CO2Meter.com
131 Business Center Unit A-3, Ormond Beach, FL 32174
TEL: 877.678.4259 | FAX: 866.422.2356
EMAIL: sales@co2meter.com | www.co2meter.com

Application Note: Reading CO2 values on a K30 sensor

This is a technical brief on syntax, signal function / relationship timing and code examples for creating CRC code for communication with a K30 series sensor.

1. Read CO2 value example

Prerequisites for the application examples:

1. A single slave (sensor) is assumed.
2. Values in <..> are hexadecimal.
3. Baud rate is set to 9600.

CO2 read sequence:

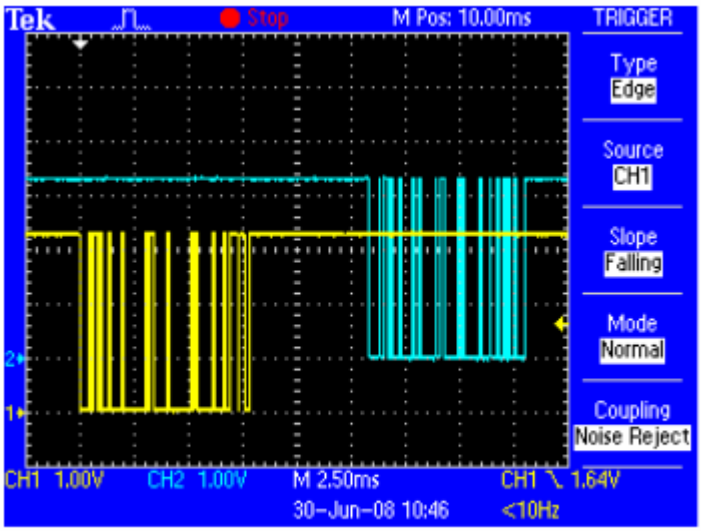
CO2 Engine K30 sensor is addressed as “Any address” (0xFE) or address set in sensors EEPROM, sensors address in EEPROM is 0x68 from factory, which is used in this example.
We read CO2 value from IR4 using “Read input registers” (function code 04). Hence, Starting address will be 0x0003 (register number-1) and Quantity of registers 0x0001. CRC calculated to 0xF3C8 is sent with low byte first.

Sensor replies with CO2 reading 769ppm (769 ppm = 0x301 hexadecimal).

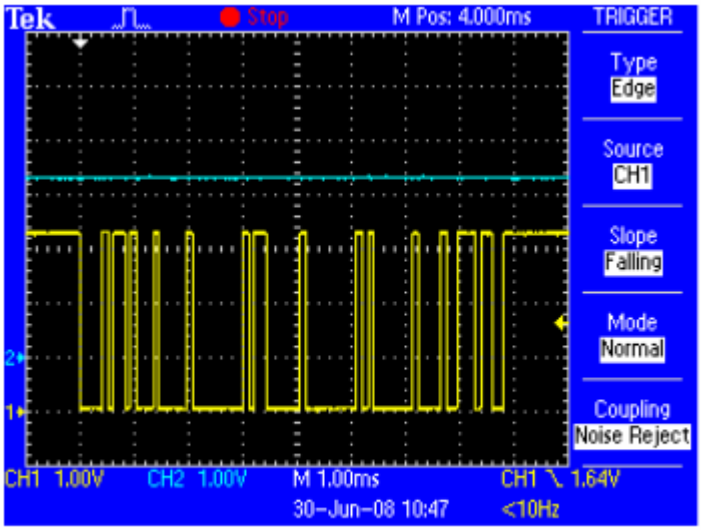
Master Transmit:
<68> <04> <00> <03> <00> <01> <C8> <F3>

