

Project Report On
**Industrial Monitoring and Control System using CAN
Protocol**



*Submitted
In partial fulfilment
For the award of the Degree of*

PG-Diploma in Embedded Systems and Design (PG-DESD)

C-DAC, ACTS (Pune)

Guided by: Mr. Tarun Bharani

Submitted By:

Mr. Chinmay Jadhav.	240840130007
Mr.Kailashnath M Sugre.	240840130012
Mr. Saurav Makde.	240840130036
Ms.Ujani Mukherjee.	240840130046
Mr. Valleboina Durgaprasad.	240840130050

Centre for Development of Advanced Computing (C-DAC), ACTS (Pune-411008)

ABSTRACT

An Industrial Monitoring and Control System using the CAN protocol enables machines and sensors in a factory to communicate efficiently and reliably. The Controller Area Network (CAN) is a robust communication protocol that reduces wiring complexity and enhances real-time monitoring and control of industrial equipment. This project implements a CAN-based system using STM32 microcontrollers, CAN bus modules, a BMP280 sensor, and an OLED display to demonstrate real-time data acquisition, transmission, and display.

The transmitter STM32 board reads environmental parameters like temperature and pressure from the BMP280 sensor and sends the data via the CAN bus. The receiver STM32 board processes the received data and displays it on an OLED screen, providing real-time monitoring. The system ensures error-free data transmission using the CAN protocol's built-in error detection and message prioritization features.

Compared to traditional industrial monitoring methods, this system offers faster response times, improved fault tolerance, and reduced maintenance costs. The implementation of embedded C programming with STM32 HAL libraries ensures efficient communication and data processing. The real-time monitoring capability helps industries optimize their operations, detect anomalies early, and reduce downtime.

The project showcases the potential of CAN-based industrial automation, which can be expanded with additional sensors, remote monitoring capabilities, and advanced control mechanisms. Future enhancements may include data logging, cloud connectivity, and alert systems to further improve industrial efficiency. This system serves as a cost-effective and scalable solution for modern industrial automation needs.

TABLE OF CONTENTS

S. No	Title	Page No.
	ABSTRACT	I
	Table of Content	II
	Acknowledgement	III
1	Introduction	4
1.1	Introduction	
1.2	Objective and Specifications	
2	Literature Review	5
3	System Design	7
4	Implementation	14
4.1	Implementation	
5	Results	20
5.1	Results	
6	Conclusion	21
6.1	Conclusion	
7	References	22
7.1	References	

Acknowledgment

We would like to express our sincere gratitude to our mentors and faculty members for their valuable guidance and support throughout this project. Their insights and expertise have been instrumental in successfully implementing the **Industrial Monitoring and Control System using CAN Protocol**. We also extend our thanks to our peers and family members for their encouragement and motivation. Lastly, we acknowledge the contributions of open-source communities and online resources that helped us understand and implement CAN-based communication effectively.

Chapter 1

Introduction

1.1 Introduction

Industrial automation and monitoring are essential for improving productivity, reducing downtime, and ensuring smooth operations in manufacturing and process industries. In industrial setups, multiple devices such as sensors, controllers, actuators, and display units must communicate efficiently for real-time monitoring and control. Traditional communication methods often suffer from high wiring complexity, data loss, and slower response times.

To address these challenges, the Controller Area Network (CAN) Protocol is widely used in industrial automation. CAN is a robust, high-speed, and reliable communication protocol that allows multiple devices to exchange data over a single two-wire bus. It supports error detection, priority-based message handling, and efficient data transmission without requiring a dedicated host computer.

This project focuses on developing an Industrial Monitoring and Control System using the CAN Protocol. The system integrates various sensors to collect real-time data such as temperature, pressure, and humidity. This sensor data is transmitted over a CAN bus network to controllers, which process the information and send appropriate control signals to actuators or display units. An OLED display or a monitoring unit provides real-time visual feedback, allowing operators to monitor industrial processes effectively.

This project demonstrates the practical application of the CAN protocol in industrial automation, showcasing its ability to improve communication efficiency, system reliability, and overall process optimization.

