Physics 524

Survey of Instrumentation and Laboratory Techniques

2023

George Gollin
University of Illinois at Urbana-Champaign

Unit 2: I2C, SPI, and the oscilloscope

Goal for this unit

- Practice using an oscilloscope;
- Learn about the I2C and SPI inter-device communication protocols.

With SparkFun's permission, this week we'll use a lot of the excellent tutorial material they have posted to the web. Some of the tutorials are here:

- *Serial Communication*: https://learn.sparkfun.com/tutorials/serial-communication
- *I2C*: https://learn.sparkfun.com/tutorials/i2c
- *Serial Peripheral Interface (SPI)*: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

Working in pairs, you'll also be using a (digital) oscilloscope to look at the I2C and SPI data and clock lines on a pair of breadboarded BME680s.

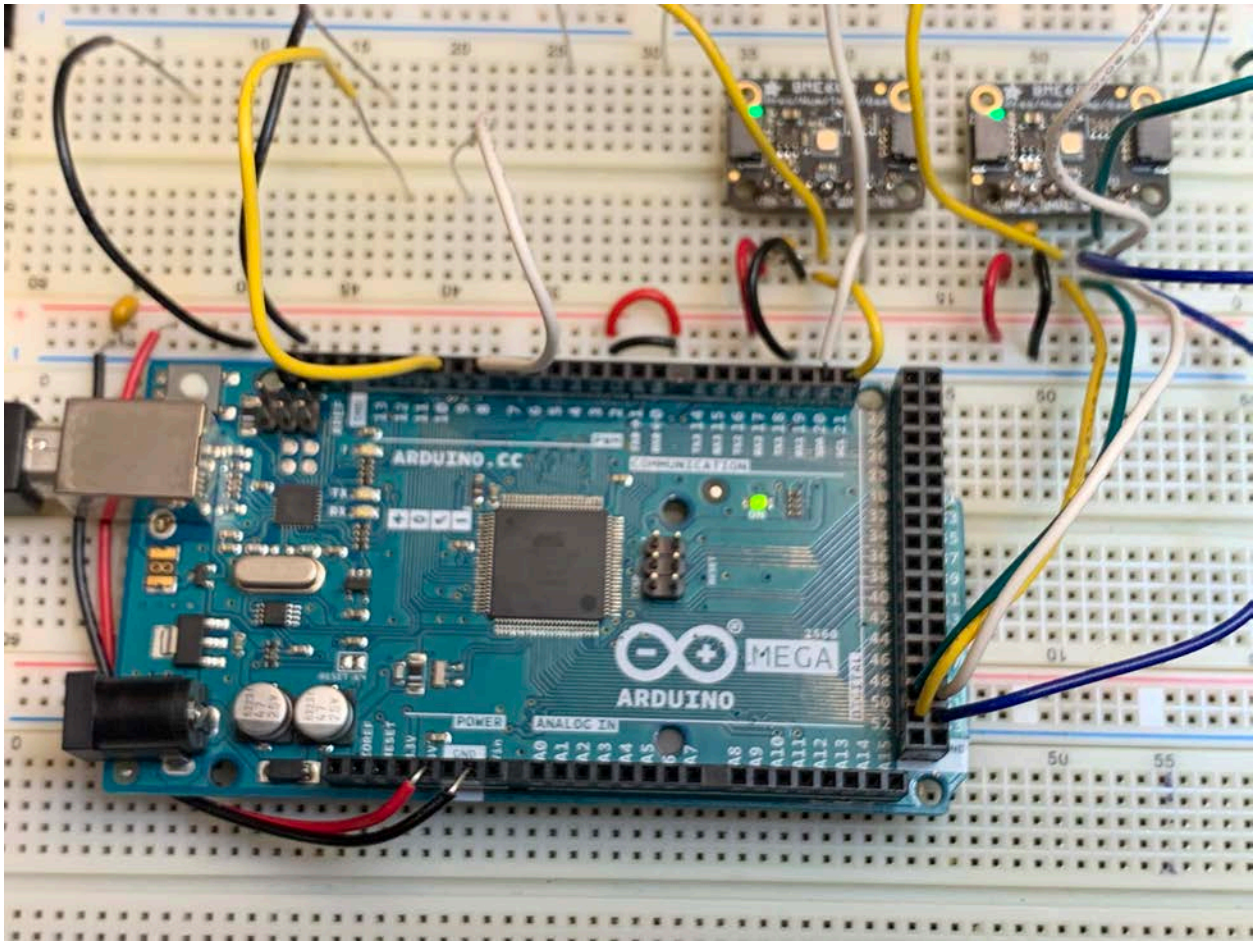
First things first

If you don't already have a BME680 on your bread board, wire up the circuit shown here: https://courses.physics.illinois.edu/phys524/fa2023/phys524_units/2/p524_week4.pdf. Since you'll be working in pairs, you only need to prepare one breadboard. Between the two people in a pair, you have at least two BME680s, so one of you should loan your BME680 to the breadboarded circuit.

