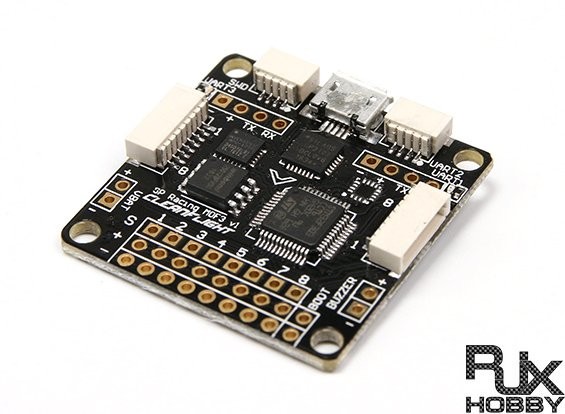
[Pick the date]

|  |
| --- |
| [Type the document subtitle] | Admin |

|  |  |
| --- | --- |
| Drishti – mini project | Aerodynamics |

## Objectives*:*

**Problem statement:**

**Develop electronic controlling systems for Quadcopters and RC Planes.**

**Develop an electronic controlling system for balancing a single axis body. The next part is to Develop an electronic controlling system for stabilising a Quadcopter in multi axis.**

## *Goals:*

**Develop a one axis self-balancing body.**

**Develop a flight controller completely from scratch.**

**To make cost effective flight controller.**

## Hardware:

## Motors:

A quad-copter uses 4 motors to generate the thrust needed to make it fly.

Most motors operate according to Faraday’s law of induction.

Motor operation is based on attraction and repulsion between magnetic poles, current flows through stator windings creating a magnetic pole that attracts the nearest permanent magnet pole on the rotor which causes the rotor to spin if the current shifts to an adjacent winding.

Pushing current through each winding in sequence will cause the rotor to start rotating.

**Brushless motors VS brush motors:**



The biggest difference between brush and brushless motors is that in brush motors the magnets are placed on stator and windings on rotor, while in brushless motors, magnets are placed on rotor and windings on stator

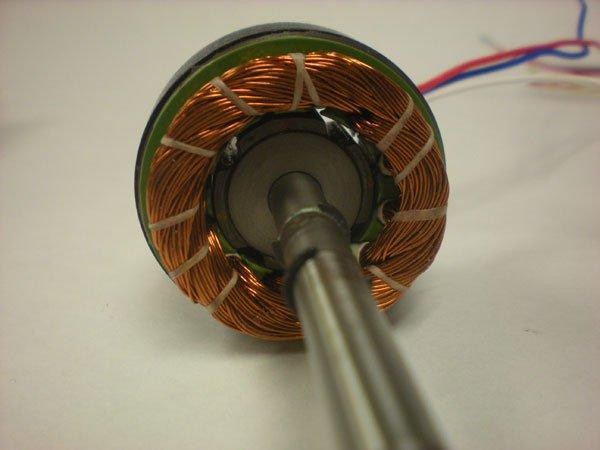
**Figure 3.2: Brushed motor with windings on rotor and magnets on stator**

## Disadvantages of brushed motors:

* + - * **Brushes wear out eventually.**
      * **Sparks and electrical noise caused by brushes making and breakingconnection.**
      * **Brushes decrease the maximum speed of the motor.**
      * **Harder to cool because the electromagnet is in the center.**
      * **Using brushes limits the number of poles the armature can have.**

**Advantages of brushless motors:**



* + - * **More precise, more efficient.**
      * **No sparks and less electrical noise due to lack of brush.**
      * **Easier to cool because electromagnets are on the stator.**
      * **You can increase the number of electromagnets on the stator for more precise control.**

The disadvantage of brushless motor is the higher cost but it’s compensated by longer motor life and higher efficiency.

**Figure 3.3: Brushless motor with windings on stator and magnets on rotor**

**Why we chose this motor:**

* + - * **Efficient motor.**
      * **Relatively cheap motor with enough thrust to**

carry the craft.

