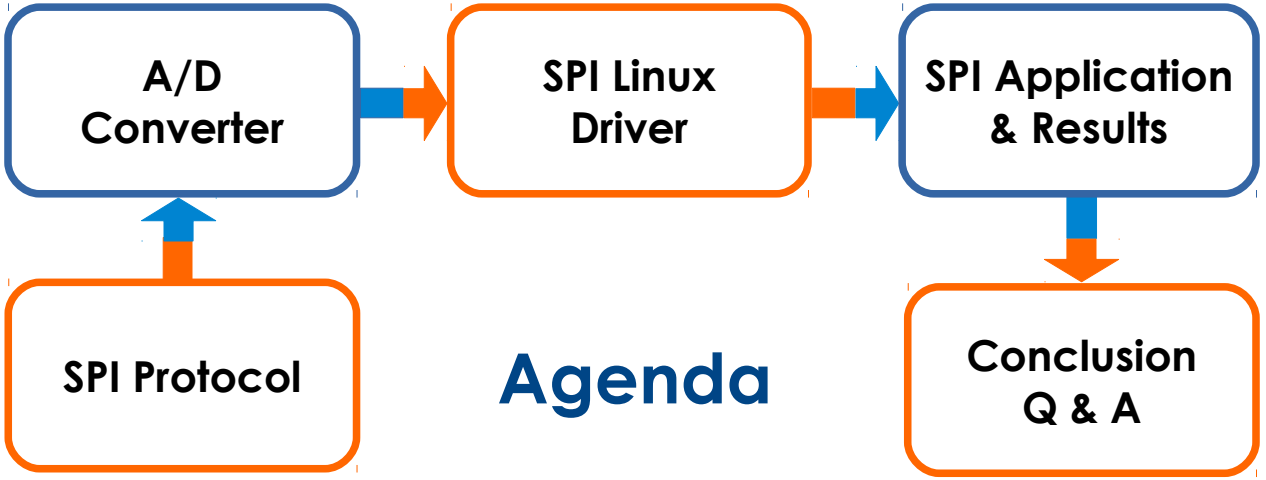
GLOBAL EDGE
Intelligence Of Things

Linux SPI Driver

Interfacing ADC with Raspberry Pi

Presented By-
Aditya Hambar

Agenda


A/D Converter
SPI Linux Driver
SPI Application & Results
SPI Protocol
Agenda
Conclusion Q & A

SPI Protocol

A/D Converter

Linux SPI Driver

SPI...???

Serial Peripheral Interface

- Serial Communication Protocol
- Developed by Motorola in 1979
- 4 – wire Serial Bus Protocol
- Simple & Efficient Interface
- Widely used with Embedded Systems

Why SPI...???

Asynchronous Communication

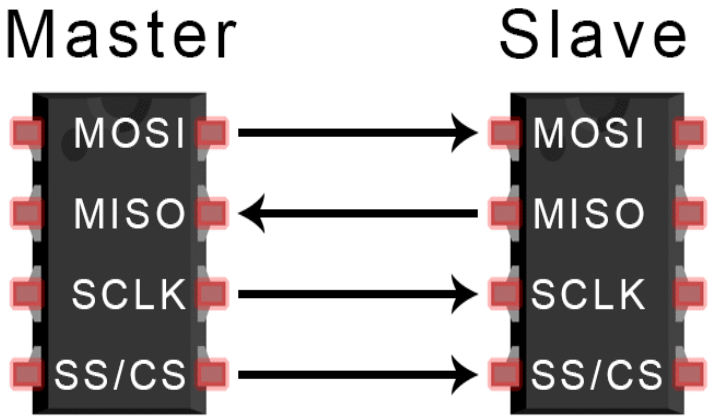
- No control over data.
- No any guarrantee that both sides are running precisely at the same rate.
- Can arise a problem due to different speeds.
- Start and Stop bits are required.
- Lot of overhead due to extra bits.

Vs

Synchronous Communication

- Separate lines for Data & Clock.
- Keeps both sides in perfect sync.
- Clock helps in sampling the data at exact time.
- Receiving hardware is much simpler.
- Useful in small distance communication.

SPI Protocol



MOSI – Master Out Slave In

MISO – Master In Slave Out

SCLK – Serial Clock

SS/CS – Slave Select/Chip Select

SPI Protocol

SCLK

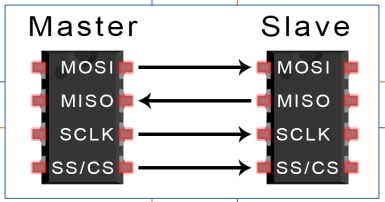
Master generates serial clock and sends to slave with SCLK pin.

Direction : Master -> Slave

MOSI

Data out pin for sending data bits to Slave.

Direction : Master -> Slave



MISO

Data in pin for receiving data bits from Slave.

