Using the MPU-6050

Inertia Measurement Systems

Gyroscopes & Accelerometers

Sensor fusion

I2C

MPU-6050

IMUs

- There are small devices indicating changing orientation in smart phones, video game remotes, quad-copters, etc.
- These devices contains gyroscopes combined with accelerometers and/or compasses and are referred to as an *IMU*, or *Inertial Measurement Unit*
- The number of sensor inputs in an IMU are referred to as "DOF" (Degrees of Freedom), so a chip with a 3-axis gyroscope and a 3-axis accelerometer would be a 6-DOF IMU.

Accelerometers

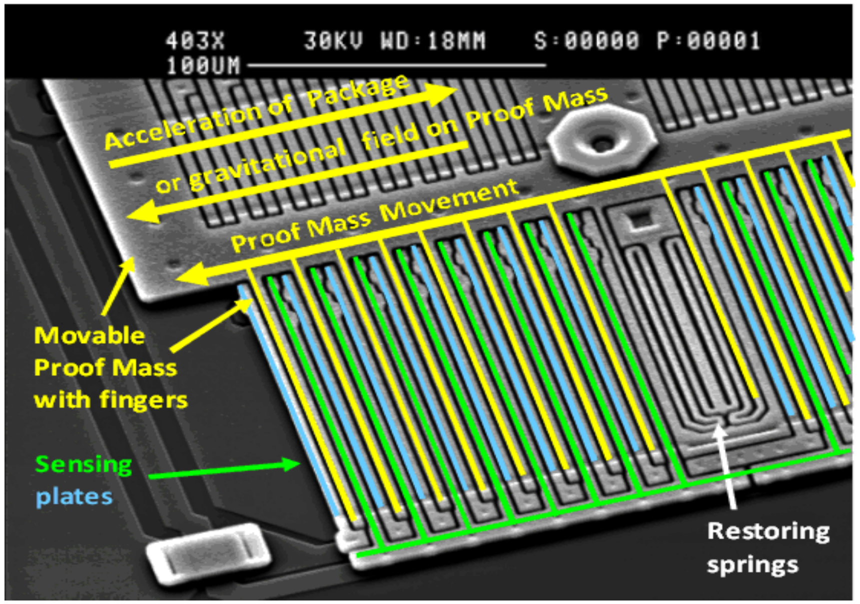
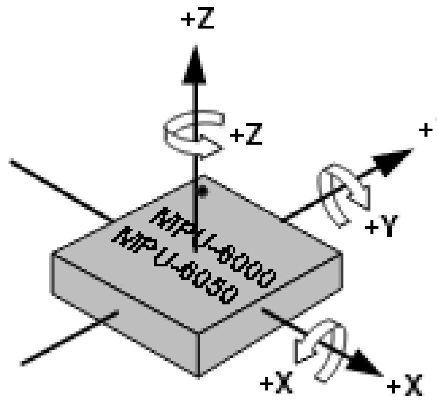

403X 30KV WD:18MM S:00000 P:00001
100UM
Acceleration of Package
or gravitational field on Proof Mass
Proof Mass Movement
Movable Proof Mass with fingers
Sensing plates
Restoring springs

Image credit:
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf

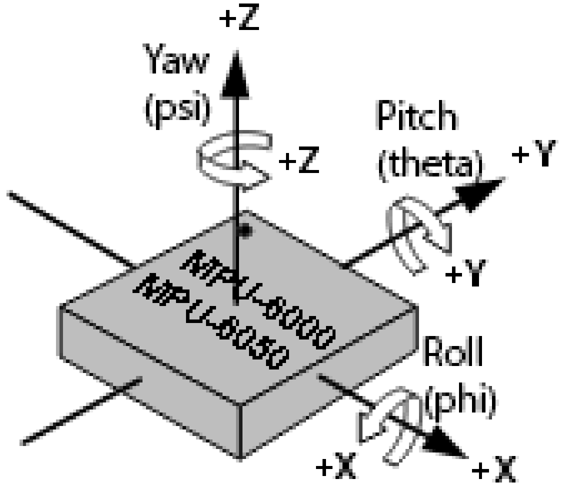


- Proof mass deflection is measured as a change in capacitance between the proof mass and sensing plates
- Internal circuitry converts the tiny capacitance to a voltage signal which is digitized and output

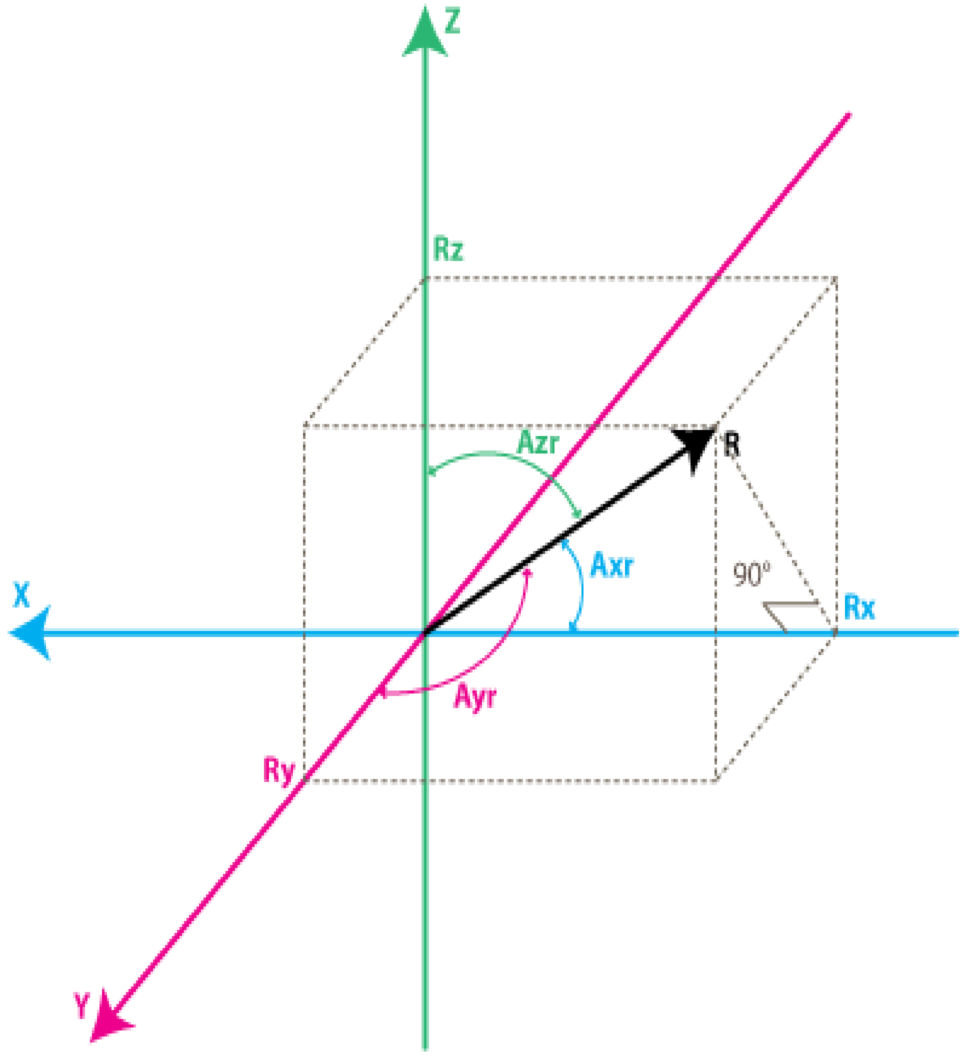
- Each accelerometer has a zero-g voltage level, you can find it in spec
- Accelerometers also have a sensitivity, usually expressed in mV/g
- Divide the zero-g level corrected reading by the sensitivity to produce the final reading

