

SMART BOREWELL CHILD RESCUE SYSTEM

A PROJECT REPORT

*Submitted by*

|  |  |
| --- | --- |
| **S. GUNASUNDARI** | **(822718106018)** |
| **R.J** | **(822718106020)** |
| **K.L** | **(822718106028)** |
| **Z.R** | **(822718106040)** |

***in partial fulfilment for the award of the degree of***

**BACHELOR OF ENGINEERING**

# IN

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING **GOVERNMENT COLLEGE OF ENGINEERING, THANJAVUR ANNA UNIVERSITY: CHENNAI - 600 025**

JUNE 2022

ANNA UNIVERSITY: CHENNAI 600 025

**BONAFIDE CERTIFICATE**

Certified that this project report **“SMART BOREWELL CHILD RESCUE SYSTEM**” is the bonafide work of **“GUNASUNDARI S (822718106018), JANANI R (822718106020), LAKSHMI K (822718106028)**

**and RUKSHANA AZEEMA Z (822718106040)**” who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Mr. R. RAMJI, M.E., Mr. S. MOHAN, M.E**

**HEAD OF THE DEPARTMENT SUPERVISOR ASSISTANT PROFESSOR**

Department of ECE Department of ECE

Government College of Engineering Government College of Engineering Thanjavur-613 402 Thanjavur-613 402

Submitted for the project viva voce held on………………………

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# ACKNOWLEDGEMENT

This project work is dedicated to Almighty God blessing with inspirational parents, teachers and good friends.

We would like to express our deep gratitude to our beloved Principal **Dr. M. NATARAJ., M.E., Ph.D.** for his moral support and words of enthusiasm which helped us a lot in completing this project.

We would like to thank **Mr. R. RAMJI M.E** Head of Electronics and Communication Engineering department for his invaluable guidance and encouragement in completing this project work.

We would like to express our deep gratitude and thanks to our well- wisher and internal guide **Mr**. **S. MOHAN M.E** for his invaluable guidance, advice, moral support and encouragement without which this project would not be successful. We would like to extend our sincere thanks to our project coordinators for their constant efforts to bring this project successful.

We also like to thank all the respected staff members of Electronics and Communication department and various department of our college for their direct and indirect involvement in successful completion of this project.

We like to express our sincere thanks and gratitude to our parents and friends for their continuous encouragement and support.

# ABSTRACT

In order to meet the ever-increasing demand for water, bore wells are dug. But these are usually left uncovered and children fall into it. The main aim of our project is to save a child from the bore well, so we proposed a system of designing an adjustable diameter robot for the rescue of a child from a bore well. We aid the child by continuous monitoring using a camera and supply of necessary items mainly; air filler which supplies oxygen the survival. Robot for bore well rescue offers a solution to this situation. This system will attach a harness to a child using robotic arms for picking up. It includes an infrared transmitter and receiver to calculate the distance to the child. A temperature sensor is used to measure temperature and gas sensor is used to detect the presence of any toxic gas. The proposed system will easily rescue the child without major injury.

# TABLE OF CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO** |  | **TITLE** | **PAGE NO** |
|  |  | **ABSTRACT** | **iv** |
|  |  | **LIST OF FIGURES** | **vii** |
|  |  | **LIST OF TABLES** | **viii** |
|  |  | **LIST OF ABBREVIATIONS** | **ix** |
| **1** |  | **INTRODUCTION** | **1** |
| **2** |  | **LITERATURE REVIEW** | **3** |
| **3** |  | **EXISTING SYSTEM** | **5** |
| **4** |  | **PROPOSED SYSTEM** | **6** |
|  | **4.1** | BLOCK DIAGRAM DESCRIPTION | **8** |
|  |  | 4.1.1 PIC MICROCONTROLLER | **8** |
|  |  | 4.1.2 LCD DISPLAY | **13** |
|  |  | 4.1.3 TEMPERATURE SENSOR | **13** |
|  |  | 4.1.4 RELAY OPERATION | **14** |
|  |  | 4.1.5 ALARM OPERATION | **14** |
|  | **4.2** | OVERALL CIRCUIT DIAGRAM  DESCRIPTION | **15** |
|  |  | 4.2.1 POWER SUPPLY | **16** |
|  |  | 4.2.2 MICROCONTROLLER CIRCUIT | **17** |
|  |  | 4.2.3 IR SENSOR CIRCUIT | **18** |
|  |  | 4.2.4 RELAY DRIVER CIRCUIT | **20** |
|  |  | 4.2.5 ALARM CIRCUIT | **21** |
|  |  | 4.2.6 GAS SENSOR CIRCUIT | **22** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **4.3** | PCB DESIGN | **22** |
|  |  | 4.3.1 INTRODUCTION | **22** |
|  |  | 4.3.2 MANUFACTURING | **23** |
|  |  | 4.3.3 PANALISATION | **23** |
|  |  | 4.3.4 DRILLING | **23** |
|  |  | 4.3.5 PLATING | **23** |
|  |  | 4.3.6 ETCHING | **24** |
|  |  | 4.3.7 SOLDER MASK | **24** |
|  |  | 4.3.8 HOT AIR LEVELING | **24** |
|  |  | 4.3.9 FLOWCHART OF WORKING  PRINCIPLE | **25** |
|  | **4.4** | SOFTWARE TOOLS | **26** |
|  |  | 4.4.1 MPLAB | **26** |
|  |  | 4.4.2 MPLAB SIMULATOR | **27** |
|  |  | 4.4.3 IC PROG | **27** |
|  |  | 4.4.4 COMPILER HI-TECH C | **28** |
|  |  | 4.4.5 PIC START PLUS PROGRAMMER | **28** |
| **5** |  | **CONCLUSION** | **29** |
|  | **5.1** | **FUTURE SCOPE** | **30** |
|  |  | **REFERENCES** | **31** |
|  |  | **APPENDIX** | **33** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **FIGURE TITLE** | **PAGE NO** |
| **4.1** | Block diagram of Equipment | **7** |
| **4.2** | Block diagram of Robotic Arm | **7** |
| **4.3** | Pin diagram | **10** |
| **4.4** | Overall circuit diagram | **15** |
| **4.5** | Pic Microcontroller circuit | **17** |
| **4.6** | IR transmitter circuit | **18** |
| **4.7** | Relay driver circuit | **20** |
| **4.8** | Alarm circuit | **21** |
| **4.9** | Gas sensor circuit | **22** |
| **4.10** | Flowchart of Working Principle | **25** |

# LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| **TABLE NO** | **TABLE TITLE** | **PAGE NO** |
| **4.1** | Specifications Of Microcontroller | **9** |
| **4.2** | Pin out Description-1 | **11** |
| **4.3** | Pin out Description-2 | **12** |

**LIST OF ABBREVIATIONS**

|  |  |  |
| --- | --- | --- |
| **SI.N0** | **ABBREVIATION** | **EXPANSION** |
| 1 | ADC | Analog to Digital Converter |
| 2 | AODV | Ad hoc On Demand Distance Vector |
| 3 | BAODV | Bus Ad-hoc On Demand Distance Vector  Routing Protocol |
| 4 | CMOS | Complementary Metal Oxide Semiconductor |
| 5 | LED | Light Emitting Diode |
| 6 | IR | Infra Red |
| 7 | PBRS | Performance Based Requirement Set |
| 8 | PDR | Portable Data Recorder |
| 9 | PIC | Peripheral Interface Controller |
| 10 | RISC | Reduced Instruction Set Computer |

# CHAPTER 1

# INTRODUCTION TO EMBEDDED C

**1. Introduction:** Embedded C is a programming language specifically tailored for developing software for embedded systems. Embedded systems are specialized computing devices designed to perform dedicated functions or control specific hardware components within larger systems. Embedded C is widely used due to its efficiency, portability, and close integration with hardware, making it an essential tool for embedded software developers.

**2. Characteristics of Embedded C:**

Efficiency: Embedded C is designed to produce efficient code with minimal overhead, making it suitable for resource-constrained environments common in embedded systems.

Low-Level Access: Embedded C allows direct manipulation of hardware registers and memory addresses, enabling fine-grained control of the hardware, which is crucial in embedded systems programming.

Portability: Though specific to each processor architecture, Embedded C can be adapted to different platforms, facilitating code reuse and porting between various embedded systems.

Real-Time Capabilities: Many embedded systems require real-time responsiveness, and Embedded C's low-level access and predictable execution times make it well-suited for real-time applications.

**3. Data Types and Memory Management:** Embedded C supports data types like integers, characters, floating-point numbers, and user-defined structures. Memory management is critical in embedded systems due to limited resources. Pointers are often used to access and manipulate memory, and developers must be mindful of memory allocation and deallocation to avoid memory leaks.

**4. I/O Operations:** Embedded C provides methods to perform Input/Output (I/O) operations, enabling communication with peripherals and sensors. Low-level I/O functions are used to read from and write to hardware registers directly.

**5. Interrupt Handling**: In embedded systems, interrupts are essential for handling time-critical events. Embedded C provides mechanisms to handle interrupts and define interrupt service routines (ISRs) to respond to external events promptly.

**6. Cross-Compilation and Toolchains:** Embedded C code is typically cross-compiled using a toolchain that generates machine code for the target architecture. Toolchains include a compiler, assembler, and linker, allowing developers to compile code on a development machine and then flash it onto the target embedded system.

**7. Development Environments:** Developers can use Integrated Development Environments (IDEs) or text editors along with debugging tools and simulators to facilitate embedded C development. IDEs often provide features like code highlighting, code completion, and hardware debugging.

**8. Applications of Embedded C:** Embedded C finds applications in various domains, including:

Automotive: In vehicle control systems, engine management, infotainment systems, etc.

Consumer Electronics: In smartphones, digital cameras, home appliances, etc.

Industrial Automation: In programmable logic controllers (PLCs), motor control systems, etc.

Internet of Things (IoT): In connected devices, smart home systems,

# CHAPTER 2

# RISC vs CISC ARCHITECTURE

**2.1 RISC Architecture**

**2.1.1 Definition:**

RISC architecture is based on the principle of simplicity and minimalism. It advocates using a reduced set of simple and atomic instructions that execute in one clock cycle, resulting in streamlined and faster execution.

