GLOBAL EDGE
Intelligence Of Things

Inter-Integrated Circuit(I2C)

Presented By-
Bhargavi G

Copyright © 2017, Global Edge Software Ltd.

Agenda

- I2C protocol
- RTC
- I2C driver Implementation

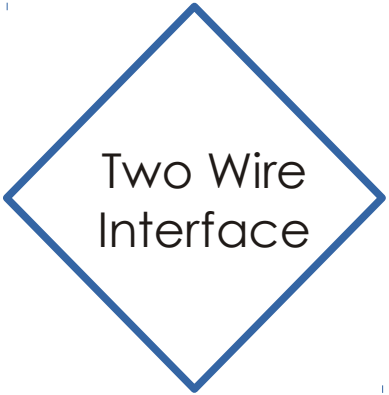
Introduction

It was developed by Philips Semiconductor (now NXP Semiconductors) in 1982.

I2C is a synchronous serial communication protocol.

I2c devices include EEPROMs,Real time clock ,Thermal sensors and other similar peripherals in embedded systems.

I2C Signals



SDA(Serial Data Line)

All the data transfer among the devices takes place through this line.

SCL(Serial Clock Line)

It is used to synchronize all the devices and the data transfer together.

Master and Slave

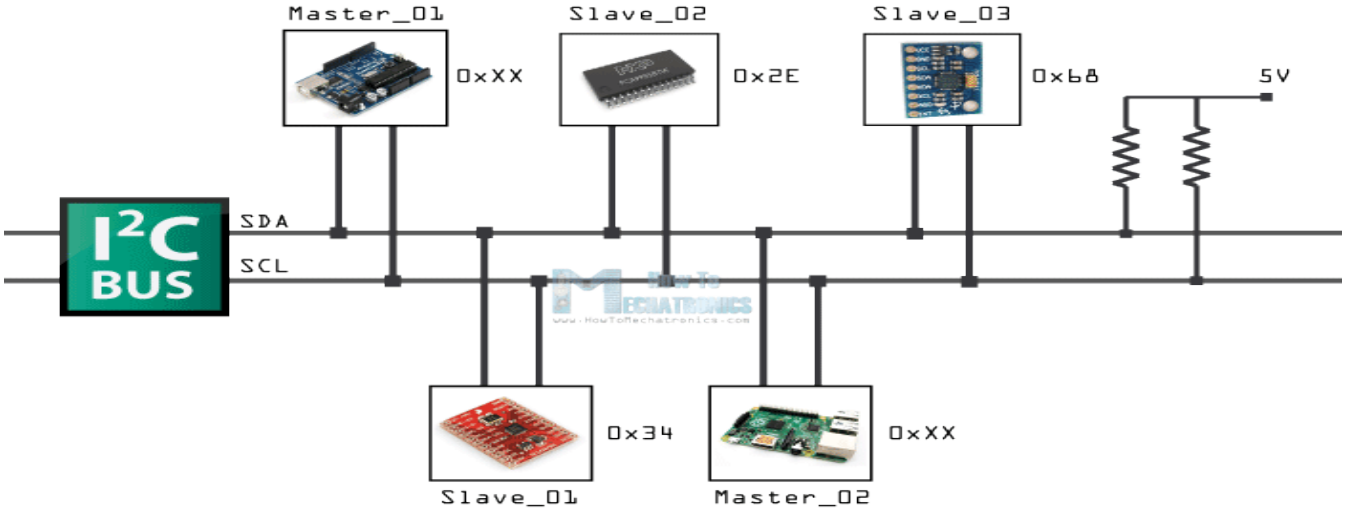
Master

- Master is the device which initiates the transfer and communication with slaves and drives the clock line SCL.

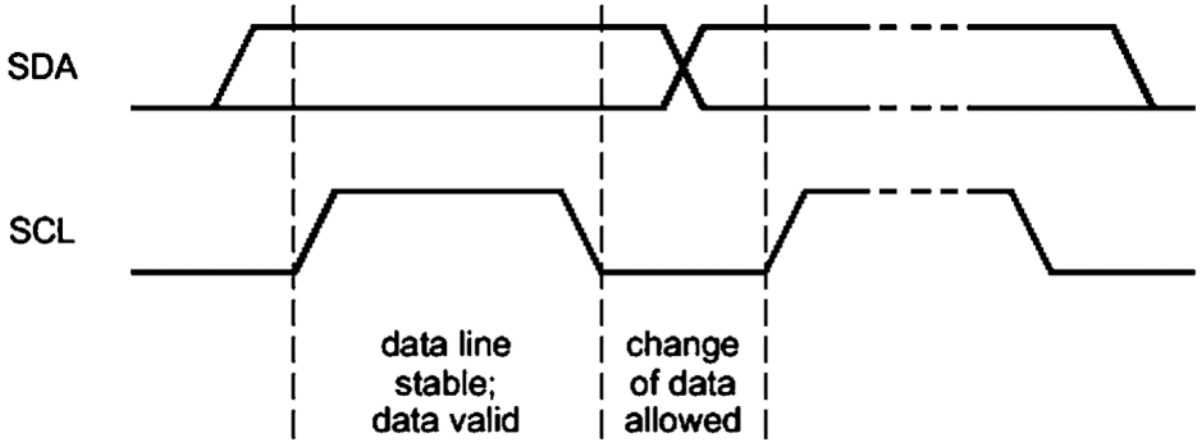
Slave

- Slave is the device that receives the clock and responds when addressed by the master.