Direction : Slave -> Master

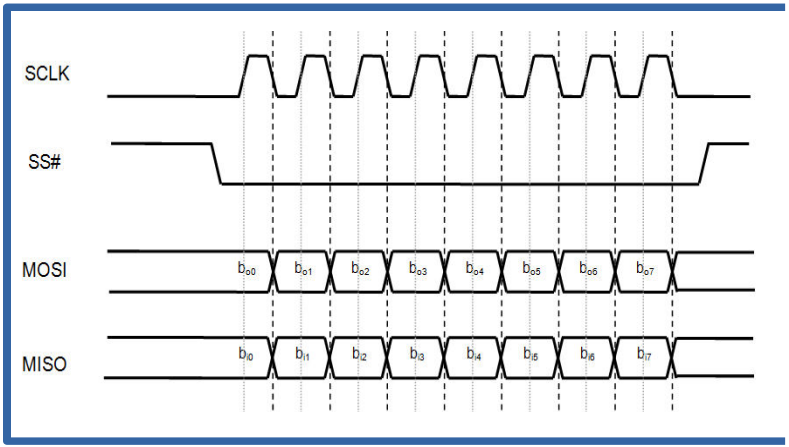
SS/CS

Slave select pin for selecting particular slave device.

Direction : Master -> Slave

Simple SPI Communication

- Master generates the serial clock on SCLK pin.
- Master pulls SS/CS pin low in order to select a particular slave device.
- Data bits are sent through MOSI pin and received through MISO pin.



- Data bits are sampled at one edge of SCLk & toggeled at other edge (depending upon SPI mode).

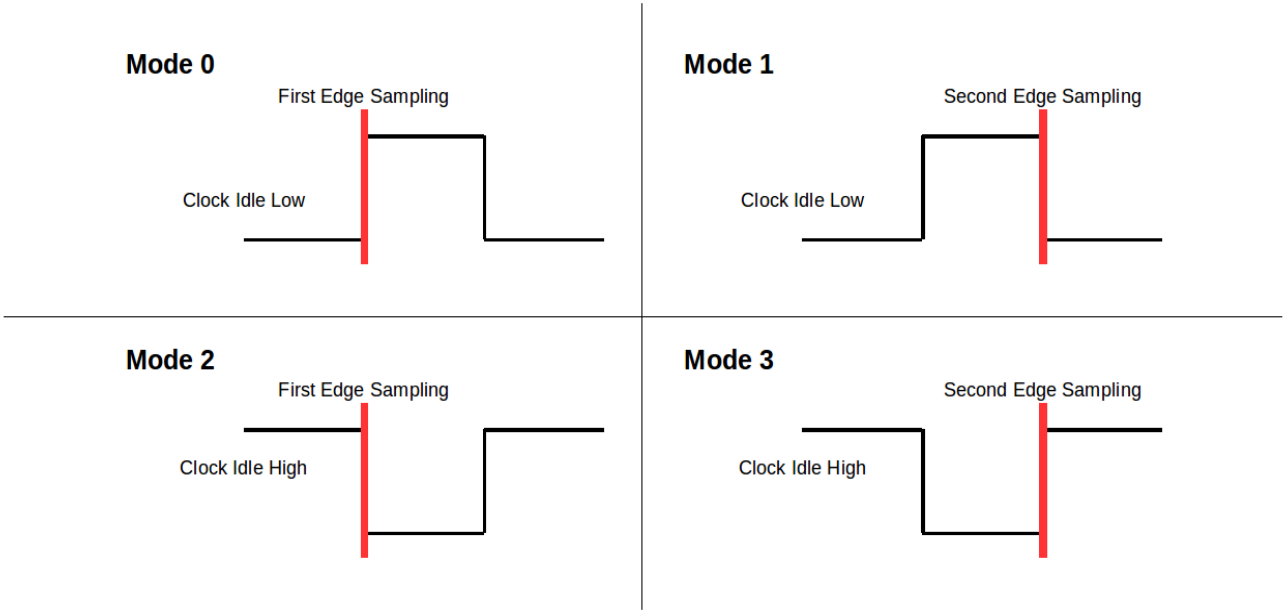
SPI Modes

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

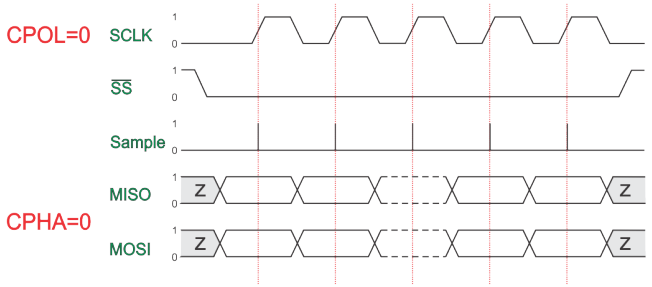
CPOL – Clock Polarity

CPHA – Clock Phase

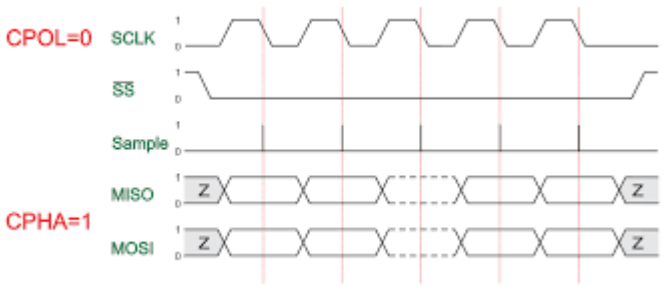
SPI Modes


Mode 0
First Edge Sampling
Clock Idle Low
Mode 1
Second Edge Sampling
Clock Idle Low
Mode 2
First Edge Sampling
Clock Idle High
Mode 3
Second Edge Sampling
Clock Idle High

