Project Report

On

Forward Collision Warning System

Submitted in partial fulfilment for the award of

Post Graduate Diploma in Embedded System& Design

From C-DAC, ACTS (Pune)

Guided by:

Mr. Rishabh Hardas

Presented by:

Ankit Kumar Choudhary	**190240130010**
Juhi Khichi	**190240130038**
Mane Rahul Ramesh	**190240130043**
Mansi Gupta	**190240130045**

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

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ankit Kumar Choudhary	**190240130010**
Juhi Khichi	**190240130038**
Mane Rahul Ramesh	**190240130043**
Mansi Gupta	**190240130045**

have successfully completed their project on

Forward Collision Warning System

Under the guidance of Mr. Rishabh Hardas

Project Guide **Project Supervisor**

HOD ACTS

Mr. Aditya K Sinha

ACKNOWLEDGEMENT

This is to acknowledge our indebtedness to our guide for his constant guidance and helpful suggestion for preparing this project "**Forward Collision Warning System**". Our deep gratitude towards him for inspiration, personal involvement, constructive criticism that he provided beyond more technical guidance during the course of this project. His shared enthusiasm taught us patience and his word of advice acted as morale booster. We are very thankful to our guide **Mr. Rishabh Hardas** for his inspiration.

I take this opportunity to thank Head of the department **Mr. Aditya K Sinha** and all staff members for cooperation provided by them in many ways. Our most heartfelt thanks goes to **Mrs. Purvi Parmar** (Course Coordinator, PG-DESD) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

Contents

1. Abstract

2. Literature Survey

    2.1 ARM Cortex M4 Architecture

    2.2 STM32F4 Discovery Board

    2.3 CAN Bus Protocol

    2.4 CAN Network

        2.4.1 CAN Layers

        2.4.2 CAN Frames

        2.4.3 Bus Arbitration

        2.4.4 CAN Specification in STM32F4

    2.5 I2C Protocol

3. Hardware Requirements

    3.1 Introduction to STM32F407 Discovery Board

    3.2 CAN Transceiver

    3.3 Ultrasonic Sensor HCSR04

    3.4 OLED

    3.5 LED and Buzzer

4. Software Requirements

    4.1 Keil IDE

5. Block Diagram and State Diagram

    5.1 Block Diagram

    5.2 State Diagram

    5.3 Working

6. Conclusion

7. References

Chapter 1

Abstract

A forward collision warning (FCW) system is an advanced safety technology that monitors a vehicle's speed, the speed of the vehicle in front of it, and the distance between the vehicles. If vehicles get too close due to the speed of the rear vehicle, the FCW system will warn that driver of an impending crash. It's important to note that FCW systems do not take full control of the vehicle or keep the driver from operating it.

FCW systems use sensors to detect slower-moving or stationary vehicles. When the distance between vehicles becomes so short that a crash is imminent, a signal alerts the driver so that the driver can apply the brakes or take evasive action, such as steering, to prevent a potential crash. Vehicles with this technology provide drivers with an audible alert, a visual display, or other warning signals.

We want to detect the collision between the two running cars in distance between 3-5(approx) meter and give the warning when the speed is more than safety Range, and to warn driver by buzzer or something else mechanism which is trending in the market. We implement the project by using proximity sensors to measure the distance between two moving cars and sending the data from sensor via CAN to output devices like Buzzer or display or LED.

Chapter 2

Literature Survey

2.1 ARM Cortex M4 Architecture

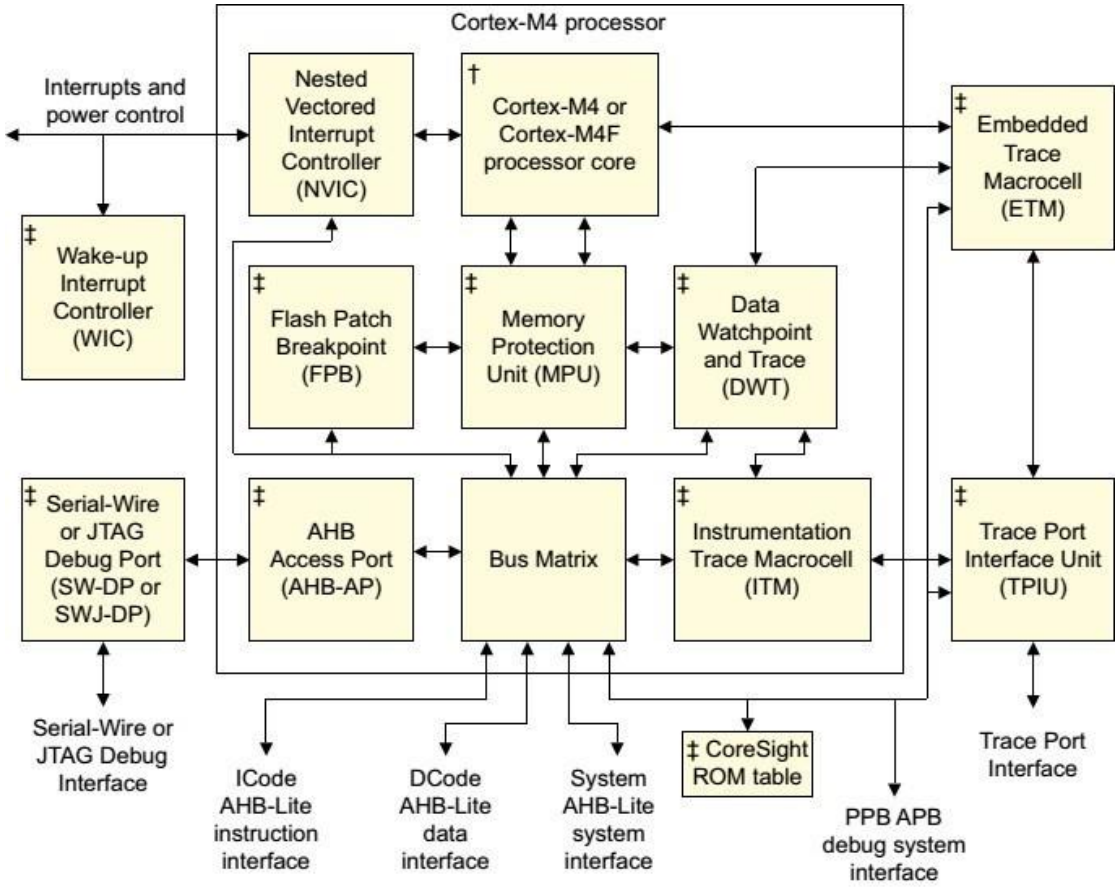


Figure 2.1- A simplified block diagram of ARM Cortex M4

The ARM® Cortex®-M4 processor is a high performance embedded processor with DSP instructions developed to address digital signal control markets that demand an efficient, easy-to-use blend of control and signal processing capabilities. The processor is highly configurable enabling a wide range of implementations from those requiring floating point operations, memory protection and powerful trace technology to cost sensitive devices requiring minimal area.