**2.1.2 Characteristics:**

Simple Instructions: RISC processors have a limited set of instructions, each performing a basic operation on data, making decoding and execution more straightforward.

Load-Store Architecture: RISC processors predominantly use a load-store architecture, where data operations are performed only between registers, and memory access is done explicitly through load and store instructions.

Hardwired Control Unit: RISC processors often employ a hardwired control unit, leading to a fixed instruction execution flow, which contributes to their speed.

**2.1.3 Advantages:**

Faster Execution: Due to the simplicity of instructions and hardwired control units, RISC processors generally achieve faster execution times.

Pipelining: The straightforward instruction set allows for efficient pipelining, further enhancing performance.

Reduced Complexity: Simplicity of design leads to easier verification, debugging, and overall implementation.

**2.1.4 Disadvantages:**

Code Size: RISC instructions being simpler may result in more instructions needed for certain complex tasks, leading to larger code size.

Memory Access Overhead: RISC architectures, with their load-store model, may incur additional memory access overhead for certain operations.

**2.2 CISC Architecture**

**2.2.1 Definition:**

CISC architecture, in contrast to RISC, emphasizes complex and multi-step instructions that can perform sophisticated tasks in a single instruction.

**2.2.2 Characteristics:**

Rich Instruction Set: CISC processors have a large and diverse instruction set that includes complex operations, memory access, and addressing modes.

Memory-to-Memory Operations: CISC instructions often allow direct memory-to-memory operations, reducing the need for explicit register manipulation.

Microcoded Control Unit: CISC processors typically use microcoded control units, which allow more flexible instruction execution paths.

**2.2.3 Advantages:**

Compact Code: CISC instructions can perform complex tasks in a single instruction, leading to more compact code.

Fewer Memory Accesses: The ability to perform memory-to-memory operations reduces the number of instructions needed for certain tasks.

Better for High-Level Languages: CISC's rich instruction set makes it well-suited for high-level language compilation, as it can closely map high-level constructs to machine instructions.

**2.2.4 Disadvantages:**

Slower Execution: The complex instruction set and microcoded control unit may lead to longer execution times for individual instructions & More Complex Design.

**2.4 Difference between RISC and CISC**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **RISC Architecture** | **CISC Architecture** |
| Definition | Based on simplicity and minimalism. | Emphasizes complex and multi-step instructions. |
| Instruction Set | Limited set of simple instructions. | Large and diverse instruction set. |
| Memory Access | Primarily load-store architecture. | Supports memory-to-memory operations. |
| Control Unit | Hardwired control unit. | Microcoded control unit. |
| Execution Speed | Generally faster execution due to simplicity. | Can be slower due to complex instructions |
| Code Size | May have larger code size for some tasks. | Can achieve more compact code for complex operations. |
| Memory Access Overhead | Potential additional overhead for memory access. | Less memory access overhead due to direct operations. |
| Suitable Applications | Embedded systems, mobile devices, networking equipment. | Desktop computers, servers, high-performance computing. |

# CHAPTER 3

# ARM CORTEX M3 MICROCONTROLLER

**1. Introduction**: The ARM Cortex-M3 microcontroller is a 32-bit processor core designed by ARM Holdings, a leading semiconductor IP company. It is part of the Cortex-M family, which is specifically optimized for microcontroller applications requiring high performance, low power consumption, and a small footprint. The Cortex-M3 is widely used in various embedded systems, ranging from consumer electronics to industrial automation and automotive applications.

**2. Features and Architecture:**

32-bit RISC Architecture: The Cortex-M3 follows a Reduced Instruction Set Computing (RISC) architecture, which simplifies instruction execution and enables efficient pipelining, resulting in better performance.

Harvard Architecture: The Cortex-M3 features a Harvard architecture, with separate buses for instruction and data, facilitating simultaneous instruction fetch and data access.

Thumb-2 Instruction Set: The Cortex-M3 supports the Thumb-2 instruction set, which combines 16-bit Thumb instructions with selected 32-bit ARM instructions. This allows for improved code density and better performance compared to pure 32-bit architectures.

Nested Vectored Interrupt Controller (NVIC): The Cortex-M3 includes a sophisticated interrupt controller that supports up to 240 external interrupts with programmable priority levels, enabling efficient handling of real-time events.

Memory Protection Unit (MPU): Some variants of the Cortex-M3 include an MPU, allowing memory regions to be protected and access permissions to be enforced, enhancing system security and safety.

Single-Cycle Multiply and Hardware Divide: The Cortex-M3 includes single-cycle multiply and hardware divide instructions, facilitating efficient mathematical operations.

Low Power Design: The Cortex-M3 is designed for low power consumption, making it suitable for battery-powered and energy-efficient applications.

**3. Applications:** The ARM Cortex-M3 microcontroller is widely used in various applications, including:

Embedded Systems: The Cortex-M3 is a popular choice for general-purpose microcontroller applications, such as industrial control, home automation, and consumer electronics.

Automotive: It is used in automotive systems, including engine control units (ECUs), airbag systems, and infotainment systems.

Medical Devices: The Cortex-M3 is employed in medical devices like patient monitors, wearable health trackers, and diagnostic equipment.

Internet of Things (IoT): The Cortex-M3 is ideal for IoT devices and edge computing applications due to its low power consumption and efficient processing capabilities.

**4. Development Ecosystem**: Developers working with the Cortex-M3 microcontroller have access to a vast development ecosystem, including various integrated development environments (IDEs), compilers, debuggers, and evaluation boards from various semiconductor vendors. Additionally, ARM provides a standardized software development framework known as CMSIS (Cortex Microcontroller Software Interface Standard), which simplifies software development and portability across different Cortex-M based devices.

# CHAPTER 4

# SPECIAL FUNCTION REGISTERS

**1. Introduction:** Special Function Registers (SFRs) are an essential feature of microcontrollers and other embedded systems. They are hardware registers that provide direct access to various control and configuration settings of the microcontroller's peripherals and core functions. SFRs play a crucial role in configuring, controlling, and interacting with the microcontroller's hardware components, such as timers, I/O ports, interrupt controllers, and communication interfaces.

**2. Purpose of Special Function Registers:** SFRs serve several purposes, including:

Peripheral Configuration: SFRs allow the programmer to configure and control the behavior of the microcontroller's peripherals. For example, they can set the baud rate for a UART module, configure the operating mode of timers, or enable and disable specific features of communication interfaces like SPI or I2C.

Interrupt Control: SFRs often include registers that manage the interrupt system of the microcontroller. Programmers can use these registers to enable or disable specific interrupts, set their priorities, and clear pending interrupts.

Status and Control Bits: SFRs contain various status and control bits that provide information about the microcontroller's state and allow the programmer to control specific functionalities. For instance, they may indicate whether a communication buffer is full or empty, the status of an ongoing operation, or the presence of an error condition.

Power Management: In some microcontrollers, SFRs are used to control power-saving features and sleep modes, allowing the microcontroller to manage power consumption efficiently.

**3. Memory Mapped I/O:** SFRs are often memory-mapped, meaning they occupy specific addresses in the microcontroller's memory space. By reading from or writing to these memory-mapped addresses, the programmer can interact with the microcontroller's peripherals and core functions as if they were accessing regular memory locations.

**4. Direct Register Access:** Accessing SFRs usually involves reading or writing data directly to the corresponding memory-mapped addresses. The processor core of the microcontroller may include specific instructions to facilitate efficient access to these registers.

**5. SFR Examples:** Examples of Special Function Registers in a microcontroller may include:

GPIO Registers: Used for configuring I/O pins as inputs or outputs and for reading and writing digital signals to these pins.

UART Registers: Configure the UART communication settings, transmit and receive data, and manage communication-related flags.

Interrupt Control Registers: Manage the enabling, disabling, and prioritization of interrupts.

ADC/DAC Registers: Set up and control analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC) operations.

**6. Importance of SFRs in Embedded Systems:** Special Function Registers are fundamental to embedded systems programming, as they provide direct control over the microcontroller's peripherals and core functionalities. Their efficient use enables developers to interact with hardware components effectively, optimize system performance, and implement complex functionalities in various applications, ranging from simple sensing and control tasks to more sophisticated automation and communication systems.

# CHAPTER 5

# BASIC INPUT/OUTPUT CONFIGURATION

**1. Introduction:** Basic Input/Output (I/O) configuration is a fundamental aspect of embedded systems programming, enabling the interaction between the microcontroller and external devices, such as sensors, actuators, displays, and communication modules. I/O configuration involves configuring the microcontroller's General-Purpose Input/Output (GPIO) pins to either read digital inputs or drive digital outputs, allowing the system to sense and control the external world.

**2. GPIO Pins in Microcontrollers:** GPIO pins are versatile pins present in microcontrollers that can be programmed as either input or output pins based on the application's requirements. Each GPIO pin can be individually configured, making them suitable for a wide range of applications, from simple button inputs to complex communication protocols.

**3. Configuring GPIO as Input:** When a GPIO pin is configured as an input, it is used to read the state of an external digital signal. The microcontroller can detect whether the input is in a high (logic 1) or low (logic 0) state. Inputs are commonly used to interface with sensors, switches, or other digital devices. To prevent erratic readings, pull-up or pull-down resistors can be used to establish a defined state when the input is not actively driven.

**4. Configuring GPIO as Output:** When a GPIO pin is configured as an output, it can drive an external device, such as an LED, motor, or relay, by setting it to a high or low state. Outputs are used to control and actuate external devices based on the microcontroller's processing or user input. For safety and protection, external components like current-limiting resistors may be used with outputs to avoid damage to the microcontroller or the connected devices.

**5. I/O Configuration Registers:** In most microcontrollers, each GPIO pin has associated configuration registers. These registers allow the programmer to set the pin direction (input or output), configure the pin's pull-up/pull-down resistors, and define other pin-specific functionalities. Writing to these registers allows the programmer to control the behavior of individual GPIO pins.

**6. I/O Multiplexing:** In some microcontrollers, GPIO pins may have multiple alternate functions beyond basic digital I/O. This feature is known as I/O multiplexing. By configuring certain registers, the GPIO pins can be assigned alternative roles, such as supporting communication interfaces (e.g., UART, SPI, I2C) or peripheral control (e.g., PWM, timers).

