ESP HOPPER LEVEL SENSING SYSTEM WITH OPC-UA INTERFACE

A PROJECT REPORT

Submitted by

S. SANJAY (511319106027)

JOGI DILEEP KUMAR REDDY (511319106017)

C. SUBASH (511319106030)

C. SUBASH (511319106029)

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING,

KINGSTON ENGINEERING COLLEGE

ANNA UNIVERSITY: CHENNAI 600025

MAY 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"ESP HOPPER LEVEL SENSING SYSTEM WITH OPC-UA INTERFACE",** is the bonafide work of

"SANJAY S (511319106027),
JOGI DILEEP KUMAR REDDY (511319106017),
SUBASH C (511319106030),
SUBASH C (511319106029)"

who carried out the project under my supervision.

SIGNATURE	**SIGNATURE**
Mrs. M RADHIKA, M.E., (Ph.D),	**MR.G SATHISHKUMAR, M.E.,**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Dept. of Electronics & Communication Engineering	Dept. of Electronics & Communication Engineering,
Kingston Engineering College Vellore-632059.	Kingston Engineering College, Vellore-632059.

Submitted for the Anna University Practical Examination held on________at

Kingston Engineering College, Chittoor Main Road, Vellore-632059

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

With profound gratitude, respect and pride we express our sincere thanks to our Chairperson **Thiru. KATHIR ANAND.D.M, M.B.A. (USA)** for having granted us permission to do the individual attachment project.

We are indebted to extend our sincere thanks to the principal **Dr.ARIVAZHAGU.U. V M.E., Ph.D.,** for her unstinted support and encouragement to take up the particular program.

We are also grateful to the Head of the department **Mrs. M RADHIKA, M.E., (Ph.D.),** for his constructive suggestions and encouragement during our project.

We take immense pleasure in thanking **Mr. V. S. Suresh Kumar**, Additional General Manager, EDC-Embedded Systems and **Mr. D. Kalyana Sundaram**, Deputy Engineer/HRDC for facilitating us to do our project.

We extend our gratitude to our Project guide **Mr. K. Murali**, Manager, EDC-Embedded Systems for his patience, motivation and immense knowledge in guiding us throughout the duration of our project.

We are also grateful to our project guide **MR. G SATHISHKUMAR, M.E.,** Assistant Professor, Department of Electronic & Communication Engineering who supported and gave us valuable advice and innovative ideas.

Last but not least our sincere thanks to one and all in the college, Governing and our staff members for providing excellent facilities and support.

ABSTRACT

This project is a novel idea of monitoring ash level in the ash hoppers in thermal power plant. Poor maintenance of ash handling system of furnace bottom ash hopper and ESP hopper will result in the accumulation of ash in the hoppers and the design capacity of load bearing structures may be exceeded.

Excessive ash builds up in the furnace bottom hopper can disturb proper combustion of fuel leading to unsafe conditions. Failure of structures are known to have happened when ash reaches abnormal level. Ash level measurement with capacitance type sensor at different levels of the hopper enables automatic shutdown during unsafe conditions.

Solid level measurement in bunkers and hoppers is required in many industries. In a coal fired thermal power plant, pulverized coal is burnt in the boiler furnace. Heat released in the furnace is conveyed to the water and steam circuit. A portion of ash produced after the completion of combustion falls to the bottom ash hopper.

In recent years, the instruments used in power plants have become more intelligent with facility to communicate, configure and control parameters by interfacing with the plant DCS system using communication protocols like OPC UA. The present project aims at implementing OPC UA interface for an ash level indicator.

The ash level indicator system implements a capacitive sensor for measuring ash at the field. The sensor signal from the field is processed in the controller and the data sent over OPC UA by implementing a server over wifi. The client device connects to the device and ash level indication, calibration and level status are viewed at the client display.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. BHEL AND ITS FIELDS OF OPERATION

Bharat Heavy Electricals Limited (BHEL) owned and founded by the Government of India, is an engineering and manufacturing company based in New Delhi, India. Established in 1964, BHEL is India's largest power generation equipment manufacturer.

HISTORY:

BHEL was established in 1964 ushering in the indigenous Heavy Electrical Equipment industry in India. Heavy Electricals (India) Limited was merged with BHEL in 1974. In 1991, BHEL was converted into a public limited company. Over time, it developed the capability to produce a variety of electrical, electronic and mechanical equipment for all sectors, including transmission, transportation, oil and gas and other allied industries. However, the bulk of the revenue of the company is derived from sale of equipment for power generation such as turbines, boilers, etc. As of 2017, BHEL supplied equipment contributed to about 55% of the total installed power generation capacity of India.[3] The company has also supplied thousands of Electric Locomotives to Indian Railway, as well as defence equipment such as the Super Rapid Gun Mount (SRGM) naval guns and Defence Simulators to the Indian Armed Forces.

OPERATIONS:

BHEL is engaged in the design, engineering, manufacturing, construction, testing, commissioning and servicing of a wide range of products, systems and services for the core sectors of the economy, viz. power, transmission, industry, transportation, renewable energy, oil & gas and defence.

It has a network of 17 manufacturing units, 2 repair units, 4 regional offices, 8 service centres, 8 overseas offices, 15 regional centres, 7 joint ventures, and infrastructure allowing it to execute more than 150 projects at sites across India and abroad. The company has established the capability to deliver 20,000 MW p.a. of power equipment to address the growing demand for power generation equipment.

BHEL has retained its market leadership position during 2015-16 with 74% market share in the Power Sector. An improved focus on project execution enabled BHEL record its highest ever commissioning/synchronization of 15059 MW of power plants in domestic and international markets in 2015-16, marking a 59% increase over 2014-15. With the all-time high commissioning of 15000 MW in a single year FY2015-16, BHEL has exceeded 170 GW installed base of power generating equipment.

It also has been exporting its power and industry segment products and services for over 40 years. BHEL's global references are spread across over 76 countries across all the six continents of the world.

<u>INITIATIVES:</u>

BHEL's investment in R&D is amongst the largest in the corporate sector in India.

During the year 2012-13, the company invested about Rs. 1,252 Crore on R&D efforts, which corresponds to nearly 2.50% of the turnover of the company, focusing on new product and system developments and improvements in existing products. The IPR (Intellectual Property Rights) capital of BHEL grew by 21.5% in the year, taking the total to 2170.

The Corporate R&D division at Hyderabad leads BHEL's research efforts in a number of areas of importance to BHEL's product range. Research & product development (RPD) Groups for each product group at the manufacturing divisions play a complementary role. BHEL has established Centres of Excellence for Simulators, Computational Fluid Dynamics, Permanent Magnet Machines, Surface Engineering, Machine Dynamics, Centre for Intelligent Machines and Robotics, Compressors & Pumps, Centre for Nano Technology, Ultra High Voltage

Laboratory at Corporate R&D; Centre of Excellence for Hydro Machines at Bhopal; Power Electronics and IGBT & Controller Technology at Electronics Division, Bengaluru, and Advanced Fabrication Technology and Coal Research Centre at Tiruchirappalli.

BHEL has established four specialized institutes, viz., Welding Research Institute (WRI) at Tiruchirappalli, Ceramic Technological Institute (CTI) at Bangalore, Centre for Electric Traction (CET) at Bhopal and Pollution Control Research Institute (PCRI) at Haridwar. Amorphous Silicon Solar Cell plant at Gurgaon pursues R&D in Photo Voltaic applications.

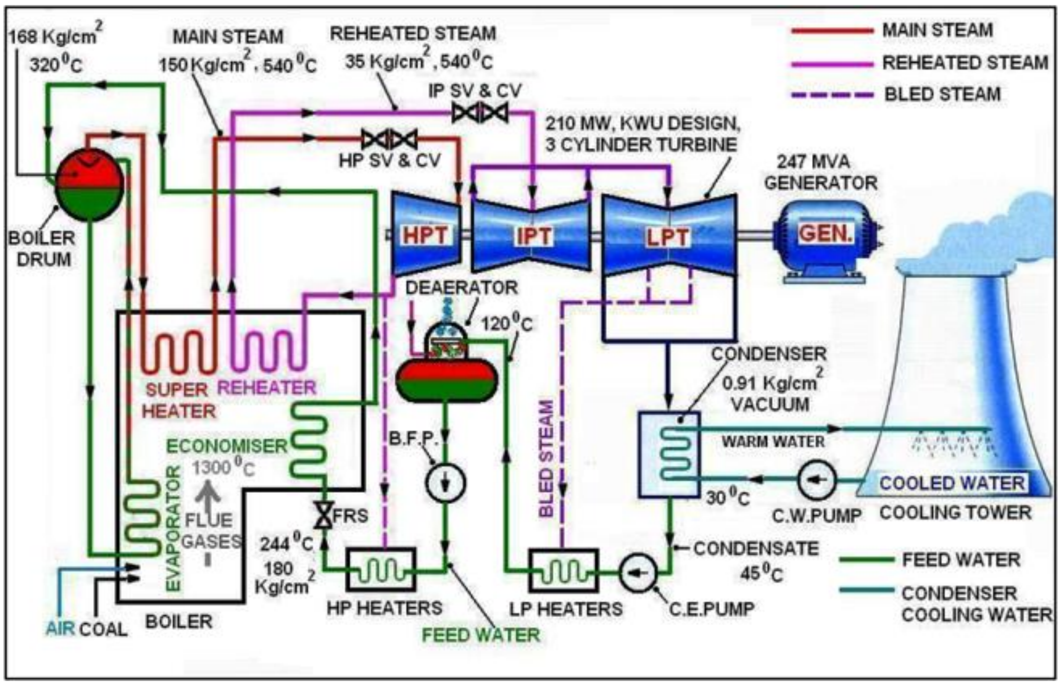
Significantly, BHEL is one of the only four Indian companies and the only Indian Public Sector Enterprise figuring in 'The Global Innovation 1000' of Booz & Co., a list of 1,000 publicly traded companies which are the biggest spenders on R&D in the world.

1.2. THERMAL POWER PLANT

INTRODUCTION:

A thermal power station is a power station in which heat energy is converted to electric power. In most of the places in the world the turbine is steam-driven. Water is heated, turns into steam and spins a steam turbine which drives an electrical generator. After it passes through the turbine, the steam is condensed in a condenser and recycled to where it was heated; this is known as a Rankine cycle. The greatest variation in the design of thermal power stations is due to the different heat sources; fossil fuel dominates here, although nuclear heat energy and solar heat energy are also used. Some prefer to use the term energy centre because such facilities convert forms of heat energy into electrical energy. Certain thermal power stations also are designed to produce heat energy for industrial purposes, or district heating, or desalination of water, in addition to generating electrical power.

LAYOUT AND WORKING OF A THERMAL POWER PLANT:



In a coal-based power plant coal is transported from coal mines to the power plant by railway in wagons or in a merry-go-round system. Coal is unloaded from the wagons to a moving underground conveyor belt. This coal from the mines is of no uniform size. So, it is taken to the Crusher house and crushed to a size of 20mm. From the crusher house the coal is either stored in dead storage (generally 40 days coal supply) which serves as coal supply in case of coal supply bottleneck or to the live storage (8 hours coal supply) in the raw coal bunker in the boiler house. Raw coal from the raw coal bunker is supplied to the Coal Mills by a Raw Coal Feeder. The Coal Mills or pulveriser pulverizes the coal to 200 mesh size. The powdered coal from the coal mills is carried to the boiler in coal pipes by high pressure hot air. The pulverized coal air mixture is burnt in the boiler in the combustion zone. Generally in modern boilers tangential firing system is used i.e. the coal nozzles/ guns form tangent to a circle. The temperature in fire ball is of the order of 1300 deg C. The boiler is a water tube boiler hanging from the top. Water is converted to steam in the boiler and steam is separated from water in the boiler Drum. The saturated steam from the boiler drum is taken to the Low Temperature Superheater, Platen Superheater and Final Superheater respectively for superheating. The superheated steam from the final superheater is taken to the High-Pressure Steam Turbine (HPT). In the HPT the steam pressure is utilized to rotate the turbine and the resultant is rotational energy. From the HPT the out coming steam is taken to the Reheater in the boiler to increase its temperature as the steam becomes wet at the HPT outlet. After reheating this steam is taken to the Intermediate Pressure Turbine (IPT) and then to the Low-Pressure Turbine (LPT). The outlet of the LPT is sent to the condenser for condensing back to water by a cooling water system. This condensed water is collected in the Hot well and is again sent to the boiler in a closed cycle. The rotational energy imparted to the turbine by high pressure steam is converted to electrical energy in the Generator.

ADVANTAGES AND DISADVANTAGES OF THERMAL POWER PLANT

ADVANTAGES:

- Less initial cost as compared to other generating stations.
- It requires less land as compared to hydro power plant.
- The fuel (i.e. coal) is cheaper.
- The cost of generation is lesser than that of diesel power plants.

DISADVANTAGES:

- It pollutes the atmosphere due to the production of large amount of smoke. This is one of the causes of global warming.
- The overall efficiency of a thermal power station is low (less than 30%).

1.2.1. BOILER AUXILIARIES PLANT

Boiler auxiliaries' plant (BAP), the 13th manufacturing unit of BHEL was set up at Ranipet in 1980. BAP at Ranipet was set up as a by-products plant for the manufacture of boiler auxiliaries. The total capacity of BAP is 57,000 tons/year.

The main products are ESP (Electrostatic Precipitator), APH (Air Pre-Heater), FANS and Dampers and Gates. Over the years, the following products are also added to BAP'S portfolio: Steel chimneys, gates and dampers, RO based desalination (for industrial and portable application), flue gas desulphurization systems (FGD), wind electric generators (WEG) and missile launchers for defence.

The unit also manufactures fuel tanks for ISRO'S GSAT, INSAT 3 and IRS P5 applications. The BHEL BAP is accredited by ISO 9001, ISO 14001, ISO 27001 and OHSAS 18001.BAP is the first unit of its whole corporation to upgrade to ISO 9001:2015 Quality Management System.

The company nourishes 400 ancillary small-scale units by off-loading structural fabrication and machining jobs through annual rate contracts.

In its quest to be global engineering enterprise, BHEL pursues continuous improvement in the quality of its product, services and performance leading to customer delight through commitment, innovation and team work of all employees.

BAP PRODUCTS

1. ID FAN (INDUCED DRAFT)
2. FD / SA FAN (FORCED DRAFT/SECONDARY)
3. PA FAN (PRIMARY AIR)
4. AIR PREHEATER(APH)
5. ELECTROSTATIC PRECIPITATOR(ESP)
6. GATES AND DAMPERS
7. FGD (FLUE GAS DESULFURISATION)
8. RO BASED DESALINATION
9. WIND ELECTRIC GENERATORS
10. MISSILE LAUNCHERS FOR DEFENCE

ID FAN:

Induced Draft Fan or ID fan is used to create a vacuum or negative pressure in a system like Steam boiler or Thermal Oil Heater. Induced draft Fan is also used to identify the combustion process used in large boilers. When mechanical ventilation is supplied to these boilers, the heat transfer rate increases Induced draft fan or ID fan is always located between dust collector and chimney. ID fan takes the hot flue gases from furnace via dust collector (dust separation system or Fume Extraction system) and will deliver to chimney. ID fan produce the pressure lower than the atmospheric pressure in the system or we may say that ID fan produces the negative pressure in the furnace to remove the flue gases from furnace via Multi Cyclone and to push the flue gases to chimney.

FD FAN:

Forced Draft Fan is a type of a fan supplying pressurized air to a system. In the case of a Steam Boiler Assembly, this fan is of a great importance. FD fan supplies air to the Air-preheater, where it captures the heat from the flue gases coming from the Boiler Outlet. Forced Draft in a fan occurs when the fluid handled by the fan (generally air), is retained beyond the atmospheric pressure. Every fan designer must never underestimate the importance of the properties and composition of the fuel to be combusted and the bed height in the furnace.

PA FAN:

Primary air fan is used to transport the pulverized coal from mills to the furnace area. Primary air is generally the basic amount of air required for complete combustion of fuel and it depends upon the composition and quantity of fuel required by the boiler. Different fuels will require a different amount of primary air. Primary air is normally provided from the bottom through the airbox and the fuel is fed either manually or through the screw feeder into the furnace. Primary air fans also help in segregating the fuel particles so that each fuel particle could come in direct contact with the air resulting in efficient combustion of fuel particles.

AIR PREHEATER:

An air preheater (APH) is a device designed to heat air before another process with the primary objective of increasing the thermal efficiency of the process. The purpose of the air preheater is to recover the heat from the boiler flue gas which increases the thermal efficiency of the boiler by reducing the useful heat lost in the flue gas. As a consequence, the flue gases are also conveyed to the flue gas stack (or chimney) at a lower temperature, allowing a simplified design of the conveyance system and the flue gas stack. It also allows control over the temperature of gases leaving the stack (to meet emissions regulations). It is installed between the economizer and chimney.

