How Altitude Encoders Use Gillham Code

by Ken Burrell

Transponders are a crucial part of aviation safety, and are required to be installed in every aircraft that operates in airspace managed by Air Traffic Control.

Transponders broadcast to Air Traffic Control the barometric pressure altitude of the aircraft, by a radio transmission in the 1090MHz band.

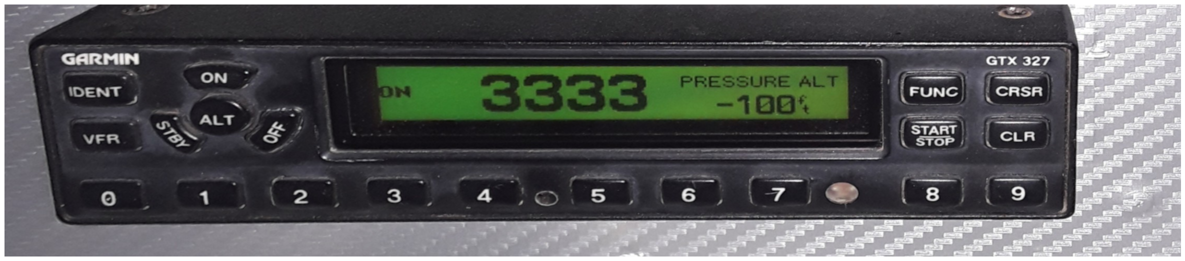

Figure 1. Garmin GTX327 Transponder

Transponders first obtain the altitude from an Altitude Encoder, which sends it over copper wires to the Transponder, by using serial or parallel encoding.

If the data encoding is serial, then the altitude data is transmitted using the RS-232 protocol, over twisted pair wires. If the data encoding is parallel, then data is transmitted over 9, 10, or 11 parallel, shielded wires, utilizing a special form of Gray code known as Gillham code.

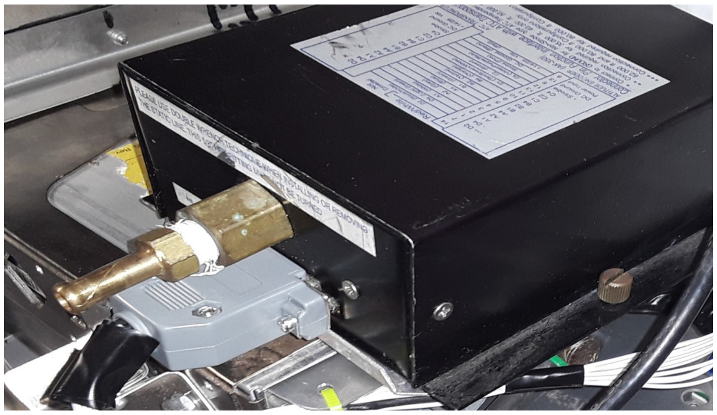

Figure 2. Literally a black box, the Altitude Encoder located behind the instrument panel. Note the brass nipple that connects to the outside air via a plastic tube, and the grey colored DB-15 connector that transmits either serial or parallel data to the Transponder over a bundle of wires.

Both RS-232 and Gillham encoding are widely used today, with Gillham code most widely used by non-commercial, general aviation aircraft. To obtain the altitude, the Altitude Encoder measures the instantaneous barometric pressure, and from that computes altitude in feet, based on the assumption that sea level atmospheric pressure is exactly 29.921 inches of Mercury.

The altitude is usually rounded to the nearest 100 feet, after computing it from this formula, or an equivalent electromechanical circuit:

$$\text{Altitude (in feet)} = 145366.45 * [\ 1 - (\ \mathbf{Barometric}_{\mathbf{pressure}}/\ 29.921)^{0.190284}\ ]$$

For example, if $\mathbf{Barometric}_{\mathbf{pressure}}$ measured by the Altitude Encoder is 29.921 inches of Mercury, then the Altitude will be zero feet, or Sea Level.