**7. Importance of Basic I/O Configuration:** Basic I/O configuration is the foundation of embedded systems design, enabling the microcontroller to interface with the external world. It is vital for developing applications involving user inputs, sensor readings, and actuator control. Proper I/O configuration ensures accurate data acquisition, precise control, and seamless communication, all of which are critical in a wide range of embedded systems, from simple home automation devices to complex industrial control systems.

# CHAPTER 6

# SERIAL COMMUNICATION INTERFACES

**1. Introduction:** Serial communication interfaces are essential communication protocols used in embedded systems to transfer data between devices in a bit-by-bit or serial fashion. Unlike parallel communication, where multiple bits are transmitted simultaneously on separate lines, serial communication uses a single data line to send and receive data. Serial interfaces are widely employed in various applications, including interconnecting microcontrollers, sensors, communication modules, and external peripherals.

**2. Types of Serial Communication Interfaces:** There are several types of serial communication interfaces, each with its own advantages and applications:

Universal Asynchronous Receiver/Transmitter (UART): UART is a simple and widely used serial communication protocol that allows asynchronous data transmission between devices. It is commonly used for communication between a microcontroller and peripherals like GPS modules, Bluetooth modules, and other microcontrollers.

Serial Peripheral Interface (SPI): SPI is a synchronous serial communication protocol that facilitates high-speed data transfer between a master device and one or more slave devices. SPI is commonly used to communicate with devices like sensors, data converters, and display controllers.

Inter-Integrated Circuit (I2C): I2C is a multi-master, multi-slave serial communication protocol that enables communication between devices using a shared bus. It is often used for connecting various peripherals, such as sensors, EEPROMs, and real-time clocks, to a microcontroller.

Universal Serial Bus (USB): USB is a versatile and widely used serial communication standard for connecting various peripherals, storage devices, and other devices to computers and embedded systems.

Controller Area Network (CAN): CAN is a robust and reliable serial communication protocol commonly used in automotive and industrial applications for communication between electronic control units (ECUs) and sensors/actuators.

**3. Serial Communication Basics:** In serial communication, data is sent as a series of bits, one after another, over a single communication line. The data is usually framed with start and stop bits (in UART) or embedded with clock signals (in SPI and I2C) to synchronize the devices. Serial communication interfaces provide different data rates (baud rates) that define the speed of data transmission.

**5. Implementing Serial Communication**: To implement serial communication in embedded systems, developers need to configure the corresponding hardware modules (e.g., UART, SPI, I2C) of the microcontroller. This involves setting up the communication parameters, such as baud rate, data format, and clock frequency. Once configured, data can be transmitted and received through the serial communication interface by reading from and writing to the appropriate registers or buffers.

**6. Application Areas:** Serial communication interfaces are used in a wide range of applications, including:

Data acquisition and sensing systems

Communication between microcontrollers in distributed systems

Industrial automation and control systems

Interfacing with various peripheral devices, such as displays, sensors, and wireless modules

Automotive applications for ECU communication and diagnostics

Communication between microcontrollers and external memory devices.

**CHAPTER 7**

**PROGRAMMING STM32 USING IDE**

**1. Introduction:** Programming STM32 microcontrollers involves writing and compiling code to control the hardware and execute desired functionalities. To streamline the development process, developers often use Integrated Development Environments (IDEs) specifically designed for STM32 microcontrollers. IDEs provide a comprehensive toolset, including code editors, compilers, debuggers, and project management features, making the development and debugging of STM32 applications more efficient and productive.

**2. Choosing an IDE for STM32 Development:** There are several IDE options available for STM32 development, including:

STM32CubeIDE: An official IDE from STMicroelectronics designed specifically for STM32 microcontrollers. It integrates the STM32CubeMX graphical tool for hardware configuration and code generation.

Keil MDK (Microcontroller Development Kit): A widely used IDE by ARM that supports STM32 microcontrollers. It includes the ARM Compiler and various debugging options.

Eclipse with STM32 Plugins: Eclipse, a popular open-source IDE, can be extended with plugins for STM32 development, offering flexibility and customization.

**3. Development Workflow**: The typical STM32 development workflow using an IDE involves the following steps:

Project Creation: Create a new STM32 project in the IDE, selecting the target STM32 microcontroller and configuring project settings.

Hardware Configuration: Use STM32CubeMX (if integrated) to configure the microcontroller's peripherals, clock settings, and pin assignments visually. The generated configuration code will be included in the project.

Code Development: Write application code in the IDE's code editor, utilizing the STM32 HAL (Hardware Abstraction Layer) libraries or other libraries based on project requirements.

Code Compilation: Compile the code using the integrated compiler, which converts the source code into machine code.

Debugging and Testing: Debug the application using the IDE's debugger, set breakpoints, watch variables, and step through the code to identify and resolve issues.

Flashing and Execution: Flash the compiled code into the STM32 microcontroller's memory using the IDE, allowing the microcontroller to execute the application.

**4. Benefits of Using an IDE for STM32 Programming:**

Simplified Project Management: IDEs provide a structured environment for managing STM32 projects, making it easy to organize and maintain source code and resources.

Efficient Debugging: IDEs offer advanced debugging tools that help identify and fix issues in the code quickly.

Accelerated Development: With code completion and intelligent code suggestions, IDEs accelerate development by reducing manual coding effort.

Seamless Integration: IDEs seamlessly integrate with STM32 hardware and software tools, providing an all-in-one solution for the development process.

**CHAPTER 8**

**SERIAL DEBUG / JTAG**

**1. Introduction:** Serial Debugging and Joint Test Action Group (JTAG) are two common techniques used for debugging and testing embedded systems and microcontrollers. These methods facilitate developers in identifying and fixing software and hardware issues, enabling efficient development and testing of complex embedded systems.

**2. Serial Debugging:** Serial debugging is a technique used to monitor and analyze the program execution flow and data within the microcontroller. It involves sending debugging information over a serial communication interface (e.g., UART) to a host computer, where it is displayed and analyzed using a terminal emulator or a dedicated debugging software.

**Key Aspects of Serial Debugging:**

Print Statements: Developers can insert print statements in their code, which send debugging messages to the host computer. This helps in tracing program execution, checking variable values, and identifying the flow of control.

**Advantages of Serial Debugging**:

Ease of Use: Serial debugging is relatively easy to set up and use, requiring only a serial communication interface and a terminal emulator on the host computer.

Minimal Hardware Requirements: Most microcontrollers come with built-in UARTs, making it readily available for serial debugging.

Limitations of Serial Debugging:

Limited Data Transfer: The amount of data that can be transferred for debugging purposes is limited by the speed of the serial communication interface.

**3. Joint Test Action Group (JTAG):** JTAG is a standard for boundary scan testing and in-system programming of microcontrollers and other integrated circuits. It provides a set of pins on the microcontroller called the JTAG port, which allows external hardware tools (e.g., JTAG debuggers) to access and control the internal circuitry of the microcontroller.

**Key Aspects of JTAG:**

Boundary Scan Testing: JTAG boundary scan testing allows developers to test the interconnections and verify proper soldering of components on the circuit board.

**In-System Programming:** JTAG enables the programming of the microcontroller's non-volatile memory (e.g., flash) in-system, without the need to remove the chip from the circuit board.

**Advantages of JTAG:**

Full Control over Hardware: JTAG provides full access to the microcontroller's internal registers, memory, and other resources, allowing precise control during debugging and testing.

High-Speed Debugging: JTAG debuggers can operate at higher speeds, making it suitable for real-time debugging and time-critical applications.

**Limitations of JTAG:**

Hardware Complexity: JTAG requires additional hardware support on the microcontroller, which might increase the cost and complexity of the design.

Special Debugging Tools: Dedicated JTAG debuggers or programmers are required for debugging, which can be more expensive than basic serial debugging setups.

**4. Choosing Between Serial Debugging and JTAG:** The choice between serial debugging and JTAG depends on the complexity of the project, debugging requirements, and available hardware resources. For simple applications with basic debugging needs, serial debugging is often sufficient. However, for complex systems with real-time requirements and the need for full hardware control, JTAG provides more extensive debugging capabilities.

**CHAPTER 9**

**INTRODUCTION TO STM32F405 - DEVELOPMENT BOARD**

**STM32F405 Development Board Overview**

The STM32F405 Development Board is a versatile and feature-rich platform designed to evaluate and prototype applications based on the STM32F405RG microcontroller from STMicroelectronics. This development board provides a convenient and efficient way for developers to kickstart their projects, benefiting from the STM32F405's powerful ARM Cortex-M4 core, rich peripheral set, and extensive development ecosystem. Below is an overview of the STM32F405 Development Board:

Microcontroller: The heart of the development board is the STM32F405RG microcontroller, which is part of the STM32F4 series. It is based on the ARM Cortex-M4 core, operating at a clock speed of up to 168 MHz, and features hardware floating-point support for efficient mathematical processing.

Memory: The STM32F405RG microcontroller comes with a generous amount of Flash memory for program storage and SRAM for data storage. This enables the implementation of complex and sophisticated applications.

Onboard Peripherals:

GPIO Pins: The board is equipped with a set of General-Purpose Input/Output (GPIO) pins, allowing developers to interface with external components such as sensors, displays, and actuators.

UART, SPI, I2C Interfaces: These communication interfaces enable easy connectivity with other devices or modules, such as sensors, displays, and communication modules, for data exchange and control.

USB Port: The USB interface allows the board to be connected to a computer for programming and debugging or to interface with USB devices.

**Ethernet Port:** Some STM32F405 development boards feature an Ethernet port for networking capabilities, enabling applications in IoT and industrial automation.

**CAN (Controller Area Network):** CAN interface provides the ability to communicate with other devices using the CAN protocol, making it suitable for automotive and industrial applications.

**Debugging and Programming Support:** The STM32F405 Development Board typically features onboard debugging and programming capabilities, utilizing an ST-Link or J-Link debugger. This allows developers to flash their programs onto the microcontroller and efficiently debug their applications in real-time.

**Power Supply and Clock Circuitry:** The development board comes with a power supply circuit that provides stable voltage levels to the microcontroller and peripherals. Additionally, it includes a crystal oscillator or other clock sources to provide accurate clock signals to the STM32F405 microcontroller.

**Additional Components:**

**LEDs and Push-buttons:** The board is equipped with LEDs that can be controlled by the microcontroller, allowing developers to implement simple visual feedback. Push-buttons are provided for user input.