1.1 Objective

- Acquire real-time temperature and pressure data
 - Use BMP280 sensor for accurate environmental measurements.
 - Implement I2C protocol for data acquisition with STM32.
- Implement high-speed and reliable CAN communication
 - Establish STM32-to-STM32 data transmission using CAN protocol.
 - Ensure low-latency and error-free communication.
- Develop a Gateway Node for data processing and transmission
 - Design the second STM32 as a Gateway Node to receive data via CAN.
 - Process and forward the data to ESP32 using UART communication.
- Enable cloud-based remote monitoring using AWS IoT Core
 - Implement MQTT protocol on ESP32 for wireless data transmission.
 - Configure AWS IoT Core for real-time data storage and visualization.
- Integrate an OLED display for local visualization
 - Display real-time temperature and pressure readings.
 - Ensure clear and accurate data representation.

Chapter 2

LITERATURE REVIEW

The Controller Area Network (CAN) Protocol, developed by Robert Bosch, is a widely used communication standard in industrial automation due to its high reliability, efficiency, and real-time performance. It is a multi-master, message-based protocol that allows multiple devices to communicate with minimal wiring while ensuring error-free data exchange through Cyclic Redundancy Check (CRC) and arbitration mechanisms.

Research studies highlight the use of CAN in sensor-based monitoring systems, particularly in real-time temperature, pressure, and machine health monitoring. These systems enable industries to track critical parameters, improve operational efficiency, and reduce downtime through predictive maintenance.

CAN-based systems have been successfully implemented using STM32, Arduino, and PIC microcontrollers, demonstrating the protocol's flexibility, ease of integration, and scalability. The availability of CAN-compatible hardware and software libraries makes it an ideal choice for industrial automation, robotics, and vehicle communication systems.

Studies comparing CAN with traditional communication protocols like RS-232 and RS-485 indicate that CAN outperforms them in terms of speed, multi-node support, error detection, and bus arbitration. Unlike RS-232, which supports only point-to-point communication, and RS-485, which requires complex addressing, CAN efficiently handles multiple nodes while ensuring data integrity and collision resolution.

Chapter 3

Methodology and Techniques

3.1 Methodology:

The Industrial Monitoring and Control System using CAN Protocol follows a structured approach to ensure efficient data transmission and real-time monitoring. The system is designed to collect sensor data, transmit it over the CAN bus, and display the results for industrial process control. The methodology consists of the following steps:

System Architecture

The system architecture consists of multiple interconnected components designed to facilitate seamless data acquisition, transmission, and display. The setup includes two STM32 microcontrollers, two CAN bus modules, a BMP280 sensor, and an OLED display. Each component plays a specific role in ensuring efficient monitoring and control of industrial parameters:

STM32 Microcontrollers: Two STM32 microcontrollers are used—one acting as a sensor node and the other as a display node. These microcontrollers handle sensor data acquisition, processing, and transmission over the CAN network.

CAN Bus Modules: Each microcontroller is connected to a CAN transceiver module, allowing reliable communication between the nodes. The CAN protocol ensures real-time, error-free data transmission.

BMP280 Sensor: This sensor is used to capture real-time temperature and pressure data, which is essential for industrial process monitoring.

OLED Display: The OLED screen is used to display sensor readings in real-time, allowing operators to monitor system parameters effectively.

Data Acquisition

Data acquisition is a crucial step where the system collects environmental data for monitoring and control. The BMP280 sensor is connected to the first STM32 microcontroller, which continuously reads temperature and pressure values. The sensor operates using I2C communication, allowing efficient data retrieval with

minimal latency. The collected data is processed and prepared for transmission over the CAN network.

CAN Communication

The first STM32 microcontroller, acting as a sender, transmits the collected sensor data via the CAN protocol. The CAN bus follows a multi-master architecture, meaning multiple devices can communicate over the same network without interference. The key steps in CAN communication include:

Data Packaging: Sensor readings are formatted into CAN frames, ensuring efficient transmission.

Transmission Process: The STM32 microcontroller sends the data packets through the CAN transceiver module, which converts the microcontroller's digital signals into differential signals suitable for CAN communication.

Error Handling: The CAN protocol incorporates error detection mechanisms such as CRC (Cyclic Redundancy Check) and arbitration, ensuring data integrity and minimizing communication errors.

