# LPC2148 I2C

# Introduction

I2C (Inter Integrated Circuit) is serial bus interface connection protocol first invented by Philips Semiconductor (Now NXP Semiconductors). It is also called as TWI (two wire interface) since it uses only two wires for communication. I2C uses handshaking mechanism for communication. Hence, it is also called as acknowledgment based communication protocol.

I2Cworks in two modes namely,

* Master Mode  
         Master is responsible for generating clock and initiating communication
* Slave Mode  
         Slave receives the clock and responds when addressed by the Master

Two wires of the I2C interface are SDA (serial data) and SCL (serial clock).

* SDA is Serial Data wire used for data transfer in between master and slave
* SCL is Serial Clock wire used for clock synchronization. Clock is provided by the master

There are two types of data transfer possible on I2C interface depending on Read/Write operation i.e.

Data transfer from Master Transmitter to Slave Receiver and

Data transfer from Slave Transmitter to Master Receiver

LPC2148 I2C

* LPC2148 has two I2C interfaces. i.e. I2C0 & I2C1
* It has Programmable clock to support adjustable I2C transfer rates
* LPC2148 I2C is byte oriented and has four operating modes :  
  Master Transmitter mode  
  Master Receiver mode  
  Slave Transmitter mode  
  Slave Receiver mode

LPC2148 I2C Pins

SCL0: - Serial Clock pin of I2C0.

SDA0: - Serial Data pin of I2C0.

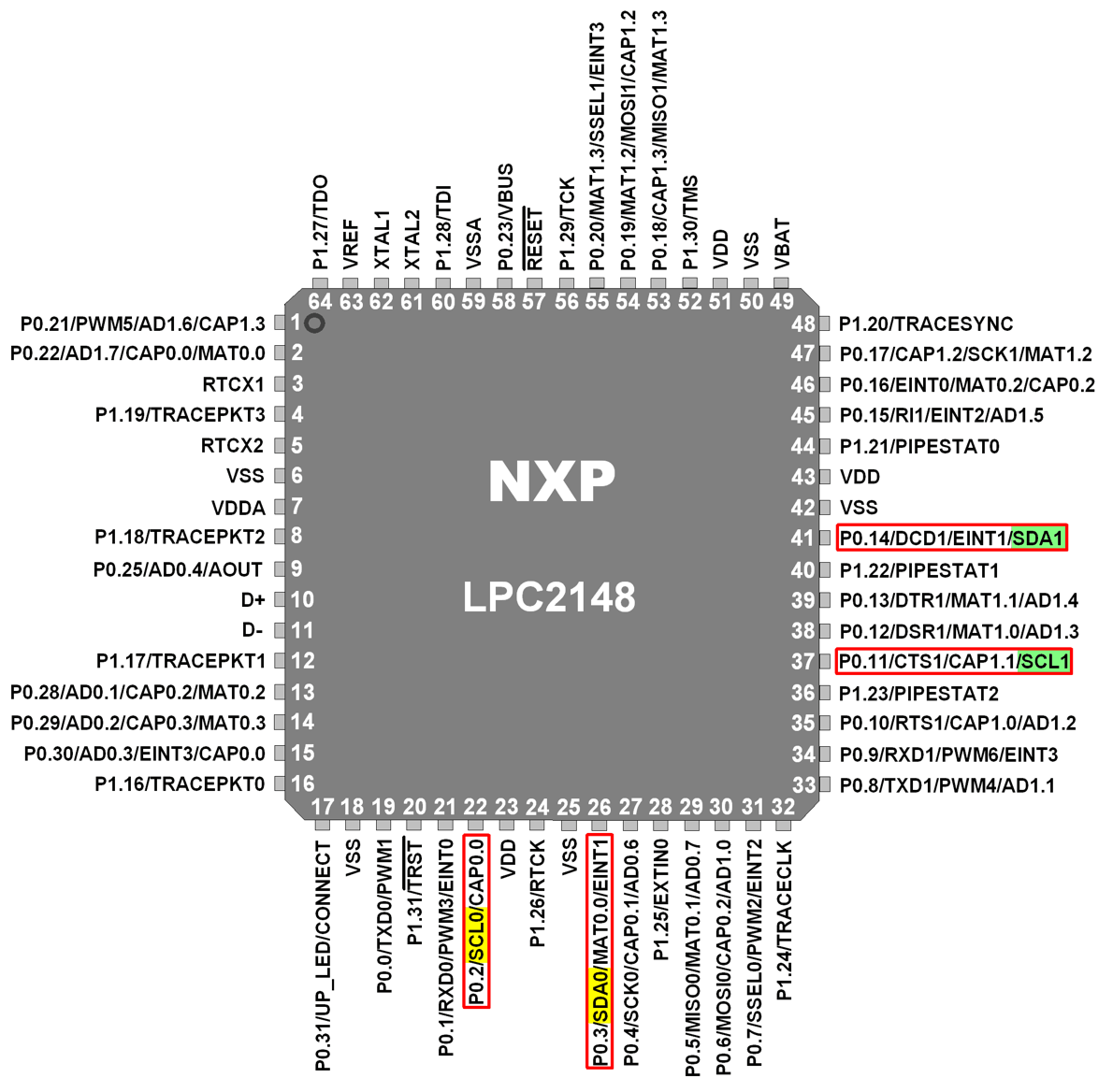
SCL1: - Serial Clock pin of I2C1.

