Sensor Interface for the Drone

Debraj Chakraborty, Kishore Chatterjee, B.G. Fernandes, Joseph John, P.C. Pandey, Dinesh Sharma, Narendra S. Shiradkar and Kushal Tuckley
with
P.K. Baburajan, Rajneesh Bharadwaj, K.P. Karunakaran, D. Pawaskar and S.V. Prabhu,

EE and ME Departments
IIT Bombay, Mumbai

Academic Year: 2023-2024, Semester: II (Spring)

Interfacing sensors to Arduino Nano

- We have already seen how to interface a mobile phone to the drone.
- The mobile phone provides, through an app, the desired values of throttle/yaw/pitch/roll to the drone.
- To find the current tilt and angular velocities of the drone, we use a sensor board. This sensor board (GY 521) uses a six axis inertial measurement unit, MPU 6050.
- We had discussed earlier how this unit measures acceleration and angular velocities.
  Now we need to see how to interface this card to Arduino Nano, so that measured values can be conveyed to it.
- Measured values are converted to digital form using ADCs and are stored in registers in MPU 6050.
- The card is connected to Arduino using an I^2C serial interface. This needs only two wires – SCL and SDA.

Interfacing sensors to Arduino Nano

We are interested in reading the following values from the intertial measurement unit:

- Acceleration values along 3 axes. (3 Values, 16 bits each)
- Angular velocities around 3 axes. (3 Values, 16 bits each)
- Chip temperature (for calibration). (1 value, 16 bits)

Thus, we are interested in reading 7 values stored in 14 bytes.

These values are stored in registers on MPU 6050 and are read serially through the I^2C serial interface.

MPU 6050 has the default address of 0x68, which can be changed to 0x69 by tying the A0 pin high. We'll use the default address.

The 14 bytes of interest are stored in register number 59 (0x3B) onwards.

Hardware: Connecting MPU 6050 to Arduino

Hardware for connecting the MPU 6050 on GY 521 card is quite simple.

- The Inertial Measurement Unit acts as a slave device on the I^2C bus, while Arduino is the Master.
- The SDA pin on the IMU is connected to A4 pin (which acts as the Serial Data line SDA) on Nano.
- The SCL pin on the IMU is connected to A5 pin of Nano (which is driven by the master device as the serial clock).
- MPU 6050 operates with a supply voltage of 3.3 V. The GY 521 card contains a voltage regulator which accepts 5 V as input and provides 3.3 V to the MPU.

  So all we need to do is to connect the ground pin to ground and the V_{in} pin to 5 V – that's all!

Software: Reading MPU 6050 registers

The hardware connections for the GY 521 card have already been made - if your drone has been soldered according to the schematic.

Let us use a simple sketch which involves no other components except Arduino Nano and MPU 6050 (on GY 521 card).

A commented sketch follows – you should be able to figure out how we read the acceleration, temperature and angular velocities.

This is a simple illustrative sketch – there is no status check for data being ready etc. Its purpose is just to illustrate how data is read and also to find if the card is working and correctly connected.

The actual sketch used during flight control is a little more complex – but for now, try this one.

(The sketch is attached as a separate .ino file).

What next ...?

To make a working drone, you will have to understand, compile and run these sketches which have been provided to you:

step(0)-remote xy: to set up the user interface. You have already done this. If you did not change the WiFi name and passwords to group specific values, you will have to re-do this.

step (1)-YMFC-AL_setup: This sketch configures the sensor board and the I^2C interface. It is very similar to the simple sketch described earlier in this document.

step(2) - YMFC-AL_esc_calibrate: This sketch calibrates the speed controllers for motors and store configuration information in the EEPROM.

step(3) - YMFC-AL_Flight_controller: This is the sketch which runs when you fly your drone. It checks that you have carried out all of the previous steps before it will let you fly the drone.

This is a project – not a series of lab experiments under the supervision of instructors!

