Al-Azhar University

Faculty of Engineering, Computers and Systems Department

Smart Charger for Electric Vehicles

A project is submitted in partial fulfillment for the requirements of the degree of Bachelor of Computers and Systems Engineering.

Submitted by

Aya Ramzy Saeed El-Sheikh

ID No.: 440102

Asmaa Anwar Qotob Hashim

ID.:440027

Supervised by

Prof. Ali Rashed

Prof. Maha Medhat

Computers and Systems Engineering

Al-Azhar University July, 2022

Acknowledgment

Above all, praise to ALLAH. By whose grace this work has been completed and never leaving us during this stage.

We would like to Thank to **Dr. Ali Rashed** and **Dr.Maha Medhat,** for supervising the project during the year.

Special Thanks to **Eng. Ahmed Assaf,** General Manager at CoreTech Innovations Company, for suggesting the idea of the project, mentorship, guidance, encouragement and support during the year.

Special Thanks to **Eng. Alaa Nofal,** Electric Engineer at CoreTech Innovations Company, for mentorship and support through the year.

We would like to thank **Eng. Abdulrhman**, Instructor at IMT Company, for Technical support.

We would like to thank **Eng. Omar Mekkawy**, Hardware Engineer, for Technical support

Table of Contents

Chapter3

ADC and Sensors

Chapter 4

Cloud and ESP8266

4.1. Introduction

4.2. ThingSpeak

4.3. ESP WIFI Module

4.3.1 Let’s talk about ESP8266

4.3.2. ESP8266 interfacing with our MCU

4.3.3. AT commands

Chapter 5

Pulse Width Modulation (PWM)

5.1 Introduction

5.2 STM32 Timers – PWM Output Channels

5.3 STM32 Timers In PWM Mode

5.4 STM32 PWM Frequency

5.5 STM32 PWM Duty Cycle

5.7 STM32 PWM Different Modes

5.6 STM32 PWM Resolution

Chapter 6

TFT LCD

6.0. TFT

6.1. TFT LCD overview

6.2. TFT LCD structure

6.3. TFT LCD Features

Chapter 7

Communication Protocols

USART

SPI

List of Figures

Figure No.	Title	Page No

Abstract

Users of electric vehicles face the problem of not monitoring the battery and the long distance between it and the charging station, so this problem was solved by using the Wallbox Charger, as well as it can monitor charging through a website or an application on the phone, where the values and charging are displayed on the site as well as on the charger screen externally. This, and if he then wants to control stopping the charging, he can do so through the site, as well as if the shipper manufacturer wants to make any change, it can do so later through the OTA.

Chapter 1: Introduction

1.1. Overview

We built a complete wall box charger with a charging unit that has power conversion, power monitoring and protection circuit. The communication happens through Control pilot, which is a communication line, used to signal charging level between the car and the EVSE, and can be manipulated by vehicle to initiate charging and CAN bus that is used to exchange the data between the ECUs. Charging information should be updated to a website through esp8622 Wi-Fi module.

1.2. Objective

Implement Smart Charger for Electric Vehicles.

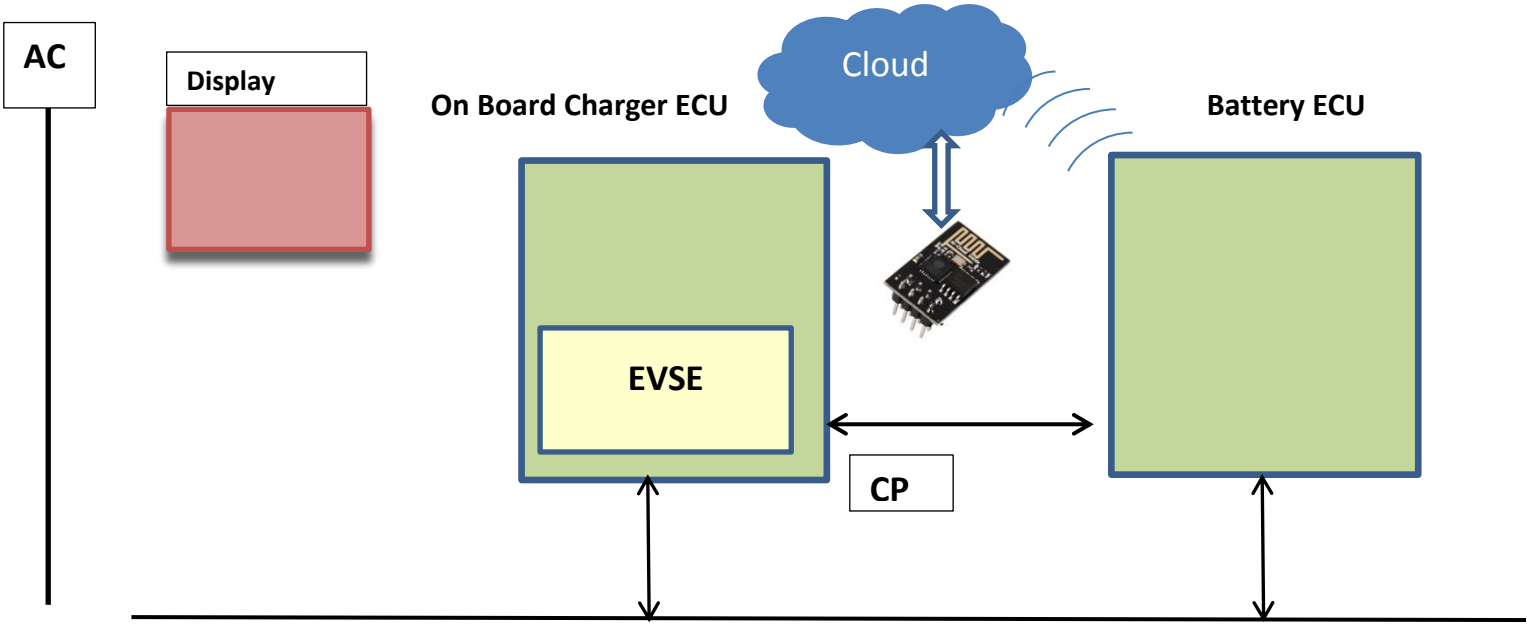
1.3. Problem Description

Users of electric vehicles face the problem of not monitoring the battery and the long distance between it and the charging station, so this problem was solved by using the Wallbox Charger, as well as it can monitor charging through a website or an application on the phone, where the values and charging are displayed on the site as well as on the charger screen externally. This, and if he then wants to control stopping the charging, he can do so through the site, as well as if the shipper manufacturer wants to make any change, it can do so later through the OTA.

.

1.5 System Design

1.5.0 Block Diagram of the System



(1.1)

1.5.1Battery ECU

The primary function of this ECU is the Battery Management System (BMS) to monitor the Battery for which it needs to measure three vital parameters such as the voltage, current and temperature from the Battery, To read the voltage value of the Battery an ADC is used, and Temperature sensor is used to measure temperature.

1.5.2 On Board Charger ECU

This ECU is responsible for charging. It has Power Supply and Regulators which consists of a DC/DC converter as Buck Boost Converter

.

1.5.3 Control Pilot Communication

Control Pilot is the main communicating wire between the electric vehicle supply environment (EVSE) and the electric vehicle (EV). The EVSE (charger) generates 1 kHz, +-12 signal and is transmitted to Electric vehicle through this control pilot (CP) interface. The vehicle respond it by placing different load (resistor) on the line which affects its voltage level which is responded back to electric vehicle supply environment t(EVSE).

1.5.4 Signal BUS

We used it to send some information about EV to the charger for starting the charging.

Used for the connection between the two ECUs.

1.5.5 Cloud

In our project we are provide for the user a website to show and control the charging information like charging percent , current and volt values , charging state and temperature value, so we used thingspeak platform to build a web application that shows information about charging operation and this website connected with MCU through esp8266 wifi module.

1.6. Implementation

We used different tools and technology to implement our system like IDEs , Hardware Tools, IOT and Web Tools and Design Tools.

1.6.1. Tools

1.6.1.0 Software Tools:

- Eclipse IDE C/C++
- Notepad ++
- Thingspeak platform

1.6.1.1 Hardware Tools

- **Microcontroller** : STM32F103C8T6 Blue Pill
- **Sensors :** Current Sensor (ACS712) , Temperature Sensor (LM35)
- **Components :** TFT Display, Relays, Capacitors , Resistors, Comparators, Transistors, Power Supply, Diodes, Zener Diodes,+-12 Generator, potentiometer, DC DC Converter (Boost Converter), Jumpers and ESP8266 (Wifi Module)

1.6.1.2. Design Tools

- EasyEDA

1.6.2. Technology

1.6.2.1. Embedded Systems

This Part is responsible for controlling the entire project by using STM32F103x

1.6.2.2. IOT

This Part is responsible for making our project smart and mentoring the progress of charging by the users, ESP8266 (Wifi Module) used.

1.6.2.3. Power Electronics

This part is used to Implement the circuit of charger and control pilot communication.

1.6.3. System Configurations

So that we can use the system we should configure and initialize each module in the system by a specific configuration.

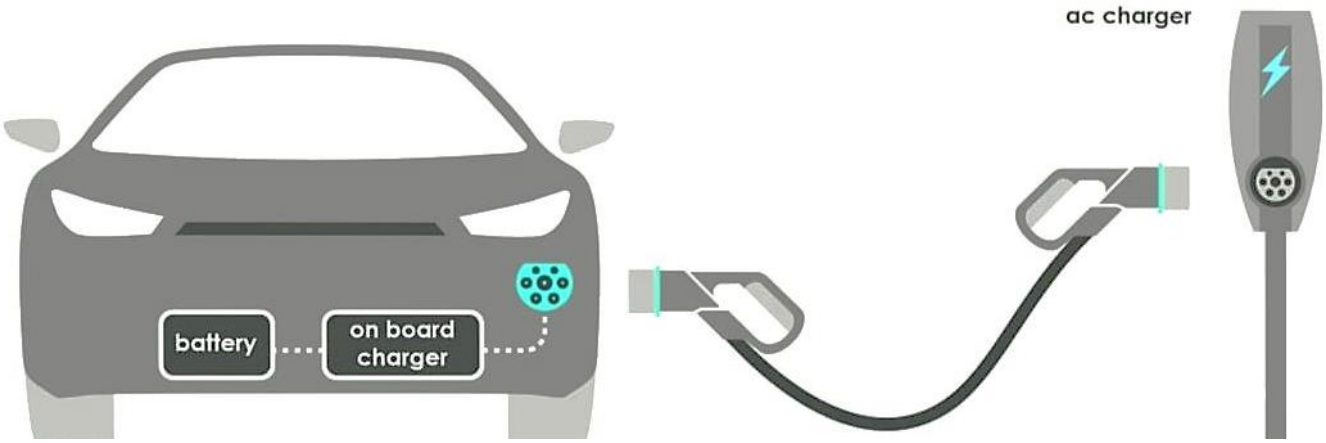
For example:

PWM must configured by a specific duty cycle and frequency.

Software drivers we have implemented in our project

- **GPIO**
- **RCC**
- **SYS-Tick**
- **PWM**
- **ADC**
- **ESP**
- **SPI**
- **UART**
- **WiFi**
- **TFT**
- **LM35 Temperature sensor**
- **Current sensor**
- **Volt sensor**

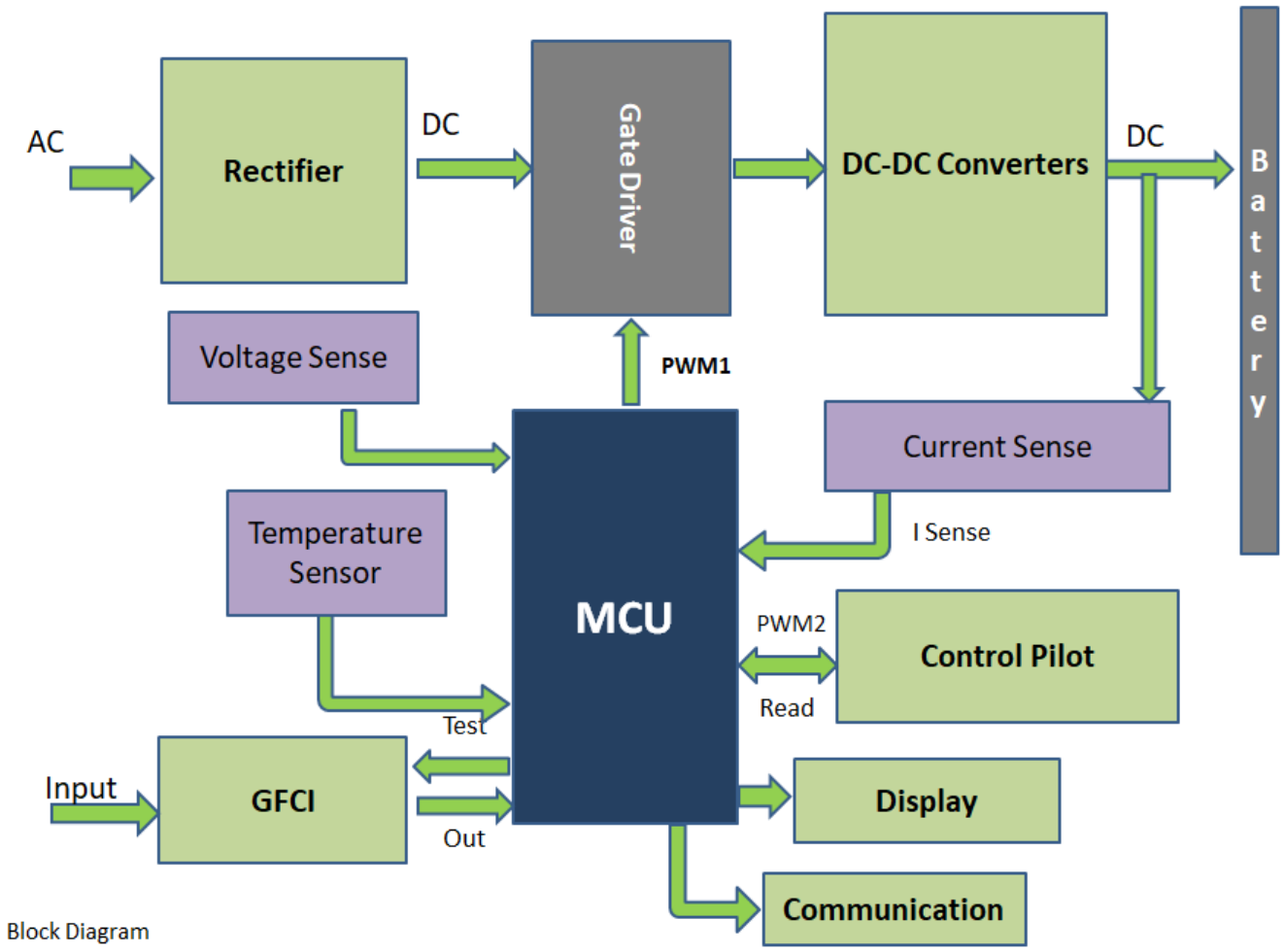
Chapter 2: Charging



2.1. Introduction

In this chapter, a quick overview of the charging part and flow will be explained in detail

2.2. Block Diagram for On Board Charger



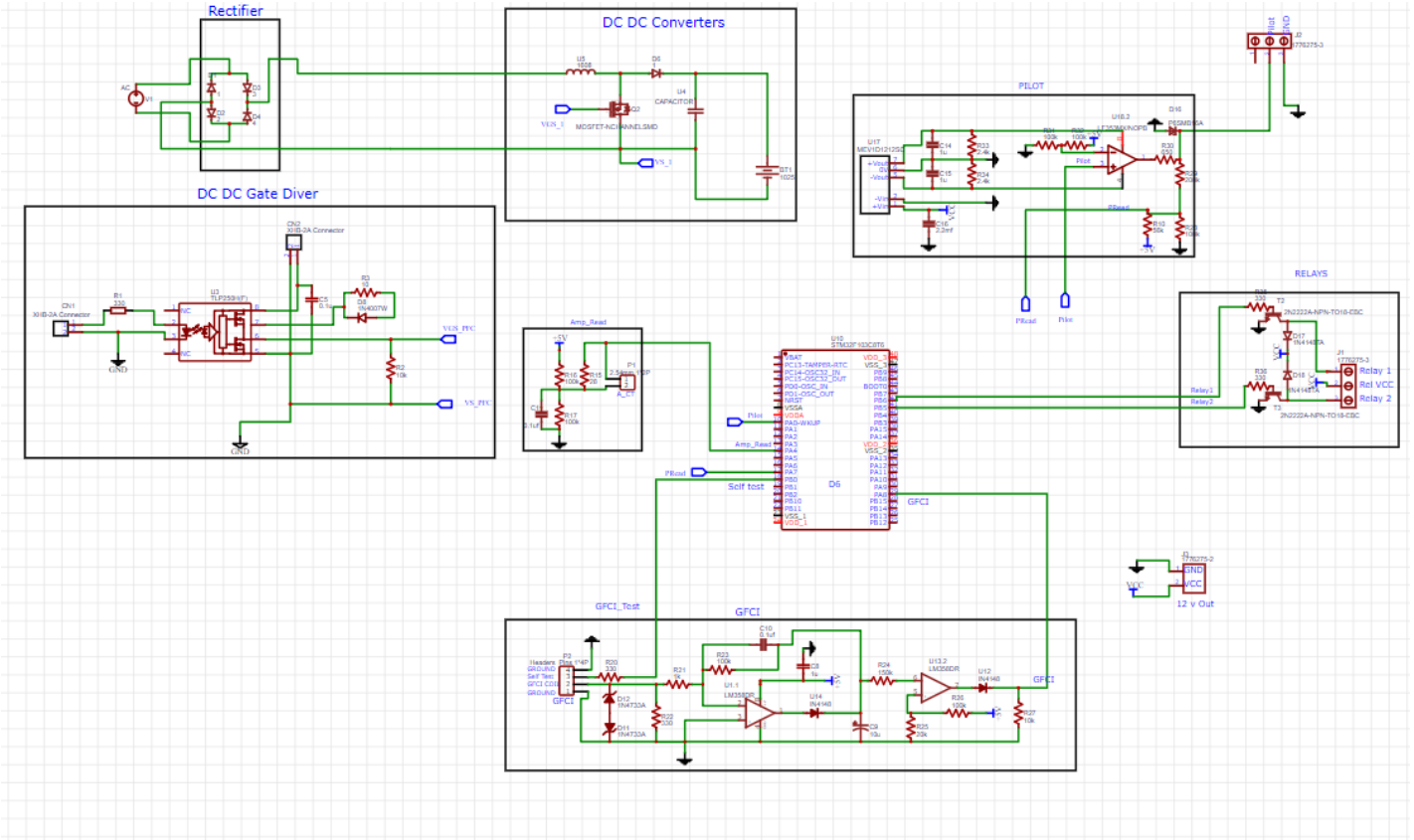
(2.1)

This On Board Charger consists of AC source which is provided by grid. The second step is Rectifier which converts alternating current (AC) to direct current (DC). It is controlled by gate driver which its input is PWM signal provided by MCU, Then DC-DC Converter which can be more types as Buck converter or Boost Converter or Flyback used for controlling the voltage.