* + - * **4 x 800 gm = a total of 3200 gm thrust.**
      * **Since the weight of the quad with everything installed is approximately 1000gm.**



**Figure 3.4: A2212/13T**

**brushless out runner motor**

* + - * **That leaves us with approximately 2200 gm, enough to carry a camera and any other sensors we might add**.



**A2212/13TSpecifications:**

|  |  |
| --- | --- |
| **No. of cells** | **2-3** |
| **Stator dimensions** | **27.5 x 27 mm** |
| **Shaft diameter** | **3.2 mm** |
| **Weight** | **52.7 gm** |
| **Recommended propeller size** | **10 x 45** |
| **Recommended model weight** | **400 – 800 gm** |
| **Max RPM** | **12500** |
| **Max current** | **12 A/60 SEC** |
| **Thrust** | **800 gm** |

Table 3.2: A2212/13Tspecifications.

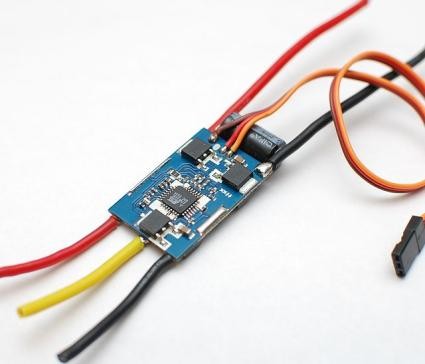
## ESC:

ESC’s or Electronic speed controls are often used on electrically powered radio control. It converts the DC power comes from the battery to AC current, and converts the voltage to the voltage required for the brushless motors , not all speed controllers have this capability, only those with BEC or UBEC written on them which stand for universal battery elimination circuit.

##### 22

ESC can control the speed and power of brushless motors; it also acts as a dynamic break. By generating a three phase electric power low voltage source.



**Types of ESCs:**

1. **Brushed ESC : this is the first ESC that have been created, and it is used for brushed motor (e.g.: electric RC vehicles)**
2. **Brushless ESC: is an advanced ESC that work with brushless motors by sending a sequence of signals.**

Figure 3.5: ESC

### How to choose the right ESC?

Each ESC is designed specifically for specific type of motor.

* **The ESC has to match the type of motor, meaning you have to use a brushless ESC for a brushless motor and a brushed ESC with a brushed motor.**
* **The ESC also has to match the motor in the amperage rating, because every motor requires different amount of amps.**

We chose in our project the EMAX Brushless ESC25A

Figure 3.6: ESC 25A

##### 23

* + 1. **ESC-30A specifications:**



|  |  |
| --- | --- |
| **Continuous current** | **30 A** |
| **Burst current** | **40 A** |
| **Lithium battery** | **2-4 cell** |
| **Dimensions** | **55 x 25 x 9 mm** |
| **Weight** | **23 gm** |
| **BEC (linear)** | **2A/5 V** |
| **Programmable** | **NO** |

Table 3.3: ESC-30A specifications.

## Battery:

* **Battery is our voltage source in this system, we need a 3 cell lithium polymer battery for our system,**
* **We chose the ORANGE-LIPO-2200battery (11.1 V, 2200 mAH-30C)**
* **Lithium polymer battery with high discharge rate/capacity and economic price. Its low weight /size and high power make it suitable for many application.**

**Battery Specifications:**

|  |  |
| --- | --- |
| **Capacity** | **2200 mAh** |
| **Configuration** | **3S1P/11.1v/3cell** |
| **Peak discharge** | **30 C** |
| **Charge** | **10 C** |
| **Pack weight** | **175 gm** |
| **Pack size** | **23 x 34 x 106 mm** |
| **Connector type** | **XT60-connector** |

Table - BAT-LIPO-2200 specification

### Microcontroller:

### Regarding the microcontroller we are using arduino uno and mega can also be used as the Master over here and we used it over ATmega due to the time constraints we were having. It has all the types of pins required for these project like PWM pins and etc. We used the Wire library for performing the I2C communication. Further information regarding the I2C has been provided in the report.