ELECTROSTATIC PRECIPITATOR:

An electrostatic precipitator (ESP) is a filtration device that removes fine particles, like dust and smoke, from a flowing gas using the force of an induced electrostatic charge minimally impeding the flow of gases through the unit. An ESP applies energy only to the particulate matter being collected and therefore is very efficient in its consumption of energy. Electrostatic precipitators are used for air pollution control, particularly for removing particles from waste gases at industrial facilities and power-generating stations.

GATES AND DAMPERS:

All systems require dampers and gates to control or shut off the flow of gases. The degree of effectiveness in which this is done influences the operation of the complete system. It is imperative that the dampers and gates are properly designed and engineered so they will effectively complement other system components and withstand the high temperature, pressure and corrosive gas common in today's process systems. The dampers and gates can modulate gas or air flow and it can isolate a process usually for inspection and repair.

FLUE GAS DESULPHURISER:

Flue-gas desulfurization (FGD) is a control device used to remove sulphur dioxide (SO2) from exhaust flue gases using an alkaline reagent to produce a solid compound. SO_2 is an acid gas, and, therefore, the typical sorbent slurries or other materials used to remove the SO_2 from the flue gases are alkaline. In this chemical reaction >90% of the sulphur dioxide (SO2) from the flue gas can be removed and converts the limestone into calcium sulphite (CaSO4).

RO BASED DESALINATION:

Desalination is a separation process used to reduce the dissolved salt content of saline water to a usable level. Although some substances dissolved in water, such as calcium carbonate, can be removed by chemical treatment, other common constituents, like sodium chloride, require more technically sophisticated methods like desalination process. The product water of the desalination process is generally water with less than 500 mg/1 dissolved solids, which is suitable for most domestic, industrial, and agricultural uses.

2. DESIGN

The solid level measurements in bunkers and hoppers are required in many industries. In a coal fired thermal power plant, pulverized coal is burnt in the boiler furnace. Heat released in the furnace is conveyed to the water and steam circuit. A portion of ash produced after the completion of combustion falls to the bottom ash hopper.

The fly ash in the flue gas is collected in the economizer and air heater hoppers. The major portion of ash is collected downstream in large electrostatic precipitators. The ash handling systems conveys the ash to a distant location in the power plant area where it is transported out of the plant.
A*Type equation here.* Large coal fired plant burns about 300-400 tonnes per hour of coal. With ash content of 30-40 percent in coal, we can imagine the ash produced each hour in the boiler. It is possible that blockage occurs at the outlet of hoppers or ash evacuation system fails leading rise of ash level in hoppers. It is therefore necessary to monitor ash level in the hoppers and raise an alarm when abnormal level is reached.

In this project, we use capacitor type of sensors to detect presence or absence of coal ash at point location. There are different designs of this type of measurement in the markets. They are marketed by reputed manufacturer ad RF admittance type point level sensors.

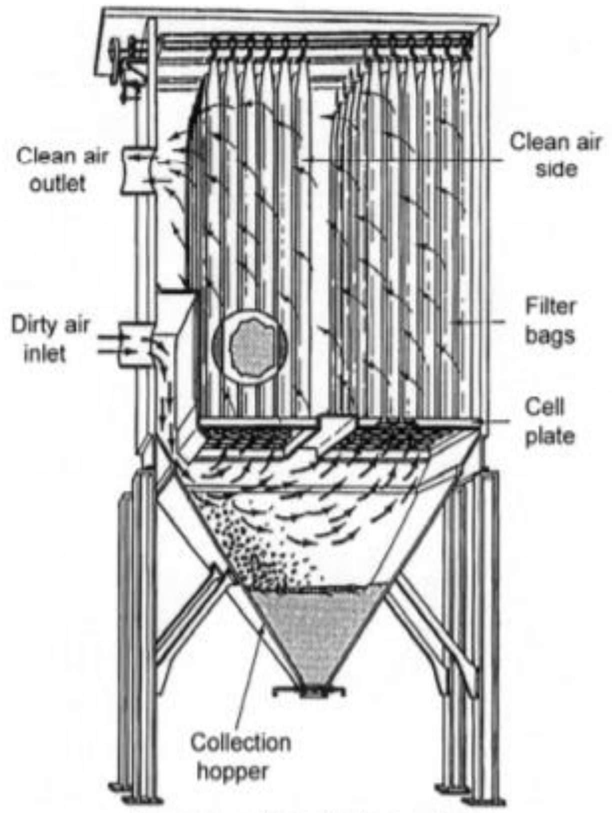
The design of this system envisages change in the capacitance due to change in the medium, ash is known to have known to have a dielectric constant between 1.8 to 2.8. The change in capacitance for dielectric medium with flue gas and ash will be almost two. We detect the change in capacitance directly with an electronic circuit. A capacitance sensor is inserted a hopper.

The electronic circuitry sends a pulse stream to the sensor through a resistor of suitable value. We know the shape of response of an RC circuit subjected to a dc pulse. It is the exponential in rise and decay. The change in capacitance for flue gas and ash causes change in time constant. This change is detected in a microcontroller and an alarm can be generated.

Multiprobe arrangement can give information of presence of ash at different hopper levels. Figure 1 shows a bunker with probes local electronics.

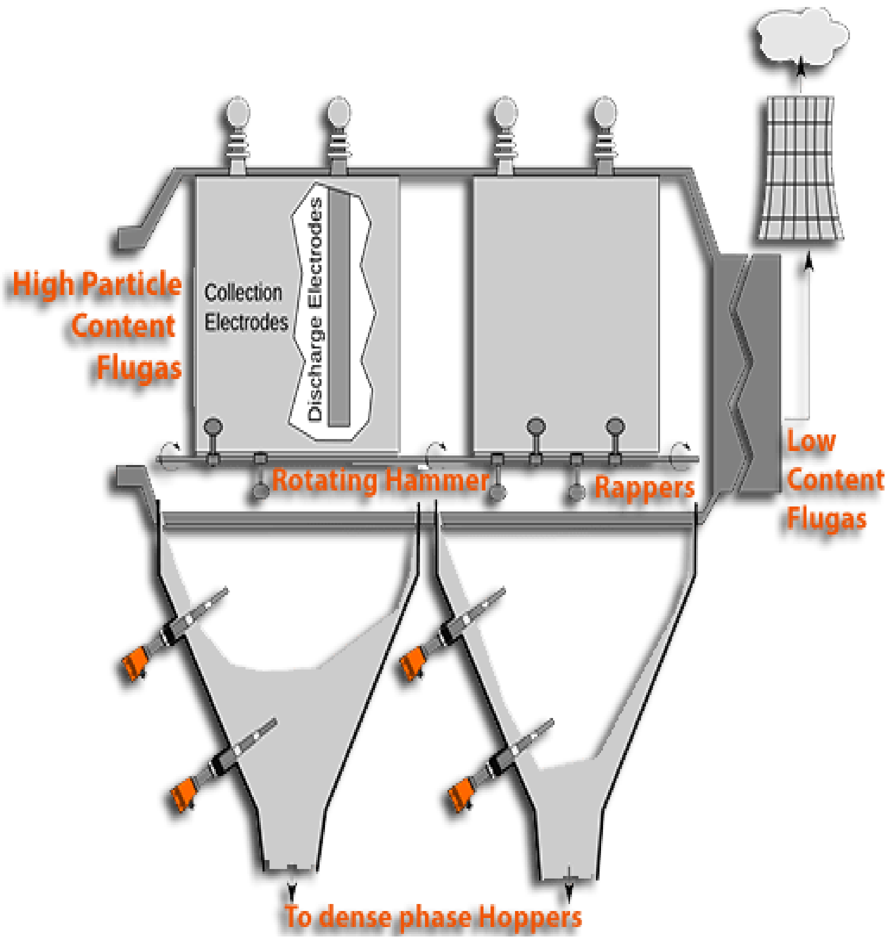
Electrostatic precipitators work better than the alternative, the fabric filter baghouse…

… especially when the gas to be treated and its particles are hot or wet.



Typical simple fabric filter baghouse
(*Source*: Wheelabrator Pollution Control)

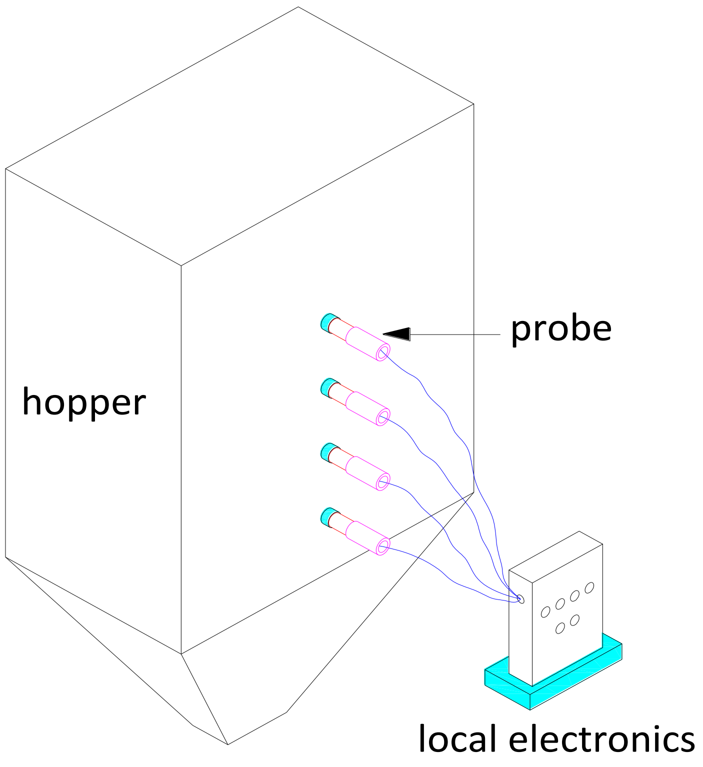
PROCESS:



In this project, we use capacitor type of sensors to detect presence or absence of coal ash at point location. There are different designs of this type of measurement in the markets.

They are marketed by reputed manufacturer ad RF admittance type point level sensors. The design of this system envisages change in the capacitance due to change in the medium, ash is known to have a dielectric constant between 1.8 to 2.8.

The silos or hoppers are the temporary holding structures for the products/ bi-products. These structures are often made of metallic body. The level measurement in hoppers plays measure role for monitoring and automation of fuel feeding, packaging or waste management systems (as in the case of fly ash). Various



techniques are used for level measurement in the hoppers depending upon the nature of the substance and the environment inside the hopper. Joseph D. Lewis presented the different conditions of the bulk solid level measurement in bins, hoppers/silos using various sensors like load cell-based level measurement, ultrasonic sensor, laser sensor and guided wave radar sensor level systems [1]. Søren Vinter Søgaard et. al. did the extensive study on the hopper design and powder flow for a small-scale process is presented and experimentally observed. For this study of prediction and experimentally validation, the level of the powder material present inside the hopper is considered as a measure factor for determining the proper out flow the powdered material [2]. Capacitive sensors are most popular sensor for level measurement because of its simple design and non-invasive nature. A lot of research works presented for designing the capacitive sensors and the signal conditioning circuits for these sensors. Coaxial capacitive sensors are widely used for liquid level measurement for relatively small and closed containers.

3.The working of a capacitor

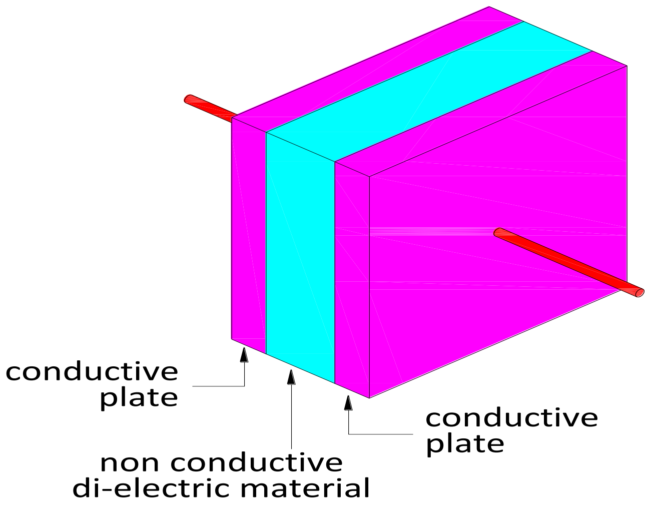
The value for the capacitance is calculated from the formula given below:

$$C = \frac{\in o A}{D}$$

where, $\in o$ is permittivity of the medium
A is area and
D is distance between plates

$$\in = \in \mathbf{o} \in \mathbf{r}$$

where, $\in o$ is permittivity of free space
$\in$r is relative permittivity

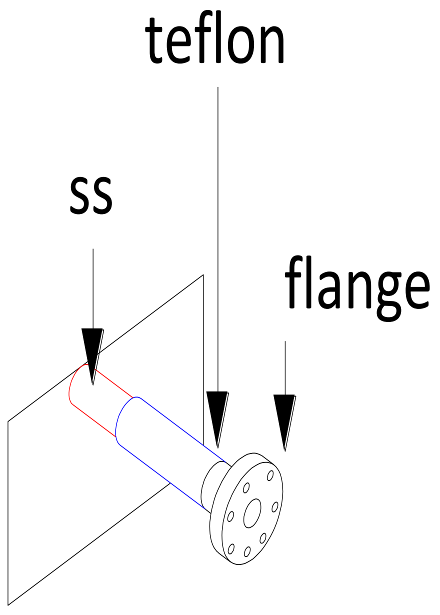


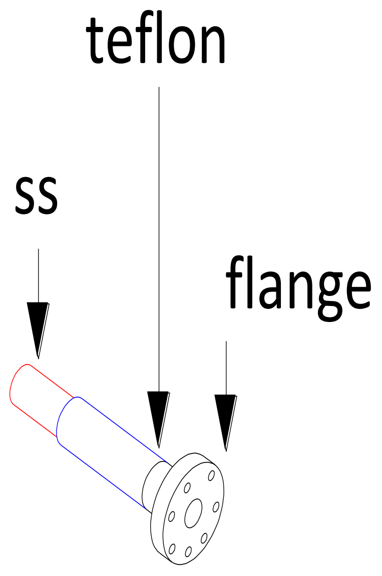
Figure–2 parallel plate capacitor

The value of relative permittivity depends on the dielectric material between the plates. A parallel plate capacitor is shown in Figure 2 below. The plates are made of conductive material while the medium between the plates is of a dielectric material. The higher the value of dielectric constant, the more will be the value of the capacitance and the more will be the quantity of charge stored in the plates. In the capacitance type of level measurement, the value of the capacitance changes when ash in the hopper reaches the level of the sensor. The dielectric medium in this case will be ash.

3.1 **<u>The sensor designs</u>**

The design of the sensor is shown in Figure 3. The sensor consists of a metallic portion and an insulator portion. While the metallic portion forms one plate of the capacitor, the hopper itself forms the other plate of the capacitor. The design of this sensing element depends on the size of the hopper. The material of the sensor is decided based on the conditions of measurement. The metal rod in the probe itself can be made of stainless steel which can withstand high temperatures. The insulator portion of the probe is selected according to medium temperature. Teflon can be used up to 250°C. For higher temperatures Ceramic material is used. The sensor probe is inserted into the hopper.

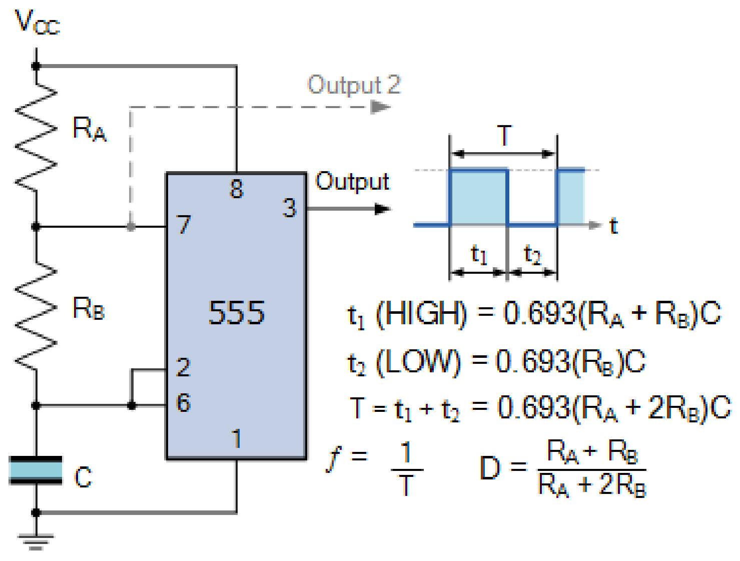




4. SENSOR ELECTRONICS CIRCUIT DIAGRAM

Researchers have used coaxial capacitive sensor and designed a very low cost microcontroller and 555 timer based signal conditioning circuit
[3]. A. Quartzoid et. al. also presented a parallel plate capacitive sensor and signal conditioning circuit with linearized output for water level measurement
[4]. A non-inductive capacitive sensor is also designed for level measurement of conductive and non-conductive liquids and the performance in analysed in inclined positions
[5]. K. Gruden et. al. has analysed the granular flow in silos.
[6]. Hemitheid et. al presented a low cost microcontroller and load cell based level monitoring system for coal bunkers/hoppers
[7]. Now a days point level capacitance switches are used in hoppers for level measurement of fly ash in hoppers
[8]. The point level switches cannot be used for continuous level measurement. It can be used as a switch to actuate any actuator or to trigger a particular process.

