# VZ89TE I²C Communication

The SGX VZ-89TE combines state-of-the-art MOS sensor technology with intelligent detection algorithms to monitor tVOCs and $CO_2$ equivalent variations in confined spaces, e.g. meeting rooms or vehicle cabins.

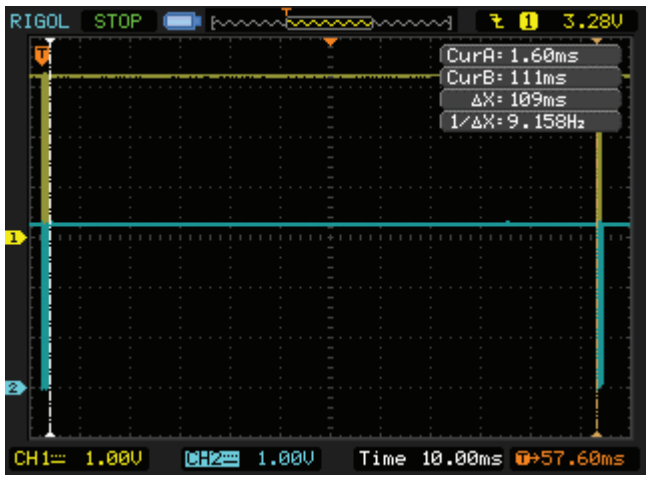
The dual signal output can be read through a multiplexed PWM output or through an I²C bus.

This datasheet will describe the I²C communication

## Theory of the operations:

When the device is connected to the I2C bus line, the device is working as a slave device. The master can write/read data to/from VZ-module using the I2C interface commands.

- The VZ89 device address contains seven fixed bits.
- The VZ89 communication must be set in “standard Mode”: bit rates up to 100 kbit/s.
- The pull-up resistors (4k7) on SDA and SCL line must be implemented on the master board (There are no pull-up resistor on VZ-89TE PCBA)
- A delay (~100ms) between the command (Write) frame and the status-request (Read) frame should be implemented.




Scope picture of the delay (100ms) between the command (Write) frame and the status-request frame (Read)

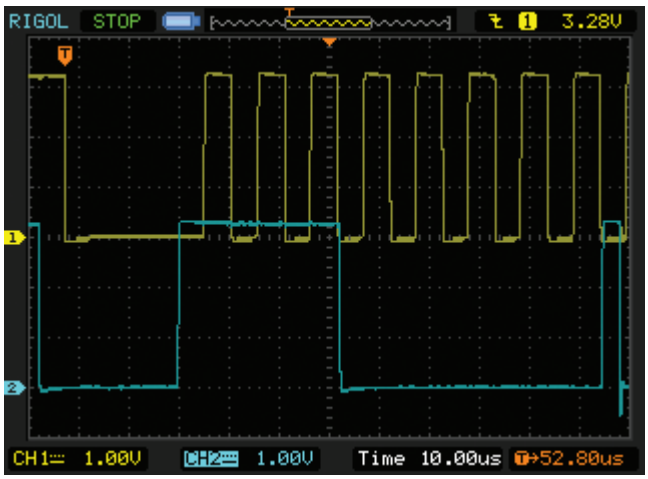
**Amphenol**

**Advanced Sensors**

# VZ89TE I²C Communication

## Device addressing:

The address byte is the first byte received following the START condition from the master device. The first part of the address byte consists of a 4-bit device code which is set to 1110 for the IAQS. The device code is followed by three address bits (A2, A1, A0) which are programmed at 0.