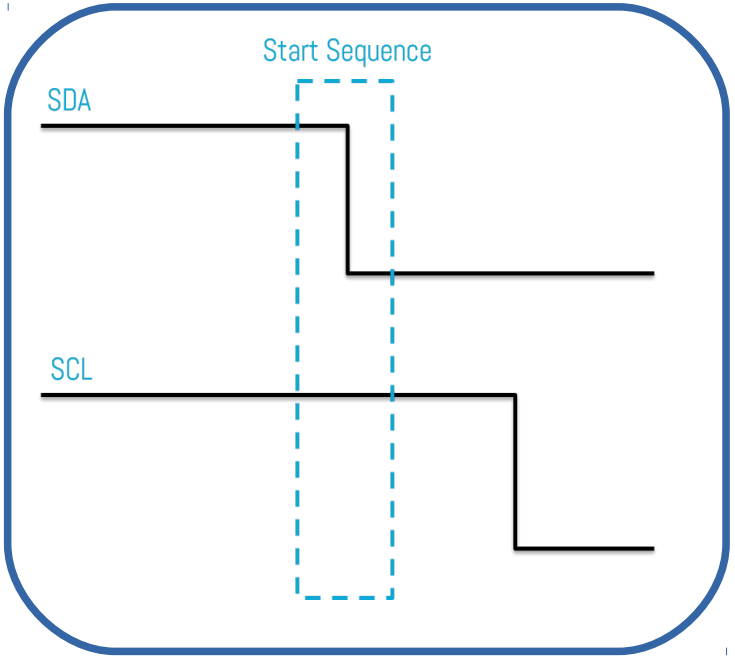
I2C Bus Interface

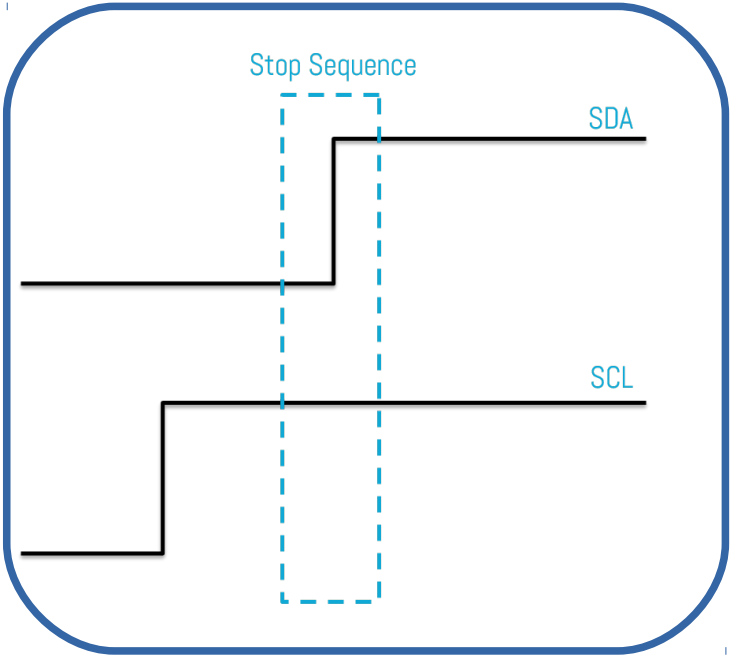


I2C Data Validity

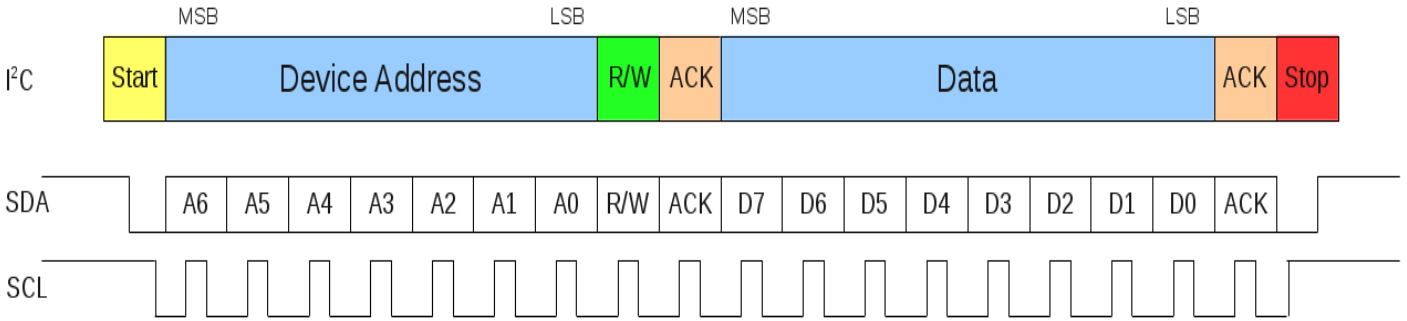


Start and Stop Conditions



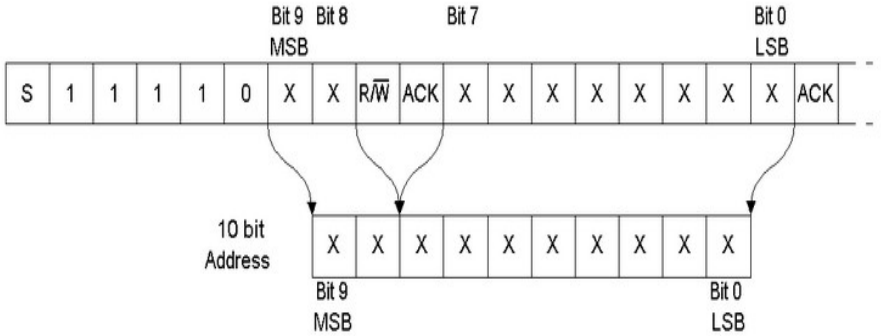


Data Format


MSB
LSB
MSB
LSB
I²C
Start
Device Address
R/W
ACK
Data
ACK
Stop
SDA
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
SCL