Most Charging Circuits must have safety device that quickly breaks an electrical circuit with leakage current to ground. It is to protect equipment and to reduce the risk of serious harm from an ongoing electric shock, so RCD or GFCI used for protection.

The Charger provides PWM signals through Control Pilot Communication Protocol to the Electrical Vehicle to define its current which can be provided for charging, Now EV can respond by putting different load resistors through this wire which will be read my MCU.

2.3. Schematic Diagram



(2.2)

2.3.2. Rectifier

Rectifier is an electrical device that converts alternating current (AC), which periodically reverses direction, to direct current (DC), which flows in only one direction. The bridge rectifier consisting of four diodes enables full wave rectification without the need for a center tapped transformer. The bridge rectifier is an electronic component that is widely used to provide full wave rectification and it is possibly the most widely used circuit for this application.

2.3.3. DC-DC Converter (Boost Converter)

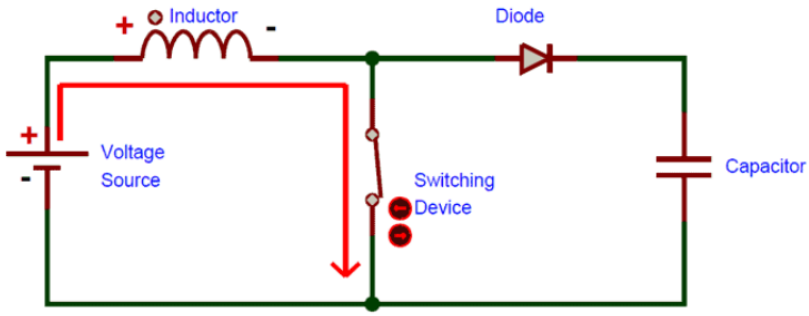
A **boost converter** is one of the simplest **types of switch mode converter**. As the name suggests, it takes an input voltage and boosts or increases it. All it consists of is an inductor, a semiconductor switch (these days it's a MOSFET, since you can get really nice ones these days), a diode, and a capacitor. Also needed is a source of a periodic square wave**.**

Working of the boost converter

We can go through the **working of the boost converter** step by step

STEP 1

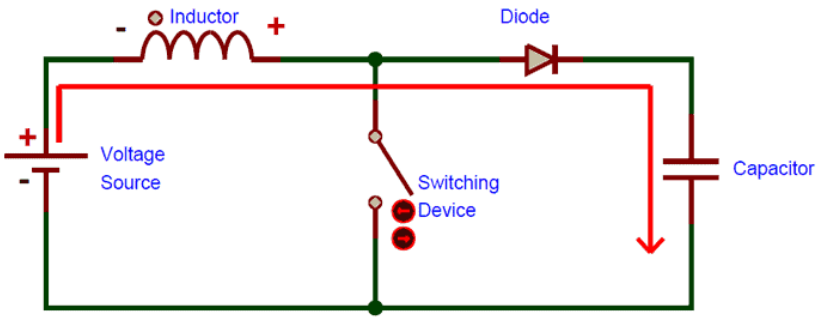
We turn the switch ON. Our signal source turns high, turning on the MOSFET. All the current is diverted through the MOSFET through the inductor. Note that during this time the capacitor stays charged since it cannot discharge through the now back-biased diode. The power source isn't immediately shorter, of course, since the inductor makes the current ramp up relatively slowly.



(2.3)

STEP 2

As the MOSFET turns off, the current to the inductor stops suddenly. The very nature of inductor is to maintain smooth current flow, it doesn't like sudden changes in current. So, it responds to this by generating a large voltage with the opposite polarity of the voltage originally supplied to it using the energy stored in the magnetic field to maintain that current flow.



(2.4)

2.3.4. Control Pilot

Control Pilot is the main communicating wire between the electric vehicle supply environment (EVSE) and the electric vehicle (EV). The EVSE (charger) generates 1 kHz, +-12 signal and is transmitted to Electric vehicle through this control pilot (CP) interface. The vehicle respond it by placing different load (resistor) on the line which affects its voltage level which is responded back to electric vehicle supply environment (EVSE).

Control Pilot Signal States

STATE	PILOT HIGH VOLTAGE	PILOT LOW VOLTAGE	FREQUENCY	RESISTANCE	DESCRIPTION
State A	12 V	N/A	DC	N/A	Not connected
State B	9 V	–12 V	1 kHz	2.74 kΩ	EV connected, ready to charge
State C	6 V	–12 V	1 kHz	882 Ω	EV charging
State D	3 V	–12 V	1 kHz	246 Ω	EV charging, ventilation required
State E	0 V	0 V	N/A	—	Error
State F	N/A	–12 V	N/A	—	Unknown error

(2.5)

State A, State B and State C are the basic functionality state of the CP interface and the remaining state D, E and E occurs only in the rare case. The normal connection process follows the following steps:

- The EVSE generates +12v signal and place it in CP interface. This transmission signals the vehicle when the plug is connected.
- When the plug is connected, the vehicle places a 2.74-kΩ load on the pilot line, which drops the voltage from 12v to 9 V.
- The EVSE moves to state B which indicates that EV is connected and it is ready to charge. The EVSE enables PWM signal which indicates vehicle how much current it can draw.
- The EVSE also on the relay so that the mains supply reaches the EV terminal for powering the vehicle.
- The vehicle starts to draw power and places a 822-Ω load on the pilot line, which drops the voltage from 9 V to 6 V, signaling the EVSE that charging has started.
- Even when the vehicle is fully charged, Most of the vehicle continues to pull a low amount of power in the state C. so we can end the charging process by unplugging the cable from the slot. Once we unplug the cable the voltage switch back to 12V and enters the state A.

2.3.4.1. Control Pilot (Current limit)

Charger uses Pulse Width Modulation (PWM) signal to determine the maximum current that a vehicle can draw from the charging station. The duty cycle of the pilot signal determine the amount of current that the vehicle can draw from the EVSE. The vehicle uses this current in the charging circuitry which has direct relation with the charging time. The current rating is primarily determined by the EVSE source and electromechanical components used in EVSE such as conductors, relays, contractors etc.

The relation between duty cycle and current is provided by two different equations depending upon the current range:

FOR 6A TO 51A SERVICE:
DUTY CYCLE = AMPS/0.6

FOR 51A TO 80A SERVICE:
DUTY CYCLE = AMPS/2.5 + 64

Pilot signal Example Duty cycle

AMPS	DUTY CYCLE
5	8.3%
15	25%
30	50%
40	66.6%
65	90%
80	96%

(2.6)

2.3.5. Current, Voltage and Temperature Sense

We need to measure current and volt to generate the current that EV wants and mentor the output voltage from Boost Converter.

2.3.6. Display

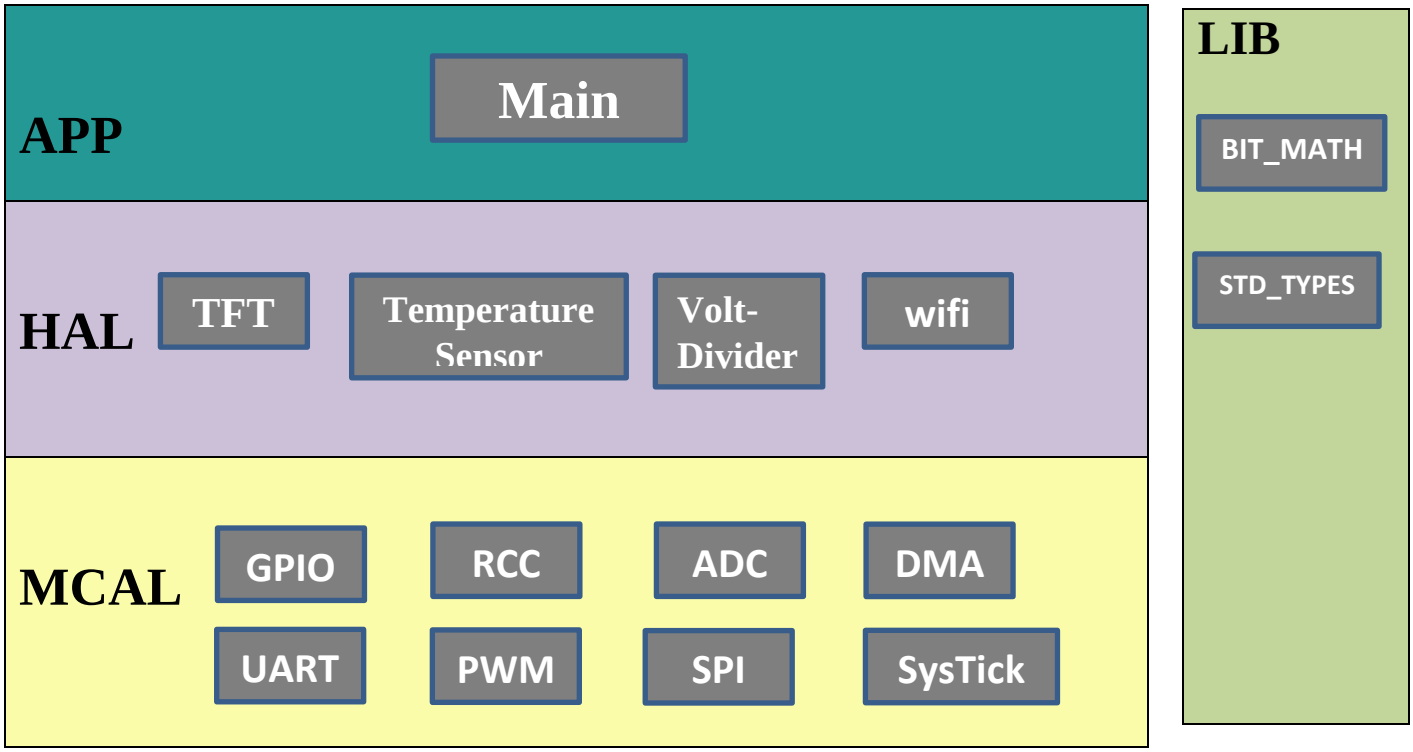
We used TFT to display the values of volt, current and the value of charging to the user for mentoring the charging.

Chapter3

ADC and Sensors

3.1. Software Architecture

Software is following layered architecture pattern, it consists of 3 main layers which are microcontroller abstraction layer (MCAL), hardware abstraction layer (HAL), application layer (APP).



(3.1)

3.1.1. MCAL layer

It is responsible to provide APIS for all microcontroller hardware peripherals.

- RCC

This peripheral function is to set microcontroller clock and reset configuration

- GPIO

This peripheral allows us to interact with microcontroller output and input pins through memory mapped registers.

- UART

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.

- ADC

An ADC (Analog-To-Digital) converter is an electronic circuit that takes in an analog voltage as input and converts it into digital data.

- DMA

Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. This keeps CPU resources free for other operations.

- CAN

  The Controller Area Network (CAN bus) is the nervous system, enabling communication with high speed, 'nodes' or 'electronic control units' (ECUS) are like parts of the body, interconnected via the CAN bus. Information sensed by one part can be shared with another.

3.1.2. HAL Layer

It is responsible to provide APIS for Hardware that are External Microcontroller .

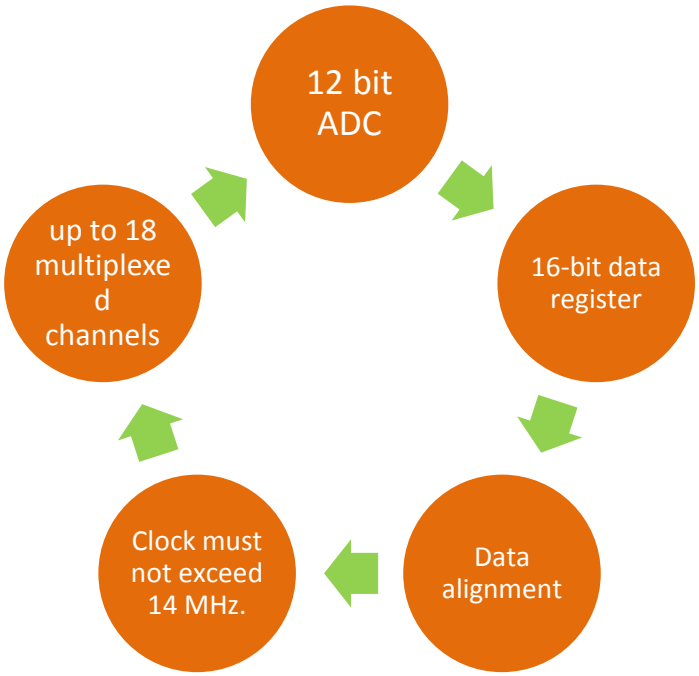
3.1.3. APP Layer

All applications

3.2 ADC

An ADC (Analog-To-Digital) converter is an electronic circuit that takes in an analog voltage as input and converts it into digital data, a value that represents the voltage level in binary code. The ADC samples the analog input whenever you trigger it to start conversion. And it performs a process called quantization so as to decide on the voltage level and its binary code that gets pushed in the output register.

3.2.1. The STM32F103C8 (Blue Pill) ADC Brief



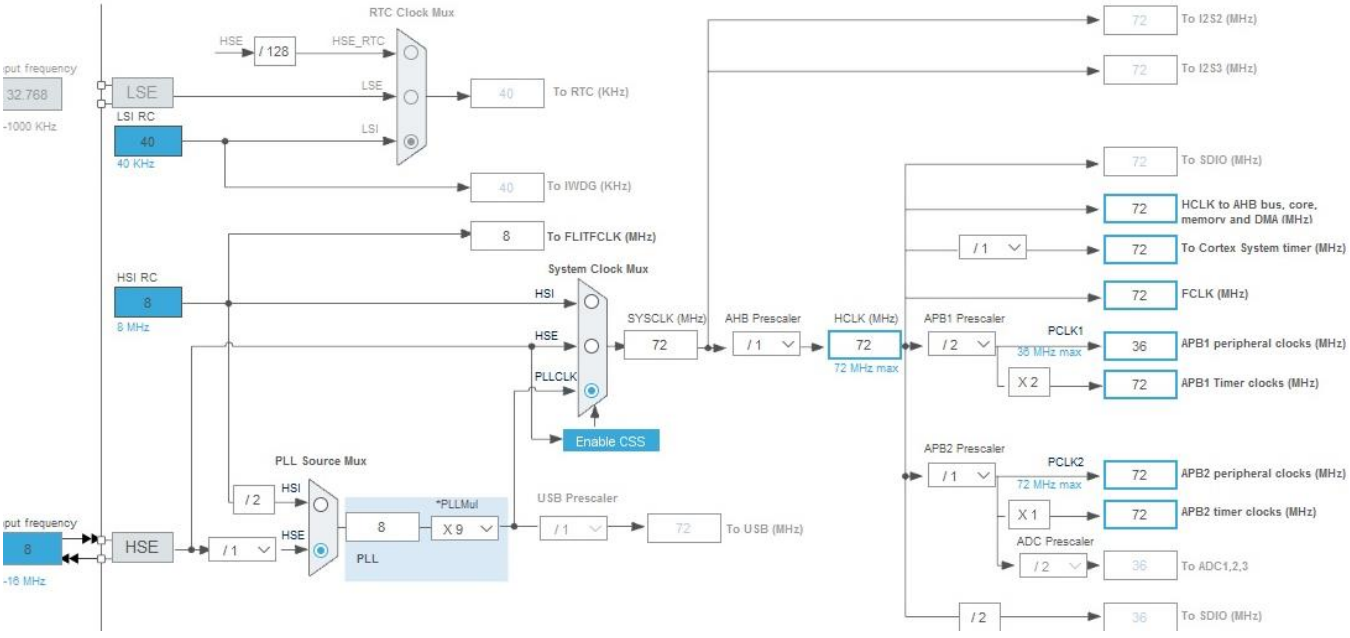
3.2.2. ADC Features

- 12-bit resolution
- Interrupt generation at End of Conversion, End of Injected conversion and Analog watchdog event
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel ‘n’
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual-mode (on devices with 2 ADCs or more)
- ADC conversion time: 1 µs at 56 MHz (1.17 µs at 72 MHz)
- ADC supply requirement: 2.4 V to 3.6 V
- ADC input range: $V_{REF}– \leq V_{IN} \leq V_{REF}+$
- DMA request generation during regular channel conversion

3.2.3. The ADC Clock

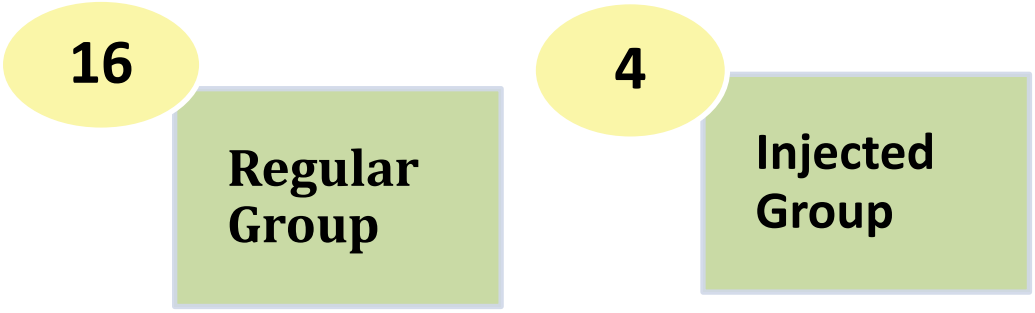
The ADC CLK clock provided by the Clock Controller is synchronous with the PCLK2 (APB2 clock). The RCC controller has a dedicated programmable prescaler for the ADC clock, and it must not exceed 14 MHz.