### Image result for arduino uno images..

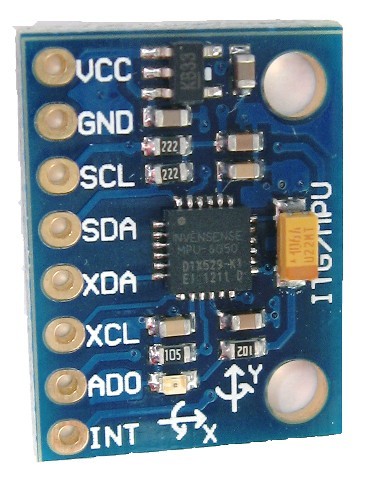
### 

# MPU-6050 Accelerometer + Gyro

## Introduction

**The InvenSense MPU-6050 sensor contains a MEMS accelerometer and a MEMS gyro in a single chip. It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefor it captures the x, y, and z channel at the same time. The sensor uses the I2C-bus to interface with the Arduino.**

**The MPU-6050 is not expensive, especially given the fact that it combines both an accelerometer and a gyro.**



**Photo: GY-521 breakout board**

**Also note that Invensense has combined the MPU-6050 with a magnetometer (compass) in a single chip called**[**MPU-9150**](https://playground.arduino.cc/Main/MPU-9150)**.**

**Reading the raw values for the accelerometer and gyro is easy. The sleep mode has to be disabled, and then the registers for the accelerometer and gyro can be read.**

**But the sensor also contains a 1024 byte FIFO buffer. The sensor values can be programmed to be placed in the FIFO buffer. And the buffer can be read by the Arduino.**

**The FIFO buffer is used together with the interrupt signal. If the MPU-6050 places data in the FIFO buffer, it signals the Arduino with the interrupt signal so the Arduino knows that there is data in the FIFO buffer waiting to be read.**

**A little more complicated is the ability to control a second I2C-device.  
The MPU-6050 always acts as a slave to the Arduino with the SDA and SCL pins connected to the I2C-bus.  
But beside the normal I2C-bus, it has it's own I2C controller to be a master on a second (sub)-I2C-bus. It uses the pins AUX\_DA and AUX\_CL for that second (sub)-I2C-bus.  
It can control, for example, a magnetometer. The values of the magnetometer can be passed on to the Arduino.**

**Things get really complex with the "DMP".  
The sensor has a "Digital Motion Processor" (DMP), also called a "Digital Motion Processing Unit". This DMP can be programmed with firmware and is able to do complex calculations with the sensor values.  
For this DMP, InvenSense has a discouragement policy, by not supplying enough information how to program the DMP. However, some have used reverse engineering to capture firmware.**

