**Group no 8**

# Vehicle Automation System Using Communication Protocols

Submitted in partial fulfillment for the award of

**Post Graduate Diploma in EMBEDDED SYSTEMS AND DESIGN**

From C-DAC, ACTS (Pune)

**Guided by:**

**Mr. Tarun Bharani** **Presented by:**

| | |
|---|---|
| **Mrs. Khushi Singh** | **240340130025** |
| **Mr. Mayur Wagh** | **240340130029** |
| **Mr. Rushi Parekh** | **240340130037** |
| **Mrs. Shikha Verma** | **240340130040** |
| **Mr. Bipin Kumar** | **240340130053** |

**Centre for Development of Advanced Computing (C-DAC), Pune**

# CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

| | |
|---|---|
| **Mrs. Khushi Singh** | **240340130025** |
| **Mr. Mayur Wagh** | **240340130029** |
| **Mr. Rushi Parekh** | **240340130037** |
| **Mrs. Shikha Verma** | **240340130040** |
| **Mr. Bipin Kumar** | **240340130053** |

Have successfully completed their project on

**Vehicle Automation System Using Communication Protocols**

Under the guidance of

**Mr. Tarun Bharani**

**Project Guide** **Project Supervisor**

## Acknowledgement

The success and final outcome of this project "**Vehicle Automation System Using Communication Protocols"** required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along with the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I am highly indebted to **Mr. Tarun Bharani** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We take this opportunity to thank **Ms. Risha P.R.** and all staff members for the cooperation provided by them in many ways. It is our great pleasure in expressing sincere and deep gratitude toward **Ms. Namrata Ailawar** for her valuable guidance and constant support throughout this work. I would like to express my gratitude towards my parents & member of CDAC ACTS, Pune for their kind co-operation and encouragement which help me in the completion of this project.

My most heartfelt thanks go to **Mrs. Srujana Bhamindi** (Course Co-Ordinator, PG-DESD) who gave all the required support to me.

My thanks and appreciations also go to my batchmates in developing the project and people who have willingly helped me out with their abilities.

From:

| | |
|---|---|
| Mrs. Khushi Singh | 240340130025 |
| Mr. Mayur Wagh | 240340130029 |
| Mr. Rushi Parekh | 240340130037 |
| Mrs. Shikha Verma | 240340130040 |
| Mr. Bipin Kumar | 240340130053 |

## Abstract

This paper presents an embedded system designed to automatically hold a vehicle on inclined surfaces.

The system leverages an STM32F407VGT6 microcontroller and a Real-Time Operating System (RTOS) to manage system tasks efficiently.
A CAN bus facilitates communication between a master and slave microcontroller.

Vehicle inclination is determined by the x-axis acceleration measured by a LIS3DSH accelerometer, while the desired engine speed is set via a variable potentiometer.

Upon simultaneous satisfaction of inclination and RPM conditions,
the master microcontroller activates the auto-hold function and transmits the command to the slave microcontroller over the CAN bus.

The slave microcontroller, orchestrated by the RTOS, prioritizes tasks based on the received CAN message.

An OLED display, interfaced via I2C, provides visual confirmation of the auto-hold status.

The system subsequently engages the brake system based on the auto-hold condition.

# Contents

# List of Figures

# Chapter 1 INTRODUCTION

## 1.1 About Project:

1. This paper presents an embedded system designed to automatically hold a vehicle on inclined surfaces.
2. The system utilizes an STM32F407VGT6 microcontroller and a Real-Time Operating System (RTOS) for efficient task management.
3. A CAN bus facilitates communication between a master and slave microcontroller.
4. Vehicle inclination is determined by the x-axis acceleration measured by a LIS3DSH accelerometer.
5. The desired engine speed is set via a variable potentiometer.
6. Upon simultaneous satisfaction of inclination and RPM conditions, the master microcontroller activates the auto-hold
7. function and transmits a command to the slave microcontroller over the CAN bus.

## 1.2 Project scope:

## 1.3 System Requirements

**Hardware Requirements:**

Hardware requirements are as followed:

1. Potentiometer B10K
2. 3-axis Linear Accelerometer LIS3DSH
3. STM32F403 Discovery Board
4. SN65HVD230 CAN Transceivers
5. OLED Display

**Software Requirements:**

The software used for implementing the project was STM32CubeIDE 1.28.0 for STM32F407VGT6 Discovery Board.

## Testing Environment

STM32CubeIDE is an all-in-one multi-OS development tool, which is part of the STM32Cube software ecosystem.

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is based on the Eclipse®/CDT™ framework and GCC toolchain for the development, and GDB for the debugging. It allows the integration of the hundreds of existing plugins that complete the features of the Eclipse® IDE.

STM32CubeIDE integrates STM32 configuration and project creation functionalities from STM32CubeMX to offer all-in-one tool experience and save installation and development time. After the selection of an empty STM32 MCU or MPU, or preconfigured microcontroller or microprocessor from the selection of a board or the selection of an example, the project is created and initialization code generated. At any time during the development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code with no impact on the user code.

STM32CubeIDE includes build and stack analysers that provide the user with useful information about project status and memory requirements.

STM32CubeIDE also includes standard and advanced debugging features including views of CPU core registers, memories, and peripheral registers, as well as live variable watch, Serial Wire Viewer interface, or fault analyser.

## All features

- Integration of services from STM32CubeMX:STM32 microcontroller, microprocessor, development platform and example project selection Pinout, clock, peripheral, and middleware configuration Project creation and generation of the initialization code Software and middleware completed with enhanced STM32Cube Expansion Packages
- Based on Eclipse®/CDT™, with support for Eclipse® add-ons, GNU C/C++ for Arm® toolchain and GDB debugger
- STM32MP1 Series: Support for Opens Linux projects: Linux Support for Linux
- Additional advanced debug features including: CPU core, peripheral register, and memory views Live variable watch view System analysis and real-time tracing (SWV)CPU fault analysis toolrooms-aware debug support including Azure
- Support for ST-LINK (STMicroelectronics) and J-Link (SEGGER) debug probes
- Import project from Atollic® TrueSTUDIO® and AC6 System Workbench for STM32 (SW4STM32)
- Multi-OS support: Windows®, Linux®, and macOS®, 64-bit versions only

## 6.1 Hardware

### STM32F407 Discovery Board

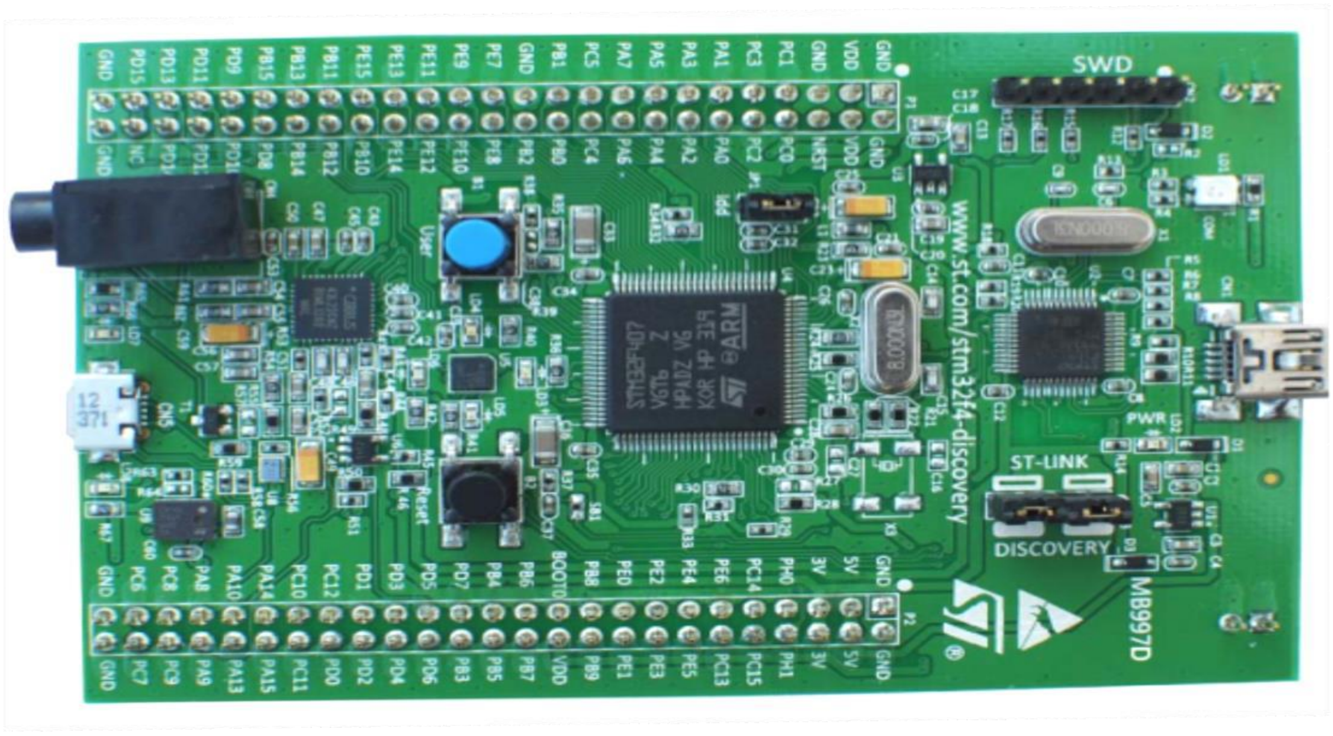


Fig .4 STM32F407 Discovery board

The STM32F405xx and STM32F407 Discovery family is based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 168 MHz The Cortex-M4 core features a Floating-point unit (FPU) single precision which supports all Arm single precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security. The STM32F405xx and STM32F407 Discovery family incorporates high-speed embedded memories (Flash memory up to 1 Mbyte, up to 192 Kbytes of SRAM), up to 4 Kbytes of backup SRAM, and an extensive range of enhanced I/OS and peripherals connected to two APB buses, three AHB buses and a 32-bit multi-AHB bus matrix.

The STM32F4DISCOVERY offers the following features:

- STM32F407 DISCOVERYVGT6 microcontroller featuring 32-bit Arm®(a) Cortex®-M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package
- USB OTG FS
- ST MEMS 3-axis accelerometer
- ST-MEMS audio sensor omni-directional digital microphone
- Audio DAC with integrated class D speaker driver
- User and reset push-buttons
- Eight LEDs:

— LD1 (red/green) for USB communication

— LD2 (red) for 3.3 V power on

— Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)

— Two USB OTG LEDs, LD7 (green) VBUS and LD8 (red) overcurrent

- Board connectors:
- USB with Micro-AB
- Stereo headphone output jack
- 2.54 mm pitch extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing.
- External application power supply: 3 V and 5 V
- Flexible power-supply options: ST-LINK, USB VBUS, or external sources.
- Comprehensive free software including a variety of examples, part of STM32CubeF4 MCU Package, or STSW-STM32068 for using legacy standard libraries
- On-board ST-LINK/V2-A debugger/programmer with USB renumeration capability: mass storage, Virtual COM port, and debug port
- Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE

# Chapter 2 LITERATURE SURVEY

## 2.1Autohold System

To effectively develop an autohold system, a thorough understanding of existing technologies and research is imperative.
The literature survey should encompass several key areas.

Firstly, the investigation should focus on embedded systems specifically designed for automotive applications.
The suitability of STM32 microcontrollers and the role of Real-Time Operating Systems (RTOS) in managing complex automotive tasks should be examined.
Understanding the successful integration of these components into previous systems will provide valuable insights into hardware and software selections.

Secondly, the literature should explore sensor fusion techniques. Since the proposed system relies on accelerometer data for inclination measurement,
effective combination of this data with inputs from other vehicle sensors is crucial.
Studies on how sensor fusion algorithms enhance system performance and accuracy should be analyzed.

Thirdly, the role of CAN bus in automotive communication networks must be investigated.
Understanding its capabilities, limitations, and real-time performance characteristics is essential for ensuring
reliable data exchange between the system components.

Furthermore, the design and implementation of human-machine interfaces (HMIs) for automotive applications should be explored.
The effectiveness of different HMI approaches in conveying autohold status information to the driver needs to be evaluated.

Furthermore, the design and implementation of human-machine interfaces (HMIs) for automotive applications should be explored.
The effectiveness of different HMI approaches in conveying autohold status information to the driver needs to be evaluated.

Finally, a comprehensive review of existing autohold systems is necessary.
This includes understanding the underlying control algorithms, system architectures, and their integration with other vehicle subsystems.

By analyzing previous research, potential challenges, and limitations can be identified, leading to innovative solutions for the proposed autohold system.

Through this in-depth literature review, a solid foundation will be established for the development of a robust and efficient autohold system.

The slave microcontroller, orchestrated by the RTOS, prioritizes tasks based on the received CAN message.
 An OLED display, interfaced via I2C, provides visual confirmation of the autohold status.
The system subsequently engages the brake system based on the autohold condition.