Scope picture of the address–byte
for a Write condition

## Sending the command frame:

The master sends a command byte (8 bits) followed by 4 data bytes and a CRC (8bits) in order to set parameters to the VZ-89TE or to request its status, as follow:

| Command (8 bits) | Data 0 (8 bits) | Data 1 (8 bits) | Data 2 (8 bits) | Data 3 (8 bits) | CRC (8 bits) |
|---|---|---|---|---|---|

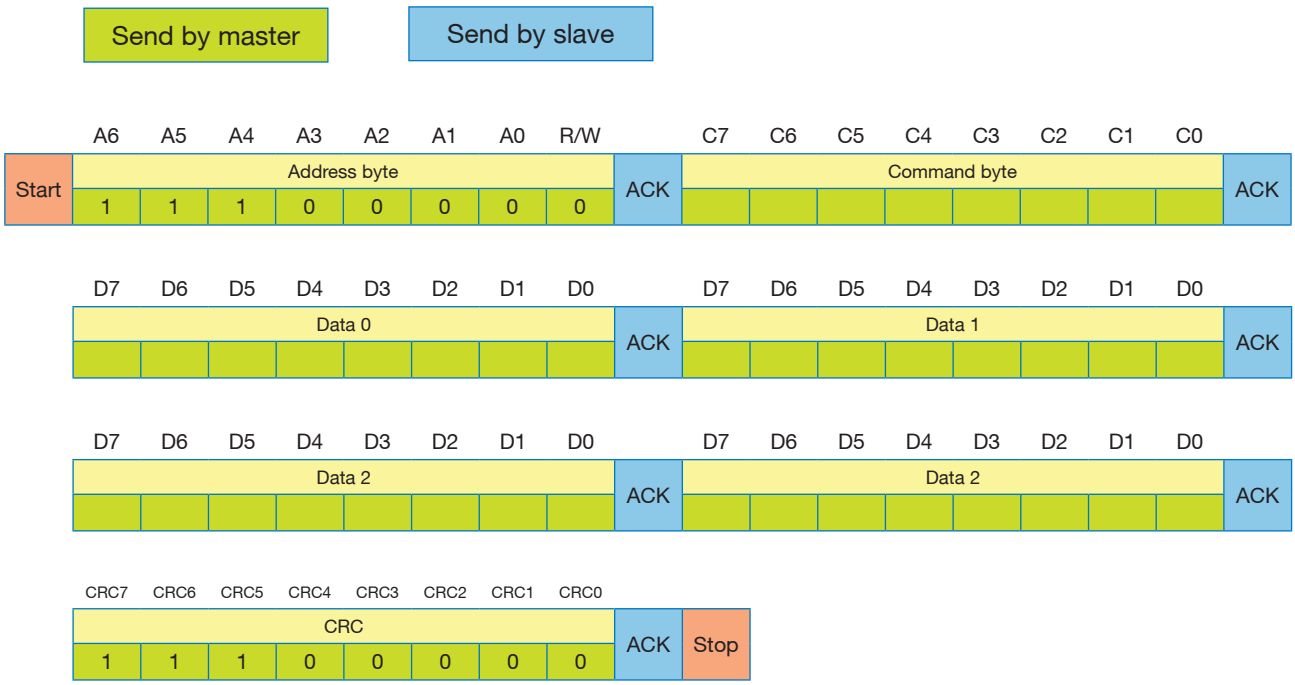
# VZ89TE I²C Communication

## Command List

| # | Command Value | Command Name | Description |
|---|---|---|---|
| 3.1 | 0b00001000 | setPPMCO2 | This command is used to send the ppm$CO_2$ value given by an external analyzer to the VZ89TE in order to recalibrate its outputs. |
| 3.2 | 0b00001001 | | Reserved for a future implementation. |
| 3.3 | 0b00001010 | | Reserved for a future implementation. |
| 3.4 | 0b00001011 | | Reserved for a future implementation. |
| 3.5 | 0b000001100 | getStatus | This command is used to read the VZ89TE status coded on 6x bytes + 1 CRC byte as follow:<br>Byte-1 = VOC signal level value<br>Byte-2 = CO2-equivalent signal level value<br>Byte-3 = Raw sensor value MSB<br>Byte-4 = Raw sensor value<br>Byte-5 = Raw sensor value LSB<br>Byte-6 = Error status byte<br>Byte-7 = CRC |
| 3.6 | 0b000001101 | getRevision | This command will return the revision code of the module as follow:<br>Byte-1 = Year<br>Byte-2 = Month<br>Byte-3 = Day<br>Byte-4 = ASCII code for a charter<br>Byte-5 = 0<br>Byte-6 = 0<br>Byte-7 = CRC |
| 3.7 | 0b000001110 | Reserved for engineering |  <br>Please don't use it because this command will overwrite the manufacturing calibration values. |
| 3.8 | | Reserved for engineering |  <br>Please don't use it because this command will overwrite the manufacturing calibration values. |
| 3.9 | 0b000010000 | getR0 | This command is used to read the R0 (calibration) value in [kOhms] coded on 6x bytes + 1 CRC byte as follow:<br>Byte-1 = LSB<br>Byte-2 = MSB<br>Byte-3 = 0<br>Byte-4 = 0<br>Byte-5 = 0<br>Byte-6 = 0<br>Byte-7 = CRC |