2.1.1 Features

The Cortex-M4 processor features:

• A low gate count processor core, with low latency interrupt processing thathas:
— A subset of the Thumb instruction set, defined in the ARMv7-M Architecture Reference manual.
— Banked Stack Pointer (SP).
— Hardware divide instructions, SDIV and UDIV.
— Handler and Thread modes.
— Thumb and Debug states.
— Support for interruptible-continued instructions LDM, STM, PUSH, and POP for low interrupt latency.
— Automatic processor state saving and restoration for low latency Interrupt Service Routine (ISR) entry and exit.
— Support for ARMv6 big-endian byte-invariant or little-endian accesses.
— Support for ARMv6 unaligned accesses.

• **Floating Point Unit (FPU) in the Cortex-M4F processor providing:**
— 32-bit instructions for single-precision (C float) data-processing operations.
— Combined Multiply and Accumulate instructions for increased precision (Fused MAC).
— Hardware support for conversion, addition, subtraction, multiplication with optional accumulate, division, and square-root.
— Hardware support for de-normal and all IEEE rounding modes.
— 32 dedicated 32-bit single precision registers, also addressable as 16 double-word registers.
— Decoupled three-stage pipeline.

• **Nested Vectored Interrupt Controller (NVIC) closely integrated with the processor core to achieve low latency interrupt processing.**
Features include:
— External interrupts, configurable from 1 to 240.
— Bits of priority, configurable from 3 to8.
— Dynamic reprioritization of interrupts.
— Priority grouping. This enables selection of preempting interrupt levels and non-preempting interrupt levels.
— Support for tail-chaining and late arrival of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
— Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.
— Optional Wake-up Interrupt Controller (WIC), providing ultra-low power sleep mode support.

• **Memory Protection Unit (MPU). An optional MPU for memory protection, including:**
— Eight memory regions.
— Sub Region Disable (SRD), enabling efficient use of memory regions.
— The ability to enable a background region that implements the default memory map attributes.

• **Bus interfaces:**
— Three Advanced High-performance Bus-Lite (AHB-Lite) interfaces: I Code, D Code, and System bus interfaces.
— Private Peripheral Bus (PPB) based on Advanced Peripheral Bus (APB) interface.
— Bit-band support that includes atomic bit-band write and read operations.
— Memory access alignment.
— Write buffer for buffering of write data.
— Exclusive access transfers for multiprocessor systems.

• **Low-cost debug solution that features:**
— Debug access to all memory and registers in the system, including access to memory mapped devices, access to internal core registers when the core is halted, and access to debug control registers even while SYS RESETn is asserted.
— Serial Wire Debug Port (SW-DP) or Serial Wire JTAG Debug Port (SWJ-DP) debug access, or both.

— Optional Flash Patch and Breakpoint (FPB) unit for implementing breakpoint sand code patches.
— Optional Data Watch point and Trace (DWT) unit for implementing watch points, data tracing, and system profiling.
— Optional Instrumentation Trace Macrocell (ITM) for support of printf style debugging.
— Optional Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer (TPA), including Single Wire Output (SWO) mode.
— Optional Embedded Trace Macrocell (ETM) for instruction trace.

2.2 STM32F4 Discovery Board

The STM32F4DISCOVERY Discovery kit allows users to easily develop applications with the STM32F407VG high performance microcontroller with the ARM® Cortex®- M4 32-bit core. It includes everything required either for beginners or for experienced users to get quickly started. Based on STM32F407VG, it includes an ST-LINK/V2 or ST-LINK/V2-A embedded debug tool, two ST-MEMS digital accelerometers, a digital microphone, one audio DAC with integrated class D speaker driver, LEDs, push buttons and a USB OTG micro-AB connector.

Figure 2.2 STM32F4 Discovery Board

To expand the functionality of the STM32F4DISCOVERY Discovery kit with the Ethernet connectivity, LCD display and more, visit the www.st.com/stm32f4dis-expansion webpage. The STM32F4DISCOVERY Discovery kit comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube package, as well as a direct access to the ARM® m-bed Enabled™.

2.3 CAN Bus Protocol

The CAN bus was developed by BOSCH as a multi-master, message broadcast system that specifies a maximum signaling rate of 1 megabit per second (bps). Unlike a traditional network such as USB or Ethernet, CAN does not send large blocks of data point-to-point from node A to node B under the supervision of a central bus master. In a CAN network, many short messages like temperature or RPM are broadcast to the entire network, which provides for data consistency in every node of the system. Once CAN basics such as message format, message identifiers, and bit-wise arbitration -- a major benefit of the CAN signaling scheme are explained, a CAN bus implementation is examined, typical waveforms presented, and transceiver features examined.

CAN is an International Standardization Organization (ISO-11898) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The specification calls for high immunity to electrical interference and the ability to self-diagnose and repair data errors. These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing.

CAN is a reliable, real-time protocol that implements a multicast, data-push, publisher /subscriber model. CAN messages are short (data payloads are a maximum of 8 bytes, headers are 11 or 29 bits), so there is no centralized master or hub to be a single point of failure and it is flexible in size. Its real-time features include deterministic message delivery time and global priority through the use of prioritized message IDs.

Bus arbitration is accomplished in CAN using bit dominance, a process where nodes begin to transmit their message headers on the bus, then drop out of the "competition" when a dominant bit is detected on the bus, indicating a message ID of higher priority being transmitted elsewhere. This means bus arbitration does not add overhead because once the bus is "won" the node simply continues sending its message. Because there is no time lost to collisions on a heavily loaded network.