In the present work a capacitive sensor is designed and experimentally validated. The present sensor is a non-invasive sensor also this sensor can be used for continuous level monitoring. The model of the hopper is made from metal sheet and copper sheets are used as capacitive electrodes. The presented design is experimentally evaluated and the experiment is repeated multiple times for two different industrial materials (fly ash and cement clinker).



t_1 (HIGH) $= 0.693(R_A + R_B)C$

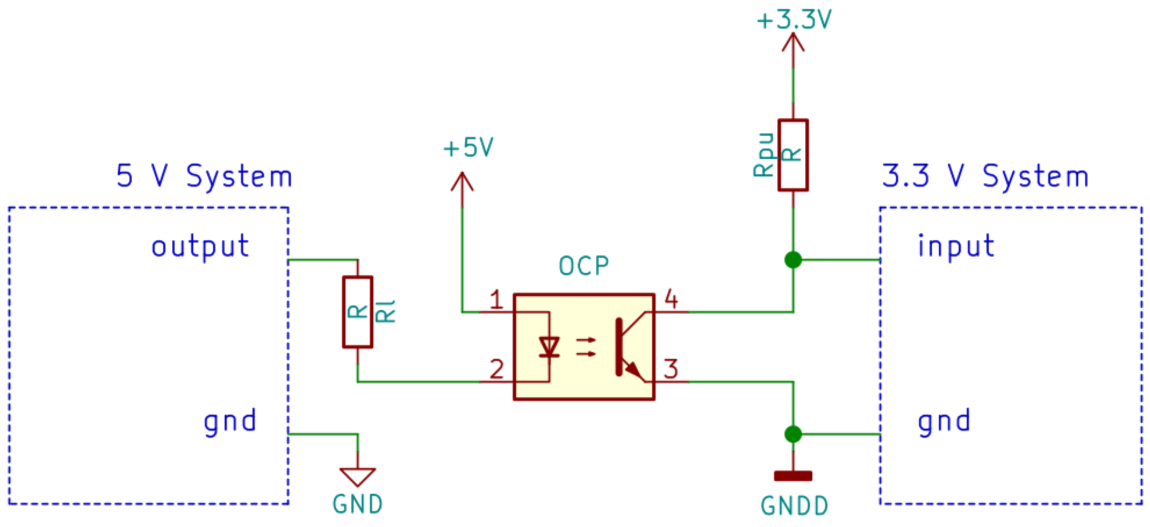
t_2 (LOW) $= 0.693(R_B)C$

$T = t_1 + t_2 = 0.693(R_A + 2R_B)C$

$f = \frac{1}{T}$ $\quad D = \frac{R_A + R_B}{R_A + 2R_B}$

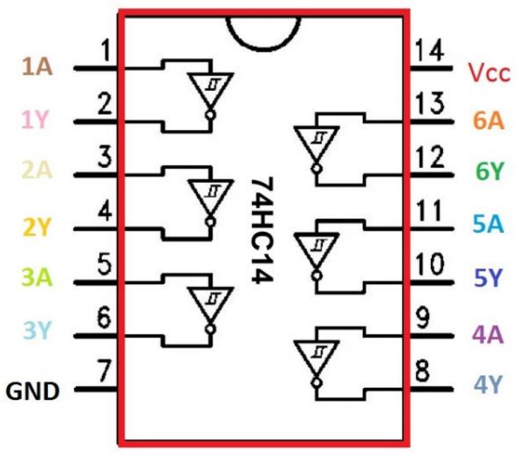
Where: t1 is the output high duration, t2 is the output low duration, T is the periodic time of the output waveform, *ƒ* is the frequency of the output waveform, and 0.693 = ln(2)
When connected as an astable oscillator, capacitor C charges through RA and RB but discharges only through RB.
Thus the duty cycle D is determined by the ratio of these two resistors. With the proper selection of resistors RA and RB, duty cycles of between 50 and 100% can be easily set.
The total time period T is given as the capacitor charging time, t1 (Output High) plus the discharging time, t2 (Output Low) as the capacitor charges and discharges between 1/3Vcc and 2/3Vcc respectively.
In this mode of operation, the charging and discharging times and therefore the frequency, *ƒ* which is given as: 1/T, is independent of the supply voltage.

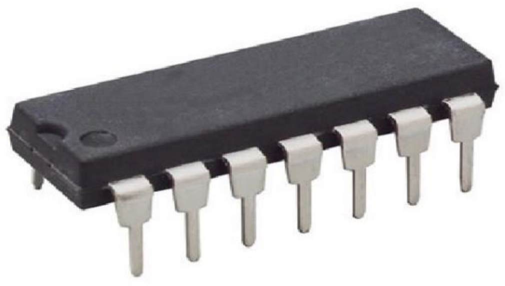
Though the 555 timers can be powered at 5V, the microcontroller input signal is limited to 3.3V. So, an opto isolator, PC817, is used to level shift the 5V signal to 3.3V signal.



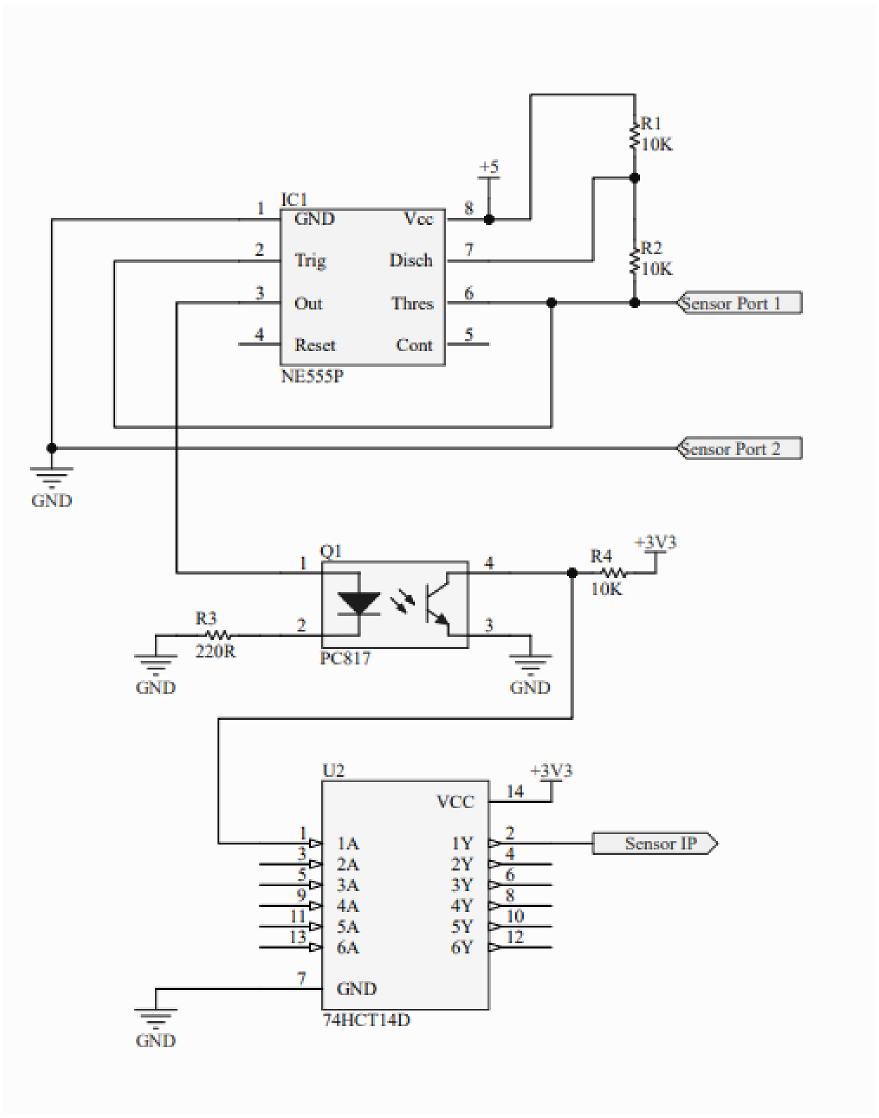
Hex Schmitt–Trigger Inverter

Due to the slower response of the opto isolator during circuit testing, it was decided to introduce a Schmitt–Trigger Inverter to improve the width of the frequency signal from the 555-timer circuit.74HC14 is a member of 74XXXX integrated circuit series, it consists of logic gates. This module is also called HEX Inverting Schmitt Trigger. It is available in six independent Schmitt trigger input inverters with standard push-pull outputs. The boolean function performed by this logic gates is $Y= \overline{A}$. It is a 14-pin module which is available in various packages. The 74HC14 works on the voltage range of 2.0V to 6.0V. This is a higher speed CMOS Schmitt Inverter mounted with a silicon gate C2MOS technology.





74HC14 Package



5.OPC UA

OPC Unified Architecture (OPC UA) is a machine-to-machine communication protocol used for industrial automation and developed by the OPC Foundation.
The OPC UA platform in an platform-independent service-oriented architecture that integrates individual OPC Classic specifications into an extensible framework
The OPC Unified Architecture (UA), released in 2008, is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework
Platform Independence
Given the wide array of available hardware platforms and operating systems, platform independence is essential. OPC UA functions on any of the following and more:
Hardware platforms: traditional PC hardware, cloud-based servers, PLCs, micro-controllers (ARM etc.)
Operating Systems: Microsoft Windows, Apple OSX, Android, or any distribution of Linux, etc.

Security:

One of the most important considerations in choosing a technology is security. OPC UA is firewall-friendly while addressing security concerns by providing a suite of controls:

Transport:
numerous protocols are defined providing options such as the ultra-fast OPC-binary transport or the more universally compatible JSON over WebSocket's, for example

Session Encryption:
messages are transmitted securely at various encryption levels

Message Signing:
with message signing the recipient can verify the origin and integrity of received messages

Sequenced Packets:
exposure to message replay attacks is eliminated with sequencing

Authentication:
each UA client and server is identified through X509 certificates providing control over which applications and systems are permitted to connect with each other

User Control:
applications can require users to authenticate (login credentials, certificate, web token etc.) and can further restrict and enhance their capabilities with access rights and address-space “views”

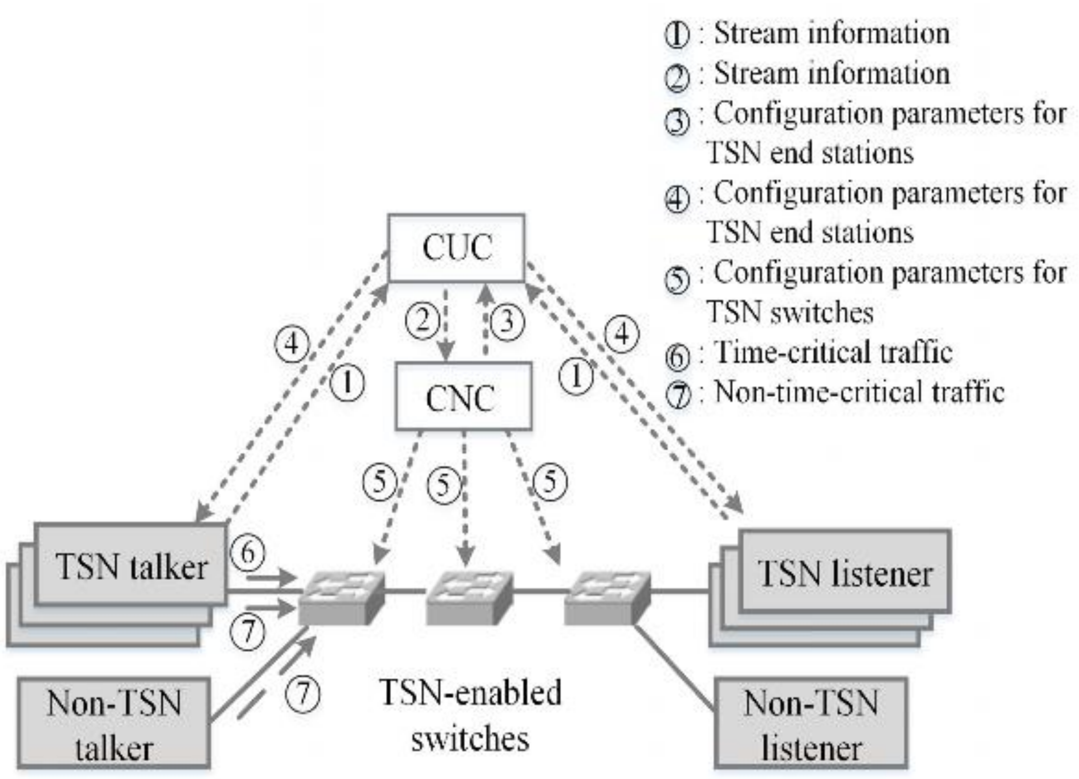
Auditing:
activities by user and/or system are logged providing an access audit trail

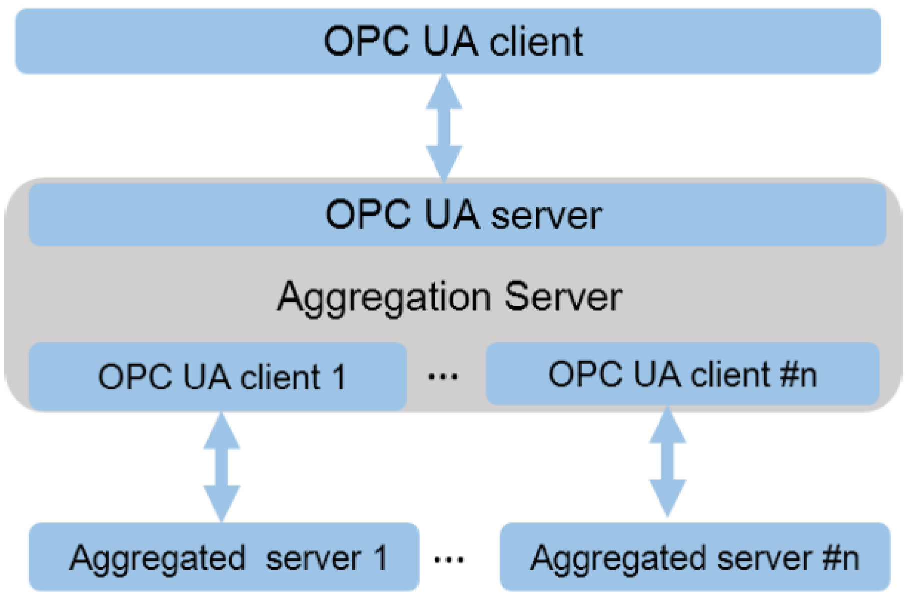
Cross-platform
not tied to one operating system or programming language

Service-oriented architecture (SOA)
The specification is freely available on the OPC Foundation website and is split into several parts to ease implementation, but only OPC UA stack vendors need to read them, end users simply leverage existing commercial and/or open-source stacks available in all popular programming languages

Open
open-source reference implementations freely available to OPC Foundation members, non-members under GPL 2.0 license [
Binaries (NuGet Packages) available free of charge under redistributable license

In modern manufacturing systems, various industrial communication systems (e.g., fieldbus systems and industrial Ethernet networks) have been used to realize reliable information exchange. However, these industrial communication solutions are largely incompatible with each other, which do not satisfy the new requirements of Industry 4.0. Recently Time-Sensitive Networking (TSN) has been developed to improve the real-time capabilities to the standard Ethernet, and is considered to be a promising real-time communication solution for Industry 4.0. In this work, we propose a communication architecture for a manufacturing system using the Open Platform Communications Unified Architecture (OPC UA) and TSN technologies. TSN is adopted as the communication backbone to connect heterogeneous industrial automation subsystems. The OPC UA is adopted to realize horizontal and vertical communication between subsystems in the field layer and the entities of the upper layers.


