab + a'b').
- gto should be 1 if gti is 1, OR eqi is 1 AND ab are 10. Thus: gto = gti + eqi(ab').
- lto should be 1 if lti is 1, OR eqi is 1 AND ab are 01. Thus: lto = lti + eqi(a'b).

Figure 4.4.1: 1-bit comparator circuit (used in questions below).



PARTICIPATION ACTIVITY 4.4.5: 1-bit Comparator.

1) Given: a = 1, b = 1, gti = 0, lti = 0, eqi = 1.
   - eqo = 0
   - eqo = 1

2) Given: a = 1, b = 0, gti = 0, lti = 0, eqi = 1.
   - eqo = 0
   - eqo = 1

3) Given: a = 0, b = 0, gti = 0, lti = 1, eqi = 0.
   - eqo = 0
   - eqo = 1

4) Given: a = 0, b = 0, gti = 0, lti = 1, eqi = 0.
   - lto = 0
   - lto = 1

5) Given: a = 0, b = 1, gti = 0, lti = 0, eqi = 1.

- ○ lto = 0
- ○ lto = 1

6) Given: a = 1, b = 0, gti = 0, lti = 0, eqi = 1.

- ○ lto = 0
- ○ lto = 1

7) Given: a = 0, b = 1, gti = 1, lti = 0, eqi = 0.

- ○ gto = 0
- ○ gto = 1

Example: Computing the minimum of two numbers using a comparator

A datapath component to select the minimum (or maximum) of two numbers can be designed using a comparator and multiplexor.

PARTICIPATION ACTIVITY 4.4.6: Computing the minimum of two numbers using a comparator.

Animation content:

undefined

Animation captions:

1. To find the minimum of two numbers, an 8-bit magnitude comparator and an 8-bit 2x1 mux is used.
2. The comparator's AltB output is connected to the mux's select line, and A is connected to the mux' input I1. If A < B, AltB = 1, so mux input I1, or A, is passed to the output.
3. B is connected to the mux' input I0. If A > B, then AgtB = 1 and AltB = 0. Since AltB = 0, the mux's input I0, or B, is passed to the output.
4. If A = B, either A or B could be passed to the output. But AltB is used as the mux select line. So when A = B, AltB = 0, and the mux's input I0, or B, is passed to the output.

PARTICIPATION ACTIVITY 4.4.7: Computing the minimum of two numbers example.

1) The comparator's eqi input is set to 0 to force the comparator to compare

the data inputs A and B.

- True
- False

2) If the AgtB input is connected to the select line of the mux, the comparator will compute and output the greater of the two data inputs, A and B.

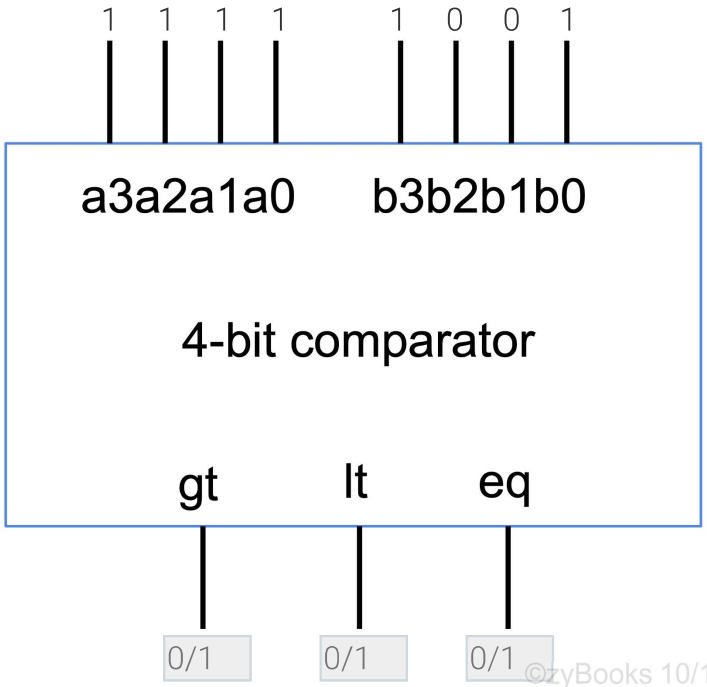
- True
- False

CHALLENGE ACTIVITY 4.4.1: Comparators.

347136.1658324.qx3zqy7

Start

Determine the output of the comparator.



1 2 3

Check Next

4.5 Example: Color space converter: RGB to CMYK

RGB and CMYK color spaces

Many electronic devices deal with color images. Pixels (picture elements) are tiny indivisible dots of color that are part of images displayed on any screen. The number of pixels in an image depends on the resolution of the display. The RGB color space is often used for displays to represent color, as any color can be created by using specific intensities of red, green, and blue light.

However, the RGB colorspace is not ideal for other media. Printers use the CMY (Cyan/Magenta/Yellow) color space, as ink absorbs some light while reflecting others to create colors. Cyan ink absorbs red, reflecting green and blue. Magenta ink absorbs green, reflecting red and blue. Yellow absorbs blue, reflecting red and green. Thus, a printer converts a received RGB image to CMY to print.

PARTICIPATION ACTIVITY 4.5.1: RGB color space to CMY color space.

Animation content:

undefined

Animation captions:

1. Displays typically use the RGB color space to represent color. Any color can be created by combining specific intensities of red, green, and blue light.
2. Each sub color (Red, Blue, Green) is represented as an 8-bit binary number, each ranging from 0 to 255. Black is (0, 0, 0). White is (255, 255, 255). The light blue is (176, 215, 250).
3. Printers use CMY (Cyan, Magenta, Yellow) color space, so images in RBG colorspace must be converted. Cyan ink absorbs red, magenta ink absorbs green, and yellow ink absorbs blue.
4. Black ink is inexpensive. Printers extend the CMY color space to CMYK, where K is black. The usage of black ink is maximized by factoring out black from the C, M, and Y values.