2.4 CAN Network

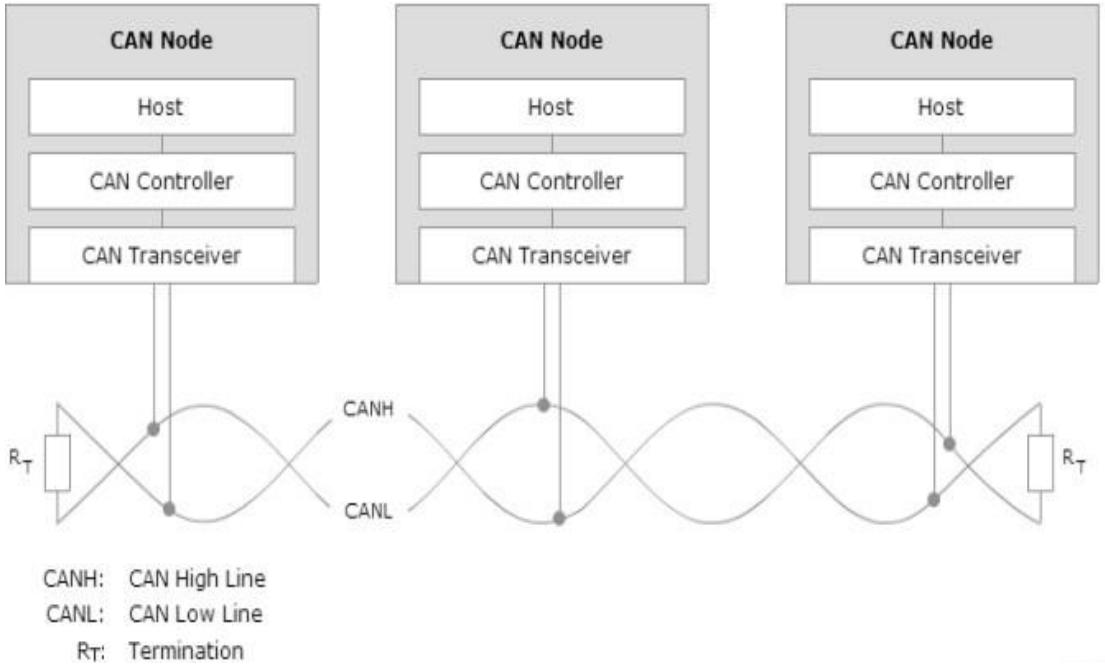


Figure 2.4 CAN Network

A CAN network consists of a number of CAN nodes which are linked via a physical transmission medium (CAN bus). In practice, the CAN network is usually based on a line topology with a linear bus to which a number of electronic control units are each connected via a CAN interface. The passive star topology may be used as an alternative. An unshielded twisted two-wire line is the physical transmission medium used most frequently in applications (Unshielded Twisted Pair — UTP), over which symmetrical signal transmission occurs. The maximum data rate is 1 Mbit/s. A maximum network extension of about 40 meters is allowed. At the ends of the CAN network, bus termination resistors contribute to preventing transient phenomena (reflections). ISO 11898 specifies the maximum number of CAN nodes as 32.

CAN Bus

Physical signal transmission in a CAN network is based on transmission of differential voltages **(differential signal transmission**). This effectively eliminates the negative effects of interference voltages induced by motors, ignition systems and switch contacts. Consequently, the transmission medium (CAN bus) consists of two lines: CAN High and CAN Low.

Twisting of the two lines reduces the magnetic field considerably. Therefore, in practice twisted pair conductors are generally used as the physical transmission medium. Due to finite signal propagation speed, the effects of transient phenomena (**reflections**) grow with increasing data rate and bus extension. Terminating the ends of the communication channel using **termination resistors** (simulation of the electrical properties of the transmission medium) prevents reflections in a high-speed CAN network.

CAN Bus Levels

Physical signal transmission in a CAN network is based on differential signal transmission. The specific differential voltages depend on the bus interface that is used. A distinction is made here between the high-speed CAN bus interface (ISO 11898-2) and the low-speed bus interface (ISO 11898-3).

ISO 11898-2 assigns logical “1” to a typical differential voltage of 0 Volt. The logical “0” is assigned with a typical differential voltage of 2 Volt. High-speed CAN transceivers interpret a differential voltage of more than 0.9 Volt as a dominant level within the common mode operating range, typically between 12 Volt and -12 Volts. Below 0.5 Volt, however, the differential voltage is interpreted as a recessive level. A hysteresis circuit increases immunity to interference voltages. ISO 11898-3 assigns a typical differential voltage of 5 Volt to logical “1”, and a typical differential voltage of 2 Volt corresponds to logical “0”. The figure “High-Speed CAN Bus Levels” and the figure “Low-Speed CAN Bus Levels” depict the different voltage relationships on the CAN bus.

High Speed CAN:

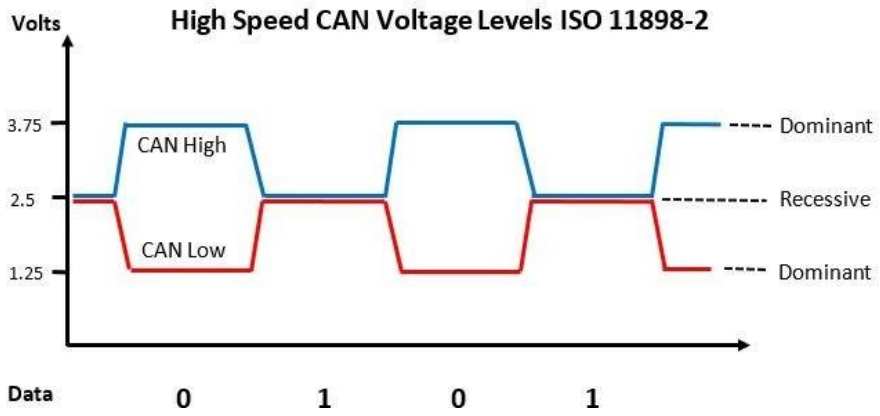


Figure 2.5 High speed CAN voltage level

Low Speed CAN:

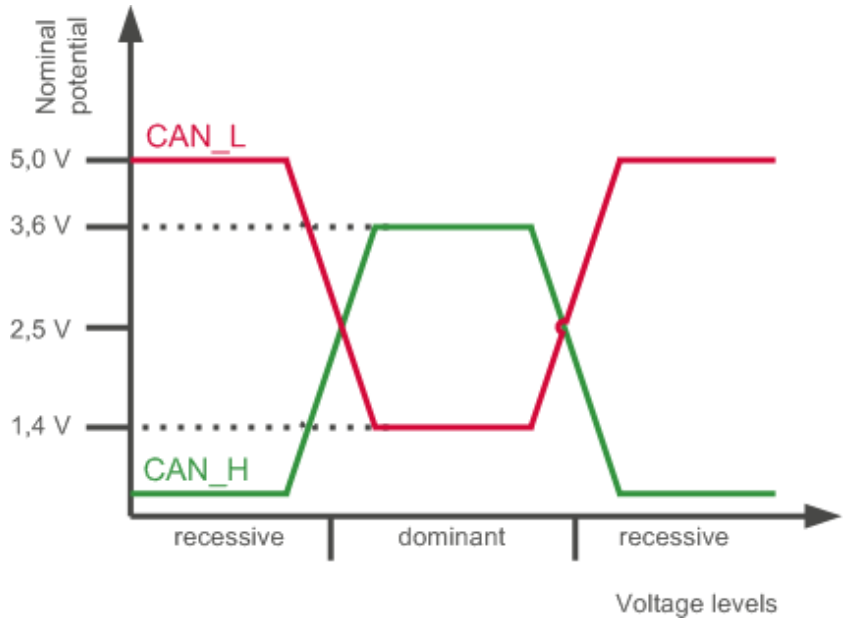
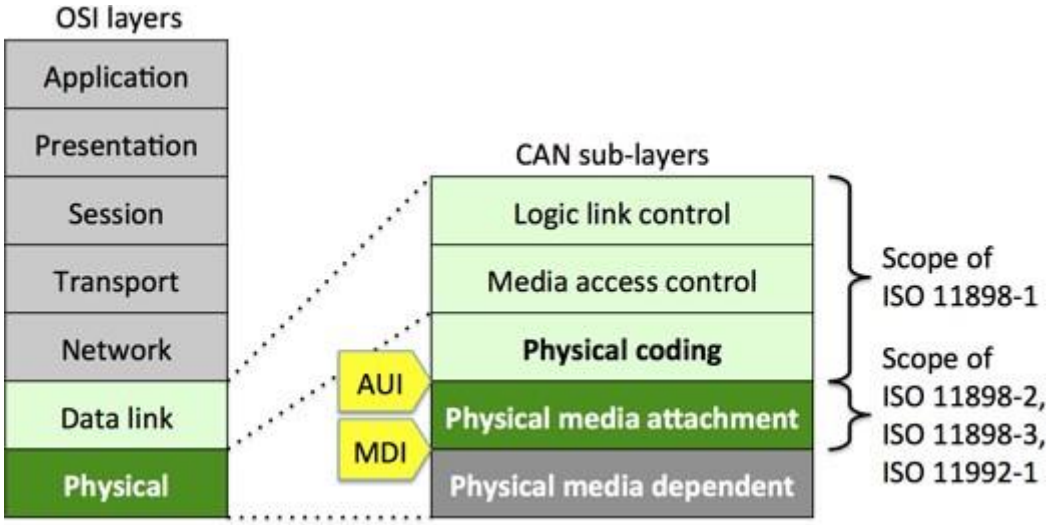


Figure 2.6 Low speed CAN voltage levels

2.4.1 CAN Layers

The CAN protocol, like many networking protocols, can be decomposed into the following abstraction layers.