**User-configurable Jumpers:** Jumper settings can be adjusted to configure various aspects of the board, such as power supply options or pin connections.

**Development Environment Support:** The STM32F405 Development Board is compatible with various Integrated Development Environments (IDEs), such as STM32CubeIDE and Keil MDK, providing a seamless development experience with access to HAL (Hardware Abstraction Layer) libraries, middleware, and other development tools.

**CHAPTER 10**

**IOC CONFIGURATION USING CUBE MX**

**1. Introduction to I/O Controller (IOC) and Cube MX:** The I/O Controller is responsible for managing the microcontroller's input and output operations, including the configuration of pins, peripherals, and communication interfaces. Cube MX is a software tool developed by STMicroelectronics that simplifies the process of configuring the IOC on STM32 microcontrollers. It provides a user-friendly interface to define and customize the microcontroller's I/Os and peripheral settings, enhancing the overall development experience.

**2. Features and Benefits of Cube MX:** Cube MX offers several features that streamline the IOC configuration process. It allows developers to visualize the pinout of the microcontroller, configure peripheral settings, and manage clock and power configurations effortlessly. The tool provides automatic checks for pin conflicts and compatibility, ensuring that the configurations are valid.

**3. Getting Started with Cube MX:** This section guides developers through the process of installing and launching STM32CubeMX. It covers selecting the target STM32 microcontroller family and device, as well as setting up the necessary project parameters. By following the steps provided, developers can quickly get started with their specific STM32 microcontroller.

**4. Creating a New Project in Cube MX**: Once Cube MX is installed, this section explains the steps to create a new project. Developers will learn how to choose the desired microcontroller and configure basic project settings such as the system clock frequency and peripherals required for the application.

**5. I/O Pin Configuration:** Cube MX simplifies I/O pin configuration by allowing developers to assign functions such as GPIO, alternate functions, and analog inputs to specific pins through an intuitive graphical interface. This section elaborates on the pin configuration process, ensuring seamless interfacing of external components and peripherals.

**6. Peripheral Configuration:** Developers can easily configure various peripherals supported by the STM32 microcontroller using Cube MX. This section demonstrates how to enable and configure peripherals such as UART, SPI, I2C, Timers, ADC, and PWM. Additionally, it illustrates how to connect these peripherals to the appropriate I/O pins.

**7. Clock Configuration:** Proper clock configuration is essential for the reliable operation of the microcontroller and its peripherals. Cube MX provides a simple way to set up clock sources, frequencies, and distribution. This section explains the clock configuration process and its impact on the application's performance.

**8. Interrupt Configuration:** In applications requiring real-time responsiveness, interrupt configuration is critical. Cube MX simplifies the setup of interrupts, including assigning priorities and handling interrupt service routines (ISRs). Developers will learn how to enable and configure interrupts effectively.

**9. GPIO Configuration for External Peripherals:** Interfacing external components and sensors often involves using GPIO pins. Cube MX offers easy GPIO configuration options. This section provides practical examples of how to configure GPIO pins for specific applications.

**10. Power Management Configuration:** Cube MX supports power management configuration, allowing developers to optimize power consumption in their applications. This section explores various low-power modes and the steps to configure them, enhancing the energy efficiency of the final product.

**11. Generating Configuration Code:** After completing the configuration in Cube MX, developers can generate initialization code based on their settings. This section demonstrates the process of generating code and explains the structure of the generated code, providing insights into code integration.

**CHAPTER 11**

**HAL LIBRARIES**

**1. Introduction:** The Hardware Abstraction Layer (HAL) is an essential software component in embedded systems development. HAL libraries are sets of functions and drivers provided by microcontroller manufacturers to abstract hardware functionalities and provide a standardized interface for accessing peripherals and core features. The primary goal of HAL libraries is to simplify application development by offering a uniform API that remains consistent across various microcontroller families, making it easier for developers to port code and work with different devices seamlessly.

**2. Purpose and Benefits**: The HAL libraries serve as a bridge between low-level hardware registers and the application layer, offering a higher-level, more intuitive API for developers. Some key purposes and benefits of HAL libraries include:

Portability: HAL libraries enable application code to be written in a hardware-independent manner, making it easier to switch between different microcontroller families or vendors.

Code Reusability: By using the same HAL API across projects, developers can reuse code modules for similar functionalities, saving time and effort.

Simplified Development: HAL libraries abstract hardware complexities, allowing developers to focus on application logic rather than dealing with low-level hardware details.

Scalability: As HAL libraries provide a standardized API, applications can be easily scaled to support different microcontroller models within the same family.

Easy Migration: When upgrading to newer microcontroller versions, the HAL API remains consistent, reducing the effort required for code migration.

**3. HAL Library Components:** HAL libraries typically include functions and drivers for various aspects of microcontroller operation, including:

GPIO (General Purpose Input/Output): Provides functions to configure and control GPIO pins for digital input and output operations.

ADC (Analog-to-Digital Converter): Offers functions for configuring and reading analog input signals.

Timers: Provides APIs for configuring and utilizing timers for timing operations, PWM generation, and more.

UART (Universal Asynchronous Receiver/Transmitter): Includes functions for serial communication using UART.

SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit): Provide APIs for interfacing with SPI and I2C communication protocols.

DMA (Direct Memory Access): Facilitates efficient data transfers between peripherals and memory without CPU intervention.

**4. Working with HAL Libraries:** Developers can leverage HAL libraries by including the appropriate header files and linking the corresponding libraries with their application code. By calling the HAL functions provided by the libraries, developers can easily configure and control the microcontroller's peripherals and features.

**5. Integration with IDEs:** HAL libraries are typically integrated into popular Integrated Development Environments (IDEs), such as STM32CubeIDE or Keil MDK. IDE integration simplifies project setup and provides code generation tools to configure HAL libraries for specific microcontroller models.

**CHAPTER 12**

**BARE METAL CODING**

**1. Introduction:** Bare metal coding refers to the practice of programming microcontrollers or embedded systems directly at the hardware level, without using any operating system or middleware. In this approach, developers have complete control over the hardware resources and can directly access and configure the microcontroller's registers to implement specific functionalities. Bare metal coding is often employed in resource-constrained systems and time-critical applications where low-level control and minimal overhead are essential.

**2. Advantages of Bare Metal Coding**: Bare metal coding offers several advantages for embedded systems development:

Minimal Overhead: Without an operating system or middleware layer, bare metal code has minimal processing and memory overhead, making it ideal for resource-constrained devices.

Maximum Performance: Direct hardware access allows developers to fine-tune the system for maximum performance, as there is no overhead associated with an operating system.

Reduced Boot Time: Bare metal code typically has faster boot times compared to systems running complex operating systems.

Predictable Timing and Real-Time Operation: Bare metal coding enables precise control over timing and real-time operations, making it suitable for critical applications.

**3. Challenges and Considerations:** While bare metal coding offers significant benefits, it also comes with challenges:

Hardware Dependencies: Bare metal code is tightly coupled with the hardware, making it less portable across different microcontroller families or architectures.

Lack of Abstraction: Developers need to deal with low-level hardware registers and memory management, which can be error-prone and time-consuming.

Limited Reusability: The lack of a standardized API or middleware makes code reuse more challenging across different projects.

**4. Key Components of Bare Metal Code:** Bare metal code typically consists of the following key components:

Startup Code: This initializes the microcontroller's hardware, sets the stack pointer, and configures the clock system before executing the main application code.

Peripheral Drivers: Low-level drivers to interact with various microcontroller peripherals, such as GPIO, UART, SPI, ADC, etc.

Interrupt Service Routines (ISRs): Handlers for hardware interrupts to respond to events or data from external sources.

Application Logic: The main application code that implements the desired functionality of the embedded system.

**5. Development Process and Tools:** Developers working with bare metal coding use Integrated Development Environments (IDEs) with cross-compilers and debuggers tailored to the target microcontroller. The development process involves writing the low-level code, compiling it to produce a binary file, and flashing it onto the microcontroller for execution.

**6. Real-World Applications:** Bare metal coding is commonly employed in various applications, including:

IoT Devices: Resource-constrained IoT devices often use bare metal coding to achieve power efficiency and real-time responsiveness.

Automotive Systems: Critical automotive systems, such as engine control units (ECUs) and airbag systems, rely on bare metal coding for predictability and safety.

**CHAPTER 13**

**INTERFACING SENSORS WITH STM32F405**

**1. Introduction:** Interfacing sensors with microcontrollers like STM32F405 is a fundamental requirement for building smart and data-driven embedded systems. Sensors play a crucial role in collecting data from the physical world, allowing microcontrollers to make informed decisions and execute specific actions based on the acquired data. This overview provides insights into the process of interfacing sensors with STM32F405, exploring the different types of sensors, communication protocols, and essential implementation considerations.

**2. Types of Sensors:** Compatible with STM32F405: STM32F405 offers a wide range of communication interfaces and GPIO pins, making it compatible with various types of sensors. Some common types of sensors compatible with STM32F405 include:

Temperature Sensors: LM35, DHT11, DHT22

Humidity Sensors: DHT11, DHT22, HDC1080

Pressure Sensors: BMP180, BMP280, BME280

Motion Sensors: Accelerometers, Gyroscopes, PIR Motion Sensors

Proximity Sensors: Ultrasonic Distance Sensors, Infrared Proximity Sensors

Light Sensors: LDR (Light Dependent Resistor), Photodiodes

Each sensor type has specific applications and characteristics, making them suitable for diverse use cases.

**3. Communication Interfaces for Sensor Interfacing:** STM32F405 microcontroller is equipped with versatile communication interfaces, facilitating seamless sensor integration. Some of the common communication interfaces used for sensor interfacing are:

UART (Universal Asynchronous Receiver/Transmitter)

SPI (Serial Peripheral Interface)

I2C (Inter-Integrated Circuit)

GPIO (General Purpose Input/Output)

The choice of communication interface depends on factors such as sensor requirements, data transfer speed, and the number of available communication channels.

**4. Interfacing Analog Sensors with ADC**: Many sensors, such as temperature and humidity sensors, provide analog outputs. To interface these analog sensors with STM32F405, the built-in ADC (Analog-to-Digital Converter) is used to convert analog signals into digital data that the microcontroller can process. The report delves into the process of configuring the ADC and reading analog sensor data.