PARTICIPATION ACTIVITY 4.5.2: RGB to CMYK color space.

1) Given a RGB value of (250, 10, 120),

what is the C value in the CMY color space?

- 5
- 245
- 135

2) Given a CMY value of (130, 150, 200), what is the value of magenta after K has been factored out?

- 150
- 130
- 20

3) Given a CMY value of (223, 200, 32), what portion can be generated using only black ink?

- (223, 223, 223)
- (200, 200, 200)
- (32, 32, 32)

RGB to CMYK converter

An RGB to CMYK converter can be implemented as a datapath using subtractors and minimum components.

PARTICIPATION ACTIVITY 4.5.3: RGB to CMYK converter.

Animation content:

undefined

Animation captions:

1. Cyan, Magenta, and Yellow are derived from Red, Blue, and Green using subtractors. C = 255 - R; M = 255 - G; Y = 255 - B.
2. Two instances of the Min component are used to factor out the black. K = Minimum(C, M, Y) is implemented as Minimum(Minimum(C, M), Y).
3. Three subtractors remove the K value from the C, M, and Y values.

PARTICIPATION

ACTIVITY 4.5.4: RGB to CMYK converter.

1) The two minimum (Min) components are used to _____.
   - compute the minimum of the three values R, G, and B
   - compute the minimum of the three values C, M, and Y
   - compute the minimum of C and M and the minimum of Y and K

4.6 N-bit muxes

Commonly, multiple data bits are muxed.

PARTICIPATION ACTIVITY 4.6.1: 3-bit 2x1 mux.

Animation captions:

1. A 3-bit 2x1 mux uses three 2x1 muxes to pass data through.
2. s = 0 allows a2a1a0 to pass through.
3. s = 1 allows b2b1b0 to pass through.

PARTICIPATION ACTIVITY 4.6.2: Configuring a 3-bit 2x1 mux.

Configure a 3-bit 2x1 to obtain the desired result.

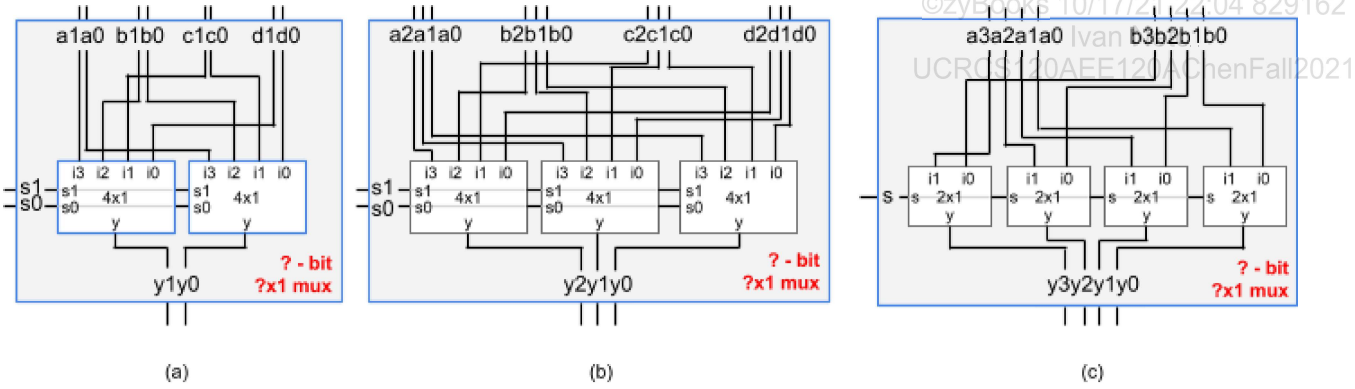
1) If s = 1, y2y1y0 = ?
   - a2a1a0
   - b2b1b0

2) To pass a2a1a0 to y2y1y0, s = ?
   - 0
   - 1

3) To pass I1 to Y, s = ?

- 0
- 1

PARTICIPATION ACTIVITY 4.6.3: Different mux sizes.



(b) (c) (a)

	4-bit 2x1 mux
	2-bit 4x1 mux
	3-bit 4x1 mux

Reset

4.7 Load registers

Load register design

A designer may want to ***load*** a register only on certain clock cycles rather than on every clock cycle. Registers commonly come with a control input named "load" or "ld". Implementing such a load register can be done using 2x1 muxes.

PARTICIPATION ACTIVITY 4.7.1: 3-bit load register implemented with 3 flip-flops and muxes.

Animation captions:

1. 2x1 muxes can be added to create a register with a load control input.
2. ld = 0 causes present bits to be reloaded back into flip-flops on next rising clock.
3. ld = 1 causes new bits to be loaded into flip-flops on next rising clock.

Many registers also come with a reset or clear input, to reset/clear the stored bits to 0's (described in another section).

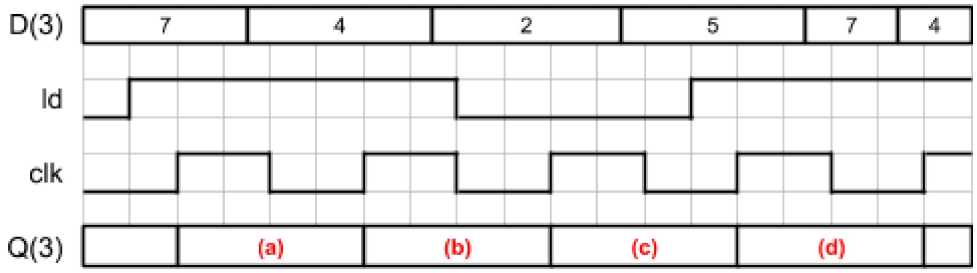
PARTICIPATION ACTIVITY 4.7.2: Multi-bit data in a timing diagram.