10 Bit Addressing

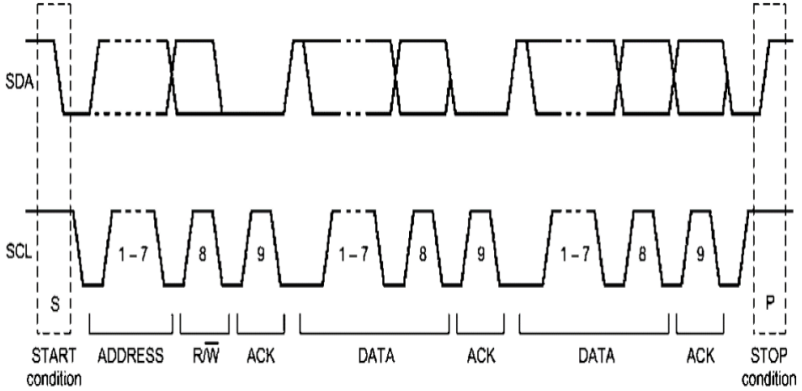
- In order to prevent address clashes, due to the limited range of the 7 bit addresses, a new 10 bit address scheme has been introduced.



- After the start condition, a leading '11110' introduces the 10 bit addressing scheme.
- The last two address bits of the first byte concatenated with the eight bits of the second byte of the whole 10 bit address.
- Devices which only use 7 bit addressing simply ignore messages with the leading '11110'.

Timing Diagram

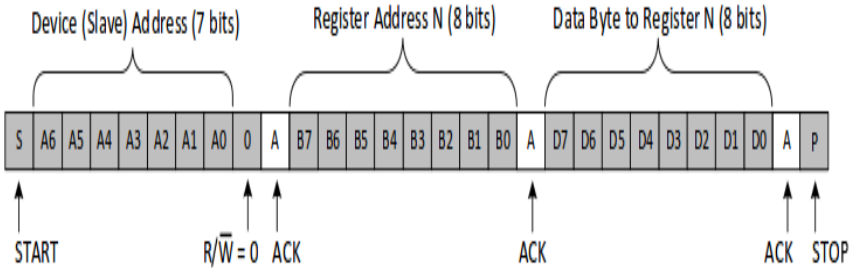
- START condition-SDA high to low.
- The first byte transmitted by the master is the 7 bit slave address.
- Next bit indicates the direction of data transfer(i.e, read or write).
- The slave returns an acknowledge bit after each received byte.



- Data is transferred with the most significant bit (MSB) first.
- This is followed by transmitting a number of data bytes.
- STOP condition-SDA low to high.

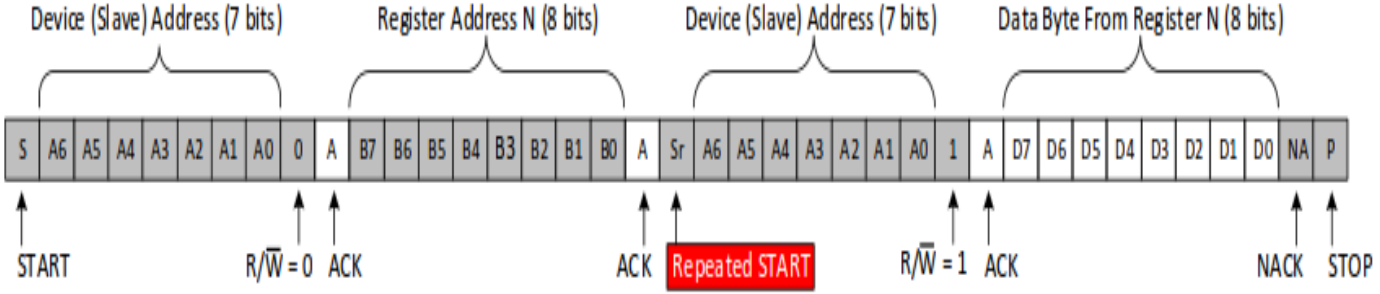
Writing to a Slave On The I2C Bus

- To write on the I2C bus, the master will send a start condition on the bus with the slave's address, as well as the last bit (the R/W bit) set to 0, which signifies a write.
- After the slave sends the acknowledge bit, the master will then send the register address of the register it wishes to write to.