SDA1: - Serial Data pin of I2C1.

Let’s see the registers that are used to initialize and control the operation of LPC2148 I2C.

I2C0 and I2C1 have similar register formats and register names. The only difference in their register names is the I2C0 or I2C1 acronym at the beginning of the register name.

Here we will be showing I2C0 registers. Same can be used for I2C1 by changing their name (Replace the I2C0 part by I2C1).

LPC2148 I2C Pins

# I2C Registers

1.  I2C0CONSET (I2C0 Configuration Set Register)

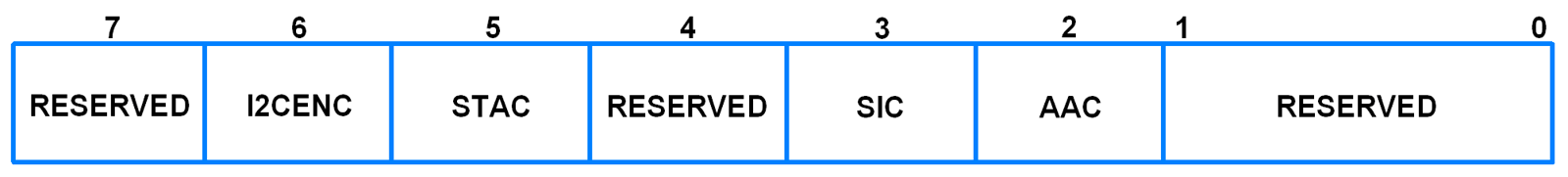
* It is an 8-bit read-write register.
* It is used to control the operation of the I2C0 interface.

 I2C0CONSET (I2C0 Configuration Set Register)

* Writing a 1 to a bit in this register causes corresponding bit in the I2C control register to set.
* Writing a 0 has no effect.
* Bit 2 – AA (Assert Acknowledge Flag)  
  When set to 1, Acknowledge (SDA LOW) is returned during acknowledge clock pulse on SCL. Otherwise, Not acknowledge (SDA HIGH) is returned.
* Bit 3 – SI (I2C Interrupt Flag)  
  This bit is set when the I2C state changes. Else it remains reset.
* Bit 4 – STO (Stop Flag)  
  In Master Mode, setting this bit causes the I2C interface to transmit a STOP condition. In Slave Mode, setting this bit causes recover from an error condition if any.  
  This bit is cleared by hardware automatically.
* Bit 5 – STA (Start Flag)  
  Setting this bit causes the I2C interface to enter in master mode and transmit a START condition or transmit a repeated START condition if it is already in master mode.
* Bit 6 – I2CEN (I2C Interface Enable)  
  Set this bit to enable I2C interface.