If the $\mathbf{Barometric}_{\mathbf{pressure}}$ measured is 29.800 in of Hg, then the Altitude will be

$$\text{Altitude} = 145366.45 * [\ \ 1 - (\ 29.800/29.921)^{0.190284}\ ] = 112.04 \text{ feet}$$

which is usually just rounded to 100 feet. If the result had been 150 feet, it would have been rounded to 200 feet.

The Altitude Encoder sends uncorrected altitude data to the Transponder, because it does not use the actual sea level pressure, but rather the ideal sea level pressure of 29.921 in of Hg, which is seldom the same as the actual sea level pressure.

When all aircraft in a given area transmit uncorrected altitude data, it provides solid information about the relative vertical separation of those aircraft. That avoids the errors that would result from each aircraft using by mistake or design, a different pressure correction.

Once the uncorrected altitude is obtained by the Altitude Encoder, it first converts that altitude to Gillham code, before sending it over the wires to the Transponder.

Gillham code is a special form of what is known as Gray code.

Gray code is a type of number representation in which the binary digits of a decimal number are arranged by noise reduction, rather than by powers of 2. For example, in the case of a powers-of-2 binary representation of a decimal number, the position of each digit represents a power of 2, as seen in Figure 3.

Decimal	**powers-of-2 binary**	**Decoding**
0	0 0 0	$0 + 0 + 0 = 0$
1	0 0 1	$0 + 0 + 2^0 = 1$
2	0 1 0	$0 + 2^1 + 0 = 2$
3	0 1 1	$0 + 2^1 + 2^0 = 3$
4	1 0 0	$2^2 + 0 + 0 = 4$
5	1 0 1	$2^2 + 0 + 2^0 = 5$
6	1 1 0	$2^2 + 2^1 + 0 = 6$
7	1 1 1	$2^2 + 2^1 + 2^0 = 7$

Figure 3. Decimal number 0 through 7 as represented by 3 binary digits, using the powers of 2 convention.

But, that is not the only way to represent a decimal number with binary digits.

When binary data is being represented electronically, as it is when being transmitted from an Altitude Encoder to a Transponder, we would like to reduce the possibility of error, or noise, in the signal.

Note in Figure 3, when going from decimal 0 to 1, there is only a change in 1 binary digit, in the place assigned to 2^0. But, in going from 1 to 2, there is a change in 2 binary digits, in the places assigned to 2^0 and 2^1.

The worst case is in going from decimal 3 to 4, in which all three binary digits have changed!

Gray code is a way of representing decimal numbers in binary digits, such that the change in the number of binary digits is minimized for each successive number being represented.

Decimal	**Gray code binary**	**powers-of-two binary**
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 1	0 1 0
3	0 1 0	0 1 1
4	1 1 0	1 0 0
5	1 1 1	1 0 1
6	1 0 1	1 1 0
7	1 0 0	1 1 1

Figure 4. Gray code binary compared to powers of 2 binary.

Note that Gray code is arranged so that each number is only 1 binary digit different from its neighbor. The change from decimal 3 to 4 now involves only a change in ***one*** binary digit, instead of ***three*** binary digits as in the powers-of-2 representation.

Obviously, Gray code doesn't use a powers-of-2 representation as demonstrated in Figure 4.

It is actually fairly easy to convert powers-of-2 binary to Gray code binary by two step process:

Step 1: shift all the bits of the power-of-2 binary to the right by 1 bit, deleting the rightmost bit, and inserting a zero at leftmost bit position.

Decimal 7 (1 1 1) shifted 1 bit right becomes: (0 1 1)

Decimal 6 (1 1 0) shifted 1 bit right becomes: (0 1 1)

Decimal 5 (1 0 1) shifted 1 bit right becomes: (0 1 0)

Decimal 4 (1 0 0) shifted 1 bit right becomes: (0 1 0)

Figure 5. Examples of shifting powers-of-2 binary numbers to the right by 1 bit.