**5. Interfacing Digital Sensors via Serial Interfaces:** Digital sensors with UART, SPI, or I2C interfaces can be directly connected to STM32F405 using the corresponding communication protocol. The report discusses the steps involved in configuring and communicating with digital sensors using these interfaces.

**6. Sensor Data Processing and Calibration:** Sensor data may require processing and calibration to ensure accuracy and reliability. The report covers techniques for filtering, scaling, and calibrating sensor data to obtain meaningful and consistent measurements.

**7. Real-World Applications:** The overview concludes with real-world examples of interfacing sensors with STM32F405 in various applications:

Home Automation: Integrating motion and proximity sensors for smart lighting and security systems.

Robotics: Using accelerometers and gyroscope sensors for motion control and orientation detection.

**CHAPTER 14**

**PROTOTYPE DEVELOPMENT**

**1. Introduction:** Prototype development is a critical phase in the product development lifecycle, where a preliminary version of the product is created to test and validate ideas before moving into full-scale production. It involves building a tangible representation of the product concept to gain insights, identify potential issues, and gather feedback from stakeholders.

**2. Types of Prototypes:** There are various types of prototypes used in different stages of the development process:

Proof-of-Concept Prototypes: These prototypes focus on demonstrating the feasibility of a new idea or technology. They often lack full functionality but help in assessing the viability of the concept.

Functional Prototypes: Functional prototypes emulate the core features and functionalities of the final product. They allow testing and validation of the product's performance and user interaction.

Visual Prototypes: Visual prototypes focus on the product's aesthetics, appearance, and user interface. They help stakeholders visualize the final product's look and feel.

**3. Prototyping Methods and Techniques:** Several prototyping methods and techniques are available, each catering to specific project requirements:

Rapid Prototyping: Rapid prototyping uses 3D printing or additive manufacturing techniques to quickly produce physical models of the product. It is ideal for fast iterations and visualizing complex designs.

Breadboarding: Breadboarding involves building electronic prototypes using electronic components and a breadboard. It is commonly used for testing and developing electronic circuits.

Simulation: Simulation-based prototyping employs computer simulations to model and analyze product behavior and performance. It is useful for complex systems and environments that are difficult to replicate physically.

**4. Benefits and Advantages of Prototyping:** The benefits of prototype development are numerous and contribute significantly to the overall success of the product development process:

Faster Development Cycles: Prototyping allows quick iterations and feedback, accelerating the development timeline.

Reduced Development Costs: Identifying design flaws and issues early in the process reduces costly rework during the later stages of development.

Improved Product Quality: Prototyping facilitates user testing and validation, leading to a more refined and user-centric final product.

**5. Challenges and Considerations**: While prototyping offers numerous advantages, it also comes with its challenges:

Time and Resource Constraints: Developing prototypes can be time-consuming and may require additional resources.

Balancing Perfection and Speed: Striking the right balance between achieving a functional prototype quickly and ensuring its accuracy can be challenging.

Scaling to Mass Production: Prototypes may not always translate seamlessly into mass production, requiring further refinement for manufacturability.

**6. Iterative Prototyping and User Feedback:** The iterative nature of prototyping encourages continuous improvement through multiple design iterations. Obtaining user feedback throughout the prototyping process helps in refining the product's design and functionality.

CHAPTER 13

**CHAPTER 15**

**USE CASES IN AUTOMOTIVE DOMAIN WITH STM32F405 - DEVELOPMENT BOARD**

**1. Introduction:** The STM32F405 microcontroller development board offers a versatile platform for implementing various automotive applications. This overview explores several key automotive applications that can be developed using the STM32F405 development board, providing insights into their functionalities.

**2. Use Cases in the Automotive Domain:** The STM32F405 development board can be utilized in the following automotive use cases:

**3. Engine Control Unit (ECU):**

Purpose: The Engine Control Unit is a critical component of modern vehicles, responsible for monitoring and controlling the engine's operations for optimal performance, fuel efficiency, and emissions control.

Implementation: The STM32F405 can serve as the ECU, interfacing with various sensors to measure engine parameters (e.g., RPM, temperature, airflow) and actuating engine components (e.g., fuel injectors, ignition coils) to maintain the engine's operation within predefined limits.

**4. Body Control Module (BCM):**

Purpose: The Body Control Module manages and controls various electrical and electronic functions within the vehicle's body, including lighting, power windows, door locks, and more.

Implementation: The STM32F405 can function as the BCM, handling inputs from switches and sensors and driving actuators to control the vehicle's body systems.

**5. Anti-Lock Braking System (ABS):**

Purpose: The ABS prevents wheel lockup during braking, allowing the driver to maintain steering control and reducing the stopping distance.

Implementation: The STM32F405 can act as the ABS controller, processing wheel speed sensor data and modulating brake pressure to prevent wheel lockup.

**6. Airbag Control Unit (ACU):**

Purpose: The Airbag Control Unit monitors various vehicle parameters and triggers airbag deployment in the event of a collision to protect the occupants.

Implementation: The STM32F405 can serve as the ACU, processing sensor data (e.g., impact sensors, seat occupancy) and deploying airbags when required.

**7. Tire Pressure Monitoring System (TPMS):**

Purpose: TPMS monitors the tire pressure and alerts the driver if the tire pressure drops below a safe level, improving safety and fuel efficiency.

Implementation: The STM32F405 can function as the TPMS controller, processing data from tire pressure sensors and providing warnings to the driver.

**8. CAN Communication Gateway:**

Purpose: Modern vehicles often have multiple ECUs that need to communicate with each other. A CAN communication gateway facilitates communication between different CAN buses within the vehicle.

Implementation: The STM32F405 can act as the CAN communication gateway, managing data exchange between different ECUs and ensuring seamless communication.

**9. Infotainment System:**

Purpose: The infotainment system provides entertainment, navigation, and connectivity features to the vehicle's occupants.

Implementation: The STM32F405 can drive the infotainment system's display, process user inputs, and interface with various communication protocols for media playback, navigation, and connectivity.

**CHAPTER 16**

**OBSTACLE DETECTION**

**1. Introduction:** Obstacle detection is a vital technology used in various applications to identify and avoid obstacles in the path of a moving object, such as vehicles, drones, robots, and autonomous systems. The primary goal of obstacle detection systems is to enhance safety, prevent collisions, and enable autonomous navigation by providing real-time information about the surroundings.

**2. Types of Obstacle Detection Technologies:** There are several technologies used for obstacle detection, each with its unique advantages and limitations:

Ultrasonic Sensors: Ultrasonic sensors emit high-frequency sound waves and measure the time taken for the sound waves to bounce back after hitting an obstacle. They are commonly used for short-range obstacle detection in parking assistance systems.

Infrared Sensors: Infrared sensors emit infrared radiation and detect reflections from nearby objects. They are suitable for close-range obstacle detection and commonly used in proximity sensors.

Lidar (Light Detection and Ranging): Lidar systems use lasers to measure the distance and create precise 3D maps of the surrounding environment. They are widely used in autonomous vehicles for obstacle detection and mapping.

Radar (Radio Detection and Ranging): Radar systems use radio waves to detect objects and their relative speed and distance. They are commonly used in automotive collision avoidance systems and aircraft.

Camera-based Vision Systems: Camera-based systems use computer vision algorithms to analyze images and identify obstacles in the field of view. They are employed in advanced driver assistance systems (ADAS) and robotics.

Time-of-Flight (ToF) Sensors: ToF sensors measure the time taken for light to travel to an object and back, enabling distance measurements. They are suitable for both short and long-range obstacle detection.

**3. Obstacle Detection Algorithms:** Obstacle detection systems rely on various algorithms to process data from sensors and determine the presence and position of obstacles. Common algorithms include:

Thresholding and Clustering: Used in camera-based systems to segment objects from the background and identify clusters representing obstacles.

Hough Transform: Used for line and circle detection in lidar and radar data, often applied in mapping and obstacle identification.

Kalman Filters: Used for sensor fusion, combining data from multiple sensors to improve accuracy and reliability.

Deep Learning and Neural Networks: Advanced obstacle detection systems employ deep learning algorithms to recognize objects and obstacles in real-time images and video streams.

**4. Applications:** Obstacle detection finds application in various industries and domains, including:

Automotive: In advanced driver assistance systems (ADAS) to enable features like automatic emergency braking, collision avoidance, and adaptive cruise control.

Robotics: In autonomous robots for obstacle avoidance and navigation in complex environments.

Industrial Automation: In warehouse robots and material handling equipment to avoid obstacles and ensure safe operation.

**5. Challenges and Considerations:** Obstacle detection systems face challenges such as varying environmental conditions, sensor accuracy, real-time processing requirements, and occlusions. Ensuring robust performance and safety is essential in critical applications like autonomous vehicles.

**CHAPTER 17**

**BLIND SPOT DETECTION**

**1. Introduction**: Blind Spot Detection (BSD) is an advanced driver assistance system (ADAS) designed to enhance vehicle safety by monitoring the areas around the vehicle that are not easily visible to the driver. The system uses various sensors and technologies to detect objects or vehicles in the vehicle's blind spots, providing visual or audible warnings to the driver to avoid potential collisions during lane changes.

**2. How Blind Spot Detection Works:** Blind Spot Detection systems typically utilize a combination of sensors and technologies to detect objects in the blind spot areas:

Radar Sensors: Radar sensors are commonly used in BSD systems. They emit radio waves and measure the time taken for the waves to bounce back after hitting an object. By analyzing the returned signals, the system can detect the presence and relative distance of vehicles in the blind spot.

Ultrasonic Sensors: Some BSD systems employ ultrasonic sensors to detect nearby objects. These sensors work on the principle of sending and receiving ultrasonic waves, which can detect obstacles in close proximity to the vehicle, such as vehicles in adjacent lanes.

Camera-Based Systems: Advanced BSD systems may use cameras and computer vision algorithms to identify vehicles or objects in the blind spot region. These cameras are typically mounted on the side mirrors or near the rear of the vehicle.