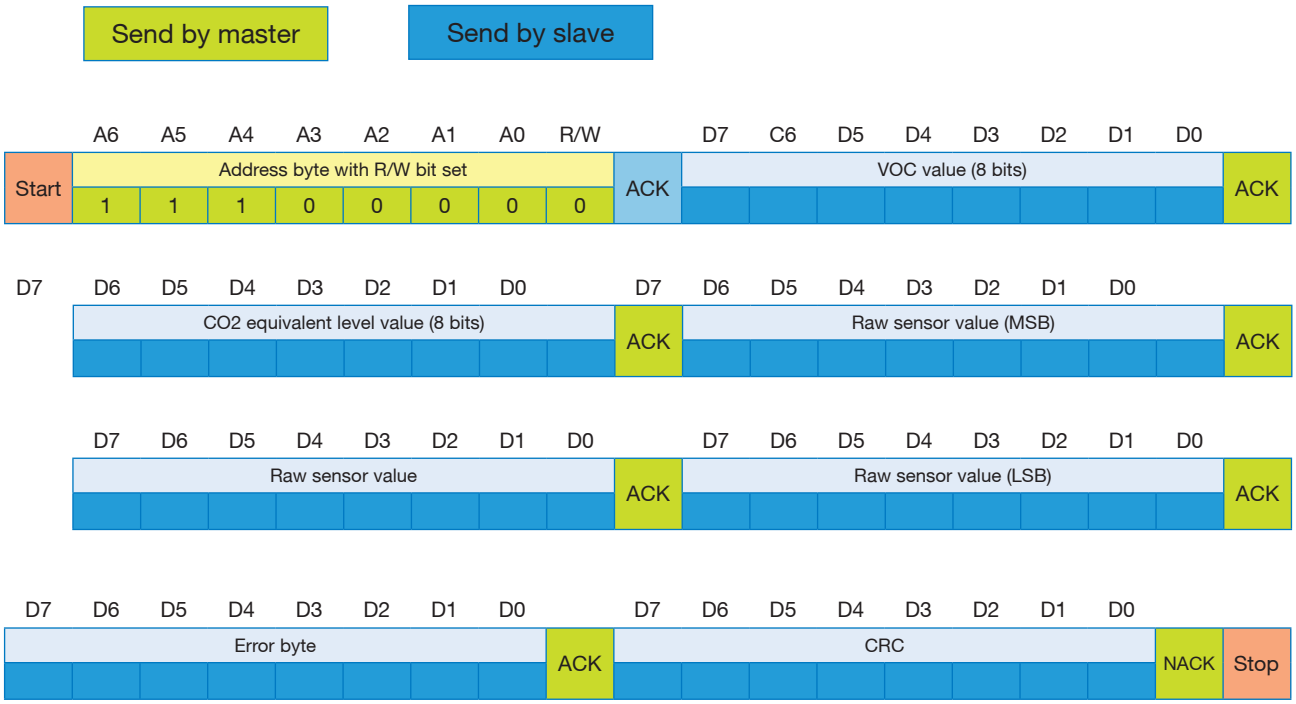
# VZ89TE I²C Communication

## Writing data to VZ module (I2C_SendBlock)



# VZ89TE I²C Communication

## Reading VZ status data (I2C_RecvBlock)



# VZ89TE I²C Communication

## The CRC byte:

The CHECKSUM byte contains the inverted modulo-256 sum over all data bytes.