Animation captions:

1. In a timing diagram, single bits are represented by a line that is high or low.
2. Multi-bit data like a 3-bit input is represented as a rectangle with a number inside.

PARTICIPATION ACTIVITY 4.7.3: Trace a load register's behavior.

For the given values of D, ld, and clk, indicate the register's Q value.



1) (a)

Q =

Check Show answer

2) (b)

Q =

Check Show answer

3) (c)

Q =

Check Show answer

4) (d)

Q =

Check Show answer

Load register behavior

An N-bit ***load register*** (such as an 8-bit load register or just 8-bit register) stores N bit values, loading new bit values when a clock input rises if a load input is 1. A load input (ld) indicates when the register should be loaded. A reset input (rst) may exist that indicates that the register's bits should be reset to 0 (having priority over ld).

All a circuit's registers may share one ***clock*** signal, whose ***rising edge*** (the instant a 0 changes to 1) synchronizes loading of all registers (like a drum beat synchronizes a marching band).

PARTICIPATION ACTIVITY 4.7.4: Registers.

Animation captions:

1. The reset input is set with 1 when the system is powered on, resetting the stored bits to 000, which then appear at the outputs.
2. When the clock rises, if ld is 0, the values on d2d1d0 are ignored. Here, the register continues to store 000.
3. When the clock rises, if ld is 1, the data input values are stored in the register, and appear at the outputs.
4. Those bits remain stored, even though the data inputs may change (until a future rising clock with ld = 1).
5. Multiple wires are typically drawn on the block diagram using one thicker wire, and on the timing diagram as values within a rectangle.
6. A single clock signal typically synchronizes the loads of all registers in a design.

PARTICIPATION ACTIVITY 4.7.5: Registers.

Consider the above 3-bit load register.

1) What are q2q1q0 when rst = 1 and ld = 1, assuming q2q1q0 were previously

110, and d2d1d0 are 111?

- 110
- 000
- 111
- Depends on clk's value

2) Assume rst = 0, ld = 1, d2d1d0 are 110, and q2q1q0 are 111. When a rising clock occurs, what do q2q1q0 become?

- 111
- 110
- 000

3) Assume rst = 0, ld = 0, d2d1d0 are 110, and q2q1q0 are 111. When a rising clock occurs, what do q2q1q0 become?

- 111
- 110
- 000

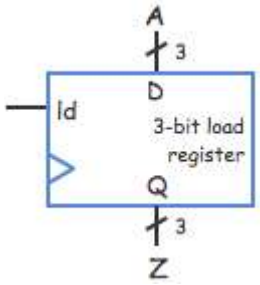
4) Assume Y connects to register R1's inputs, and R1's outputs connect to R2's data inputs. On rising clock 1, Y was 100. On rising clock 2, Y was 000. On rising clock 3, Y was 111. What is now in R2?

- 100
- 000
- 111

CHALLENGE ACTIVITY 4.7.1: Indicate Z's value over time.

347136.1658324.qx3zqy7

Start



A	6	6	4	4	0	0	4
ld							
clk							
Z	0	0	0	0	0	0	0

1 2 3 4 5 6

Check Next

4.8 Example: Above-mirror display using parallel-load registers

A previous example introduced an above-mirror display system using a mux. The system could display four 8-bit inputs, including the temperature (T), average mpg (A), instantaneous mpg (I), and miles remaining (M). The car's computer was connected to the display using 32 wires (4 inputs, 8 bits each). The number of wires connecting the car's computer to the above mirror display can be reduced using registers and a decoder.

PARTICIPATION ACTIVITY 4.8.1: Above-mirror display using registers.

Animation content:

undefined

Animation captions:

1. The temperature (T), average mpg (A), instantaneous mpg (I), and miles remaining (M) need to be displayed on the above-mirror display. Four parallel-load registers are used.
2. The car's computer connects to the display using an 8-bit bus (C). a1a0 specifies which data item appears on C. A decoder decodes a1a0 to load C into the corresponding register.
3. The car computer's load output enables the decoder. a1a0 = 00 loads reg0 (T), a1a0 = 01 loads reg1 (A), a1a0 = 10 loads reg2 (I), and a1a0 = 11 loads reg3 (M).
4. Initially all registers store 0, and xy = 0. The mux select lines are controlled by a button in the car. Each button press causes xy to sequence through 00, 01, 10, and 11.
5. xy is currently 00, so reg0 (T) is output, and D = 00000000. If a1a0 = 01, load = 1, and C = 00001010, reg1 (A) is loaded with 00001010.
6. If the button is pressed, xy = 01, and the value of reg1 (A) will be displayed.
7. If the button is pressed again, xy = 10. the value in reg2 is displayed which is 0. The values of the registers are maintained until new values are loaded.

PARTICIPATION ACTIVITY 4.8.2: Above-mirror display using registers.

Given:
reg0 = 00010001,
reg1 = 11110000,
reg2 = 10101010,
reg3 = 11111111.

1) If a1a0 = 01, load = 1, and C = 00000111, and xy = 10. What is D?

- ○ 00000111
- ○ 10101010
- ○ 11110000

2) If a1a0 = 01, load = 1, and C = 00000111, and xy = 01. What is D?

- ○ 00000111
- ○ 10101010
- ○

11110000

3) How many control inputs does the system have?

- 2
- 3
- 5

4) How many data inputs does the system have?

- 0
- 1
- 2

5) How many wires are needed to connect the car's computer to the display?

- 8
- 11
- 32

4.9 Shifters

Shift operations

Shifting is a process used to reposition bits of data, either to the left or to the right. Shift operations can be used to perform multiplication or division operations on binary data. For numbers in base two, shifting the data left n times multiplies the number by 2^n and shifting the data right n times divides the number by 2^n. Ex: The number 0101 (5 in binary) shifted left one position yields 1010 (10 in binary).