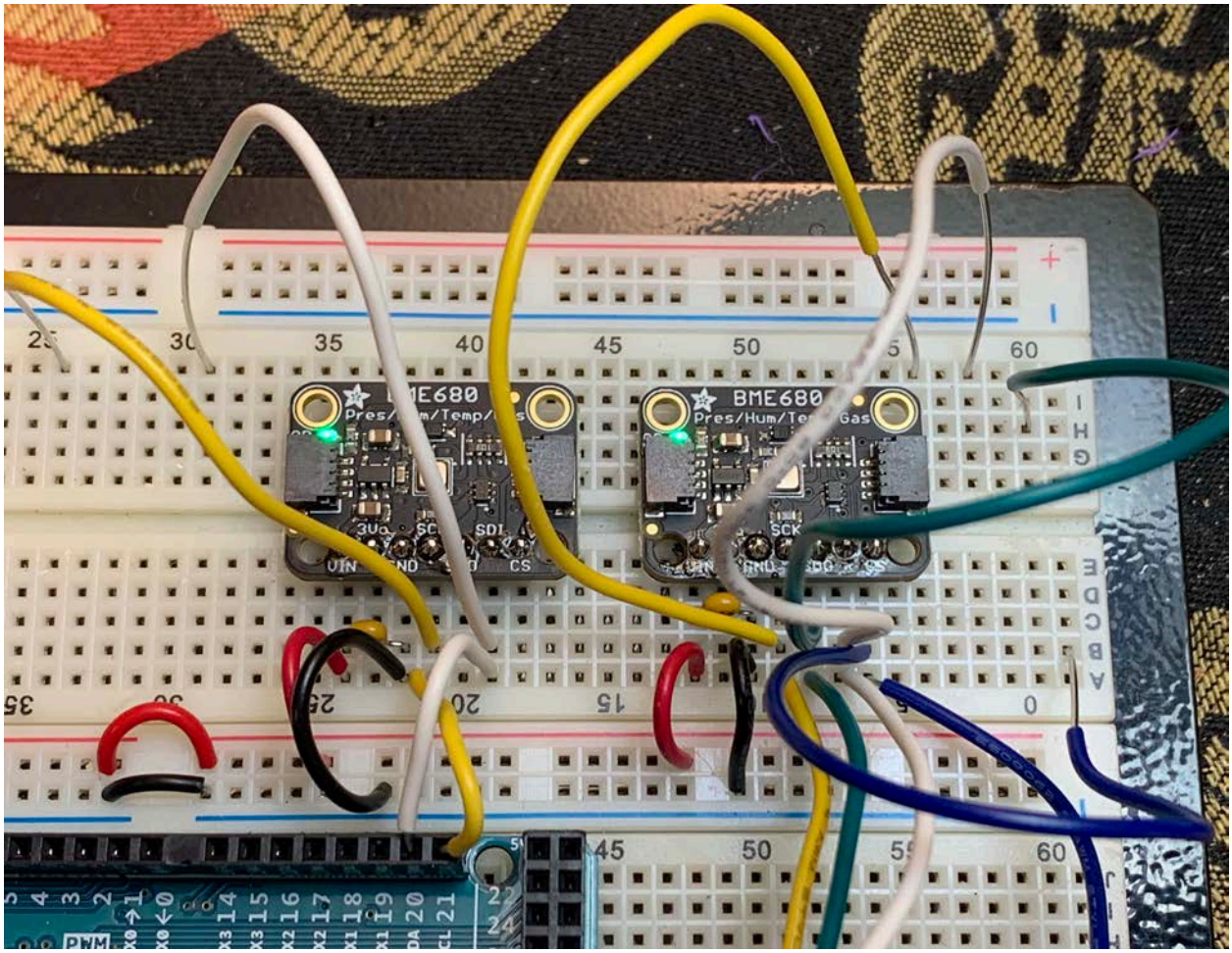
Add a couple of grounded (black) wires to the breadboard to which you'll clip the scope probes' grounding alligator clips. (Leave one end of the wires hanging or, better yet, anchor them in unused holes.) Do something similar to the data and control lines for the two BME680s.

