10.1 Gray code

Gray code: Only one bit changes

Some systems put a sequence of bit codes on wires, such as counting up on three wires as in: 000, 001, 010, 011, 100, 101, 110, 111. Some problems exist for such sequences:

- When the next value changes two (or more) bits, a wrong value may briefly appear due to varying wire/gate delays for each bit. Ex: Instead of a change from 001 to 010, a change may be from 001 to 000 (briefly) to 010, due to a slower-changing middle bit.
- Due to the nature of digital circuitry, each bit change consumes power.

For either problem, a sequence that minimizes bit changes is desirable. ***Gray code*** is a bit encoding for a value sequence where successive values differ by only one bit, named for its inventor Frank Gray. A 3-bit Gray code is 000, 001, 011, 010, 110, 111, 101, 100.

PARTICIPATION ACTIVITY 10.1.1: Multiple bit changes may cause briefly wrong values due to different wire/gate delays.

Animation content:

Two devices are connected with wires a, b, c. On the left is a table counting up in binary: 000, 001, ..., 110, 111. A timing diagram shows cba values 000, then 001, then 010. Then b signal's change from 0 to 1 is shown delayed slightly in red, causing the cba values to actually be 000, 001, 000, 010. Then, on the right is shown another table counting up in Gray code: 000, 001, 011, 010, 110, 111, 101, 100. Another timing diagram shows cba as 000, 001, then 011. b's change from 0 to 1 is again shown delayed, but that just shifts the 011 to the right; no briefly wrong value appears.

Animation captions:

1. One device may transmit values to another. Sometimes those values are a sequence, like 000, 001, 010, etc.
2. Ideally, bit changes occur simultaneously. But different wire/gate delays cause some bits to change more slowly, briefly yielding a wrong value, like 001 to 000 (briefly) to 010.
3. In Gray code, successive values differ in only one bit. Since two bits never change simultaneously, a delay doesn't cause a briefly wrong value.

PARTICIPATION ACTIVITY 10.1.2: Gray code.

1) In Gray code, which value can follow 011?

- 100
- 000
- 010

2) In Gray code, which value can follow 0100?

- 1011
- 1100

3) Is the following a valid Gray code? 00, 01, 11, 10

- Yes
- No

4) Gray code wraps around, meaning the last value and first value also differ by only one bit. Do the first and last values of this encoding obey Gray code? 000, 001, ..., 101, 100

- Yes
- No

Gray code uses

One use of Gray code is to sense the position of a rotating device, like a wheel, disk drive, or motor. For eight positions, three sensors can detect whether a magnet is below. The magnets can be configured so successive positions differ by one bit.

PARTICIPATION ACTIVITY 10.1.3: Gray code use example: Sensing a wheel's position.

Animation content:

A wheel with three tracks of colored strips is shown. Each 1/8th of the perimeter has a different combination of colored strips. The first (at the top right) has three uncolored strips, and is labeled 000. Going clockwise, the next has the outermost track colored, and is labeled 001. The next has the outermost two tracks colored, and is labeled 011. And so on, following Gray code. Three green circles are shown over the three tracks, labeled Sensors.

Next, the wheel rotates, starting with tracks colored, non-colored, non-colored under the sensors and 100 being output, next 000, and finally 001.

Animation captions:

1. A wheel may have magnets that identify one of eight positions. Sensors sense those magnets, yielding a value indicating the wheel's current rotated position.
2. As the wheel rotates, the detected values change by only one bit, avoiding briefly wrong values (due perhaps to different sensor/wire delays).

PARTICIPATION ACTIVITY 10.1.4: Gray code in a wheel position sensor.

Consider the animation above. Assume the wheel rotates counterclockwise, as in the animation.

1) How many circular "tracks" exist where metal can appear on the wheel?

Check Show answer

2) What color is used in the animation to indicate a 1? Valid answers: black, grey.

Check Show answer

3) If the wheel's position is currently detected as 001, what will be the next position detected?

Check Show answer

4) One rotation of the wheel from 000 to 100 yields how many bit changes?

Check Show answer

5) If a sensor is slightly slower than another sensor (but the delay is much less than 1/8th of the wheel's rotation), is this sequence likely? 001 to 010 (wrong) to 011. Type yes or no.

Check Show answer

Converting to/from Gray code

If a sending device will transmit a sequence of binary values, the device may wish to first convert those binary values to Gray code values. Conversely, a receiver may need to convert Gray code values to binary values. For values involving small numbers of bits, such conversion is straightforwardly done by starting with a truth table, and proceeding with standard combinational circuit design.

PARTICIPATION ACTIVITY 10.1.5: Converting to/from Gray code with a truth table.

Animation content:

undefined

Animation captions:

1. Binary can be converted to Gray code by starting with a truth table.
2. Obviously, c = z. For b, the minterms can be simplified to b = z'y + zy'. Similarly, a can be simplified to a = y'x + yx'.
3. For Gray to binary, the table is drawn in reverse. The rows can be reordered to have increasing input values, as is usual. Expressions for z, y, x can then be derived.

Actually, a pattern emerges where Gray code bits can be obtained using a series of XORs, but that topic is beyond this material's scope.

PARTICIPATION ACTIVITY 10.1.6: Converting to/from Gray code.

Consider the animation above.