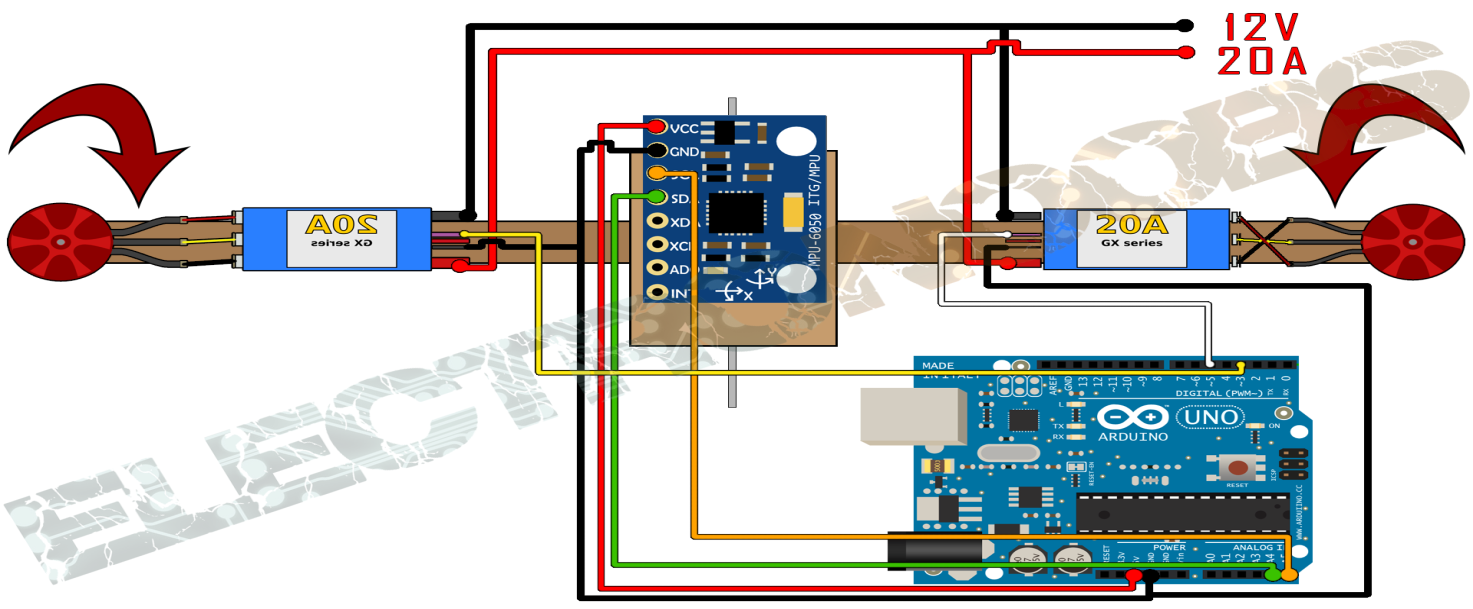
**The DMP ("Digital Motion Processor") can do fast calculations directly on the chip. This reduces the load for the microcontroller (like the Arduino). The DMP is even able to do calculations with the sensor values of another chip, for example a magnetometer connected to the second (sub)-I2C-bus.**

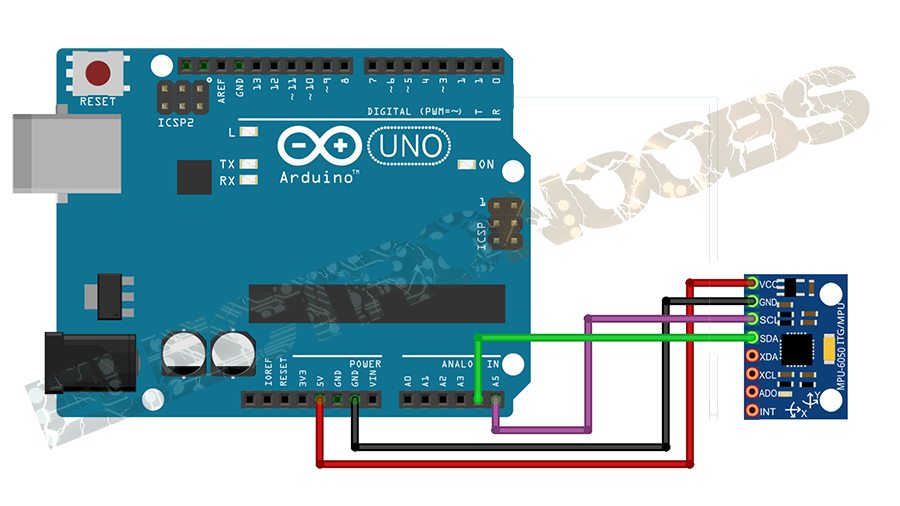
**Overall System**



Here we will discuss the main components of the system’s block diagram, the function of each component and how all the components can be successfully integrated together to produce an autopilot system as well as dynamic control system capable of flight.

## Circuit schematic

**First of all place the brushless motors in the end of each side of the balance. Make sure that each of the motor is spinning in the direction shown in the schematic below. Also that the screw of the propeller is getting tighten in the opposite direction as the rotation, otherwise the propeller will fly away during the tests. If the motor is not spinning in the desired direction, just reverse the top and bottom wires from the esc, leave the middle as it is. Make sure that the ESCs are calibrated and that both have the same scale. In this case this these ESCs both have a scale from 1000us to 2000us This is important. If the motors don't have the same scale, this system won't work**.   
  