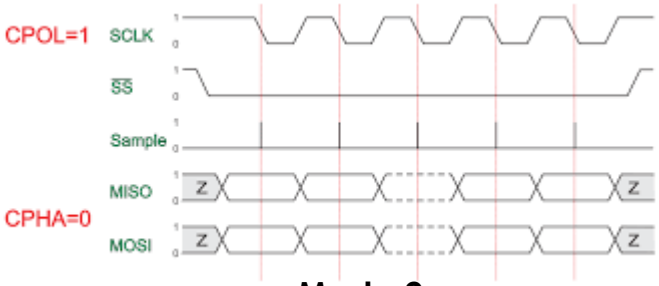
SPI Modes



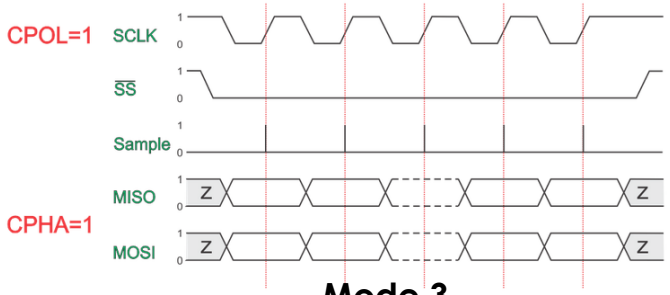
Mode 0



Mode 1

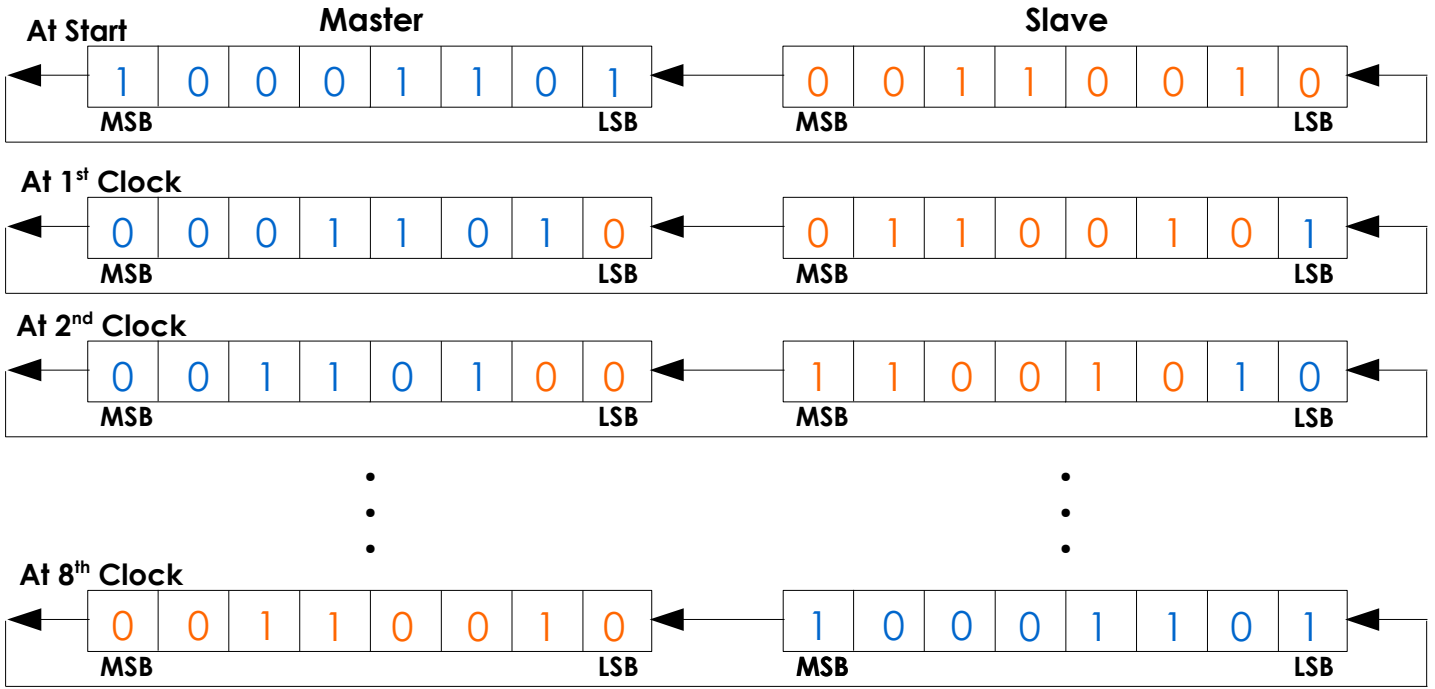


Mode 2

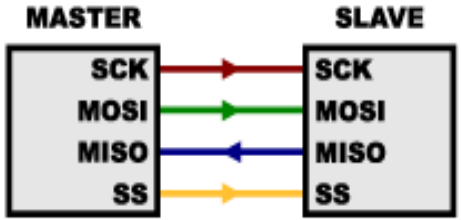


Mode 3

Shift Registers

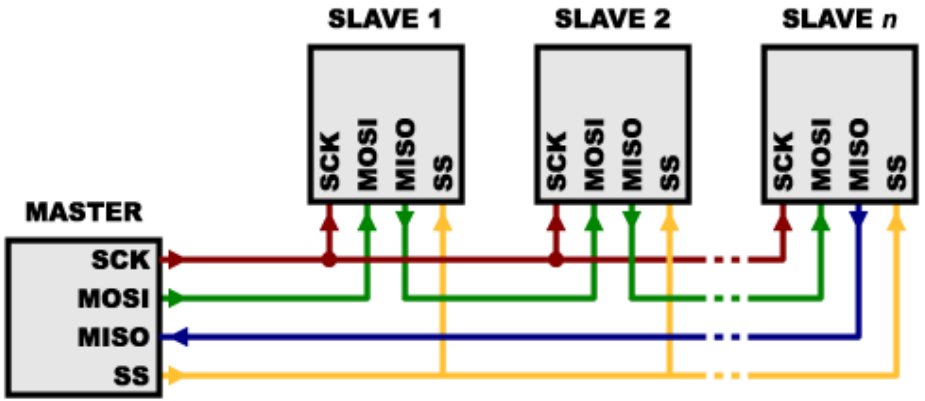


SPI Interfacing



Single Slave Interface

Multi Slave Interface



Positives & Negatives

POSITIVES

- Simple hardware interfacing.
- Faster than asynchronous serial communication.
- Flexibility for the bits transfer.
- Supports multiple slaves.
- No need for individual addresses.
- Full duplex communication.
- Does not require processing overheads.
- High speed of data transfer.

NEGATIVES

- Requires more pins on IC.
- Supports only one master device.
- Separate SS lines to each slave.
- Master must control all communications.
- No acknowledgment mechanism.
- No flow control.

SPI Protocol

A/D Converter

Linux SPI Driver

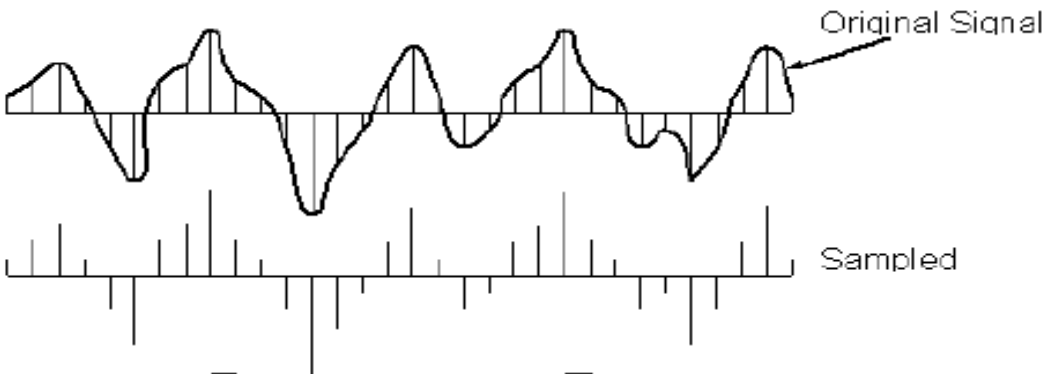