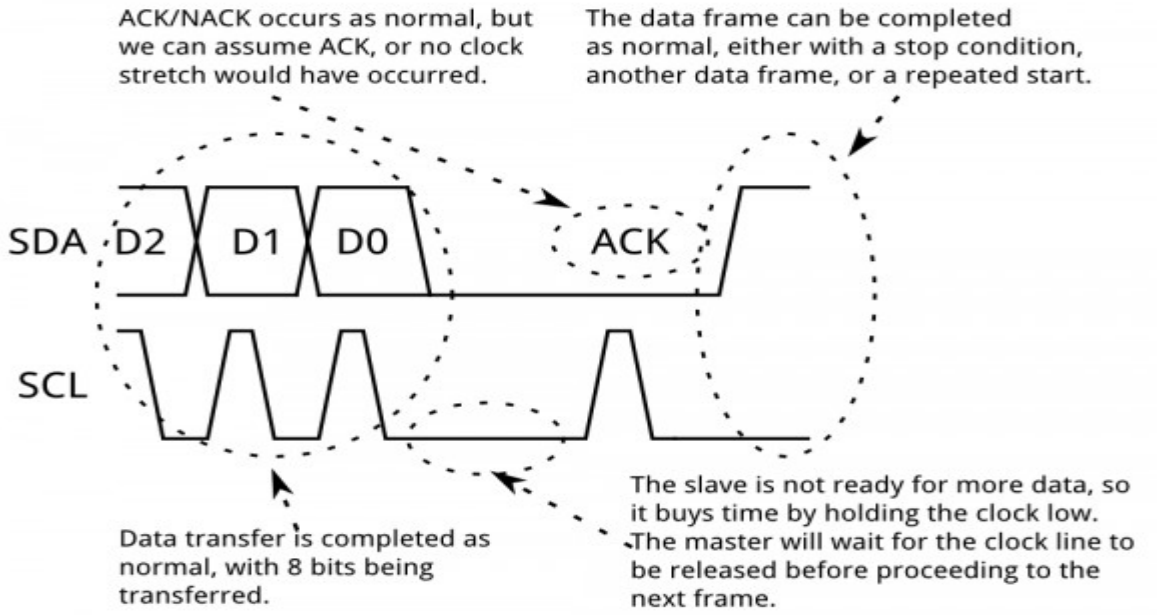
- The slave will acknowledge again, letting the master know it is ready.
- After this, the master will start sending the register data to the slave, until the master has sent all the data it needs to.
- The master will terminate the transmission with a STOP condition.

Reading from a Slave On The I2C Bus

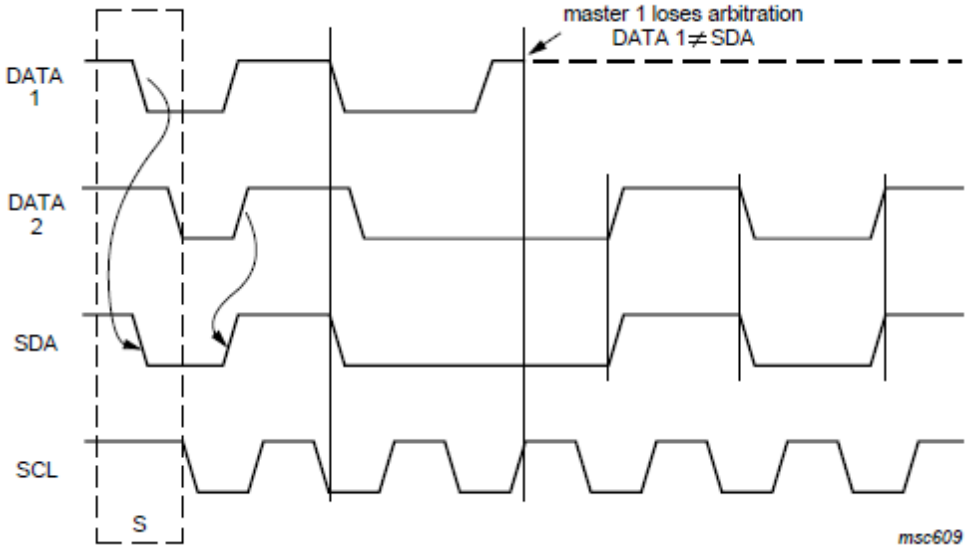


- In order to read from a slave,the master must first instruct the slave which register it wishes to read from.This is done by the master startingoff the transmission in a similarfashionas the write,by sending the address with the R/W bit equal to 0 (signifying a write),followed by the register address it wishes to read from.
- Once the master has received the numberof bytes it is expecting,it will send a NACK,signaling to the slave to halt communications and release the bus. The master will follow this up with a STOP condition.

Clock Stretching



Arbitration



Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted.

Advantages

Why I2C

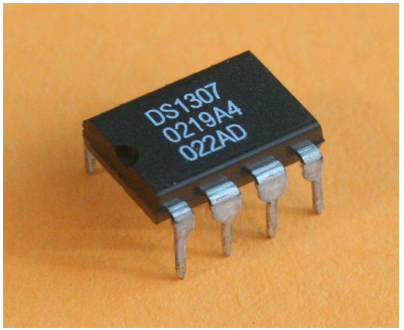
- Only uses two wires
- Supports multiple masters and multiple slaves
- ACK/NACK bit gives confirmation that each frame is transferred successfully

Disadvantages

- Slower data transfer rate than SPI
- The size of the data frame is limited to 8 bits

Real Time Clock

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year.
- 56-byte, battery-backed, nonvolatile (NV)RAM for data storage.
- Two-wire serial interface.
- Available in 8-pin DIP or SOIC.