PARTICIPATION ACTIVITY 4.9.1: Shifting data.

Animation content:

undefined

Animation captions:

1. A binary number shifted left once multiplies the number by 2.
2. A binary number shifted right once divides the number by 2.
3. A binary number shifted left twice multiplies the number by 4, and a binary number shifted right twice divides the number by 4.

PARTICIPATION ACTIVITY 4.9.2: Shifting.

1) Shifting 0100 left one time results in 0010.
   - True
   - False
2) Eight shift operations are required to divide a number by 8.
   - True
   - False
3) Shifting 1100 right two times divides the number by 4.
   - True
   - False
4) Dividing a number by powers of two can be done using only shift operations.
   - True
   - False

Simple shifters

An N-bit ***shifter*** is a combinational logic component that can shift an N-bit input by a fixed amount. A simple shifter can shift an N-bit input to the left (or right) by one bit position. A shifter with a shift control input can either shift the N-bit input left or right by one position or pass the input through unchanged. Shifting left by one position is indicated by <<1, and shifting right by one position is indicated by >>1. Shifters can also be designed to support shift left by one position, shift right by one position, and pass through functionality.

PARTICIPATION ACTIVITY 4.9.3: Simple shifters.

Animation content:

undefined

Animation captions:

1. A simple left shifter shifts bits left by one position. The least significant bit of the output is assigned with the in input.
2. A shifter with shift left and pass through operations has a control input sh. The value of sh determines the mode of operation for the shifter.
3. If sh = 1, the shifter shifts the input bits one position to the left and sets the least significant bit to the input in.
4. If sh = 0, the shifter passes through the inputs without shifting.

PARTICIPATION ACTIVITY 4.9.4: Shifters.

1) A simple shift left shifter has no control input.
   - True
   - False

2) All shifters can shift the N-bit input left or right by one position.
   - True
   - False

3) For a simple 4-bit right shifter, what is output if the input i is 0100 and the shift input in is 0?
   - 0010
   - 0000

4) For a simple 4-bit left shifter, what is output if the input i is 1101 and the shift input in is 1?
   - 1010
   - 1011

5) For a 4-bit right shifter with a shift control input, what is output if the input i is 1011, sh = 0, and in = 0?

- 0101
- 1011

6) For an 8-bit left shifter with control input, what is output if the input i is 11101110, sh = 1, and in = 0?

- 11011100
- 11101110

Barrel shifter

A N-bit ***barrel shifter*** is a shifter component that can shift data by any number of positions. An N-bit barrel shifter uses $log_2 N$ shift control inputs to control the amount of the shift. Ex: An 8-bit barrel shifter can shift by 0, 1, 2, 3, 4, 5, 6, or 7 positions, requiring three control inputs. An 8-bit barrel shifter can be designed by cascading three shifters: a 4-bit shifter, a 2-bit shifter, and a 1-bit shifter.

PARTICIPATION ACTIVITY 4.9.5: Barrel shifters.

Animation content:

undefined

Animation captions:

1. A barrel shifter is designed by cascading multiple shifters. The numbers on the shifters indicate the number of positions each shifter will shift. << indicates left shift.
2. x = 1 enables a 4-bit shift, y = 1 enables a 2-bit shift, and z = 1 enables a 1-bit shift. The shifts add to one another, so if y and z = 1, the resulting shift is 3 bits.
3. To shift a number by 4 using an 8-bit barrel shifter, only the 4-bit shifter component is used by setting xyz = 100.
4. To shift a number by 7 using an 8-bit barrel shifter, xyz is set to 111, which sets all three shifters' sh input to 1.

PARTICIPATION ACTIVITY 4.9.6: Barrel shifters.

1) How many control inputs will a 16-bit barrel shifter need?

- None
- 4
-

8

2) What configuration of simple shifters are needed to design a 16-bit barrel shifter?

- 4, 2, 1
- 4, 4, 4, 4
- 8, 4, 2, 1

3) For an 8-bit barrel shifter (right shift only), if the input is 11000000, what is output if xyz = 101?

- 00000110
- 00001100
- 00000001

4) For an 8-bit barrel shifter (left shift only), if the input is 00000011, and the output is 00000110, what is xyz?

- 010
- 001
- 100

5) For an 8-bit barrel shifter (left shift only), if the input is 11000011, what is output if xyz = 110?

- 10000000
- 00000011
- 11000000

CHALLENGE ACTIVITY 4.9.1: Shifters.

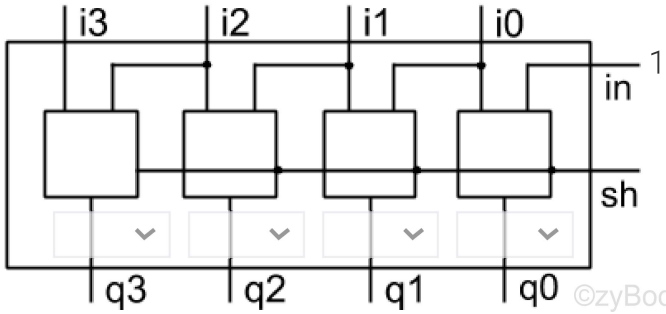
347136.1658324.qx3zqy7

Start

Select the values of q3-q0.

0 1 1 0

1



1 2

Check Next

4.10 Strength reduction

Strength reduction

Multiplication by a constant number that is a power of 2 can be done using left shifts, but systems commonly multiply by other constant numbers like 5 or 10. Although a multiplier can be used to implement such a circuit, multipliers use more transistors than shifters or adders. ***Strength reduction*** is a technique that replaces costly operations with a series of less costly operations. Strength reduction can be used for multiplication and division.

PARTICIPATION ACTIVITY 4.10.1: Strength reduction.