Accelerometers

- Computing orientation from an accelerometer relies on a constant gravitational pull of 1g (9.8 m/s^2) downwards
- If no additional forces act on the accelerometer (a risky assumption), the magnitude of the acceleration is 1g, and the sensor's rotation can be computed from the position of the acceleration vector
- If the Z-axis is aligned along the gravitational acceleration vector, it is impossible to compute rotation around the Z-axis from the accelerometer.
- Digital accelerometers give information using a serial protocol like I2C , SPI or USART; analog accelerometers output a voltage level within a predefined range



Accelerometer



- cos(Axr) = Rx / R
  cos(Ayr) = Ry / R
  cos(Azr) = Rz / R
- R = SQRT(Rx^2 + Ry^2 + Rz^2)
- Find angles by using arccos() function (the inverse cos() function):
- Axr = arccos(Rx/R)
  Ayr = arccos(Ry/R)
  Azr = arccos(Rz/R)

Image credit: http://www.starlino.com/imu_guide.html

Gyroscope

- A gyroscope measures *angular velocity (the rate of change in orientation angle),* not angular orientation itself
- Must first initialize the sensor position with a known value (possibly from the accelerometer), then measure the angular velocity (ω) around the X, Y and Z axes at measured intervals (Δt)
  - $\omega \times \Delta t$ = change in angle
  - The new orientation angle is the original angle plus this change
- This is *integrating* - adding up many small computed intervals - to find orientation
  - Repeatedly adding up increments of $\omega \times \Delta t$ results in small systematic errors becoming magnified over time
  - *Gyroscopic drift*---over long timescales the gyroscope data will become increasingly inaccurate