# Chapter 3

# SYSTEM DESIGN

## 3.1 Implementation

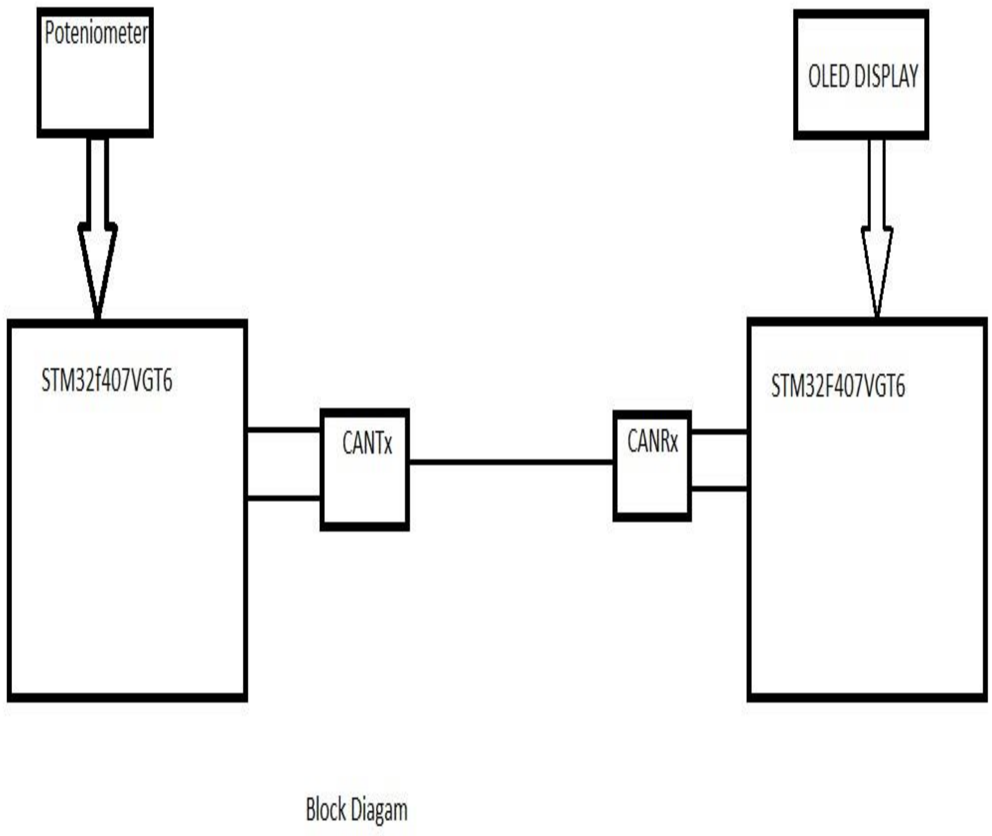


**Fig 3.1.1 Block Diagram Of Vehicle automation system using communication protocols**

## 3.2 Circuit Diagram:

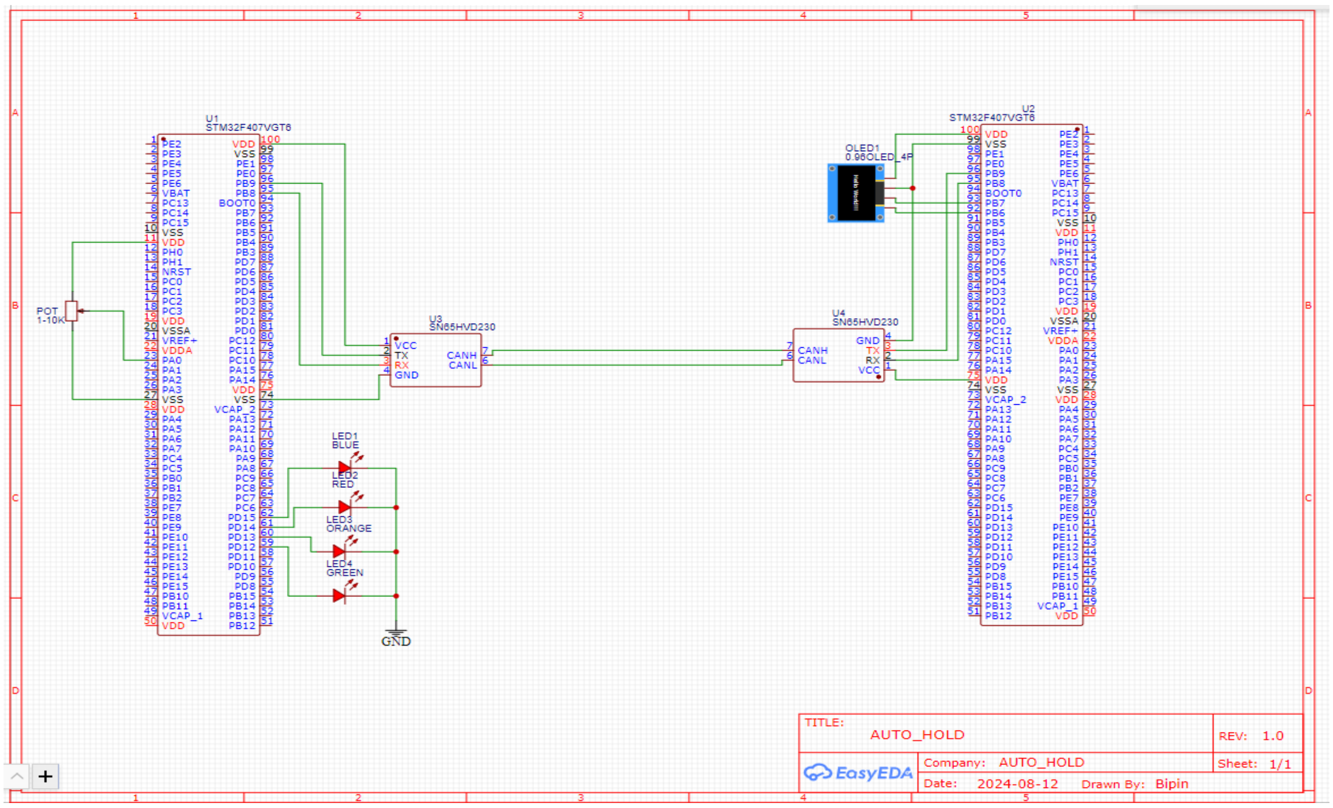


**Fig 3.2.1 Circuit Diagram Of Vehicle Automation System Using Communication Protocols**

## 3.3 1.3inch I2C Display

**SPECIFICATIONS -**

- Use CHIP No.SH1106
- Use 3.3V-5V POWER SUPPLY
- Graphic LCD 1.3” in width with 128x64 Dot Resolution
- White Display is used for the model OLED 1.3 I2C WHITE and blue Display is used for the model OLED 1.3 I2C BLUE
- Use I2C Interface
- Directly connect signal to Microcontroller 3.3V and 5V without connecting through Voltage Regulator Circuit

- Total Current when running together is 8 mA - PCB Size: 33.7 mm x 35.5 mm

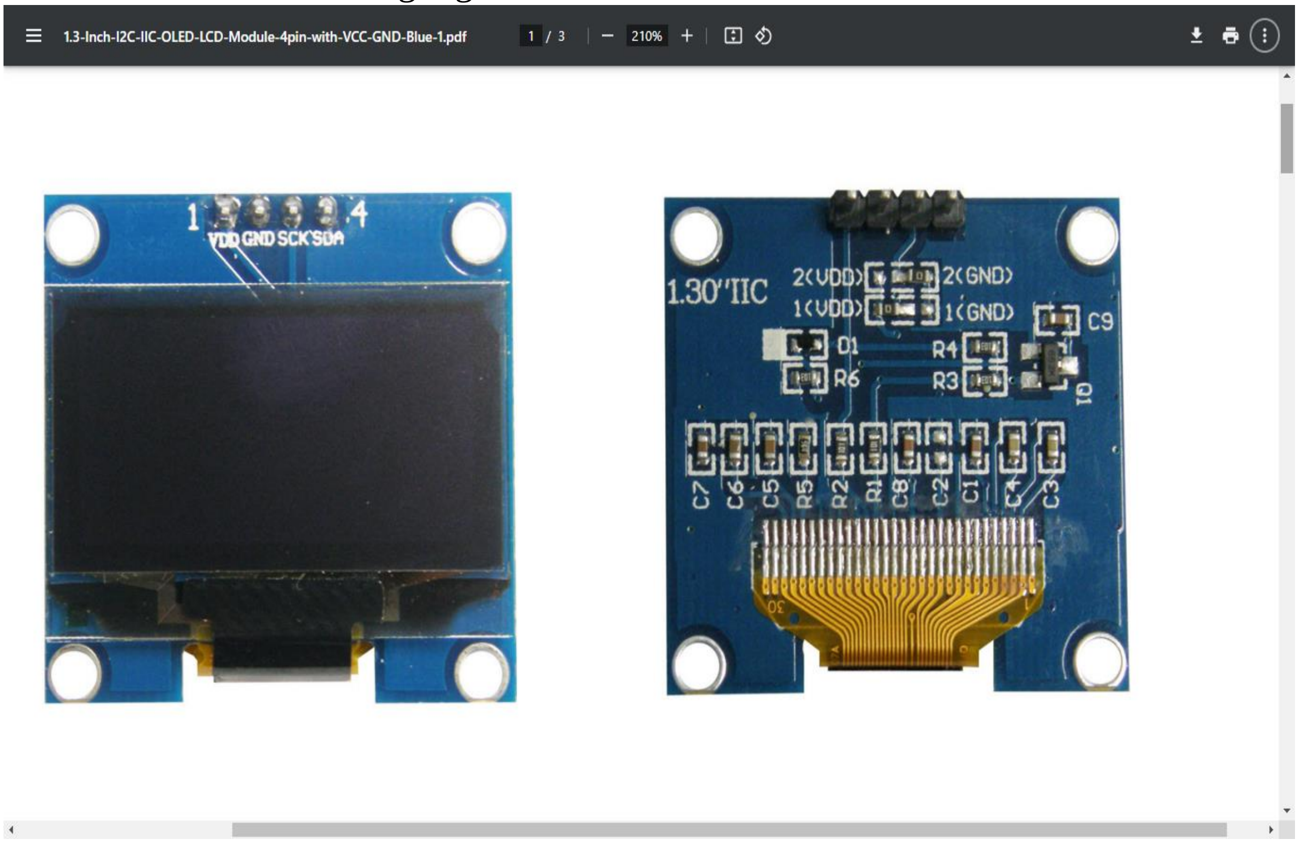

1.3-Inch-I2C-IIC-OLED-LCD-Module-4pin-with-VCC-GND-Blue-1.pdf
1 / 3
210%
1
4
VDD GND SCK SDA
1.30"IIC
2(VDD)
2(GND)
1(VDD)
1(GND)
C9
D1
R4
R6
R3
Q1
C7 C6 C5 R5 R2 R1 C8 C2 C1 C4 C3

**Table shows name and function of Pin OLED**

1. VDD Pin Power Supply for LCD, using 3.3V-5V
2. GND Pin Ground
3. SCK Pin SCL of I2C Interface
4. SDA Pin SDA of I2C Interface is basically a device that measures strain and then converts force into electric energy which serves as a measurement for scientists and workers.

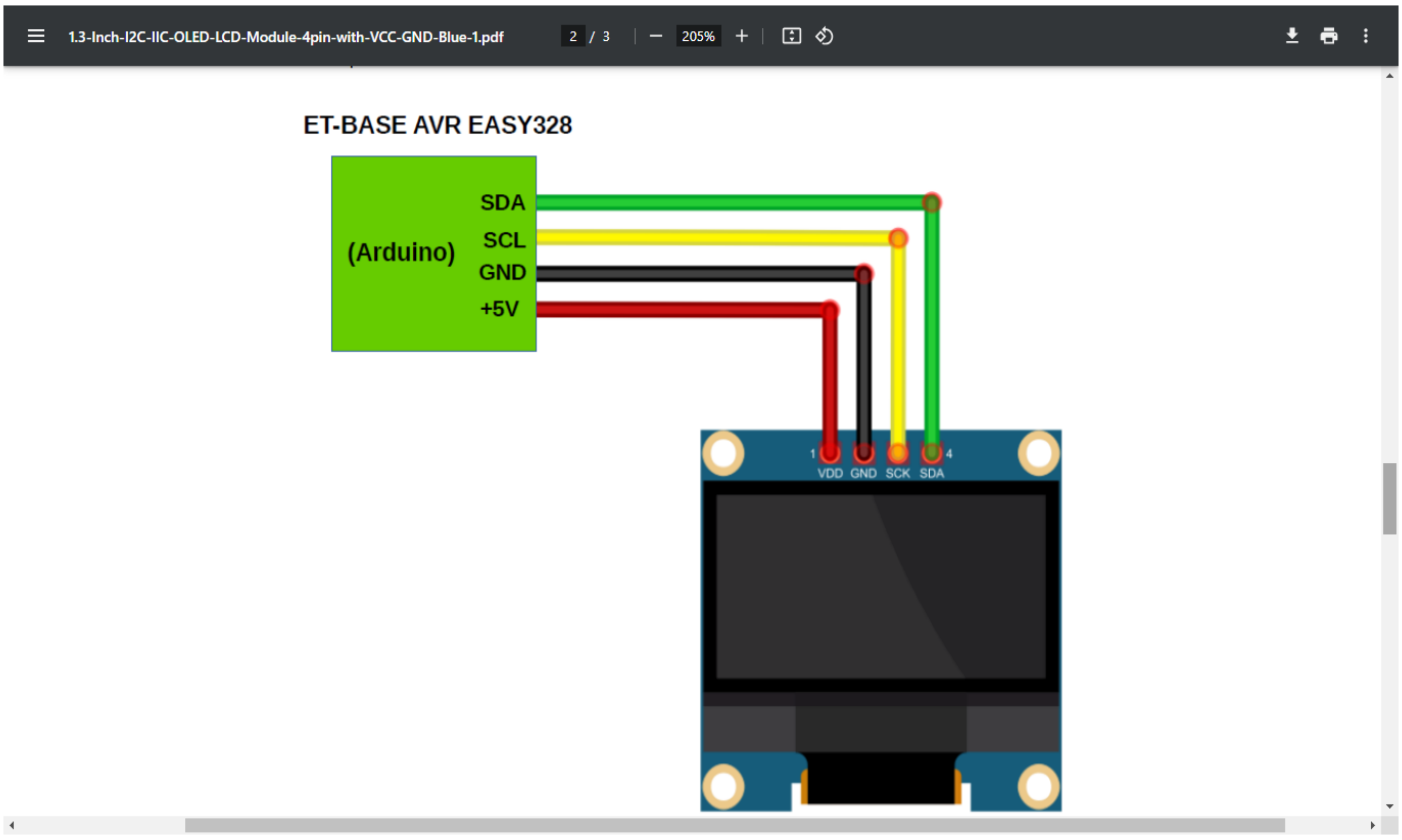


**Fig 3.3.1 Wire Circuit as shown in the picture below**

Fig 3.3.1 Load

**Where to apply weight on load cell?**

You can see an arrow is shown on the Load cell. This arrow shows the direction of the force on the load cell. The arrangements can be made as shown in the figure using metal strips. Attach metal strip on the Load cell using bolts.