So we used 8 MHz HSE then used the PLL, MUL Value is 9, so the output of PLL = 72MHz.
Presaler of ADC is equal to 6, so the ADC Clock is equal to 12Mhz.



(3.2)

3.2.4. Channel Selection



The **Regular Group** is composed of up to 16 conversions and The **Injected Group** is composed of up to 4 conversions.

3.2.5. ADC Result Data Alignment

ALIGN bit in the ADC_CR2 register selects the alignment of data stored after conversion. Data can be left or right-aligned as shown in the diagram below.

Figure 27. Right alignment of data

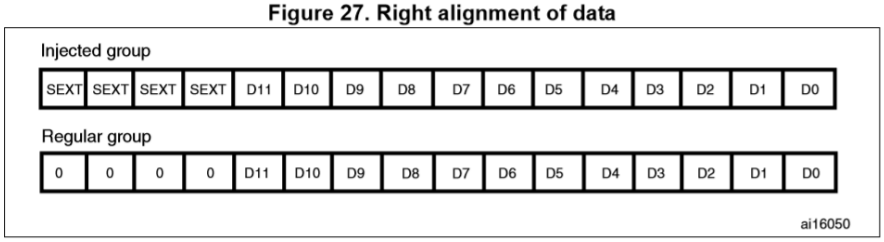
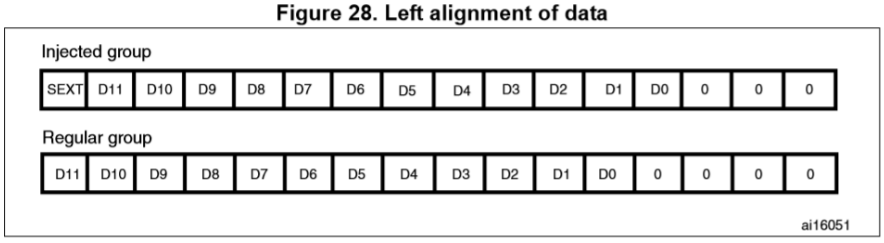


Figure 28. Left alignment of data

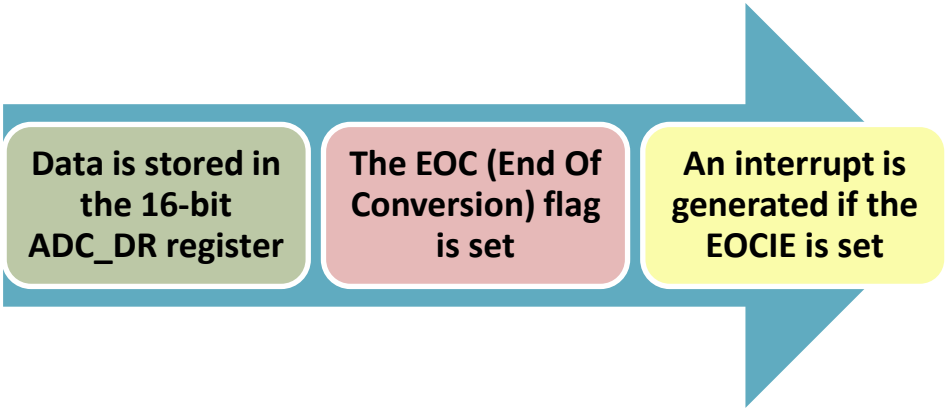


(3.3)

3.2.6. STM32 ADC Modes of Operation

In Single Conversion mode, the ADC does one conversion. This mode is started either by setting the ADON bit in the ADC_CR2 register (for a regular channel only) or by an external trigger (for a regular or injected channel), while the CONT bit is 0. Once the conversion of the selected channel is complete:

If a regular channel was converted:



The ADC is then stopped.

3.2.7. STM32 ADC Sampling Time

ADC samples the input voltage for a number of ADC_CLK cycles which can be modified using the SMP[2:0] bits in the ADC_SMPR1 and ADC_SMPR2 registers. Each channel can be sampled with different sample times.

The **Total ADC Conversion Time** is calculated as follows:

Tconv = Sampling time + 12.5 cycles
Example:
With an ADCCLK = 14 MHz and a sampling time of 1.5 cycles: Tconv = 1.5 + 12.5 = 14 cycles = 1 µs
The **ADC Sampling Rate (Frequency)** is calculated using this formula:
Sampling Rate = 1 / Tconv
For The Previous example where Tconv = 1µs, The sampling Rate = 1000000 = 1Ms/sec.

3.2.8. STM32 ADC Resolution

The STM32 ADC has a resolution of 12-Bit which results in a total conversion time of SamplingTime+12.5 clock cycle.

3.2.9. ADC & DMA

Since converted regular channels value are stored in a unique data register, it is necessary to use DMA for the conversion of more than one regular channel. This avoids the loss of data already stored in the ADC_DR register. Only the end of conversion of a regular channel generates a DMA request, which allows the transfer of its converted data from the ADC_DR register to the destination location selected by the user

3.3. Measuring Analog Voltage

3.3.1. 12 Bit Resolution

This ADC is a 10 channel 12 -bit ADC. Here the term 10 channel implies that there are 10 ADC pins using which we can measure analog voltage. The term 12-bit implies the resolution of the ADC. 12-bit means 2 to the power of ten (2^{12}) which is 4096. This is the number of sample steps for our ADC, so the range of our ADC values will be from 0 to 4095. The value will increase from 0 to 4095 based on the value of voltage per step, which can be calculated by formula

VOLTAGE / STEP = REFERENCE VOLTAGE / 4096 = (3.3/4096= 8.056mV) per unit

Analog signals like sensor's output in volts has to be converted into digital values for processing and the conversion needs to be accurate .When a input analog voltage is given to STM32 at its Analog inputs, the analog value is read and stored in a integer variable. That stored Analog value (0-3.3V) is converted into integers values (0-4096) using the formula below:

INPUT VOLTAGE = (ADC Value / ADC Resolution) * Reference Voltage

```
/**********************************************************/
/* Author     : Aya Ramzy                                 */
/* Date       : 31 MAR 2022                               */
/* Version    : V01                                       */
/**********************************************************/

#include STD_TYPES.h
#include BIT_MATH.h

#include ADC_Interface.h
#include VOLT_Interface.h
#include VOLT_Config.h



f32 VOLT_u32ReadValue (void)
{
    u16 Local_u16ReadValue = 0 ;
    f32 Local_f32ReadVolt =0.0;
    //Read ADC Value
    Local_u16ReadValue = u16 ADC_u16ReadData();
    //Calculate the voltage value from the ADC value
    Local_f32ReadVolt = (f32(Local_u16ReadValue)/RESOLUTION)*VREF ;

}
```

3.4. ACS712 Current Sensor

(3.4)

3.4.1. Introduction

Hall Effect Sensors are transducer type components that can convert magnetic information into electrical signals for subsequent electronic circuit processing. Generally, current sensors use the Hall Effect to convert current inputs into voltage outputs. In the Hall effect, electrons from an electric current flow through a magnetic field plate. The field then causes the electrons to "push" to one side of the plate and produce a voltage difference between the two sides. The difference in voltage from the side of the plate is the output of the sensor.

ACS712 is a current sensor that can operate on both AC and DC. This sensor operates at 5V and produces an analog voltage output proportional to the measured current. This tool consists of a series of precision Hall sensors with copper lines.

The output of this instrument has a positive slope when the current increases through the copper primary conduction path (from pins 1 and 2 to pins 3 and 4). The internal resistance of the conduction path is 1.2 m ohm.

This sensor has an output voltage of **Vcc x 0.5 = 2.5** at the input current and a 5V Vcc power supply.

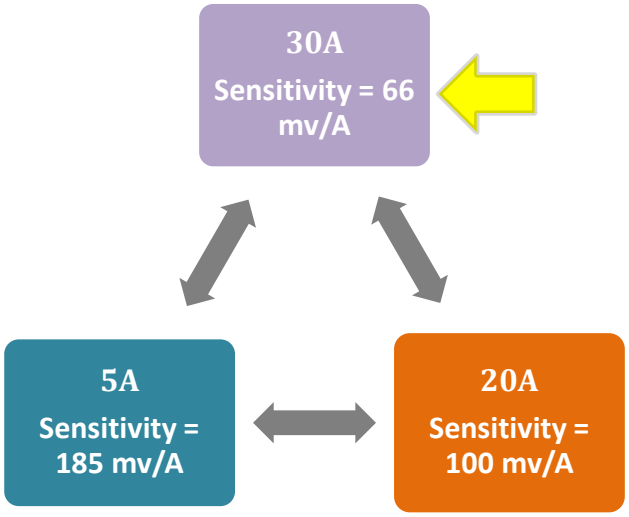
From DataSheet

COMMON OPERATING CHARACTERISTICS[1] over full range of T_A, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
ELECTRICAL CHARACTERISTICS						
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Supply Current	I_{CC}	V_{CC} = 5.0 V, output open	–	10	13	mA
Output Capacitance Load	C_{LOAD}	VIOUT to GND	–	–	10	nF
Output Resistive Load	R_{LOAD}	VIOUT to GND	4.7	–	–	kΩ
Primary Conductor Resistance	$R_{PRIMARY}$	T_A = 25°C	–	1.2	–	mΩ
Rise Time	t_r	I_P = I_P(max), T_A = 25°C, C_{OUT} = open	–	5	–	µs
Frequency Bandwidth	f	–3 dB, T_A = 25°C; I_P is 10 A peak-to-peak	–	80	–	kHz
Nonlinearity	E_{LIN}	Over full range of I_P	–	1.5	–	%
Symmetry	E_{SYM}	Over full range of I_P	98	100	102	%
Zero Current Output Voltage	$V_{IOUT(Q)}$	Bidirectional; I_P = 0 A, T_A = 25°C	–	V_{CC} × 0.5	–	V
Power-On Time	t_{PO}	Output reaches 90% of steady-state level, T_J =25°C, 20 A present on leadframe	–	35	–	µs
Magnetic Coupling[2]			–	12	–	G/A
Internal Filter Resistance[3]	$R_{F(INT)}$			1.7		kΩ

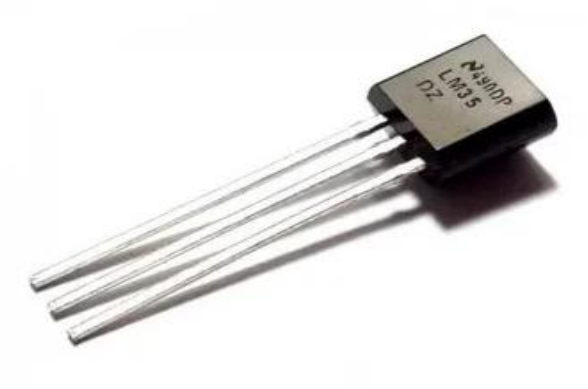
(3.5)

There are three types based on the readable current range:



3.5. LM35 Temperature Sensor

3.5.1 LM35 overview



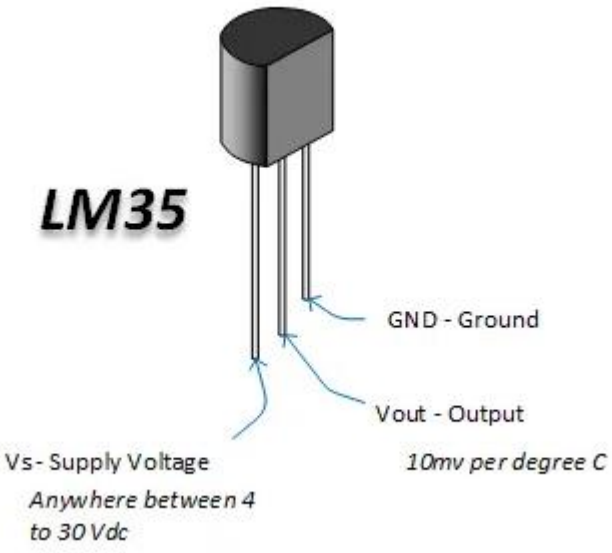
Figure(3-5-1)

The LM35 is a temperature sensor widely used in electronic projects and midrange devices. It has limited usage in industrial applications due to maximum temperature range limitations. It's rated to a full range of −55°C to 150°C. You can just power it up and instantly read the voltage level on the output terminal. The Vout of the sensor directly maps to the sensor's temperature as we'll see hereafter.

3.5.2 LM35 Features

- Calibrated Directly in Celsius (Centigrade)
- Linear + 10-mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at 25°C)
- Rated for Full −55°C to 150°C Range
- Low-Cost
- Operates From 4 V to 30 V
- Less Than 60-µA Current Drain
- Non-Linearity Only ±¼°C Typical
- Low-Impedance Output, 0.1 Ω for 1-mA Load

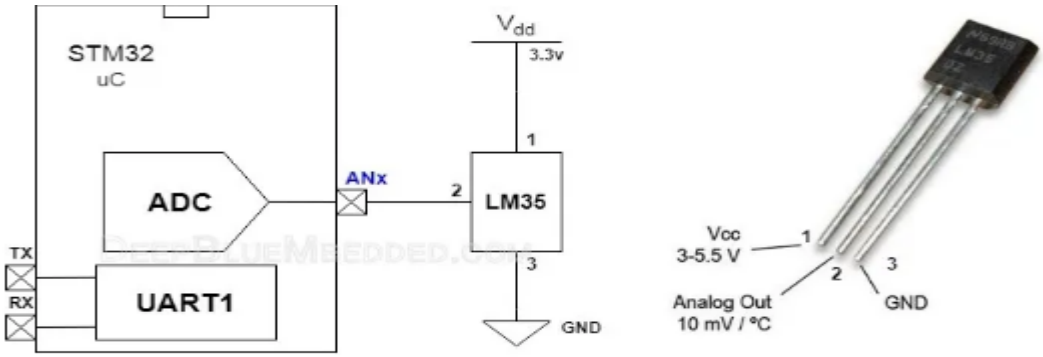
3.5.3 LM35 Pins Description



Figure(3-5-2)

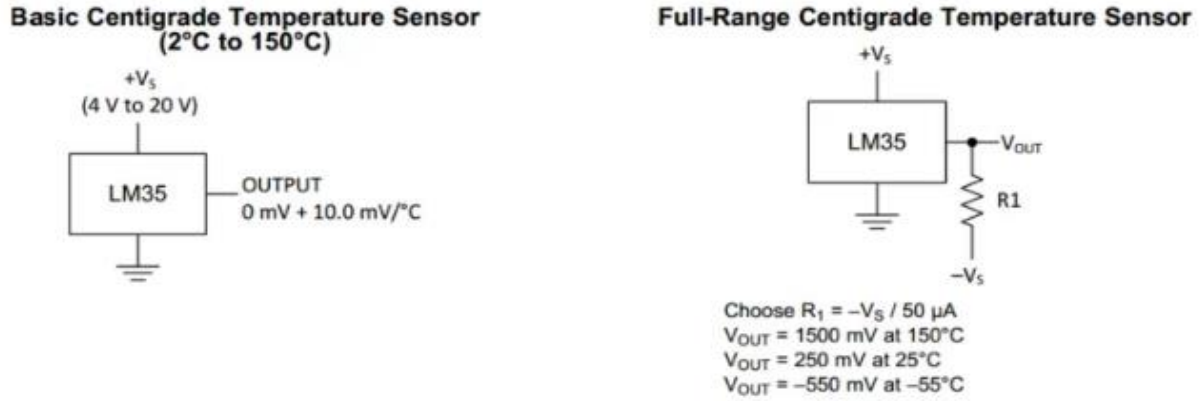
3.5.4 LM35 Interfacing

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature. It has a linear + 10-mV/°C Scale Factor. Which means you'll have to measure its output voltage and divide it by 0.01 to get the temperature reading in °C, that's all.



Figure(3-5-3)

The LM35 temperature sensor can be used in a couple of configurations. The basic one is the full positive temperature range (from 2°C up to +150°C). And the full range that can go below zero degrees (from -55°C up to +150°C). The basic configuration requires no external components besides the LM35 itself, and the full range configuration requires an additional resistor which can be calculated using the formula in the diagram below.