Figure 2.7 CAN Layers



Data Link Layer:

Most of the CAN standard applies to the transfer layer. The transfer layer receives messages from the physical layer and transmits those messages to the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgement, error detection and signaling and fault confinement.

It performs:

1. Error Detection
2. Message Validation
3. Acknowledgement
4. Arbitration
5. Message Framing

Physical Layer:

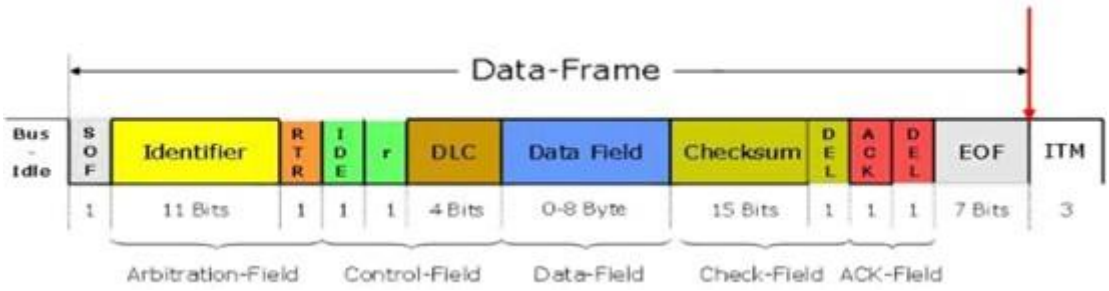
CAN bus specify the link layer protocol with only abstract requirements for the physical layer. As a result, implementation often employs custom connector with various sorts of cables, of which two are the CAN bus lines. Noise immunity is achieved by maintaining the differential impedance of the bus at a low level with low-value resistors (120 ohms) at each end of the bus.

2.4.2 CAN Frames

1. Data Frame
2. Remote Frame
3. Error frame
4. Overload Frame

Data Frame

The data frame is the most common message type, and comprises the Arbitration Field, the Data Field, the CRC Field, and the Acknowledgment Field. The Arbitration Field contains an 11-bit identifier and the RTR bit, which is dominant for data frames. Next is the Data Field which contains zero to eight bytes of data, and the CRC Field which contains the 16-bit



checksum used for error detection. Last is the Acknowledgment Field.

Figure 2.8 Data Frame

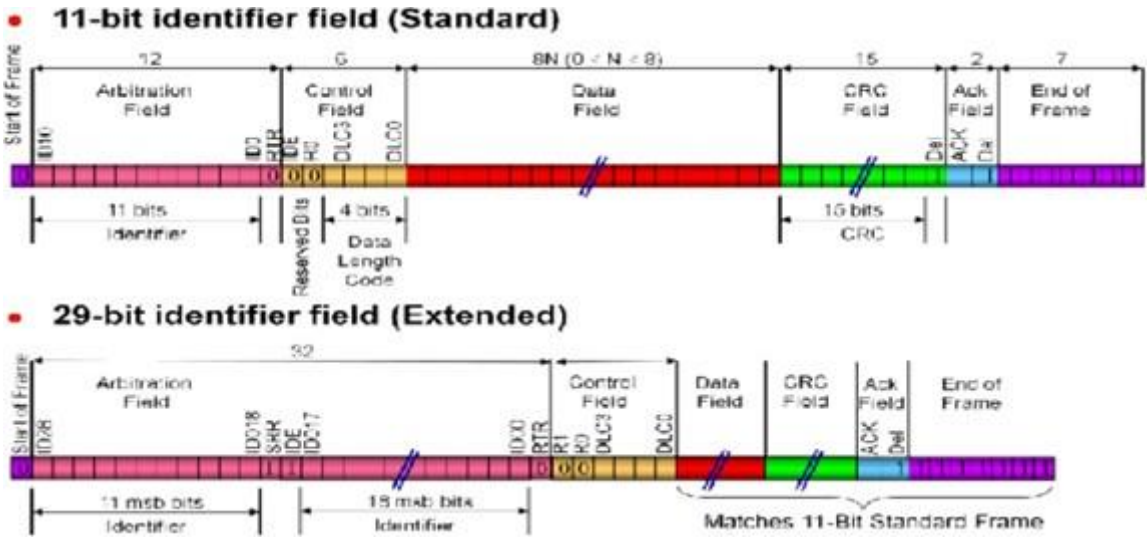


Figure 2.9 Standard and Extended identifier fields

Remote Frame:

The intended purpose of the remote frame is to solicit the transmission of data from another node. The remote frame is similar to the data frame, with two important differences. First, this type of message is explicitly marked as a remote frame by a recessive RTR bit in the arbitration field, and secondly, there is no data

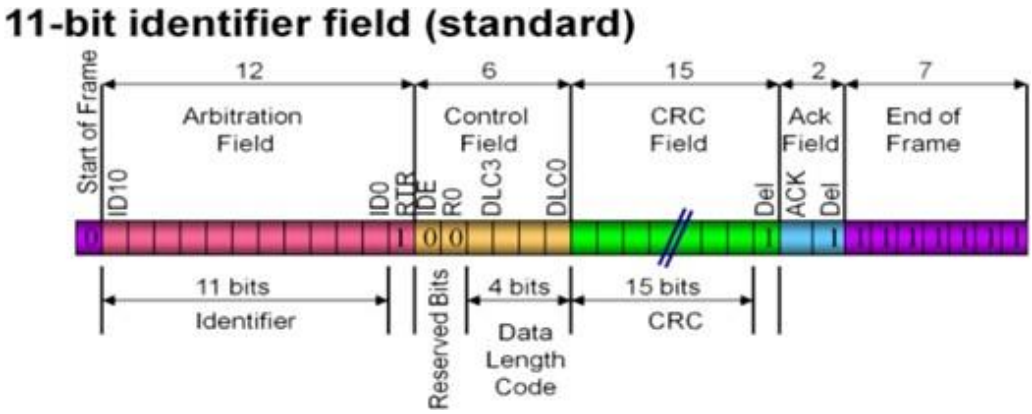


Figure 2.10 Remote frame

Error Frame:

The error frame is a special message that violates the formatting rules of a CAN message. It is transmitted when a node detects an error in a message, and causes all other nodes in the network to send an error frame as well. The original transmitter then automatically retransmits the message. An elaborate system of error counters in the CAN controller ensures that a node cannot tie up a bus by repeatedly transmitting error frames.