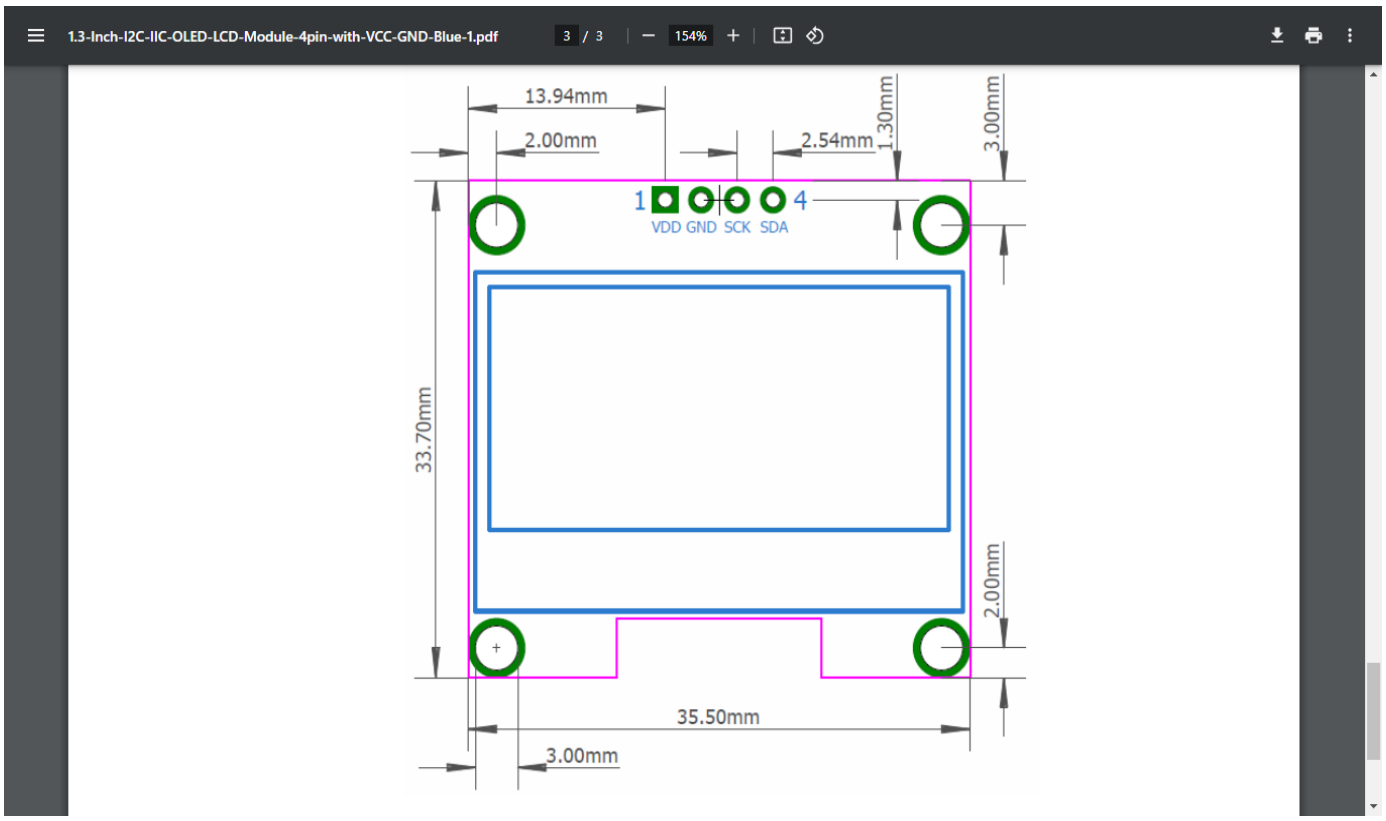
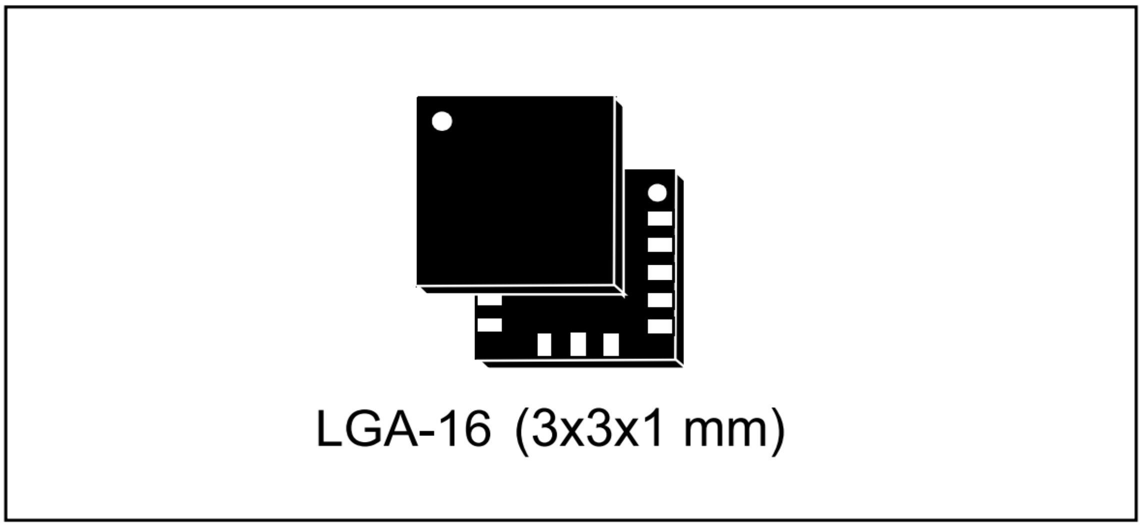


**Fig 3.3.2 Diagram shows dimensions of Module OLED 1.3 LED**

MEMS digital output motion sensor: ultra-low-power high-performance three-axis"nano"accelerometer **[LIS3DSH]**



**Features**

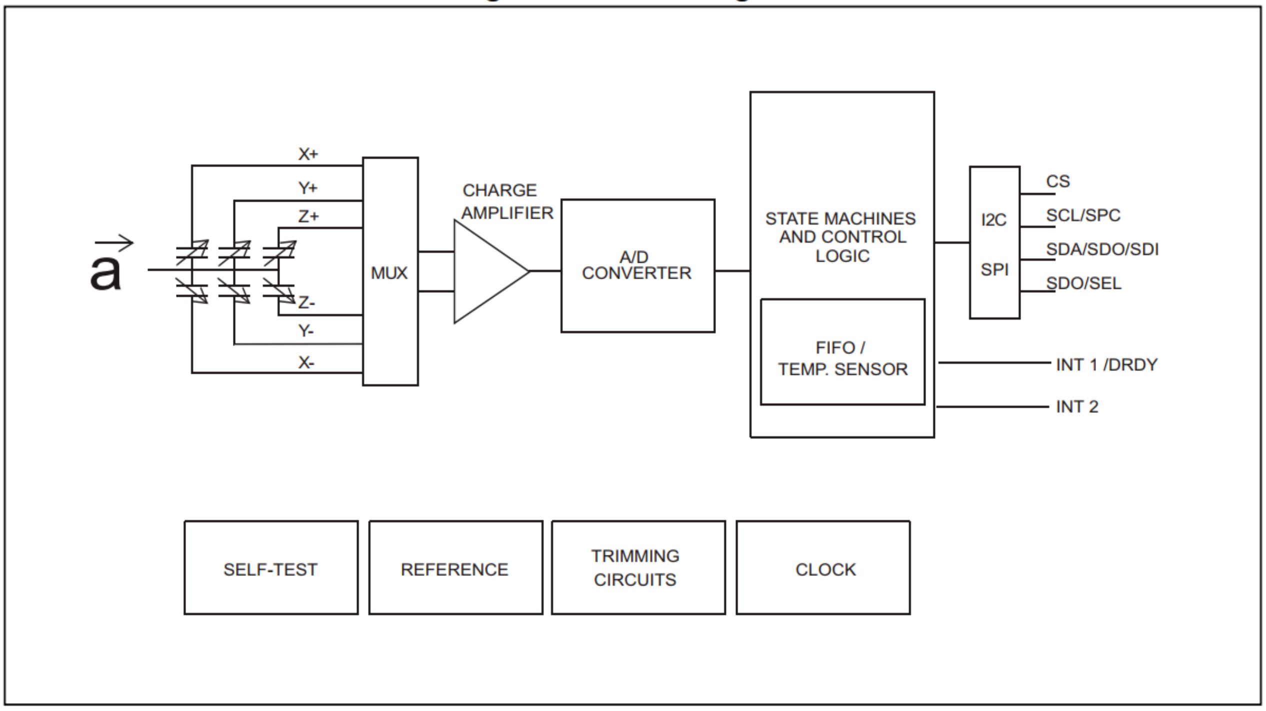
- Wide supply voltage, 1.71 V to 3.6 V
- Independent IOs supply (1.8 V) and supply voltage compatible
- Ultra-low power consumption
- ±2*g*/±4*g*/±6*g*/±8*g*/±16*g* dynamically selectable full scale
- I2C/SPI digital output interface
- 16-bit data output
- Programmable embedded state machines
- Embedded temperature sensor
- Embedded self-test
- Embedded FIFO
- 10000 *g* high shock survivability
- ECOPACK®, RoHS and “Green” compliant

## Description

- The LIS3DSH is an ultra-low-power high performance three-axis linear accelerometer belonging to the “nano” family with an embedded state machine that can be programmed to implement autonomous applications.
- The LIS3DSH has dynamically selectable full scales of ±2*g*/±4*g*/±6*g*/±8*g*/±16*g* and is capable of measuring accelerations with output data rates from 3.125 Hz to 1.6 kHz.
- The self-test capability allows the user to check the functioning of the sensor in the final application.
- The device can be configured to generate interrupt signals activated by user-defined motion patterns.
- The LIS3DSH has an integrated first-in, first-out (FIFO) buffer allowing the user to store data in order to limit intervention by the host processor.
- The LIS3DSH is available in a small thin plastic land grid array package (LGA) and is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

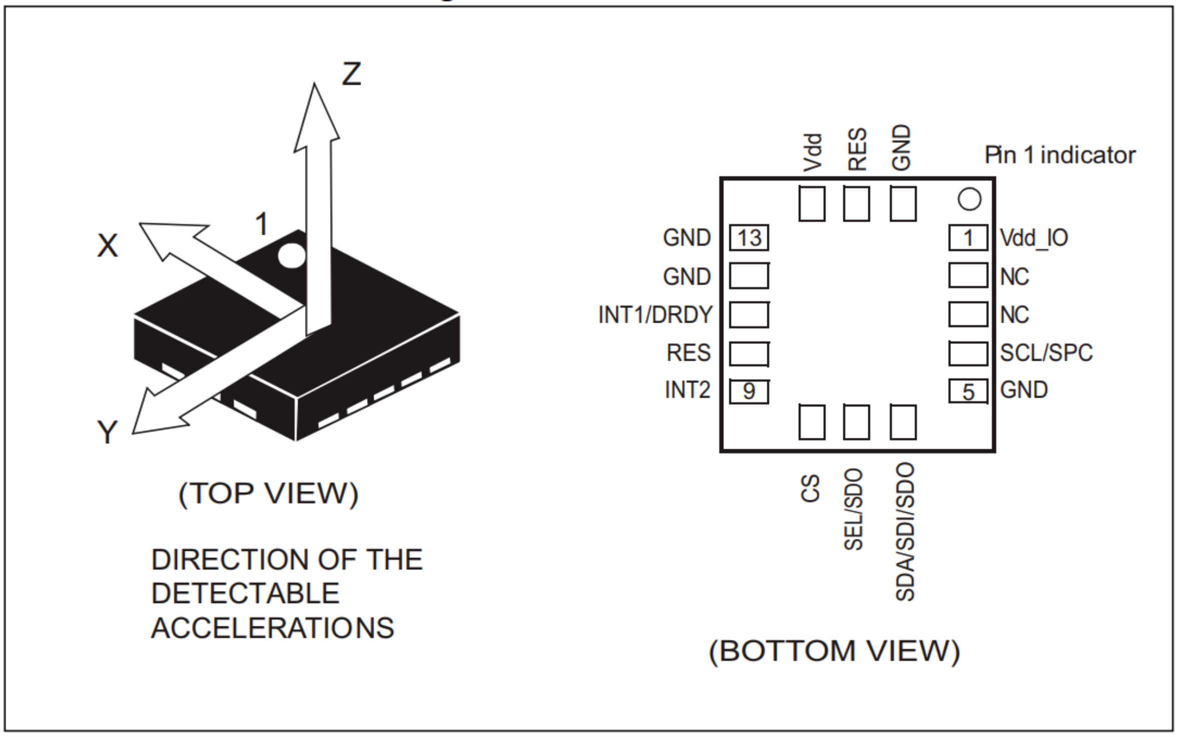
## Block diagram

**Figure 1. Block diagram**



## Pin description

**Figure 2. Pin connections**



## 2.5 Terminology

### 2.5.1 Sensitivity

Sensitivity describes the gain of the sensor and can be determined, for example, by applying 1 *g* acceleration to it. As the sensor can measure DC accelerations this can be done easily by pointing the axis of interest towards the center of the Earth, noting the output value, rotating the sensor by 180 degrees (pointing to the sky) and noting the output value again. By doing so, ±1 *g* acceleration is applied to the sensor. Subtracting the larger output value from the smaller one, and dividing the result by 2, leads to the actual sensitivity of the sensor. This value changes very little over temperature and also time. The sensitivity tolerance describes the range of sensitivities of a large population of sensors.