Step 2: perform an exclusive OR (symbol **XOR**) comparison of the unshifted and shifted bits, and the resulting bits will be the Gray code representation. An ***exclusive OR*** (symbolized as **XOR**), is a binary-bit operation that compares the bits of two numbers, and outputs a **1** when the corresponding bits are different, and a **0** when they are the same.

A power-of-2 binary 7 (1 1 1) becomes Gray code (1 0 0).

Decimal 7 (1 1 1) shifted 1 bit right becomes: (0 1 1)

(1 1 1) — powers-of-2 bits, unshifted

(0 1 1) — powers-of-2 bits, shifted 1 bit to the right

$\updownarrow \updownarrow \updownarrow$ — XOR comparison of unshifted and shifted bits

(1 0 0) — Gray code results from XOR comparison

Figure 6. The two-step process by which a powers-of-2 binary can be converted to the Gray code binary representation of a decimal number.

Using **Figures 5** and **6** as a guide, show that the binary code for decimal 4 in powers-of-2 (1 0 0), is Gray code (1 1 0).

Gillham code actually is Gray code, because it uses Gray code to represent decimal numbers, exactly as shown in the above figures. But, Gillham code adds one additional layer of encoding to the decimal numbers, before being converted from decimal to Gray code.

This extra layer of adjusting the decimal numbers, before converting to Gray code, is what makes Gillham code, and it is best explained by example.

As explained above, Gillham code is Gray code, and it is defined for 11 parallel wires of binary data bits, labeled as follows, with the most significant bit on the left and the least significant bit on the right:

$\mathbf{D_2}$ $\mathbf{D_4}$ $\mathbf{A_1}$ $\mathbf{A_2}$ $\mathbf{A_4}$ $\mathbf{B_1}$ $\mathbf{B_2}$ $\mathbf{B_4}$ $\mathbf{C_1}$ $\mathbf{C_2}$ $\mathbf{C_4}$

(There is a place holder for a 12th line, $\mathbf{D_1}$, but it is currently unused.)

Many Altitude Encoders, especially for low altitude aircraft, use only 9 wires.

Altitude Range (Feet)	**Number of wires**	**Labels**
-1200 to 30,700	9	A1 though C4
-1200 to 62,700	10	D4, and A1 through C4
-1200 to 126,700	11	D2, D4, and A1 through C4

Figure 7. The number of wires required to represent various altitude ranges increases the larger the value of the altitude.

Each wire contains only one binary bit of information, either a zero or a one, using current as the analog representation. This method of current control is usually accomplished with a MOSFET, using a technique called "open-drain", meaning that the MOSFET will either present a floating, open circuit(meaning **0**), or a closed circuit(meaning **1**), draining current directly to ground.

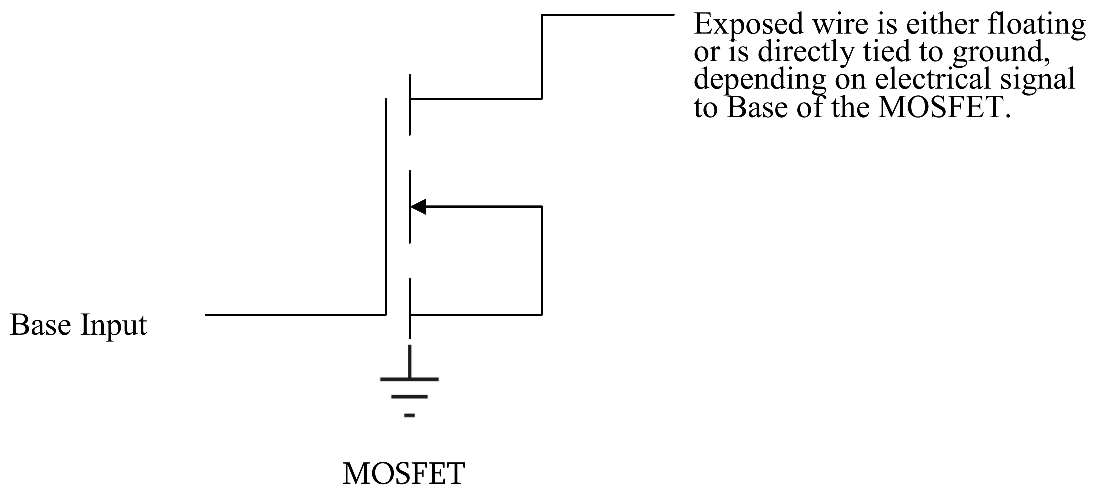
Exposed wire is either floating or is directly tied to ground, depending on electrical signal to Base of the MOSFET.
Base Input
MOSFET