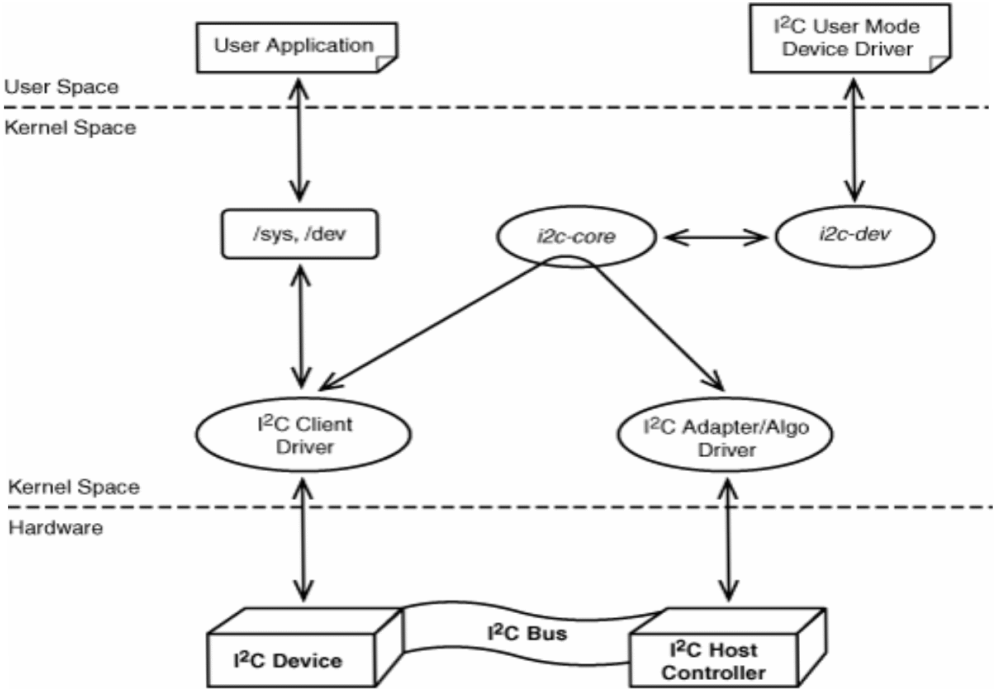
RTC(DS1307)

- The DS1307 uses an external 32.768kHz crystal.
- The RTC registers are located in address locations 00h to 07h.
- The RAM registers are located in address locations 08h to 3Fh.
- The contents of the time and calendar registers are in the BCD format.

Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

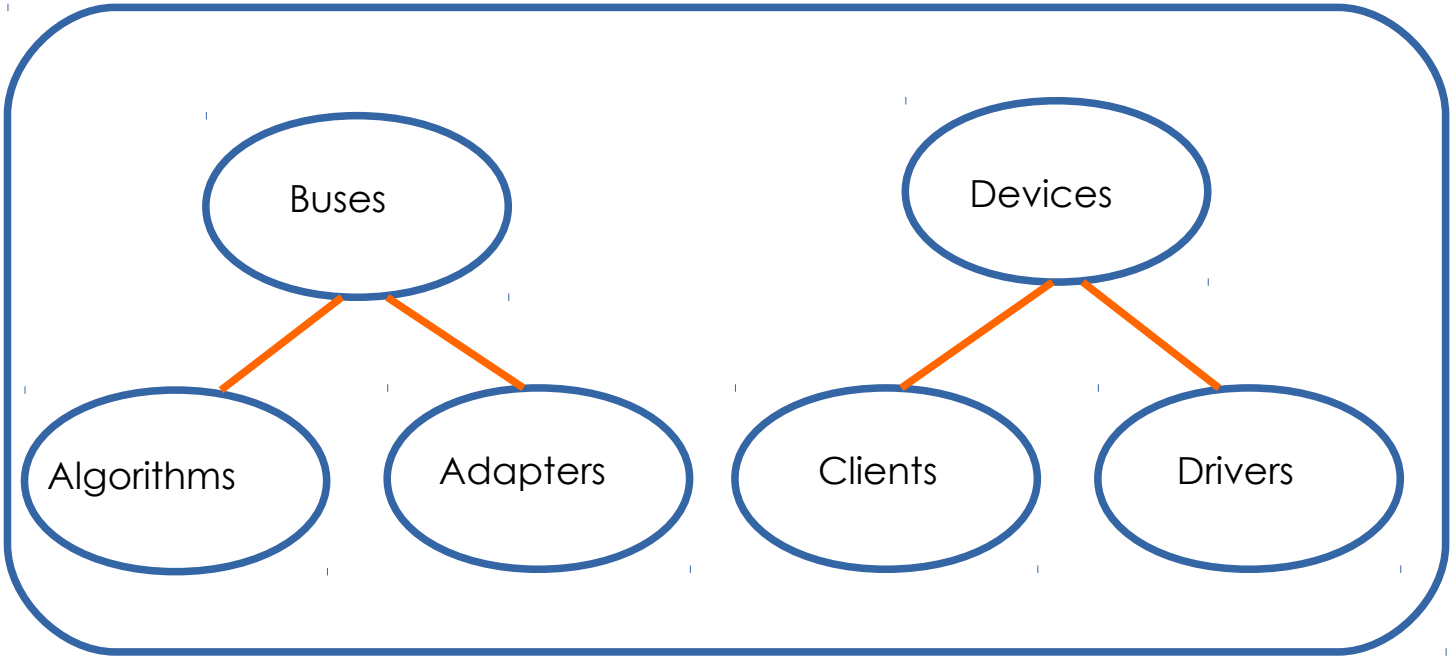
I2C Architecture



Device Tree

- A device tree is a tree data structure with nodes that describe the physical devices in a system.
- On ARM all device tree source are located at /arch/arm/boot/dts/.
- Device tree compiler (dtc) is used to compile dts files and produce .dtb.
- The Device Tree Blob(.dtb) is the binary that gets loaded by the bootloader and parsed by the kernel at boot time.