Data Processing and Display

The second STM32 microcontroller, acting as a receiver, listens to the CAN network for incoming data packets. Upon receiving the sensor data, it performs the following operations:

Data Decoding: The received CAN frames are parsed, and temperature and pressure values are extracted.

Processing and Filtering: The microcontroller verifies the received data for consistency and removes any erroneous values that may affect accuracy.

Displaying Data: The processed sensor readings are displayed on the OLED screen, enabling real-time monitoring of industrial conditions.

Control and Response: Based on the monitored values, corrective actions can be taken to maintain industrial efficiency. The system can be expanded to include actuators, alarms, or automated control mechanisms that respond to threshold violations, ensuring optimal industrial operations.

3.2 Techniques:-

Embedded System Design

The STM32 microcontrollers are programmed using Embedded C and the STM32 HAL (Hardware Abstraction Layer) library. These libraries provide an abstraction layer for easier handling of peripherals like GPIO, I2C, and CAN. The main functionalities of embedded system design in this project include:

Sensor Interfacing: Writing firmware to communicate with the BMP280 sensor using I2C.

CAN Protocol Implementation: Configuring the CAN peripheral on the STM32 microcontrollers to enable data transmission and reception.

Interrupt Handling: Implementing interrupt-driven communication to ensure efficient data processing with minimal delay.

Power Management: Optimizing power consumption by configuring sleep modes where necessary.

CAN Bus Communication

To enable reliable and efficient communication, the CAN protocol is implemented with key features such as:

Message Filtering: The CAN controller is programmed to accept only relevant messages, reducing processing overhead.

Priority Handling: Messages with higher priority (lower ID values) are transmitted first, ensuring critical data is not delayed.

Error Detection: CRC and acknowledgment mechanisms detect and correct errors in transmitted data.

Bus Arbitration: The non-destructive arbitration method allows multiple nodes to transmit without conflicts, ensuring seamless data flow.

Real-Time Data Processing

Efficient real-time data processing is achieved using a combination of polling and interrupt-based techniques. The system ensures continuous monitoring and fast response times through:

Polling Method: Periodic reading of sensor values at predefined intervals.

Interrupt-Driven Communication: Using interrupts to immediately handle critical data reception over CAN, minimizing latency.

Data Filtering: Removing outliers and inconsistent sensor values to enhance reliability.

Timestamping: Assigning timestamps to received data to track historical trends and improve monitoring accuracy.

OLED Display Integration

The OLED display is connected via the I2C protocol to the STM32 microcontroller and is used for real-time data visualization. The key steps in OLED display integration include:

I2C Communication Setup: Configuring the I2C peripheral for communication between the microcontroller and the OLED screen.

Data Formatting: Converting sensor readings into a readable format before displaying.

Graphical Representation: Using graphical libraries to display temperature and pressure values in an organized manner.

Refreshing Mechanism: Updating the OLED screen at regular intervals to ensure real-time monitoring.

This structured approach ensures that industrial monitoring and control are effectively implemented, leading to improved efficiency and reduced downtime.