Animation content:

undefined

Animation captions:

1. Multiplication by a number that is a power of 2 (2, 4, 8, 16 etc.) can be done using left shifts. Each left shift by one position is equivalent to multiplying by 2.

2. 5*C can be rewritten as 4*C + C, thus requiring a left-shift-by-2 component (for 4*C) and an adder. The shift and adder combined are smaller and faster than a multiplier.

PARTICIPATION ACTIVITY 4.10.2: Strength reduction.

1) Which components are required to perform the operation C*68?
   - (C<<64) + (C<<4)
   - (C<<6) + (C<<2)
   - (C<<6) + (C<<4)

2) Given operation C*34, what is the minimum number of adders and shifters to perform the operation using strength reduction?
   - One shift left by 5 component
     Two shift left by 1 components
     Two adders
   - Two shift left by 4 components
     One shift left by 1 component
     Two adders
   - One shift left by 5 component
     One shift left by 1 component
     One adder

Example: Celsius to Fahrenheit converter.

Strength reduction can be used to implement a circuit that converts a temperature from Celsius to Fahrenheit.

PARTICIPATION ACTIVITY 4.10.3: Example: Celsius to Fahrenheit converter.

Animation content:

undefined

Animation captions:

1. A digital thermometer digitizes a temperature into an 8-bit unsigned binary number with C representing the temperature in Celsius. To convert to Fahrenheit, an equation is used.
2. The equation can be rewritten by using a strength reduction approach that only uses multiplications or divisions by powers of 2, which can be replaced by left and right shifts, respectively.
3. Consider an input C = 00011110 representing 30 degrees Celsius. Shifting left by six positions results in information loss. So, input C is padded with zeros to create a 16-bit representation.
4. The leftmost 8 bits of the final addition can be dropped to yield the 8-bit output. F is 01010101, which is decimal value 85.

PARTICIPATION ACTIVITY 4.10.4: Example: Celsius to Fahrenheit converter.

Consider the example above.

1) Why is the number of bits for C (00011110) increased from 8 bits to 16 bits?
   - Increasing to 16 bits makes the circuit smaller.
   - To avoid information loss.

2) Can strength reduction be applied to circuits that implement division?
   - No
   - Yes

4.11 Example: Above-mirror display using shift registers

An above-mirror display system can be designed using shift registers to reduce the number of wires needed to connect the car's computer with the display. The number of wires can be reduced from 11 wires (for a system with parallel load registers) to 4 wires.

PARTICIPATION ACTIVITY 4.11.1: Above-mirror display using shift registers.

Animation content:

```
undefined
```

Animation captions:

1. A single data bit (C) is used along with shift registers to send the data from the car's computer to the display.
2. In addition to C, two address lines a1a0 and a shift control input are used to shift data into the registers. A total of only 4 wires are needed.
3. When the computer needs to write to a register, a1a0 is set appropriately and shift is set to 1 for exactly 8 clock cycles. During each clock cycle, the computer will set C to one bit of the 8-bit data to be loaded.
4. If xy = 10, the value of reg2 is output to the display.

PARTICIPATION ACTIVITY 4.11.2: Above-mirror display with shift registers.

1) Using a shift register enables _____.

- serial communication
- parallel communication

2) Since the data input C is one bit wide, the mux used in the above example can be replaced with a 1-bit 4x1 mux.

- True
- False

4.12 Counters and timers

Up-counters

A N-bit ***counter*** is a datapath component that can increment or decrement an N-bit count value on each clock cycle.

An N-bit ***up-counter*** increases the counter stored value by one every clock cycle. When an up-counter reaches the highest value, the counter wraps around to 0 and continues counting. Ex: The sequence for a 4-bit up-counter is 0000, 0001, 0010, 0011, 0100, ..., 1110, 1111, 0000, 0001, etc.

If the control input *clr* = 1, the value in the register synchronously clears, setting the count value to 0. If clr = 0 and the control input *cnt* = 1, the counter is enabled and counts up each clock cycles. A

counter's ***terminal count***, or tc, output is 1 when the counter has reached the last (or terminal) value before wrapping around.

PARTICIPATION ACTIVITY 4.12.1: 4-bit up-counter.

Animation content:

undefined

Animation captions:

1. A 4-bit up-counter has a 4-bit register to store the count.
2. An incrementer is used to increment the count by 1 during every clock cycle.
3. The terminal count control, tc, output uses an AND gate to determine the highest count value (1111).
4. If clr = 1, the register's value is cleared and count C becomes 0.
5. Otherwise, if cnt = 1, the register is loaded with the increment's output, thereby increasing the count by one.
6. A 4-bit up-counter counts up until the highest value and then rolls over.

PARTICIPATION ACTIVITY 4.12.2: Up-counter.

Consider a 4-bit up-counter, unless otherwise specified.

1) Given C = 1001, cnt = 1, and clr = 0, what is the value of the counter at the next clock cycle?

Check Show answer

2) Given C = 1111, cnt = 1, and clr = 0, what is the value of the counter at the next clock cycle?

Check Show answer

3) Given C = 0011, cnt = 1, and clr = 1, what is the value of the counter at the next clock cycle?

Check Show answer

4) Given C = 0110, cnt = 0, and clr = 0, what is the value of the counter at the next clock cycle?

Check Show answer

5) For an 8-bit up-counter, what is the counter value at which tc is set to 1?

Check Show answer

Down counters

An N-bit ***down-counter*** decreases the counter's stored value by one every clock cycle. A down-counter's terminal count is 1 when the count is 0, after which the counter's value will wrap around to the highest N-bit value. A down-counter's design is similar to an up-counter with two main changes:

1. Use a decrementer in place of the incrementer
2. Use a NOR gate for the terminal count to detect when the counter value is 0

PARTICIPATION ACTIVITY 4.12.3: 4-bit down-counter.