A/D Converter...???

Analog to Digital Converter

Electronic Integrated Circuit which converts Analog Signal into Digital Signals.

Continuous Time & Continuous Amplitude Into Discrete Time & Discrete Amplitude

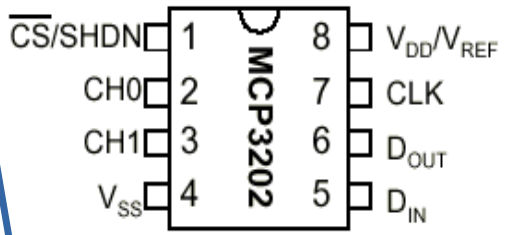
Analog to Digital Conversion



- Analog input signal is sampled at discrete time.
- Sampled discrete amplitude levels are then converted into digital format i.e. sequence of 1's and 0's.

ADC - “MCP 3202”

- 8 – Pin Package.
- High performance and low power consumption
- 12 bit ADC Capability.
- Works on Successive Approximation principle.
- An industry-standard SPI serial interface.



Working of ADC - “MCP 3202”

Initialization of ADC -

- Communication is done by pulling CS pin low.
- DIN will constitute a start bit.
- SGL/DIFF bit and ODD/SIGN bit follow the start bit.
- Then MSBF bit is transmitted.

SGL/DIFF – For selecting Single-Ended mode or Pseudo-differntial mode.