1) For the conversion of Gray code to

binary (the rightmost table above), what is the obvious equation for z?

Check Show answer

Building a Gray code

A three-bit Gray code can be built from the two-bit Gray code of 00, 01, 11, 10:

1. Two-bit Gray code values are appended in reverse: 00, 01, 11, 10, *10, 11, 01, 00*.
2. 0s are prepended to the first half, 1s to the second half: 000, 001, 011, 010, 110, 111, 101, 100.

Likewise, a four-bit Gray code can be built from a three-bit Gray code:

1. Append in reverse: 000, 001, 011, 010, 110, 111, 101, 100, *100, 101, 111, 110, 010, 011, 001, 000*
2. Prepend 0s to first half, 1s to second half: 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000

An N-bit Gray code can be built just by repeating the above process until reaching N bits.

PARTICIPATION ACTIVITY 10.1.7: Building a Gray code: Step 1.

Given that a two-bit Gray code is 00, 01, 11, 10, in Step 1 of creating a 3-bit Gray code, order the four values that should be appended to form the 5th, 6th, 7th, and 8th values in the list.

01 10 11 00

5th

6th

7th

8th

Reset

PARTICIPATION ACTIVITY 10.1.8: Building a Gray code: Step 2.

1) In building a three-bit Gray code from a two-bit Gray code, Step 1 creates a list of eight values: 00, 01, 11, 10, 10, 11, 01, 00. In Step 2, what bit is prepended to the first four of those values?
   - 0
   - 1

2) Does the three-bit Gray code built using the above technique wrap around, meaning the last code and first code also differ in only one bit?
   - No
   - Yes

10.2 JK and T latches and flip-flops

JK latch and JK flip-flop

A ***JK latch*** stores a bit and supports operations to set the bit 1, reset the bit to 0, toggle the bit, or maintain the stored value. The JK latch's operation is controlled by two inputs j and k. When j = 1 and k = 0, the bit is set to 1. When k = 1 and j = 0, the bit is reset to 0. When j and k are both 0, the latch maintains the previously stored q value. When j and k are both 1, the latch toggles the previously stored q value.

An edge-triggered JK flip-flop can be implemented by cascading two JK latches.

PARTICIPATION ACTIVITY 10.2.1: JK flip-flop.

Animation content:

`undefined`

Animation captions:

1. A JK latch can be implemented using an SR latch, two AND gates, and an enable input.
2. Assume the JK latch was previously set. When en = 1, if j and k are both 0, q remains 1, and q' remains 0. So the latch maintains the previous value of q.
3. If j = 0 and k = 1, q is 0 and q' is 1. So the latch is reset to 0.

4. If j = 1 and k = 0, q is set to 1 and q' is 0. So the latch is set to 1.
5. If j and k are both 1, q changes from a 1 to a 0, and q' changes from a 0 to a 1. So the latch toggles the value of q.
6. A JK flip-flop can be implemented by cascading two JK latches.

PARTICIPATION ACTIVITY 10.2.2: JK latch.

Indicate q's present value for the given input sequence. "j: 0..1" means j was 0 and is presently 1.

1) j: 0
k: 0
Previous q: 0
   - 0
   - 1

2) j: 1
k: 1
Previous q: 0
   - 0
   - 1

3) j: 0..1
k: 0..0
q: 0..?
   - 0
   - 1

4) j: 1..0
k: 1..1
q: 1..?
   - 0
   - 1

T latch and T flip-flop

A ***T latch*** stores a bit and toggles the stored bit when the input t is 1. When t is 0, the stored bit does not change. A ***T flip-flop*** stores a bit and will toggle the stored bit on a rising clock edge when the input t is 1.

PARTICIPATION

ACTIVITY 10.2.3: T latch.

Animation content:

undefined

Animation captions:

1. A T latch can be implemented using an SR latch, two AND gates, and an enable input.
2. Assume the T latch was previously set, so q = 1. If t = 0, q remains 1. So the T latch maintains the previously-stored value of q.
3. If t = 1, q is toggled from 1 to 0. So the T latch toggles the previously-stored value of q.
4. A T flip-flop toggles the stored bit on a rising clock edge when input t is 1.

PARTICIPATION ACTIVITY 10.2.4: T latch.

Indicate q's present value for the given input sequence. "t: 0..1" means t was 0 and is presently 1.

1) t: 0
Previous q: 0
   - 0
   - 1

2) t: 1
Previous q: 0
   - 0
   - 1

3) t: 0..1..1..1
q: 1..?..?..?
   - 0..0..0
   - 0..1..0

A bit of history - SR, JK, T and D latches and flip-flops

In the 1980s, transistors on ICs were more costly and scarcer than today. Designing a system using a D flip-flop uses more transistors than an SR latch based design. The system may require more external circuitry to set D to the appropriate value. For a given desired behavior, using a particular flip-flop type could save transistors. Designing

sequential circuits for any flip-flop type was a challenging task, involving excitation tables and comparison of different designs, and was helpful for reducing circuit transistors. But today, in the era of billion-transistor ICs, the savings of such flip-flops are trivial. Nearly all modern sequential circuits use D flip-flops and hence are created using the more straightforward design process introduced in this zyBook.

©zyBooks 10/17/21 22:11 829162
Ivan Neto
UCRCS120AEE120AChenFall2021