Animation content:

undefined

Animation captions:

1. A down-counter has a component that decrements the count by 1 every clock cycle.
2. A NOR gate sets tc to 1 when the count is 0000, after which the counter wraps around.
3. cnt = 1, enables counting by loading the output of the decrement component into the register.
4. A 4-bit down-counter starts at the highest value (1111 for a 4-bit down counter), counts down until it reaches 0, after which it wraps around.

PARTICIPATION ACTIVITY 4.12.4: 4-bit down counter.

Given the values for C, cnt, and clr, what is the value of the counter at the next clock cycle?

1) C = 1001
cnt = 1
clr = 0
   - 1010
   - 1000

2) C = 1111
cnt = 1
clr = 0
   - 1110
   - 0000

3) C = 0000
cnt = 1
clr = 1
   - 1111
   - 0000

4) C = 0000
cnt = 1
clr = 0
   - 0001
   - 1111

5) C = 0101
cnt = 0
clr = 0
   - 0101
   - 0100

Counters with load input

A counter with a load input enables a counter to start counting from a particular input value by loading the counter's register with that value.

PARTICIPATION ACTIVITY 4.12.5: 4-bit down-counter with a load input.

Animation content:

undefined

Animation captions:

1. A counter with a load input has a 4-bit 2x1 mux that enables the counter to be loaded with an input value, L.
2. When ld = 1, the register is loaded with the input value L, and the counter starts counting down from that value, in this case 0100, or 4.
3. If ld = 0 and cnt = 1, the counter counts from 4 down to 0, wraps around to 15, and then keeps counting down until one of the control inputs change.

PARTICIPATION ACTIVITY 4.12.6: 4-bit counter with a load input.

1) An up-counter can be designed to have a load input.
   - True
   - False

2) The terminal count output is unused for down-counters.
   - True
   - False

3) A 32-bit up-counter requires multiple muxes to support a load input.
   - True
   - False

Timers

A ***timer*** is a sequential component that generates a short pulse at a specific time intervals. A timer uses a down-counter, a register, and a base time unit. The base time unit, which is usually the clock period of an oscillator, defines the rate at which the counter counts down. The register holds the value that will be loaded into the counter after 0 is reached. Ex: If the timer's base unit is 1 microsecond, and a user wants the timer to pulse every 20 microseconds, the user loads 20 into the timer. If a pulse is required every 1 millisecond, the user loads 1000 into the timer.

A timer's bitwidth, also known as width, determines the maximum time interval that the timer can achieve. Ex: An 8-bit timer with a base time unit of 1 millisecond, has a maximum time interval of 2^8

milliseconds, or 256 milliseconds.

PARTICIPATION ACTIVITY 4.12.7: A 32-bit timer with a 1 microsecond base time unit.

Animation content:

undefined

Animation captions:

1. A timer uses a down-counter, a base time unit specifying the counting rate, and a register that holds the value that will be loaded into the counter after 0 is reached.
2. When ld = 1, the timer's register is loaded with M - 1, since the down-counter's count will include 0.
3. enable = 1 causes the counter to count down at 1 microsecond intervals. The timer's output Q is connected to the counter's tc output and outputs 1 when the counter reaches 0.
4. When tc = 1, the down-counter is loaded with the value stored in the timer's register. Every 50 microseconds, this timer outputs a 1 for 1 microsecond.

PARTICIPATION ACTIVITY 4.12.8: Timers.

1) Given a timer with a base time unit of 1 second, what number must be loaded into the timer to generate a pulse every 60 seconds?

Check Show answer

2) Given a timer with a base time unit of 1 microsecond, what number must be loaded into the timer to generate a pulse every 30 milliseconds?

Check Show answer

3) What is the maximum time interval in microseconds that a 4-bit timer

can achieve? Assume the base unit is 1 microsecond.

Check Show answer

4) What is the maximum time interval in milliseconds that a 16-bit timer can achieve? Assume the base unit is 1 millisecond.

Check Show answer

4.13 Example: Laser surgery system using a timer

A previous example introduced a laser surgery system using only an FSM. That system required the laser to stay on for exactly 30 ns. The system's clock period was 10 ns, so the laser needs to be on for 3 clock cycles. The FSM implementation had three states and a wait state.

If the time the laser is on needs to be increased to 300 milliseconds, the FSM implementation would require introducing 30 million states. Instead, the system can be designed by connecting a controller to a 32-bit microsecond timer.

PARTICIPATION ACTIVITY 4.13.1: Laser surgery system with an external timer.

Animation content:

undefined

Animation captions:

1. A controller is connected to a 32-bit 1 microsecond timer. The controller's inputs are the button press (b), and the timer's output Q. The controller's outputs are load and enable to the timer, and x to the laser.
2. The controller is captured as an FSM. The Off state loads the timer with 300,000 (300,000 microseconds = 300 milliseconds). On a button press, b = 1, the FSM enters the Start state.

The Start state enables the timer.

3. On the next clock cycle, the FSM enters the On State, which keeps the timer enabled and sets x = 1 to turn on the laser. After 300 ms, the timer's output Q = 1, and the FSM enters the Off state, setting x = 0.

PARTICIPATION ACTIVITY 4.13.2: Laser surgery system using a timer.

1) If the desired time interval for the laser is 45 microseconds, what number must be loaded into the timer?
   - 45,000
   - 45

2) If the desired time interval for the laser is 5 milliseconds, what number must be loaded into the timer?
   - 5
   - 5000

3) In the above system, the laser is on for _____.
   - 300 ms
   - 300 ms + 10 ns

4.14 Multipliers (array-style)

Assuming unsigned binary numbers, an N-bit ***multiplier*** multiplies two N-bit numbers to yield a 2N-bit product. A multiplier can be designed by mimicking multiplication by hand. Each multiplier digit is multiplied with the multiplicand and appended with 0's based on the digit's location, yielding a partial product, all of which are summed to yield the product.