2.  I2C0CONCLR (I2C0 Configuration Clear Register)

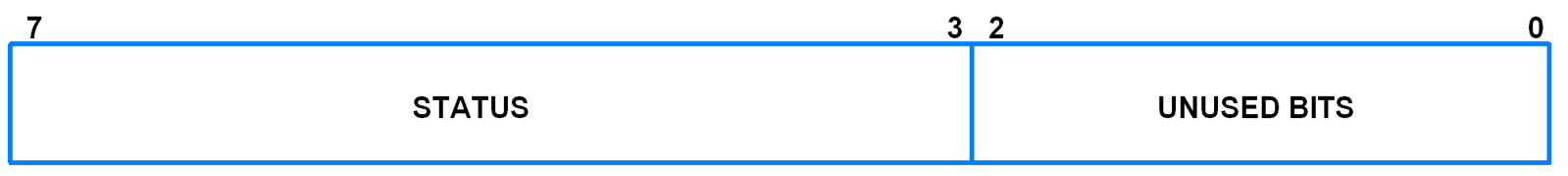
* It is an 8-bit write only register.

 I2C0CONCLR (I2C0 Configuration Clear Register)

* Writing a 1 to a bit in this register causes corresponding bit in the I2C control register to clear.
* Writing a 0 has no effect.
* Bit 2 – AAC (Assert Acknowledge Flag Clear)  
  Setting this bit clears the AA bit in I2C0CONSET register.
* Bit 3 – SIC (I2C Interrupt Flag Clear)  
  Setting this bit clears the SI bit in I2C0CONSET register.
* Bit 5 – STAC (Start Flag Clear)  
  Setting this bit clears the STA bit in I2C0CONSET register.
* Bit 6 – I2CENC (I2C Interface Disable)  
  Setting this bit clears the I2CEN bit in I2C0CONSET register.

3.  I2C0STAT (I2C0 Status Register)

* It is an 8-bit read only register.
* It reflects the present status of the I2C interface.

 I2C0STAT (I2C0 Status Register)

* Bit 2:0 – Unused bits  
  These are unused bits and are always 0.
* Bit 7:3 – Status  
  These are status bits which provide the status of the current event on I2C bus.  
  There are 26 possible status codes.  
  Example, I2C0STAT = 0x08. This code indicates that a start condition has been transmitted.  
  For other status codes, refer Table 232 (Page 228) to table 235 of the datasheet given in the attachments section below.

4.  I2C0DAT (I2C0 Data Register)

* It is an 8-bit read-write register.

 I2C0DAT (I2C0 Data Register)

* It contains the data to be transmitted or received.
* This register can be read or written to only when SI = 1.

5.      I2C0SCLL (I2C0 SCL Low Duty Cycle Register)

* It is a 16-bit register.



I2C0SCLL (I2C0 SCL Low Duty Cycle Register)

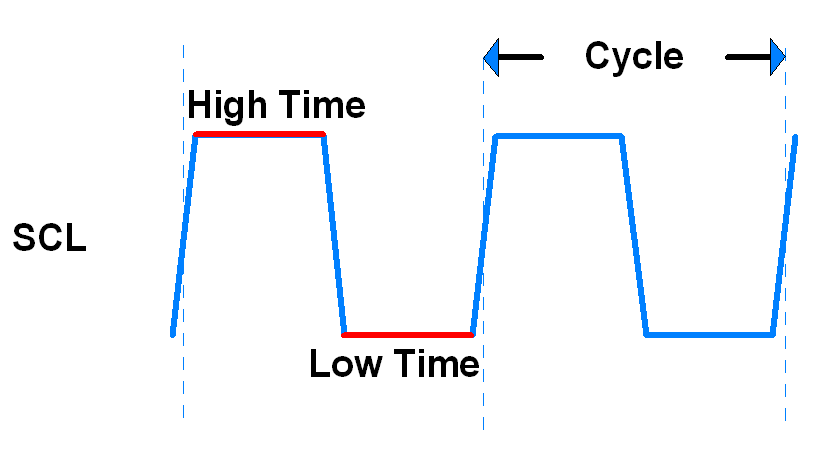
* This register contains the value for SCL Low time of the cycle.