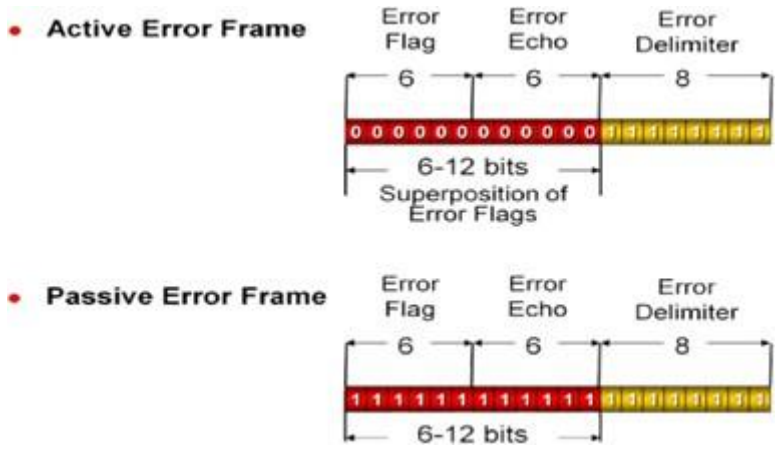


Figure 2.11 Error frames

Overload Frame:

The overload frame is mentioned for completeness. It is similar to the error frame with regard to the format, and it is transmitted by a node that becomes too busy. It is primarily used to provide for an extra delay between messages.

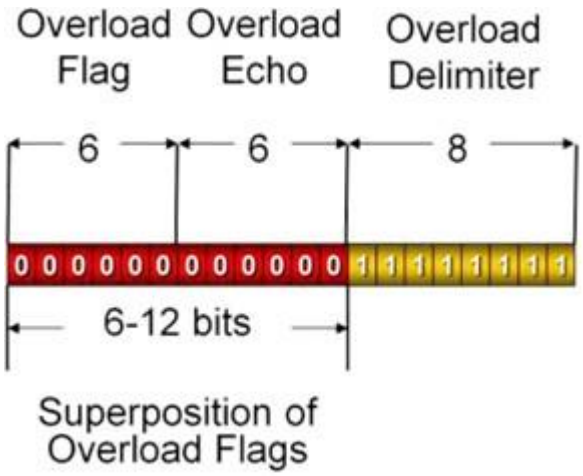


Figure 2.12 Overload frame

2.4.3 Bus Arbitration

The CAN communication protocol is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP mean that collisions are resolved through a bit-wise arbitration, based on a pre-programmed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access. That is, the last logic-high in the identifier keeps on transmitting because it is the highest priority.

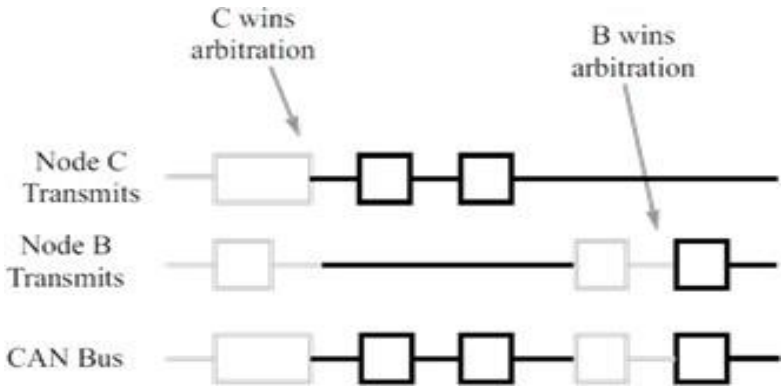


Figure 2.13 Bus Arbitration

Whenever the bus is free, any unit may start to transmit a message. If two or more units start transmitting messages at the same time, the bus access conflict is resolved by bit-wise arbitration using the Identifier. The mechanism of arbitration guarantees that neither information nor time is lost. If a data frame and a remote frame with the same identifier are initiated at the same time, the data frame prevails over the remote frame. During arbitration every transmitter compares the level of the bit transmitted with the level that is monitored onthe bus. If these levels are equal the unit may continue to send. When a 'recessive' level is sent and a 'dominant' level is monitored, the unit has lost arbitration and must withdraw without sending one more bit.

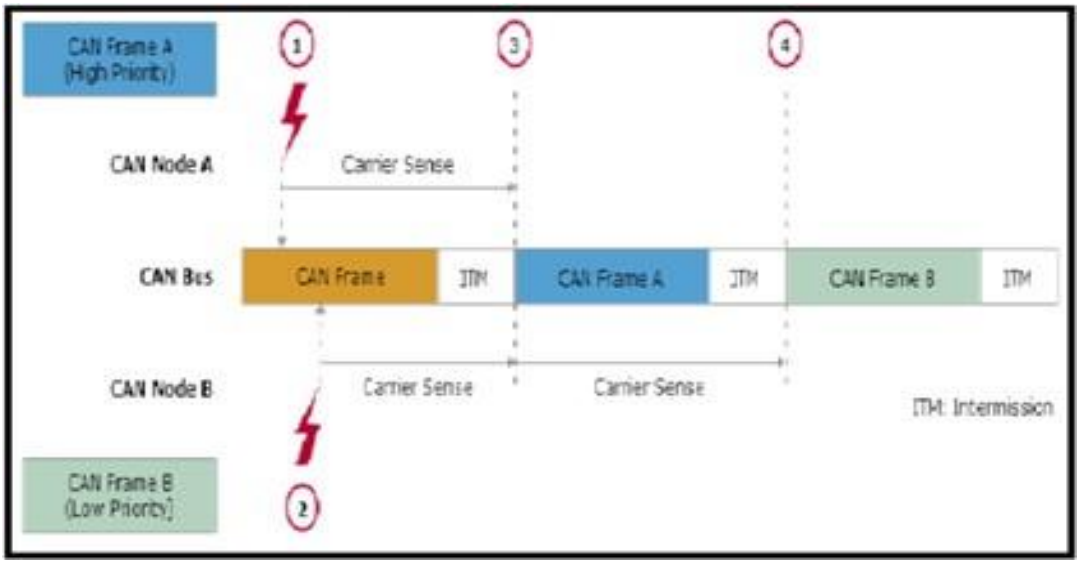


Figure 2.14 Carrier Sense

2.4.4 CAN specification in STM32F4

The **Basic Extended CAN** peripheral, named **bxCAN**, interfaces the CAN network. It supports the CAN protocols version 2.0A and B. It has been designed to manage a high number of incoming messages efficiently with a minimum CPU load. It also meets the priority requirements for transmit messages.
For safety-critical applications, the CAN controller provides all hardware functions for supporting the CAN Time Triggered Communication option.

bxCAN main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

Reception

- Two receive FIFOs with three stages
- Scalable filter banks:
- 28 filter banks shared between CAN1 and CAN2
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

Time-triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Time Stamp sent in last two data bytes