Gyroscope

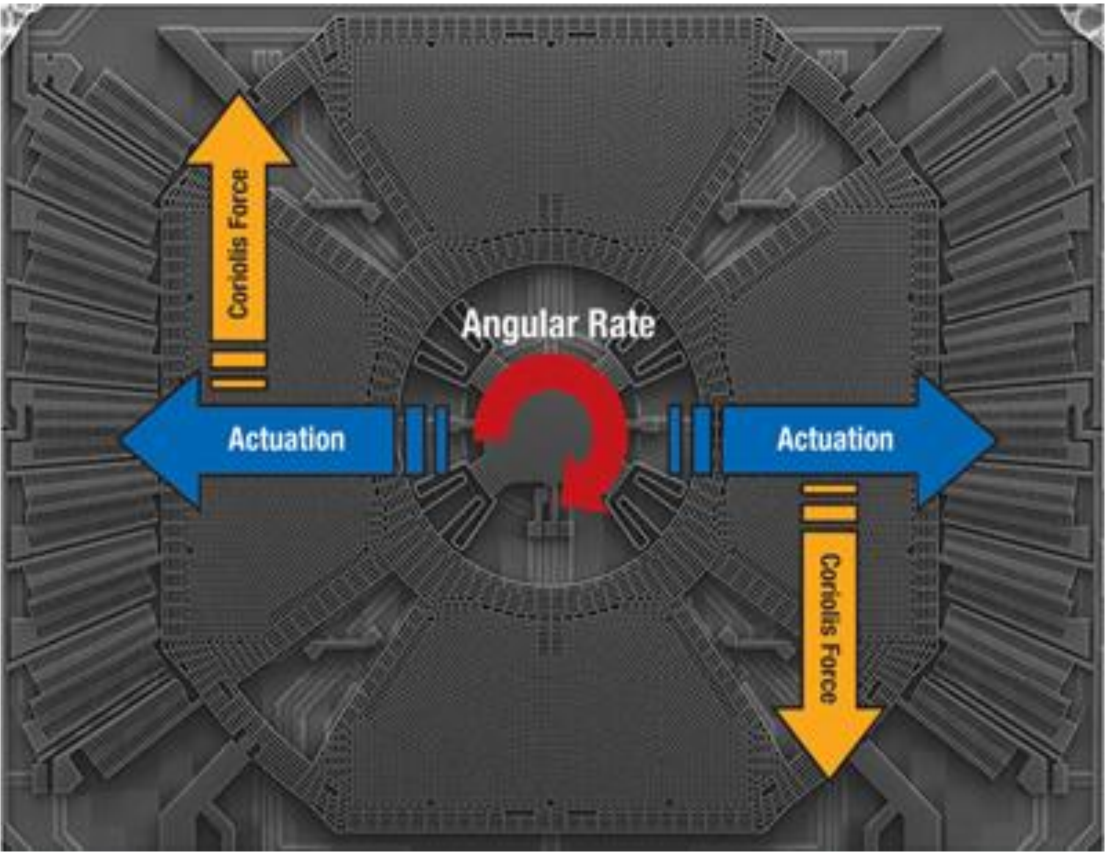
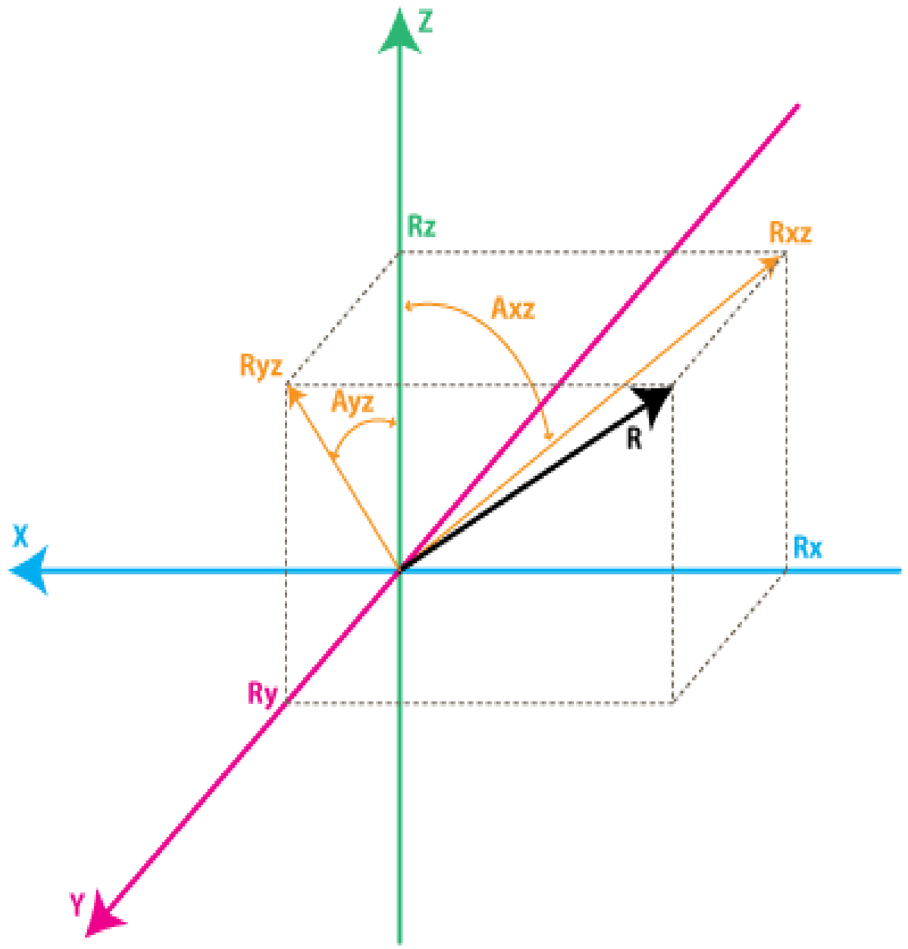


Image credit:
http://www.digikey.com/us/en/techzone/sensors/resources/articles/MEMS-Accelerometers.html

- Uses Coriolis effect to transform an angular velocity into a displacement
- The Coriolis force acts perpendicular to the rotation axis and to the velocity of the body in the rotating frame
  - **$F_c = -2m\,\Omega \times v$**
- The displacement induces a change in capacitance between the mass and the housing, thus transforming the angular rate input to the gyroscope into an electrical output