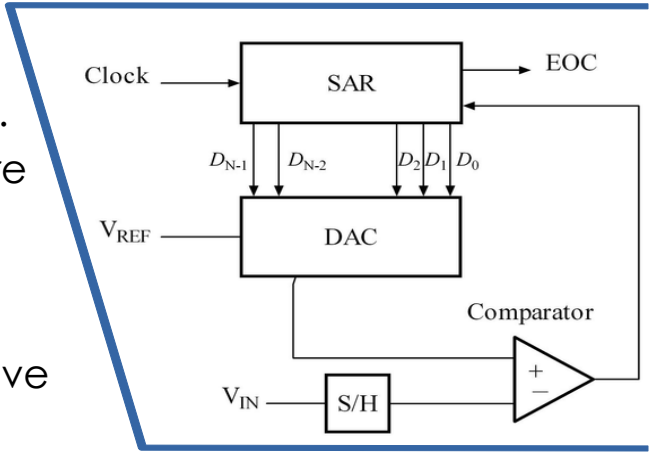
ODD/SIGN – For selecting one of the Channels in Single-Ended mode & Polarity in Pseudo-differntial mode.

MSBF – For selecting MSB first or LSB first data format.

Working of ADC - “MCP 3202”

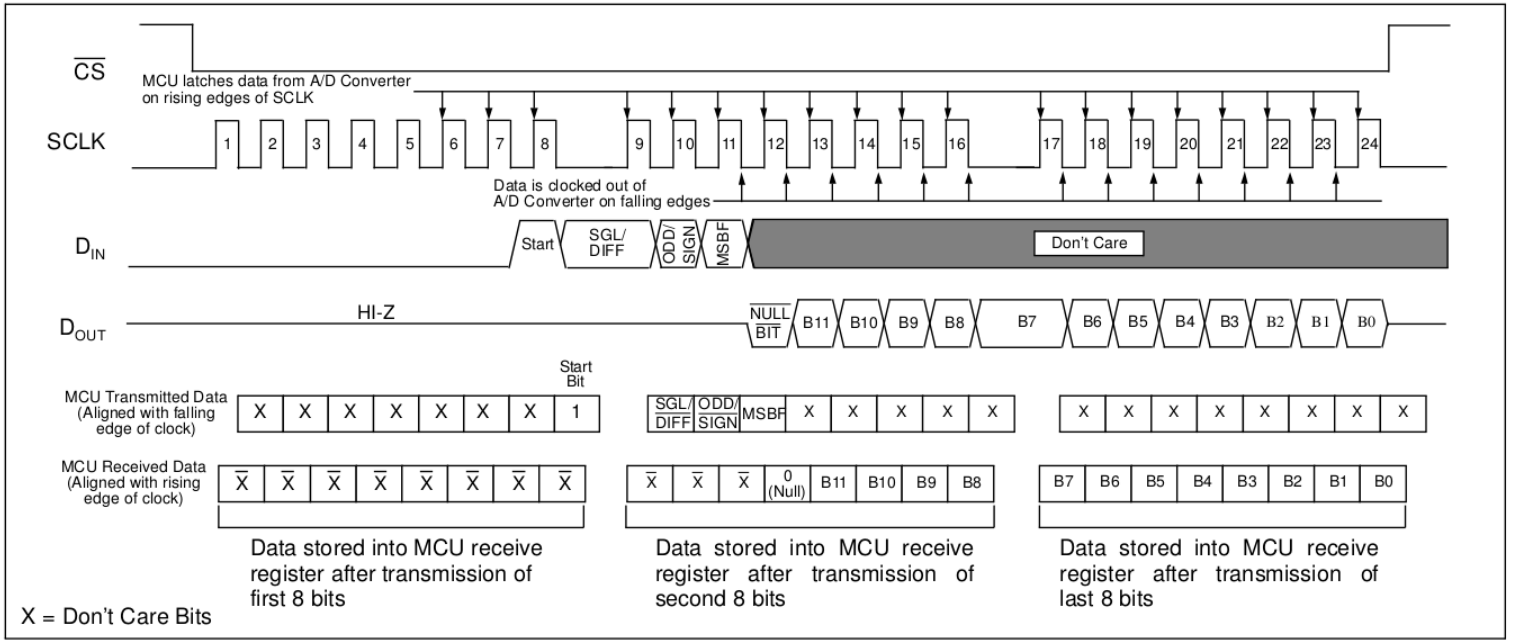
Conversion of A to D -

- Successive Approximation Principle.
- A sample and hold circuit to acquire the input voltage (Vin).
- Comparator compares Vin to the output of the internal DAC & gives result of the comparison to successive approximation register (SAR).



- SAR supplies an approximate digital code of Vin to the internal DAC.
- DAC supplies an analog voltage equal to the digital code output of the SAR to Comparator.

Timing Diagram of “MCP 3202”



SPI Protocol

A/D Converter

Linux SPI Driver

Linux SPI Driver API...???