**3. Warning Mechanisms**: When the Blind Spot Detection system detects an object in the blind spot, it provides warnings to the driver to alert them of the potential danger. The warning mechanisms can include:

Visual Alerts: Visual warnings are typically displayed on the side mirrors or inside the vehicle, near the driver's line of sight. This can be in the form of a warning light or an icon indicating the presence of a vehicle in the blind spot.

Audible Alerts: Audible warnings, such as a beep or chime, may accompany the visual alerts to draw the driver's attention to the presence of an object in the blind spot.

**4. Benefits of Blind Spot Detection:** Blind Spot Detection offers several key benefits that enhance overall driving safety and reduce the risk of collisions:

Improved Visibility: BSD systems provide drivers with additional visibility of vehicles or objects in the blind spot areas, supplementing the driver's field of view.

Collision Avoidance: By warning the driver of potential collisions during lane changes, BSD helps prevent accidents and reduces the likelihood of side-impact collisions.

Enhanced Lane Changing Safety: BSD promotes safer lane-changing maneuvers, as drivers are alerted to potential hazards that they may not have seen otherwise.

Reduced Driver Stress: The system's alerts alleviate the stress of checking blind spots constantly, especially in heavy traffic or complex road conditions.

**5. Limitations and Considerations:** While Blind Spot Detection is a valuable safety feature, it does have certain limitations to be aware of:

Sensor Range: The detection range of sensors may vary, and objects too close to the vehicle or beyond the sensor's range may not be detected.

False Positives: Adverse weather conditions, road debris, or large roadside structures may sometimes trigger false alerts.

**CHAPTER 18**

**CABIN PRESSURE MONITORING**

**1. Introduction:** Cabin Pressure Monitoring is a critical safety feature employed in various modes of transportation, including aircraft, submarines, and pressurized cabins in some vehicles. The system's primary objective is to monitor and regulate the air pressure inside the cabin to ensure a safe and comfortable environment for occupants. By maintaining an optimal cabin pressure, passengers and crew can avoid physiological discomfort and potential health risks associated with changes in atmospheric pressure at high altitudes or in confined spaces.

**2. Importance of Cabin Pressure Regulation:** At high altitudes, the atmospheric pressure decreases significantly, leading to reduced oxygen levels. This can result in various health issues for passengers and crew, including hypoxia, altitude sickness, and discomfort due to pressure changes. To counteract these effects, cabins in pressurized vehicles are maintained at controlled pressure levels, providing a breathable and safe environment similar to that experienced at lower altitudes.

**3. How Cabin Pressure Monitoring Works:** Cabin Pressure Monitoring systems utilize various sensors, control mechanisms, and safety measures to maintain the desired pressure levels inside the cabin:

Pressure Sensors: These sensors continuously measure the air pressure within the cabin. The data from these sensors is fed into the cabin pressure monitoring system.

Control System: The cabin pressure monitoring system is connected to a control unit that regulates the cabin pressure by controlling the flow of air in and out of the cabin.

Outflow Valves: The system uses outflow valves to regulate the airflow, enabling it to adjust the cabin pressure during ascent and descent.

**4. Automatic Pressure Regulation**: In modern aircraft, the cabin pressure monitoring system is often integrated with the aircraft's avionics and flight management systems. The control system automatically adjusts the outflow valves and cabin air supply to maintain the cabin pressure at safe and comfortable levels as the aircraft ascends or descends. This process is known as "cabin pressurization."

**5. Passenger Comfort and Well-Being:** Maintaining the cabin pressure at appropriate levels is crucial for passenger comfort and well-being. Proper cabin pressure ensures that passengers do not experience discomfort or physiological issues, such as ear pain or sinus pressure, commonly associated with rapid changes in pressure.

**6. Emergency Scenarios:** In case of emergencies, such as rapid decompression due to a breach in the aircraft's hull, the cabin pressure monitoring system plays a critical role. It rapidly equalizes the pressure to prevent harm to passengers and allows the crew to respond effectively to the situation.

**7. Monitoring and Indicators:** Cabin Pressure Monitoring systems provide continuous monitoring of cabin pressure, and pilots or crew members are alerted through visual and auditory indicators if pressure levels deviate from the desired range. This allows for prompt corrective action to ensure passenger safety.

**8. Redundancy and Safety Measures:** Given the critical nature of cabin pressure regulation, aircraft typically incorporate redundant systems to ensure fail-safe operation. Redundant pressure sensors, control units, and actuators contribute to enhanced safety and reliability.

**CHAPTER 19**

**AUTOMATIC CLIMATE CONTROL SYSTEM**

**1. Introduction**: An Automatic Climate Control System is an advanced feature in modern vehicles and buildings that automatically regulates the interior temperature, humidity, and airflow to maintain a comfortable and consistent environment for occupants. The system is designed to provide convenience, comfort, and energy efficiency by adjusting various climate control parameters based on external and internal conditions.

**2. How Automatic Climate Control Works:** The Automatic Climate Control System uses a combination of sensors, actuators, and control algorithms to manage the interior climate. The key components of the system include:

Temperature Sensors: These sensors measure the interior and exterior temperatures to assess the current climate conditions.

Humidity Sensors: Humidity sensors monitor the level of moisture in the air, which affects occupant comfort.

Occupancy Sensors: Some advanced systems incorporate occupancy sensors to detect the presence of occupants and adjust climate settings accordingly.

Sunlight Sensors: Sunlight sensors determine the amount of incoming sunlight, which affects the temperature inside the vehicle or building.

Climate Control Unit: The control unit processes data from various sensors and actuates the heating, ventilation, and air conditioning (HVAC) system components to achieve the desired climate conditions.

Actuators: Actuators control the opening and closing of air vents, modulate the heating and cooling systems, and regulate the fan speed.

**3. Customization and User Interface:** Automatic Climate Control Systems typically offer customizable settings, allowing occupants to set their preferred temperature, fan speed, and airflow distribution. Many systems also include user-friendly interfaces such as touchscreens or rotary knobs to adjust the settings easily.

**4. Dual-Zone and Multi-Zone Climate Control:** In some vehicles or buildings, the Automatic Climate Control System can be equipped with dual-zone or multi-zone capabilities. This feature allows different areas of the vehicle or building to have separate temperature and airflow settings to cater to individual preferences.

**5. Adaptive Climate Control:** Advanced Automatic Climate Control Systems may incorporate adaptive features that learn from user behavior and adjust climate settings accordingly. For example, the system may remember preferred temperature settings at specific times of the day and adapt to those patterns.

**6. Energy Efficiency and Environmental Benefits:** By automatically adjusting climate control parameters based on actual conditions, the Automatic Climate Control System can optimize energy usage and reduce fuel consumption in vehicles or electricity consumption in buildings. This contributes to energy efficiency and a reduced carbon footprint.

**7. Automotive Applications:** In vehicles, Automatic Climate Control is commonly found in modern cars, trucks, and luxury vehicles. The system ensures a comfortable interior environment for passengers during all weather conditions, reducing the need for manual adjustments and distractions for the driver.

8. Building Applications: In buildings, Automatic Climate Control Systems are used in commercial and residential spaces to maintain a comfortable indoor environment. They can be integrated with building management systems to optimize energy usage and enhance occupant comfort.

**CHAPTER 20**

**ELECTRONIC STABILITY CONTROL**

**1. Introduction:** Electronic Stability Control (ESC), also known as Electronic Stability Program (ESP) or Dynamic Stability Control (DSC), is an advanced automotive safety system designed to enhance vehicle stability and prevent loss of control during various driving maneuvers. ESC uses a combination of sensors, actuators, and control algorithms to assist the driver in maintaining control of the vehicle, especially in challenging road conditions or emergency situations.

**2. Working:** ESC continuously monitors various parameters related to vehicle motion, such as steering angle, wheel speed, lateral acceleration, and yaw rate. The key components and functionalities of ESC include:

Sensors: ESC relies on multiple sensors, including wheel speed sensors, steering angle sensor, gyroscope, and accelerometers, to gather real-time data about the vehicle's dynamics.

Electronic Control Unit (ECU): The ECU is the brain of the ESC system. It processes the sensor data and compares the actual vehicle behavior with the intended behavior based on driver inputs.

Actuators: ESC uses actuators, such as individual wheel brakes and engine throttle control, to apply selective braking or adjust engine torque to stabilize the vehicle.

**3. Preventing Understeer and Oversteer:** ESC is primarily designed to address two common types of instability: understeer and oversteer.

Understeer: Understeer occurs when the front tires lose traction, causing the vehicle to push wide in a turn. ESC intervenes by selectively applying the brakes to the appropriate wheels to help the vehicle turn more effectively.

Oversteer: Oversteer happens when the rear tires lose traction, causing the rear of the vehicle to slide out. ESC counteracts this by applying the brakes to specific wheels and reducing engine torque to regain control.

**4. Slip Control and Traction Assistance**: ESC also plays a vital role in managing wheel slip during acceleration on slippery surfaces. It can detect wheel spin and apply brake force to the spinning wheel, redistributing power to the wheels with better traction, effectively enhancing overall vehicle stability and traction.

**5. Rollover Prevention:** In some advanced ESC systems, rollover prevention features are integrated. These systems use additional sensors to monitor vehicle roll angle and lateral acceleration to predict and prevent potential rollover situations.

**6. Benefits of Electronic Stability Control:**

Enhanced Safety: ESC helps prevent loss of control and reduces the risk of skidding or sliding, significantly improving vehicle safety and reducing the likelihood of accidents.

Improved Handling: By assisting the driver in maintaining control during maneuvers, ESC enhances vehicle stability and handling, especially in challenging road conditions.

All-Weather Performance: ESC is particularly effective in adverse weather conditions, such as rain, snow, or ice, where maintaining control is critical.

**7. Regulatory Requirements**: Electronic Stability Control has become a mandatory safety feature in many countries due to its proven effectiveness in reducing accidents and enhancing vehicle stability. In some regions, ESC is a requirement for new vehicles to meet safety standards.

**CHAPTER 21**

**INTRODUCTION TO CAN PROTOCOL**

1. Introduction: The Controller Area Network (CAN) protocol is a widely used communication standard in the automotive and industrial sectors for transmitting data between electronic control units (ECUs). It was originally developed by Robert Bosch GmbH in the early 1980s to address the increasing complexity of wiring harnesses and the need for reliable and efficient communication between various components in vehicles. Since then, CAN has become a fundamental communication protocol in modern vehicles, facilitating real-time data exchange and control over a single bus.