Figure(3-5-4)

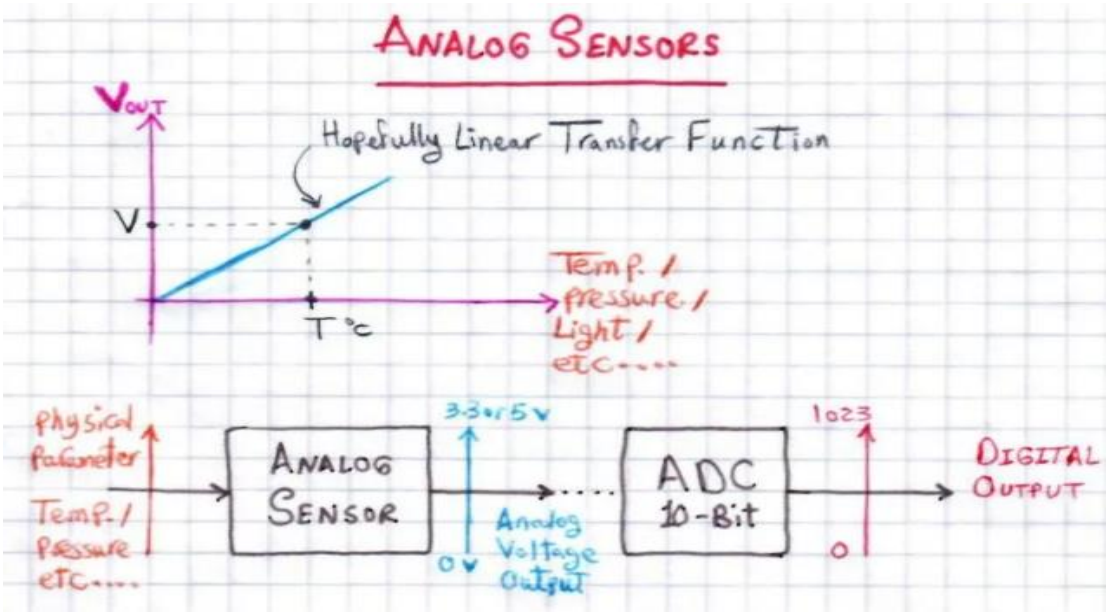
3.5.5 LM35 Transfer Function

The accuracy specifications of the LM35 are given with respect to a simple linear transfer function:

V_{OUT} = (10 mv / °C) × T

where **V_{OUT}** is the LM35 output voltage & **T** is the temperature in °C

Result Conversion



Figure(3-5-5)

The output of an ADC will be a digital (numeric) value stored in a memory register. Which means you'll have to manipulate & convert this value to estimate the voltage level on the analog input pin. Therefore, you'll be able to use the transfer function equation of your sensor to know the physical parameter real measurement. Whether it's temperature, pressure, or light intensity, it's all the same as seen by the processor.

Numeric Example

[Assuming 10-Bit ADC with Vref=5v and Temperature Sensor Transfer Function of 10mV/°C]

Let's start with an A/D Result of 100

ADC Result = 100

So, The Sensor's Output Voltage (V_{OUT}) is calculated as follows

ADC_Resolution = Vref / 2^10 = 5v/1024 = 4.88mV

V_{OUT} = ADC_Resolution **x** ADC Result = 0.488Volts

Therefore, The Temperature Measured By The Sensor is as follows

Temperature = V_{OUT}/10mV = **48.82**°C

Chapter4

Cloud and ESP8266

4.1. Introduction

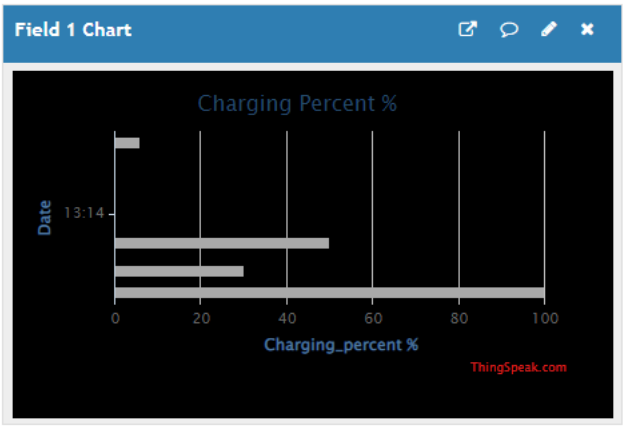
In our project we are provide for the user a website to know the charging information like charging percent , current and volt values , charging state and temperature value, so we used thingspeak platform to build a web application that shows information about charging operation and this website connected with MCU through esp wifi module .

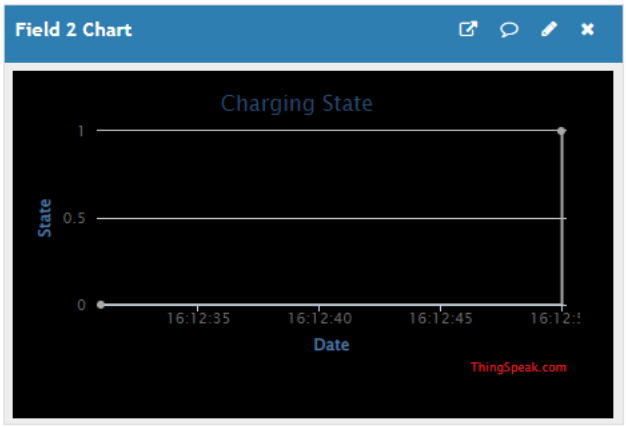
4.2. ThingSpeak

Platform Thingspeak is an open source software written in Ruby which allows users to communicate with internet enabled devices It facilitates data access, retrieval and logging of data by providing an API to both the devices and social network websites. ThingSpeak was originally launched by ioBridge in 2010 as a service in support of IoT applications. ThingSpeak has integrated support from the numerical computing software MATLAB from Math Works allowing ThingSpeak users to analyze and visualize uploaded data using MATLAB without requiring the purchase of a MATLAB license from MathWorks 4.2.1. Our webpage on ThingSpeak.

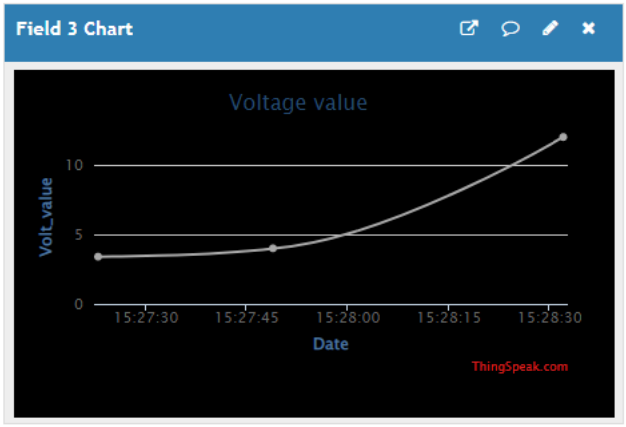
We have built five fields on the channel:

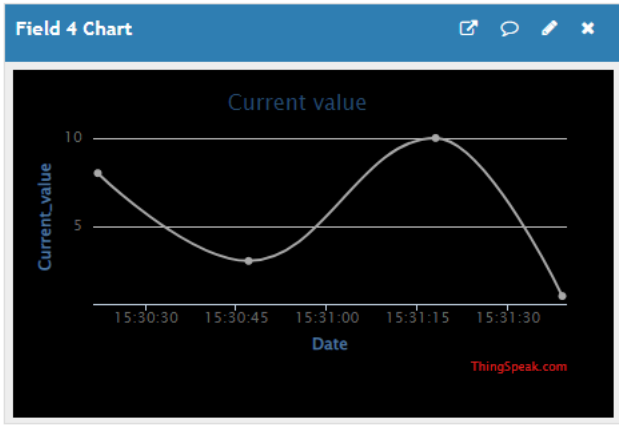
- **Field 1:** Charging percent that represent s the percentage 0% to 100% of the charging with the date .
- **Field 2:** Charging states that represents the state of the charging: that represented with 1 value, Not charging : that represented with 0 value.
- **Field 3:** Voltage value that represent the value of the volt with the date
- **Field 4:** Current value that represent the value of the current with date
- **Field 5:** Temperature value that represent the value of the temperature with the date


Field 1 Chart
Charging Percent %
Date
13:14
0
20
40
60
80
100
Charging_percent %
ThingSpeak.com

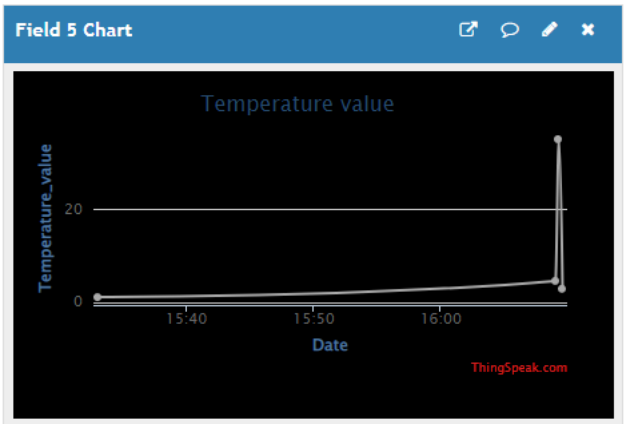

Field 2 Chart
Charging State
State
1
0.5
0
16:12:35
16:12:40
16:12:45
Date
ThingSpeak.com

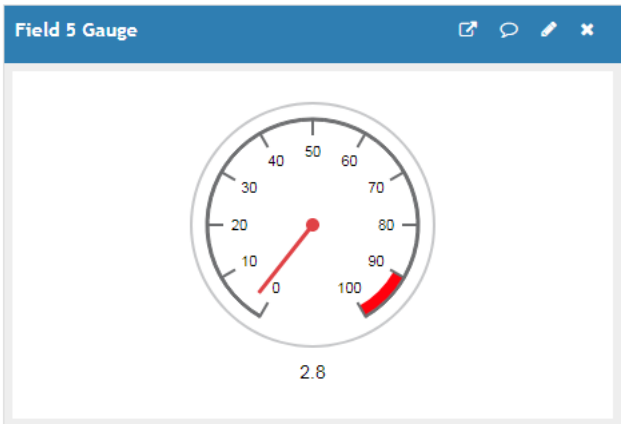
(4.2.1)


Field 3 Chart
Voltage value
Volt_value
10
5
0
15:27:30
15:27:45
15:28:00
15:28:15
15:28:30
Date
ThingSpeak.com


Field 4 Chart
Current value
Current_value
10
5
15:30:30
15:30:45
15:31:00
15:31:15
15:31:30
Date
ThingSpeak.com

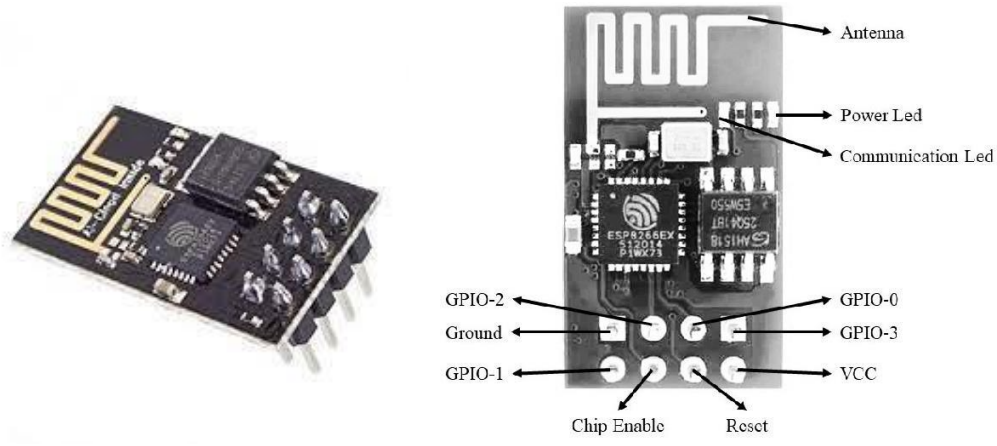
(4.2.2)


Field 5 Chart
Temperature value
Temperature_value
20
0
15:40
15:50
16:00
Date
ThingSpeak.com


Field 5 Gauge
0
10
20
30
40
50
60
70
80
90
100
2.8

(4.2.3)

4.3. ESP WIFI Module



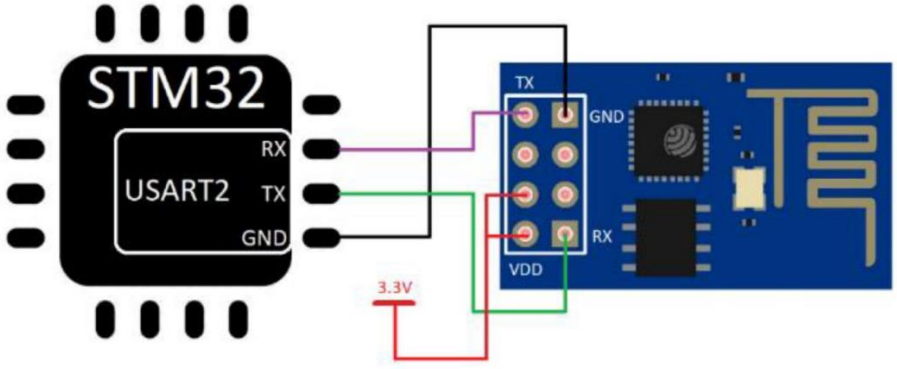
(4.3)

4.3.1 Let's talk about ESP8266

ESP8266 WiFi Module offers complete networking solutions to our DIY (Do-it-yourself) and IoT (Internet of Things) projects. It provides WiFi connectivity to any microcontroller through its full TCP/IP Stack. The ESP8266 WiFi module and the microcontroller can be interface through UART and with the help of a wide range of AT Commands, The Microcontroller can control the ESP Module.

4.3.2. ESP8266 interfacing with our MCU

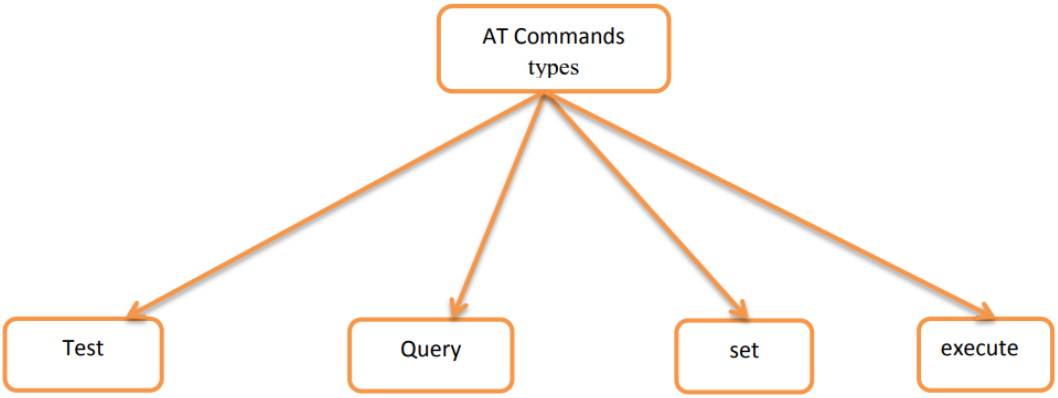
Our MCU (STM32) communicates with esp8266 over UART communication protocol through transmitter and receiver pins in STM32 and esp8266.



(4.3.2)

4.3.3. AT commands

The AT Commands of the ESP8266 WiFi Module are responsible for controlling all the operations of the module like restart, connect to WiFi, change mode of operation and so forth. There are a total of 88 AT Commands for ESP8266 WiFi Module. I will not talk about all the 88 AT Commands but just commands that we used on our project.



Test Commands:

The Test AT Commands of ESP8266 WiFi Module are used to get the parameters of a command and their range.

Query Commands:

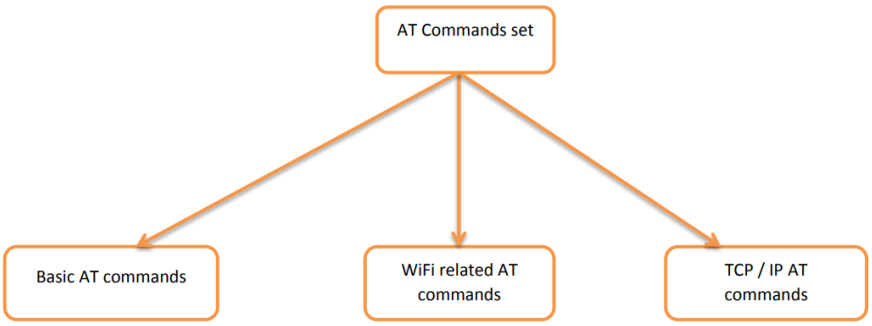
The Query Commands returns the present value of the parameters of a command.

Set Commands:

The Set Commands are used set the values of the parameters in the commands and also runs the commands.

Execute Commands:

The Execute Commands will run the commands without parameters.