I2C driver Implementation



Programming with I2C - Structures

struct i2c_adapter

```
struct i2c_adapter {
      char name[32];
      unsigned int id;
      struct i2c_algorithm *algo;
      unsigned int flags;

};
```

- An Adapter effectively represents a bus – it is used to tie up a particular I2C with an algorithm and bus number.
- i2c_adapter is the structure used to identify a physical i2c bus along with the access algorithms necessary to access it.

Programming with I2C - Structures

struct i2c_algorithm

```
struct i2c_algorithm {
  int (* master_xfer) (struct i2c_adapter
*adap, struct i2c_msg *msgs, int num);
  int (* smbus_xfer) (struct i2c_adapter
*adap, u16 addr,unsigned short flags, char
read_write, u8 command, int size, union
i2c_smbus_data *data);
  u32 (* functionality) (struct i2c_adapter
*);
};
```

- An Algorithm performs the actual reading and writing of I2C messages to the hardware – this may involve bit banging GPIO lines or writing to an I2C controller chip.

Programming with I2C - Structures

struct i2c_client

```
struct i2c_client {
  unsigned short flags;
  unsigned short addr;
  char name;
  struct i2c_adapter * adapter;
  struct device dev;
};
```

- A client represents an I2C slave device
- Client is represented by struct i2c_client that identifies a single device (i.e. chip) connected to an i2c bus.
- This includes various members such as chip address, name and pointers to the adapter.

Programming with I2C - Structures

Struct i2c_driver

```
struct i2c_driver {

  int (* probe) (struct i2c_client *, const
  struct i2c_device_id *);

  int (* remove) (struct i2c_client *);

  const struct i2c_device_id * id_table;


};
```

- Represent an I2C device driver.

I2C driver Implementation

i2c_add_driver

We tell the kernel about our I2C slave driver by registering the 'struct i2c_driver' structure with the I2C core.

The id_table member allows us to tell the framework which I2C slaves chips are supported.

Programming with I2C - Structures

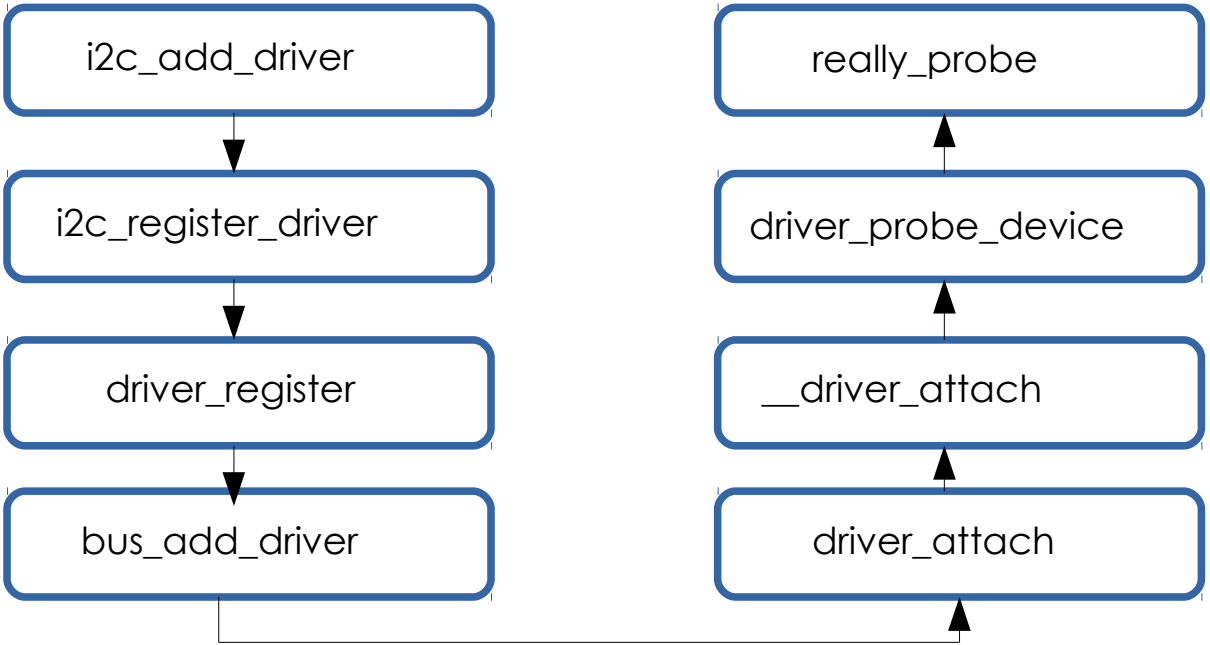
probe

probe() in general verify that the specified device hardware actually exists.

int probe (struct i2c_client *client, const struct i2c_device_id *idp)

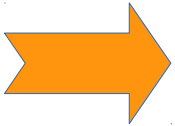
Passed along as a parameter to the probe function is a ‘struct i2c_client’ structure – this represents our I2C slave chip.

Code flow of i2c_add_driver



I2C driver Implementation

How can we can communicate with the slave device????



With the help of i2c_transfer..