### 2.5.2 Zero-*g* level

Zero-*g* level offset (TyOff) describes the deviation of an actual output signal from the ideal output signal if no acceleration is present. A sensor in a steady-state on a horizontal surface measures 0 *g* for the X-axis and 0 *g* for the Y-axis, whereas the Z-axis measures 1 *g*. The output is ideally in the middle of the dynamic range of the sensor (content of OUT registers 00h, data expressed as 2’s complement number). A deviation from the ideal value in this case is called Zero-*g* offset. Offset is to some extent a result of stress to MEMS sensor and therefore the offset can slightly change after mounting the sensor on a printed circuit board or exposing it to extensive mechanical stress. Offset changes little over temperature, see “Zero-*g* level change vs. temperature”. The Zero-*g* level tolerance (TyOff) describes the standard deviation of the range of Zero-*g* levels of a population of sensors.

## 2.6 Functionality

### 2.6.1 Self-test

The self-test allows checking the sensor functionality without moving it. The self-test function is off when the self-test bit (ST) is programmed to ‘0‘. When the self-test bit is programmed to ‘1’, an actuation force is applied to the sensor, simulating a definite input acceleration. In this case the sensor outputs exhibit a change in their DC levels which are related to the selected full scale through the device sensitivity. When the self-test is activated, the device output level is given by the algebraic sum of the signals produced by the acceleration acting on the sensor and by the electrostatic test-force. If the output signals change within the amplitude specified in *Table 3*, then the sensor is working properly and the parameters of the chip interface are within the defined specifications.

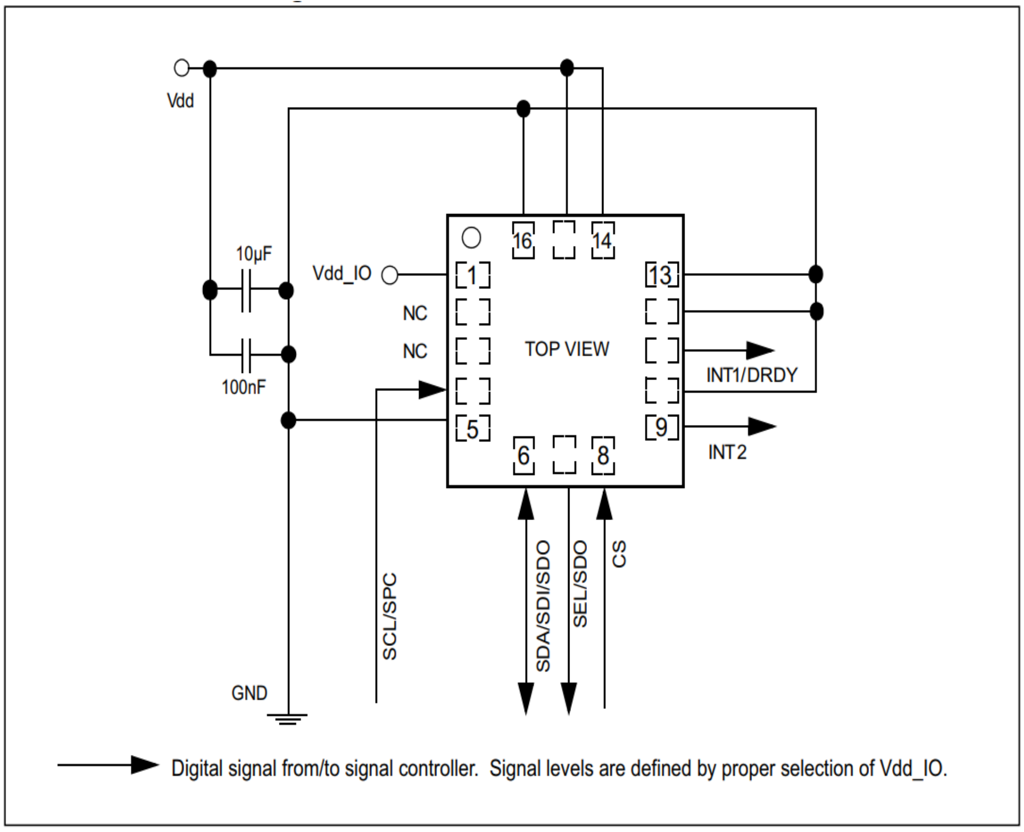
## 2.8 IC interface

The complete measurement chain is made up of a low-noise capacitive amplifier which converts the capacitive unbalancing of the MEMS sensor into an analog voltage that is available to the user through an analog-to-digital converter.
The acceleration data may be accessed through an I2C/SPI interface, therefore making the device particularly suitable for direct interfacing with a microcontroller.
The LIS3DSH features a Data-Ready signal (DRDY) which indicates when a new set of measured acceleration data is available, therefore simplifying data synchronization in the digital system that uses the device.

## 2.9 Factory calibration

The IC interface is factory calibrated for sensitivity (So) and Zero-*g* level (TyOff).
The trimming values are stored inside the device in nonvolatile memory. Any time the device is turned on, the trimming parameters are downloaded into the registers to be used during the active operation. This allows using the device without further calibration.

**Figure 5. LIS3DSH electrical connections**



Vdd
10µF
100nF
Vdd_IO
NC
NC
TOP VIEW
1
5
6
8
9
13
14
16
INT1/DRDY
INT2
GND
SCL/SPC
SDA/SDI/SDO
SEL/SDO
CS
Digital signal from/to signal controller. Signal levels are defined by proper selection of Vdd_IO.

***Note****: Pin 15 can be connected to Vdd or GND or left unconnected.*
The device core is supplied through the Vdd line while the I/O pins are supplied through the Vdd_IO line. Power supply decoupling capacitors (100 nF ceramic, 10 µF) should be placed as near as possible to pin 14 of the device (common design practice).
All the voltage and ground supplies must be present at the same time to have proper behavior of the IC (refer to *Figure 5*). It is possible to remove Vdd while maintaining Vdd_IO without blocking the communication bus, in this condition the measurement chain is powered off.
The functionality of the device and the measured acceleration data are selectable and accessible through the I2C or SPI interfaces. When using the I2C, CS must be tied high.

## SPI bus interface

The LIS3DSH SPI is a bus slave. The SPI allows writing to and reading from the registers of the device.
The serial interface interacts with the outside world with 4 wires: CS, SPC, SDI and SDO.

**Figure 7. Read and write protocol**

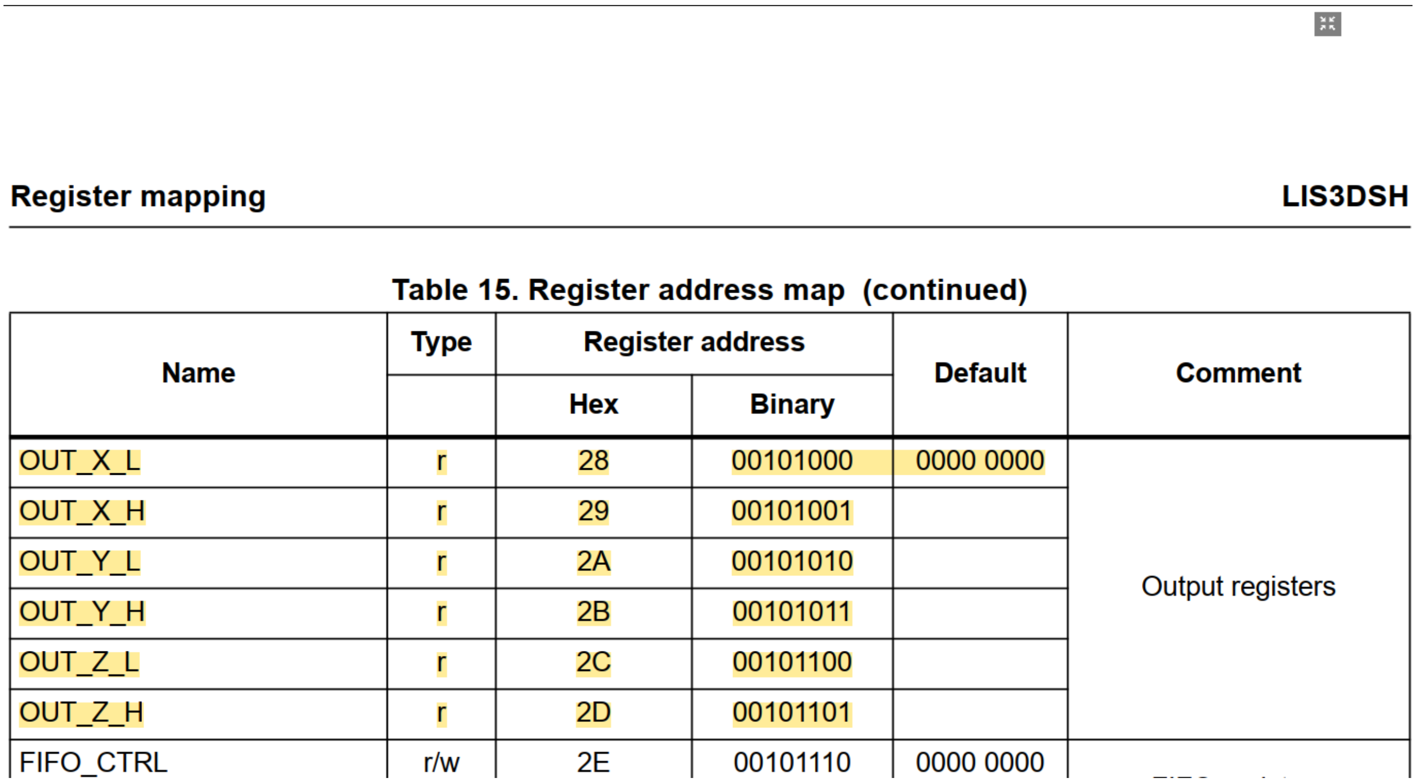
CS is the serial port enable and it is controlled by the SPI master. It goes low at the start of the transmission and goes back high at the end. SPC is the serial port clock and it is controlled by the SPI master. It is stopped high when CS is high (no transmission). SDI and SDO are respectively the serial port data input and output. Those lines are driven at the falling edge of SPC and should be captured at the rising edge of SPC.