Chapter 4

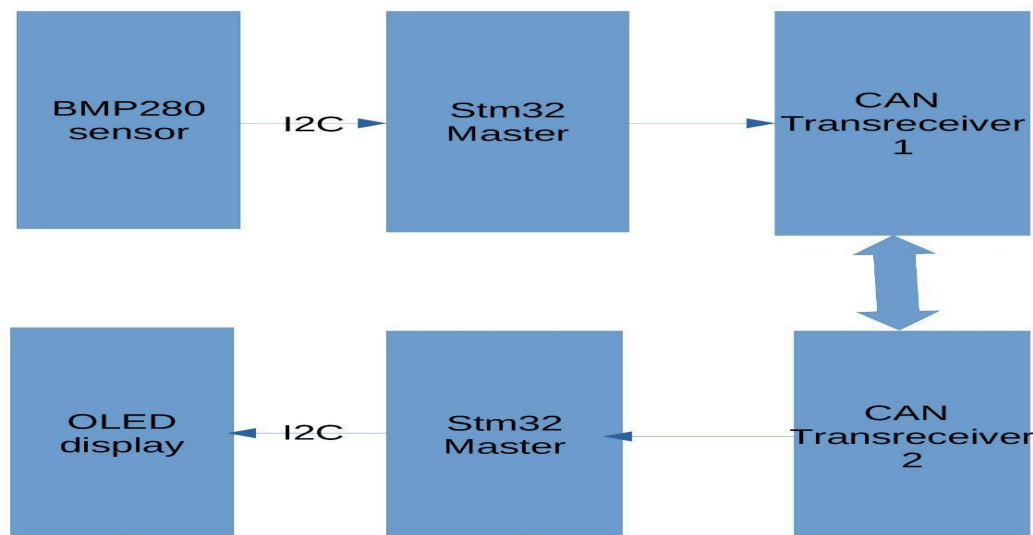
Implementation

1. Introduction

The implementation of the Industrial Monitoring and Control System using the CAN protocol involves setting up hardware components, configuring software, and ensuring proper communication between the transmitter and receiver. The system consists of two STM32 microcontrollers, where one acts as a transmitter and the other as a receiver. The transmitter reads sensor data and sends it over the CAN bus, while the receiver processes the data and displays it on an OLED screen. This section details the step-by-step implementation process, covering hardware setup, software configuration, coding, testing, and debugging procedures.

2. STM32 Pin Configuration using STM32CubeIDE

To ensure proper communication between the BMP280 sensor, CAN bus, and OLED display, STM32CubeIDE is used to configure the required pins and peripherals. The microcontrollers are set up with specific configurations to facilitate seamless data exchange.



2.1 Transmitter STM32 (Reading BMP280 and Sending Data via CAN) I2C Interface Configuration

The BMP280 sensor operates using the I2C protocol.

The STM32 microcontroller's SDA (data) and SCL (clock) pins are assigned to enable communication with the sensor.

The I2C peripheral is initialized in STM32CubeIDE, setting the clock speed and addressing mode.

CAN Interface Configuration

The CAN peripheral is initialized to enable message transmission over the CAN bus.

The CAN_TX and CAN_RX pins are mapped correctly to ensure seamless communication.

The CAN baud rate is set appropriately to match the receiver.

Filters and interrupts are configured for efficient data transmission.

GPIO Configuration

Additional GPIO pins are set up to provide power to the sensor and manage control signals.

The microcontroller's onboard LED is programmed to blink when data transmission occurs, serving as an activity indicator.

2.2 Receiver STM32 (Receiving CAN Data and Displaying on OLED)

CAN Interface Configuration

The receiver STM32 is programmed to listen for incoming data on the CAN bus.

The same CAN_RX and CAN_TX pins are configured for message reception.

Filters are set up to accept only relevant messages.

The interrupt service routine (ISR) is enabled to handle incoming data efficiently.

I2C Interface Configuration

The OLED display communicates via I2C protocol.

The STM32 microcontroller acts as a master device to send display updates.

The necessary I2C parameters such as clock speed and addressing mode are configured.

GPIO Configuration

Additional GPIO settings are configured for controlling the OLED display.

A user button is set up for potential interaction, such as switching between different display modes.

3. Code Implementation

The firmware for the STM32 microcontrollers is written in Embedded C using the STM32 HAL (Hardware Abstraction Layer) library. The code ensures smooth operation of data acquisition, transmission, reception, and display updates.

3.1 Transmitter Code (Sensor Data Acquisition and Transmission)

Initialize the BMP280 sensor using the I2C protocol

Configure I2C in master mode.

Set up BMP280-specific registers to read temperature and pressure data.

Read temperature and pressure data from the sensor

Use HAL functions to retrieve raw sensor values.

Convert raw data to readable temperature and pressure values using calibration formulas.

Format data into CAN message frames

Store sensor values in a structured format within a CAN message.

Assign appropriate message IDs and priorities.

Send the formatted data using the CAN transmit function

Load the data into the CAN transmit buffer.

Enable transmission using HAL_CAN_Transmit() function.

Implement error handling to ensure reliable communication

Configure error-detection mechanisms using CAN error status registers.

Implement retransmission in case of message failure.

3.2 Receiver Code (Processing and Displaying Data)

3.2.1: Initialize the CAN module and set up message filtering:

Define message acceptance criteria to filter unnecessary data.