Components have been provided to you and have been soldered.

Interfacing of subsystems has been illustrated using simple programs.

You have seen that the whole design works – a flying drone has been demonstrated.

THERE WILL BE NO MORE HAND HOLDING –

GO AHEAD AND MAKE YOUR DRONE FLY!!

APPENDIX
Synchronous Serial Data Transfer
Using
I2C Bus

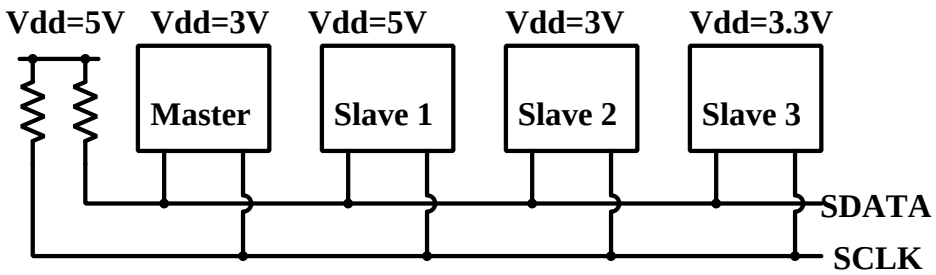
Synchronous Serial Data Transfer

- If the clock is sent along with data, the sampling rate as well as timing for data change and sampling can be easily managed. This is called synchronous serial communication.
- However, this requires that the skew (differences in delay) between the clock and data must be very small. This will be the case if the transmitter and the receiver are located close to each other – Typically on the same PCB or in the same system.
- The advantage of this method is that it can achieve very high data rates.
- Two popular protocols for synchronous serial communication are:
  - Inter Integrated Circuit (I^2C) bus
  - Serial Peripheral Interface or SPI.

Inter Integrated Circuit Bus or I^2C bus

- The I^2C bus is a 2 wire serial interface originally developed by Philips.
- The name stands for Inter Integrated circuit bus. Typesetting 'I^2C' is difficult in ordinary text, so it is often called the 'I2C' bus.
- Its main application is in data communication between ICs on a board – such as peripherals chips and processors and between smart home appliances.
- Subsequent to its introduction by Philips, Intel made some modifications to it to ensure inter-operability between devices from various manufacturers. That version is sometimes referred to as System Management Bus or SMBUS.
- Arduino cards include modules to support the I^2C Bus/SMBUS.
- Restricted versions of this protocol are also known as the '2 wire protocol'.

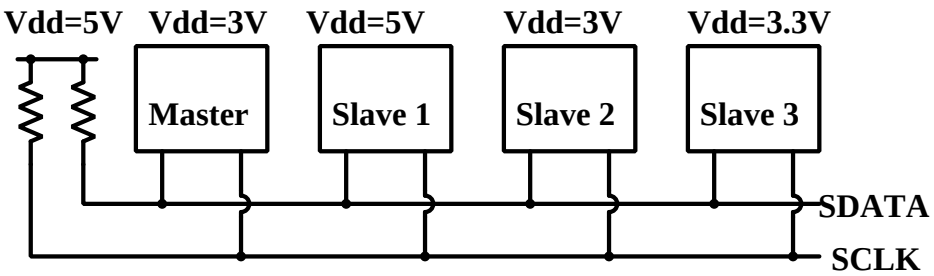
Synchronous Serial Data Transfer using I^2C Bus



Devices communicating with this protocol are connected in parallel across 2 wires, one of which carries serial data while the other carries the clock.

- All devices drive the bus lines using open drain drivers.
- Each line has a pull up resistor.
- Since devices provide only the pull down function, individual devices can use different supply voltages.
- The ability to connect devices using different supply voltages and the use of only 2 wires are the most attractive features of the I^2C Bus.

Synchronous Serial Data Transfer using I^2C Bus



The bus should have a master device, which provides the clock. Multiple slave devices can be connected to the bus.