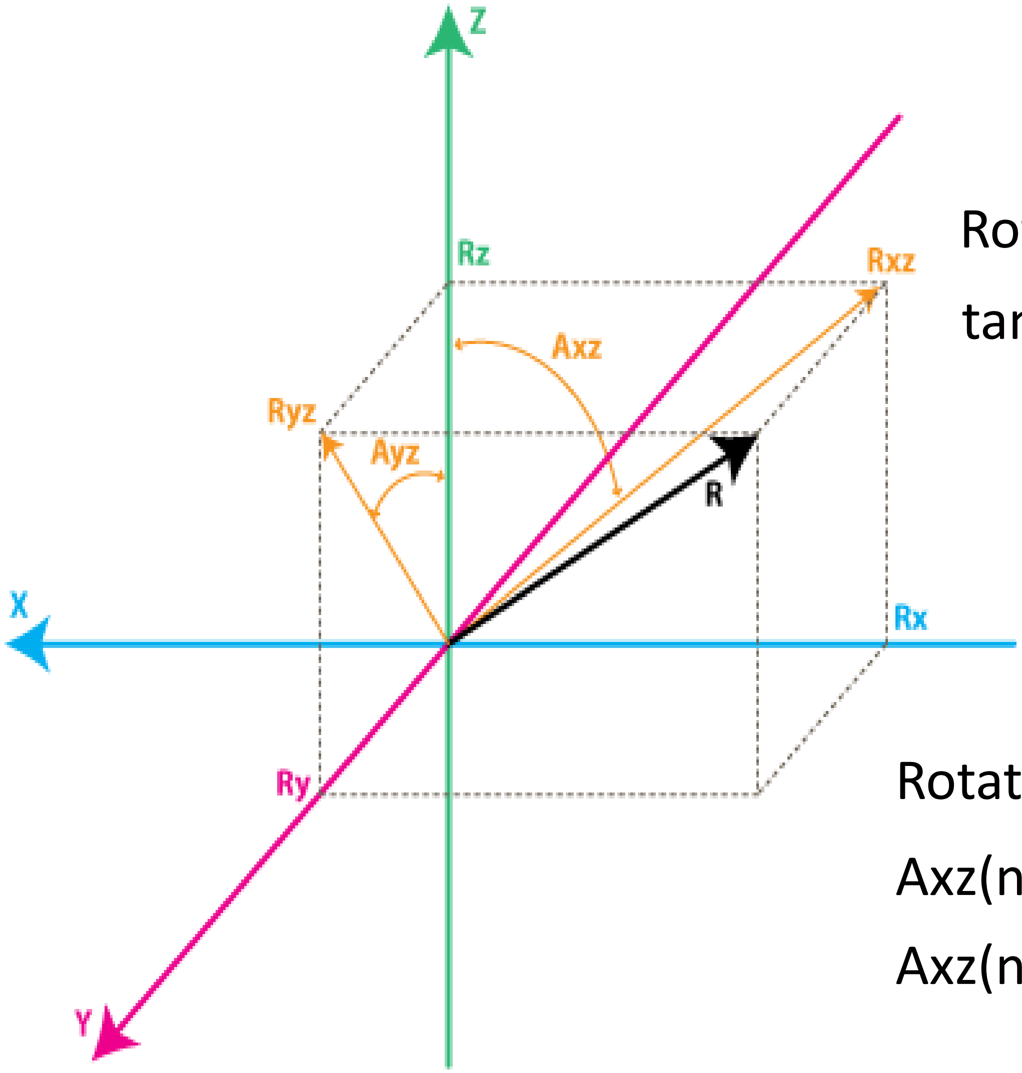
Gyroscopes



- Each gyroscope measures the rotation around one axis
- Axz – is the angle between the Rxz (projection of R on XZ plane) and Z axis
- Ayz – is the angle between the Ryz (projection of R on YZ plane) and Z axis
- Gyroscopes measure the rate of change of these angles.

Image credit: http://www.starlino.com/imu_guide.html

Computing Rotation Angles



Rotation from accelerometer data:

tan(Axz) = Rx/Rz => Axz = atan2(Rx,Rz)

Rotation from gyroscope data:

Axz(n-1) = atan2(RxEst(n-1) , RzEst(n-1))

Axz(n) = Axz(n-1) + RateAxz(n) * T

Image credit: http://www.starlino.com/imu_guide.html

Sensor Fusion

- An accelerometer measures inertial force, such as gravity (and ideally only by gravity), but it might also be caused by acceleration (movement) of the device. Even if the accelerometer is relatively stable, it is very sensitive to vibration and mechanical noise.
- A gyroscope is less sensitive to linear mechanical movements, the type of noise that accelerometer suffers from. Gyroscopes have other types of problems like drift (not coming back to zero-rate value when rotation stops).
- Averaging the data that comes from accelerometers and gyroscopes can produce a better estimate of orientation than obtained using accelerometer data alone.

Fusion Algorithms

- Several choices: Kalman Filter, Complementary Filter, …
- Combine orientation estimated from Accelerometer readings with that estimated from the Gyroscope readings
- Racc – current readings from accelerometer Rgyro – obtained from Rest(n-1) and current gyroscope readings
- A weighted average:

  Rest(n) = (Racc * w1 + Rgyro * w2) / (w1 + w2)

Sensor Fusion

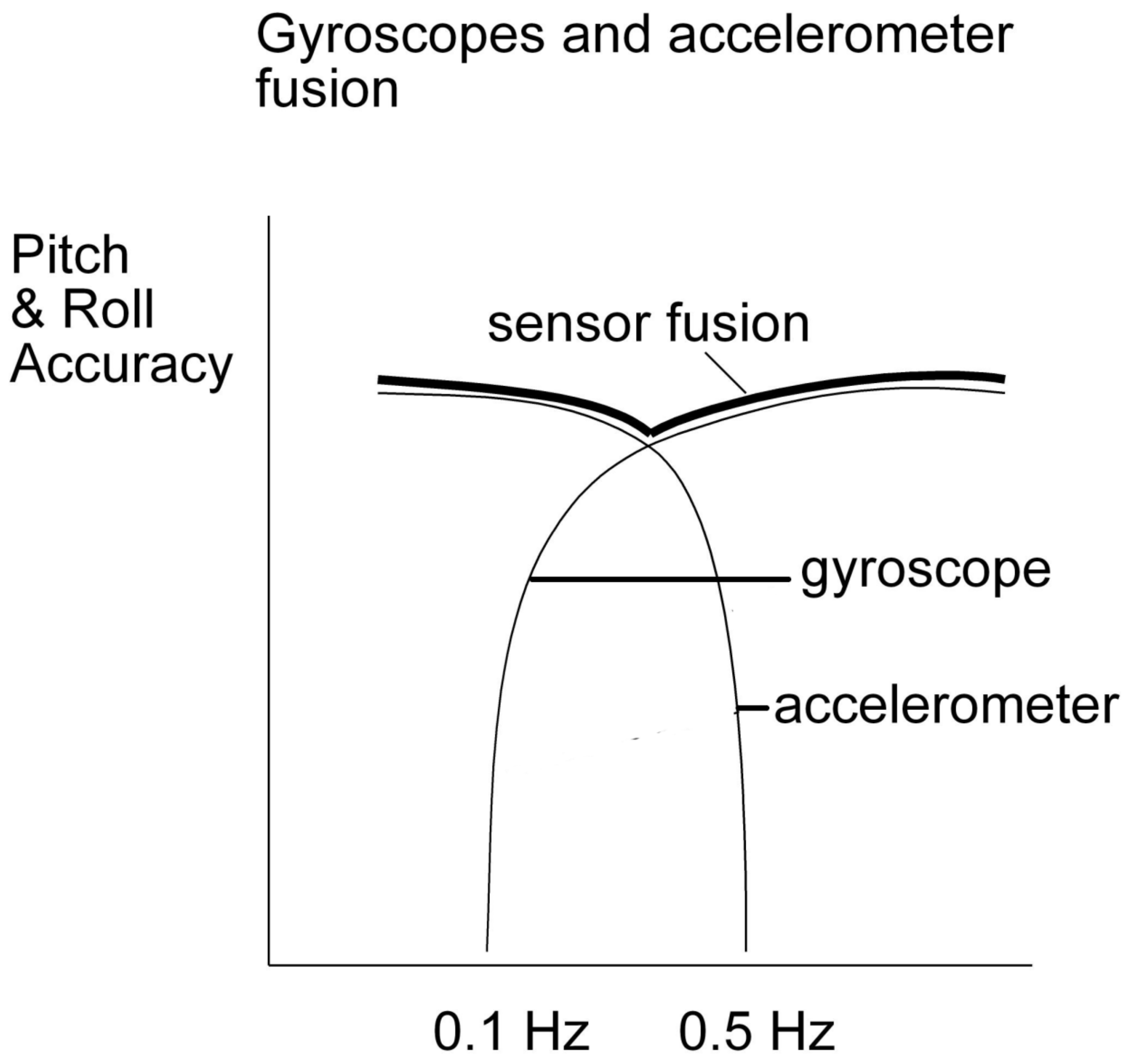


Image credit: http://memscentral.com/Secondlayer/ mems_motion_sensor_perspectives-sensor-susion-high-res.htm