6.  I2C0SCLH (I2C0 SCL High Duty Cycle Register)

* It is a 16-bit register.

 I2C0SCLH (I2C0 SCL High Duty Cycle Register)

* This register contains the value for SCL High time of the cycle.



The frequency and duty cycle of SCL is decided using I2C0SCLL and I2C0SCLH. I2C0SCLH contains the TON (High) time and I2C0SCLL contains the TOFF (Low) time.

The frequency is calculated as follows:



Example: Assume FPCLK = 30MHz, I2CBitFrequency = 300KHz, Duty Cycle = 50%.

As duty cycle is 50%, I2C0SCLL = I2C0SCLH.

Hence,  

Hence, I2C0SCLL = I2C0SCLH = 50

7.  I2C0ADR (I2C0 Slave Address Register)

* It is an 8-bit read-write register.

 I2C0ADR (I2C0 Slave Address Register)

* It is only used when device is in Slave Mode.
* Bit 0 – GC (General Call)  
  When this bit is set, the general call address (0x00) is recognized.
* ​​​​​​​Bit 7:1 – Address  
  It contains the I2C device address for slave mode.

# Programming I2C0 in LPC2148

We will see how to use I2C0 in master mode to read and write data to and from a slave I2C device.

1.  Initialization

* Configure P0.2 and P0.3 as SCL0 and SDA0 using PINSEL0 register.
* Enable I2C interface using the I2C0CONSET register.
* Select appropriate values of I2C0SCLL and I2C0SCLH depending on the duty cycle and frequency of I2C SCL required.

void I2C\_INIT(void)

{

PINSEL0 = PINSEL0 | 0x00000050; /\* Configure P0.2 and P0.3 as SCL0 and SDA0 respectively \*/

I2C0CONSET = 0x40; /\* Enable I2C \*/

I2C0SCLL = 0x32; /\* I2C data rate 300Khz and 50% duty cycle \*/

I2C0SCLH = 0x32; /\* I2C data rate 300Khz and 50% duty cycle \*/

}

2. Start

* Using I2C0CONSET register, transmit Start condition.
* Wait till SI = 1, i.e. till start condition transmission is completed.
* Clear the Start and SI bits using I2C0CONCLR register.

void I2C\_START(void)

{

I2C0CONSET = 0x20; /\* Set Start bit for Start condition \*/

while ( (I2C0CONSET & 0x08) == 0 ); /\* Wait till SI = 1 \*/

I2C0CONCLR = 0x28; /\* Clear Start bit and SI bit \*/

}

3.  Write

* Load the data to be written into the data register.
* Enable I2C interface using I2C0CONSET register.
* Wait till SI = 1, i.e. till write complete.
* Clear the SI bit using I2C0CONCLR register.

void I2C\_WRITE( char data )

{

I2C0DAT = data; /\* Load data to be written into the data register \*/

I2C0CONSET = 0x40; /\* Enable I2C \*/

while( (I2C0CONSET & 0x08) == 0 ); /\* Wait till SI = 1 \*/

I2C0CONCLR = 0x08; /\* Clear SI bit \*/

}

4. Read

* Enable I2C with or without acknowledge depending on whether acknowledge is required.
* Wait till SI = 1, i.e. till reception is complete.
* Clear SI ( and AA if acknowledge is enabled ) bit using I2C0CONCLR register.
* Read the data in the data register.

unsigned char I2C\_READ\_ACK( void )

{

I2C0CONSET = 0x44; /\* Enable I2C with Acknowledge \*/

while( (I2C0CONSET & 0x08) == 0 ); /\* Wait till SI = 1 \*/

I2C0CONCLR = 0x0C; /\* Clear SI and Acknowledge bits \*/

return I2C0DAT; /\* Return received data \*/

}

unsigned char I2C\_READ\_NACK( void )