- Each device has a unique address, which is normally 7 bit wide.
- The number of devices which can be connected across the bus is limited by the address space, or the maximum capacitive load of 400pF that can be placed on the bus.
- The master can act as the transmitter or the receiver.
- The addressed slave device then assumes a complementary role (receiver/transmitter).
- Multi-master protocols are also supported.

Data speed on the I^2C Bus

- Because of its open drain drivers and resistive pull ups, this bus is used for low and medium speed communication.
- In the standard mode, Data is transferred at rates of up to 100 kbit/s on this bus.
- The protocol does not restrict one from using much lower clock frequencies.
- Data can be sent at up to 400 kbit/s in the Fast-mode and up to 1 Mbit/s in Fast-mode Plus.
- Recent versions of the bus protocol can support a speed of up to 3.4 Mbit/s in the High-speed mode.

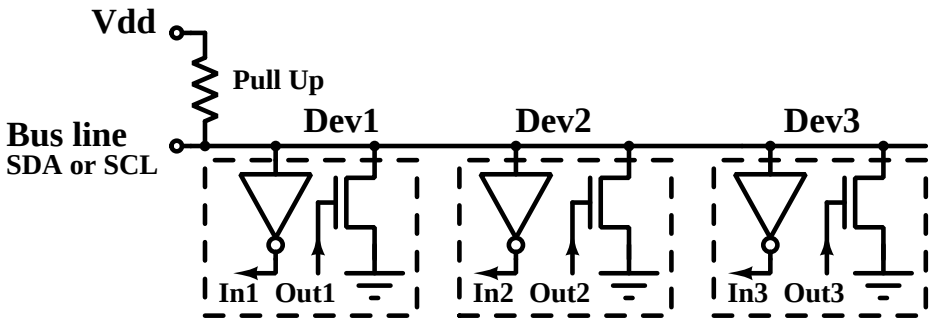
Evolution of I^2C Bus Specifications

Year	Version	Max. speed	Comments
1982	Original	100 kbit/s	Introduced by Philips
1992	1	400 kbit/s	Fast-mode and 10-bit addressing.
1998	2	3.4 Mbit/s	High-speed mode added.
2007	3	1 Mbit/s	Fast-mode plus with 20 mA drivers
2012	4	5 Mbit/s	Unidirectional Ultra Fast-mode (UFm) using push-pull logic without pull-up resistors

To use the 10-bit address scheme, the address frame consists of two bytes instead of one. A specific combination (‘11110’) of five most significant bits of the first byte is used to signal the 10-bit address mode. These 5 bits, the 10 bit address and the read/$\overline{\text{write}}$ bit make up the 16 bit address frame.

Open Drain drivers

Both lines (SDA and SCL) use open drain drivers. So it is worth recalling some characteristics of open drain logic.

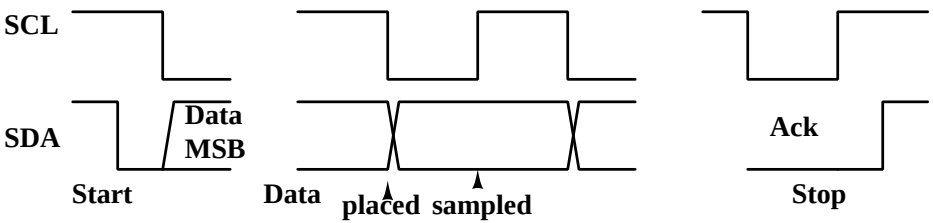


- Each driver has only a pull down transistor. Pull up is provided by the common pull up resistor on the bus.

- The bus can be 'High' only if *all* driver transistors are OFF.
- Any device can pull down the bus wire unconditionally.
- Different devices using different supply voltages can be easily connected as pull down devices.
- The bus driver transistor should be able to withstand the voltage provided by external V_{DD} – typically 5V.