MPU-6050

- The MPU-6050 is the world’s first integrated 6-axis MotionTracking device
- It combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package.
- It uses a standard I2C bus for data transmission.
  - With it’s I2C bus, it can accepts inputs from an external 3-axis compass to provide a complete 9-axis MotionFusion output.
- A number of different breakout boards are available containing the MPU-6050 chip, we have the GY-521.

I^2C

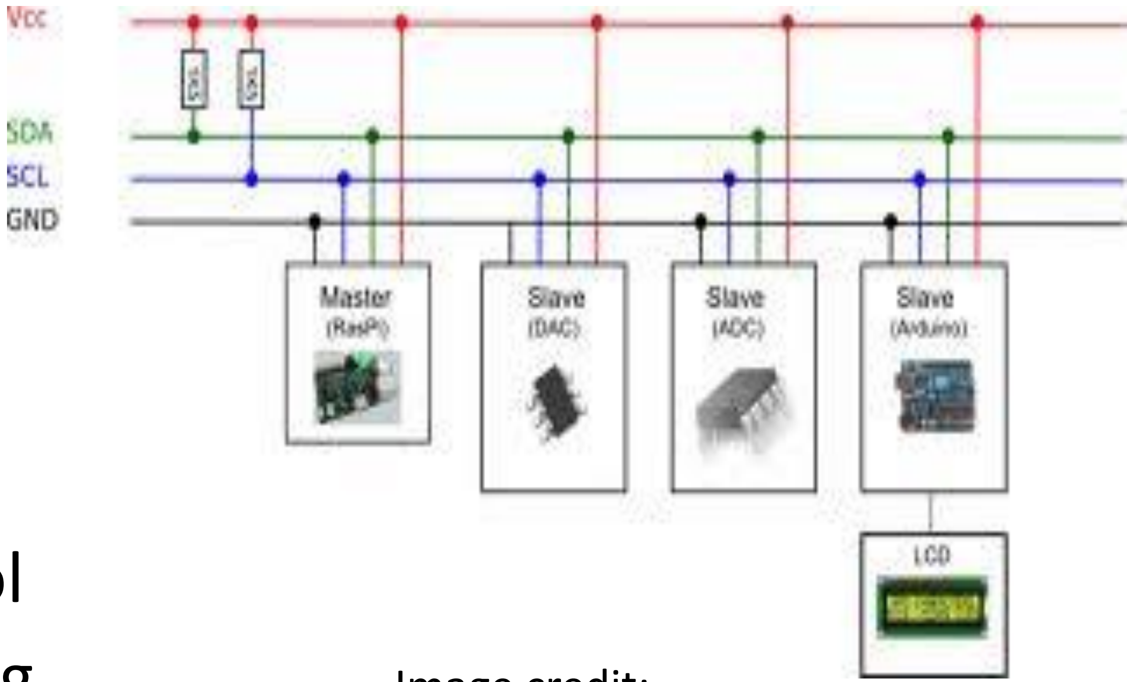


Image credit: http://quick2wire.com/articles/i2c-and-spi/

- Understanding I2C
  - The physical I2C bus
  - Masters and Slaves
  - The physical protocol
  - I2C device addressing
  - The software protocol
  - I2C support in the WiringPi Library
- A good Tutorial