Enable interrupts for handling incoming messages in real-time.

3.2.2: Receive data frames from the transmitter over the CAN bus:

Read incoming messages using `HAL_CAN_GetRxMessage()`.

Check message integrity and validity before processing.

3.2.3: Extract temperature and pressure values from the received messages:

Parse the received CAN frame to retrieve temperature and pressure values.

3.2.4: Format the extracted data for display:

Convert numerical values into a readable format.

Prepare display strings for the OLED.

3.2.5: Send formatted data to the OLED screen for real-time monitoring:

Use I2C communication to update the OLED display.

Implement periodic refresh to ensure updated values are shown.

4. Testing and Debugging

To validate the implementation, multiple tests are performed to ensure the system operates as expected.

4.1 CAN Communication Verification

Verify that both STM32 microcontrollers successfully send and receive messages over the CAN bus.

Check if messages are received without data corruption.

Use debugging LEDs to indicate successful transmission and reception.

4.2 Sensor Data Accuracy

Compare BMP280 sensor readings with expected values to confirm correct data acquisition.

Cross-check sensor readings with a known temperature and pressure source.

4.3 OLED Display Testing

Ensure the received data is correctly displayed on the OLED screen.

Verify that the display updates without flickering or lag.

4.4 Debugging Using STM32CubeIDE

Utilize breakpoints and real-time debugging tools to identify and fix errors.

Check peripheral registers for correct initialization and operation.

4.5 Monitoring CAN Signals

Use an oscilloscope or logic analyzer to visualize CAN bus signals.

Confirm that message timing and structure follow the expected format.

Chapter 5

System Output and Observations

This chapter presents the actual output obtained from the Automotive Features Monitoring via CAN Communication Protocol project. The results demonstrate the successful implementation of sensor data acquisition, transmission via CAN and UART, and real-time visualization on OLED and AWS IoT Core.

5.1. Data Acquisition and Transmission

The BMP280 sensor successfully collects temperature and pressure data and transmits it to the STM32 Data Acquisition Node via I2C communication. The acquired data is formatted and sent to the STM32 Gateway Node over the CAN Bus, ensuring low-latency and reliable communication.

The screenshot displays the STM32CubeIDE interface. The main editor shows the `CAN_Loopback.ioc` file with the following code:

```

101
102
103 /* MCU Configuration-----*/
104
105 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
106 HAL_Init();
107
108 /* USER CODE BEGIN Init */
109
110 /* USER CODE END Init */
111
112 /* Configure the system clock */
113 SystemClock_Config();
114
115 /* USER CODE BEGIN SysInit */
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

The console output shows the following data:

```

Port 0 X
Temperature: 28.79A°C, Pressure: 242445.97 hPa
CAN Transmission SUCCESS
Temperature: 28.80A°C, Pressure: 242458.79 hPa
CAN Transmission SUCCESS
Temperature: 28.79A°C, Pressure: 242439.3 hPa
CAN Transmission SUCCESS
Temperature: 28.79A°C, Pressure: 242452.90 hPa
CAN Transmission SUCCESS
Temperature: 28.79A°C, Pressure: 242459.84 hPa
CAN Transmission SUCCESS
Temperature: 28.80A°C, Pressure: 242449.92 hPa
CAN Transmission SUCCESS
Temperature: 28.80A°C, Pressure: 242465.74 hPa
CAN Transmission SUCCESS