Management

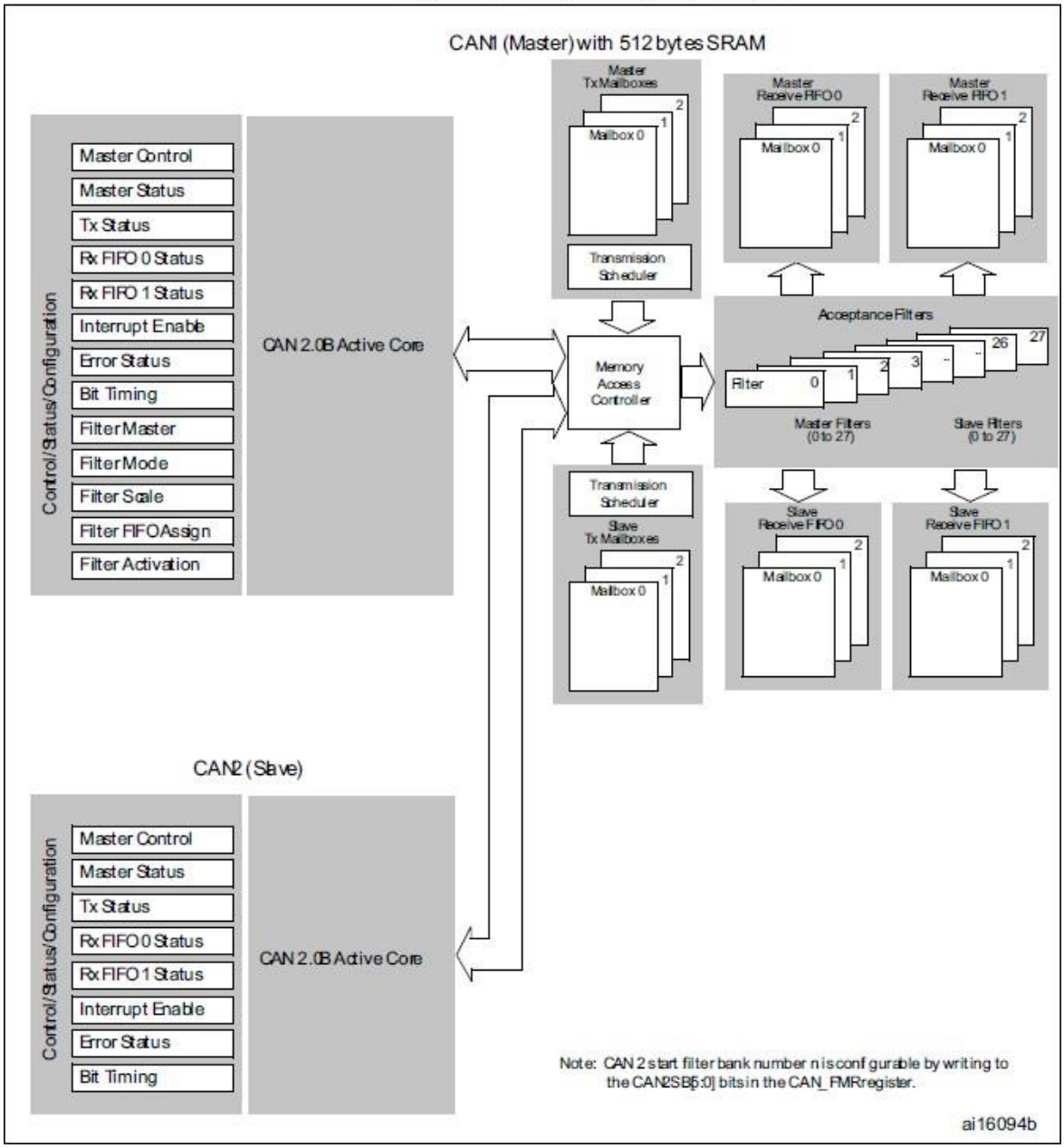
- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

Dual CAN

- CAN1: Master bxCAN for managing the communication between a Slave bxCAN and the 512-byte SRAM memory
- CAN2: Slave bxCAN, with no direct access to the SRAM memory.
- The two bxCAN cells share the 512-byte SRAM memory

.

Figure 335. Dual CAN block diagram



bxCAN operating modes

BxCAN has three main operating modes: **initialization**, **normal** and **Sleep**. After a hardware reset, bxCAN is in Sleep mode to reduce power consumption and an internal pull- up is active on CANTX. The software requests bxCAN to enter **initialization** or **Sleep** mode by setting the INRQ or SLEEP bits in the CAN_MCR register. Once the mode has been entered, bxCAN confirms it by setting the INAK or SLAK bits in the CAN_MSR register and the internal pull-up is disabled.

Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter this mode the software sets the INRQ bit in the CAN_MCR register and waits until the hardware has confirmed the request by setting the INAK bit in the CAN_MSR register .To leave Initialization mode, the software clears the INQR bit. bxCAN has left Initialization mode once the INAK bit has been cleared by hardware.
While in Initialization Mode, all message transfers to and from the CAN bus are stopped and the status of the CAN bus output CANTX is recessive (high).

Entering Initialization Mode does not change any of the configuration registers.
To initialize the CAN Controller, software has to set up the Bit Timing (CAN_BTR) and CAN options (CAN_MCR) registers .To initialize the registers associated with the CAN filter banks (mode, scale, FIFO assignment, activation and filter values), software has to set the FINIT bit (CAN_FMR). Filter initialization also can be done outside the initialization mode. *Note:* When FINIT=1, CAN reception is deactivated.

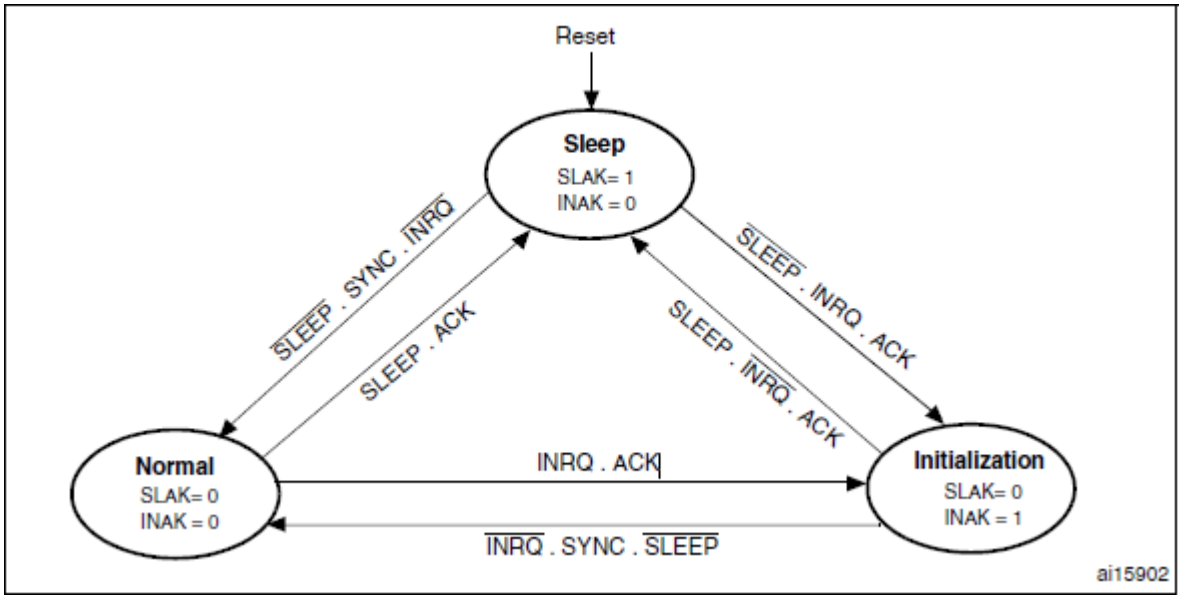
The filter values also can be modified by deactivating the associated filter activation bits (in the CAN_FA1R register).
If a filter bank is not used, it is recommended to leave it non active (leave the corresponding FACT bit cleared).