One last thing: fly wires from a coupe of digital pins to the breadboard. I'm using Arduino pins 10 and 8.

Here are a couple of pictures of my circuit:



:



I've written a simple program that will run on a MEGA 2560 to read one of the BME680s; the code is linked here: https://courses.physics.illinois.edu/phys524/fa2023/phys524_units/2/bme680test.ino . Note that you'll need to put it inside a folder named "bme680test" to run it. Also note the presence of a compiler directive in the code in which DO_I2C can be set true (to use the I2C BME680) or false (to use the SPI BME680).

Please run the code to make sure everything is working.

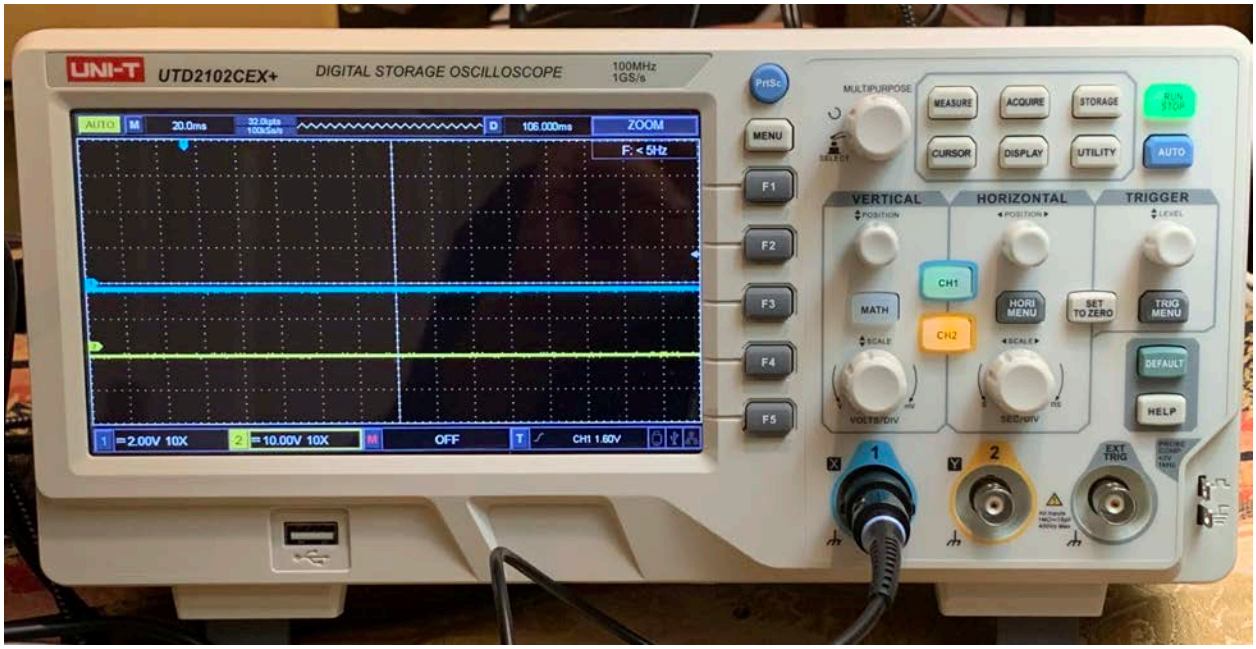
Oscilloscope!