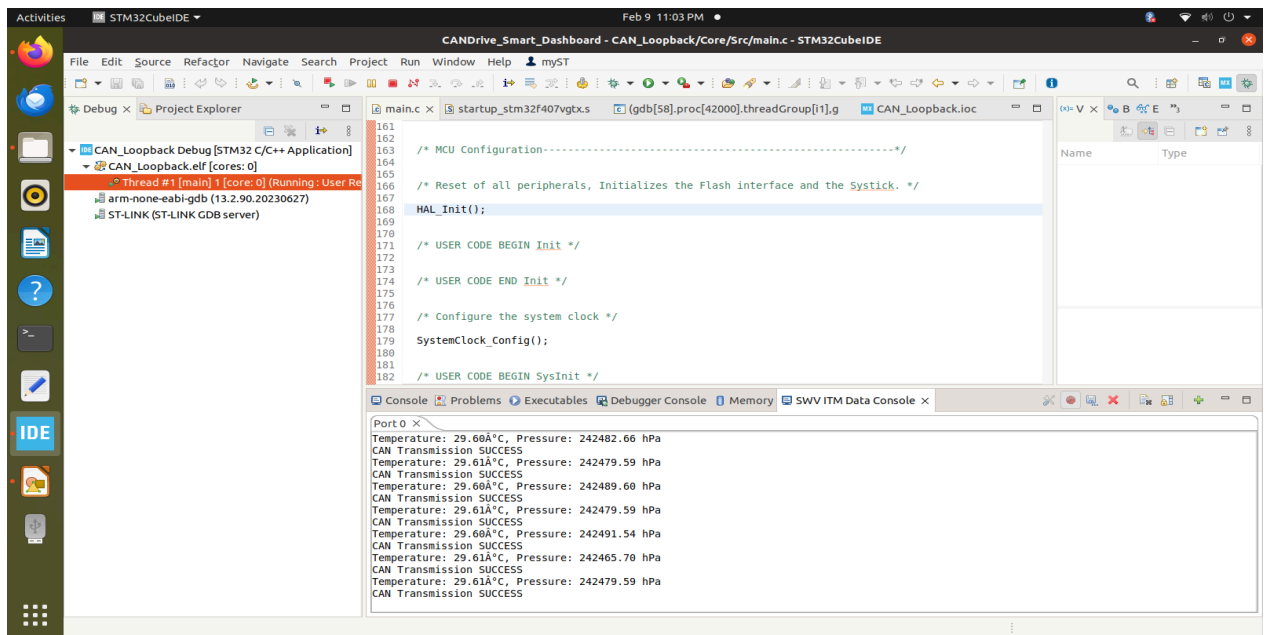
```

5.2. OLED Display Output

The real-time sensor data is displayed on the OLED screen, ensuring local monitoring. The output on the OLED includes:

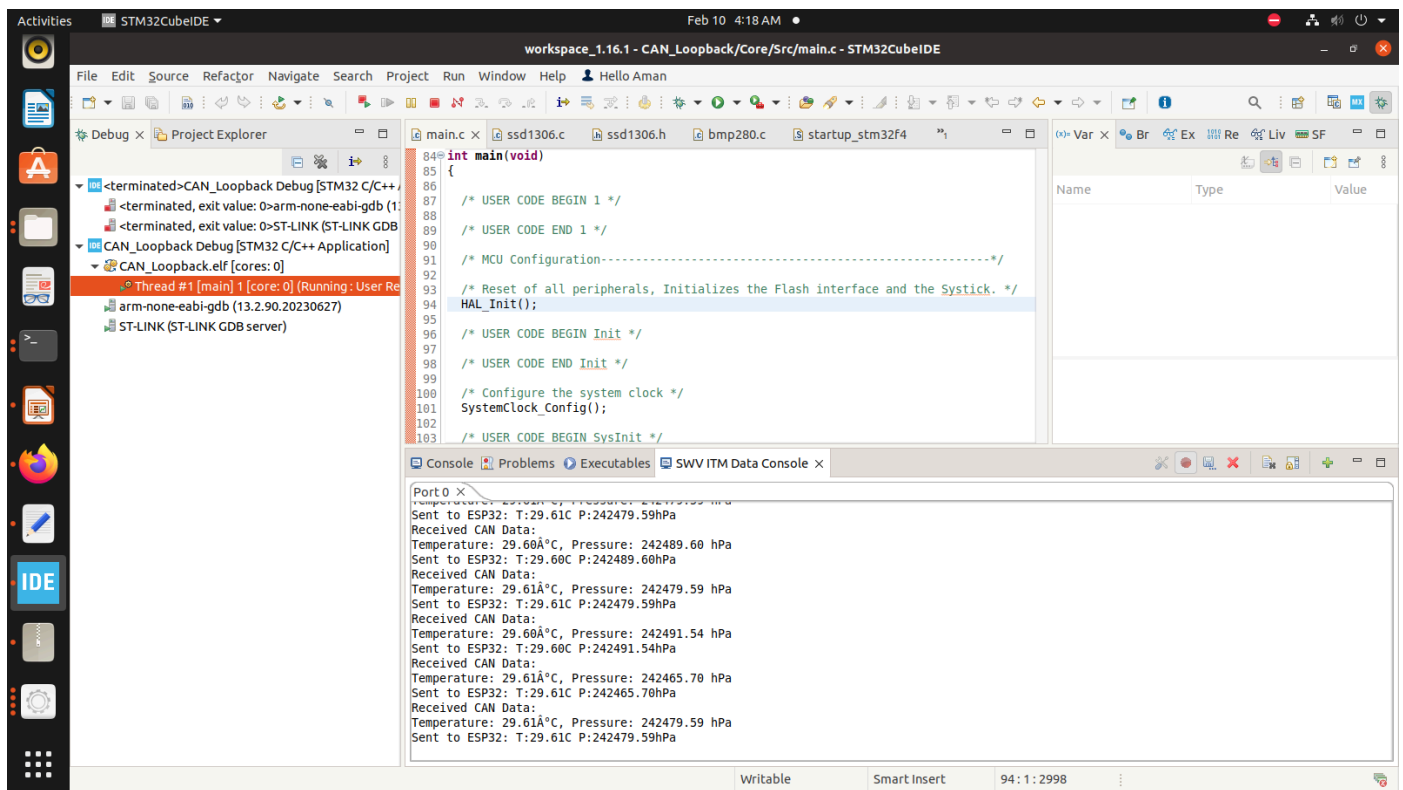
- Temperature readings in Celsius
- Pressure readings in hPa

(screenshot of OLED displaying sensor data.)



5.3. CAN Communication Output

The CAN Bus enables robust communication between STM32 nodes. The received CAN messages contain sensor data in a structured format.



The screenshot shows the STM32CubeIDE interface with the following components:

- Project Explorer:** Shows the project structure with files like `main.c`, `ssd1306.c`, `ssd1306.h`, `bmp280.c`, and `startup_stm32f4`.
- Code Editor:** Displays the `main.c` file with the following code:


```

84 int main(void)
85 {
86     /* USER CODE BEGIN 1 */
87     /* USER CODE END 1 */
88     /* MCU Configuration-----*/
89     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
90     HAL_Init();
91     /* USER CODE BEGIN Init */
92     /* USER CODE END Init */
93     /* Configure the system clock */
94     SystemClock_Config();
95     /* USER CODE BEGIN SysInit */
96     /* USER CODE END SysInit */
97
98     /* Configure the system clock */
99     SystemClock_Config();
100
101     /* USER CODE BEGIN SysInit */
102     /* USER CODE END SysInit */
103

```
- Console:** Shows the output of the program, including sensor data received via CAN and sent to ESP32:


```

Port 0 X
Received CAN Data:
Temperature: 29.60°C, Pressure: 242489.60 hPa
Sent to ESP32: T:29.61C P:242479.59hPa
Received CAN Data:
Temperature: 29.61°C, Pressure: 242479.59 hPa
Sent to ESP32: T:29.61C P:242479.59hPa
Received CAN Data:
Temperature: 29.60°C, Pressure: 242491.54 hPa
Sent to ESP32: T:29.60C P:242491.54hPa
Received CAN Data:
Temperature: 29.61°C, Pressure: 242465.70 hPa
Sent to ESP32: T:29.61C P:242465.70hPa
Received CAN Data:
Temperature: 29.61°C, Pressure: 242479.59 hPa
Sent to ESP32: T:29.61C P:242479.59hPa

```

Chapter 6

Conclusion

6.1 Conclusion

This project successfully implements an Industrial Monitoring and Control System using the CAN (Controller Area Network) protocol. The system is designed to facilitate real-time data acquisition, transmission, and display, ensuring efficient monitoring and control of industrial processes.

By utilizing STM32 microcontrollers, a BMP280 sensor, and an OLED display, the project demonstrates a seamless integration of hardware and software for industrial automation. The BMP280 sensor is responsible for measuring temperature and pressure, which are crucial parameters in industrial environments. The sensor communicates with the transmitter STM32 microcontroller via the I2C protocol, ensuring accurate data acquisition. The collected data is then formatted into CAN message frames and transmitted over the CAN bus.

On the receiving end, another STM32 microcontroller is configured to listen for incoming CAN messages, extract sensor values, and update the OLED display using the I2C interface. This real-time monitoring capability allows industrial operators to track environmental conditions with high accuracy and minimal delay.

The CAN bus plays a pivotal role in ensuring fast, reliable, and error-free communication. Its robust error-handling mechanisms, such as automatic retransmission and error detection, make it an ideal choice for industrial applications where data integrity and real-time performance are critical. The system also benefits from the noise immunity and multi-node communication capabilities of the CAN protocol, making it scalable for larger industrial networks.

In addition to its core functionality, the project highlights the advantages of using STM32CubeIDE for hardware configuration and debugging. By leveraging STM32 HAL libraries, the firmware implementation is simplified while maintaining efficiency and modularity.

Furthermore, testing procedures such as CAN communication verification, sensor data validation, and OLED display accuracy checks ensure system reliability.

Overall, this project successfully showcases the effectiveness of CAN-based monitoring systems in enhancing industrial automation and control. By implementing a reliable and scalable communication framework, the system reduces downtime, improves operational efficiency, and contributes to a smarter industrial environment. Future enhancements could include integrating additional sensors, implementing data logging capabilities, and extending the system with wireless connectivity for remote monitoring.

6.2 Future Enhancement –

Enhancements for Industrial Monitoring and Control System Using CAN Protocol

1. Multiple Sensor Integration

Expanding the system to incorporate additional sensors can significantly enhance monitoring capabilities. By integrating sensors for humidity, gas detection, and vibration, the system can provide a more comprehensive assessment of industrial environments. Humidity sensors help monitor moisture levels to prevent equipment corrosion, gas sensors detect hazardous leaks for worker safety, and vibration sensors assess machine health to predict failures. The CAN protocol allows seamless data acquisition from multiple sensors without signal interference, ensuring real-time, reliable monitoring.

2. Wireless Connectivity

To extend the reach of industrial monitoring beyond wired networks, CAN-to-Wi-Fi or CAN-to-IoT integration can be implemented. This enables remote monitoring, where sensor data is transmitted to a central server, cloud platform, or mobile application. Industrial supervisors can receive real-time updates on environmental conditions, machine status, and potential hazards, even when they are off-site. Wi-Fi connectivity ensures flexibility, while IoT integration supports predictive maintenance by analyzing trends and sending alerts before failures occur.

3. Data Logging

A key improvement to the system is the addition of data logging capabilities. By storing sensor data on an SD card or cloud platform, industries can analyze historical trends, detect anomalies, and optimize performance. Data logging is essential for regulatory compliance, quality control, and failure analysis. Implementing a timestamp feature allows accurate tracking of environmental changes over time, helping industries make informed decisions and improve operational efficiency.

4. Alert System

An automated alert system can be introduced to enhance safety and responsiveness. If critical conditions such as high temperatures, excessive humidity, or gas leaks are detected, the system can trigger alarms, LED indicators, or push notifications. SMS or email alerts can be integrated

to notify relevant personnel instantly. This proactive approach reduces risks, prevents equipment damage, and ensures a safer industrial environment. Configurable thresholds can be set to customize alert levels based on specific industry requirements.

5. Improved Display Interface

Upgrading from a basic OLED display to a graphical LCD or touchscreen interface can improve data visualization. A graphical display allows better representation of sensor readings through charts, gauges, and real-time graphs. A touchscreen interface enhances user interaction by enabling menu navigation, sensor calibration, and data analysis directly on the device. With a more intuitive display, operators can quickly interpret data, take corrective actions, and optimize industrial processes effectively.

By incorporating these enhancements, the Industrial Monitoring and Control System can become more robust, versatile, and efficient in addressing modern industrial automation challenges.

Chapter 7

References

Here are the links for your reference:

1. [BME280 Sensor with STM32 Nucleo using STM32CubeIDE](#)
2. [How to Interface BME280 Sensor with STM32](#)
3. [BMP280_STM32 Interface \(GitHub\)](#)
4. [How to use CAN Protocol in STM32](#)
5. [CAN Protocol – How It Works and Where It's Used?](#)