①: Stream information
②: Stream information
③: Configuration parameters for TSN end stations
④: Configuration parameters for TSN end stations
⑤: Configuration parameters for TSN switches
⑥: Time-critical traffic
⑦: Non-time-critical traffic
CUC
CNC
TSN talker
TSN listener
Non-TSN talker
Non-TSN listener
TSN-enabled switches


OPC UA client
OPC UA server
Aggregation Server
OPC UA client 1
···
OPC UA client #n
Aggregated server 1
···
Aggregated server #n

5.1. open62541

open62541 (http://open62541.org) is an open source and free implementation of OPC UA (OPC Unified Architecture) written in the common subset of the C99 and C++98 languages. The library is usable with all major compilers and provides the necessary tools to implement dedicated OPC UA clients and servers, or to integrate OPC UA-based communication into existing applications. open62541 library is platform independent. All platform-specific functionality is implemented via exchangeable plugins. Plugin implementations are provided for the major operating systems.
open62541 is licensed under the Mozilla Public License v2.0 (MPLv2). This allows the open62541 library to be combined and distributed with any proprietary software. Only changes to the open62541 library itself need to be licensed under the MPLv2 when copied and distributed. The plugins, as well as the server and client examples are in the public domain (CC0 license). They can be reused under any license and changes do not have to be published.
The sample server (server_ctt) built using open62541 v1.0 is in conformance with the 'Micro Embedded Device Server' Profile of OPC Foundation supporting OPC UA client/server communication, subscriptions, method calls and security (encryption) with the security policies 'Basic128Rsa15', 'Basic256' and 'Basic256Sha256' and the facets 'method server' and 'node management'.
See https://open62541.org/certified-sdk for more details.

OPC Unified Architecture

OPC UA is a protocol for industrial communication and has been standardized in the IEC 62541 series.
At its core, OPC UA defines an asynchronous protocol (built upon TCP, HTTP or SOAP) that defines the exchange of messages via sessions, (on top of) secure communication channels, (on top of) raw connections,
a type system for protocol messages with a binary and XML-based encoding scheme,
a meta-model for information modelling, that combines object-orientation with semantic triple-relations, and a set of 37 standard services to interact with server-side information models.
The signature of each service is defined as a request and response message in the protocol type system.
The standard itself can be purchased from IEC or downloaded for free on the website of the OPC Foundation at https://opcfoundation.org/ (you need to register with a valid email).

The OPC Foundation drives the continuous improvement of the standard and the development of companion specifications. Companion specifications translate

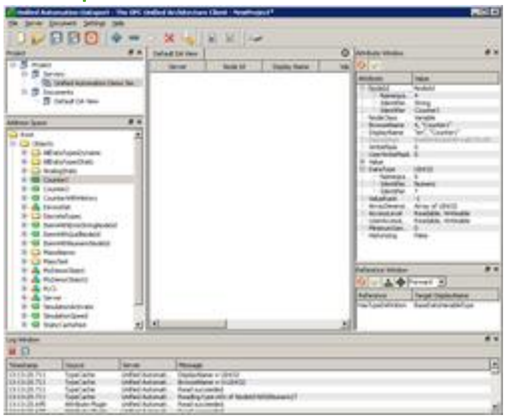
established concepts and reusable components from an application domain into OPC UA. They are created jointly with an established industry council or standardization body from the application domain. Furthermore, the OPC Foundation organizes events for the dissemination of the standard and provides the infrastructure and tools for compliance certification.

open62541 Features

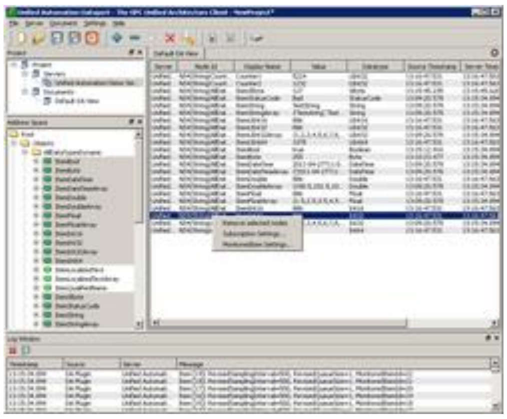
open62541 implements the OPC UA binary protocol stack as well as a client and server SDK. It currently supports the Micro Embedded Device Server Profile plus some additional features. Server binaries can be well under 100kb in size, depending on the contained information model.
Communication Stack
OPC UA binary protocol
Chunking (splitting of large messages)
Exchangeable network layer (plugin) for using custom networking APIs (e.g. on embedded targets)
Encrypted communication
Asynchronous service requests in the client
Information model
Support for all OPC UA node types (including method nodes)
Support for adding and removing nodes and references also at runtime.
Support for inheritance and instantiation of object- and variable-types (custom constructor/destructor, instantiation of child nodes)
Access control for individual nodes
Subscriptions
Support for subscriptions/monitoreditems for data change notifications
Very low resource consumption for each monitored value (event-based server architecture)
Code-Generation
Support for generating data types from standard XML definitions
Support for generating server-side information models (nodesets) from standard XML definitions

5.2.OPC UA CLIENT

UaExpert—A Full-Featured OPC UA Client

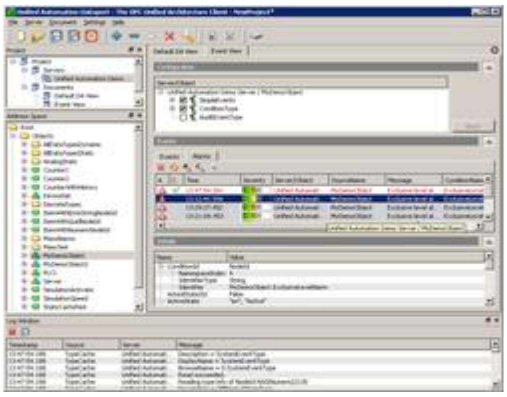
The UaExpert® is a full-featured OPC UA Client demonstrating the capabilities of our C++ OPC UA Client SDK/Toolkit. The UaExpert is designed as a general purpose test client supporting OPC UA features like Data Access, Alarms & Conditions, Historical Access and calling of UA Methods. The UaExpert is a cross-platform OPC UA test client programmed in C++. It uses the sophisticated GUI library QT form Nokia (formerly Trolltech) forming the basic framework which is extendable by Plugins.
Free version of UaExpert comes with following Plugins:
OPC UA Data Access View
OPC UA Alarms & Conditions View
OPC UA Historical Trend View
Server Diagnostics View
Simple Datalogger CSV Plugin
OPC UA Performance Plugin
GDS Push-Model Plugin
XMLNodeSet-Export View (requires license)
The UaExpert is available for Windows and Linux.
UaExpert Common Framework

The basic framework of UaExpert includes general functionality like certificate handling, discovering UA Servers, connecting with UA Servers, browsing the information model, displaying attributes and references of particular UA Nodes. The Project pane (upper left window) shows the connected UA Servers and the open document plugins. The Address Space pane (lower left window) shows the UA Servers information model. Depending on the Node selected in the Browser the Attribute and Reference Windows (upper right and lower right windows) show the attribute of the selected Node and its references within the meshed network of the servers address space.

OPC UA DataAccess View



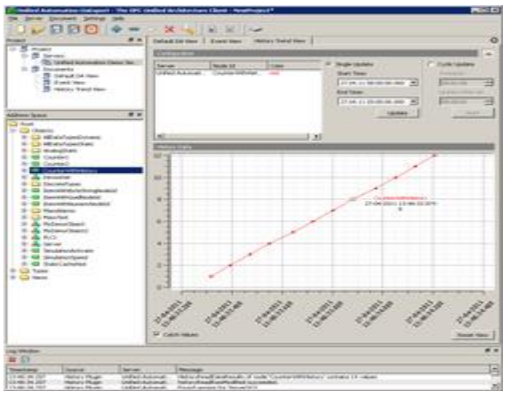
This plugin is shown in the centre pane of UaExpert by default. You can (multi-) select UA Nodes in the Address Space window and drag-and-drop them into the DA View. The DA View creates a subscription and monitors the Nodes. Sampling rate and subscription interval can be changed by right-click into the DA View. When double-clicking into the Value column of a specific Node you can write new values to that Node, writing of scalar, array and matrix types is supported. The DA View was designed to show the classic view on OPC Servers solely concentrating on item monitoring and displaying values, timestamp and status of individual nodes.

OPC UA Alarms & Conditions View



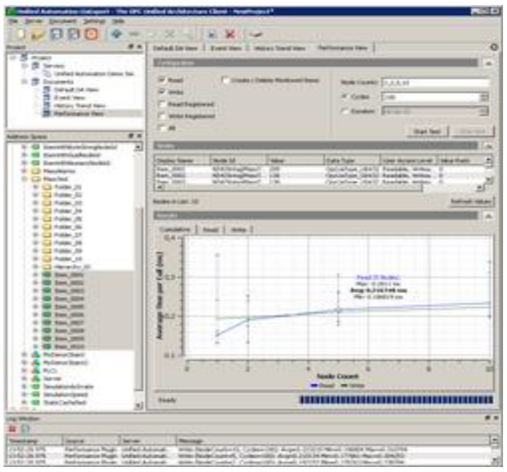
The Event View document can be added using the Add Document button in the menu bar. The Event plugin will be displayed in the centre pane and consists of three major groups, the Configuration, the Event/Alarm view and the Details view showing the detailed information of an individually selected alarm.
You need to select a UA Node (this object must have the Has Events attribute) in the Address Space browser an drag-and-drop it into the Configuration group of the Event View. Now you can select Event Fields you are interested in for this eventing object, some fields are already configured by default (check boxes in the tree view). Here you can switch tabs between Events and Alarm showing you the historical list of events or the current state of pending alarm. When clicking on an event the lower pane group will show you all the details of this particular event according to the selected event fields you ticked in the configuration.

OPC UA Historical Trend View



The Historical View document can be added using the Add Document button in the menu bar. The Historical plugin will be displayed in the center pane and consists of two major groups, the Configuration and the historical data view showing the values in a graphical trend view related to the requested time frame. You need to select a UA Node (this object must have set the HistoryReadable flag of the (User)AccessLevel) in the Address Space browser an drag-and-drop it into the Configuration group of the Historical View. Here you can select the drawing color, especially when you add several Nodes to the list. TheHistorical Trend View supports two modes for fetching the data from the UA Server, the Single Update and the Cyclic Update.
For Single Update you need to specify the time frame, defined by start and end date/time and the UaExpert will perform an historical read raw when pressing the Update button. In the Cyclic Update mode you must specify the time span (backwards from now) and the interval the UaExpert should fetch new data. When pressing the Start button the UaExpert will cyclic (interval) perform an historical read raw using now as end time and now-timespan and end time. This will give a typical chart recorder use case.

OPC UA Performance View



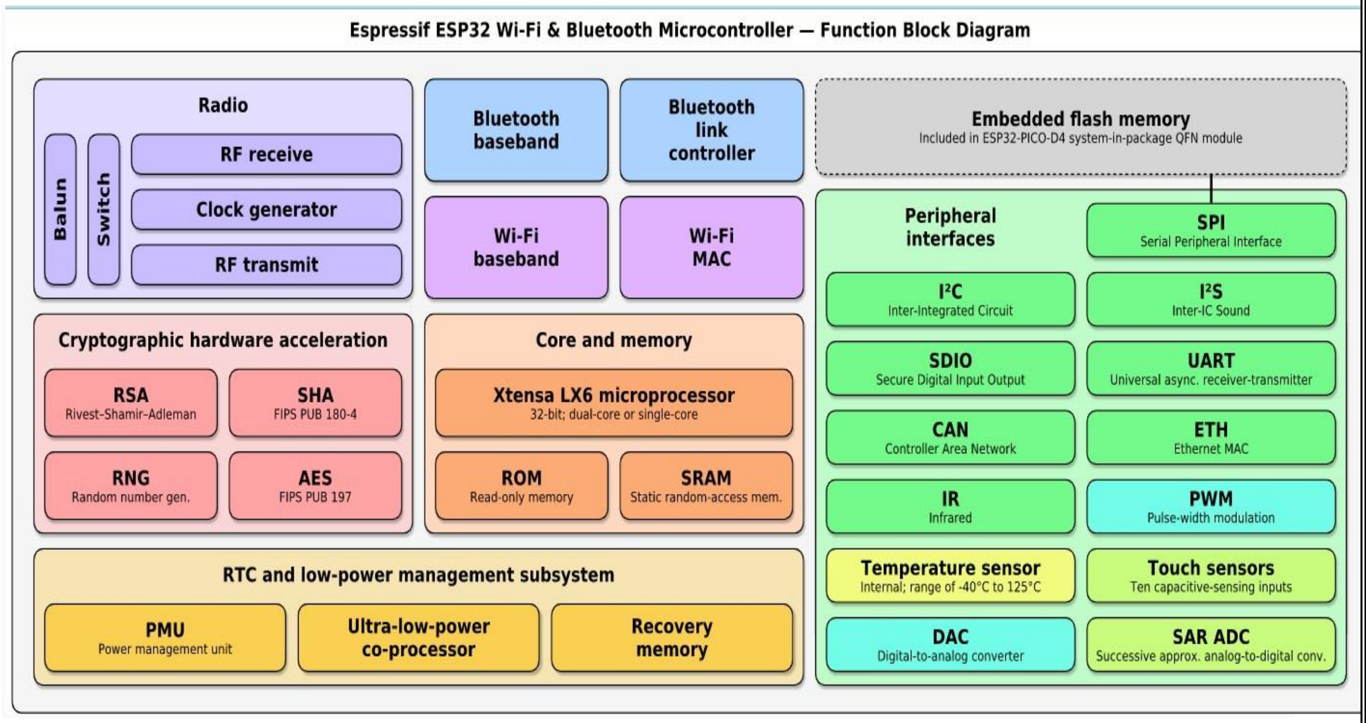
The Performance View document can be added using the Add Document button in the menu bar. The performance plugin will be displayed in the center pane and

consists of three major groups, the Configuration, the list of used nodes and the results showing the measurement in a graphical view.
First you need to select the UA Nodes that should be used for testing (they all must have proper accessrights for write tests and they should move - automonously change data - for subscription tests). You can (multi-) select the nodes in the AddressSpace browser and drag-and-drop them into the centerlist of nodes. All nodes must be from the same UA Server and should have the same data type for easier interpretation of the results. Now the performance measurement must be configured, you need to select the UA services you want to measure and you must configure the number of nodes used for the set of measurements. The number of cycles gives you the number of calls performed for each measurement (high number will give acurate average measurement, but may take long time). The UaExpert will call the UA Service and measure the duration of each call. Alternatively you can choose the Duration option instead. Here the UaExpert will call the UA service as fast as possible in that time span and count how many calls could be performed (this option should be used on very fast operation to give an accurate result).
The results will be displayed in the lower graphical pane. The cumulative graphic shows the comparison between different UA services (comparing calculated average values). In addition every individual UA service measurement is displayed separately to validate constancy and trustability of the measurement.

6.ESP32

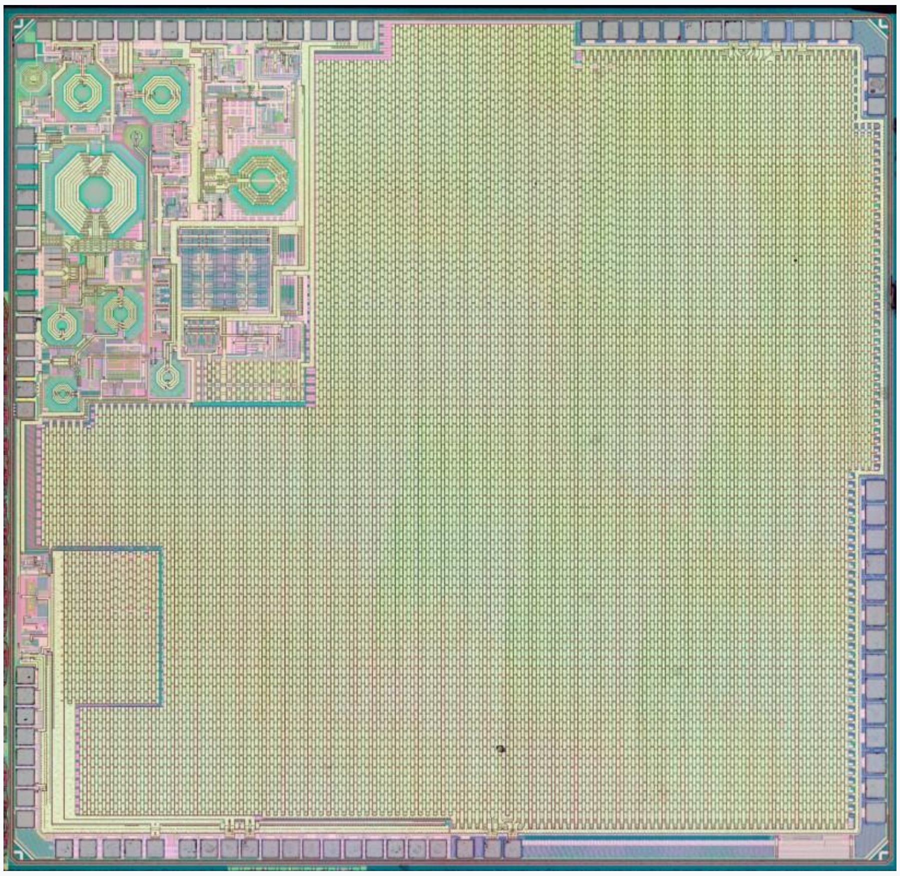
ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs either a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations, Xtensa LX7 dual-core microprocessor or a single-core RISC-V microprocessor and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules. ESP32 is created and developed by Espressif Systems, a Shanghai-based Chinese company, and is manufactured by TSMC using their 40 nm process.[2] It is a successor to the ESP8266 microcontroller.