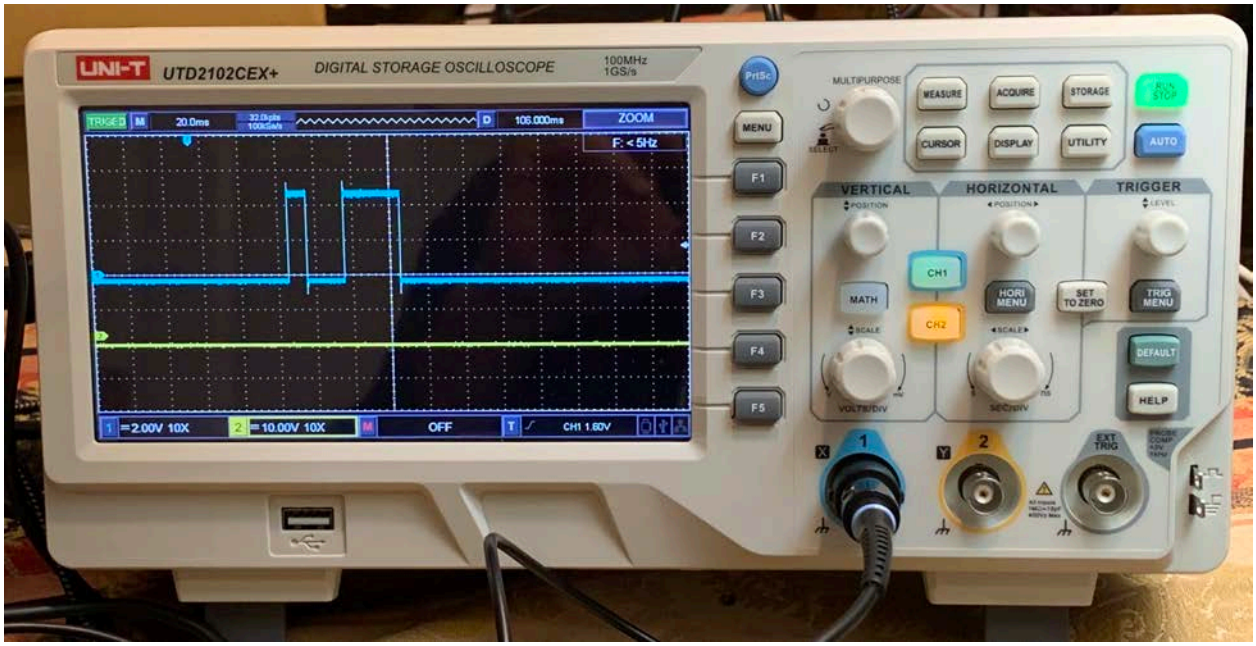
An oscilloscope is nothing more than a fancy voltmeter that represents the voltage seen by a probe by changing the vertical position of a bright spot on the screen according to how large the voltage is. When a "trigger condition" is satisfied the device sweeps the bright spot from left to right, changing the vertical position as the voltage changes.

Please write a very simple program that declares an Arduino pin to be an output, then "bit bang" a few pulses to it in each pass through loop. For example, you could try this:


```
/*****************************************************************************
  Code for Physics 524 week 4 exercises studying I2C and SPI communications:
  pulse the d10 line on the arduino.
 *****************************************************************************/

// Arduino digital pin we'll look at with the oscilloscope
#define the_pin 10

void setup()
{
  // open the line to the serial monitor
  Serial.begin(115200);

  // wait for it to be ready
  while (!Serial);

  // print a few blank lines
  Serial.print("\n\n\n");

  // message to the user:
  Serial.println("oscilloscope test program");

  // now set the Arduino pin to be an output, then set it LOW.
  pinMode(the_pin, OUTPUT);
  digitalWrite(the_pin, LOW);
}

void loop()
{
  // approximately once per second generate a 10 msec-wide positive pulse
  // followed by a 20 msec gap during which the pin stays low, then a 30
  // msec positive pulse. I am going to IGNORE the time to execute the
  // code in loop my duration calculations.

  // let's not hard-code the numerical values, a style thing.
  int dt1 = 10;
  int dt2 = 20;
  int dt3 = 30;
  int quiet_time = 1000 - dt1 - dt2 - dt3;

  Serial.println("...generate pulses");

  digitalWrite(the_pin, HIGH);
  delay(dt1);

  digitalWrite(the_pin, LOW);
  delay(dt2);

  digitalWrite(the_pin, HIGH);
  delay(dt3);

  digitalWrite(the_pin, LOW);
  delay(quiet_time);
}
```