{

I2C0CONSET = 0x40; /\* Enable I2C without Acknowledge \*/

while( (I2C0CONSET & 0x08) == 0 ); /\* Wait till SI = 1 \*/

I2C0CONCLR = 0x08; /\* Clear SI bit \*/

return I2C0DAT; /\* Return received data \*/

}

void I2C\_MULTIREAD( char\* arr , int bytes ) /\* For reading multiple bytes at a time \*/

{

uint8\_t i = 0;

while( ( bytes - 1 ) != 0 )

{

I2C0CONSET = 0x44; /\* Enable I2C with Acknowledge \*/

while( (I2C0CONSET & 0x08) == 0 ); /\* Wait till SI = 1 \*/

I2C0CONCLR = 0x0C; /\* Clear SI and Acknowledge bits \*/

\*( arr + i ) = I2C0DAT ;

bytes--;

i++;

}

I2C0CONSET = 0x40; /\* Enable I2C without Acknowledge \*/

while( (I2C0CONSET & 0x08) == 0 ); /\* Wait till SI = 1 \*/

I2C0CONCLR = 0x08; /\* Clear SI bit \*/

\*( arr + i ) = I2C0DAT ;

}

5. Stop

* Using I2C0CONSET register, transmit Stop condition.

void I2C\_STOP( void )

{

I2C0CONSET = 0x50; /\* Set Stop bit for Stop condition \*/

}

**How I2C protocol work** – The I2C protocol requires only 2 signals: clock and data.  Clock is known as SCL or SCK (for Serial Clock), while data is known as SDA (for Serial Data).  What makes I2C unique is the use of special combinations of signal conditions and changes.  Fundamentally, there are just two: Start and Stop.  A ‘START” condition is generated by the Master, followed by 7 bits of address, then a ReadWrite bit.  If a slave device detects an address match, it will send an ACK by driving SDA low during the next clock cycle; if no slave recognizes the address then the SDA line will be left alone to be pulled up high.  Following a successful ACK, data will be either sent to the slave device or read from the slave device (depending on what was indicated by the Read/Write bit).  Therefore, each byte is 9 bits: either 7 address plus one R/W plus one ACK/NAK, or 8 data plus one ACK/NAK.  The last data byte of a transaction should generally be followed by a NAK, to indicate that it is intended to be the final byte.  After this, either a STOP or a ReSTART should be issued by the Master. Bus errors are rarely introduced when using a dedicated I2C peripheral on the Master.

****

**Feature of I2C protocol**

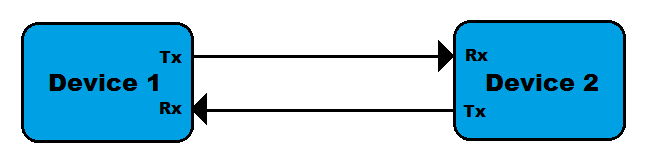
* I2C protocol supports multiple data speeds: standard (100 kbps), fast (400 kbps) and high speed (3.4 Mbps) communications.
* Built in collision detection
* 10-bit Addressing
* Supports both Multi-master and Multi-master with Slave functions.
* Data broadcast (general call).
* Since only two wires are required, I2C is well suited for boards with many devices connected on the bus. This helps reduce the cost and complexity of the circuit as additional devices are added to the system.

****

**Applications-** The I2C bus is a great option for applications that require low cost and simple implementation rather than high speed. For example, reading certain memory ICs, accessing DACs and ADCs,reading sensors,access LCDs, transmitting and controlling user directed actions, reading hardware sensors, and communicating with multiple microcontrollers are common uses of the I2C communication protocol.I2C also used for attaching low-speed peripherals to a motherboard, embedded system, cellphone, or other digital electronic devices. Several competitors, such as Siemens AG (later Infineon Technologies AG, now Intel mobile communications), NEC, Texas Instruments, STMicroelectronics (formerly SGS-Thomson), Motorola (later Freescale), and Intersil, have introduced compatible I²C products to the market since the mid-1990s