Data transfer protocol

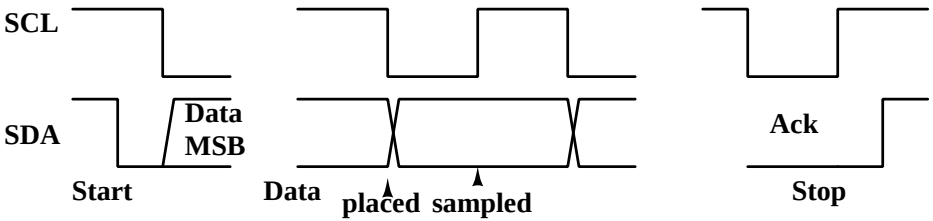
The actual data transfer is controlled by the Master, which provides the clock used for data transmission.



- It signals the ‘Start’ of transmission by a ‘High’ to ‘Low’ transition on the data line while the clock line is high.
- Similarly, the ‘Stop’ message is signaled by a ‘Low’ to ‘High’ transition on the data line while the clock line is high.
- These are the only instances when there are transitions on the data bus while the clock is high.
- Therefore these messages are easily distinguished from data transitions.

Data transfer protocol

For normal data transmission, the transmitter places data on the data line at the falling edge of clock.



Therefore all data transitions are seen after the clock goes low.

- The receiver samples data on the rising edge of the clock so that the data is stable at the time of sampling.
- In this protocol, the most significant bit is sent first.
- The master device can be the talker or the listener.
- Listener and talker roles are decided during the first message from the master.

Data transfer with master as the talker

Suppose the micro-controller is the master and it wants to send data serially to a slave device.

- The master generates the ‘Start’ message on the bus. (High SCL, ‘High’ to ‘Low’ transition on SDA).
- It then sends 8 bits on the serial data bus (Most significant bit first) which include the 7 bit address of the slave device and a R/$\overline{W}$ bit as the least significant bit.
- The last bit is 0 for a write operation (– Master being the talker in this example).
- After the transmitter has sent these 8 bits on the SDA line, the transmitter driver transistor goes off during the ninth bit time.
- The receiver pulls the data line ‘Low’ during this time to acknowledge successful receipt of the signal. (Recall that this is possible because of the open drain connection discussed earlier).

Data transfer with master as the talker

When the master is the talker, it generates the start condition, then sends the 7 bit address of the slave, followed by a ‘0’ to indicate that the master is the talker and the slave is the listener.

- During the ninth bit time, the driver transistor of the master is turned off and the receiver should pull down the SDA line if it received the data successfully.
- This is called the ‘Ack’ message.
- If the receiver driver transistor is also ‘OFF’ during the ninth bit period, the SDA line will be seen to be ‘High’ during this time.
- This is known as a ’Nack’ message. It is used by the slave to signal failure to receive when it is a listener (during write operation by master).

Data transfer with master as the talker

To summarize:

- The first transmission from the master establishes it as the talker and the slave as the listener if the last bit sent from the master is '0'.
- After the address has been acknowledged by the receiver, the master, (which is the talker), sends data serially to the slave receiver, checking for the 'Ack' signal at the end of each byte.
- After all bytes have been sent, the transmission is terminated by a 'Stop' signal on the line.

Data transfer with master as the listener

- When the master wants to be the listener, it creates the start condition, followed by a 7 bit address and then a ‘1’.
- As before, during the 9th bit time, the slave should pull down the SDA line to acknowledge that the byte has been received.
- After receiving the ‘Ack’ signal, the master releases the SDA line (turns off its open drain driver). It continues to drive SCL.
- The slave now becomes the transmitter and sends data. Each data byte is acknowledged by the master after reading it, through an ‘Ack’ signal.
- After reading the last byte, the master sends the ‘Nack’ signal to stop the slave from transmitting any further.
- The master follows up the ‘Nack’ signal with a stop condition to terminate this round of communication.