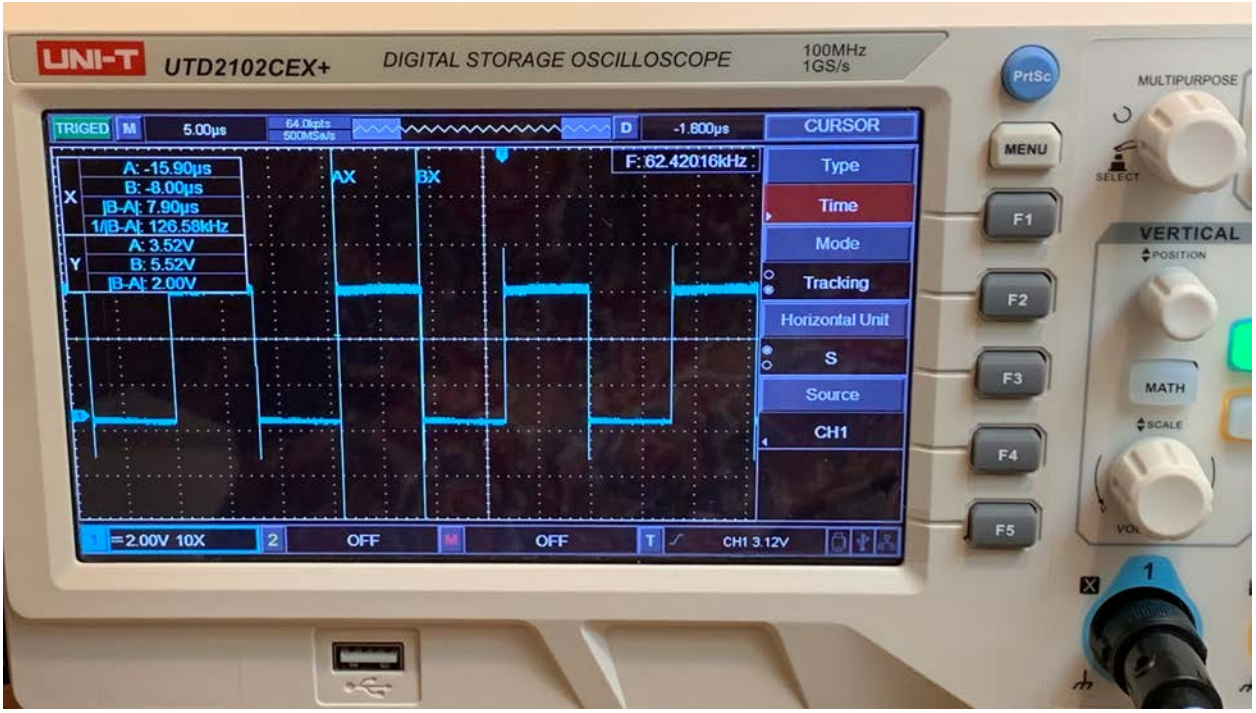
Now what? Now it's time to deal with the oscilloscope! We have on hand twenty UNI-T UTD2102CEX+ "digital storage oscilloscopes." Here are two photos of the one I'm working with:





In the first, the scope trace is generated automatically at regular intervals even though nothing interesting is coming down the scope probe cable. In the second, the scope has triggered its trace in response to detecting a rising edge on channel 1.

Now do this: explore the various scope functions and set the horizontal sweep speed to 5μs per division. Modify your pulse train code to do nothing but change the output on the digital pin (10 in my case) from HIGH to LOW, or LOW to HIGH each time your program reenters loop. Using the cursor function on the scope, figure out how long it takes for a single pass through loop. Here's a photo of my scope screen.