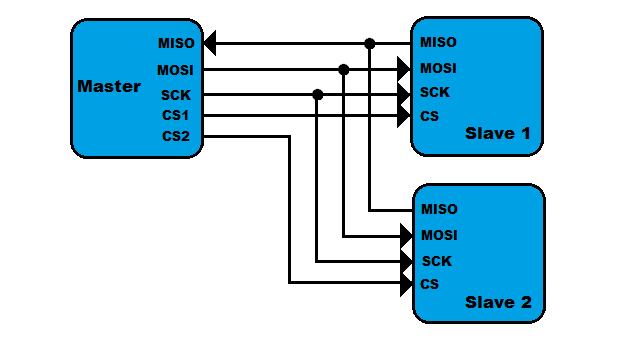
# Why I2C Protocol ?

Comparing I2C with UART



* In UART, separate clock line is not used with data transmission so both device should agree on same Baud-rate. If differences occur between Baud-rate on either end will cause garbled data while in I2C separate clock line is used.
* UART uses hardware overheads like start bit, stop bit, partiy bit or CRC bit in each frame of transmission which will lead to increase in transmission time while I2C use only acknowledgement bit(ACK) or no acknowledgement bit.
* Another drawback of UART is that they are suited to communications between two, and only two, devices while I2C can used with multiple devices and different modes like multi-master or single master, etc.
* Data transfer speed is also an issue for UART, they are limited to some extent and maximum baud-rate is around 230400 bps.

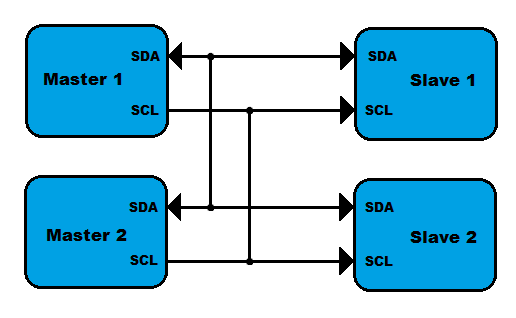
Comparing I2C with SPI



The first drawback of SPI is pin used to connect multiple devices. If we connect two device with SPI standard it will use four pins MISO, MOSI, SCK, CS.

* In SPI protocol, there is only single master but it can support arbitrary number of slaves. While in I2C, multiple master can be possible.
* If we compare data transmission speed of SPI and I2C, then SPI will win definitely. SPI support speed upto 10MHz (10 Mbps).

# What’s good in I2C ?

I2C requires only two line(two wire), but those two wires can support up to 1008 slave devices.

* 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 of I2C devices can communicate at the rate of 100kHz to 400kHz.

# How I2C Protocol Works ?

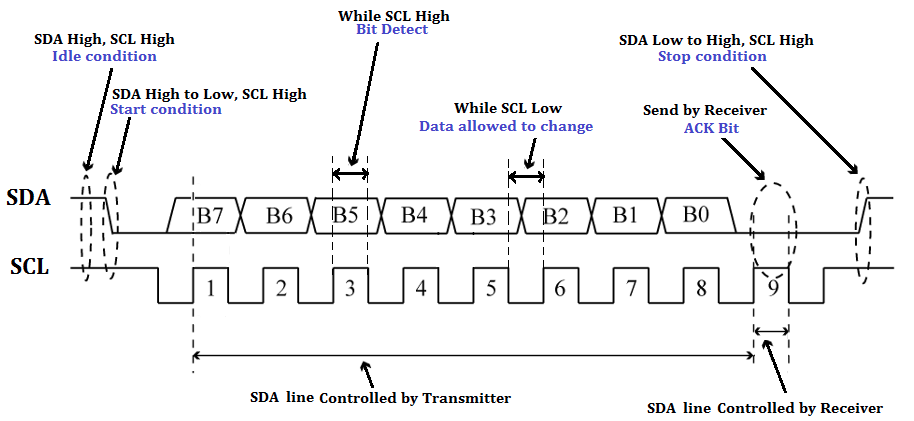
To understand the working of I2C Protocol, Let we take some practical examples