Slave Reply:
<68> <04> <02> <03> <01> <24> <09>

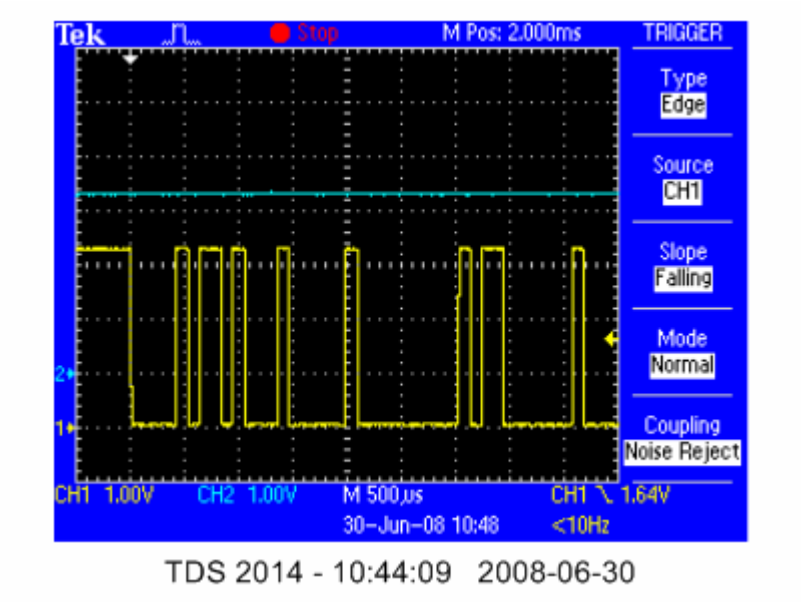
Following figures show Rx (yellow) and Tx (blue) for the example above (all figures show the same sequence, but different parts of it)