Both the read register and write register commands are completed in 16 clock pulses or in multiples of 8 in case of multiple read/write bytes. Bit duration is the time between two falling edges of SPC. The first bit (bit 0) starts at the first falling edge of SPC after the falling edge of CS while the last bit (bit 15, bit 23, ...) starts at the last falling edge of SPC just before the rising edge of CS.
***bit 0***: RW bit. When 0, the data DI(7:0) is written into the device. When 1, the data DO(7:0) from the device is read. In the latter case, the chip drives **SDO** at the start of bit 8.
***bit 1-7***: address AD(6:0). This is the address field of the indexed register.
***bit 8-15***: data DI(7:0) (write mode). This is the data that is written into the device (MSb first). ***bit 8-15***: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

## SPI read

**Figure 8. SPI read protocol**

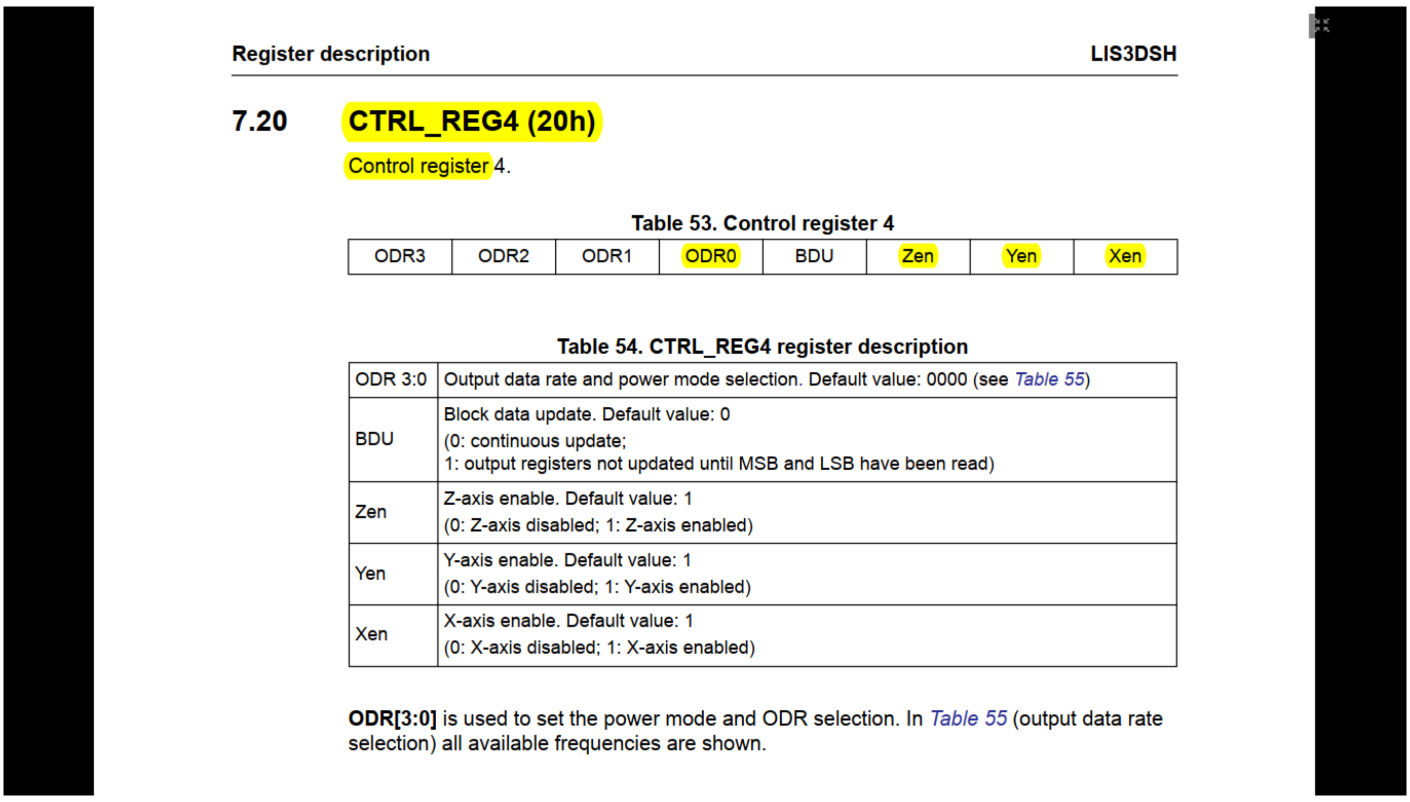
The SPI Read command is performed with 16 clock pulses. A multiple byte read command is performed by adding blocks of 8 clock pulses to the previous one. ***bit 0***: READ bit. The value is 1. ***bit 1-7***: address AD(6:0). This is the address field of the indexed register. ***bit 8-15***: data DO(7:0) (read mode). This is the data that is read from the device (MSb first). ***bit 16-...*** : data DO(...-8). Further data in multiple byte reads.



Register mapping LIS3DSH

Table 15. Register address map (continued)

| Name | Type | Register address | | Default | Comment |
|---|---|---|---|---|---|
| | | Hex | Binary | | |
| OUT_X_L | r | 28 | 00101000 | 0000 0000 | Output registers |
| OUT_X_H | r | 29 | 00101001 | | |
| OUT_Y_L | r | 2A | 00101010 | | |
| OUT_Y_H | r | 2B | 00101011 | | |
| OUT_Z_L | r | 2C | 00101100 | | |
| OUT_Z_H | r | 2D | 00101101 | | |
| FIFO_CTRL | r/w | 2E | 00101110 | 0000 0000 | |



Register description LIS3DSH

## 7.20 CTRL_REG4 (20h)

Control register 4.

Table 53. Control register 4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ODR3 | ODR2 | ODR1 | ODR0 | BDU | Zen | Yen | Xen |

Table 54. CTRL_REG4 register description

| | |
|---|---|
| ODR 3:0 | Output data rate and power mode selection. Default value: 0000 (see *Table 55*) |
| BDU | Block data update. Default value: 0<br>(0: continuous update;<br>1: output registers not updated until MSB and LSB have been read) |
| Zen | Z-axis enable. Default value: 1<br>(0: Z-axis disabled; 1: Z-axis enabled) |
| Yen | Y-axis enable. Default value: 1<br>(0: Y-axis disabled; 1: Y-axis enabled) |
| Xen | X-axis enable. Default value: 1<br>(0: X-axis disabled; 1: X-axis enabled) |

**ODR[3:0]** is used to set the power mode and ODR selection. In *Table 55* (output data rate selection) all available frequencies are shown.

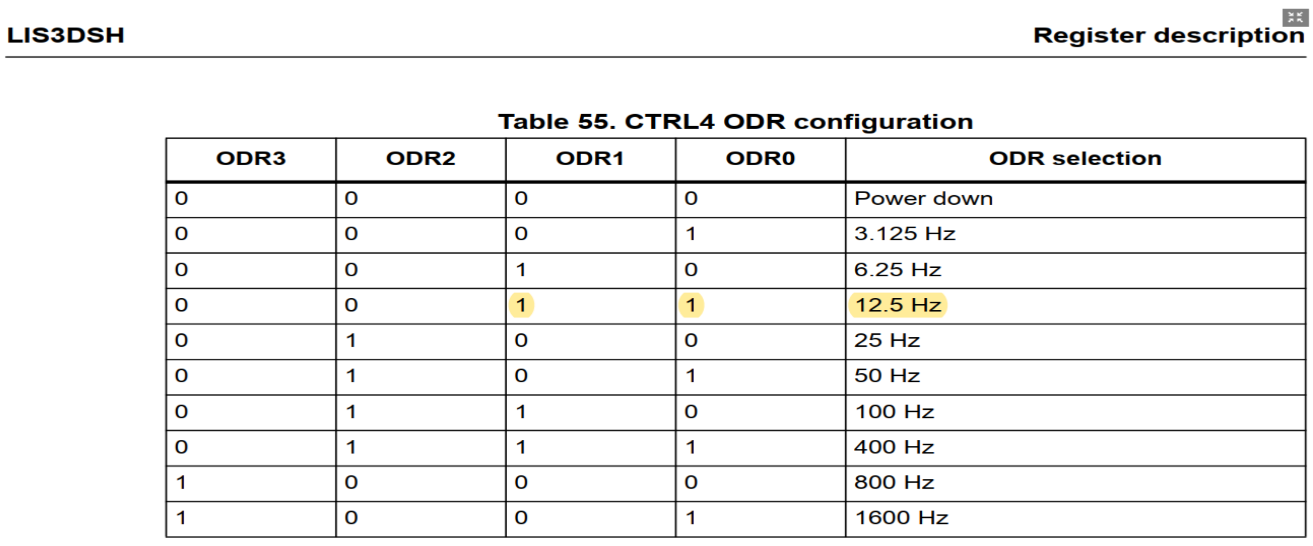
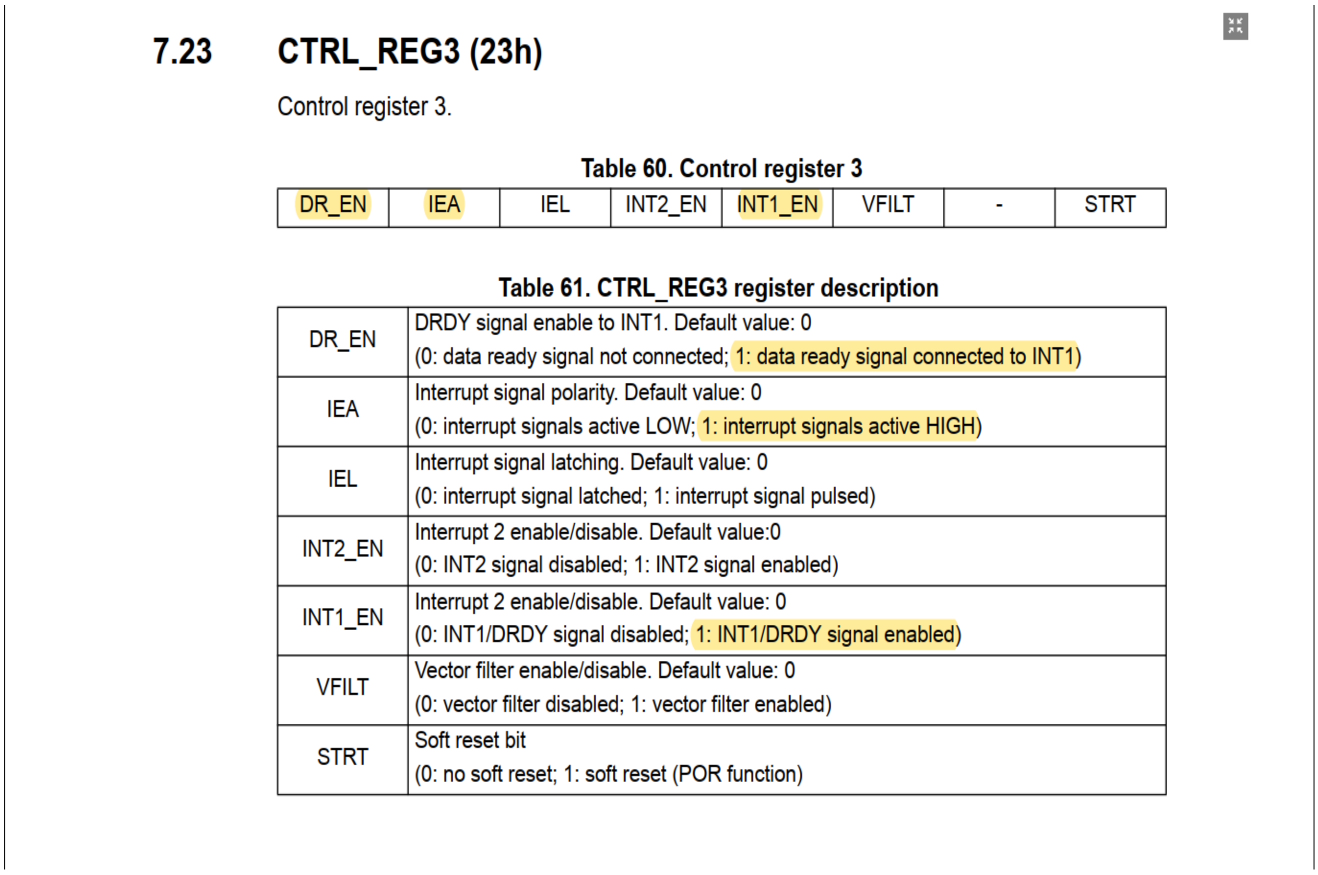


Table 55. CTRL4 ODR configuration

| ODR3 | ODR2 | ODR1 | ODR0 | ODR selection |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Power down |
| 0 | 0 | 0 | 1 | 3.125 Hz |
| 0 | 0 | 1 | 0 | 6.25 Hz |
| 0 | 0 | 1 | 1 | 12.5 Hz |
| 0 | 1 | 0 | 0 | 25 Hz |
| 0 | 1 | 0 | 1 | 50 Hz |
| 0 | 1 | 1 | 0 | 100 Hz |
| 0 | 1 | 1 | 1 | 400 Hz |
| 1 | 0 | 0 | 0 | 800 Hz |
| 1 | 0 | 0 | 1 | 1600 Hz |

The **BDU** bit is used to inhibit the update of the output registers until both upper and lower register parts are read. In default mode (BDU = ‘0’) the output register values are updated continuously. When the BDU is activated (BDU = ‘1’), the content of the output registers is not updated until both MSB and LSB are read which avoids reading values related to different sample times.