Command	Response	Description
AT	OK	It's used to test the setup function of your wireless WiFi module
. AT+RST	OK	It's used to restart the module
ATE	OK	This command ATE is an AT trigger command echo. It means that entered commands can be echoed back to the sender when ATE
		command is used. Two parameters are possible. The command returns "OK" in normal cases and "ERROR" when a parameter other than 0 or 1 was specified.
AT+CWMODE	OK	The function of this AT command is to get the value scope of WiFi mode, including station mode, softAP mode, and station+softAP mode, enquiry about the information of WiFi mode, or set the WiFi mode.
AT+CWJAP_CUR	OK	The function of this AT command is to connect with wifi , it takes wifi name as a first parameter and password as a second parameter
AT+CIPSTART	OK Or ERROR If connection already exists, returns ALREAY CONNECT	This is used to connect our wifi module with the server
AT+CIPSEND	ERROR OR SEND OK	This is used to send the length of the GET command to the server

Chapter 5

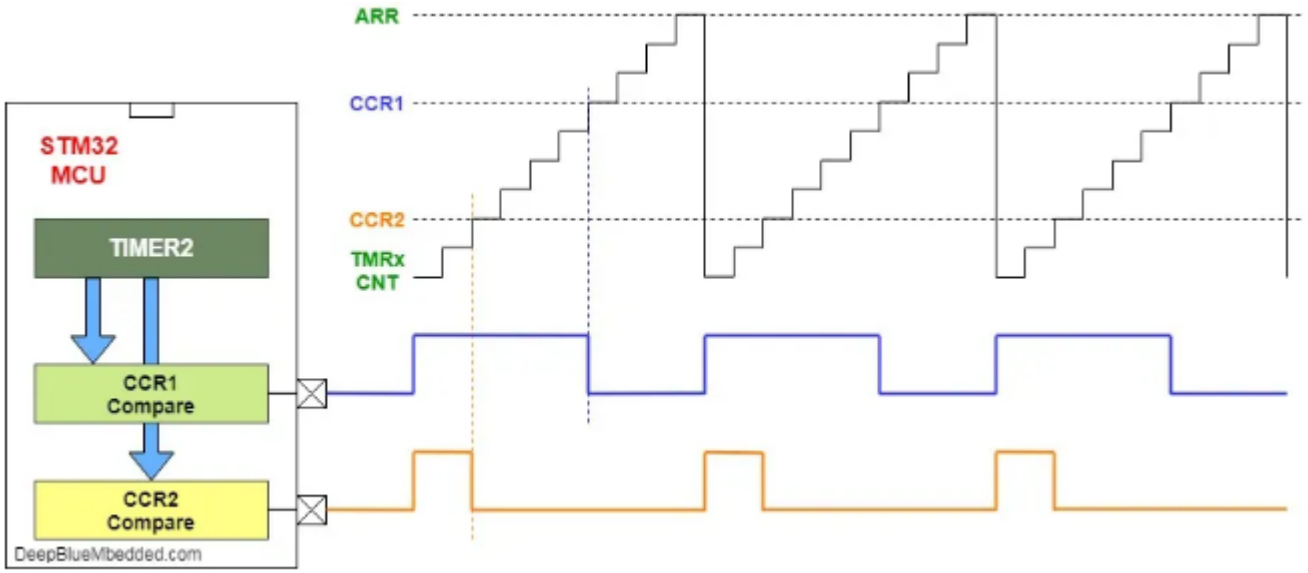
Pulse Width Modulation

(PWM)

5.1 Introduction

Pulse Width Modulation is one of the most important functions of timer . PWM is a technique to control analogue circuits with output from microcontroller. As we all know that Analogue signal is the one whose value varies continuously with time whereas digital signals can only be either high or low . PWM is used for generating an analogue signal using digital source.
the timer modules can operate a variety of modes one of which is the PWM mode.
Where the timer gets clocked from an internal source and counts up to the auto-reload register value, then the output channel pin is driven HIGH,And it remains until the timer counts reach the CCRx register value, the match event causes the output channel pin to be driven LOW.
And it remains until the timer counts up to the auto-reload register value, and so on.
The resulting waveform is called PWM (pulse-width modulated) signal. Whose frequency is determined by the internal clock, the Prescaler, and the ARRx register. And its duty cycle is defined by the channel CCRx register value.
The PWM doesn't always have to be following this exact same procedure for PWM generation, however, it's the very basic one and the easier to understand the concept. It's called the up-counting PWM mode.

The following diagram shows you how the ARR value affects the period (frequency) of the PWM signal. And how the CCRx value affects the corresponding PWM signal's duty cycle. And illustrates the whole process of PWM signal generation in the up-counting normal mode.

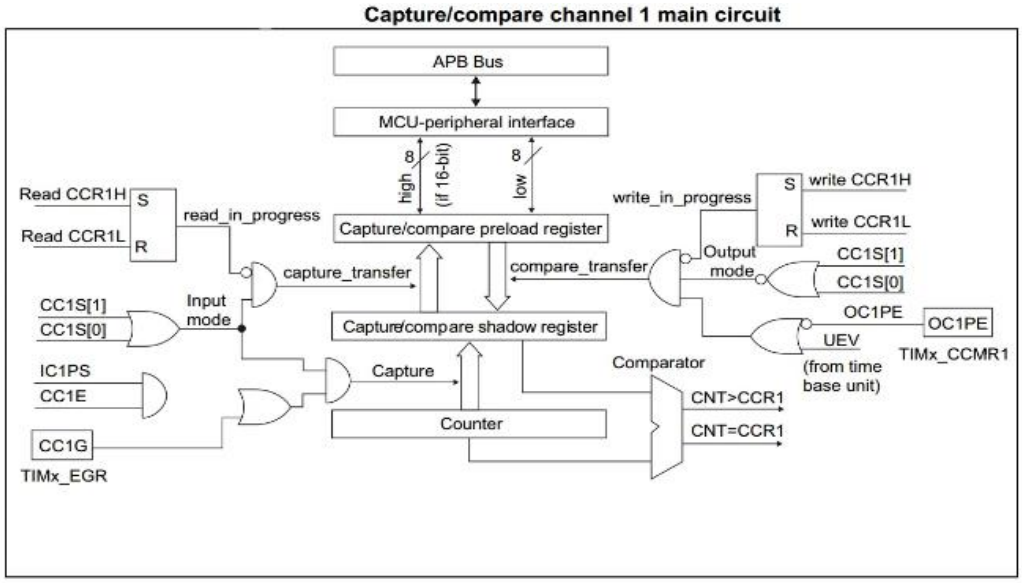


Figure(5.1)

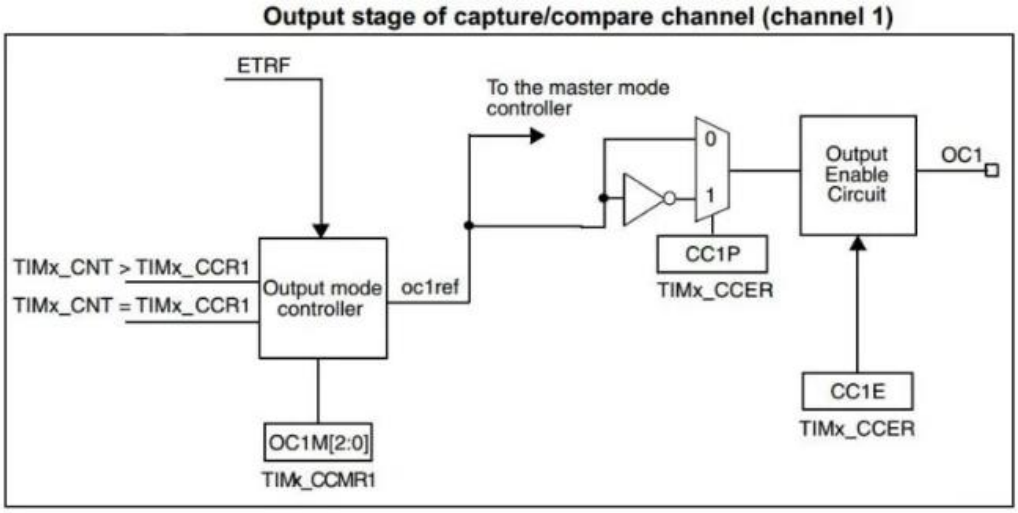
5.2 STM32 Timers – PWM Output Channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an input stage for capture (with a digital filter, multiplexing, and Prescaler) and an output stage (with comparator and output control). The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

And here is a diagram for the capture/compare channel 1 Full Circuitry



Figure(5.2)



Figure(5.3)

A single STM32 timer usually has multiple channels (4, 6, or whatever found in the datasheet). Therefore, using a single timer you can independently generate multiple PWM signals with different duty cycles of course, but they'll share the same timing (same frequency), and all of them will be in sync. We'll do this in the 2nd LAB in this tutorial after we set up a single PWM channel and get everything up and running.
Here is a snapshot of the general-purpose Timer2 diagram, which highlights the presence of multiple output compare channels and output drivers.

Figure(5.4)

5.3 STM32 Timers In PWM Mode

Pulse width modulation mode allows generating a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register. The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or '111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. The user must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx_CR1 register.
OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. For applications where you need to generate complementary PWM signals, this option will be suitable for you.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx≤TIMx_CNT or TIMx_CNT≤TIMx_CCRx (depending on the direction of the counter).
The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

5.4 **STM32 PWM Frequency**

In various applications, you'll be in need to generate a PWM signal with a specific frequency. In servo motor control, LED drivers, motor drivers, and many more situations where you'll be in need to set your desired frequency for the output PWM signal.
The PWM period ($1/F_{PWM}$) is defined by the following parameters: ARR value, the Prescaler value, and the internal clock itself which drives the timer module F_{CLK}. The formula down below is to be used for calculating the FPWM for the output. You can set the clock you're using, the Prescaler, and solve for the ARR value in order to control the FPWM and get what you want.

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

5.5 **STM32 PWM Duty Cycle**

In normal settings, assuming you're using the timer module in PWM mode and generating PWM signal in edge-aligned mode up-counting configuration. The duty cycle percentage is controlled by changing the value of the CCRx register. And the duty cycle equals (CCRx/ARR) [%].

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx}[\%]$$

5.6 STM32 PWM Resolution

One of the most important properties for a PWM signal is the resolution. It's the number of discrete duty cycle levels that you can set it to. This number determines how many steps the duty cycle can take until it reaches the maximum value. So, the step size or the number of duty cycle steps can tell how fine can you change the duty cycle in order to achieve a certain percentage. This can be extremely important in some audio applications, motor control, or even light control systems

This is the STM32 PWM resolution formula that can be used to calculate the resolution of the PWM signal at a specific frequency or even the opposite. If you're willing to get a 10-Bit resolution PWM signal, what should the frequency be in order to achieve this? And so on.

$$Resolution_{PWM} = \frac{log(\frac{F_{CLK}}{F_{PWM}})}{log(2)}[Bits]$$

5.7 STM32 PWM Different Modes

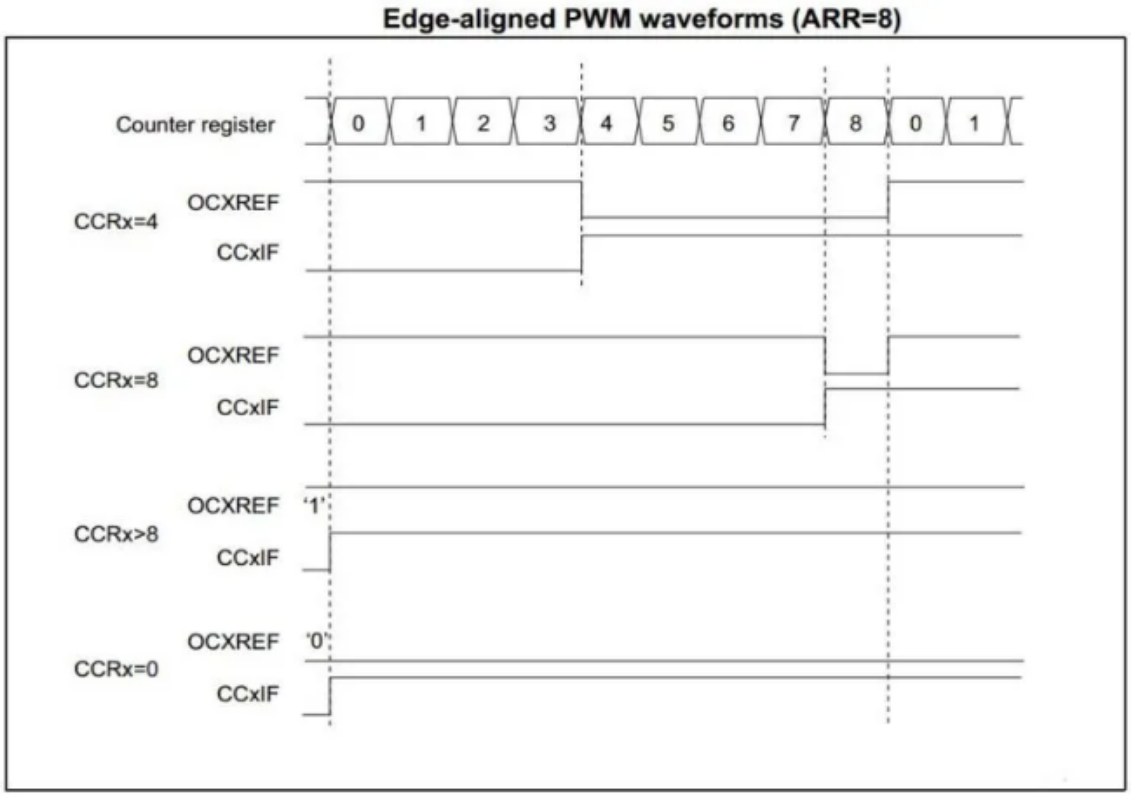
The PWM signal generation can be done in different modes, I'll be discussing two of them in this section. The edge-aligned and the center-aligned modes.

1- Edge-Aligned Mode

In the edge-aligned PWM mode there exist a couple of possible configurations:

- Up-Counting Configuration
- Down-Counting Configuration

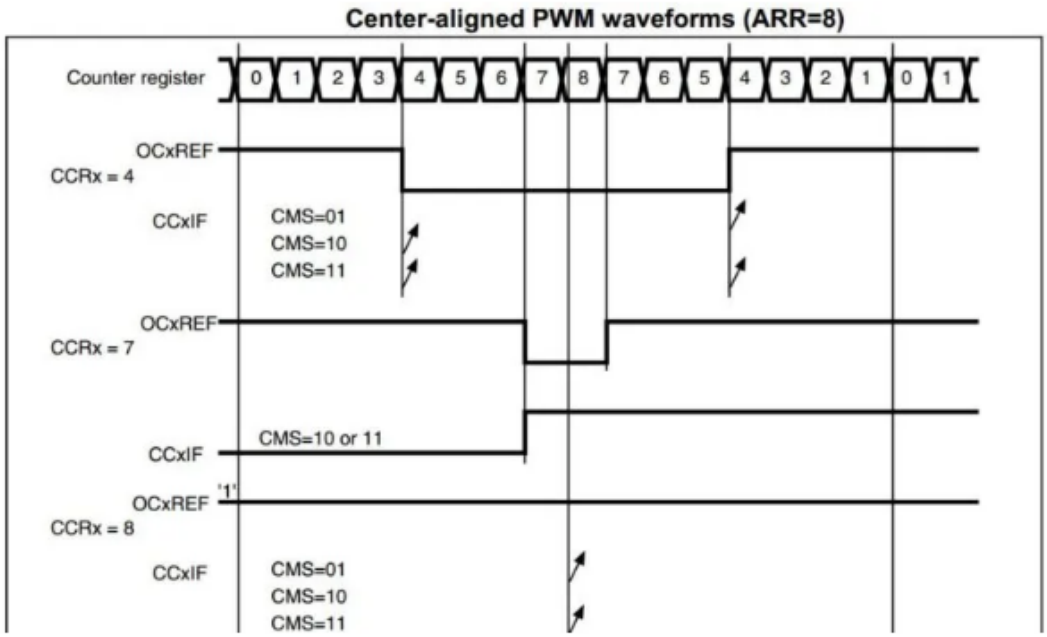
In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as TIMx_CNT <TIMx_CCRx else it becomes low. If the compare value in TIMx_CCRx is greater than the auto-reload value (in TIMx_ARR) then OCxREF is held at ‘1. If the compare value is 0 then OCxREF is held at ‘0



Figure(5-5)

2- Center-Aligned Mode

The compare flag is set when the counter counts up when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx_CR1 register is updated by hardware and must not be changed by software.
The diagram below shows some center-aligned PWM waveforms in an example where: TIMx_ARR=8, PWM mode is the PWM mode 1



Figure(5-6)

Chapter 6

TFT LCD

6.0. TFT

132RGB x 162dot 262K Color with Frame Memory Single-Chip TFT Controller/Driver.

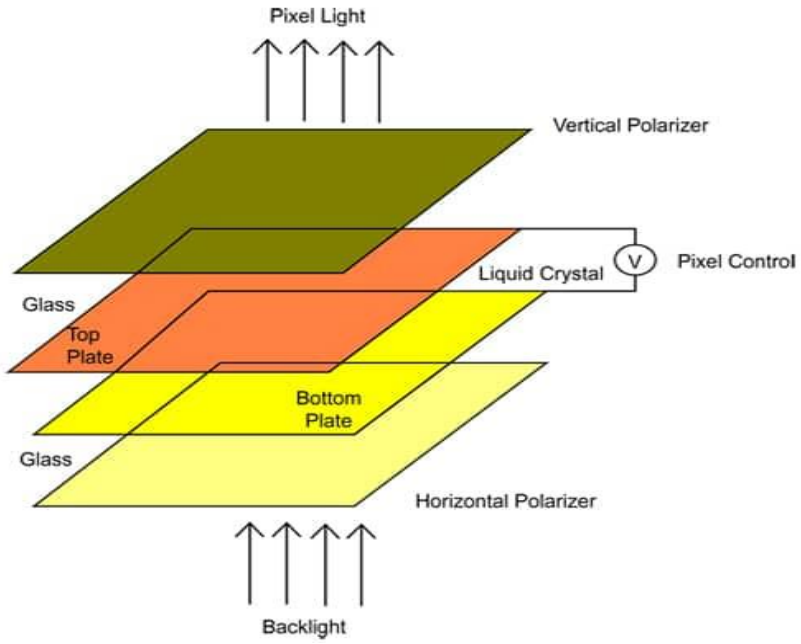
(6.1)

6.1. TFT LCD overview

The ST7735S is a single-chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source line and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI), 8-bit/9-bit/16-bit/18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to minimize power consumption. In addition, because of the integrated power supply circuits necessary to drive liquid crystal, it is possible to make a display system with fewer components.

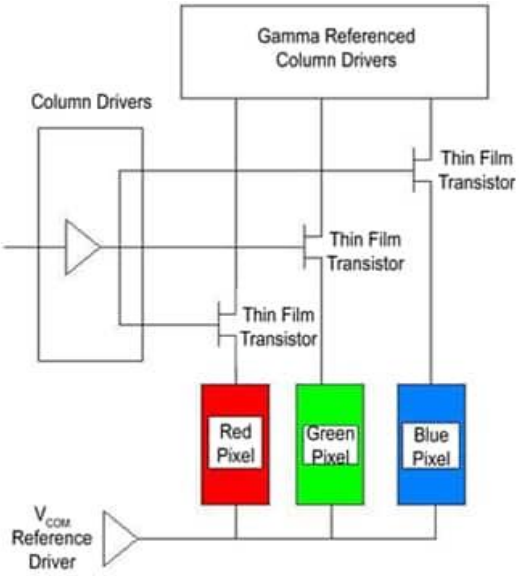
6.2. TFT LCD structure

TFT displays are a type of liquid crystal display in which the transistor controlling the pixel's crystal is etched into a layer of amorphous silicon deposited on the glass (see Figure 1). As in an IC process, very small transistors are geometrically formed. The small size of the transistor means it will not significantly attenuate the light passing through.