**2. Key Characteristics of CAN:**

Serial Communication: CAN is a serial communication protocol, which means data is transmitted bit by bit over a single twisted-pair cable (CAN bus) shared by all connected ECUs.

Deterministic and Real-Time: CAN is deterministic, meaning messages are prioritized, and the system ensures real-time communication with predictable latencies.

Multimaster Architecture: CAN uses a multimaster architecture, allowing any ECU on the bus to initiate a message transmission.

Error Detection and Correction: CAN implements error detection and error correction mechanisms, ensuring the integrity and reliability of transmitted data.

Low Overhead: CAN has a relatively low overhead compared to other protocols, making it efficient and suitable for high-speed communication.

**3. CAN Message Format:** CAN messages consist of an identifier (ID), data length code (DLC), data bytes, and other fields that determine the message's priority and nature. Two message formats are commonly used in CAN:

Standard Frame Format (SFF): Utilizes an 11-bit identifier, suitable for applications with a limited number of nodes.

Extended Frame Format (EFF): Employs a 29-bit identifier, allowing a larger address space and accommodating more nodes in the network.

**4. Arbitration and Prioritization:** CAN uses a non-destructive bitwise arbitration mechanism to resolve conflicts when multiple ECUs attempt to transmit data simultaneously. Messages with lower IDs have higher priority and gain bus access before higher-priority messages.

**5. CAN Bit Timing and Speed:** CAN bit timing is crucial for reliable communication. The bit timing parameters, such as the bit rate, synchronization, and sampling points, need to be precisely set to ensure proper data reception. Common bit rates for CAN communication include 125 kbps, 250 kbps, 500 kbps, and 1 Mbps.

**6. Applications of CAN Protocol**: CAN protocol finds extensive use in various applications, including:

Industrial Automation: In industrial settings, CAN is employed for communication between programmable logic controllers (PLCs), sensors, and actuators to control manufacturing processes and machinery.

Heavy Machinery: CAN is used in construction and agricultural machinery to manage engine control, hydraulics, and other vehicle functions.

**7. CAN Versions and Variants:** Over the years, several versions and variants of the CAN protocol have been developed, including:

CAN 2.0A and CAN 2.0B: Two primary versions, differing in the length of the identifier and error handling capabilities.

CAN FD (Flexible Data Rate): An enhanced version of CAN that offers higher data rates and increased data payload size, enabling faster data transfer.

**CHAPTER 22**

**ECU TO ECU COMMUNICATION USING CAN PROTOCOL**

**1. Introduction:** ECU to ECU communication using the Controller Area Network (CAN) protocol is a fundamental aspect of modern vehicle and industrial control systems. Electronic Control Units (ECUs) are microcontrollers or embedded systems responsible for specific functions, such as engine control, transmission control, ABS, airbags, and more. CAN protocol facilitates seamless and efficient data exchange between these ECUs, enabling coordinated operation and real-time control of various vehicle or industrial functions.

**2. Importance:** In complex systems like modern vehicles or industrial machinery, multiple ECUs are interconnected to work collaboratively and share vital information. ECU to ECU communication enables different control units to exchange data, instructions, and feedback, allowing them to cooperate effectively and perform their respective tasks in harmony.

**3. CAN Protocol for ECU Communication**: CAN is well-suited for ECU to ECU communication due to its key characteristics:

Reliability: CAN's error detection and correction mechanisms ensure data integrity, making it reliable for critical applications.

Deterministic and Real-Time Communication: CAN provides predictable latencies, ensuring that messages are transmitted and received in a timely manner, crucial for real-time control in vehicles and industrial systems.

Prioritization and Arbitration: CAN's prioritization scheme ensures that messages with higher priority gain bus access first, allowing time-critical data to take precedence.

**4. ECU Network Architecture:** In a typical ECU network, multiple ECUs are interconnected through a CAN bus. Each ECU may have specific functionalities, and they communicate with each other based on the requirements of the system. For example, in a modern vehicle, the Engine Control Unit (ECU), Transmission Control Unit (TCU), Anti-lock Braking System (ABS) ECU, and Airbag Control Unit (ACU) may be interconnected through the CAN bus to exchange data for coordinated operation.

**5. Data Exchange between ECUs:** ECU to ECU communication involves the exchange of various types of data, such as:

Control Commands: ECUs send control commands to instruct other ECUs to perform specific actions or change their operating parameters.

Sensor Data: ECUs share data from various sensors, such as speed sensors, temperature sensors, or pressure sensors, to inform other ECUs about the current operating conditions.

Status and Feedback: ECUs provide feedback and status updates to indicate the completion of tasks, report errors, or request additional information.

**6. Benefits:**

System Integration: CAN enables seamless integration of various ECUs, promoting effective collaboration and operation of the entire system.

Reduced Wiring Complexity: CAN significantly reduces wiring complexity, as multiple ECUs can communicate over a single bus, simplifying the overall design.

Modularity and Flexibility: By using CAN, manufacturers can add or update functionalities by integrating new ECUs into the existing network without major hardware modifications.

Real-Time Control: CAN's real-time capabilities allow for precise control and coordination of critical functions in vehicles and industrial systems.

**7. Security Considerations**: With the increasing reliance on connected systems, ensuring the security of ECU to ECU communication is essential to prevent unauthorized access and potential cyber threats. Implementing secure communication protocols and encryption mechanisms is crucial to safeguard data integrity and system operation.

**CHAPTER 23**

**VEHICLE SPEED DETECTION**

**1. Introduction:** Vehicle Speed Detection is a critical aspect of modern transportation systems, enabling the measurement and monitoring of a vehicle's speed for various purposes. Accurate speed detection is essential for ensuring safety, compliance with traffic regulations, optimizing fuel efficiency, and implementing advanced driver assistance systems (ADAS) in vehicles.

**2. Methods of Vehicle Speed Detection:** Several methods are employed to detect the speed of a vehicle, each with its own advantages and applications:

Wheel Speed Sensors: Wheel speed sensors are commonly used in modern vehicles equipped with anti-lock braking systems (ABS). These sensors measure the rotational speed of individual wheels, and the vehicle's speed can be estimated based on the average of all four wheel speeds.

GPS Speed Measurement: Global Positioning System (GPS) technology can determine a vehicle's speed by tracking its position over time. GPS speed detection is commonly used in navigation systems and provides accurate speed readings based on satellite data.

Radar Technology: Doppler radar is employed in traffic speed enforcement systems and certain ADAS applications. Radar speed detectors emit radio waves that bounce off the moving vehicle, and the frequency shift of the returned waves is used to calculate the vehicle's speed.

Laser Technology: Laser speed guns are handheld devices used by law enforcement to measure the speed of vehicles. These guns use the time-of-flight principle to calculate the distance traveled by the laser beam and, subsequently, the vehicle's speed.

Inductive Loop Detectors: Inductive loop detectors are buried under the road surface and detect changes in inductance caused by passing vehicles. These detectors are often used in traffic signal systems and for traffic flow monitoring.

**3. Vehicle Speed Detection for Safety and Regulation Compliance:** Accurate speed detection is crucial for enforcing speed limits and ensuring compliance with traffic regulations. Law enforcement agencies use various speed detection methods to monitor vehicle speeds and issue citations to drivers exceeding the speed limit. Speed detection also plays a significant role in maintaining road safety, reducing the risk of accidents, and minimizing the severity of collisions.

**4. Vehicle Speed Detection for ADAS:** Vehicle Speed Detection is a key component of advanced driver assistance systems (ADAS) that require precise vehicle speed information to perform various functions. ADAS technologies, such as adaptive cruise control, lane-keeping assist, and collision avoidance systems, rely on accurate speed detection to make real-time decisions and adjustments to enhance driving safety.

**5. Fuel Efficiency and Eco-Driving:** Monitoring and displaying real-time vehicle speed to drivers can encourage eco-driving habits, where drivers maintain speeds within an optimal range for fuel efficiency. Eco-driving practices can lead to reduced fuel consumption, lower emissions, and overall environmental benefits.

**6. Challenges in Vehicle Speed Detection:** Vehicle Speed Detection may face challenges in certain conditions, such as adverse weather, radar interference, and calibration accuracy. Ensuring the reliability and accuracy of speed detection methods is essential to maintain the safety and integrity of transportation systems.

**CHAPTER 24**

**REVERSE PARKING ASSIST**

**1. Introduction:** Reverse Parking Assist, also known as Reverse Parking Sensors or Rear Parking Sensors, is an automotive safety feature designed to assist drivers when parking their vehicles in reverse. It utilizes various sensors and warning systems to provide drivers with audible or visual cues, enabling them to park more accurately and avoid potential collisions with obstacles or pedestrians.

**2. Working:** Reverse Parking Assist systems typically consist of ultrasonic sensors mounted on the rear bumper of the vehicle. These sensors emit ultrasonic waves that bounce off nearby obstacles and return to the sensors. By measuring the time it takes for the waves to return, the system can calculate the distance between the vehicle and the obstacle.

**3. Warning and Alert Mechanisms:** When the Reverse Parking Assist system detects an obstacle within a certain distance from the vehicle, it activates warning mechanisms to alert the driver. Common warning mechanisms include:

Audible Alerts: The system emits beeps or tones with varying frequencies, indicating the proximity of the obstacle. The beeps become more frequent as the vehicle gets closer to the obstacle.

Visual Alerts: Many vehicles equipped with Reverse Parking Assist display visual alerts on the infotainment screen or rearview mirror. These visual cues may include colored bars or graphics representing the distance to the obstacle.

**4. Benefits of Reverse Parking Assist:**

Enhanced Safety: Reverse Parking Assist helps prevent accidents and collisions while parking, especially in tight spaces or areas with limited visibility.

Saves Time and Effort: The system assists drivers in maneuvering the vehicle into parking spots accurately, reducing the time and effort required for parking.

Protects Pedestrians and Objects: Reverse Parking Assist not only protects the vehicle from damage but also helps prevent collisions with pedestrians and stationary objects, enhancing overall safety.

**5. Different Levels of Reverse Parking Assist:**

Basic Reverse Parking Sensors: Entry-level systems provide audible alerts when the vehicle approaches an obstacle during reverse parking.