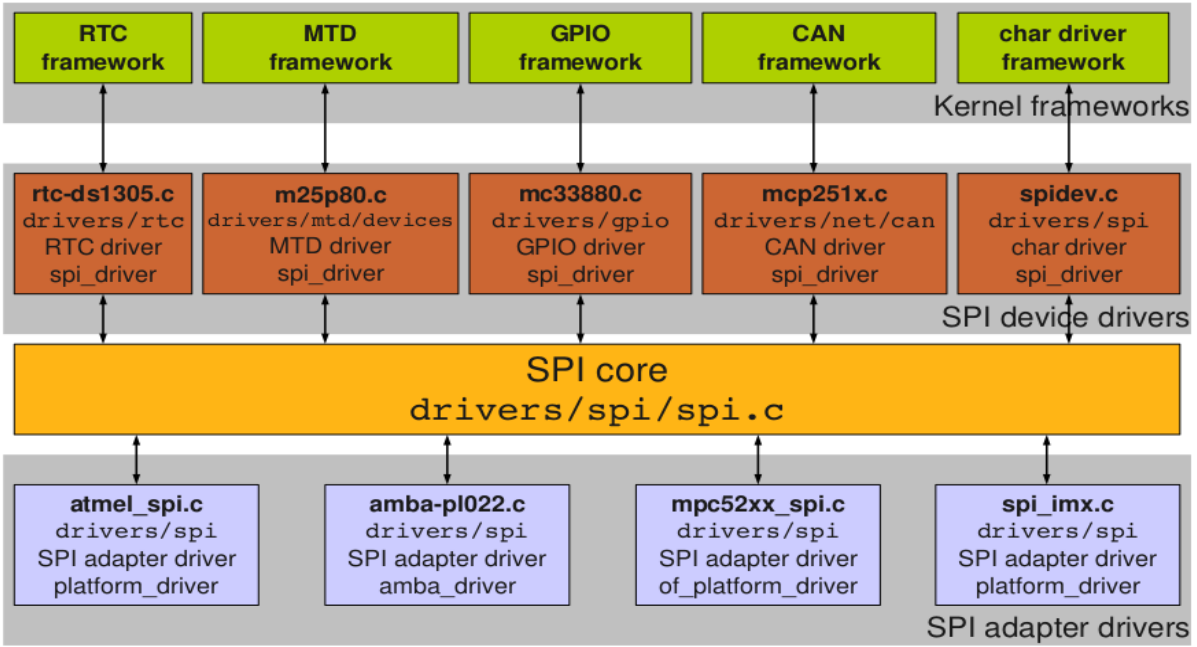
Linux SPI Driver API

Linux talks to SPI peripherals...

A generalized interface to declare SPI buses and devices.

Manage them according to the standard Linux driver model, and perform input/output operations...!!!

Linux SPI Components



Programming with SPI

SPI Driver

Controller Driver

- Abstracts controller hardware (GPIO/FIFOs/SPI shift register).
- Represented as a *struct spi_master*
- SPI devices are children of master, represented as a *struct spi_device*

Protocol Driver

- Represented as *struct spi_driver*
- Bound to a *spi_device* using normal driver model calls.

Programming with SPI - Structures

struct spi_master

```
{
        struct device dev;
        u16 num_chipselect;
        size_t (*)(struct spi_device *spi)
                max_transfer_size;
        int (*)(struct spi_device *spi,
                struct spi_message *mesg)transfer;
        void (*)(struct spi_device *spi) cleanup;
        ...
};
```

- SPI Controller Driver
- Allows easy communication with SPI slave devices.
- Takes care of multiplexing between different SPI slaves on the same master.

Programming with SPI - Structures

struct spi_device

```
{
        struct device dev;
        struct spi_controller *master;
        u8 chip_select;
        void *controller_state;
        void *controller_data;
        ...
        ...
};
```

- Controller side proxy for an SPI slave device.
- Used to interchange data between an SPI slave & CPU memory.
- Device specific template.

Programming with SPI - Structures

struct spi_board_info

```
{
        const void * platform_data;
        void * controller_data;
        u16 chip_select;
        u16 mode;
        ...
        ...
};
```

- Board-specific template for a SPI device.
- Useful when adding new SPI devices to the device tree.
- Hold information which can't always be determined by drivers.

Programming with SPI - Structures

struct spi_driver

```
{
        const struct spi_device_id * id_table;
        int (*probe) (struct spi_device *spi);
        int (*remove) (struct spi_device *spi);
        void (*shutdown) (struct spi_device *spi);
        struct device_driver driver;
};
```

- Host side "protocol" driver.
- Represents the kind of device driver that uses SPI messages.
- Works through messages rather than talking directly to SPI hardware.

Programming with SPI - Structures

struct spi_message

```
{
        struct spi_device * spi;
        unsigned frame_length;
        unsigned actual_length;
        int status;
        ...
        ...
};
```

- One multi-segment SPI transaction.
- Used to execute an atomic sequence of data transfers.
- Messages sent to a given spi_device are always executed in FIFO order.

Programming with SPI - Structures

struct spi_transfer

```
{
        const void *tx_buf;
        void *rx_buf;
        unsigned tx_nbits:3;
        unsigned rx_nbits:3;
        u8 bits_per_word;
        u32 speed_hz;
        ...
        ...
};
```

- A read/write buffer pair.
- SPI transfers always write the same number of bytes as they read.
- Protocol drivers should always provide rx_buf and/or tx_buf.

SPI Driver Implementation

Character Device & Character Driver -

A character device is one that can be accessed as a stream of bytes.

A char driver is in charge of implementing this behavior. Such a driver usually implements at least the open, close, read, and write system calls.

SPI Devices can be accessed as a character devices.