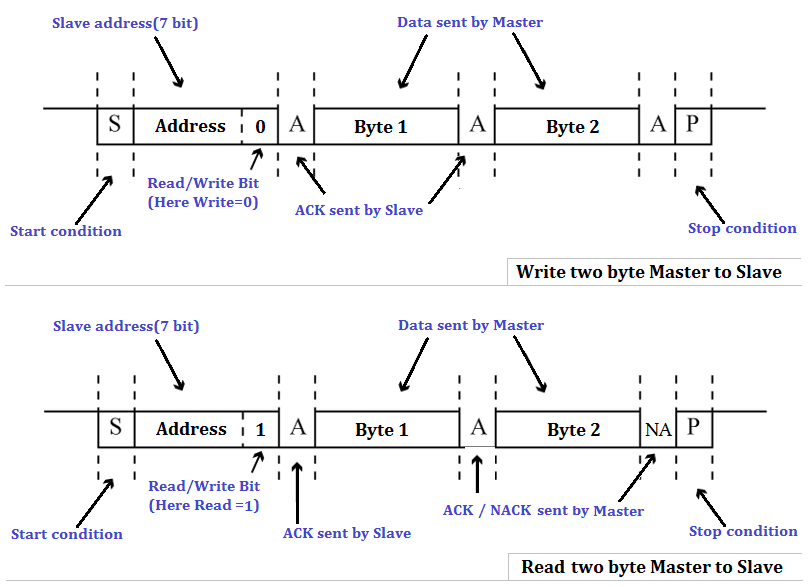
Single Byte Transfer

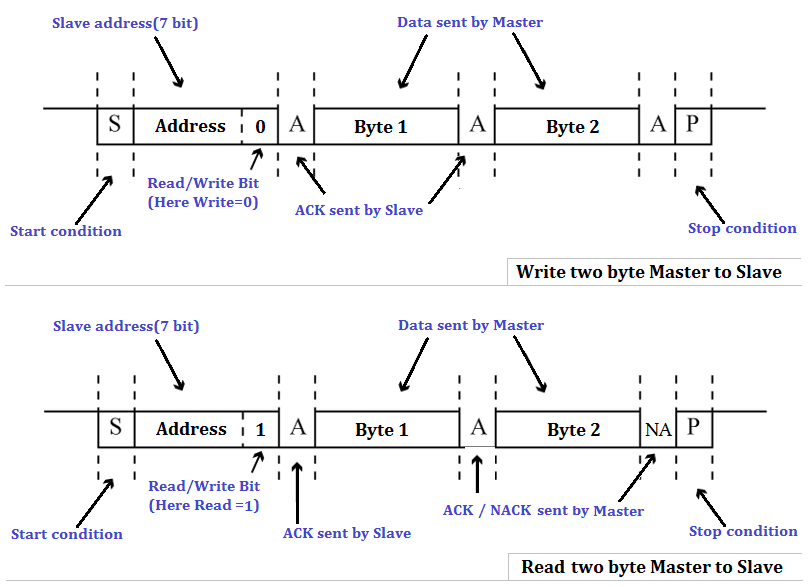
Idle condition: As you can see in above figure, at the initial condition SDA and SCL both High because of both line pulled High with pull-up resister.

* Start condition: Data transfer is start when SDA line pulled down mean High to Low when SCL line is High.
* Bit Detect: Slave can understand bit reception when it receives single bit ‘1’ or ‘0’ when SCL line is High.
* Data allowed to change: Transition of bit from 1 to 0 or 0 to 1 is only allowed when SCL line is Low as seen in pick.
* ACK Bit: Transmitter must release the SDA line after transmitting 8 bit to allow the Receiver to pull SDA line Low to acknowledgement the previous 8 bit data reception.
* Stop condition: is only understood when SDA line goes Low to High while SCL line is High.