## 7.23 CTRL_REG3 (23h)

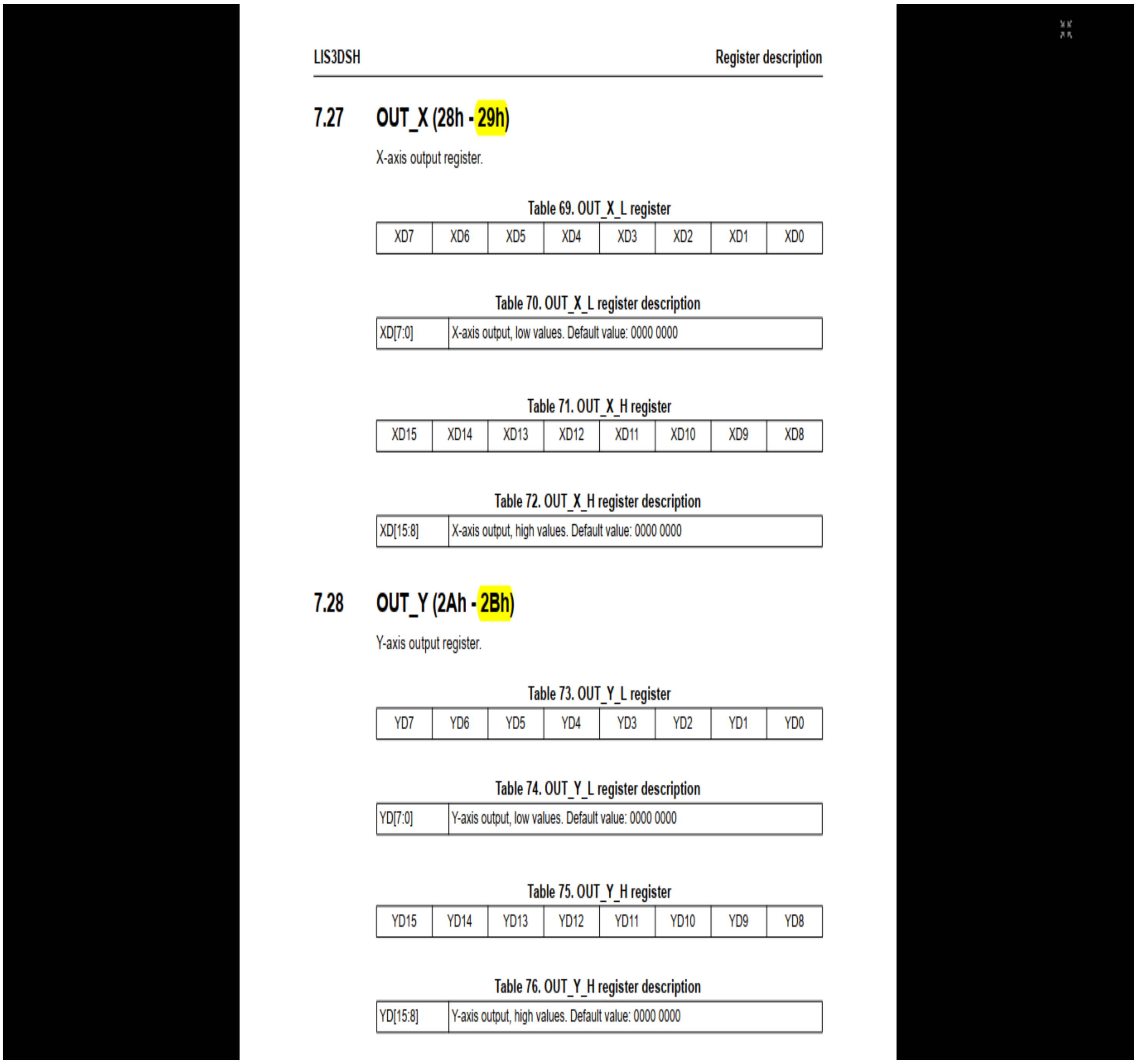
Control register 3.

Table 60. Control register 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| DR_EN | IEA | IEL | INT2_EN | INT1_EN | VFILT | - | STRT |

Table 61. CTRL_REG3 register description

| | |
|---|---|
| DR_EN | DRDY signal enable to INT1. Default value: 0<br>(0: data ready signal not connected; 1: data ready signal connected to INT1) |
| IEA | Interrupt signal polarity. Default value: 0<br>(0: interrupt signals active LOW; 1: interrupt signals active HIGH) |
| IEL | Interrupt signal latching. Default value: 0<br>(0: interrupt signal latched; 1: interrupt signal pulsed) |
| INT2_EN | Interrupt 2 enable/disable. Default value:0<br>(0: INT2 signal disabled; 1: INT2 signal enabled) |
| INT1_EN | Interrupt 2 enable/disable. Default value: 0<br>(0: INT1/DRDY signal disabled; 1: INT1/DRDY signal enabled) |
| VFILT | Vector filter enable/disable. Default value: 0<br>(0: vector filter disabled; 1: vector filter enabled) |
| STRT | Soft reset bit<br>(0: no soft reset; 1: soft reset (POR function) |



## 7.27 OUT_X (28h - 29h)

X-axis output register.

Table 69. OUT_X_L register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| XD7 | XD6 | XD5 | XD4 | XD3 | XD2 | XD1 | XD0 |

Table 70. OUT_X_L register description

| | |
|---|---|
| XD[7:0] | X-axis output, low values. Default value: 0000 0000 |

Table 71. OUT_X_H register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| XD15 | XD14 | XD13 | XD12 | XD11 | XD10 | XD9 | XD8 |

Table 72. OUT_X_H register description

| | |
|---|---|
| XD[15:8] | X-axis output, high values. Default value: 0000 0000 |

## 7.28 OUT_Y (2Ah - 2Bh)

Y-axis output register.

Table 73. OUT_Y_L register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| YD7 | YD6 | YD5 | YD4 | YD3 | YD2 | YD1 | YD0 |

Table 74. OUT_Y_L register description

| | |
|---|---|
| YD[7:0] | Y-axis output, low values. Default value: 0000 0000 |

Table 75. OUT_Y_H register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| YD15 | YD14 | YD13 | YD12 | YD11 | YD10 | YD9 | YD8 |

Table 76. OUT_Y_H register description

| | |
|---|---|
| YD[15:8] | Y-axis output, high values. Default value: 0000 0000 |

## 3.6 SN65HVD230 CAN Transceivers

The SN65HVD230is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The SN65HVD230device provides differential transmit and receive capability for the CAN protocol controller and is fully compatible with the ISO-11898 standard, including 24V requirements. It will operate at speeds of up to 1 Mb/s. Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus cabling (differential output). It also provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources. Some of its features are:

- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Detection of ground fault (permanent dominant) on TXD input
- Power-on Reset and voltage brown-out protection
- An unpowered node or brown-out event will not disturb the CAN bus
- Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
- Up to 112 nodes can be connected
- High-noise immunity due to differential bus implementation
- Temperature ranges: - Industrial (I): -400C to +850C - Extended (E): -400C to +1250C

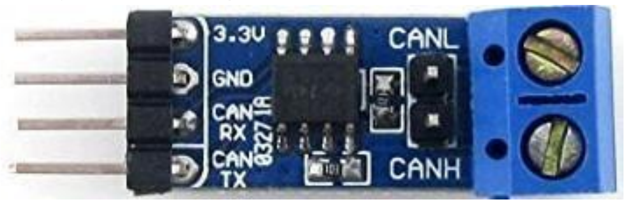


Fig 3.6.1 CAN Transceiver

**CAN Protocol in STM32**

The Basic Can protocol in STM32. Here we will see, how to communicate between two STM32 boards using the CAN protocol.

CAN (Controlled Area Network) Protocol is a way of communication between different devices but under certain rules. These rules must be followed when a message is transmitted over the CAN bus. Shown below is the Standard CAN Frame

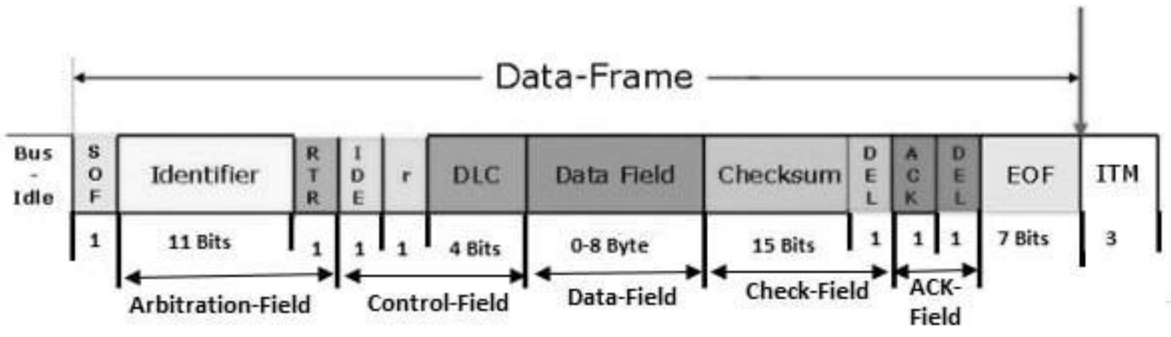


Fig 3.6.2 CAN Data Frame The
CRC and ACK will be handled by the HAL Library.

Here, Identifier is the ID of the transmitting Device
RTR (Remote Transmission Request) Specifies if the data is Remote frame or Data frame IDE specifies if we are using Standard ID or Extended ID r is the Reserved bit
DLC specifies the data length in Bytes
Data Field is where we can send the data, which should be up to 8 bytes
Checksum and DEL are the CRC data and it's Delimiter
ACK and DEL is the acknowledgment bit and its Delimiter

**Connection between CAN transceiver and STM32f3 board**

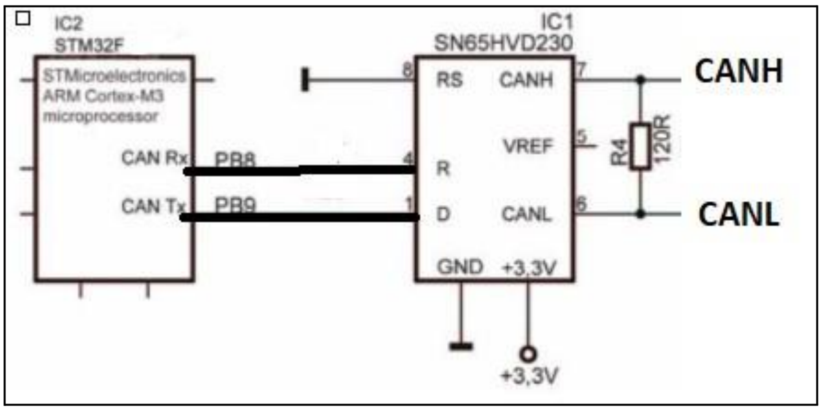
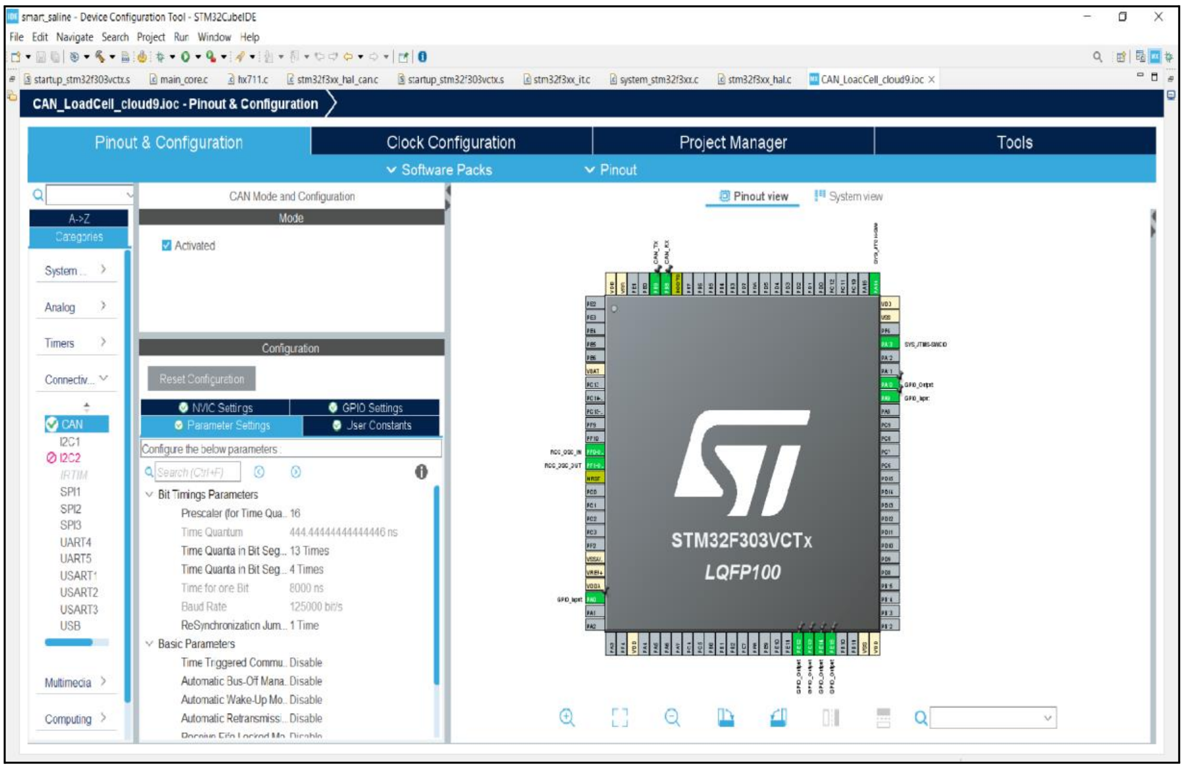


Fig 3.6.4 Connection between STM32f3 board and CAN transceiver

Here the Tx and Rx from the Transceivers are connected to PB9 and PB10 of the Respective controllers CANH and CANL are connected to each other

## Connection and Configuration



CAN Mode and Configuration

Mode

☑ Activated

Configuration

Reset Configuration

NVIC Settings | GPIO Settings | Parameter Settings | User Constants

Configure the below parameters :

```
Bit Timings Parameters
    Prescaler (for Time Qua...  16
    Time Quantum                444.44444444444446 ns
    Time Quanta in Bit Seg...   13 Times
    Time Quanta in Bit Seg...   4 Times
    Time for one Bit            8000 ns
    Baud Rate                   125000 bit/s
    ReSynchronization Jum...    1 Time
Basic Parameters
    Time Triggered Commu...     Disable
    Automatic Bus-Off Mana...   Disable
    Automatic Wake-Up Mo...     Disable
    Automatic Retransmissi...   Disable
    Receive Fifo Locked Mo...   Disable
```

Fig 3.6.3 STM32CUBE IDE

Here the BAUD RATE is set to 125000 bps. The Operating Mode is NORMAL Mode.