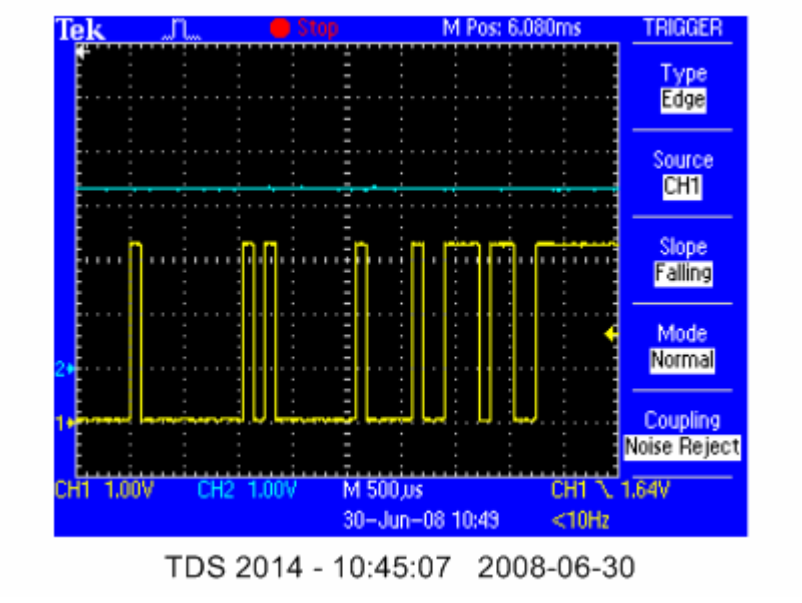
TDS 2014 - 10:42:13 2008-06-30

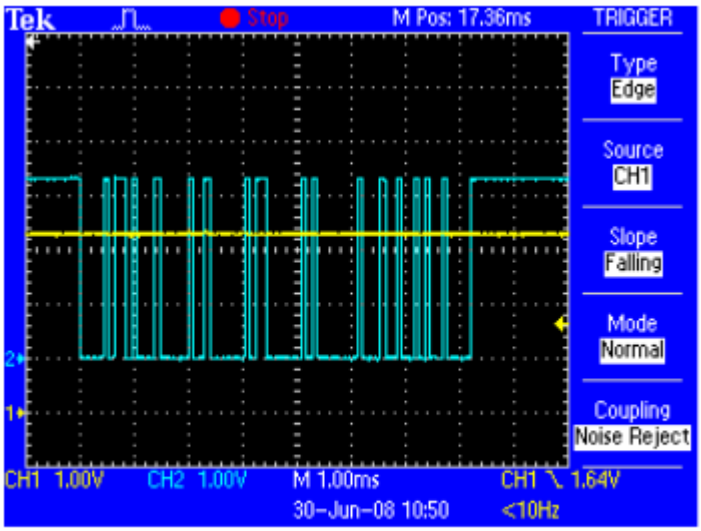


TDS 2014 - 10:43:12 2008-06-30

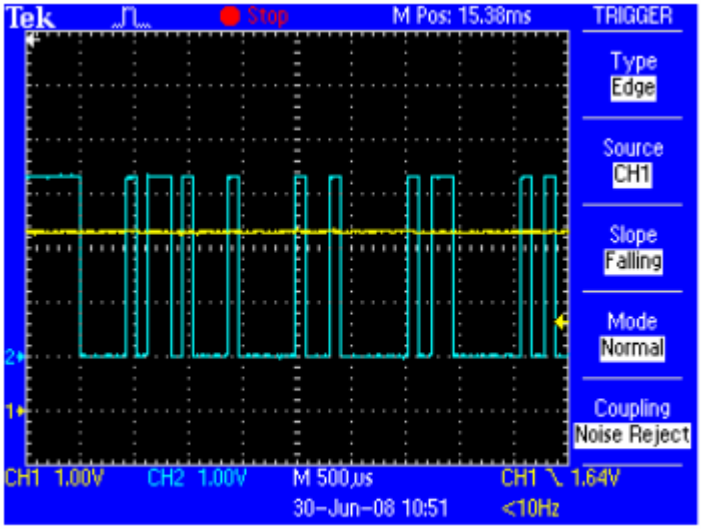


In the figure above one can see that start bit is followed by 0001 0110, least significant bit send first mean that we get 0110 1000 ⇔ 0x68, the data bits are followed by one parity bit (not checked by the sensor) and one stop bit.

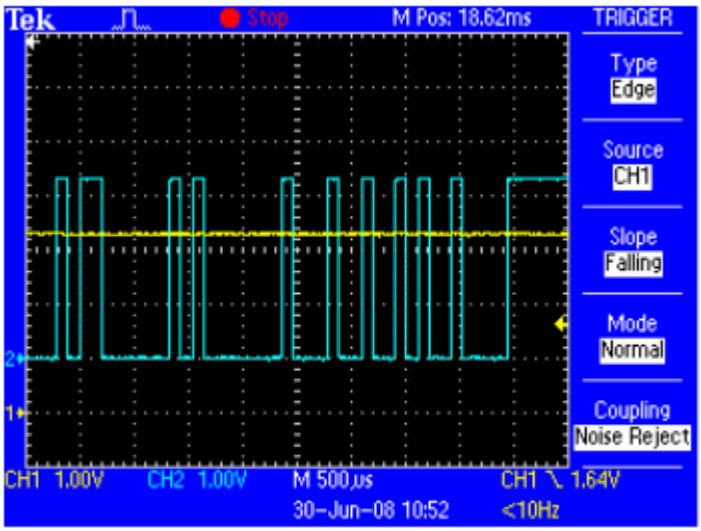




TDS 2014 - 10:46:13 2008-06-30



TDS 2014 - 10:47:13 2008-06-30


Tek
Stop
M Pos: 18.62ms
TRIGGER
Type
Edge
Source
CH1
Slope
Falling
Mode
Normal
Coupling
Noise Reject
CH1 1.00V
CH2 1.00V
M 500µs
CH1 \ 1.64V
30-Jun-08 10:52
<10Hz
TDS 2014 - 10:48:06 2008-06-30

CRC calculation:

1. Load a 16–bit register with FFFF hex (all 1's). Call this the CRC register.

2. Exclusive OR the first 8–bit byte of the message with the low–order byte of the 16–bit CRC register, putting the result in the CRC register.

3. Shift the CRC register one bit to the right (toward the LSB), zero–filling the MSB. Extract and examine the LSB.

4. (If the LSB was 0): Repeat Step 3 (another shift).
   (If the LSB was 1): Exclusive OR the CRC register with the polynomial value A001 hex (1010 0000 0000 0001).

5. Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8–bit byte will have been processed.

CRC calculation example in assembler:

```
// X - points to last byte to be included in CRC calculation
_calcCRC:
calcCRC:
  mov   [_CRC],0xFF                  // Initialize CRC register to all 1's
  mov   [_CRC+1],0xFF

  mov   [_Param],<_CommBuf           // Load LSB part of pointer to RAM based

.next_byte:
  mov   [_Param+1],8
  mvi   A,[_Param]          // Get byte from comm buffer into A and increment pointer
  xor   [_CRC],0            // Exclusive-OR MSB with 0
  xor   [_CRC+1],A          // Exclusive-OR LSB with byte from comm buffer
.loop8times:
  rrc   [_CRC]              // Shift CRC register right one step
  and   [_CRC],~0x80        // Set highest bit to 0
  rrc   [_CRC+1]
  jnc   .skip_xor

  xor   [_CRC],0xA0         // Xor with polynomial
  xor   [_CRC+1],0x01

.skip_xor:
  dec   [_Param+1]          // Loop 8 times
  jnz   .loop8times

  mov   A,X
  cmp   A,[_Param]          // Last character?
  jnz   .next_byte          // Get next character

  ret
```