Figure 8. A MOSFET can be used to present a wire that is either floating and will not accept current, or a wire that drains current directly to ground. This "open-drain" method of controlling current flow is how an Encoder represents binary data. There is a similar technique called "open-collector" that uses a transistor to control current, instead of a MOSFET.

This method of representing a binary **0** or **1** avoids the use of a specific voltage level, allowing encoders to operate over a range of voltage levels, typically 13.8 to 27.5 Volts.

There are ***five*** steps to generate Gillham code, which transforms the decimal altitude to a special decimal value, before it is converted to Gray code.

Step 1: Adjust the Altitude measured by the Encoder by adding 1200 feet

Adjusted Altitude = Altitude + 1200 (this makes all altitudes positive)

Step 2: Determine the multiple of 500 feet, $\mathbf{M_{500}}$, in the Adjusted Altitude

$\mathbf{M_{500}}$ = (Altitude + 1200) / 500 = (Adjusted Altitude) / 500

Altitude	Adjusted	Decimal M_{500}	Gray code(lines $D_2D_4\ A_1A_2A_4B_1B_2B_4$)
-1200	0	0	0 0 0 0 0 0 0 0
-700	500	1	0 0 0 0 0 0 0 1
-200	1000	2	0 0 0 0 0 0 1 1
+300	1500	3	0 0 0 0 0 0 1 0
+800	2000	4	0 0 0 0 0 1 1 0
+1300	2500	5	0 0 0 0 0 1 1 1
+1800	3000	6	0 0 0 0 0 1 0 1
+2300	3500	7	0 0 0 0 0 1 0 0
…		…	etc.

Figure 9. The number of 500 foot multiples in the measured altitude ($\mathbf{M_{500}}$)is converted directly to Gray code, and sent out over the first 8 wires. The remaining 3 wires are handled differently.

Step 3: Subtract ($\mathbf{M_{500}}$ * 500) from the Adjusted Altitude, divide it by 100 and then add 1, to obtain $\mathbf{N_{100}}$

$$\mathbf{N_{100}} = [\ (\text{Altitude} + 1200\ -\ \mathbf{M_{500}} * 500\ ) / 100\ ]\ + 1$$

Step 4: Reverse the $\mathbf{N_{100}}$ value if $\mathbf{M_{500}}$ is odd:

$$\text{If } (\mathbf{M_{500}} \,\%2)\ \mathbf{N_{100}} = 6 - \mathbf{N_{100}}$$

Step 5: If $\mathbf{N_{100}}$ is 5, replace it with a 7:

$$\text{If } (\mathbf{N_{100}} == 5)\ \mathbf{N_{100}} = 7$$

Altitude	$\mathbf{N_{100}}$	
+800	1	= (800 + 1200 - 4 * 500)/ 100 + 1
+900	2	= (900 + 1200 - 4 * 500)/ 100 + 1
+1000	3	= (1000 + 1200 - 4 * 500)/ 100 + 1
+1100	4	= (1100 + 1200 - 4 * 500)/ 100 + 1
+1200	5	= (1200 + 1200 - 4 * 500)/ 100 + 1
+1300	5	= 6 - [(1300 + 1200 - 5 * 500)/ 100 + 1]
+1400	4	= 6 - [(1400 + 1200 - 5 * 500)/ 100 + 1]
+1500	3	= 6 - [(1500 + 1200 - 5 * 500)/ 100 + 1]
+1600	2	= 6 - [(1600 + 1200 - 5 * 500)/ 100 + 1]
+1700	1	= 6 - [(1700 + 1200 - 5 * 500)/ 100 + 1]

Figure 10. Note that when $\mathbf{M_{500}}$ is even (a value of 4 in the above table) the value of $\mathbf{N_{100}}$ is increasing by 1. But, decreases by 1 when the value of $\mathbf{M_{500}}$ is odd. This is required in order to preserve the rule that Gray code must only change by 1 bit from its neighbor. This results in the fourth rule required to calculate the decimal Gillham values.