PARTICIPATION ACTIVITY 4.14.1: 4-bit array-style multiplier.

Animation captions:

1. Multiply multiplicand with the rightmost digit of the multiplier to determine the first partial product: 0110 x 1 = 0110.

2. Multiply multiplicand with the next digit of the multiplier: 0110 x 0 = 0000. Append one 0 to the partial product.
3. Continue for the remaining multiplier digits. Add partial products to determine the product.
4. Multiplying two 4-bit numbers can be broken down into smaller 1-bit multiplications, followed by addition.
5. 1-bit multiplication is implemented with an AND gate.
6. An array of AND gates calculates the partial products.
7. Partial products are added together to determine the product.
8. A multiplier is commonly represented as a block symbol.

PARTICIPATION ACTIVITY 4.14.2: Multiplying binary numbers.

Multiply the following 3-bit values. Include any leading zeros.

```
  010
x 110
---------
```

1) pp1 = ???

Check Show answer

2) pp2 = ????

Check Show answer

3) pp3 = ?????

Check Show answer

4) product = ??????

Check Show answer

PARTICIPATION ACTIVITY 4.14.3: Multiplier.

Assume a 4-bit array-style multiplier. (Note: Assume every gate input requires 2 transistors and ignore inverters.)

1) Which component performs 1-bit multiplication?
   - AND gate
   - Adder

2) The product is the sum of partial products.
   - True
   - False

3) What is the size (in transistors) used to compute all partial products?
   - 16
   - 64

4) What is the size (in transistors) of a 5-bit carry-ripple adder built from full adders, if a full adder has 50 transistors?
   - 100
   - 250

5) What is close in size (in transistors) of a 4-bit array-style multiplier? Assume: 6-bit adder has 300 transistors, 7-bit adder has 350.
   - 250
   - 1000

6) What is the delay (in gate-delays) to compute all partial products?
   - 1
   - 4

7) What is the delay (in gate-delays) of a 5-bit carry-ripple adder built from full adders, assuming a full adder's delay is 2 gate-delays?
   - 2
   -

10

8) Which is closer to the total delay of a 4-bit array-style multiplier?

- 10
- 15

4.15 Product Profile: Ultrasound

Ultrasound basics

An ultrasound image of a fetus is created by placing an ultrasound device on the mother's abdomen. Ultrasound machines rely heavily on fast digital circuits to generate sound waves, listen to the echoes, and process the echo data to generate images in real time. The term ultrasound refers to the fact that the frequency is beyond human hearing, typically 2 to 15 MHz. Human hearing is typically in the range of 20 Hz to 20 kHz. The ultrasound example shown is greatly simplified from a real machine.

PARTICIPATION ACTIVITY 4.15.1: Ultrasound imaging basics.

Animation content:

undefined

Animation captions:

1. Ultrasound imaging works by sending sound waves into the body and listening to the echoes that return.
2. Objects like bones yield different echoes than skin and fluids so an ultrasound machine processes the different echoes to generate images. Strong echoes might be displayed as white, and weak echoes as black.
3. A transducer converts electrical pulses into sound pulses, and vice versa using piezoelectric crystals. Applying electric current to the crystals causes vibrations, generating sound waves.
4. A beamformer component electrically focuses the sound beam using an array of transducers to or from particular focal points.
5. A signal processor analyzes echo data of every point in the scanned region and assigns a level of gray to each point depending on the echoes heard. The scan converter generates necessary signals for the monitor.

PARTICIPATION ACTIVITY 4.15.2: Ultrasound basics.

1) An ultrasound machine processes the different echoes from bones, skin, and fluids to build an image.
   - True
   - False
2) An ultrasound machine generates sound waves in the range of 20 Hz to 20 kHz.
   - True
   - False

Additive sound

A beamformer works on the principle of additive sound. Using this principle, a sound beam can be electronically focused by introducing appropriate delays. Focusing the sound to a particular point is useful because then that point will produce a much louder echo than all other points.

PARTICIPATION ACTIVITY 4.15.3: Additive sound.

Animation content:

undefined

Animation captions:

1. Consider two loud fireworks exploding at the same time, one 1 mile away and the other 2 miles away. Assuming sound travels at 0.2 miles/second, A will be heard in 5 s, and B in 10 s.
2. Instead, if A exploded 5 seconds after B, both A and B will be heard at the same time, resulting in one big boom. The two sounds add together to generate a louder sound.

PARTICIPATION ACTIVITY 4.15.4: Additive sound.

1) In the above example, if firework B was 4 miles away and firework A was 1 mile away, which how many seconds must A be delayed to produce a louder sound?

Check Show answer

Beamformer

The ultrasound machine's transducer component contains an array of hundreds of crystals. Each crystal can be thought of as a separate transducer. Applying an electric current to the crystal causes the crystal to change shape rapidly and vibrate, thus generating sound waves. Conversely, sound waves hitting the crystal create an electrical current. The sound from multiple sound sources (transducers of an ultrasound machine) in different locations, can be added together by carefully timing the generation of sound from each source.

PARTICIPATION ACTIVITY 4.15.5: Beamformer basics.

Animation content:

undefined

Animation captions:

1. A beamformer uses two transducers A and B to focus a beam on focal point X. A is closer to focal point than B, and should generate sound later than B. So, at the first time step, only transducer B generates sound.
2. At the second time step, transducer A generates sound.
3. At the third time step, the two sound waves add at the focal point.
4. The sounds then echoes back to the transducers. At the first time step, the echo is generated. At the second time step, the transducer A hears the echo. At the third time step, transducer B hears the echo.
5. Delaying the top transducer by one time step results in echo waves from the focal point adding, amplifying the sound. Echoes from other points will be weaker than echoes from the focal point and can be filtered out.