Set the SPI parameters such as:

- **Mode:** Master or Slave
- **Direction:** 2 Lines (Full Duplex), 1 Line (Rx only, or Tx only)
- **Data Size:** 8-bit, 16-bit, etc.
- **Clock Polarity (CPOL):** Low or High
- **Clock Phase (CPHA):** 1st Edge or 2nd Edge
- **Baud Rate Prescaler:** Adjust according to your clock settings
- **NSS (Slave Select) Management:** Hardware or Software

**Generate Code**

- **Generate Code:**
  - Click the "Project" menu and then "Generate Code" or click the "Generate Code" button on the toolbar.
  - STM32CubeIDE will generate the initialization code and setup files based on your configuration.

**Test and Debug**

- **Monitor Communication:**
  - Use a logic analyzer or oscilloscope to monitor the SPI signals (MOSI, MISO, SCK, and SS) to verify correct operation.
- **Debugging:**
  - Set breakpoints and use debugging tools provided by STM32CubeIDE to step through your code and inspect variables.

## Additional Resources:

- **Reference Manual:**
  - Consult the STM32 reference manual for detailed information on the SPI peripheral and its capabilities.
- **STM32 HAL Documentation:**
  - STM32 HAL library documentation provides additional details and examples for using SPI functions.

## Potentiometer

Using a potentiometer in an automatic hold system involves leveraging its variable resistance to control or measure a parameter that determines when to engage or disengage a holding mechanism. Here’s a detailed overview of how a potentiometer can be integrated into such a system:

Overview of Automatic Hold Systems

An automatic hold system generally refers to a mechanism that automatically engages or disengages based on certain conditions. This could be used in various applications such as:

- Automotive systems (e.g., automatic parking brakes)
- Industrial machinery (e.g., automated clamping systems)
- Consumer electronics (e.g., automatic hold in audio devices)

Role of a Potentiometer

A potentiometer is a variable resistor that can be adjusted manually or automatically to vary resistance. In an automatic hold system, a potentiometer can be used to:

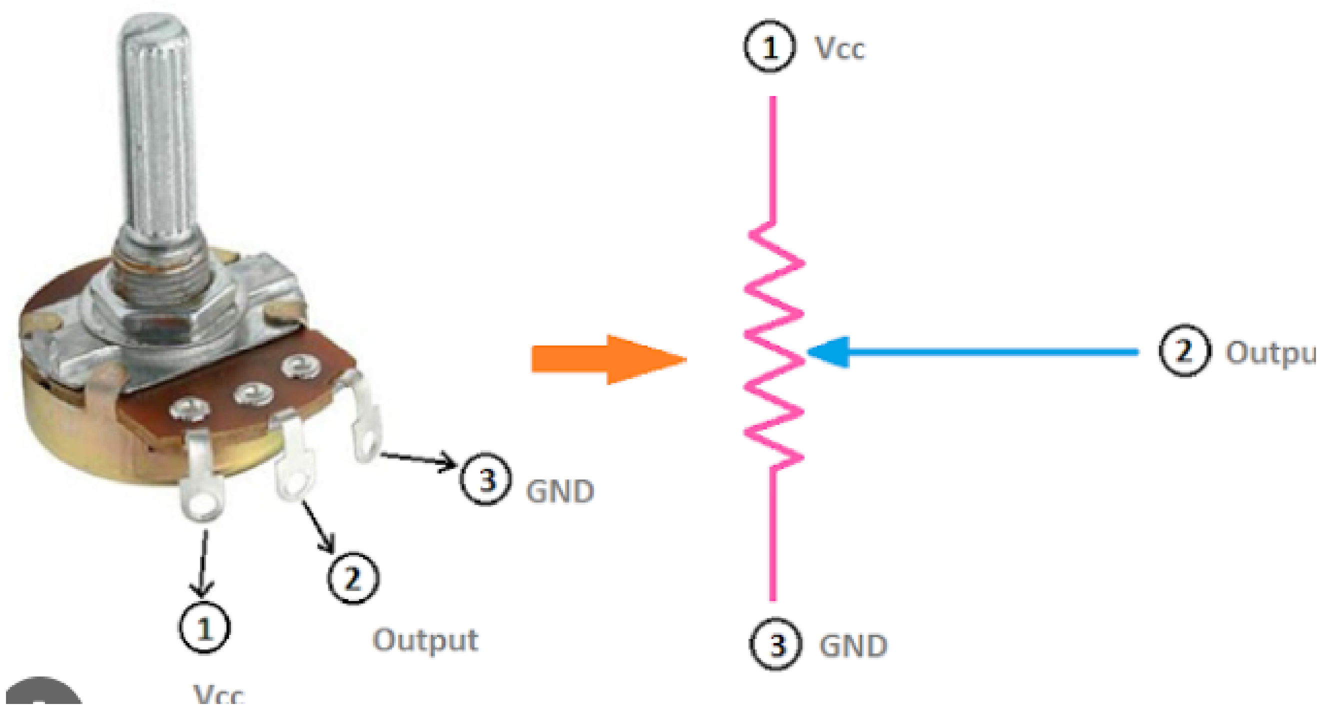
Measure Position or Angle:

Potentiometers can measure the position of a moving part or the angle of a rotating component. For instance, in an automotive parking brake system, a potentiometer might measure the position of the brake pedal or the state of the parking brake lever.

Adjust Control Parameters:

The potentiometer can be used to adjust thresholds or setpoints in the control system. For example, in an industrial clamping system, it might be used to set the pressure at which a clamp engages.

And at the Time of covid we have to safeguard nurse/doctor that they should be away from contaminated zone , and ensure good service to the patients.



A/D conversion of the various channels can be performed in single, continuous, scan or discontinu-ous mode. The result of the ADC is stored in a left- or right-aligned 16-bit data register. Moreover,

the ADC also implements the analog watchdog feature, which allows the application to detect if the
input voltage goes outside the user-defined higher or lower thresholds: if this happens, a dedicated IRQ fires.

Figure 1 schematizes the structure of the ADC2. An input selection and scan control unit performs the selection of the input source to the ADC. Depending on the conversion mode (single, scan or continuous mode), this unit automatically switches among the input channels, so that every one can
be sampled periodically. The output from this unit feeds the ADC.

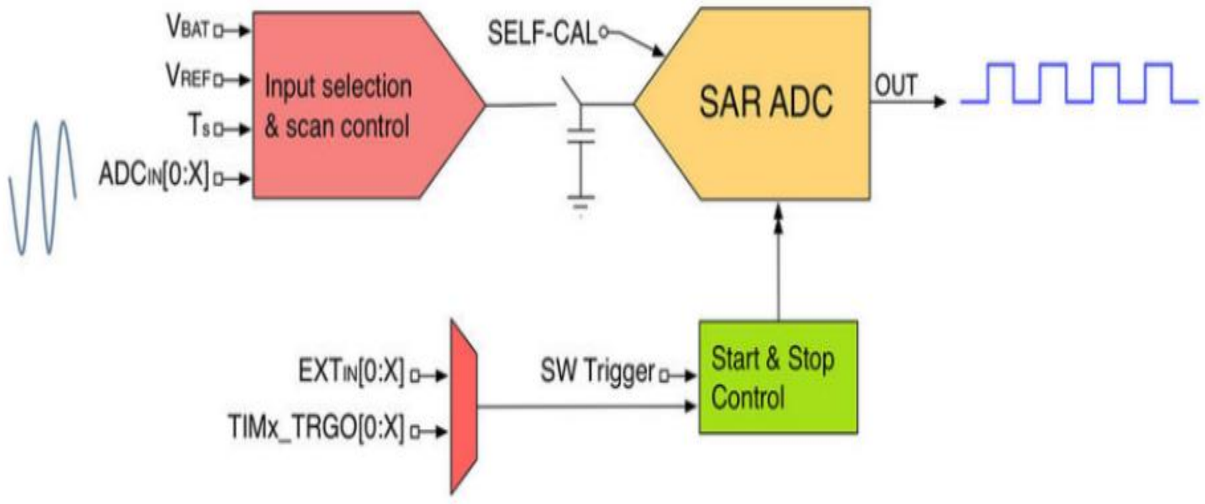


Figure 1: The simplified structure of an ADC

Figure 1 also shows another important part of the ADC: the start and stop control unit. Its role is to control the A/D conversion process, and it can be triggered by software or by a variable number

of input sources. Moreover, it is internally connected to the TRGO line of some timers so that time-
driven conversions can be automatically performed in DMA mode. We will analyze this important

mode of the ADC peripheral later.

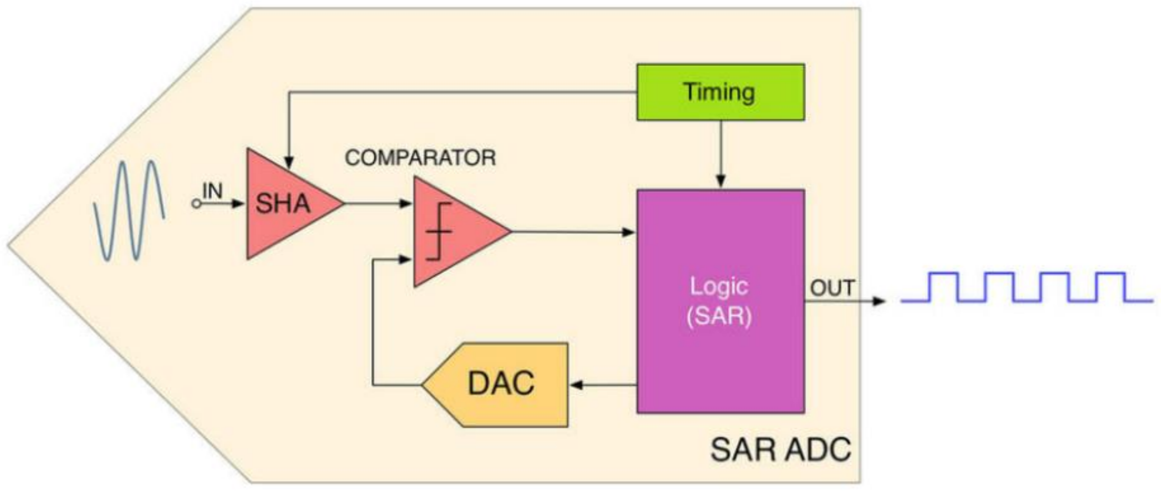


Figure 2: The internal structure of a SAR ADC

Figure 2 shows the main blocks forming the SAR ADC unit shown in Figure 1. The input signal goes
through the SHA unit. As you can see in Figure 1, a switch and a capacitor are in series with the ADC
input. That part represents the Sample-and-Hold (SHA) unit shown in Figure 2, which is a feature available in all ADCs. This unit plays the important role to keep the input signal constant during the conversion cycle. Thanks to an internal timing unit, which is regulated by a configurable clock
as we will see later, the SAR constantly connects/disconnects the input source by closing/opening the "switch" in Figure 1. To keep the voltage level of the input constant, the SHA is implemented with a network of capacitors: this ensure that source signal is kept at a certain level during the A/D
conversion, which is a procedure that requires a given amount of time, depending on the conversion
frequency chosen.
The output from the SHA module feeds a comparator that compares it with another signal generated
by an internal DAC. The result of comparison is sent to the logic unit, which computes the numerical
representation of the input signal according a well-characterized algorithm. This algorithm is what
distinguishes SAR ADC from other A/D converters.
The Successive Approximation algorithm computes the voltage of the input signal by comparing it
with the one generated by the internal DAC, which is a fraction of the VREF voltage: if the input signal is higher than this internal reference voltage, then this is further increased until the input signal is lower. The final result will correspond to a number ranging from zero to the maximum 12-bit unsigned integer, that is 2

12 − 1 = 4095. Assuming VREF = 3300mV , we have that 3300mV is

represented with 4095. This means that 110 =
3300
4095 ≈ 0.8mV . For example, an input voltage equal to

2.5V will be converted to: x =
4095
3300mV

× 2500mV = 3102

The SAR algorithm works in the following way:

1. The output data register is zeroed and the MSB bit is set to 1. This will correspond to a well-
defined voltage level generated by the internal DAC.

2. The output of the DAC is compared with the input signal VIN :
1. if VIN is higher, than the bit is left to 1;
2. if VIN is lower, than the bit is set back to 0;
3. The algorithm proceeds to the next MSB bit in the data register until all bits are either set to 1
or 0.

The algorithm now sets the 1st bit to 1, which leads to an internal voltage
equal to 2860mV. This value is still higher than VIN and the algorithm resets the last bit to zero. The
ADC so detects that the input voltage is something close to 2640mV. Clearly, the more resolution the ADC provides, the more close to VIN the converted value will be.
As you can see, the SAR algorithm essentially performs a search in a binary tree. The great advantage
of this algorithm is that the conversion is performed in N-cycles, where N corresponds to the ADC resolution. So a 12-bit ADC requires twelve cycles to perform a conversion. But how long a cycle can last? The number of cycles per seconds, that is the ADC frequency, is a performance evaluation
parameter of the ADC. SAR ADCs can be really fast, especially if the ADC resolution is decreased (less sampled bit corresponds to less cycles per conversion). However, the impedance of the analog
signal source, or series resistance (RIN ), between the source and the MCU pin causes a voltage drop
across it because of the current flowing into the pin.