Visual Display: Some systems include a visual display, showing the distance and location of obstacles on the infotainment screen or rearview mirror.

Rear Cross-Traffic Alert: Advanced systems may incorporate Rear Cross-Traffic Alert, which warns the driver of approaching vehicles or pedestrians while reversing out of parking spots.

**6. Limitations of Reverse Parking Assist:**

Limited Range: The ultrasonic sensors used in Reverse Parking Assist have a limited range, typically up to a few meters. Drivers should always visually confirm their surroundings and exercise caution while parking.

Sensor Blind Spots: Some obstacles may be too low or narrow for the sensors to detect, leading to potential blind spots in the warning system.

Weather Conditions: Adverse weather conditions, such as heavy rain or snow, may affect the performance of the ultrasonic sensors.

**CHAPTER 25**

**DIGITAL INSTRUMENT CLUSTER DESIGN**

**1**. Introduction: A Digital Instrument Cluster, also known as a Digital Dash or Digital Cockpit, is a modern replacement for traditional analog instrument clusters found in vehicles. It utilizes digital displays, typically LCD or OLED screens, to present important vehicle information to the driver in a more visually appealing and customizable manner. Digital Instrument Clusters have become increasingly popular in the automotive industry due to their enhanced functionality, aesthetics, and potential for personalization.

2. Components of a Digital Instrument Cluster:

Main Display Screen: The main component of the digital instrument cluster is the high-resolution display screen. It serves as the primary interface to present essential driving information and vehicle data.

Virtual Gauges: Digital Instrument Clusters can emulate traditional analog gauges, such as speedometers, tachometers, fuel level indicators, and engine temperature gauges, using realistic graphical representations.

Information Panels: Alongside virtual gauges, the display can feature information panels to provide additional data, including navigation directions, media information, vehicle status, and more.

User Interface (UI): The user interface enables drivers to interact with the digital instrument cluster, accessing menus, settings, and customization options through buttons, touchscreens, or steering wheel controls.

3. Customization and Personalization: One of the significant advantages of digital instrument clusters is their high level of customization. Drivers can personalize the display to suit their preferences, choosing from various themes, color schemes, and layouts. This customization allows for a more user-friendly and immersive driving experience.

4. Advanced Features and Integration: Digital Instrument Clusters can integrate with other vehicle systems and technologies, enhancing the overall driving experience. Some advanced features include:

Advanced Driver Assistance System (ADAS) Integration: The cluster can display alerts and warnings related to ADAS functionalities, such as lane departure warnings and collision avoidance notifications.

Navigation Integration: GPS navigation information can be displayed directly on the instrument cluster, providing real-time turn-by-turn directions.

Connectivity and Infotainment: The digital cluster can display media information, incoming calls, and other connected features.

5. Safety and Driver Focus: Digital Instrument Clusters are designed to minimize driver distraction. They offer clear and concise information presentation, reducing the need for drivers to look away from the road extensively.

6. Hybrid Instrument Clusters: Some vehicles feature a hybrid instrument cluster, combining analog gauges with digital elements. In such clusters, traditional gauges are complemented by a smaller digital display, offering additional data and features.

7. Future Trends: As automotive technology continues to advance, digital instrument clusters are expected to incorporate even more sophisticated features, including augmented reality displays, advanced voice control, and seamless integration with vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication systems.

# REFERENCES

1. B. Bharathi, B. Suchitha Samuel, ‘Design and Construction of Rescue Robot and Pipeline Inspection Using Zigbee’, International Journal Of Scientific Engineering and Research (IJSER), vol. 1, no. 1, September 2013.
2. S. Gopinath, T. Devika, L. Manivannan, N. Suthanthira Vanitha, ‘Rescue Child from Bore well using Embedded System’, 2015.
3. John Jose pottery, ‘robot for bore well rescue’, amal jothi college of engineering, vol. 10, Jun 2009.
4. G Kavianand; K Gowri Ganesh ; P Karthikeyan, Smart child rescue system from borewell (SCRS), Published in: Emerging Trends in Engineering, Technology and Science (ICETETS), International Conference on , 24-26 Feb. 2016,IEEE Xplore: 24 October 2016
5. C. Kemp, A. Edsinger 2007, ‘Challenges for Robot Manipulation in Human Environments’, IEEE Robotics & Automation Magazine, 2007 pp. 229.
6. G. Nithin, G. Gowtham, G. Venkatachalam, and S. Narayanan ‘Design and Simulation of Bore well rescue robot’ Advanced School of Mechanical Building Sciences, VIT University, India, ARPN Journal of Engineering and Applied Sciences, pp. MAY 2014.
7. Pal winder kaur, Ravinder kaur, Gurpreet Singh, ‘Pipeline Inspection And Bore well Rescue Robot’, International Journal of Research In Engineering and Technology(IJRET), vol. 03, no. 04, April 2014.
8. Raj Manish, P. Chakraborty, G.C. Nandi, "Rescue robotics in bore well Environment", Cornell university library [v1] Mon, Jun 2014.
9. Dr. C.N. Sakhale, 2D.M. Mate Subhasis Saha, Tomar Dharmpal, Pranjit Kar, Arindam Sarkar, Roam Choudhury, Shahil Kumar, ‘An Approach to Design of Child Saver Machine for Child Trapped in Borehole’ International Journal of Research in Mechanical Engineering, October- December, 2013, pp. 26-38.
10. K. Saran1, S.Vignesh2, Marlon Jones Louis ‘Bore-well rescue robot’, International Journal of Research in Aeronautical and Mechanical Engineering, Boar well rescue robot, pp. 20-30 April 2014.
11. Sridhar Palaniswamy, "Life Saving Machine", first International Conference on Interdisciplinary Research and Development, 31May-1 June 2011.
12. V. Venatic, E. Poornima, S. Sumathi, ‘Borewell Rescue Robot’, International Journal of Computer Applications, vol. 113, no. 14, 2015.

# APPENDIX

#include <htc.h>

CONFIG (0x2F0A);

#define \_XTAL\_FREQ 8000000 #define rs RB0

#define rw RB1 #define en RB2

#define lcdport PORTB #define FFF RD0 #define BBB RD1 #define RRR RD2 #define LLL RD3 #define RELAY1 RC0 #define RELAY2 RC1 #define RELAY3 RC2 #define RELAY4 RC3 #define RELAY5 RC4

#define RELAY6 RC5

#define buzzer RC6 #define ir RC7

void lcd\_ini();

void dis\_cmd(unsigned char); void dis\_data(unsigned char); void lcdcmd(unsigned char); void lcddata(unsigned char);

void lcd\_data\_string(unsigned char \*str) ; void conv2(unsigned int count);

void init\_a2d(void);

unsigned char read\_a2d(unsigned char channel); unsigned int xx,yy,zz,ww,ss,bb,tt;

void conv(unsigned int count); void delay(unsigned int msec )

{

int i ,j ; for(i=0;i<msec;i++)

for(j=0; j<700; j++);

}

void init\_a2d(void)

{

ADCON0=0; // select Fosc/2

ADCON1=0; // select left justify result. A/D port configuration 0 ADON=1; // turn on the A2D conversion module

}

unsigned char read\_a2d(unsigned char channel)

{

channel&=0x07; // truncate channel to 3 bits ADCON0&=0xC5; // clear current channel select ADCON0|=(channel<<3); // apply the new channel select ADGO=1; // initiate conversion on the selected channel while(ADGO)continue;

return(ADRESH); // return 8 MSB of the result

}

void conv(unsigned int count)

{

unsigned int dig1,dig2,dig3,temp; dig1=count/100; dig1=dig1+=0x30; temp=count%100; dig2=temp/10;

dig2=dig2+0x30; dig3=count%10; dig3=dig3+0x30; dis\_data(dig1); dis\_data(dig2); dis\_data(dig3);

}

void conv2(unsigned int count)

{

unsigned int dig1,dig2,dig3,temp; dig1=count/100; dig1=dig1+=0x30;

temp=count%100;

dig2=temp/10; dig2=dig2+0x30; dig3=count%10; dig3=dig3+0x30;

// dis\_data(dig1); dis\_data(dig2); dis\_data(dig3);

}

void main(void)

{

TRISA=0xFF; TRISB=0x00; TRISC=0x80; TRISD=0xFF;

init\_a2d();

lcd\_ini(); // LCD initialization buzzer=0;

while(1)

{

tt=0; tt=read\_a2d(0); tt=tt\*2; dis\_cmd(0x80);

lcd\_data\_string("T:"); conv(tt);

dis\_data('o');dis\_data('C'); ww=0;

ww=read\_a2d(1); ww=ww\*0.08\*4; dis\_cmd(0x89); lcd\_data\_string("G:");

conv2(ww); lcd\_data\_string (“ppm”);

f(tt>=50)

{

buzzer=1;

}

else

{

buzzer=0;

}

}

}

void lcd\_ini()

{

dis\_cmd(0x02); // To initialize LCD in 4-bit mode. dis\_cmd(0x28); // To initialize LCD in 2 lines, 5x7 dots and

4bit mode.

dis\_cmd(0x0C); dis\_cmd(0x06); dis\_cmd(0x80);

}

void dis\_cmd(unsigned char cmd\_value)

{

unsigned char cmd\_value1;

cmd\_value1 = (cmd\_value & 0xF0); // Mask lower nibble because RB4-RB7 pins are being used

lcdcmd(cmd\_value1); // Send to LCD

cmd\_value1 = ((cmd\_value<<4) & 0xF0); // Shift 4-bit and mask lcdcmd(cmd\_value1); // Send to LCD

}

void dis\_data(unsigned char data\_value)

{

unsigned char data\_value1; data\_value1=(data\_value&0xF0); lcddata(data\_value1); data\_value1=((data\_value<<4)&0xF0); lcddata(data\_value1);

}

void lcdcmd(unsigned char cmdout)

{

lcdport=cmdout; //Send command to lcdport=PORTB rs=0;

rw=0; en=1; delay(1); en=0;

}

void lcddata(unsigned char dataout)

{

lcdport=dataout; //Send data to lcdport=PORTB rs=1;

rw=0; en=1; delay(1); en=0;

}

void lcd\_data\_string(unsigned char \*str)

{

int i=0; while(str[i]!='\0')

{

dis\_data(str[i]); i++;

}

return;

}