(6.2)

The advantage of TFTs is that they are fast enough for video, provide a large and smooth color palette, and are pixel addressable through an electronic two-dimensional control matrix (see Figure 2). Most low-cost displays use an amorphous silicon crystal layer deposited onto the glass through a plasma-enhanced chemical vapor deposition.



6.3. TFT LCD Features

- Single Chip TFT-LCD Controller/Driver with RAM On-chip Display Data RAM 132 (H) x RGB x 162 (V) Bits

- LCD Driver Output Circuits :
  Source Outputs: 132 RGB Channels
  Gate Outputs: 162
  Channels Common Electrode Output

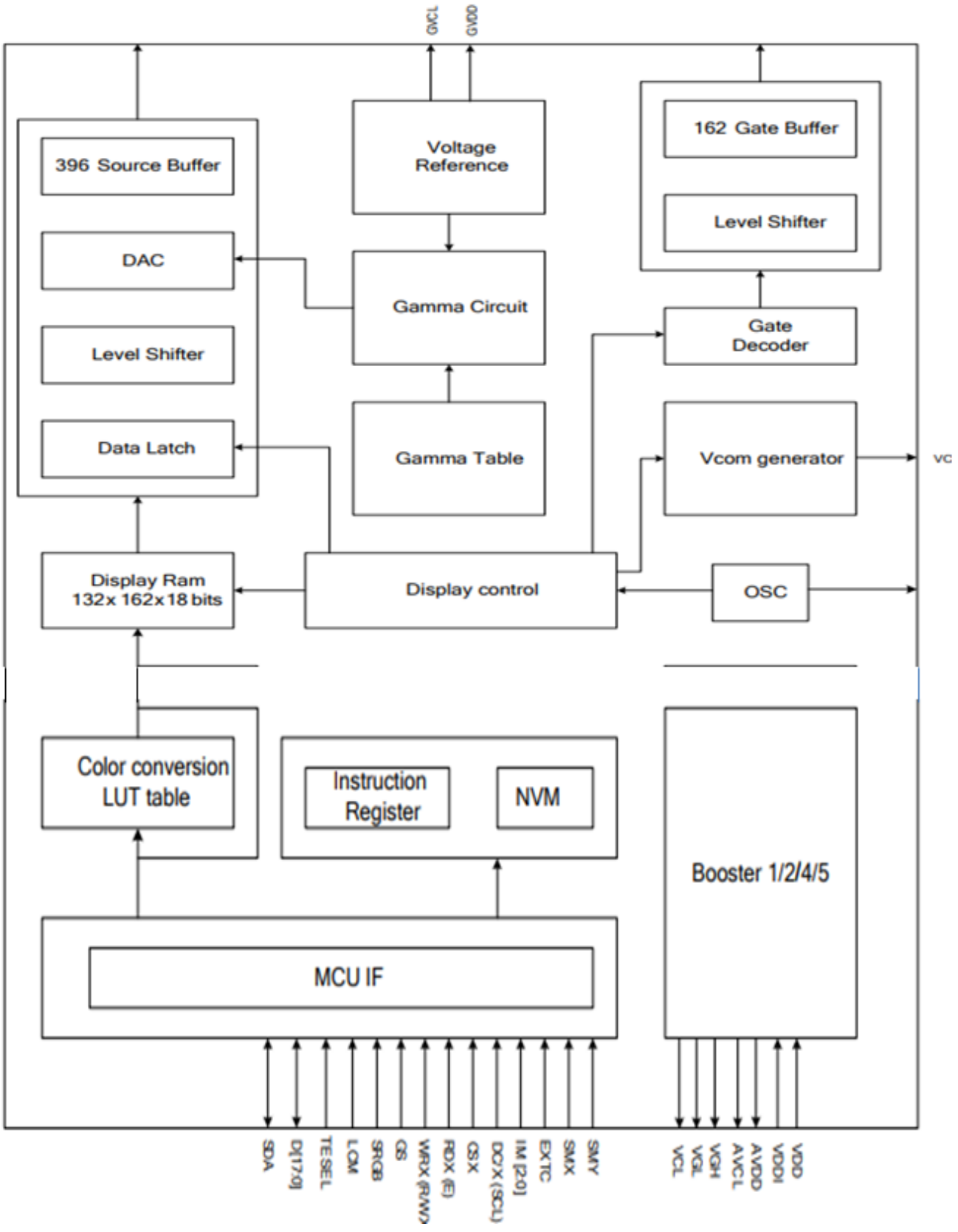
- Display Colors (Color Mode)
  Full Color: 262K, RGB=(666) Max., Idle Mode OFF
  Color Reduce: 8-color, RGB=(111), Idle Mode ON

- Programmable Pixel Color Format (Color Depth) for Various Display Data input Format
  12-bit/pixel: RGB=(444) Using the 384k-bit Frame Memory and LUT
  16-bit/pixel: RGB=(565) Using the 384k-bit Frame Memory and LUT
  18-bit/pixel: RGB=(666) Using the 384k-bit Frame Memory and LUT

- Various Interfaces
  Parallel 8080-series MCU Interface (8-bit, 9-bit, 16-bit & 18-bit)
  Parallel 6800-series MCU Interface (8-bit, 9-bit, 16-bit & 18-bit)
  3-line Serial Interface 4-line Serial Interface

- Display Features
  Support Both Normal-black & Normal-white LC
  Software Programmable Color Depth Mode
  Partial Window Moving & Data Scrolling

6.4. TFT LCD Block Diagram



(6.3)

6.5. TFT LCD Pins and connections.

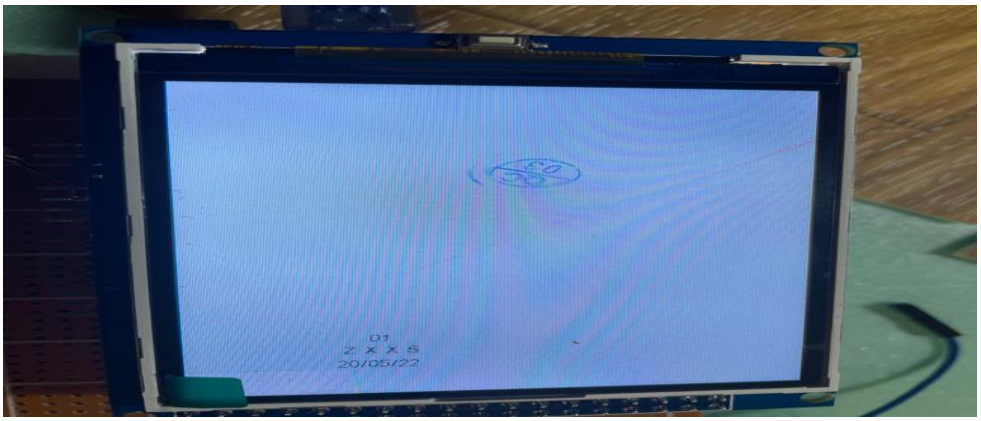
(6.4)

The LCD'S display controller is a ST7735S chip with a block diagram in side as shown in figure 2 and has the model name used in programming which is GRENTAB80x160 that operate through the SPI bus and has display area size (Width and height of the display) 0.108x0.21696 centimeters with 8 external pins as shown in Figure 3 which uses 4 pins to control the operation, 4 pins are SCL (SCK) ,SDA(MOSI) , DC and CS and details of all 8 pins are as follows:

a. GND for connecting to the ground
b. VCC for connecting 3.3 VDC voltage
c. SCL for connecting to the SCK pin of the SPI bus
a. SDA connecting with MOSI pin of SPI
b. RES connecting a reset signal to allow the display module to reset itself
c. DC used to indicate the type of data being sent in SDA, 1 for data and 0 for command
d. CS for connecting with the control pin to determine the module to work or not
e. BLK for connecting to back light

6.6. Programming the Display

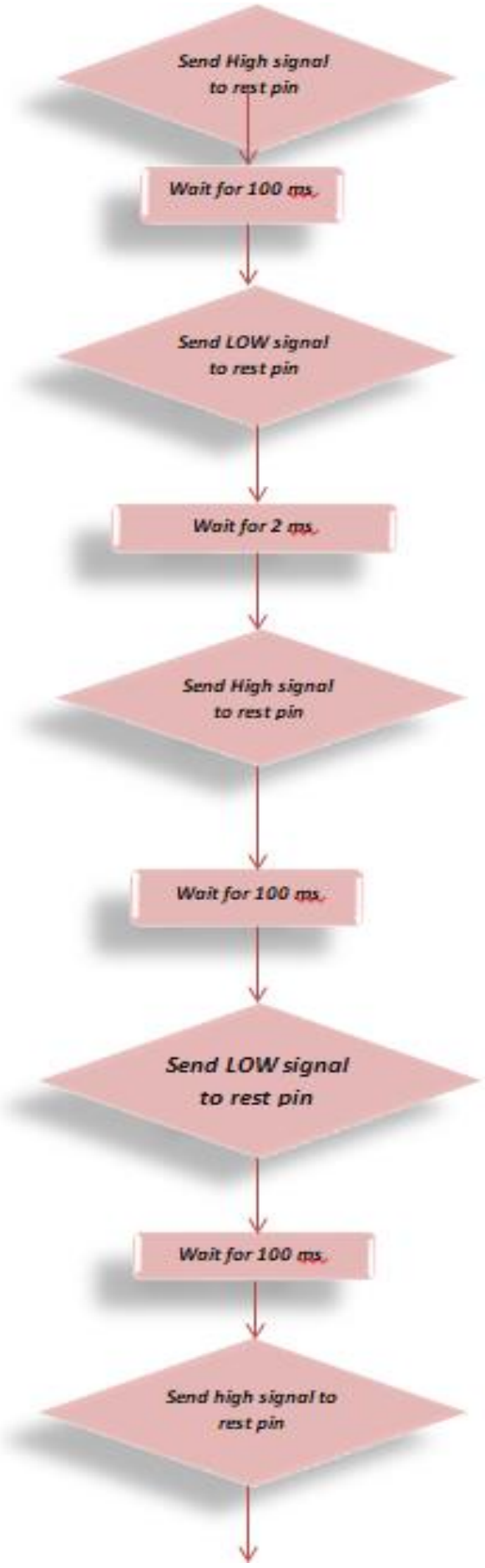
When the ST7735s first powers on it should show a uniform bright white color as shown in figure 5, but that’s just the backlight LEDs, The display will not try to show anything at all until it is initialized , Be aware that a broken display might still show a bright white screen when power is applied.

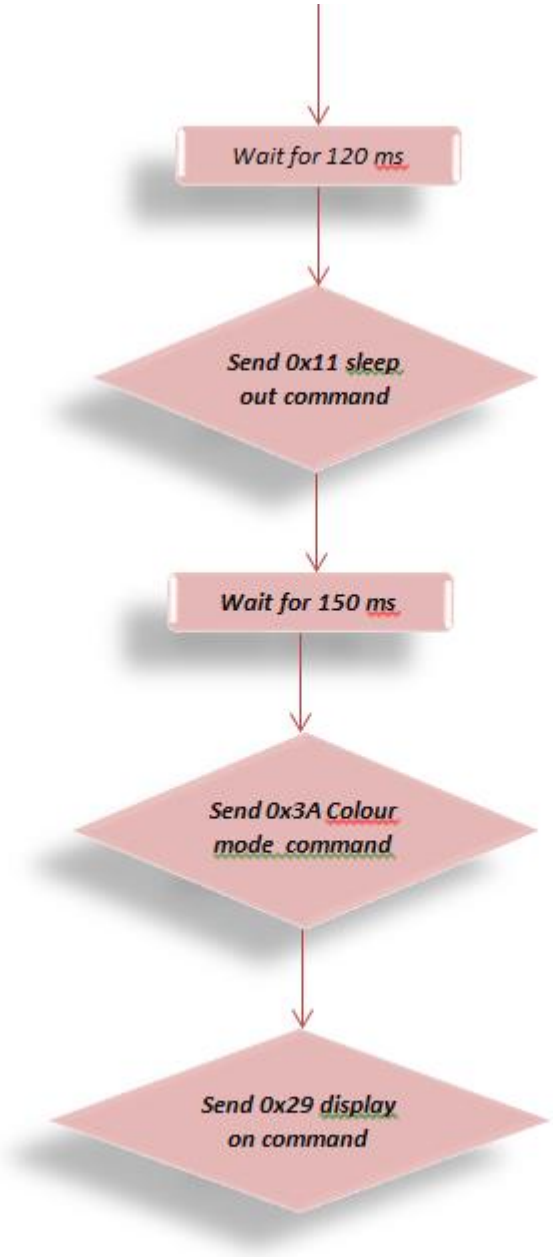


(6.5)

To start the display and put it into a state where it can draw things, we need to send it a series of startup commands.

6.6.1 Initialization sequence



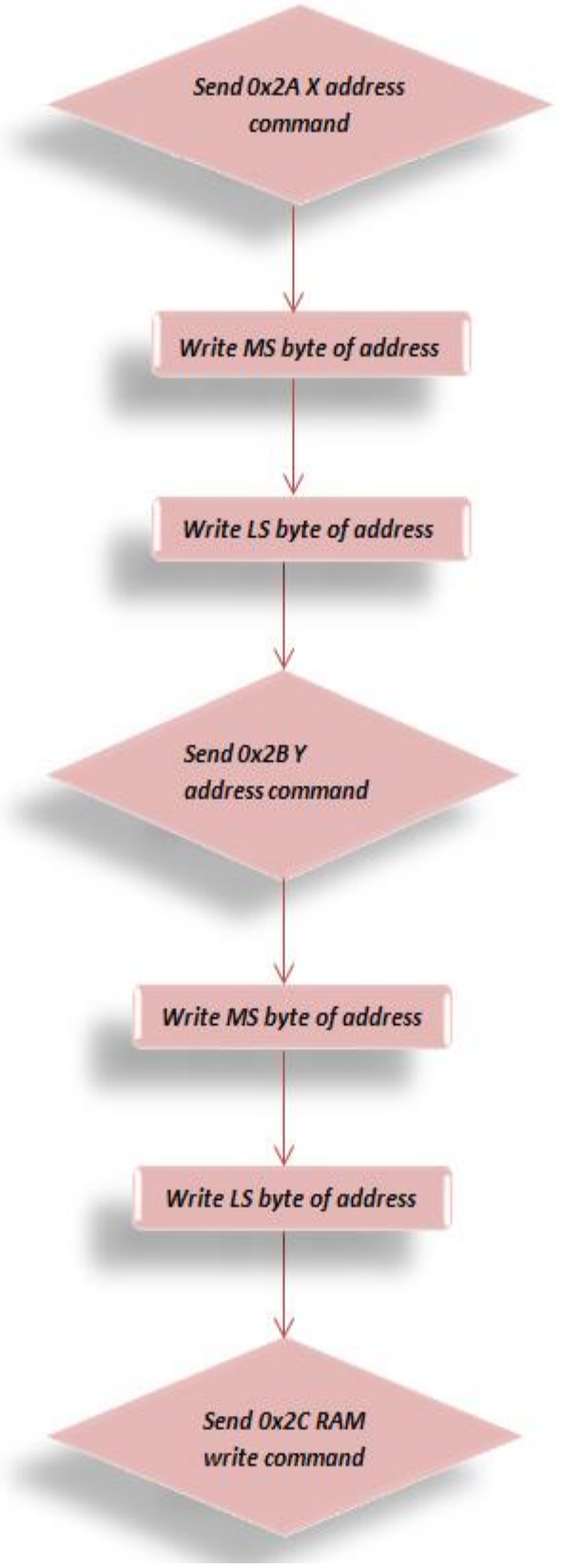


After the display is reset and those commands are sent, the display should change to a flickering grey color.

That tells you that the display is all set up and ready to go, but it has not received any pixel data yet so it is not showing any colors.

6.6.2. Setting address to detect the area the we write in

To refresh the entire 128 x 160 display, we can set the drawing area to be between (0, 0) and (128, 160) and then draw 128 * 160 pixels of data, That's a lot of data - even at one bit per pixel, the small chips used in this module would not have enough RAM to store a full frame buffer and the following diagram will show how to set x and y address:



Chapter 7

Communication Protocols

1-Universal synchronous asynchronous receiver transmitter (USART)

Used in our project to manage MCU to communicate with esp wifi module .

7.1.0 Introduction

There are actually two forms of UART hardware as follows:

- **UART** – Universal Asynchronous Receiver/Transmitter
- **USART** – Universal Synchronous/Asynchronous Receiver/Transmitter

The **Synchronous** type of transmitters generates the data clock and sends it to the receiver which works accordingly in a synchronized manner. On the other hand, the **Asynchronous** type of transmitter generates the data clock internally. There is no incoming serial clock signal, so in order to achieve proper communication between the two ends, both of them must be using the same **baud rate**.

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator. It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication. High speed data communication is possible by using the DMA for multibuffer configuration.