**Now we should have the schematic above. Supply 12V to the ESCs. From the ESCs BEC supply 5V to the arduino or use the USB to supply it if you want. Connect the IMU using the i2c connection pins SCL and SDA. In the schematic below you can see with better details the mpu6050 connection to the arduino. Then connect pin 3 to the right ESC and pin 5 to the left one. Also connect gnd between the arduino and the ESCs.**  
   
  
**Place the MPU6050 module as centered as you can on the balance. We’ve use a breadboard to make all the connections. Also I recommend you to wind each of the i2c cables around a GND cable. This will reduce noise of the i2c communication. For that get two wires of GND from the arduino to the MPU6050 and wire around each of this the SCL and SDA cables of the i2c**

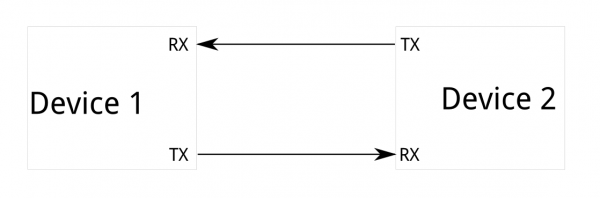
### Communication protocol- I2C

**The Inter-integrated Circuit (I2C) Protocol is a protocol intended to allow multiple “slave” digital integrated circuits (“chips”) to communicate with one or more “master” chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.**

**WHY WE USE I2C?**

**To figure out why one might want to communicate over I2C, you must first compare it to the other available options to see how it differs.**

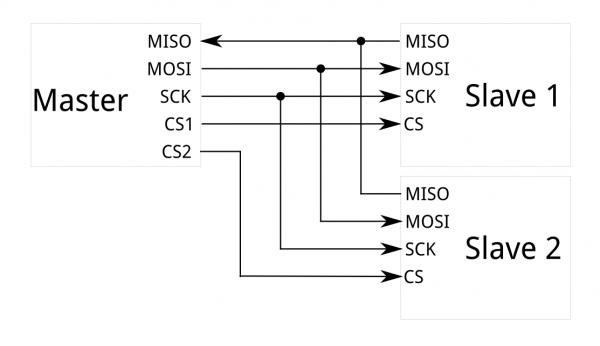
1. **What is wrong with Serail Ports?**



**Core fault in asynchronous serial ports is that they are inherently suited to communications between two, and only two, devices. While it is possible to connect multiple devices to a single serial port, bus contention (where two devices attempt to drive the same line at the same time) is always an issue and must be dealt with carefully to prevent damage to the devices in question, usually through external hardware.**

**Finally, data rate is an issue. While there is no theoretical limit to asynchronous serial communications, most UART devices only support a certain set of fixed baud rates, and the highest of these is usually around 230400 bits per second.**

1. **What is wrong with SPI?**



**The most obvious drawback of SPI is the number of pins required. Connecting a single master to a single slave with an SPI bus requires four lines; each additional slave requires one additional chip select I/O pin on the master. The rapid proliferation of pin connections makes it undesirable in situations where lots of devices must be slaved to one master. Also, the large number of connections for each device can make routing signals more difficult in tight PCB layout situations. SPI only allows one master on the bus, but it does support an arbitrary number of slaves (subject only to the drive capability of the devices connected to the bus and the number of chip select pins available).**

**SPI is good for high data rate full-duplex (simultaneous sending and receiving of data) connections, supporting clock rates upwards of 10MHz (and thus, 10 million bits per second) for some devices, and the speed scales nicely. The hardware at either end is usually a very simple shift register, allowing easy implementation in software**

I2C

**I2C requires a mere two wires, like asynchronous serial. Unlike SPI, I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can’t talk to each other over the bus and must take turns using the bus lines).**

**Data rates fall between asynchronous serial and SPI; most I2C devices can communicate at 100kHz or 400kHz. There is some overhead with I2C; for every 8 bits of data to be sent, one extra bit of meta data must be transmitted.**