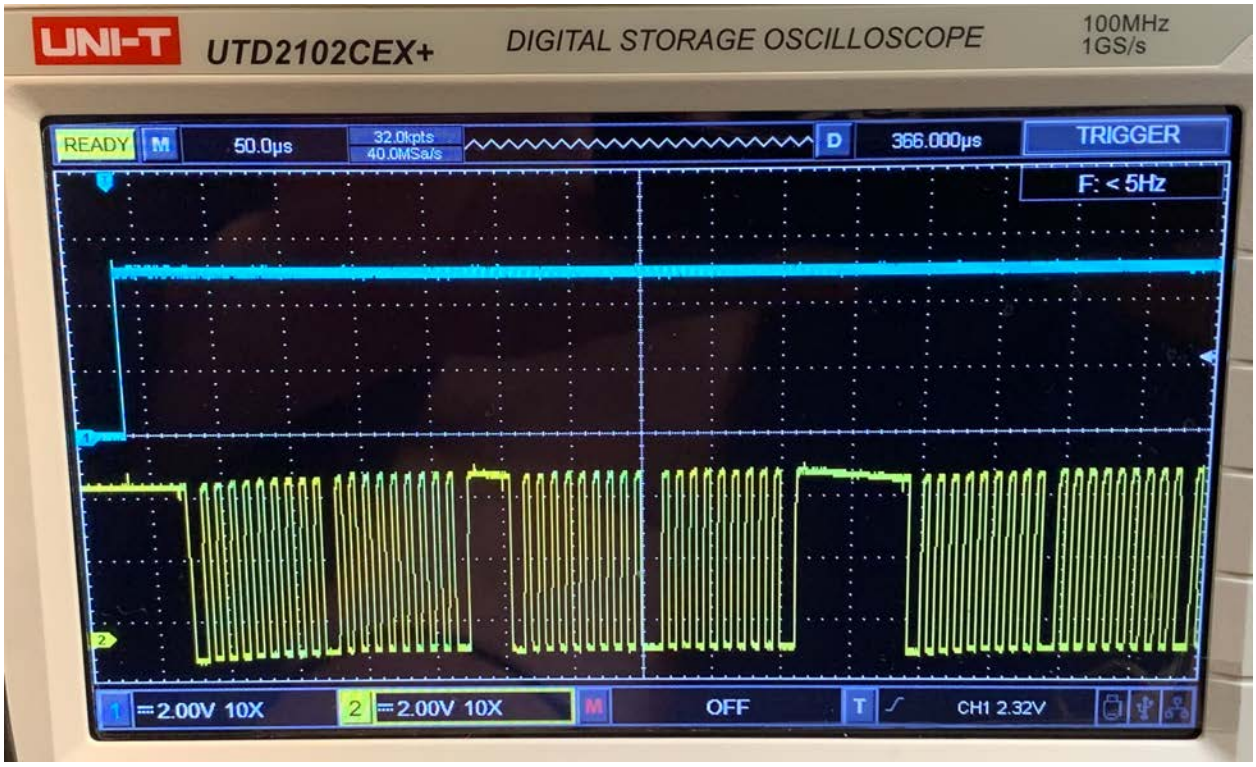


I2C

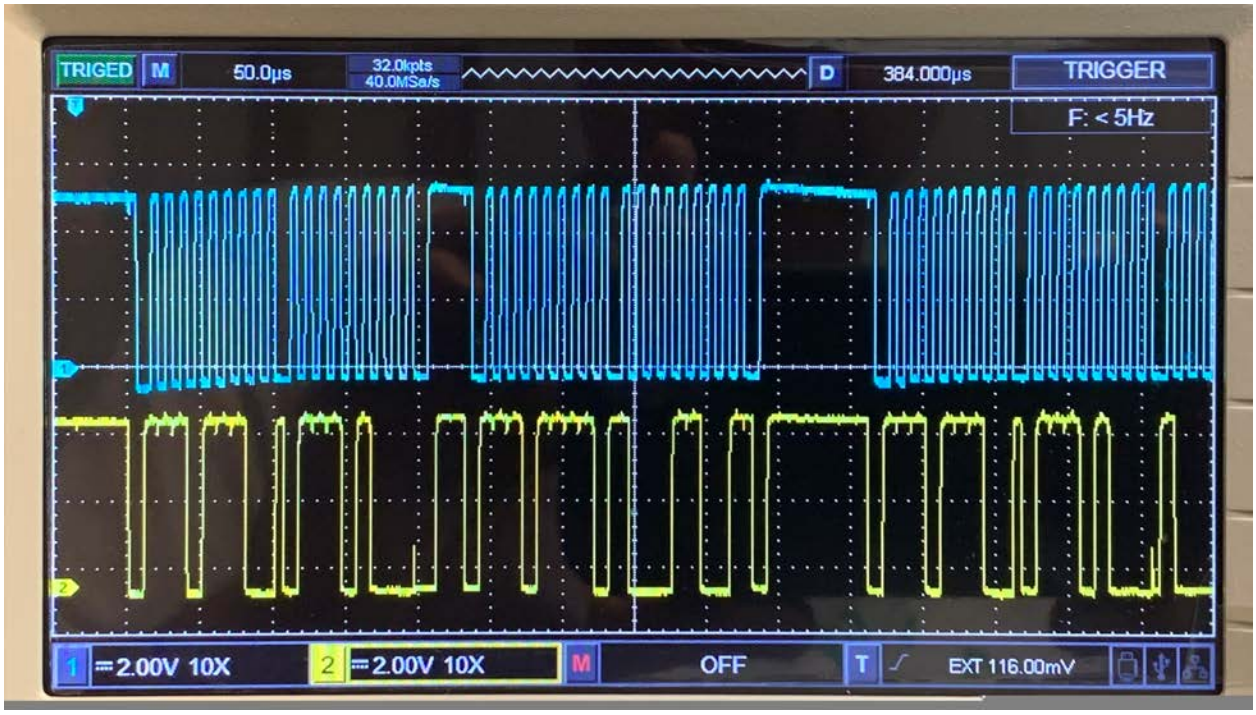
Modify your BME680 code so that it is talking to the I2C-connected device. Attach the channel 1 scope probe to your Arduino digital pin 10, and the channel 2 probe to BME680's SCK clock line. (Of course, make sure the probe's ground line is connected to your breadboard's ground.)