PARTICIPATION ACTIVITY 4.15.6: Beamformer basics.

1) When listening to sound from a particular point using a beamformer, adding delays can ____.
   - filter the noise.
   - amplify the sound.

2) The beamformer only hears echoes from the focal point.

- True
- False

3) Consider the above example. If transducer A hears the sound at the second time step, and transducer B hears the sound at the fourth time step, how much delay must be introduced to the top transducer to achieve sound amplification?

- 1 time step
- 2 time steps

Beamforming applications

Beamforming is common in a wide variety of sonar applications, such as observing a fetus, observing a human heart, searching for oil underground, monitoring the surroundings of a submarine, etc. Beamforming is used in some hearing aids having multiple microphones, to focus on the source of detected speech. Beamforming can be used in multi-microphone cell phones to focus on the user's voice, and can even be used in cellular telephone base stations to focus a signal going to or coming from a cell phone.

Sound generation circuit

Much of the control and signal processing tasks in an ultrasound machine are implemented using software running on one or more special microprocessors, specifically designed for digital signal processing, known as digital signal processors or DSPs. But some tasks like the beamformer are more amenable to custom digital circuits. The beamformer's sound generation circuit focuses the beam produced by the transducer by precisely controlling the delay between when each transducer is enabled.

PARTICIPATION ACTIVITY 4.15.7: Sound generation circuits.

Animation content:

```
undefined
```

Animation captions:

1. The sound generation circuit implements a delay circuit for each transducer output. For a given focal point, the DSP writes the appropriate delay value into each delay circuit using delay_out.
2. The DSP writes the delay to each transducer addr and wr, which connect to a decoder. The decoder sets the load line of the corresponding OutDelay component.
3. After writing the delay for each transducer, the DSP starts all OutDelay components by setting start_out to 1.
4. OutDelay is implemented using a down counter with a parallel load. The load inputs L and ld load the counter with the delay value and count down once started.
5. When tc reaches 0, each OutDelay generates a pulse on the output line o that connects to the corresponding transducer. The DSP then sets the start_out line to 0 and listens for the echo.

PARTICIPATION ACTIVITY 4.15.8: Sound generation circuits.

1) The beamformer circuit of an ultrasound machine has _____.
   - ○ one OutDelay component per ultrasound machine
   - ○ an OutDelay component for each transducer

2) The OutDelay component uses a down counter _____.
   - ○ to introduce a delay in the sound pulse
   - ○ to listen for an echo

Listening: Echo delay circuit

After the ultrasound machine sends out sound waves focused on a particular focal point, the machine listens for the echo coming back from the focal point. The echo delay circuit enables listening by introducing appropriate delays for each transducer to account for the differing distances of each transducer from the focal point.

PARTICIPATION ACTIVITY 4.15.9: Echo delay circuits.

Animation content:

undefined

Animation captions:

1. After generating the sound beam, the ultrasound listens for the echo from the focal point. Listening requires delays to account for the different distances between the transducers and the focal point.
2. The EchoDelay component using a series of registers to introduce various delays. The EchoDelay component here uses three registers to support delays of 0, 1, 2, or 3 clock cycles.
3. The DSP configures the delay amount by writing to a delay control register. The delay control register's output connects to the 4x1 mux select lines.
4. The EchoDelay component receives the t signal from the transducer and outputs the appropriately delayed signal t_delayed, which then connects to adders to be summed.

PARTICIPATION ACTIVITY 4.15.10: Echo delay circuit.

1) Which component configures the amount of delay for the t_delayed signal?
   - Transducer
   - EchoDelay component
   - DSP

2) If the EchoDelay component has 7 registers to support more delays, what mux configuration is required?
   - 4x1 mux
   - 7x1 mux
   - 8x1 mux

Multipliers and summation circuits

The appropriately delayed output of each transducer is summed to create a single echo signal from the focal point. For an ultrasound machine with hundreds of transducers, the addition can be efficiently implemented using an adder tree. The output of the adder tree is written to memory to keep track of the results for the DSP, which may access the results after the results are generated.

PARTICIPATION ACTIVITY 4.15.11: Multipliers and adder tree.

Animation content:

undefined

Animation captions:

1. The DSP writes a constant value to a register for each transducer that is multiplied with the transducer signal.
2. The t_delayed*c signals from all transducers are summed to create a single echo signal from the focal point. For 8 transducers, linearly adding the signals results in a 7-adder delay.
3. The summation can be more efficiently implemented using an adder tree, which uses the same number of adders but reduces the delay. For 8 transducers, the adder tree has only a 3-adder delay.

PARTICIPATION ACTIVITY 4.15.12: Sound generation and echo delay circuits.

1) How many adders are needed to sum 16 transducer signals?
   - 4
   - 15
   - 32

2) How many adder delays occur when adding 16 transducer signals using a linear adder arrangement?
   - 4-adder delay
   - 15-adder delay
   - 32-adder delay

3) How many adderdelays occur when adding 16 transducer signals using an adder tree?
   - 3-adder delay
   - 4-adder delay
   - 15-adder delay

©zyBooks 10/17/21 22:04 829162
Ivan Neto
UCRCS120AEE120AChenFall2021

4) If each adder has a delay of 4 ns, what is the total delay for adding 256 transducer signals using an adder tree?
   -

4 ns

- 32 ns
- 1020 ns

3-D Ultrasound

One of the main trends in ultrasound machines involves creating three dimensional (3-D) images in real time. In contrast to 2-D ultrasounds, generating 3-D images requires viewing the region of interest from different perspectives, just like people view things from their two eyes. Such generation also requires extensive computations to create a 3-D image from the two (or more) perspectives. The computations take time, but faster processors coupled with custom digital circuits help achieve 3-D images.

©zyBooks 10/17/21 22:04 829162
Ivan Neto
UCRCS120AEE120AChenFall2021