**HAL_ADC Module :**

After a brief introduction to the most important features offered by the ADC peripheral in STM32 microcontrollers, it is the right time to dive into the related CubeHAL APIs.
To manipulate the ADC peripheral, the HAL defines the C struct ADC_HandleTypeDef, which is defined in the following way:

```
typedef struct {
ADC_TypeDef *Instance; /* Pointer to ADC descriptor */
ADC_InitTypeDef Init; /* ADC initialization parameters */
__IO uint32_t NbrOfCurrentConversionRank; /* ADC number of current conversion rank */
DMA_HandleTypeDef *DMA_Handle; /* Pointer to the DMA Handler */
HAL_LockTypeDef Lock; /* ADC locking object */
__IO uint32_t State; /* ADC communication state */
__IO uint32_t ErrorCode; /* Error code */
} ADC_HandleTypeDef;
```

Let us analyze the most important fields of this struct.
• Instance: is the pointer to the ADC descriptor we are going to use. For example, ADC1 is the descriptor of the first ADC peripheral.
• Init: is an instance of the C struct ADC_InitTypeDef, which is used to configure the ADC. We will study it more in depth in a while.
• NbrOfCurrentConversionRank: corresponds to the current i-th channel (rank) in a regular conversion group. We will describe it better soon.

**ClockPrescaler**: defines the speed of the clock (ADCCLK) for the analog circuitry part of ADC. In the previous paragraph we have seen that the ADC has an internal timing unit that controls the switching frequency of the input switch (see Figure 2). The ADCCLK establishes the speed of this timing unit and it impacts on the number of samples per seconds, because it defines the amount of time used by each conversion cycle. This clock is generated from the peripheral clock divided by a programmable prescaler that allows the ADC to work at fP CLK/2,/4,/6 or /8 (refer to the datasheets of the specific MCU for the maximum values of ADCCLK and its prescaler). In some STM32 MCUs the ADCCLK can also be derived from the HSI oscillator. The value of this field affects the ADCCLK speed of all ADCs implemented in the MCU.

## Scan Single Conversion Mode

This mode, also called multichannel single mode in some ST documents, is used to convert some channels successively in independent mode. Using ranks, you can use this ADC mode to configure any sequence of up to 16 channels successively with different sampling times and in custom orders.
You can, for example, carry out the sequence shown in Figure 6. In this way, you do not have to stop the ADC during the conversion process in order to reconfigure the next channel with a different sampling time. This mode saves additional CPU load and heavy software development. Scan conversions are carried out in DMA mode.

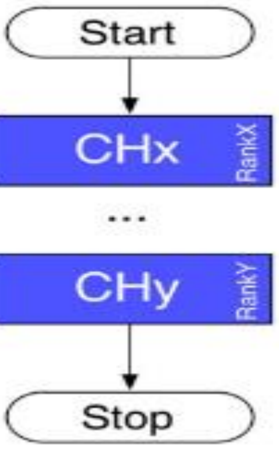


Figure 6: Scan single conversion mode

For example, this mode can be used when starting a system depends on some parameters like knowing the coordinates of the arm’s tip in a manipulator arm system. In this case, you have to read the position of each articulation in the manipulator arm system at power-on to determine the coordinates of the arm’s tip. This mode can also be used to make single measurements of multiple signal levels (voltage, pressure, temperature, etc.) to decide if the system can be started or not in order to protect the people and equipment.

We are now finally ready to analyze a complete example. We will start by seeing the APIs used to perform conversions in polling mode. As you will see, there is nothing new compared to what seen so far with other peripherals.
The example we are going to study does a simple thing: it uses the internal temperature sensor available in all STM32 MCUs as source for the ADC. The temperature sensor is connected to an internal ADC input. The exact input number depends on the specific MCU family and package.
For example, in an STM32F401RE MCU the temperature sensor is connected to the IN18 of ADC1 peripheral. However, the HAL is designed to abstract this specific aspect. Before we analyze the real code, it is best to give a quick look at the electrical characteristics of the temperature sensor, which are reported in the datasheet of the MCU you are considering.

The HAL_ADC module in the CubeF1 HAL slightly differs from the other HALs. To start a conversion

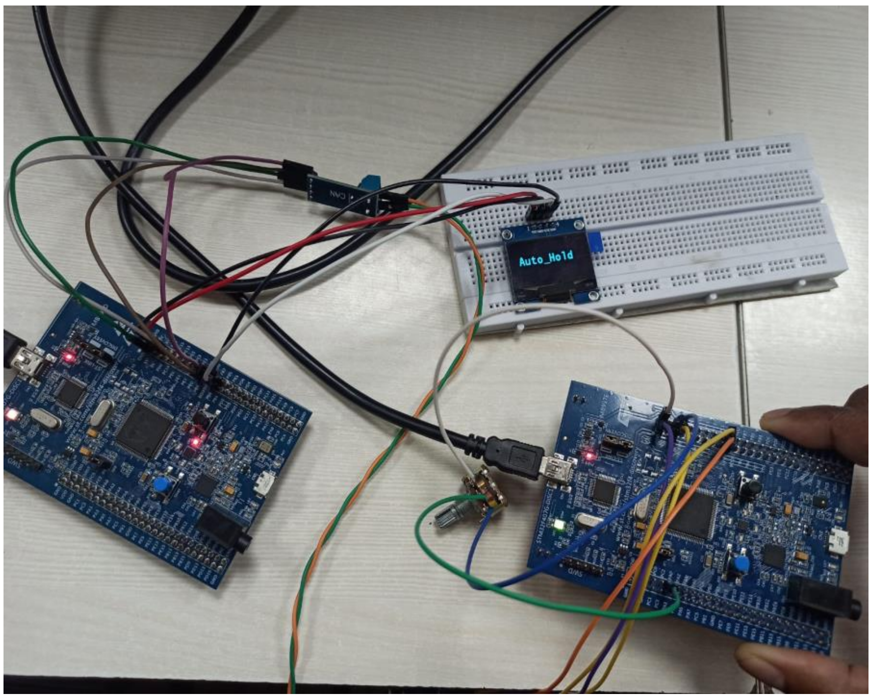
driven by software it is required that the parameter hadc.Init.ExternalTrigConv = ADC_SOFT-WARE_START is specified during the ADC initialization. This completely differs from what other HALs

do, and it is not clear why ST developers have adopted this different approach. Moreover, even CubeMX offers a different configuration to take in account this peculiarity when it generates the corresponding initialization code. Refer to book examples for the complete configuration procedure.

# Chapter 4 Results

The auto hold system’s output primarily involves controlling the vehicle’s braking or transmission to maintain its stationary position. This includes signals to brake or transmission actuators, visual and audio alerts to the driver, and maintaining various vehicle parameters. The system enhances safety and convenience by automatically holding the vehicle in place, reducing driver fatigue and preventing unintentional movement.

Many times manual saline monitoring does not work well or fails because of certain disadvantages like the unavailability of doctors, nurses, caretakers for continuous saline monitoring, and large number of patients in the hospital. There is the possibility that sometimes serious patients' saline bottle may be about to finish and nobody is near the patient. Due to this negligence, the complete saline bottle may get fed to the patient''s body and if the needle is not removed instantly from the patient’s body, then the blood may flow reverse in the saline due to pressure difference between empty saline bottle and blood pressure of the patient, which may cause death of the patient or creates critical situation.



OUTPUT

The saline level monitoring system overcomes drawbacks of the manual saline level monitoring system. This project has weight sensor which will monitor the saline liquid level in the bottle and passes the signal to STM32 which indicate blink the led and also uses IOT concept to alarm the

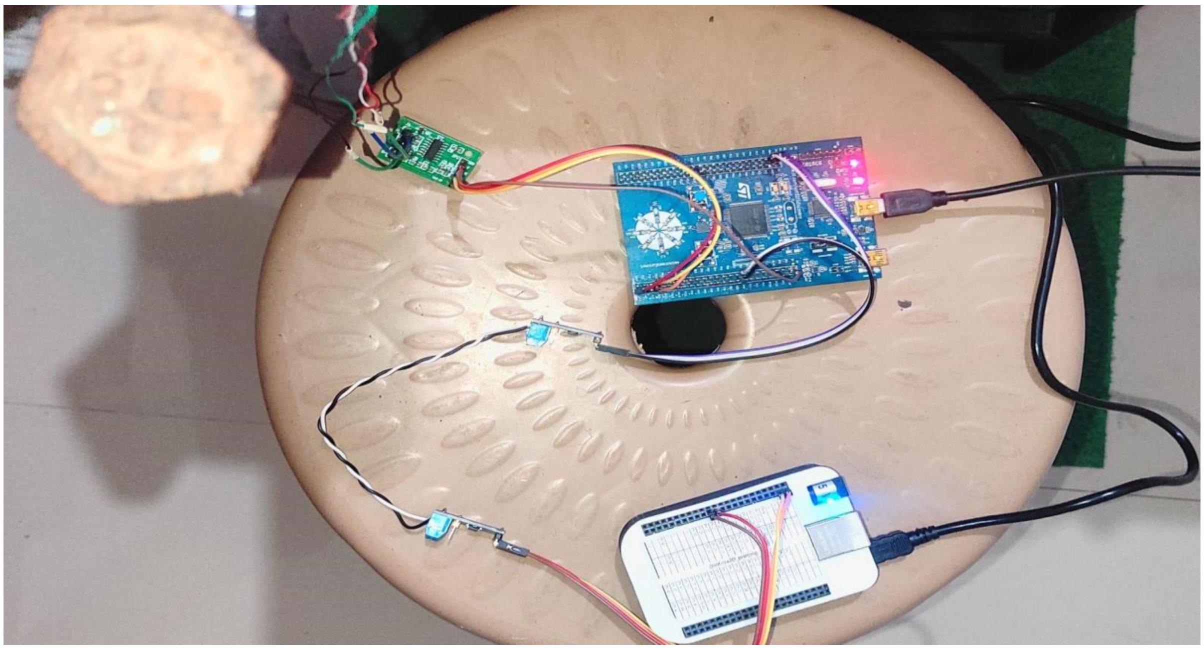


Fig 4.1.: Practical Hardware Connection

## Conclusion

The proposed embedded system offers a promising solution for enhancing vehicle safety and driver comfort on inclined surfaces.
By integrating an STM32F407VGT6 microcontroller, a Real-Time Operating System, and a CAN bus network, the system effectively addresses
the challenges associated with maintaining vehicle stability on inclines.

The utilization of a LIS3DSH accelerometer for inclination measurement and a variable potentiometer for setting desired engine speed
provides a practical approach to determining the need for auto-hold activation. The distributed control architecture, employing a master
and slave microcontroller, ensures efficient task management and reliable system operation.

The integration of an OLED display enhances user experience by providing clear visual feedback on the auto-hold status.
Overall, the proposed system demonstrates the potential to improve vehicle safety and driver convenience, particularly in hilly or mountainous terrains.

Future research may focus on refining the auto-hold activation and deactivation criteria, exploring advanced sensor fusion techniques for
improved inclination estimation, and investigating the integration of the system with other vehicle subsystems such as hill descent control.

By addressing these areas, the performance and capabilities of the auto-hold system can be further enhanced.

## REFERENCES

[1] Othman, H.F.; Aji, Y.R.; Fakhreddin, F.T.; Al-Ali, A.R. Controller Area Networks: Evolution and Applications, 2nd Information and Communication Technologies, 2006, vol. 2, pp. 3088 - 3093.
[2] Robert Bosch GmbH, "CAN Specification", Version 2.0, 1991.
[3] Vehicle Control System using Controller Area Network [Can] Protocol T. Rajasekar1, K. Bhaskar 21 Student, M.Tech, Embedded System Technologies, Vel Tech Dr RR & Dr SR Technical University 2Assistant Professor, Department of EEE.
[4]Controller Area Networks: Evolution and Applications, Othman, H. F.Aji, Y. R.; Fakhreddin, F. T.; Al-Ali, A.R., "Controller Area Networks:
Evolution and Applications," Information and Communication Technologies, 2006. ICTTA '06. 2nd , vol.2, no., pp.3088,3093, 0-0 0
[5]J.M .Irazabel & S.Blozis, Philips Semiconductors, "I2CManual, Application Note, ref. AN10216-0" March24, 2012.
[6]Vehicle Parameter Monitoring Using CAN Protocol IJCST Jan-Feb 2015 Pratiksha Nawale, Anjali Vekhande, Priyanka Waje
[7]Vehicle Control System using Controller Area Network [Can] Protocol T. Rajasekar1, K.Bhaskar2 1Student, M.Tech, Embedded System Technologies, Vel Tech Dr RR & Dr SR Technical University 2Assistant Professor, Department of EEE.
[8]Vehicle Control Using CAN Protocol For Implementing the Intelligent System (IBS) IJAREEIE March 2014
[9] Venkatesh H. and Rajashri Y Manakwad, "Driver Alerting System Using CAN Protocol", International Journal of Electrical and Electronics Research, Vol. 3, Issue 1, pp. 218-223, 2015.
[10]N.Q. B. M. Noor and A. Saparon, "FPGA implementation of high speed serial peripheral interface for motion controller," in Proc. 2012 IEEE Symposium on Industrial Electronics and Applications (ISIEA), pp.78-83, Sept. 2012.