Have your code toggle the pin 10 value (switch it from HIGH to LOW, or else LOW to HIGH) each time the software makes its next pass through loop. Trigger your scope on a rising edge in channel 1, with the sweep speed set to 50 µs per division, when channel 2 is looking at the BME680's clock input. Here's what I get, with the first channel showing the pin 10 trigger and the second showing the I2C clock, generated by the Arduino.

A small subtle point: the BME680 breakout board includes 10k resistors connecting the I2C data and clock lines to +5V. In general, somewhere on a circuit board's I2C lines there need to be "pull-up" resistors since I2C devices are unable to push the clock and data lines up to the HIGH level. They can only pull the line down towards ground.



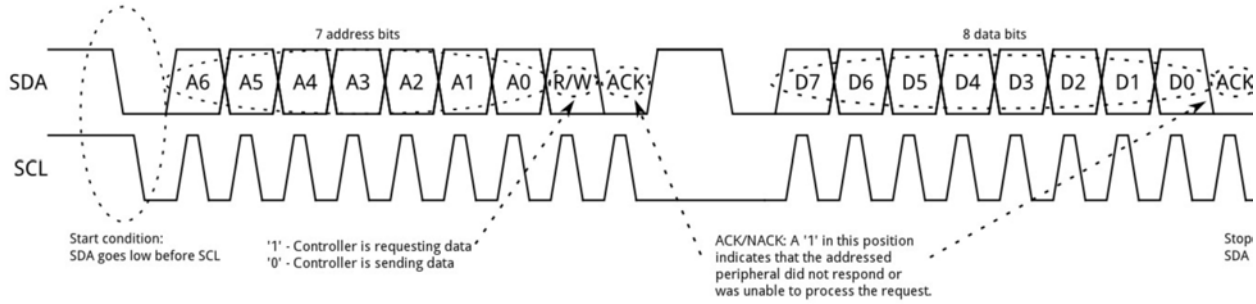
Now move your channel 1 signal to the "EXT TRIG" connector on the scope, and use a third probe so that the clock signal is on channel 1 and the data (SDI) line on channel 2. You should see something like this:



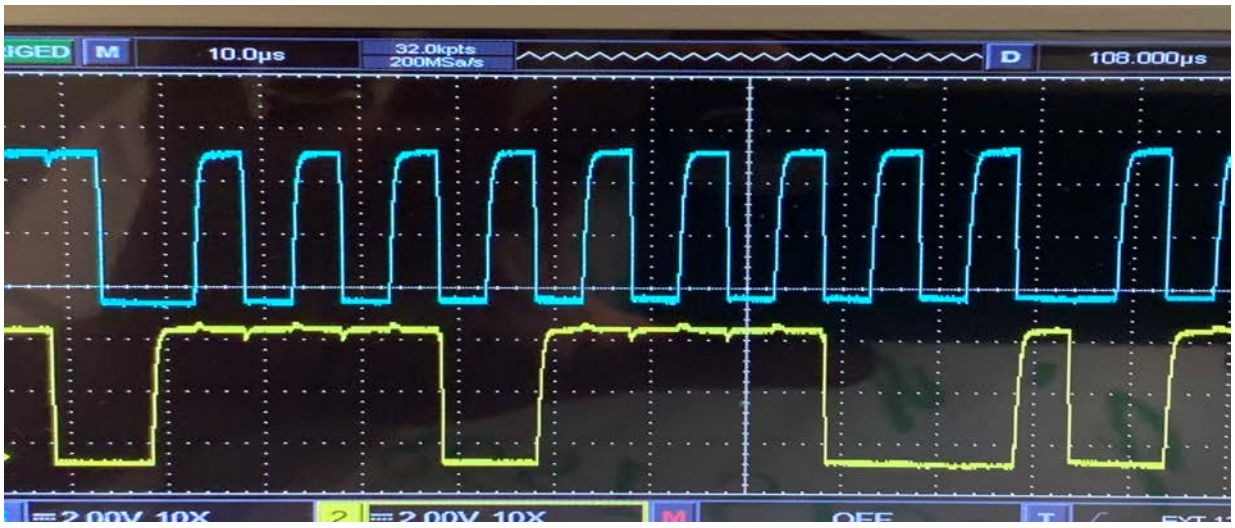
×

Note that the clock and data lines are initially HIGH, then the data line drops to low. This serves as the "start" signal to alert devices sharing on the I2C lines that something is about to happen. After the start signal, the next seven bits of data specify the I2C address of the device the Arduino wants to contact. A device's I2C address is hardwired into the device itself; the address of an unmodified BME680 will always be (hexadecimal) 0x77. Recall that 0x77 is the same as decimal 7×16 + 7, or 119. In binary it's 0b01110111.