Programming with I2C - Structures

i2c_transfer

```
int i2c_transfer(struct i2c_adapter * adap, struct i2c_msg * msgs,
int num)

Execute a single or combined I2C message.

Parameters:
struct i2c_adapter * adap - Handle to I2C bus.
struct i2c_msg * msgs-One or more messages to execute.
int num-Number of messages to be executed.

Returns negative errno, else the number of messages executed.
```

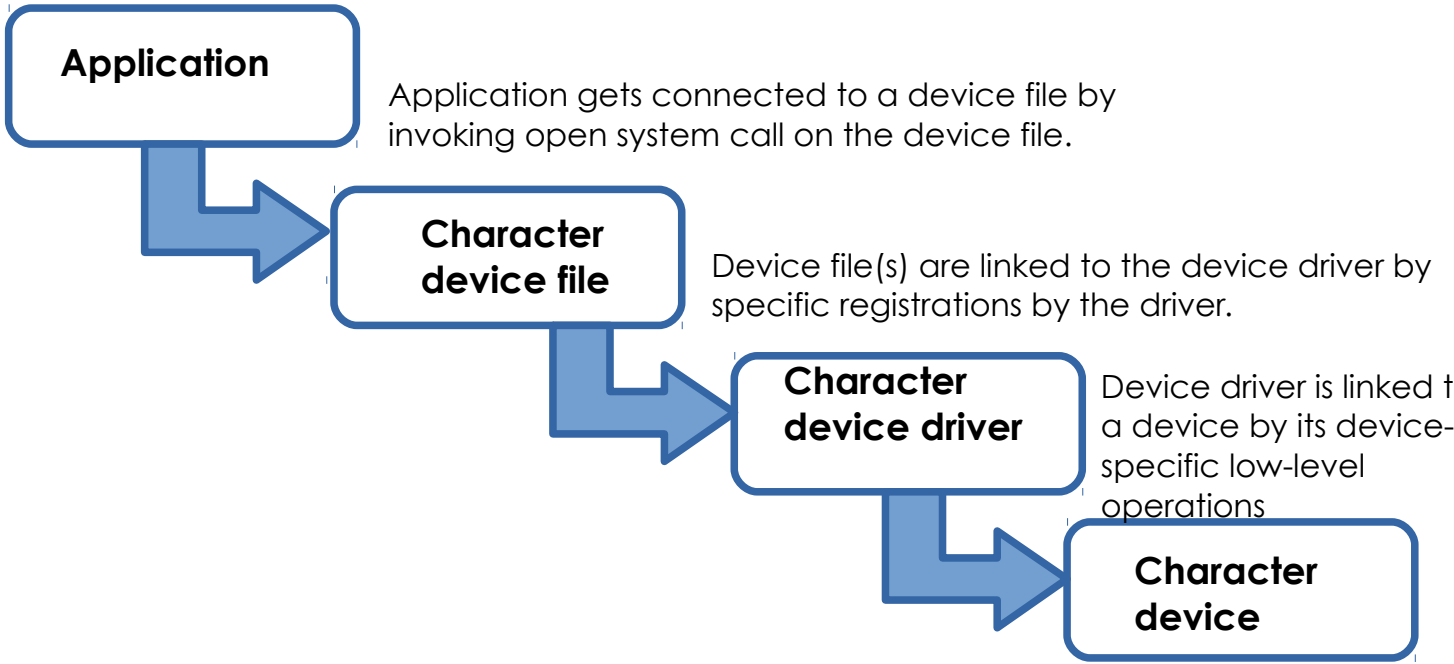
Programming with I2C - Structures

struct i2c_msg

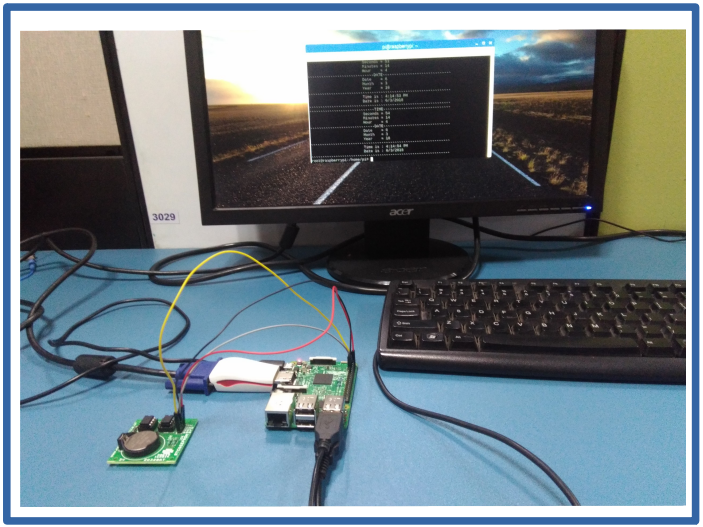
```
struct i2c_msg {
        __u16 addr;
        __u16 flags;
        __u16 len;
        __u8 *buf;
};
```

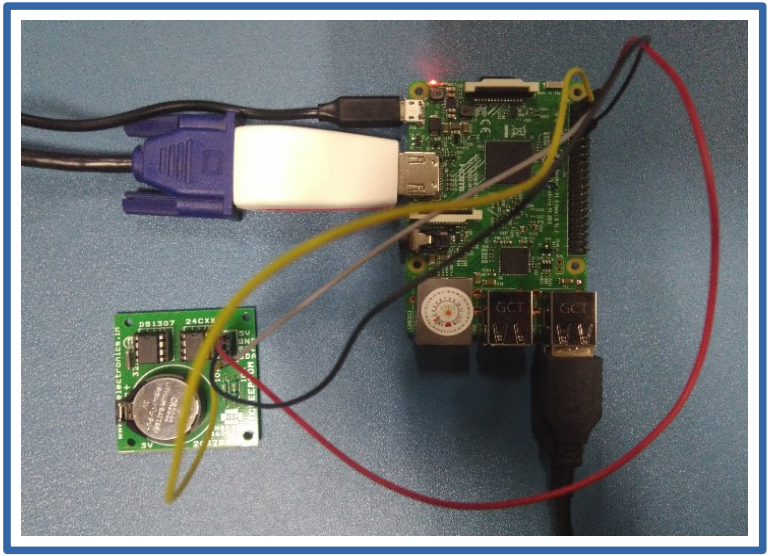
- An I2C transaction segment beginning with START.
- An i2c_msg is the low level representation of one segment of an I2C transaction.