Features



Espressif ESP32 Wi-Fi & Bluetooth Microcontroller — Function Block Diagram
Radio
Balun
Switch
RF receive
Clock generator
RF transmit
Bluetooth baseband
Bluetooth link controller
Wi-Fi baseband
Wi-Fi MAC
Embedded flash memory
Included in ESP32-PICO-D4 system-in-package QFN module
Peripheral interfaces
SPI
Serial Peripheral Interface
I²C
Inter-Integrated Circuit
I²S
Inter-IC Sound
SDIO
Secure Digital Input Output
UART
Universal async. receiver-transmitter
CAN
Controller Area Network
ETH
Ethernet MAC
IR
Infrared
PWM
Pulse-width modulation
Temperature sensor
Internal; range of -40°C to 125°C
Touch sensors
Ten capacitive-sensing inputs
DAC
Digital-to-analog converter
SAR ADC
Successive approx. analog-to-digital conv.
Cryptographic hardware acceleration
RSA
Rivest-Shamir-Adleman
SHA
FIPS PUB 180-4
RNG
Random number gen.
AES
FIPS PUB 197
Core and memory
Xtensa LX6 microprocessor
32-bit; dual-core or single-core
ROM
Read-only memory
SRAM
Static random-access mem.
RTC and low-power management subsystem
PMU
Power management unit
Ultra-low-power co-processor
Recovery memory

ESP32 function block diagram.



ESP32 Die shot

6.1. **Features of the ESP32 include the following**

- Processors:
  - CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
  - Ultra-low power (ULP) co-processor
- Memory: 320 KiB RAM, 448 KiB ROM
- Wireless connectivity:
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
- Peripheral interfaces:
  - 34 × programmable GPIOs
  - 12-bit SAR ADC up to 18 channels
  - 2 × 8-bit DACs
  - 10 × touch sensors (capacitive sensing GPIOs)
  - 4 × SPI
  - 2 × I²S interfaces
  - 2 × I²C interfaces
  - 3 × UART
  - SD/SDIO/CE-ATA/MMC/eMMC host controller
  - SDIO/SPI slave controller
  - Ethernet MAC interface with dedicated DMA and planned IEEE 1588 Precision Time Protocol support[4]
  - CAN bus 2.0
  - Infrared remote controller (TX/RX, up to 8 channels)
  - Pulse counter (capable of full quadrature decoding)
  - Motor PWM
  - LED PWM (up to 16 channels)
  - Hall effect sensor
  - Ultra low power analog pre-amplifier

- Security:
  - IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 (depending on

    - version)[5] and WLAN Authentication and Privacy Infrastructure (WAPI)
    - Secure boot
    - Flash encryption
    - 1024-bit OTP, up to 768-bit for customers
    - Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)

- Power management:
    - Internal low-dropout regulator
    - Individual power domain for RTC
    - 5 µA deep sleep current
    - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

6.2.Inter-Integrated Circuit (I2C)

I2C is a serial, synchronous, half-duplex communication protocol that allows co-existence of multiple masters and slaves on the same bus. The I2C bus consists of two lines: serial data line (SDA) and serial clock (SCL). Both lines require pull-up resistors.
With such advantages as simplicity and low manufacturing cost, I2C is mostly used for communication of low-speed peripheral devices over short distances (within one foot).
ESP32 has 2 I2C controller (also referred to as port), responsible for handling communications on the I2C bus. A single I2C controller can operate as master or slave.

Driver Features

I2C driver governs communications of devices over the I2C bus. The driver supports the following features:
Reading and writing bytes in Master mode
Slave mode
Reading and writing to registers which are in turn read/written by the master

Communication as Master_

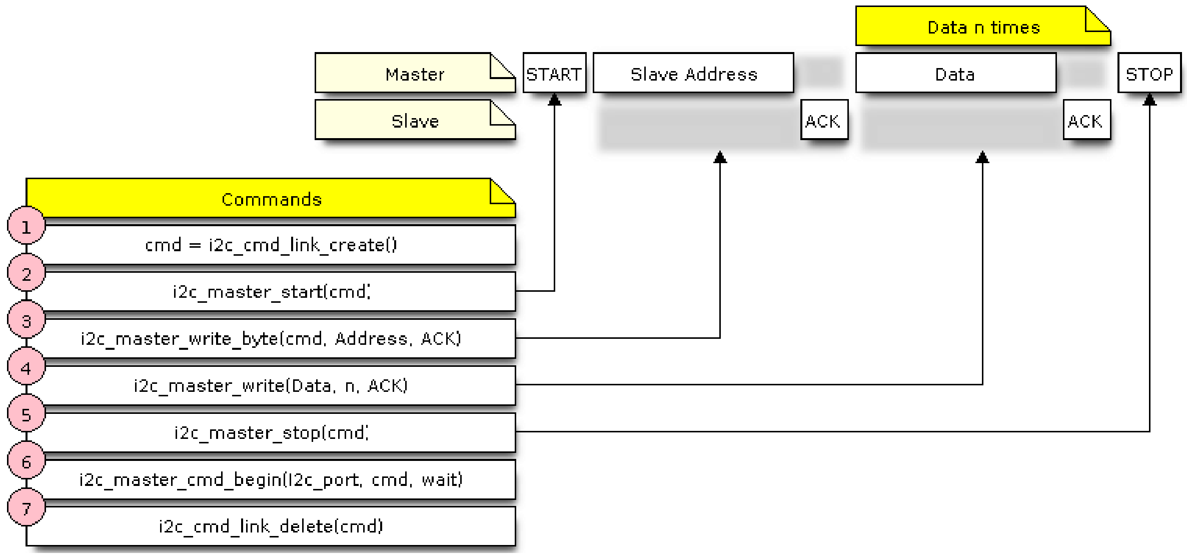
After installing the I2C driver, ESP32 is ready to communicate with other I2C devices.

ESP32's I2C controller operating as master is responsible for establishing communication with I2C slave devices and sending commands to trigger a slave to action, for example, to take a measurement and send the readings back to the master.

For better process organization, the driver provides a container, called a "command link", that should be populated with a sequence of commands and then passed to the I2C controller for execution.

Master Write_

The example below shows how to build a command link for an I2C master to send *n* bytes to a slave.



Master
Slave
START
Slave Address
ACK
Data n times
Data
ACK
STOP
Commands
1
cmd = i2c_cmd_link_create()
2
i2c_master_start(cmd;
3
i2c_master_write_byte(cmd, Address, ACK)
4
i2c_master_write(Data, n, ACK)
5
i2c_master_stop(cmd;
6
i2c_master_cmd_begin(I2c_port, cmd, wait)
7
i2c_cmd_link_delete(cmd)

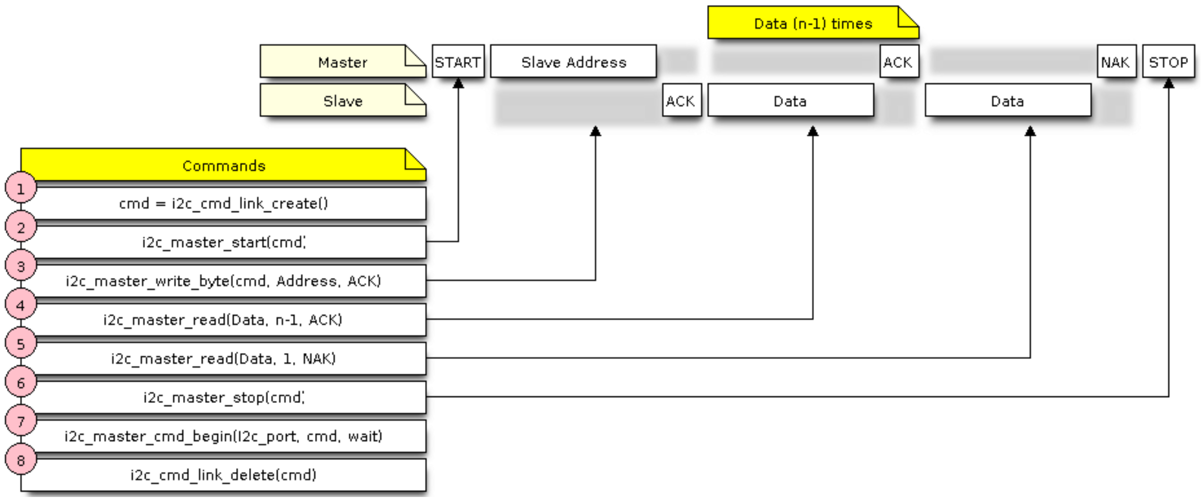
*I2C command link - master write example_*

The following describes how a command link for a "master write" is set up and what comes inside:
Create a command link with `i2c_cmd_link_create()`.
Then, populate it with the series of data to be sent to the slave:
Start bit - `i2c_master_start()`
Slave address - `i2c_master_write_byte()`. The single byte address is provided as an argument of this function call.
Data - One or more bytes as an argument of `i2c_master_write()`
Stop bit - `i2c_master_stop()`

Both functions **i2c_master_write_byte()** and **i2c_master_write()** have an additional argument specifying whether the master should ensure that it has received the ACK bit.
Trigger the execution of the command link by I2C controller by calling **i2c_master_cmd_begin()**. Once the execution is triggered, the command link cannot be modified.
After the commands are transmitted, release the resources used by the command link by calling **i2c_cmd_link_delete()**.

Master Read_

The example below shows how to build a command link for an I2C master to read *n* bytes from a slave.


Data (n-1) times
Master
Slave
START
Slave Address
ACK
ACK
Data
NAK
STOP
Data
Commands
1 cmd = i2c_cmd_link_create()
2 i2c_master_start(cmd)
3 i2c_master_write_byte(cmd, Address, ACK)
4 i2c_master_read(Data, n-1, ACK)
5 i2c_master_read(Data, 1, NAK)
6 i2c_master_stop(cmd)
7 i2c_master_cmd_begin(I2c_port, cmd, wait)
8 i2c_cmd_link_delete(cmd)

I2C command link - master read example

Compared to writing data, the command link is populated in Step 4 not with `i2c_master_write...` functions but with **i2c_master_read_byte()** and/or **i2c_master_read()**. Also, the last read in Step 5 is configured so that the master does not provide the ACK bit.

Indicating Write or Read_

After sending a slave address (see Step 3 on both diagrams above), the master either writes or reads from the slave.
The information on what the master will actually do is hidden in the least significant bit of the slave's address.
For this reason, the command link sent by the master to write data to the slave contains the address `(ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE` and looks as follows:

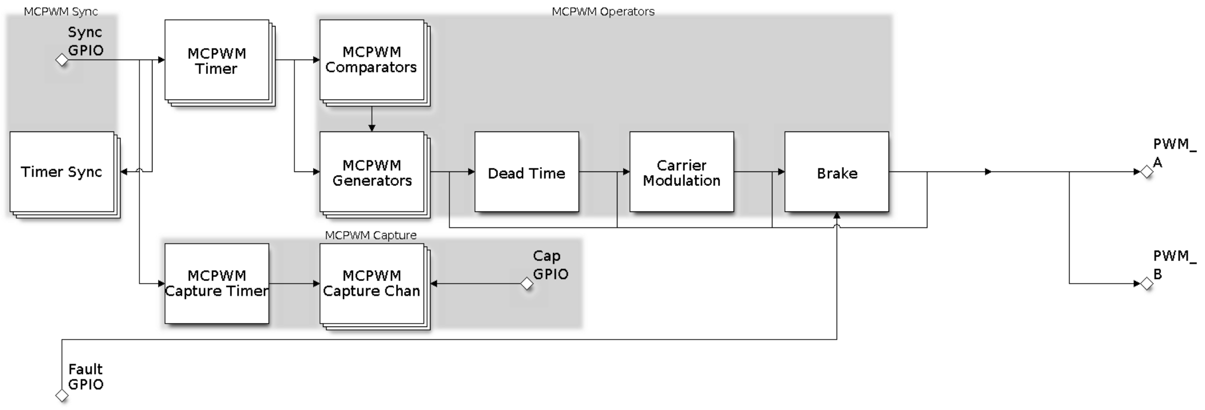
```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) |
I2C_MASTER_WRITE, ACK_EN);
```

Likewise, the command link to read from the slave looks as follows:

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) |
I2C_MASTER_READ, ACK_EN);
```

6.3. **Motor Control Pulse Width Modulator (MCPWM)**

The MCPWM peripheral is a versatile PWM generator, which contains various submodules to make it a key element in power electronic applications like motor control, digital power and so on. Typically, the MCPWM peripheral can be used in the following scenarios:
Digital motor control, e.g. brushed/brushless DC motor, RC servo motor
Switch mode based digital power conversion
Power DAC, where the duty cycle is equivalent to a DAC analog value
Calculate external pulse width, and convert it into other analog value like speed, distance
Generate Space Vector PWM (SVPWM) signals for Field Oriented Control (FOC)
The main submodules are listed in the following diagram:



*MCPWM Overview_*

MCPWM Timer: The time base of the final PWM signal, it also determines the event timing of other submodules.
MCPWM Operator: The key module that is responsible for generating the PWM waveforms. It consists of other submodules, like comparator, PWM generator, dead-time and carrier modulator.
MCPWM Comparator: The compare module takes the time-base count value as input, and continuously compare to the threshold value that configured by user. When the time-base counter is equal to any of the threshold value, an compare event will be generated and the MCPWM generator can update its level accordingly.

MCPWM Generator: One MCPWM generator can generate a pair of PWM waves, complementarily or independently, based on various events triggered from other submodules like MCPWM Timer, MCPWM Comparator.
MCPWM Fault: The fault module is used to detect the fault condition from outside, mainly via GPIO matrix. Once the fault signal is active, MCPWM Operator will force all the generators into a predefined state, to protect the system from damage.
MCPWM Sync: The sync module is used to synchronize the MCPWM timers, so that the final PWM signals generated by different MCPWM generators can have a fixed phase difference. The sync signal can be routed from GPIO matrix or from MCPWM Timer event.
Dead Time: This submodule is used to insert extra delay to the existing PWM edges that generated in the previous steps.
Carrier Modulation: The carrier submodule allows a high-frequency carrier signal to modulate the PWM waveforms generated by the generator and dead time submodules. This capability is mandatory if you need pulse transformer-based gate drivers to control the power switching elements.
Brake: MCPWM operator can set how to brake the generators when particular fault is detected. We can shut down the PWM output immediately or regulate the PWM output cycle by cycle, depends on how critical the fault is.
MCPWM Capture: This is a standalone submodule which can work even without the above MCPWM operators. The capture consists one dedicated timer and several independent channels. Each channel is connected to the GPIO, a pulse on the GPIO will trigger the capture timer to store the time-base count value and then notify the user by interrupt. Using this feature, we can measure a pulse width precisely. What's more, the capture timer can also be synchronized by the MCPWM Sync submodule.

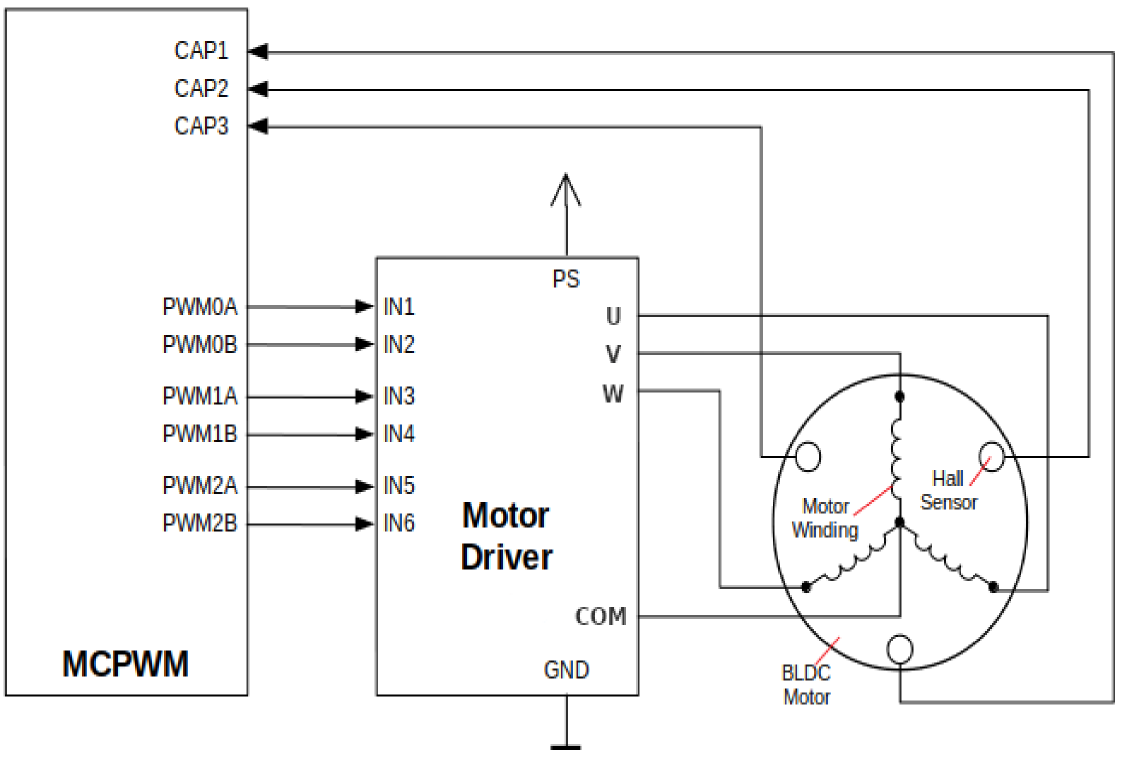
MCPWM Capture Timer and Channels_

The MCPWM group has a dedicated timer which is used to capture the timestamp when specific event occurred. The capture timer is connected with several independent channels, each channel is assigned with a GPIO.
To allocate a capture timer, you can call **`mcpwm_new_capture_timer()`** function, with configuration

structure **`mcpwm_capture_timer_config_t`** as the parameter. The configuration structure is defined as:
`mcpwm_capture_timer_config_t::group_id` sets the MCPWM group ID. The ID should belong to [0, **SOC_MCPWM_GROUPS** - 1] range.
`mcpwm_capture_timer_config_t::clk_src` sets the clock source of the capture timer.
The **`mcpwm_new_capture_timer()`** will return a pointer to the allocated capture timer object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no free capture timer left in the MCPWM group, this function will return **ESP_ERR_NOT_FOUND** error. 1
Next, to allocate a capture channel, you can call **`mcpwm_new_capture_channel()`** function, with a capture timer handle and configuration structure **`mcpwm_capture_channel_config_t`** as the parameter. The configuration structure is defined as:
`mcpwm_capture_channel_config_t::gpio_num` sets the GPIO number used by the capture channel.
`mcpwm_capture_channel_config_t::prescale` sets the prescaler of the input signal.
`mcpwm_capture_channel_config_t::pos_edge` and **`mcpwm_capture_channel_config_t::neg_edge`** set whether to capture on the positive and/or negative edge of the input signal.
`mcpwm_capture_channel_config_t::pull_up` and **`mcpwm_capture_channel_config_t::pull_down`** set whether to pull up and/or pull down the GPIO internally.
`mcpwm_capture_channel_config_t::invert_cap_signal` sets whether to invert the capture signal.
`mcpwm_capture_channel_config_t::io_loop_back` sets whether to enable the loop back mode. It is for debugging purposes only. It enables both the GPIO's input and output ability through the GPIO matrix peripheral.
The **`mcpwm_new_capture_channel()`** will return a pointer to the allocated capture channel object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no free capture channel left in the capture timer, this function will return **ESP_ERR_NOT_FOUND** error.
On the contrary, calling **`mcpwm_del_capture_channel()`** and **`mcpwm_del_capture_timer()`** function will free the allocated capture channel and timer object accordingly.

Capture_

The basic functionality of MCPWM capture is to record the time when any pulse edge of the capture signal turns active. Then you can get the pulse width and convert it into other physical quantity like distance or speed in the capture callback function. For example, in the BLDC (Brushless DC, see figure below) scenario, we can use the capture submodule to sense the rotor position from Hall sensor.



*MCPWM BLDC with Hall Sensor_*

The capture timer is usually connected with several capture channels, please refer to MCPWM Capture Timer and Channels for resource allocation.

Register Event Callbacks_

The MCPWM capture channel can inform the user when there's a valid edge detected on the signal. You have to register a callback function to get the timer count value of the capture moment, by calling `mcpwm_capture_channel_register_event_callbacks()`. The callback function prototype is declared in `mcpwm_capture_event_cb_t`. All

supported capture callbacks are listed in the `mcpwm_capture_event_callbacks_t`:
`mcpwm_capture_event_callbacks_t::on_cap` sets callback function for the capture channel when a valid edge is detected.

The callback function will provide event specific data of type `mcpwm_capture_event_data_t`, so that you can get the edge of the capture signal in `mcpwm_capture_event_data_t::cap_edge` and the count value of that moment in `mcpwm_capture_event_data_t::cap_value`. To convert the capture count into timestamp, you need to know the resolution of the capture timer by calling `mcpwm_capture_timer_get_resolution()`.
The callback function is called within the ISR context, so is should **not** attempt to block (e.g., make sure that only FreeRTOS APIs with `ISR` suffix is called within the function).

The parameter `user_data` of `mcpwm_capture_channel_register_event_callbacks()` function is used to save user's own context, it will be passed to the callback function directly.
This function will lazy install interrupt service for the MCPWM capture channel, whereas the service can only be removed in `mcpwm_del_capture_channel`.

Enable and Disable Capture Channel_

The capture channel is not enabled after allocation by `mcpwm_new_capture_channel()`. You should call `mcpwm_capture_channel_enable()` and `mcpwm_capture_channel_disable()` accordingly to enable or disable the channel. If the interrupt service is lazy installed during registering event callbacks for the channel in `mcpwm_capture_channel_register_event_callbacks()`, `mcpwm_capture_channel_enable()` will enable the interrupt service as well.

Enable and Disable Capture Timer_

Before doing IO control to the capture timer, user needs to enable the timer first, by calling `mcpwm_capture_timer_enable()`. Internally, this function will:
switch the capture timer state from **init** to **enable**.
acquire a proper power management lock if a specific clock source (e.g. APB clock) is selected. See also Power management for more information.

On the contrary, calling `mcpwm_capture_timer_disable()` will put the timer driver back to **init** state, and release the power management lock.

Start and Stop Capture Timer_

The basic IO operation of a capture timer is to start and stop.
Calling `mcpwm_capture_timer_start()` can start the timer and calling `mcpwm_capture_timer_stop()` can stop the timer immediately.

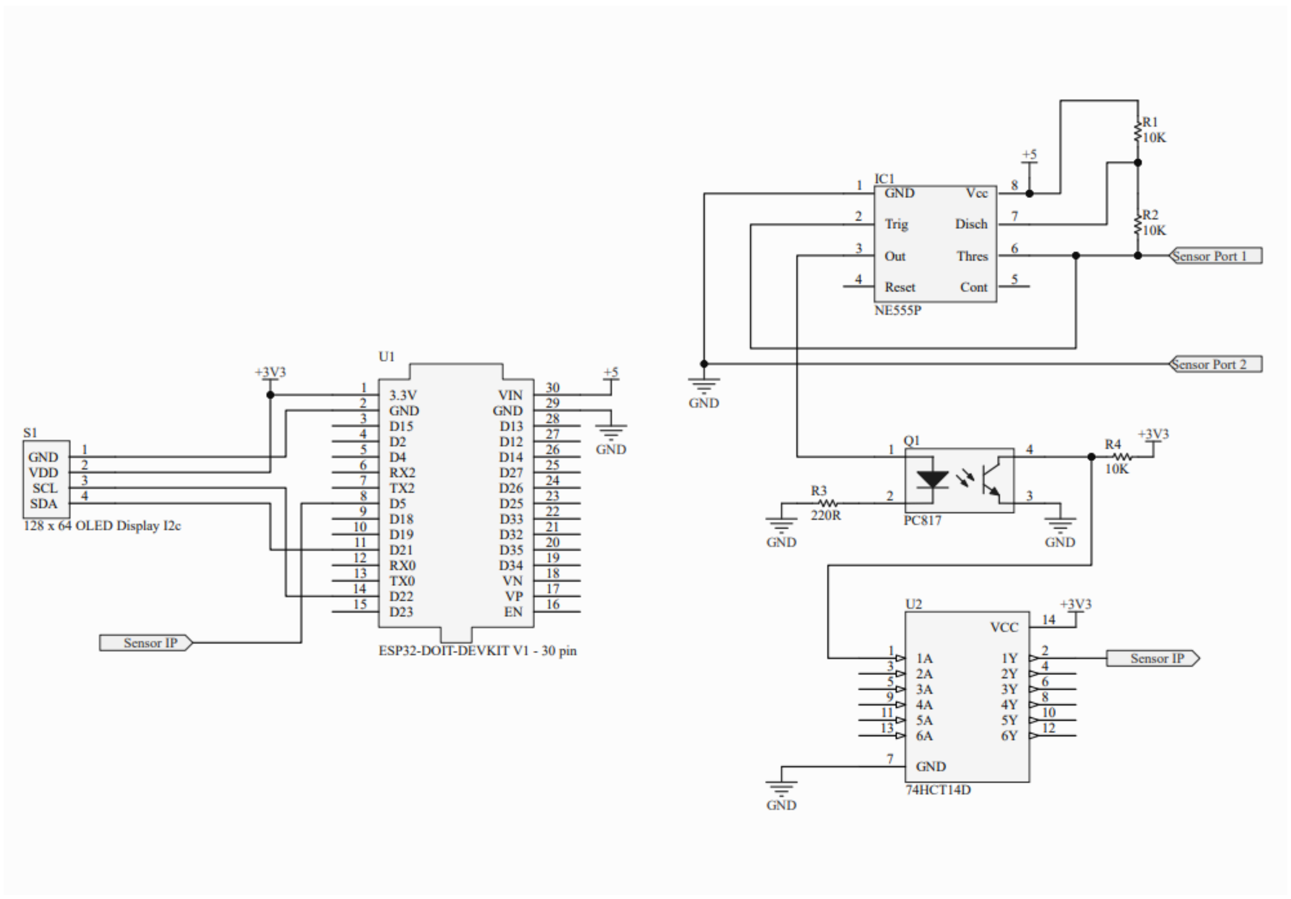
6.4.WI-FI

The **Wi-Fi** libraries provide support for configuring and monitoring the ESP32 Wi-Fi networking functionality. This includes configuration for:
Station mode (aka STA mode or **Wi-Fi** client mode). ESP32 connects to an access point.

AP mode (aka Soft-AP mode or Access Point mode). Stations connect to the ESP32.

Station/AP-coexistence mode (ESP32 is concurrently an access point and a station connected to another access point).
Various security modes for the above (WPA, WPA2, WEP, etc.)
Scanning for access points (active & passive scanning).
Promiscuous mode for monitoring of IEEE802.11 **Wi-Fi** packets.

7. COMPLETE CIRCUIT

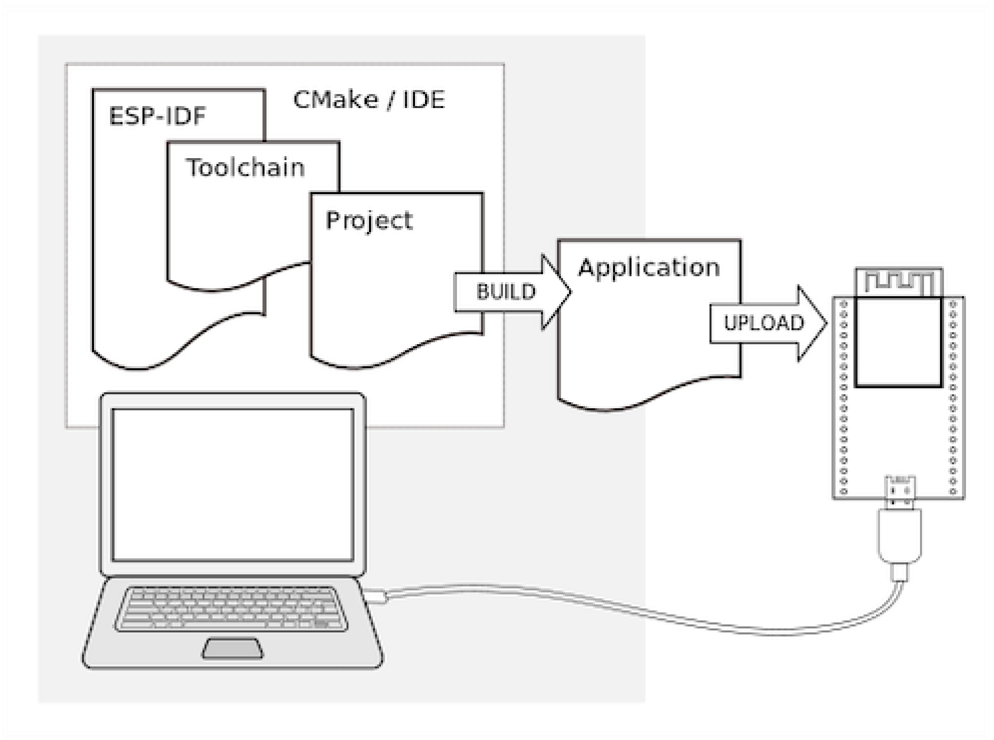


The complete circuit can be battery operated, with the ESP32 devkit being USB powered, supplying a 5V to the 55 timer, while the on-board regulator supplying 3.3V supply to the OLED display, PC817 opto coupler and 7414 inverter.

The capacitor plates are connected to the sensor port signals and a frequency signal is generated at pin 3 of the 555 timer circuit. This signal is level shifted and extended using the opto coupler and Schmitt−Trigger Inverter. The final processed sensor signal is available at pin 2 of the 7414 IC which is connected to pin 5 as digital input to the ESP32 controller.

ESP-IDF is the development framework for Espressif SoCs supported on Windows, Linux and macOS.

- **Toolchain** to compile code for ESP32
- **Build tools -** CMake and Ninja to build a full **Application** for ESP32
- **ESP-IDF** that essentially contains API (software libraries and source code) for ESP32 and scripts to operate the **Toolchain**



The major libraries used in the project are :

- Ethernet Library
- SSD1306 OLED library
- Open62541 Library

The sensor input signal is processed by the MCPWM capture module to measure the pulse width.

A port of the open62541 is used with the ESP32to implement the OPC UA server.

The ESP32 is configured for wifi connection and the following tags are implemented in the OPC server:

TCount: Read Only, Integer, displays the width of the frequency generator by the capacitive sensor. This is shown in counts.

Setpoint: Read Only, Integer, the set point beyond which the sensor would sense ash. The setpoint is updated when the calibrate event is initiated by setting the calibrate bit high.

Calibrate: Read/Write, Boolean, used to trigger a calibrate event. The bit is set on the OPC client. Once the bit is set, calibration starts and the bit is again set low.

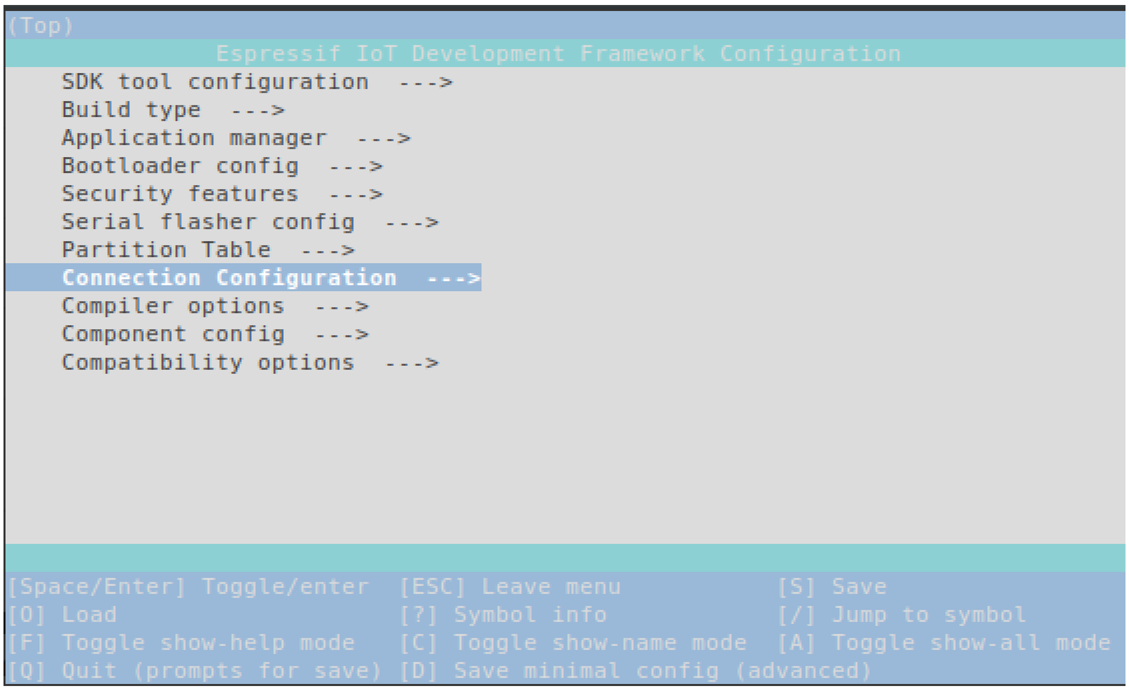
Ash Level Alarm: Read Only, Boolean, used to indicate ash level high. The tcount is compared with the setpoint and when exceeded, the alarm bit is set high.

7.1.Steps to establish the OPC UA client- server connection

Configure the ESP32 with "menuconfig" command.

- `(python) idf.py menuconfig`

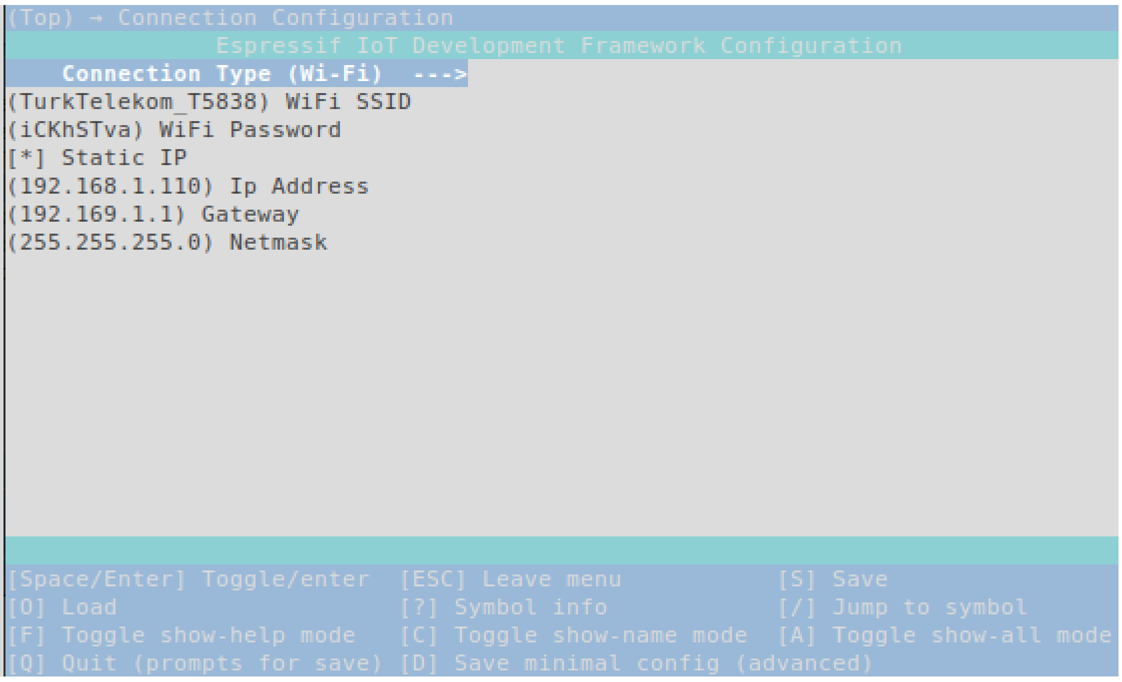
Navigate to "Connection Configuration" to set the type of connection (Wi-Fi or Ethernet), Wi-Fi SSID, password and static IP configuration. Currently static IP can only be set for ethernet.