From SparkFun: here's what is in the first part of the data flowing on the SDO line. Since the I2C address only uses the low order seven bits of a byte, we expect to see the first seven bits after the start signal sent as 1110111.



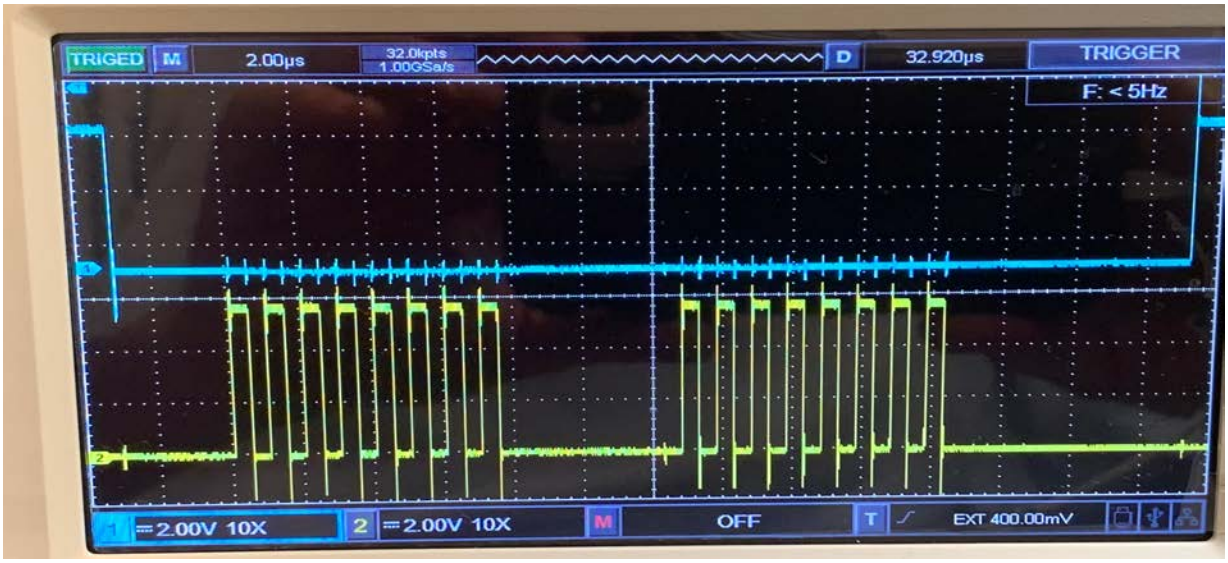
Let me expand the first part of the scope image for you. You'll see that the address is just what we expect. Also note that the R/W and ACK bits are both low since the Arduino will next tell the BME680 what it wants it to do, and the BME680 has acknowledged that it recognized its address.



Play around with the "cursor" function to measure the clock speed. I find, with my setup, that the MEGA 2560 generates a clock pulse every 10 µs, corresponding to a 100 kHz clocking rate.

SPI

Switch to running the SPI-connected BME680, and moving the channel 1 and channel 2 probes to the CS (chip select) and clock lines to the other BME680. With the scope still triggering externally on Arduino pin 10, this is what I see:



Note that the CS line is "active LOW": clock pulses are present when CS is LOW, not HIGH. Fool around with the cursor function to determine the clock speed… you should find that there's a clock pulse every microsecond, which is a factor of ten faster than the I2C clock.

The basic functions performed by the BME680 are the same, regardless of your choice of I2C or SPI.

This week's homework assignment (due at the first class meeting next week)

1. SparkFun

Please read all three SparkFun tutorials listed above, near the start of this unit's material.

2. SPI vs I2C speeds

Figure out how to disable the VOC measurement on your BME680s. Turn off all Serial.print statements in loop except for an initial print announcing your intentions and a final one giving your result. Compare how long it takes to read the BME680 100 times using I2C and SPI. Yeah, SPI is much faster, but only if your device's time to perform a measurement is small compared to the time to execute an I2C or SPI instruction!