The physical I2C bus

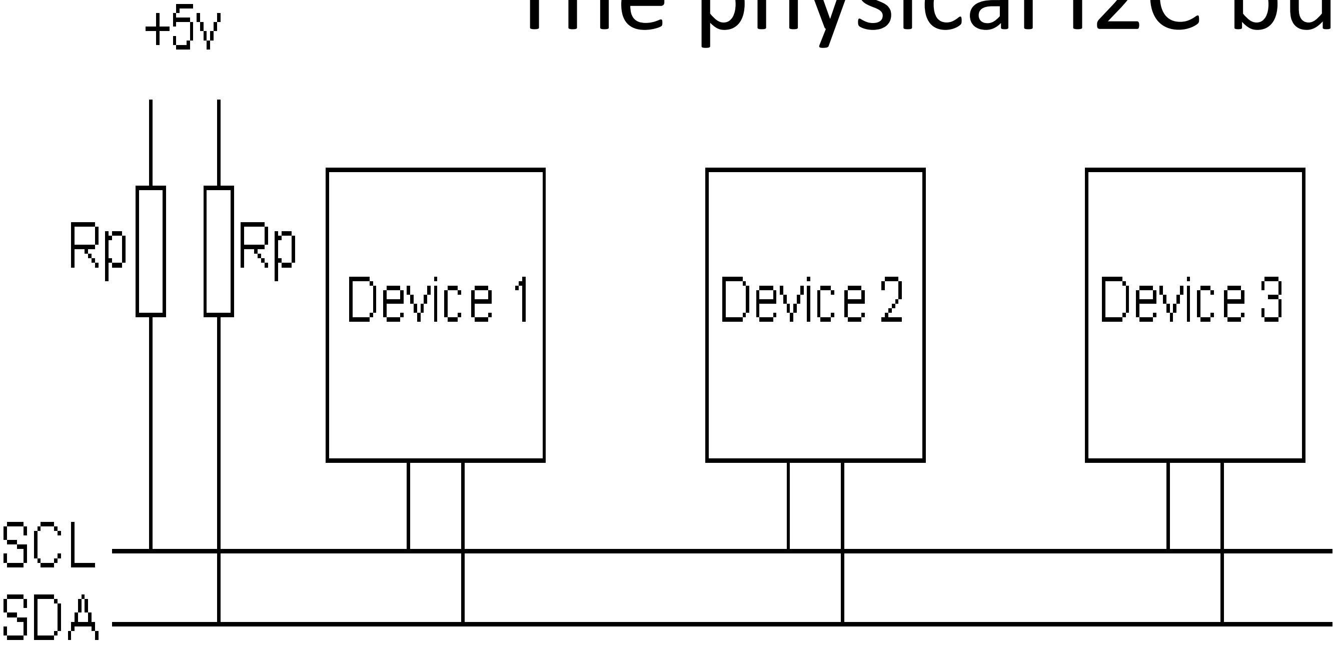


Image credit: http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

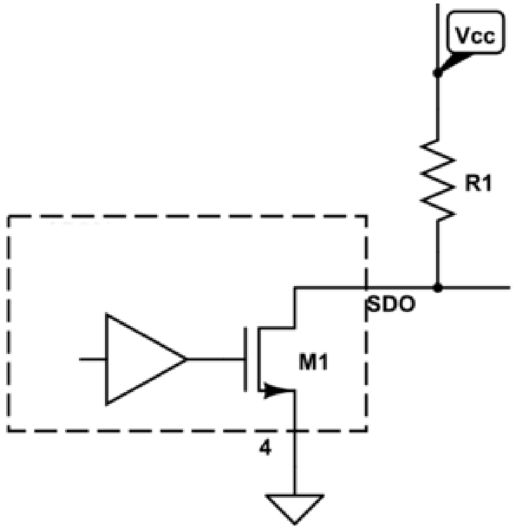


Image credit: http://electronics.stackexc hange.com/questions/703 12/n-ch-fet-with-open-drain-output

- Two wires: SCL and SDA
  - SCL is the clock line: used to synchronize all data transfers
  - SDA is the data line
- Both SCL and SDA lines are "open drain" drivers
  - Can only be driven low
  - For the line to go high provide a pull-up resistors to 5v

Masters and Slaves

- The devices on the I2C bus are either masters or slaves
- The master drives the clock & initiates the transfers
- Multiple slaves on the I2C bus, but there is typically only one master.
- Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

The I2C Physical Protocol

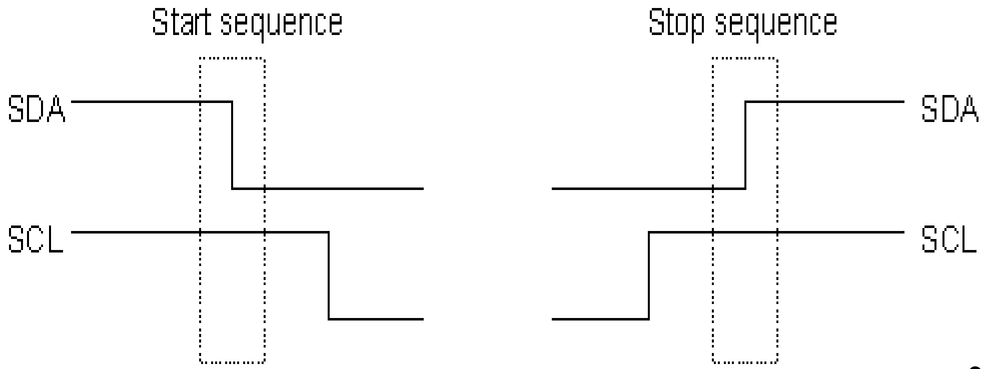
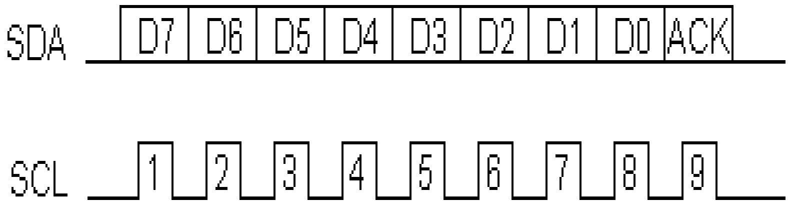


Image credit: http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html



- Start and stop sequences mark the beginning and end of a transaction
  - Initiated by the master
- The only time the SDA (data line) is changed while the SCL (clock line) is high.
- During data transfer, SDA must not change while SCL is high

- Data is transferred in sequences of 8 bits.
- Bits are sent with the MSB (Most Significant Bit) first.
- The SCL line is pulsed high, then low for each bit
- After each 8 bits transfer, the slave sends back an acknowledge bit
  - It takes 9 SCL clock pulses to transfer 8 bytes of data
- The standard clock (SCL) speed for I2C is up to 100KHz

I2C Device Addressing

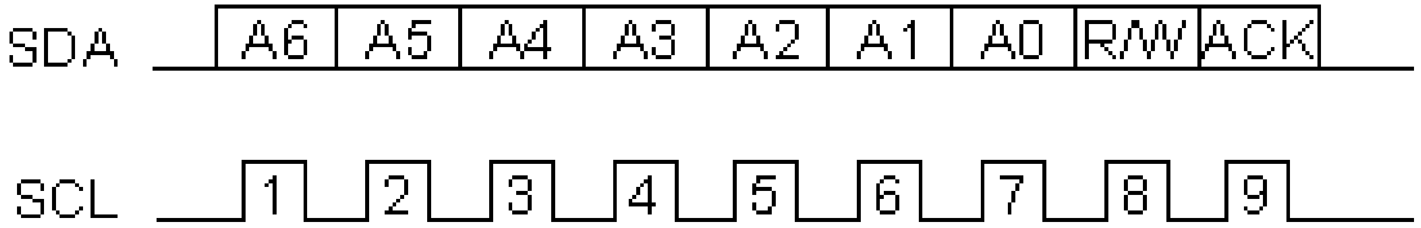


Image credit: http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

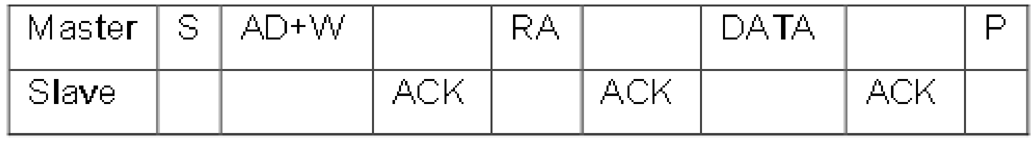
- All I2C addresses are 7 bits or 10 bits---most are 7 (ours are)
  - Can have up to 128 devices on the I2C bus
- Addresses are still sent in 8 bits
  - The extra bit (the last bit) indicates read or write
    - If the bit is zero the master is writing to the slave.
    - If the bit is 1 the master is reading from the slave