The sum is calculated by ADD with carry where the carry bit of each addition is added to the LSB of its resulting sum. This guarantees security also for the MSBs of the data bytes.

The CRC byte sent by the MASTER is calculated with all data bytes including the command byte.

The CRC byte sent by the SLAVE is also calculated with all data bytes.

# VZ89TE I²C Communication

```
/*************************************************************
 * getCRC
 *
 *      Description :
 *      This function calcul, then return the CRC value
 *      of a data buffer.
 *
 *      Input parameters :
 *              1. A pointer on the data to process.
 *              2. The size of the data buffer
 *
 *      Return :
 *              the CRC value
 *************************************************************/

byte getCRC (byte *buffer, byte size) {
   /* Local variable */
     byte crc       = 0x00;
     byte i         = 0x00;
     word sum       = 0x0000;
   /* Summation with carry */
     for (i=0; i < size; i++) {
        sum += buffer[i];
     } //end loop
     crc =  (byte) sum;
     crc += (sum / 0x0100);    // Add with carry
     crc = 0xFF-crc;           // Complement
   /* Returning results*/
         return(crc);
  //end function
```

# VZ89TE I²C Communication

**Example Code snippet for Arduino in C++ communicating over I²C:**

```
  VZ89TEReadData(0x0D);  //date code & revision DD – MM - YYYY
  Serial.print("date code & revision  ");  Serial.print(data[2]); Serial.print("-"); Serial.print(data[1]); Serial.print("-"); Serial.
print(data[0]); Serial.print(", ");
  Serial.print("Revision: "); Serial.println(data[3]);
  Serial.print("R0 Calibration Value  ");

  VZ89TEReadData(0x10);  //R0 Calibration Value
  R0Value = ((data[1] & 0x3F ) << 8) | data[0];
  Serial.print(R0Value); Serial.println(" kohm");

  VZ89TEReadData(0x0C);  //status
  VOCvalue = ((data[0] - 13) * (1000 / 229));
  CO2value = ((data[1] - 13) * (1600 / 229) + 400);
  ResistorValue = 10 * (data[4] + (256 * data[3]) + (65536 * data[2]));
  Serial.print(VOCvalue); Serial.print(" ppb, ");
  Serial.print(CO2value); Serial.print(" ppm, ");
  Serial.print(ResistorValue); Serial.println(" ohm");

void VZ89TEReadData(long request)
{

  /*  CRC example processing:
    0x0F + 0x0A + 0x0F + 0x42 + 0x00 + 0x00 = 0x6A
    CRC = 0xFF – 0x6A = 0x95
*/

  //calculate crc for transmission
  crc = request;
  crc = (byte)(crc + (crc / 0x100));
  crc = 0xFF - crc;

  // Initiate Comms to device, initiate measure and read bytes of data
  Wire.beginTransmission(I2CAddress);
  Wire.write(request);  Wire.write(0);  Wire.write(0);  Wire.write(0);  Wire.write(0);  Wire.write(crc);
  Wire.endTransmission();
  delay(2);
  Wire.requestFrom(I2CAddress, 7);

  for (i = 0; i < 7; i++)
  {
    data[i] = Wire.read();
  }

  //calculate crc for received data
  crc = data[0] + data[1] + data[2] + data[3] + data[4] + data[5];
  crc = (byte)(crc + (crc / 0x100));
  crc = 0xFF - crc;

  /*  if (data[6] = crc) {
      Serial.print(" OK   ");
    } else {
      Serial.print(" NOK ");
    }
  */
}
```

SGX SENSORTECH
An Amphenol Company

www.telaire.com

www.amphenol-sensors.com

**Amphenol**
**Advanced Sensors**



AAS-920-694A - 06/2017