Normal mode

Once the initialization is complete, the software must request the hardware to enter Normal mode to be able to synchronize on the CAN bus and start reception and transmission. The request to enter Normal mode is issued by clearing the INRQ bit in the CAN_MCR register. The bxCAN enters Normal mode and is ready to take part in bus activities when it has synchronized with the data transfer on the CAN bus. This is done by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle state). The switch to Normal mode is confirmed by the hardware by clearing the INAK bit in the CAN_MSR register.
The initialization of the filter values is independent from Initialization Mode but must be done while the filter is not active (corresponding FACTx bit cleared). The filter scale and mode configuration must be configured before entering Normal Mode.

Sleep mode (low power)

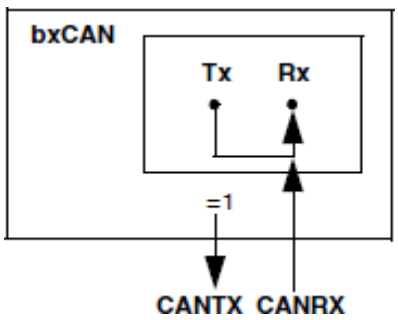
To reduce power consumption, bxCAN has a low-power mode called Sleep mode. This mode is entered on software request by setting the SLEEP bit in the CAN_MCR register. In this mode, the bxCAN clock is stopped, however software can still access the bxCAN mailboxes. If software requests entry to **initialization** mode by setting the INRQ bit while bxCAN is in **Sleep** mode, it must also clear the SLEEP bit.



Test modes

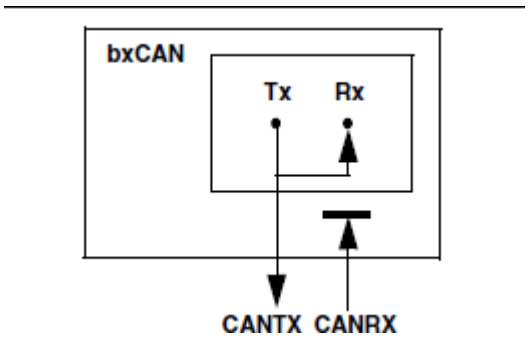
Silent mode

The bxCAN can be put in Silent mode by setting the SILM bit in the CAN_BTR register. In Silent mode, the bxCAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the bxCAN has to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).



Loop back mode

The bxCAN can be set in Loop Back Mode by setting the LBKM bit in the CAN_BTR register. In Loop Back Mode, the bxCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in a Receive mailbox.



This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

Transmission handling

In order to transmit a message, the application must select one **empty** transmit mailbox, set up the identifier, the data length code (DLC) and the data before requesting the transmission by setting the corresponding TXRQ bit in the CAN_TIxR register. Once the mailbox has left **empty** state, the software no longer has write access to the mailbox registers. Immediately after the TXRQ bit has been set, the mailbox enters **pending** state and waits to become the highest priority mailbox, see *Transmit Priority*. As soon as the mailbox has the highest priority it will be **scheduled** for transmission. The transmission of the message of the scheduled mailbox will start (enter **transmit** state) when the CAN bus becomes idle. Once the mailbox has been successfully transmitted, it will become **empty** again. The hardware indicates a successful transmission by setting the RQCP and TXOK bits in the CAN_TSR register.
If the transmission fails, the cause is indicated by the ALST bit in the CAN_TSR register in case of an Arbitration Lost, and/or the TERR bit, in case of transmission error detection.

Transmit priority

By identifier When more than one transmit mailbox is pending, the transmission order is given by the identifier of the message stored in the mailbox. The message with the lowest identifier value has the highest priority according to the arbitration of the CAN protocol. If the identifier values are equal, the lower mailbox number will be scheduled first. By transmit request order the transmit mailboxes can be configured as a transmit FIFO by setting the TXFP bit in the CAN_MCR register. In this mode the priority order is given by the transmit request order.
This mode is very useful for segmented transmission.

Abort

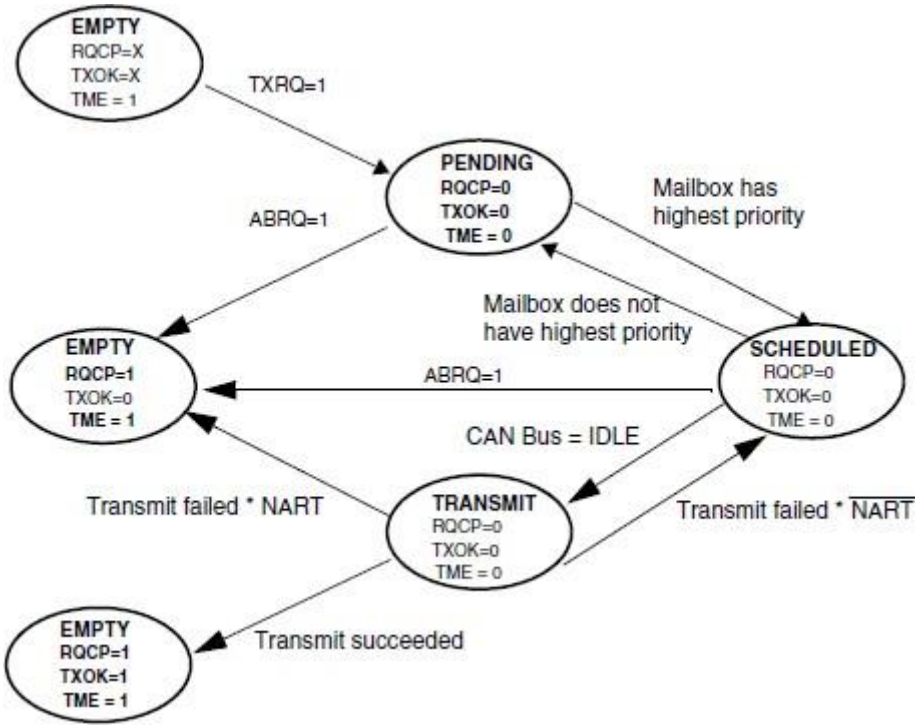
A transmission request can be aborted by the user setting the ABRQ bit in the CAN_TSR register. In **pending** or **scheduled** state, the mailbox is aborted immediately. An abort request while the mailbox is in **transmit** state can have two results. If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN_TSR register. If the transmission fails, the mailbox becomes **scheduled;** the transmission is aborted and becomes **empty** with TXOK cleared. In all cases the mailbox will become **empty** again at least at the end of the current transmission.