**The hardware required to implement I2C is more complex than SPI, but less than asynchronous serial. It can be fairly trivially implemented in software.**

## I2C at the Hardware Level

### Signals

**Each I2C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is called “clock stretching” .**

**Unlike UART or SPI connections, the I2C bus drivers are**[**“open drain”**](http://en.wikipedia.org/wiki/Open_collector)**, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a**[**pull-up resistor**](https://learn.sparkfun.com/tutorials/pull-up-resistors)**on it, to restore the signal to high when no device is asserting it low.**

**Resistor selection varies with devices on the bus, but a good rule of thumb is to start with 4.7k and adjust down if necessary. I2C is a fairly robust protocol, and can be used with short runs of wire (2-3m). For long runs, or systems with lots of devices, smaller resistors are better.**

### Signal Levels

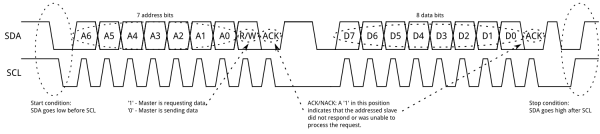
**Since the devices on the bus don’t actually drive the signals high, I2C allows for some flexibility in connecting devices with different I/O voltages. In general, in a system where one device is at a higher voltage than another, it may be possible to connect the two devices via I2C without any level shifting circuitry in between them. The trick is to connect the pull-up resistors to the lower of the two voltages. This only works in some cases, where the lower of the two system voltages exceeds the high-level input voltage of the the higher voltage system–for example, a 5V Arduino and a 3.3V accelerometer.**

**If the voltage difference between the two systems is too great (say, 5V and 2.5V), SparkFun offers a**[**simple I2C level shifter board**](https://www.sparkfun.com/products/10403)**. Since the board also includes an enable line, it can be used to disable communications to selected devices. This is useful in cases where more than one device with the same address is to be connected to a single master–Wii Nunchucks are a good example.**

## Protocol

**Communication via I2C is more complex than with a UART or SPI solution. The signalling must adhere to a certain protocol for the devices on the bus to recognize it as valid I2C communications. Fortunately, most devices take care of all the fiddly details for you, allowing you to concentrate on the data you wish to exchange.**

### Basics



**Messages are broken up into two types of frame: an address frame, where the master indicates the slave to which the message is being sent, and one or more data frames, which are 8-bit data messages passed from master to slave or vice versa. Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high. The time between clock edge and data read/write is defined by the devices on the bus and will vary from chip to chip.**

#### Start Condition

**To initiate the address frame, the master device leaves SCL high and pulls SDA low. This puts all slave devices on notice that a transmission is about to start. If two master devices wish to take ownership of the bus at one time, whichever device pulls SDA low first wins the race and gains control of the bus. It is possible to issue repeated starts, initiating a new communication sequence without relinquishing control of the bus to other masters**.

#### Address Frame

**The address frame is always first in any new communication sequence. For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by a R/W bit indicating whether this is a read (1) or write (0) operation.**

**The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address). Once the first 8 bits of the frame are sent, the receiving device is given control over SDA. If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that the receiving device either did not receive the data or did not know how to parse the message. In that case, the exchange halts, and it’s up to the master of the system to decide how to proceed.**

#### Data Frames

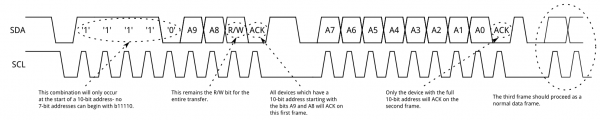
**After the address frame has been sent, data can begin being transmitted. The master will simply continue generating clock pulses at a regular interval, and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation. The number of data frames is arbitrary, and most slave devices will auto-increment the internal register, meaning that subsequent reads or writes will come from the next register in line.**

#### Stop condition

**Once all the data frames have been sent, the master will generate a stop condition. Stop conditions are defined by a 0->1 (low to high) transition on SDA after a 0->1 transition on SCL, with SCL remaining high. During normal data writing operation, the value on SDA should not change when SCL is high, to avoid false stop conditions.**