SPI Driver Implementation

File Operation structure -

```
struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = mySpiOpen,
    .release = mySpiRelease,
    .read = mySpiRead,
};
```

File Operation Function Definitions -

```
static int mySpiInit(void);
static void mySpiExit(void);
int mySpiOpen(struct inode *, struct file *);
int mySpiRelease(struct inode *, struct file *);
ssize_t mySpiRead(struct file *, char __user *, size_t, loff_t *);
```

SPI Driver Implementation

Function - module_init()

Module initialization function.

- Setting up a char device - *alloc_chrdev_region()*
- Allocate and register structure of type struct cdev - *cdev_alloc()*
- Tell the kernel about it - *cdev_add()*
- Create the class of character device - *class_create()*
- Create the device of the class - *device_create()*
- Register spi driver to the Kernel - *spi_register_driver()*

SPI Driver Implementation

Function - *probe()*

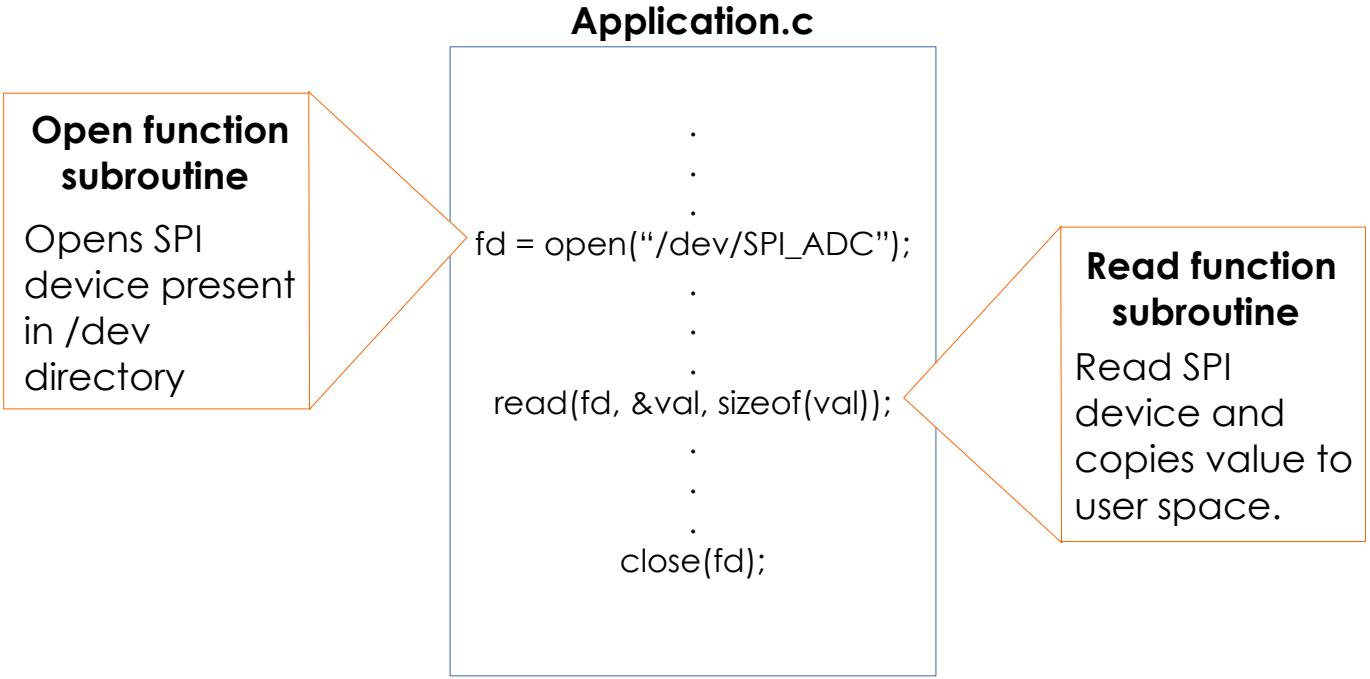
- Purpose is to detect devices residing on the bus & to create device nodes corresponding to these devices.
- The kernel calls the driver's probe() function once for each device.
- Probe function starts the per-device initialization: initializing hardware, allocating resources, and registering the device with the kernel.

SPI Driver Implementation

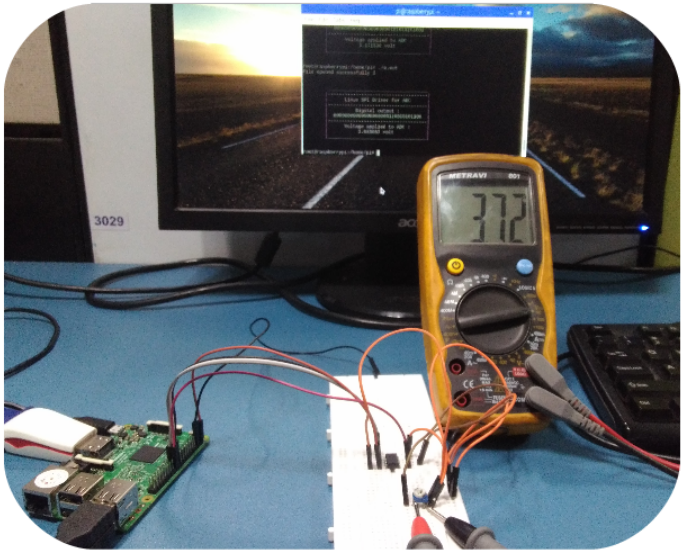
Function - read()