From Application to the device



Interfacing RTC(DS1307) with Raspberry pi (Setup)



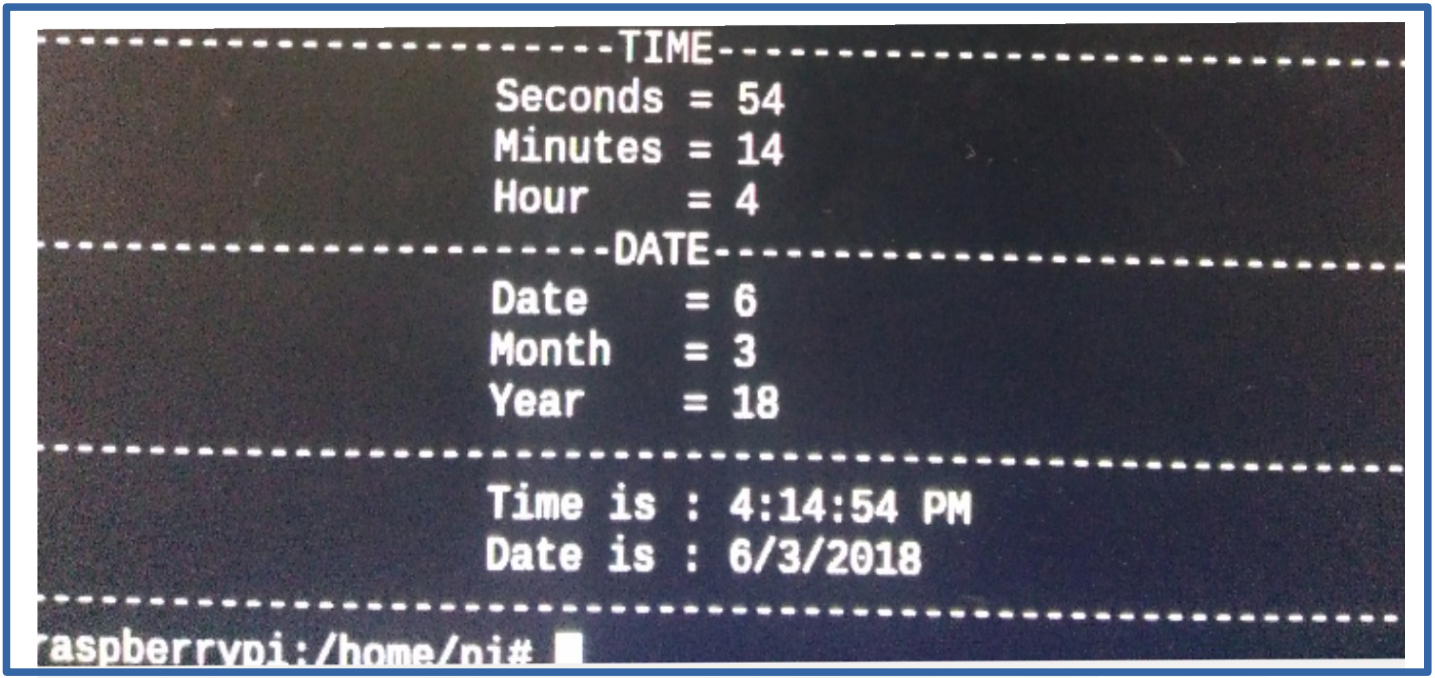


Interfacing RTC(DS1307) with Raspberry pi

To insert the module → insmod i2c_client.ko

To compile and run the application →
- gcc i2c_app.c
- ./a.out

Interfacing RTC(DS1307) with Raspberry pi(Result)



References

- http://wiki.dreamrunner.org/public_html/Embedded-System/Linux-Device-Tree.html
- https://learn.sparkfun.com/tutorials/i2c
- https://www.nxp.com/docs/en/user-guide/UM10204.pdf
- http://www.ti.com/lit/an/slva704/slva704.pdf
- https://datasheets.maximintegrated.com/en/ds/DS1307.pdf

Large enough to Deliver, Small enough to Care

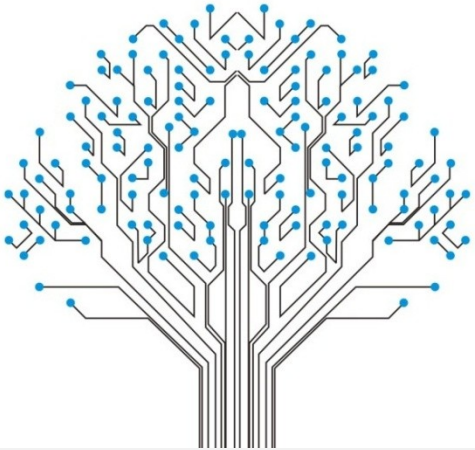
Global Village
IT SEZ
Bangalore

South Main Street
Milpitas
California

Raheja Mindspace
IT Park
Hyderabad

www.globaledgesoft.com

Thank you



Fairness
Learning
Responsibility
Innovation
Respect