### Advanced Protocol Topics

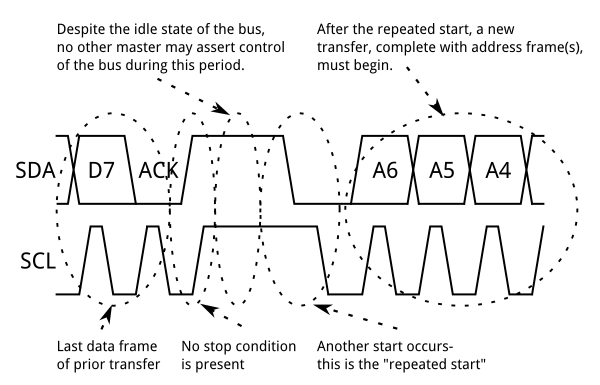
#### 10-bit Addresses



**In a 10-bit addressing system, two frames are required to transmit the slave address. The first frame will consist of the code b11110xyz, where ‘x’ is the MSB of the slave address, y is bit 8 of the slave address, and z is the read/write bit as described above. The first frame’s ACK bit will be asserted by all slaves which match the first two bits of the address. As with a normal 7-bit transfer, another transfer begins immediately, and this transfer contains bits 7:0 of the address. At this point, the addressed slave should respond with an ACK bit. If it doesn’t, the failure mode is the same as a 7-bit system.**

**Note that 10-bit address devices can coexist with 7-bit address devices, since the leading ‘11110’ part of the address is not a part of any valid 7-bit addresses.**

#### Repeated Start Conditions

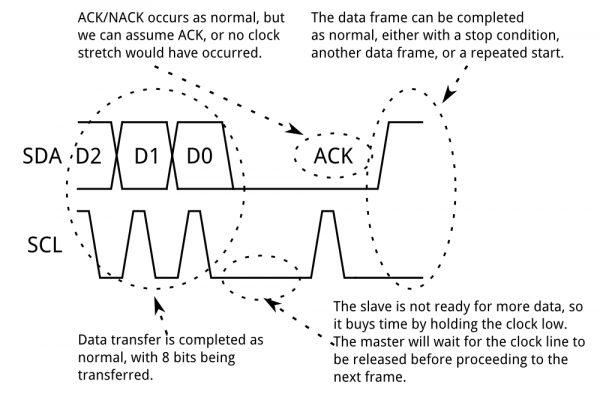


**Sometimes, it is important that a master device be allowed to exchange several messages in one go, without allowing other master devices on the bus to interfere. For this reason, the repeated start condition has been defined.**

**To perform a repeated start, SDA is allowed to go high while SCL is low, SCL is allowed to go high, and then SDA is brought low again while SCL is high. Because there was no stop condition on the bus, the previous communication wasn’t truly completed and the current master maintains control of the bus.**

**At this point, the next message can begin transmission. The syntax of this new message is the same as any other message–an address frame followed by data frames. Any number of repeated starts is allowed, and the master will maintain control of the bus until it issues a stop condition.**

#### Clock stretching



**At times, the master’s data rate will exceed the slave’s ability to provide that data. This can be because the data isn’t ready yet (for instance, the slave hasn’t completed an analog-to-digital conversion yet) or because a previous operation hasn’t yet completed (say, an EEPROM which hasn’t completed writing to non-volatile memory yet and needs to finish that before it can service other requests).**

**In this case, some slave devices will execute what is referred to as “clock stretching”. Nominally, all clocking is driven by the master device–slaves simply put data on the bus or take data off the bus in response to the master’s clock pulses. At any point in the data transfer process, an addressed slave can hold the SCL line low after the master releases it. The master is required to refrain from additional clock pulses or data transfer until such time as the slave releases the SCL line.**

**Conclusion** :



An autopilot is a mechanical, electrical, or hydraulic system used to guide a vehicle With no assistance from a human being. Many projects are based on different type of vehicles. One of the powerful rotor-crafts is aquad-rotor.