- Purpose is to read data from slave device i.e ADC.
- This function sends command to initialize ADC and simultaneously reads the conversion data from ADC, in its receiver buffer.
- Functions & structures used by read -
  - *struct spi_transfer*
  - *struct spi_message*
  - *spi_message_init()*
  - *spi_message_add_tail()*
  - *spi_sync()*

SPI Application Implementation

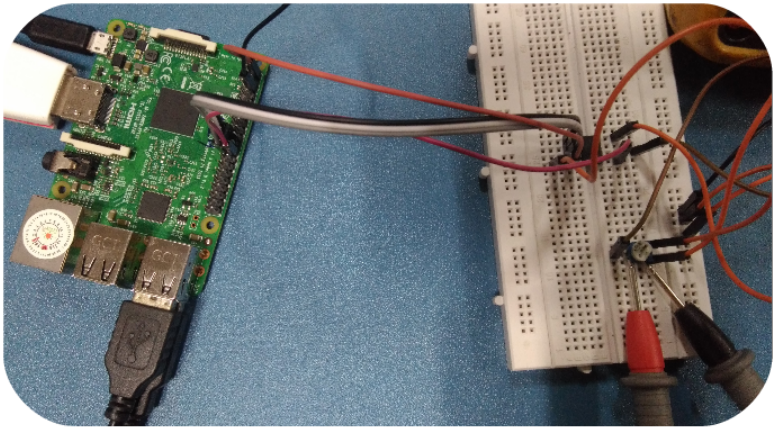


Snapshots



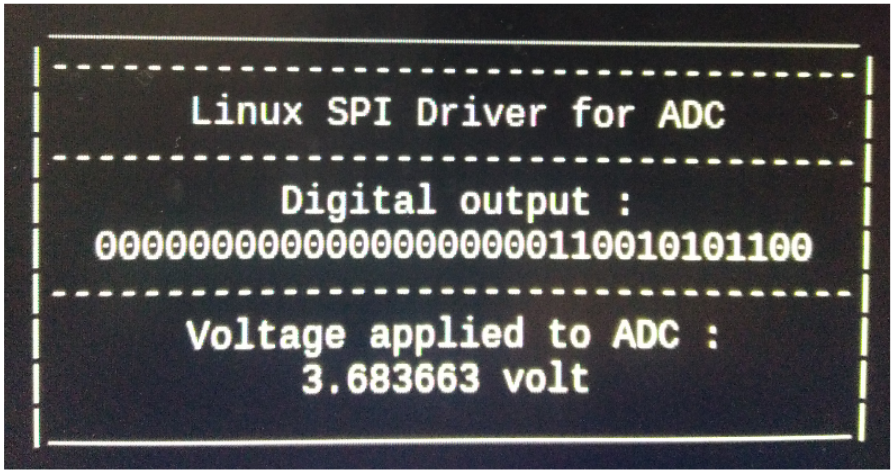
Whole Setup

Connection of ADC with Raspberry-Pi



Results

Voltage applied to ADC's input



Voltage obtained by SPI Application using Linux SPI Driver

Conclusion

SPI – A simple & easy protocol for Embedded Systems.

SPI have 4 modes depending on CPOL & CPHA.

Outcomes

A/D conversion using MCP3202.

Linux SPI Driver has 2 types – Protocol & Controller.

It is easy to write your own SPI driver using Linux's spi.c core.

References

- https://www.kernel.org/doc/Documentation/spi/spi-summary
- http://ww1.microchip.com/downloads/en/DeviceDoc/21034D.pdf
- https://www.kernel.org/doc/html/v4.14/driver-api/spi.html
- https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi#res
- http://ww2.cs.fsu.edu/~rosentha/linux/2.6.26.5/docs/DocBook/kernel-api/ch25.html
- https://e2e.ti.com/support/embedded/linux/f/354/t/518439
- http://www.farnell.com/datasheets/1669376.pdf
- https://www.raspberrypi.org/
- https://elinux.org/Device_Tree_Reference
- https://www.raspberrypi.org/documentation/linux/kernel/building.md

QUESTIONS?

Large enough to Deliver, Small enough to Care

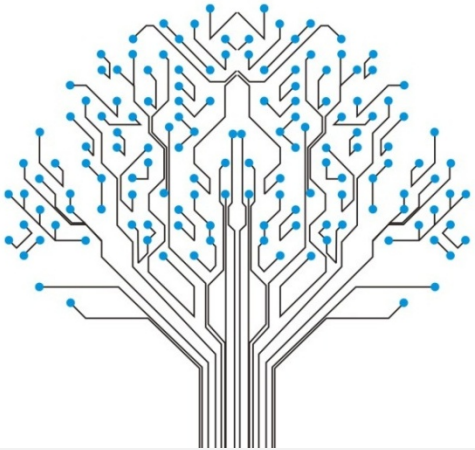
Global Village
IT SEZ
Bangalore

South Main Street
Milpitas
California

Raheja Mindspace
IT Park
Hyderabad

www.globaledgesoft.com

Thank you



Fairness
Learning
Responsibility
Innovation
Respect