The I2C Write Protocol

- Procedure to write to a slave device:
  1. Send a start sequence
  2. Send the I2C address of the slave with the R/W bit low (even address)
  3. Send the internal register number you want to write to
  4. Send the data byte
  5. [Optionally, send any further data bytes]
     - slave will automatically increment the internal register address after each byte
  6. Send the stop sequence.

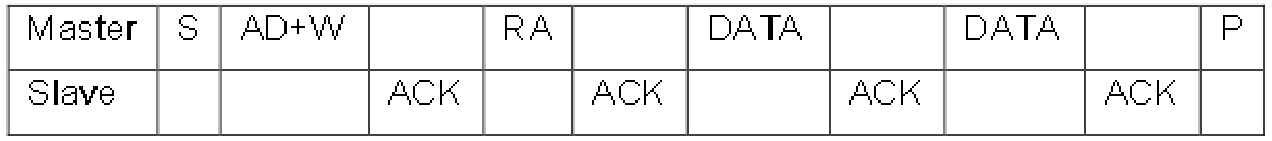
The I2C Write Protocol

Single-Byte Write Sequence

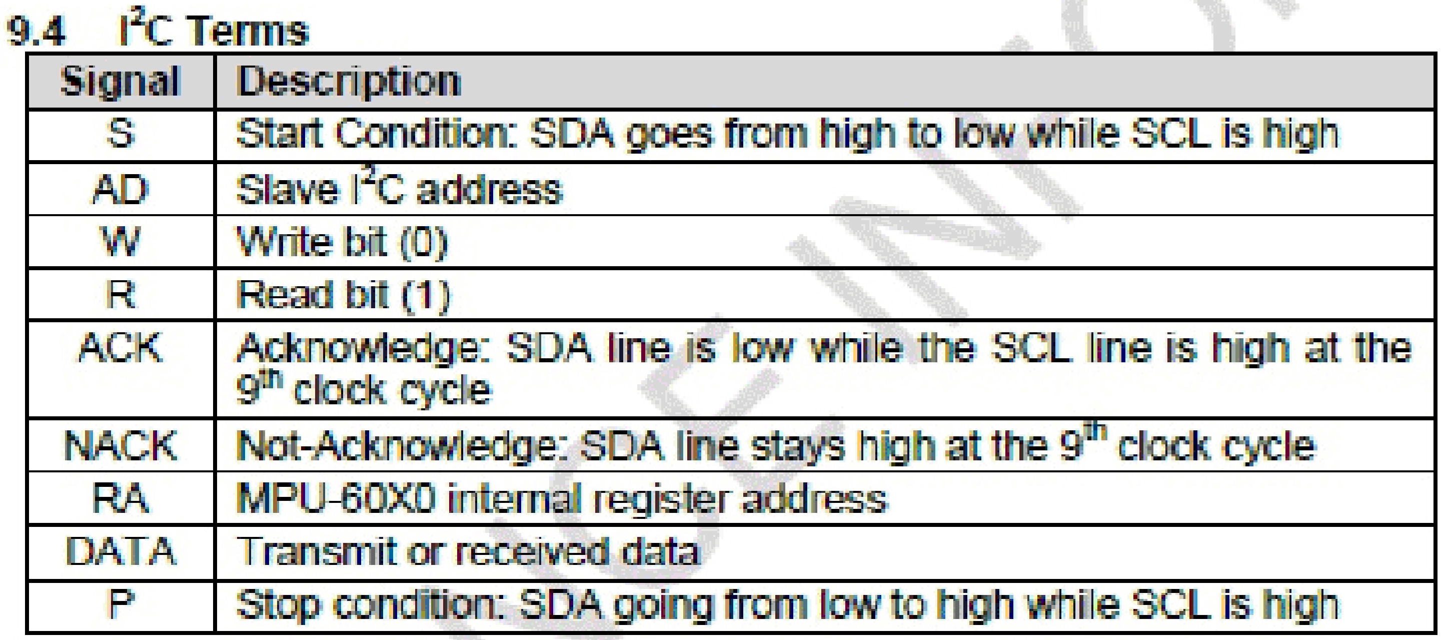


Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence



Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	



9.4 I²C Terms

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I²C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9th clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