```
(Top)
                Espressif IoT Development Framework Configuration
    SDK tool configuration  --->
    Build type  --->
    Application manager  --->
    Bootloader config  --->
    Security features  --->
    Serial flasher config  --->
    Partition Table  --->
    Connection Configuration  --->
    Compiler options  --->
    Component config  --->
    Compatibility options  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

ESP-IDF menuconfig

You can see the config options you can set for your OPC UA server under "Connection Configuration" as;



```
(Top) → Connection Configuration
                Espressif IoT Development Framework Configuration
    Connection Type (Wi-Fi)  --->
(TurkTelekom_T5838) WiFi SSID
(iCKhSTva) WiFi Password
[*] Static IP
(192.168.1.110) Ip Address
(192.169.1.1) Gateway
(255.255.255.0) Netmask

[Space/Enter] Toggle/enter   [ESC] Leave menu           [S] Save
[O] Load                     [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode    [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

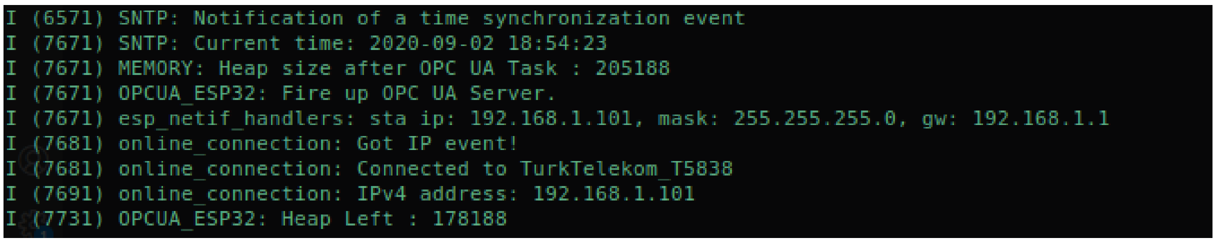
OPCUA-ESP32 Connection Configuration

5 – Flash the build to the board.

- `(python) idf.py` flash

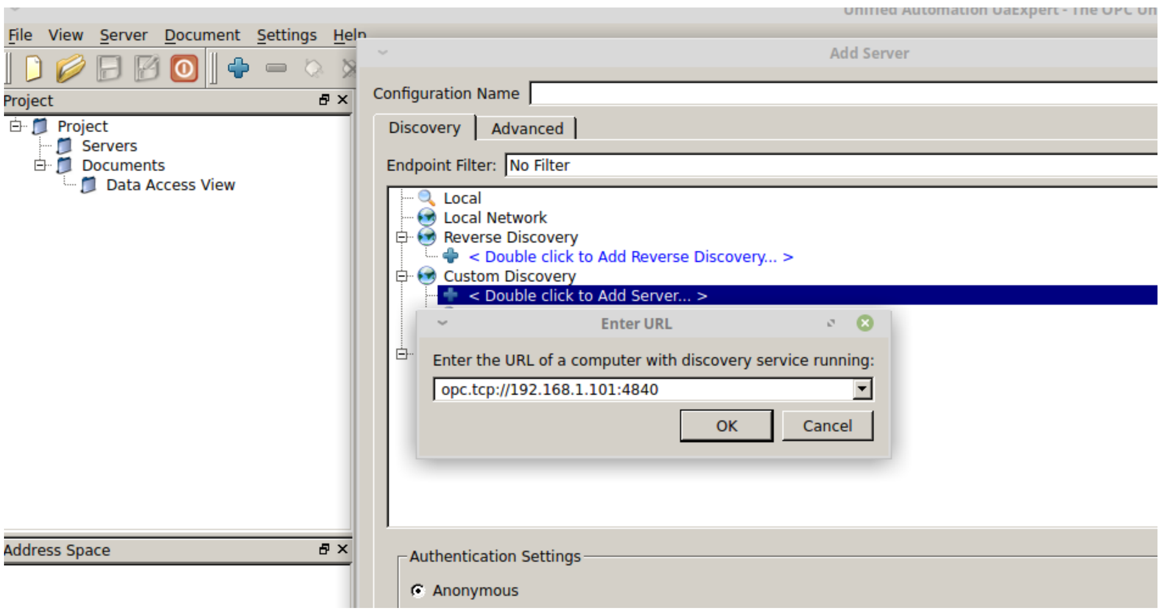
6 – Run monitor to check if the board got IP and started the OPC UA Server. This is required if you have not set an IP address to ESP32 board via DHCP or not connecting over ethernet with a static IP address.

- `(python) idf.py` monitor



```
I (6571) SNTP: Notification of a time synchronization event
I (7671) SNTP: Current time: 2020-09-02 18:54:23
I (7671) MEMORY: Heap size after OPC UA Task : 205188
I (7671) OPCUA_ESP32: Fire up OPC UA Server.
I (7671) esp_netif_handlers: sta ip: 192.168.1.101, mask: 255.255.255.0, gw: 192.168.1.1
I (7681) online_connection: Got IP event!
I (7681) online_connection: Connected to TurkTelekom_T5838
I (7691) online_connection: IPv4 address: 192.168.1.101
I (7731) OPCUA_ESP32: Heap Left : 178188
```

7 – Run an OPC UA Client to connect to the server. I have been using UAExpert from Unified Automation.



Configuration Name:

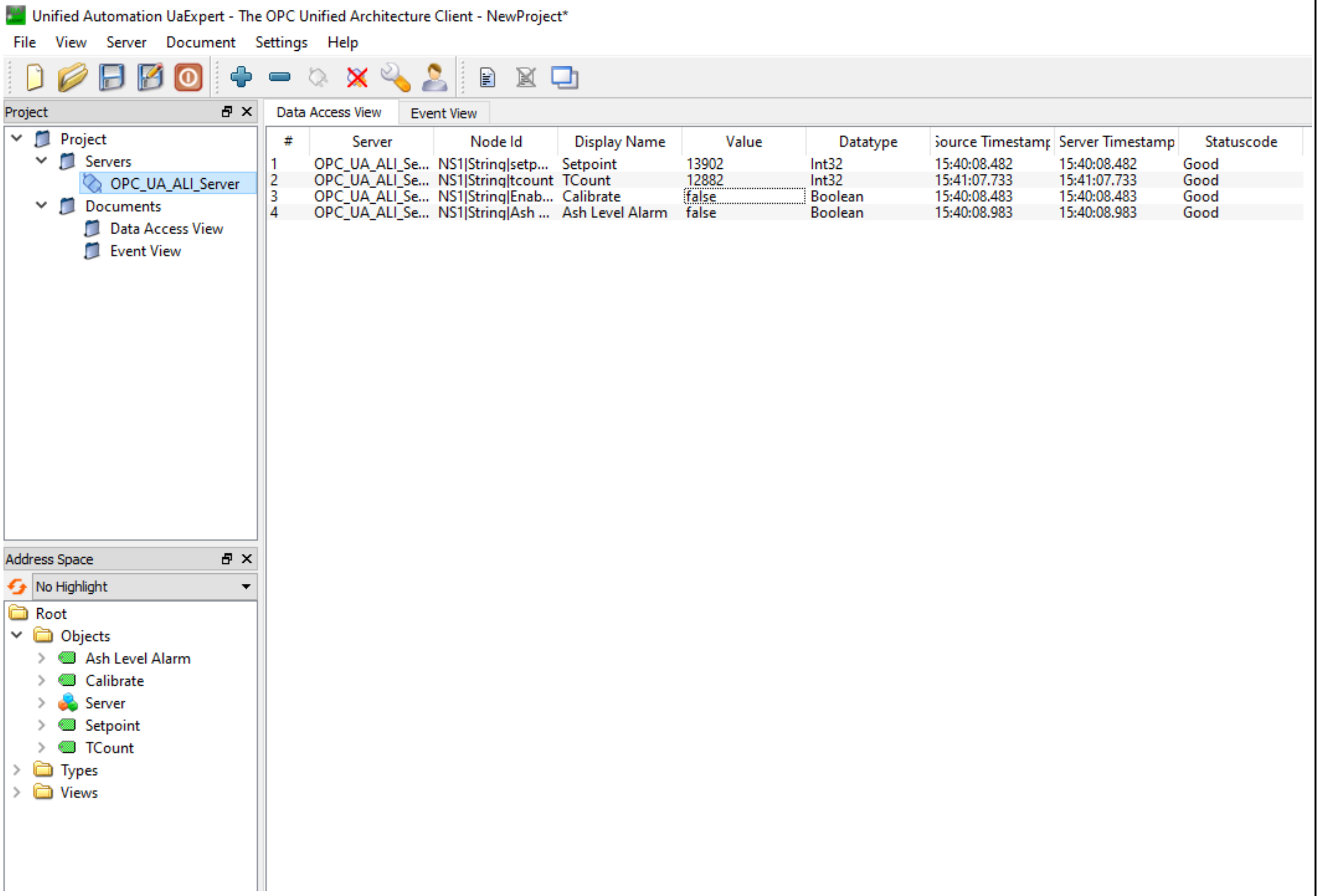
Endpoint Filter: No Filter

Enter the URL of a computer with discovery service running:

opc.tcp://192.168.1.101:4840

[✓ Anonymous]

After connecting properly, you will be able to browse/subscribe/control



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	OPC_UA_ALI_Se...	NS1|String|setp...	Setpoint	13902	Int32	15:40:08.482	15:40:08.482	Good
2	OPC_UA_ALI_Se...	NS1|String|tcount	TCount	12882	Int32	15:41:07.733	15:41:07.733	Good
3	OPC_UA_ALI_Se...	NS1|String|Enab...	Calibrate	false	Boolean	15:40:08.483	15:40:08.483	Good
4	OPC_UA_ALI_Se...	NS1|String|Ash ...	Ash Level Alarm	false	Boolean	15:40:08.983	15:40:08.983	Good

8. PROGRAM:

```
#include "opcua_esp32.h"

#include "esp_private/esp_clk.h"

#include "driver/mcpwm_cap.h"

#include "driver/gpio.h"

#include "driver/adc.h"

#define EXAMPLE_ESP_MAXIMUM_RETRY 10

#include "freertos/queue.h"

#include "freertos/task.h"

#define TAG "OPCUA_ESP32"

#define SNTP_TAG "SNTP"

#define MEMORY_TAG "MEMORY"

#define ENABLE_MDNS 1

#define TRIGGER_GPIO 18

#define INPUT_PIN 18

#define LED_PIN 2


#define GPIO_OUTPUT_IO_0    CONFIG_GPIO_OUTPUT_0

#define GPIO_OUTPUT_IO_1    CONFIG_GPIO_OUTPUT_1

#define GPIO_OUTPUT_PIN_SEL  ((1ULL<<GPIO_OUTPUT_IO_0) | (1ULL<<GPIO_OUTPUT_IO_1))

#define GPIO_INPUT_IO_0     CONFIG_GPIO_INPUT_0

#define GPIO_INPUT_IO_1     CONFIG_GPIO_INPUT_1

#define GPIO_INPUT_PIN_SEL  ((1ULL<<GPIO_INPUT_IO_0) | (1ULL<<GPIO_INPUT_IO_1))

#define ESP_INTR_FLAG_DEFAULT 0

#define HC_SR04_TRIG_GPIO  4


#define HC_SR04_ECHO_GPIO  5

const static char *TAG1 = "example";


bool ash_level_alarm = 0;

static bool obtain_time(void);

static void initialize_sntp(void);

int t_count =0;

int set_point =100;

UA_ServerConfig *config;
```

```
static UA_Boolean sntp_initialized = false;
static UA_Boolean running = true;
static UA_Boolean isServerCreated = false;
RTC_DATA_ATTR static int boot_count = 0;
static struct tm timeinfo;
static time_t now = 0;
bool sp;
SSD1306_t dev;
static uint8_t t_state = 0;
static UA_StatusCode
UA_ServerConfig_setUriName(UA_ServerConfig *uaServerConfig, const char *uri, const char *name)
{
  // delete pre-initialized values
  UA_String_clear(&uaServerConfig->applicationDescription.applicationUri);
  UA_LocalizedText_clear(&uaServerConfig->applicationDescription.applicationName);

  uaServerConfig->applicationDescription.applicationUri = UA_String_fromChars(uri);
  uaServerConfig->applicationDescription.applicationName.locale = UA_STRING_NULL;
  uaServerConfig->applicationDescription.applicationName.text = UA_String_fromChars(name);

  for (size_t i = 0; i < uaServerConfig->endpointsSize; i++)
  {
    UA_String_clear(&uaServerConfig->endpoints[i].server.applicationUri);
    UA_LocalizedText_clear(
      &uaServerConfig->endpoints[i].server.applicationName);

    UA_String_copy(&uaServerConfig->applicationDescription.applicationUri,
           &uaServerConfig->endpoints[i].server.applicationUri);