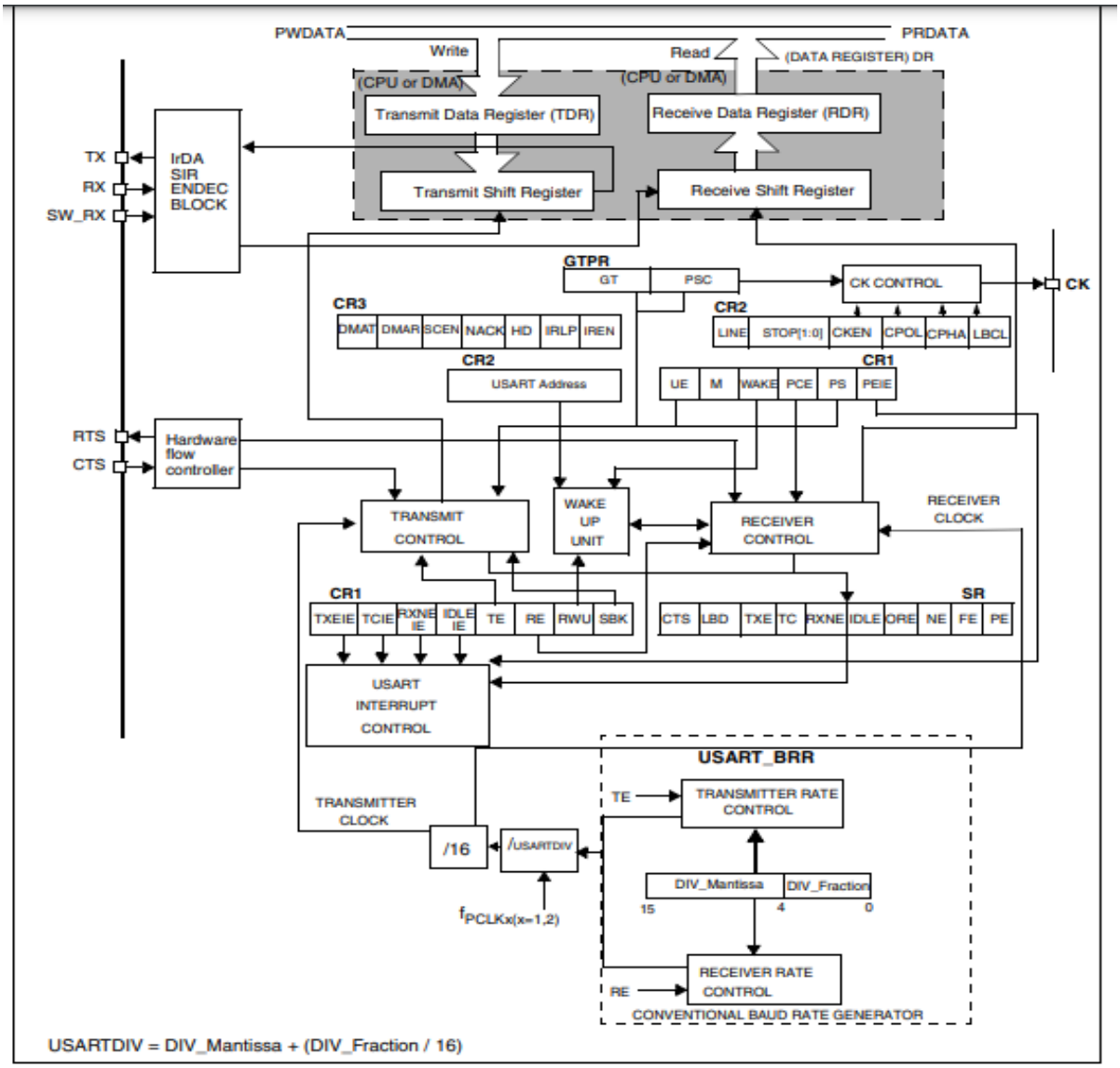
7.1.1 Features

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- Fractional baud rate generator systems
- A common programmable transmit and receive baud rates up to 4.5 MBits/s
- Programmable data word length (8 or 9 bits)
- Configurable stop bits
- support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
- 13-bit break generation and 10/11 bit break detection when USART is hardware configured for LIN
- Support for 3/16 bit duration for normal mode
- 0.5, 1.5 Stop Bits for Smartcard operation
- Single wire half duplex communication
- Configurable multibuffer communication using DMA (direct memory access)
  -Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for Transmitter and Receiver
- Transfer detection flags:
  – Receive buffer full
  – Transmit buffer empty
  – End of Transmission flags
- Parity control:
  – Transmits parity bit
  – Checks parity of received data byte
- Four error detection flags:
  – Overrun error
  – Noise error
  – Frame error

– Parity error

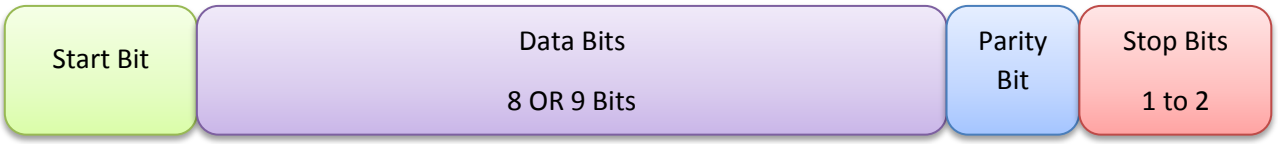
- Ten interrupt sources with flags:
  – CTS change
  – LIN break detection
  – Transmit data register empty
  – Transmission complete
  – Receive data register full
  – Idle line received
  – Overrun error
  – Framing error
  – Noise error
  – Parity error
- Two receiver wakeup modes:
  Address bit (MSB, 9th bit), Idle line

7.1.2 USART Block diagram



Figure(7-1-1)

7.1.3 USART Frame Format



Figure(7-1-2)

Start Bit

The UART data transmission line (TX) is normally held High (1) logic-level when there is no data transmission on the line. To start data transfer, the transmitting UART pulls-down the transmission line from high to low for one clock cycle. When the receiving UART module detects the high to low voltage transition, it begins reading the incoming bits of the entire data frame at the same frequency of the specified baud rate.

Data Bits

These are the actual data bits being transmitted from transmitter to receiver. The length of the data frame can be anywhere between 8 and 9 (9-Bits if parity is not used and 8-Bits if parity is used). In general settings, the LSB is the first bit to be shifted-out, transmitted, (unless otherwise specified).

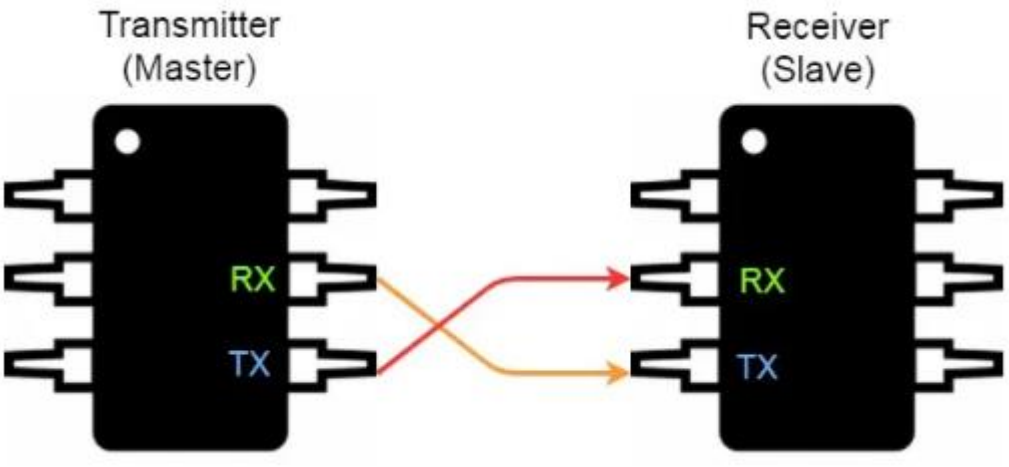
Parity Bit

Parity bit allows the receiver to check the correctness of the received data. Parity is a low–level error checking mechanism and has two different options to choose from: Even Parity and Odd Parity. After the receiver UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total value is an even or odd number. If the parity bit is a 0 (**even parity**), the number of bits with a value of 1 in the data frame should sum-up to an even number. If the parity bit is a 1 (**odd parity**), the number of bits with a value of 0 in the data frame should sum-up to an odd number. When the parity bit matches the data, the receiver UART knows that the transmission was a success. Frankly speaking, the parity bit is optional and it's actually not widely used.

Stop Bit

As the name suggests, this bit is used to signal the end of the data packet being sent. The sending UART drives the data transmission line from low to high for at least two-bit durations but most often only one bit is used. The data line is held high back to the idle state and ready for any future transmission.

7.1.3 Modes of Operation for UART Devices



Figure(7-1-3)

The UART serial communication bus can only have 2 devices communicating with each other in one of the 3 modes shown in the previous sub-section. Which necessarily means, each device can either be a transmitter or a receiver during the data transmission process.

- **Transmitter**
- **Receiver**

You should know that the transmitter in synchronous mode must generate the serial data clock for the receiver. On the other hand, in asynchronous mode, there is no need to do so.
And there can't be more than 2 devices on the UART serial bus. Otherwise, there will be some issues introduced to the system which you'll have to work around.
The proper connection for any 2 devices for UART serial communication goes as follows: the transmitter's TX goes to the receiver's RX and the receiver's TX goes to the transmitter's RX. Frankly speaking, the 2-wires are basically crossed!

7.1.4 Transmitting Operation

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

Character transmission

During a USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register . Every character is preceded by a start bit, which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits. The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

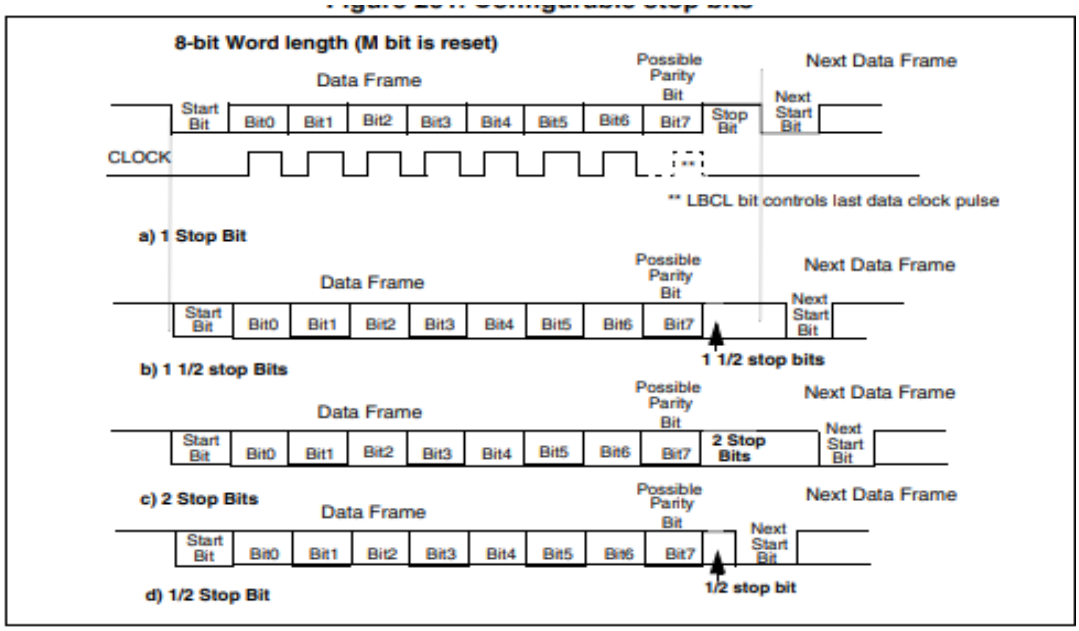
Note: The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost. An idle frame will be sent after the TE bit is enabled.

Configurable stop bits

The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

1. **1 stop bit**: This is the default value.
2. **2 stop bits**: This is supported by normal USART, single-wire and modem modes.
3. **0.5 stop bit**: To be used when receiving data in Smartcard mode.
4. **1.5 stop bits**: To be used when transmitting and receiving data in Smartcard mode.

An idle frame transmission will include the stop bits. A break transmission will be 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).



Figure(7-1-4) Configurable Stop Bits

USART Data Transition steps (Procedure):

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place.

   Configure the DMA register as explained in multibuffer communication.
5. Select the desired baud rate using the USART_BRR register.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_DR register (this clears the TXE bit).

Repeat this for each data to be transmitted in case of single buffer.

8. After writing the last data into the USART_DR register, wait until TC=1.
   This indicates that the transmission of the last frame is complete.
   This is required for instance when the USART is disabled or enters the Halt mode
   to avoid corrupting the last transmission.

7.1.5 Receiving Operation

The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Character reception

During a USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

USART Receive steps (Procedure):

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place.
   Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received

• The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR.

In other words, data has been received and can be read (as well as its associated error flags).

• An interrupt is generated if the RXNEIE bit is set.

• The error flags can be set if a frame error, noise or an overrun error has been detected during reception.

• In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.

• In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register.

The RXNE flag can also be cleared by writing a zero to it.

The RXNE bit must be cleared before the end of the reception of the next character to avoid an

overrun error.

Note: The RE bit should not be reset while receiving data. If the RE bit is disabled during reception,

the reception of the current byte will be aborted.

7.1.6 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$Tx/\ Rx\ baud\ =\ fck\ /\ fck(16*USARTDIV)$$

legend: - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1) .

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

Note: The baud counters are updated with the new value of the Baud registers after a write to USART_BRR.

Hence the Baud rate register value should not be changed during communication.

How to derive USARTDIV from USART_BRR register values

Example 1:

If DIV_Mantissa = 0d27 and DIV_Fraction = 0d12 (USART_BRR = 0x1BC), then

Mantissa (USARTDIV) = 0d27

Fraction (USARTDIV) = 12/16 = 0d0.75

Therefore USARTDIV = 0d27.75

Tx/ Rx baud = legend: fCK - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

Example 2:

To program USARTDIV = 0d25.62

This leads to:

DIV_Fraction = 16*0d0.62 = 0d9.92 T

he nearest real number is 0d10 = 0xA

DIV_Mantissa = mantissa (0d25.620) = 0d25 = 0x19

Then, USART_BRR = 0x19A hence USARTDIV = 0d25.625

Example 3:

To program USARTDIV = 0d50.99

This leads to: DIV_Fraction = 16*0d0.99 = 0d15.84

The nearest real number is 0d16 = 0x10 => overflow of DIV_frac[3:0] => carry must be added up to the mantissa

DIV_Mantissa = mantissa (0d50.990 + carry) = 0d51 = 0x33

Then, USART_BRR = 0x330 hence USARTDIV = 0d51.000

2- Serial peripheral interface(SPI)

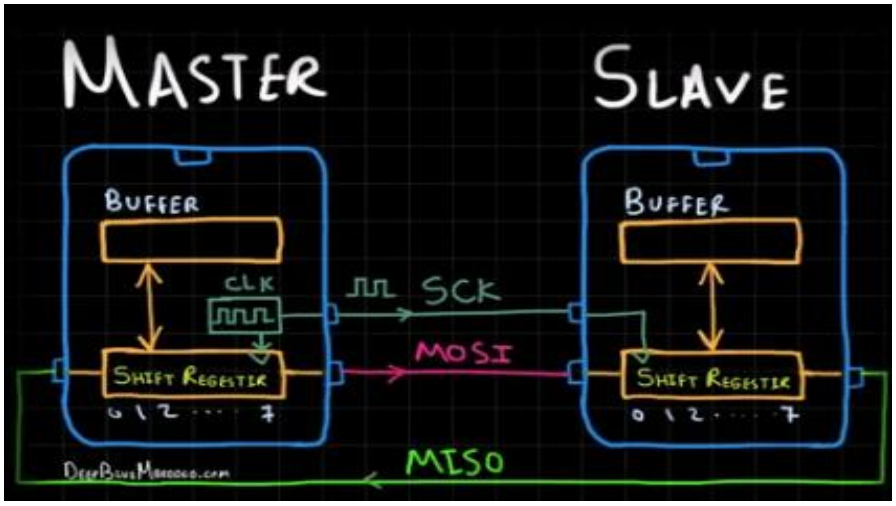
Used in our project to manage MCU to communicate with TFT module .

7.2.0 Introduction

SPI is an acronym for (Serial Peripheral Interface) pronounced as “S-P-I” or “Spy”. Which is an interface bus typically used for serial communication between microcomputer systems and other devices, memories, and sensors. Usually used to interface Flash Memories, ADC, DAC, RTC, LCD, SDcards, and much more. The SPI was originally developed by Motorola back in the 80s to provide full-duplex serial communication to be used in embedded systems applications

7.2.1 SPI pin configurations and connections

In typical SPI communication, there should be at least 2 devices attached to the SPI bus. One of them should be the master and the other will essentially be a slave. The master initiates communication by generating a serial clock signal to shift a data frame out, at the same time serial data is being shifted-in to the master. This process is the same whether it’s a read or write operation

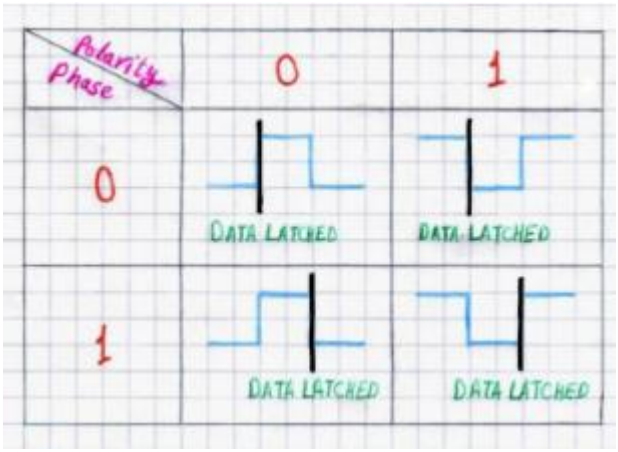


Figure(7-2-1)

- **MOSI** -> Master output slave input (D_{OUT} From Master)
- **MISO** -> Master input slave output (D_{OUT} From Slave)
- **SCLK** -> Serial Clock, generated by the master and goes to the slave.
- **SS** -> Slave Select. Generated by the master to control which slave to talk to.
  - It’s usually an active-low signal.