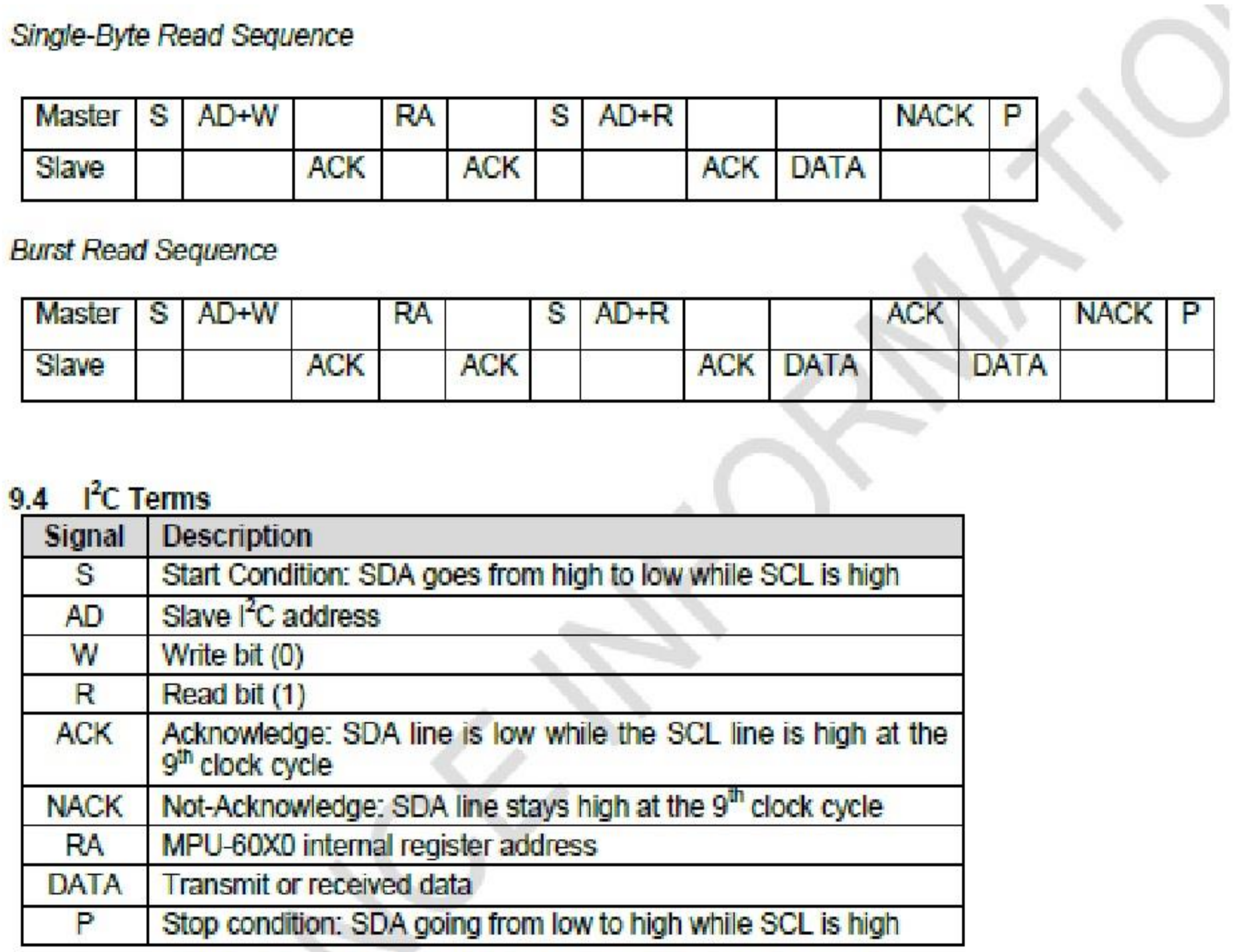
Image credit: http://invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf

The I2C Read Protocol

- A read is more complicated
  - Before reading data from a slave device, you must tell it which of its internal addresses you want to read
  - A read starts off by writing to the slave
- Procedure

1. Send a start sequence
2. Send I2C address of the device with the R/W bit low (even address)
3. Send the Internal register address
4. Send a start sequence again (repeated start)
5. Send the I2C address of the device with the R/W bit high (odd address)
6. Read data byte from the register
7. Send the stop sequence.

The I2C Read Protocol



Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

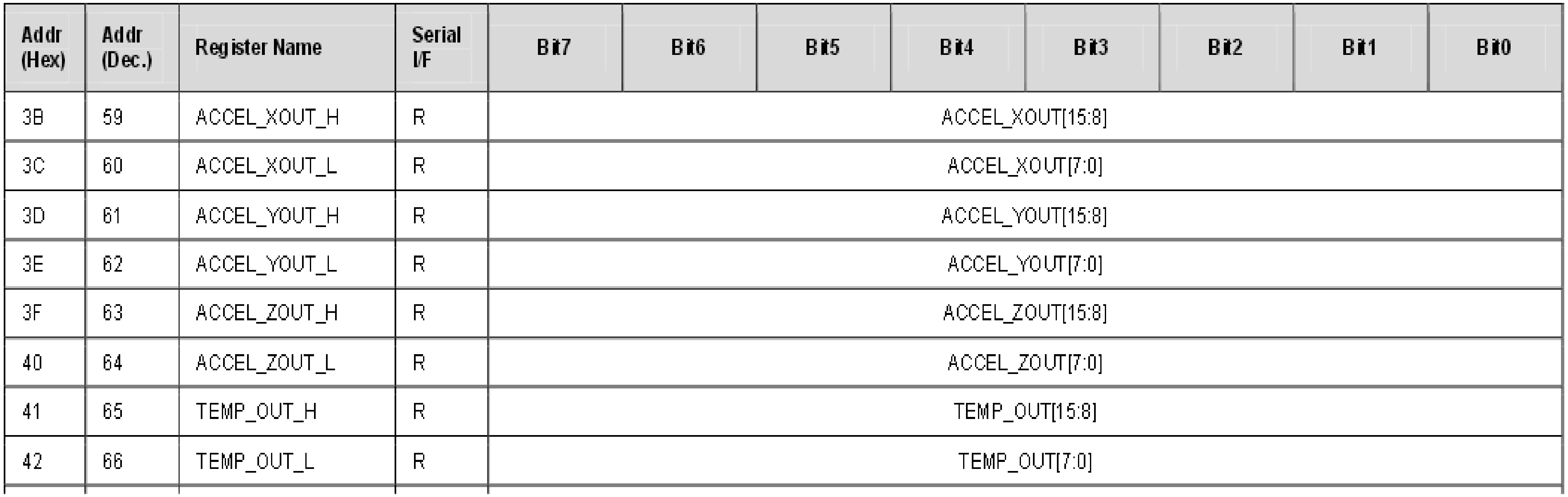
Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

9.4 I²C Terms

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I²C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9th clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

Image credit: http://invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf

ACC Data Registers on MPU 6050



Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							

For MPU-6050:

- ACCEL_XOUT_H → register 3B
- ACCEL_XOUT_L → register 3C
- ACCEL_YOUT_H → register 3D
- ACCEL_YOUT_L → register 3E
- ACCEL_ZOUT_H → register 3F
- ACCEL_ZOUT_L → register 40