Note: In I2C Protocol Byte is transmitted MSB to LSB. And in multi-Byte transmission, every 8 bits has a 9th bit that is an acknowledge

Write/Read Two Byte  from Master to Slave





Start Condition

Every communication in I2C Protocol is begin with start condition which we already studied above how start condition form. This alerts all slave devices that a transmission is about to start.

Address Frame Byte

The address frame byte include device address which is of 7-bit long and Read(1)/Write(0) bit.  The address frame byte is always sent first when any new communication sequence initiate.

ACK/NACK Bit

The 9th bit of the frame is the NACK/ACK bit(here NA or A). 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 some error occur.

              In multi-byte write process, Acknowledgement is send by slave at every single byte reception. But in mulit-byte read process, first byte(which is address of slave) Acknowledgement is send by slave and then Acknowledgement  is send by master at every single byte reception and if master don’t want more byte then it send No Acknowledgement to slave and generate stop condition.

Data Bytes

After the address frame byte has been sent, data bytes 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 byte 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

Every communication in I2C Protocol is end with stop condition which we already studied above. During normal data writing operation, the value on SDA should not change when SCL is high, to avoid false stop conditions.

# Speed of I2C Protocol

Common I²C bus speeds are the 100 kbit/s standard mode and the 10 kbit/s low-speed mode, but arbitrarily low clock frequencies are also allowed. Recent revisions of I²C can run at faster speeds typically 400 kbit/s Fast mode, 1 Mbit/s Fast mode plus or Fm+, and 3.4 Mbit/s High Speed mode.

# More About I2C Protocol

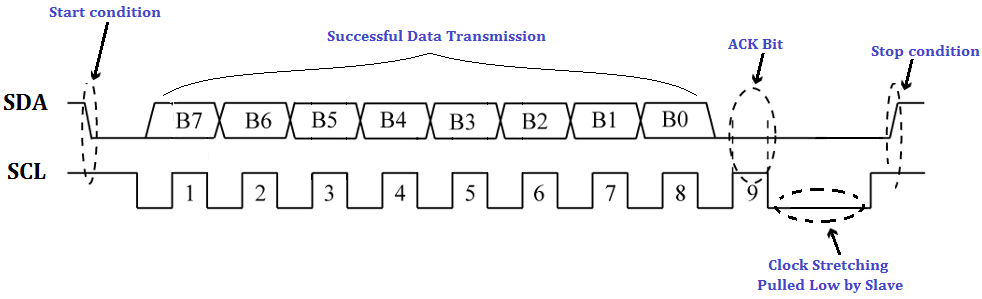
10-bit Addressing supports more number of devices

In order to increase device address capability, 10-bit addressing scheme is introduced to I2C Protocol by which we can connect more number of devices by using I2C Protocol.



In a 10-bit addressing system, two frames are required to transmit the slave address. The first frame will consist of the code 11110xyz, where ‘x’ is the MSB of the slave address, y is  8th bit of the slave address, and z is the Read/Write bit as seen 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 bit 0 to bit 7 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.

Clock stretching



* In an I2C communication the master device determines the clock speed. However, there are situations where an I2C slave is not able to co-operate with the clock speed given by the master and needs to slow down a little. This is done by a mechanism referred to as clock stretching.
* An I2C slave is stretching the clock by pull down the SCL if it needs to reduce the bus speed. The master on the other hand is required to read back the clock signal after releasing it to high state and wait until the line has actually gone high.

#### Repeated Start Conditions

* When we communicate through I2C bus, its obvious in most cases that first you need to send command and then you get response from slave, so each time you dont need to terminate communication by stop condition. You can use start condition reapetedly without terminating previous transaction. The I2C protocol defines a so-called repeated start condition.
* For example, After having sent the address byte (address and read/write bit) the master may send any number of bytes followed by a stop condition. Instead of sending the stop condition it is also allowed to send another start condition again followed by an address (and of course including a read/write bit) and more data. This is defined recursively allowing any number of start conditions to be sent.
* The purpose of this is to allow combined write/read operations to one or more devices without releasing the bus and thus with the guarantee that the operation is not interrupted.