7.2.2 SPI Modes of Operation

Devices on the SPI bus can operate in either mode of the following: **Master** or **Slave**. There must be at least one master who initiates the serial communication process (For Reading/Writing). On the other hand, there can be single or multiple devices operating in slave mode.
The master device can select which slave to talk to by setting the **SS** (slave select) pin to logic low. If a single slave is being addressed, you can tie the SS pin of this slave device to logic low without the need to control this line by the master.



Figure(7-2-2)

7.2.3 STM32 SPI Hardware Overview

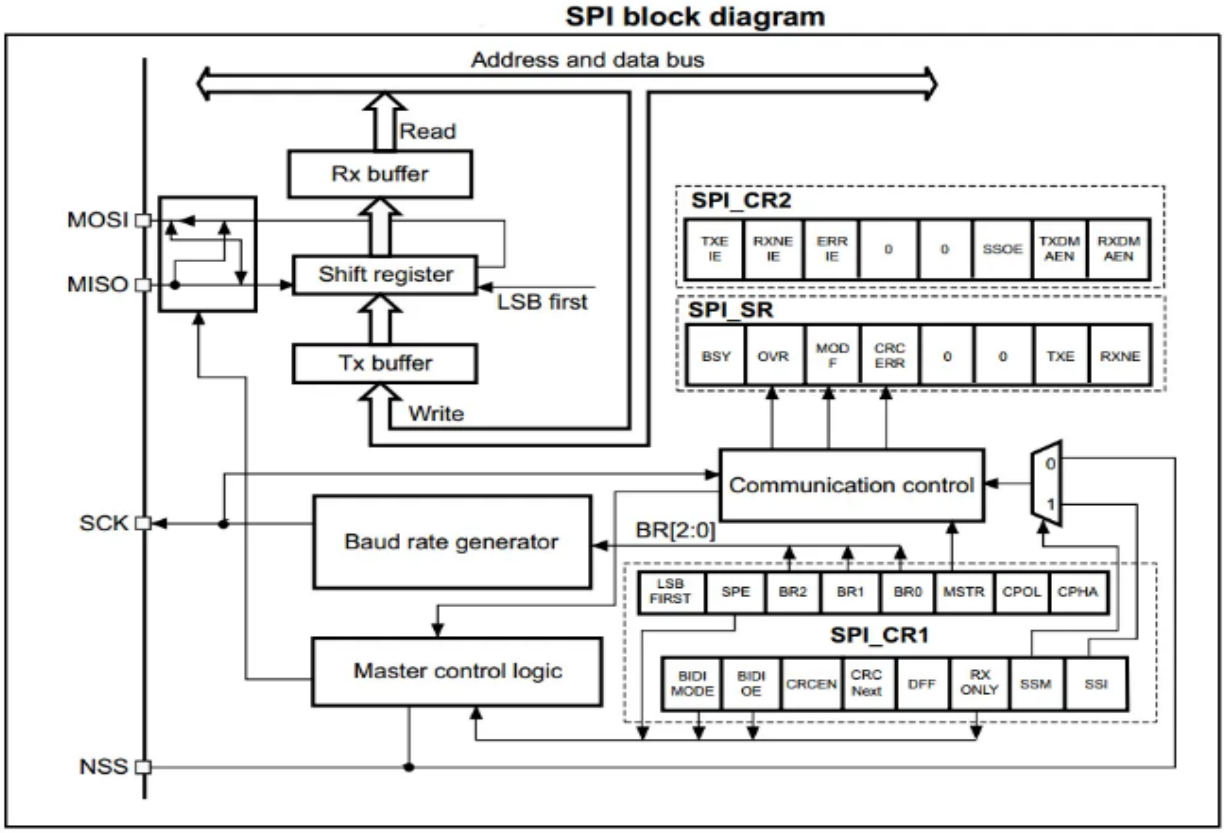
the STM32 SPI interface provides two main functions, supporting either the SPI protocol or the I2S audio protocol. By default, it is the SPI function that is selected. It is possible to switch the interface from SPI to I2S by software.
The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case, it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in a multi-master configuration

7.2.4 STM32 SPI Main Features

- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation

- Multimaster mode capability
- 8 master mode baud rate prescalers (fPCLK/2 max.)
- Slave mode frequency (fPCLK/2 max)
- NSS pin management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Hardware CRC feature for reliable communication: [CRC value can be transmitted as the last byte in Tx mode – Automatic CRC error checking for last received byte]
- Master mode fault, overrun, and CRC error flags with interrupt capability
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

7.2.5 STM32 SPI Block Diagram



Figure(7-2-3)

As you can see in the SPI block diagram above, there is the main shift register lying between two buffer registers one for transmission (TX) and the other for reception (RX). And a logical control unit on the right side with a bunch of signals coming in and out to the control registers.

The control registers give you the ability to change any of the SPI hardware configurations by software instructions. And there is also the status register that gives you some signals about the ongoing and last-done transaction and any error if happened.
The baud rate generator (Clock signal SCK) is also configurable by software as we can tell and there is a control logic dedicated for the NSS pin (if you're going to use it).

7.2.6. STM32 SPI Data Frame Format

Data can be shifted out either MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 Register.
Each data frame is 8 or 16 bits long depending on the size of the data programmed using the DFF bit in the SPI_CR1 register. The selected data frame format is applicable for transmission and/or reception.
It has to be the same on both ends of the communication. The slave should know what to expect in order to read the received data correctly. It has to be formatted coordinately

7.2.7. STM32 SPI In Slave Mode

In the slave configuration, the serial clock is received on the SCK pin from the master device. The baud rate value set in the BR[2:0] bits in the SPI_CR1 register, does not affect the data transfer rate. In this configuration, the MOSI pin is a data input pin and the MISO pin is a data output.

Note that you need to have the SPI slave device running and configured before the master starts to send anything just to avoid any possible data corruption at the beginning of the first communication session.

7.2.8 Transmit Sequence in Slave Mode

The data byte is parallel-loaded into the Tx buffer during a write cycle. The transmit sequence begins when the slave device receives the clock signal and the most significant bit of the data on its MOSI pin. The remaining bits (the 7 bits in 8-bit data frame format, and the 15 bits in 16-bit data frame format) are loaded into the shift-register. The TXE flag in the SPI_SR register is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

7.2.9 Receive Sequence in Slave Mode

For the receiver, when data transfer is complete: The Data in the shift register is transferred to Rx Buffer and the RXNE flag (SPI_SR register) is set. An Interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register.
After the last sampling clock edge, the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value. Clearing of the RXNE bit is performed by reading the SPI_DR register.

7.2.10 STM32 SPI In Slave Mode

In the slave configuration, the serial clock is received on the SCK pin from the master device. The baud rate value set in the BR[2:0] bits in the SPI_CR1 register, does not affect the data transfer rate. In this configuration, the MOSI pin is a data input pin and the MISO pin is a data output.

Note that you need to have the SPI slave device running and configured before the master starts to send anything just to avoid any possible data corruption at the beginning of the first communication session.

Transmit Sequence in Slave Mode

The data byte is parallel-loaded into the Tx buffer during a write cycle. The transmit sequence begins when the slave device receives the clock signal and the most significant bit of the data on its MOSI pin. The remaining bits (the 7 bits in 8-bit data frame format, and the 15 bits in 16-bit data frame format) are loaded into the shift-register. The TXE flag in the SPI_SR register is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

Receive Sequence in Slave Mode

For the receiver, when data transfer is complete: The Data in the shift register is transferred to Rx Buffer and the RXNE flag (SPI_SR register) is set. An Interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register.
After the last sampling clock edge, the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value. Clearing of the RXNE bit is performed by reading the SPI_DR register.

7.2.11 STM32 SPI In Master Mode

In the master configuration, the serial clock is generated on the SCK pin. In this configuration, the MOSI pin is a data output pin and the MISO pin is data input.

Transmit Sequence in Master Mode

The transmit sequence begins when a byte is written in the Tx Buffer. The data byte is parallel-loaded into the shift register (from the internal bus) during the first-bit transmission and then shifted out serially to the MOSI pin MSB first or LSB first depending on the LSBFIRST bit in the SPI_CR1 register.
The TXE flag is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

Receive Sequence in Master Mode

For the receiver, when data transfer is complete: The data in the shift register is transferred to the RX Buffer and the RXNE flag is set. An interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register
At the last sampling clock edge, the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value.
Clearing the RXNE bit is performed by reading the SPI_DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in the Tx buffer once the transmission is started. Note that the TXE flag should be ‘1 before any attempt to write the Tx buffer is made.

7.2.12 STM32 SPI Data Transmission & Reception

In reception, data are received and then stored into an internal Rx buffer while In transmission, data are first stored into an internal Tx buffer before being transmitted.

A read access to the SPI_DR register returns the Rx buffered value, whereas a write access to the SPI_DR stores the written data into the Tx buffer.

7.2.13 STM32 SPI Flags

There are some status flags provided for the application to completely monitor the state of the SPI bus.

- **Tx buffer empty flag (TXE)** – When it is set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can be loaded into the buffer. The TXE flag is cleared when writing to the SPI_DR register.
- **Rx buffer not empty (RXNE)** – When set, this flag indicates that there are valid received data in the Rx buffer. It is cleared when SPI_DR is read.
- **BUSY flag** – The BSY flag is useful to detect the end of a transfer if the software wants to disable the SPI and enter Halt mode (or disable the peripheral clock). This avoids corrupting the last transfer. For this, the procedure described below must be strictly respected. The BSY flag is also useful to avoid write collisions in a multi-master system.
- There are also other SPI flags that indicate whether a specific type of error has occurred or not.
- **SPI Master mode fault (MODF) –** Master mode fault occurs when the master device has its NSS pin pulled low (in NSS hardware mode) or SSI bit low (in NSS software mode), this automatically sets the MODF bit.
- **SPI Overrun condition –** An overrun condition occurs when the master device has sent data bytes and the slave device has not cleared the RXNE bit resulting from the previous data byte transmitted.
- **SPI CRC error –** This flag is used to verify the validity of the value received when the CRCEN bit in the SPI_CR1 register is set. The CRCERR flag in the SPI_SR register is set if the value received in the shift register does not match the receiver SPI_RXCRCR value.

7.2.14 STM32 SPI Interrupt

The SPI interrupt events are connected to the same interrupt vector. So the SPI fires a single interrupt signal regardless of the source of it. The software will have to detect it. These events generate an interrupt if the corresponding Enable Control Bit is set.

SPI interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
CRC error flag	CRCERR	

Figure(7-2-4)

Chapter 8
Conclusion

8.1. Future Work

Smart EV charging or intelligent charging refers to a system where an electric vehicle and a charging device share a data connection, and the charging device shares a data connection with a charging operator.

As opposed to traditional (or dumb) charging devices that aren't connected to the cloud, smart charging allows the charging station owner to monitor, manage, and restrict the use of their devices remotely to optimize energy consumption.

8.1.1. Protection

8.1.1.1. Adding EMI Filter

We Can Improve this Project by adding EMI Filter before Rectifier Electronic devices in different fields mainly work through electricity but sometimes, the performance of these devices may interrupt or they will break down or crash when an Electromagnetic interference (EMI) occurs. So, based on the severe interruption, Electromagnetic interference may impact the signal quality of the device received, so the component or device may get damaged permanently. So to protect these devices from damage, an **EMI filter** is used.

8.1.1.2. Adding Boost PFC

Boost power factor correction (PFC) converter can significantly improve both the power factor and the harmonic distortion.

8.1.1.3. Adding CFGI

It works by comparing the amount of current going to and returning from equipment along the circuit conductors. When the amount going differs from the amount returning by approximately 5 milli amperes, the GFCI interrupts the current.

8.1.2. Software

8.1.2.1. Building our web application

We will build our web application from scratch instead of using a Framework like ThingSpeak that we use it, this will improve our skills in the software field.

8.1.2.2. Embedded Linux

We will use Embedded Linux to build our GUI, and will build our custom image of Linux that will meet memory constrain and provide to us the minimum feature we will use it

This will improve our skills in Embedded Software field.

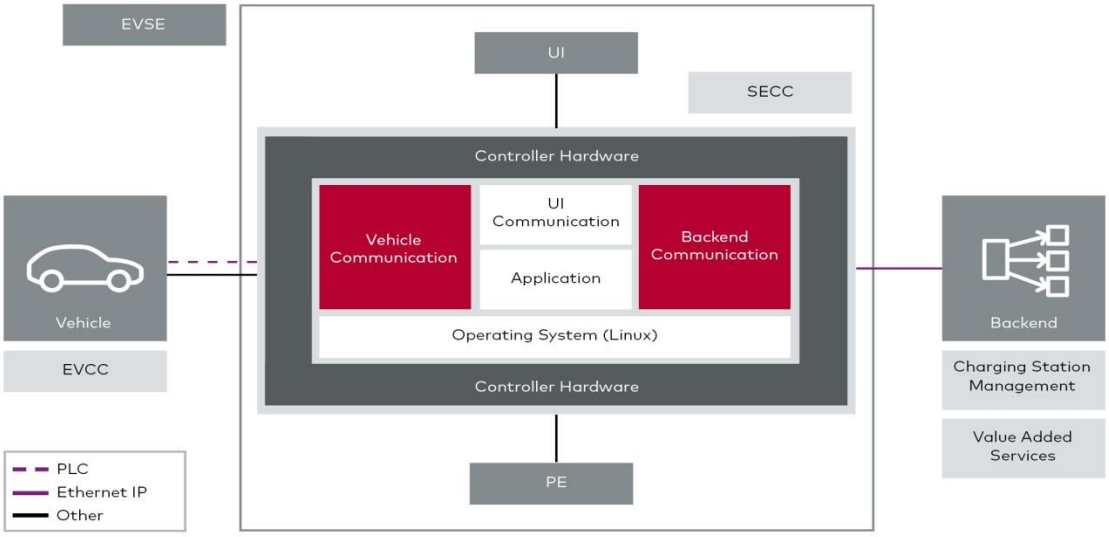


Figure (8-1) Smart Charger Stack

8.1.2.3. AUTOSAR

We will improve the software of the system by applying the AUTOSAR rules on the code

This will make our code more configurable, and will motivate us to learn the AUTOSAR methodology

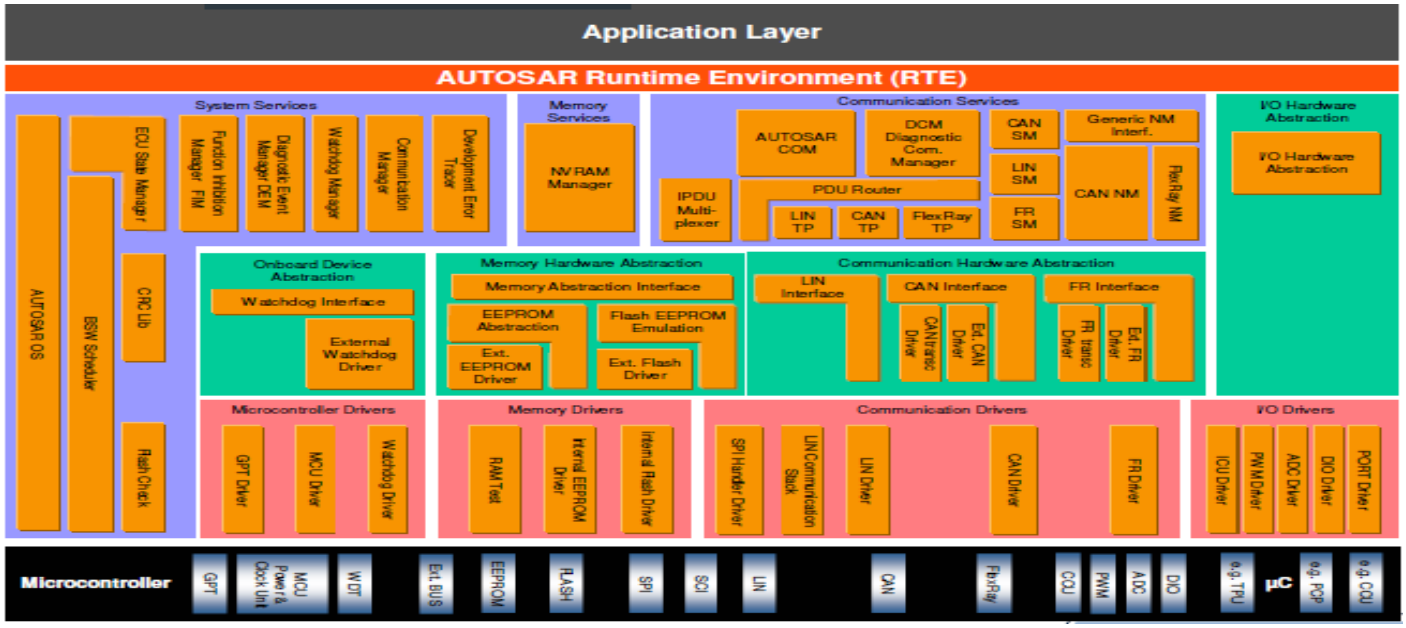


Figure (8-2) Autosar layer architecture

8.1.3. Switching

We can add more type of DC DC Converter to provide different voltage as Buck Converter or Buck-Boost Converter to controlling the voltage.

Code

You can find the project full repository here:

https://github.com/AsmaaHashim/WallBox-Charger

References

[1] STM32F103x Data Sheet.

[2] EV Charging Ali Bahrami Book

[3] Hand Book for EV charging Infrastructure Implementation

[4] https://www.virta.global/smart-charging

[5] https://electronicsworkshops.com/

[6] https://www.avnet.com/wps/portal/abacus

[6] https://www.electronicproducts.com/

[7] Texas Instrument

[8] ARM Course by Ahmed Aldeep and Ahmed Assaf

[9] https://www.electrical4u.com/

[10] https://www.allaboutcircuits.com/