    UA_LocalizedText_copy(&uaServerConfig->applicationDescription.applicationName,
              &uaServerConfig->endpoints[i].server.applicationName);
  }

  return UA_STATUSCODE_GOOD;
}
```

```
static void opcua_task(void *arg)
{
  // BufferSize's got to be decreased due to latest refactorings in open62541 v1.2rc.
  UA_Int32 sendBufferSize = 16384;
  UA_Int32 recvBufferSize = 16384;
  ESP_ERROR_CHECK(esp_task_wdt_add(NULL));
  ESP_LOGI(TAG, "Fire up OPC UA Server.");
  UA_Server *server = UA_Server_new();
  UA_ServerConfig *config = UA_Server_getConfig(server);
  UA_ServerConfig_setMinimalCustomBuffer(config, 4840, 0, sendBufferSize, recvBufferSize);

  const char *appUri = "open62541.esp32.server";
  UA_String hostName = UA_STRING("opcua-esp32");
    UA_ServerConfig_setUriName(config, appUri, "OPC_UA_ALI_Server");
  UA_ServerConfig_setCustomHostname(config, hostName);

  /* Add Information Model Objects Here */
  addCurrentTCountDataSourceVariable(server);
  addSetpointDataSourceVariable(server);
  addCalibrateControlNode(server);
   addAlarmNode(server);

  ESP_LOGI(TAG, "Heap Left : %d", xPortGetFreeHeapSize());
  UA_StatusCode retval = UA_Server_run_startup(server);
  if (retval == UA_STATUSCODE_GOOD)
  {
    while (running)
    {
      UA_Server_run_iterate(server, false);
       vTaskDelay(100 / portTICK_PERIOD_MS);
      ESP_ERROR_CHECK(esp_task_wdt_reset());
      taskYIELD();
    }
    UA_Server_run_shutdown(server);
  }
  ESP_ERROR_CHECK(esp_task_wdt_delete(NULL));
}

void time_sync_notification_cb(struct timeval *tv)
{
  ESP_LOGI(SNTP_TAG, "Notification of a time synchronization event");
}

static void initialize_sntp(void)
{
  ESP_LOGI(SNTP_TAG, "Initializing SNTP");
  sntp_setoperatingmode(SNTP_OPMODE_POLL);
  sntp_setservername(0, "pool.ntp.org");
  sntp_setservername(1, "time.google.com");
  sntp_set_time_sync_notification_cb(time_sync_notification_cb);
  sntp_init();
  sntp_initialized = true;
}

static bool obtain_time(void)
{
  initialize_sntp();
  ESP_ERROR_CHECK(esp_task_wdt_add(NULL));
  memset(&timeinfo, 0, sizeof(struct tm));
  int retry = 0;
  const int retry_count = 10;
  while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry <= retry_count)
  {
    ESP_LOGI(SNTP_TAG, "Waiting for system time to be set... (%d/%d)", retry, retry_count);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
    ESP_ERROR_CHECK(esp_task_wdt_reset());
  }
  time(&now);
  localtime_r(&now, &timeinfo);
  ESP_ERROR_CHECK(esp_task_wdt_delete(NULL));
  return timeinfo.tm_year > (2016 - 1900);
}
```

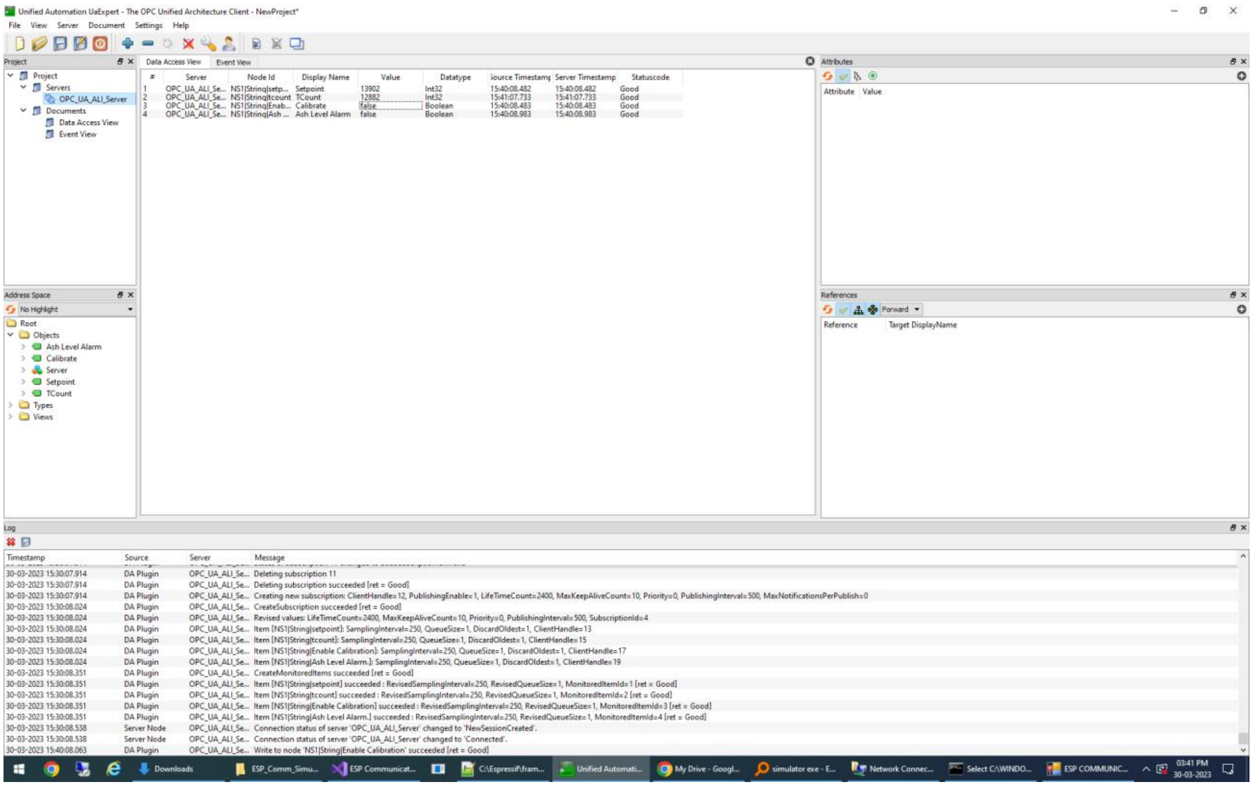
```
static void opc_event_handler(void *arg, esp_event_base_t event_base,
                        int32_t event_id, void *event_data)
{

   if (sntp_initialized != true)
    {
        if (timeinfo.tm_year < (2016 - 1900))
        {
            ESP_LOGI(SNTP_TAG, "Time is not set yet. Settting up network connection and getting time over NTP.");
            if (!obtain_time())
            {
                ESP_LOGE(SNTP_TAG, "Could not get time from NTP. Using default timestamp.");
            }
            time(&now);
        }
        localtime_r(&now, &timeinfo);
        ESP_LOGI(SNTP_TAG, "Current time: %d-%02d-%02d %02d:%02d:%02d", timeinfo.tm_year + 1900,
timeinfo.tm_mon + 1, timeinfo.tm_mday, timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);
    }

    if (!isServerCreated)
    {
        xTaskCreatePinnedToCore(opcua_task, "opcua_task", 24336, NULL, 10, NULL, 1);
        ESP_LOGI(MEMORY_TAG, "Heap size after OPC UA Task : %d", esp_get_free_heap_size());
        isServerCreated = true;
    }

}

static void disconnect_handler(void *arg, esp_event_base_t event_base,
                        int32_t event_id, void *event_data)
{

}
```

```
static void connection_scan(void)
{


            ESP_ERROR_CHECK(esp_netif_init());
   ESP_ERROR_CHECK(esp_event_loop_create_default());
   ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT, GOT_IP_EVENT, &opc_event_handler, NULL));
   ESP_ERROR_CHECK(esp_event_handler_register(BASE_IP_EVENT, DISCONNECT_EVENT, &disconnect_handler,
NULL));
   ESP_ERROR_CHECK(example_connect());


}


static bool tcount_callback(mcpwm_cap_channel_handle_t cap_chan, const mcpwm_capture_event_data_t
*edata, void *user_data)
{
   static uint32_t cap_val_begin_of_sample = 0;
   static uint32_t cap_val_end_of_sample = 0;
   TaskHandle_t task_to_notify = (TaskHandle_t)user_data;
   BaseType_t high_task_wakeup = pdFALSE;


   //calculate the interval in the ISR,
   //so that the interval will be always correct even when capture_queue is not handled in time and overflow.
   if (edata->cap_edge == MCPWM_CAP_EDGE_POS) {
      // store the timestamp when pos edge is detected
      cap_val_begin_of_sample = edata->cap_value;
      cap_val_end_of_sample = cap_val_begin_of_sample;


   } else {
      cap_val_end_of_sample = edata->cap_value;
      uint32_t tof_ticks = cap_val_end_of_sample - cap_val_begin_of_sample;
      // notify the task to calculate the distance
      xTaskNotifyFromISR(task_to_notify, tof_ticks, eSetValueWithOverwrite, &high_task_wakeup);
   }


   return high_task_wakeup == pdTRUE;
}
```

```
void app_main(void)
{
            ++boot_count;



  // Workaround for CVE-2019-15894
  nvs_flash_init();
  if (esp_flash_encryption_enabled())
  {
    esp_flash_write_protect_crypt_cnt();
  }

  esp_err_t ret = nvs_flash_init();
  if (ret == ESP_ERR_NVS_NO_FREE_PAGES)
  {
    ESP_ERROR_CHECK(nvs_flash_erase());
    ESP_ERROR_CHECK(nvs_flash_init());
  }

        i2c_master_init(&dev, 21, 22, 15);
        ssd1306_init(&dev, 128, 64);
  ssd1306_clear_screen(&dev, false);
        ssd1306_contrast(&dev, 0xff);
        ssd1306_display_text(&dev, 0, "Initialising...", 15, false);



  connection_scan();
        //   ESP_LOGI(TAG, "Install capture timer");
  mcpwm_cap_timer_handle_t cap_timer = NULL;
  mcpwm_capture_timer_config_t cap_conf = {
    .clk_src = MCPWM_CAPTURE_CLK_SRC_DEFAULT,
    .group_id = 0,
  };
  ESP_ERROR_CHECK(mcpwm_new_capture_timer(&cap_conf, &cap_timer));

 //  ESP_LOGI(TAG, "Install capture channel");
 mcpwm_cap_channel_handle_t cap_chan = NULL;
 mcpwm_capture_channel_config_t cap_ch_conf = {
    .gpio_num = HC_SR04_ECHO_GPIO,
    .prescale = 1,
    // capture on both edge
    .flags.neg_edge = true,
    .flags.pos_edge = true,
    // pull up internally
    .flags.pull_up = true,
 };
 ESP_ERROR_CHECK(mcpwm_new_capture_channel(cap_timer, &cap_ch_conf, &cap_chan));

// ESP_LOGI(TAG, "Register capture callback");
 TaskHandle_t cur_task = xTaskGetCurrentTaskHandle();
 mcpwm_capture_event_callbacks_t cbs = {
    .on_cap = tcount_callback,
 };
 ESP_ERROR_CHECK(mcpwm_capture_channel_register_event_callbacks(cap_chan, &cbs, cur_task));

// ESP_LOGI(TAG, "Enable capture channel");
 ESP_ERROR_CHECK(mcpwm_capture_channel_enable(cap_chan));

//ESP_LOGI(TAG, "Enable and start capture timer");
 ESP_ERROR_CHECK(mcpwm_capture_timer_enable(cap_timer));
 ESP_ERROR_CHECK(mcpwm_capture_timer_start(cap_timer));
            char buf[20];
 uint32_t tof_ticks, len , t_count_old;
            t_count = 0;
            set_point = 5000;
            ssd1306_display_text(&dev, 4, "Setpoint:   5000",16, false);
            sp =0;
 while (1) {
    // trigger the sensor to start a new sample
    //gen_trig_output();
    // wait for echo done signal
    if (xTaskNotifyWait(0x00, ULONG_MAX, &tof_ticks, pdMS_TO_TICKS(1000)) == pdTRUE) {
      int pulse_width_us = tof_ticks * (1000000.0 / esp_clk_apb_freq());
      // convert the pulse width into measure distance
                        if(pulse_width_us > 300)
                            t_count = pulse_width_us;
                        if(t_count > set_point)
                        {       ash_level_alarm = 1;
                        ssd1306_display_text(&dev, 6, "Ash Level : High", 16, false);
                        }
                        else
                        {       ash_level_alarm = 0;
                        ssd1306_display_text(&dev, 6, "Ash Level : Low ", 16, false);
                        }
                        sprintf(buf, "Tcount:%7d",t_count);
                        len = strlen(buf);

                        ssd1306_display_text(&dev, 2, buf, len, false);
     }
                        set_point = t_count + 1000;
                        sprintf(buf, "Setpoint:%7d",set_point);
                        len = strlen(buf);
                        ssd1306_display_text(&dev, 4, buf, len, false);
                        sp=0;
                  }
                  // ESP_LOGI(TAG, "Heap Left : %d", xPortGetFreeHeapSize());
     vTaskDelay(pdMS_TO_TICKS(500));
   }

}
```

9. CONCLUSION:

The ash level indicator system is implemented with a capacitive sensor for measuring ash at the field. The sensor signal from the field is processed in the controller and the data sent over OPC UA by implementing a server over wifi. The client device connects to the device and ash level indication, calibration and level status are viewed at the client display.



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	OPC_UA_ALI_Se...	NS1|String|setp...	Setpoint	13902	Int32	15:40:08.482	15:40:08.482	Good
2	OPC_UA_ALI_Se...	NS1|String|tcount	TCount	12882	Int32	15:41:07.733	15:41:07.733	Good
3	OPC_UA_ALI_Se...	NS1|String|Enab...	Calibrate	false	Boolean	15:40:08.483	15:40:08.483	Good
4	OPC_UA_ALI_Se...	NS1|String|Ash ...	Ash Level Alarm	false	Boolean	15:40:08.983	15:40:08.983	Good

```
Timestamp                Source       Server             Message
30-03-2023 15:30:07.914  DA Plugin    OPC_UA_ALI_Se...   Deleting subscription 11
30-03-2023 15:30:07.914  DA Plugin    OPC_UA_ALI_Se...   Deleting subscription succeeded [ret = Good]
30-03-2023 15:30:07.914  DA Plugin    OPC_UA_ALI_Se...   Creating new subscription: ClientHandle=12, PublishingEnable=1, LifeTimeCount=2400, MaxKeepAliveCount=10, Priority=0, PublishingInterval=500, MaxNotificationsPerPublish=0
30-03-2023 15:30:08.024  DA Plugin    OPC_UA_ALI_Se...   CreateSubscription succeeded [ret = Good]
30-03-2023 15:30:08.024  DA Plugin    OPC_UA_ALI_Se...   Revised values: LifeTimeCount=2400, MaxKeepAliveCount=10, Priority=0, PublishingInterval=500, SubscriptionId=4
30-03-2023 15:30:08.024  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|setpoint]: SamplingInterval=250, QueueSize=1, DiscardOldest=1, ClientHandle=13
30-03-2023 15:30:08.024  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|tcount]: SamplingInterval=250, QueueSize=1, DiscardOldest=1, ClientHandle=15
30-03-2023 15:30:08.024  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|Enable Calibration]: SamplingInterval=250, QueueSize=1, DiscardOldest=1, ClientHandle=17
30-03-2023 15:30:08.024  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|Ash Level Alarm.]: SamplingInterval=250, QueueSize=1, DiscardOldest=1, ClientHandle=19
30-03-2023 15:30:08.351  DA Plugin    OPC_UA_ALI_Se...   CreateMonitoredItems succeeded [ret = Good]
30-03-2023 15:30:08.351  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|setpoint] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=1 [ret = Good]
30-03-2023 15:30:08.351  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|tcount] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=2 [ret = Good]
30-03-2023 15:30:08.351  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|Enable Calibration] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=3 [ret = Good]
30-03-2023 15:30:08.351  DA Plugin    OPC_UA_ALI_Se...   Item [NS1|String|Ash Level Alarm.] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=4 [ret = Good]
30-03-2023 15:30:08.538  Server Node  OPC_UA_ALI_Se...   Connection status of server 'OPC_UA_ALI_Server' changed to 'NewSessionCreated'.
30-03-2023 15:30:08.538  Server Node  OPC_UA_ALI_Se...   Connection status of server 'OPC_UA_ALI_Server' changed to 'Connected'.
30-03-2023 15:40:08.063  DA Plugin    OPC_UA_ALI_Se...   Write to node 'NS1|String|Enable Calibration' succeeded [ret = Good]
```