After Steps 1 through 5 obtain the Gillham code (numbers $\mathbf{M_{500}}$ and $\mathbf{N_{100}}$), they are converted to Gray code separately.

$\mathbf{N_{100}}$ is directly converted to Gray code, and sent over the 3 wires $\mathbf{C_1}$ $\mathbf{C_2}$ $\mathbf{C_4}$, and

$\mathbf{M_{500}}$ is directly converted to Gray code, and sent over the 8 wires $\mathbf{D_2}$ through $\mathbf{B_4}$.

The reason for using Gray code is to make certain that each successive 100 foot increment in altitude only changes 1 binary bit, which is the signal on one wire, at a time.

The purpose of separating the $\mathbf{N_{100}}$ value from the $\mathbf{M_{500}}$ value is to isolate the most rapidly changing numbers to the 3 data wires $\mathbf{C_1}$ $\mathbf{C_2}$ $\mathbf{C_4}$ .The reversal of the $\mathbf{N_{100}}$ offsets as $\mathbf{M_{500}}$ changes from even to odd preserves the incremental change in $\mathbf{C_1}$ $\mathbf{C_2}$ $\mathbf{C_4}$ to 1 binary bit.

Incrementing $\mathbf{N_{100}}$ by 1 avoids the use of Gray code (0 0 0), and replacing $\mathbf{N_{100}}$ = 5 with $\mathbf{N_{100}}$ = 7 avoids the use of Gray code (1 1 1). Also, since $\mathbf{N_{100}}$ can only take on 5 values, the Gray code (1 0 1) is also not used.

The reason Gray codes (0 0 0) and (1 1 1) are avoided for the 3 wires $\mathbf{C_1}$ $\mathbf{C_2}$ $\mathbf{C_4}$**,** is so that at least one line is always active, and a single fault in one line will either produce an invalid code or only change the altitude by 100 feet, in keeping with the Gray code minimization of error.

Transponders immediately will detect an invalid code if the $\mathbf{C_1C_2C_4}$ wires transmit any of these three codes: (0 0 0), (1 1 1), or (1 0 1).

See Figure 11 for an example of how to generate the Gillham Gray code for any altitude between –1200 and 126,700 feet, using C code.

```
void ( int Altitude)
{
Int  $N_{100}$, $M_{500}$, $G_{100}$, $G_{500}$, Value;

Value = Altitude + 1200;
$M_{500}$   = Value / 500;
$N_{100}$    = ( (Value – 500*$M_{500}$ ) / 100 ) + 1;

If ( $M_{500}$ %2 ) $N_{100}$ = 6 – $N_{100}$ ;

If ( $N_{100}$ == 5 )  $N_{100}$ = 7;

$G_{100}$ = ConvertToGray ( $N_{100}$ );
$G_{500}$ = ConvertToGray( $M_{500}$);

// Send $G_{100}$ over data lines $C_1C_2C_4$
// Send $G_{500}$ over data lines $D_2D_4A_1A_2A_4B_1B_2B_4$

}

 int ConvertToGray( int  decimal)
{
 return decimal ^ ( decimal >>1 );
}
```

Figure 11. Some pseudo-C code to generate the Gillham decimal numbers N_{100} M_{500} before converting to Gray code, and sending out over the 3-wire, and 8-wire bundles, respectively. A very simple C subroutine is used to do the right-shift and XOR operations to convert a decimal number to Gray code.