Non-automatic retransmission mode

This mode has been implemented in order to fulfil the requirement of the Time Triggered Communication option of the CAN standard. To configure the hardware in this mode the NART bit in the CAN_MCR register must be set.

In this mode, each transmission is started only once. If the first attempt fails, due to an arbitration loss or an error, the hardware will not automatically restart the message transmission.

At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP bit in the CAN_TSR register. The TXOK, ALST and TERR bits indicate the result of the transmission in the CAN_TSR register.



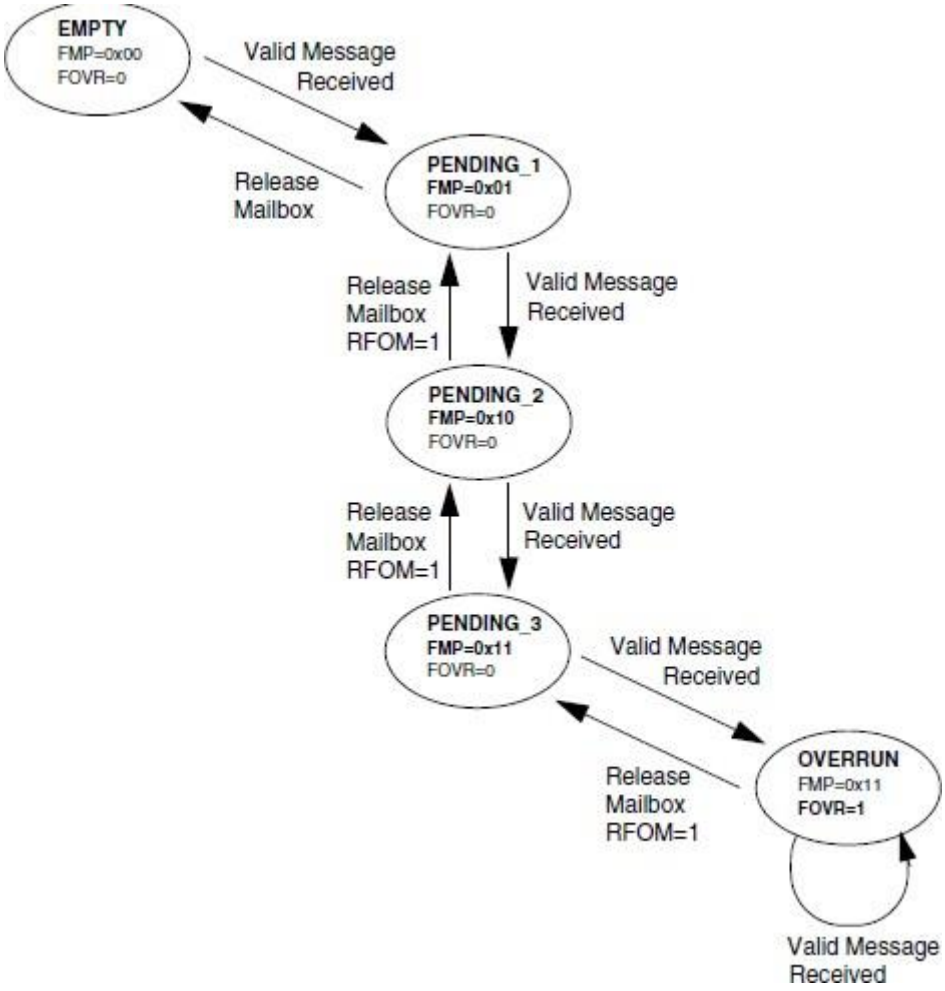
Transmit Mail Box state

Reception handling

For the reception of CAN messages, three mailboxes organized as a FIFO are provided. In order to save CPU load, simplify the software and guarantee data consistency, the FIFO is managed completely by hardware. The application accesses the messages stored in the FIFO through the FIFO output mailbox.

Valid message

A received message is considered as valid **when** it has been received correctly according tothe CAN protocol (no error until the last but one bit of the EOF field) **and** It passed throughthe identifier filtering successfully.



Receive FIFO stage

FIFO management

Starting from the **empty** state, the first valid message received is stored in the FIFO that becomes **pending_1**. The hardware signals the event setting the FMP[1:0] bits in the CAN_RFR register to the value 01b. The message is available in the FIFO output mailbox. The software reads out the mailbox content and releases it by setting the RFOM bit in the CAN_RFR register. The FIFO becomes **empty** again. If a new valid message has been received in the meantime, the FIFO stays in **pending_1** state and the new message is available in the output mailbox.

If the application does not release the mailbox, the next valid message will be stored in the FIFO which enters **pending_2** state (FMP[1:0] = 10b). The storage process is repeated for the next valid message putting the FIFO into **pending_3** state (FMP[1:0] = 11b). At this point, the software must release the output mailbox by setting the RFOM bit, so that a mailbox is free to store the next valid message. Otherwise, the next valid message received will cause a loss of message.

Overrun

Once the FIFO is in **pending_3** state (i.e. the three mailboxes are full) the next valid message reception will lead to an **overrun** and a message will be lost. The hardware signals the overrun condition by setting the FOVR bit in the CAN_RFR register. Which message is lost depends on the configuration of the FIFO:

• If the FIFO lock function is disabled (RFLM bit in the CAN_MCR register cleared) the last message stored in the FIFO will be overwritten by the new incoming message. In this case the latest messages will be always available to the application.

• If the FIFO lock function is enabled (RFLM bit in the CAN_MCR register set) the most recent message will be discarded and the software will have the three oldest messages in the FIFO available.

Reception related interrupts

Once a message has been stored in the FIFO, the FMP[1:0] bits are updated and an interrupt request is generated if the FMPIE bit in the CAN_IER register is set. When the FIFO becomes full (i.e. a third message is stored) the FULL bit in the CAN_RFR register is set and an interrupt is generated if the FFIE bit in the CAN_IER register is set. On overrun condition, the FOVR bit is set and an interrupt is generated if the FOVIE bit in the CAN_IER register is set.

Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.

To fulfil this requirement, the bxCAN Controller provides 28 configurable and scalable filter banks (27-0) to the application. This hardware filtering saves CPU resources that would be otherwise needed to perform filtering by software. Each filter bank x consists of two 32-bitregisters, CAN_FxR0 and CAN_FxR1.

Filter bank scale and mode configuration

The filter banks are configured by means of the corresponding CAN_FMR register. To configure a filter bank it must be deactivated by clearing the FACT bit in the CAN_FAR register. The filter scale is configured by means of the corresponding FSCx bit in the CAN_FS1R register. The **identifier list** or **identifier mask** mode for the corresponding Mask/Identifier registers is configured by means of the FBMx bits in the CAN_FMR register. To filter a group of identifiers, configure the Mask/Identifier registers in mask mode. To select single identifiers, configure the Mask/Identifier registers in identifier list mode. Filters not used by the application should be left deactivated. Each filter within a filter bank is numbered (called the *Filter Number*) from 0 to a maximum dependent on the mode and the scale of each of the filter banks.

Filter match index

Once a message has been received in the FIFO it is available to the application. Typically, application data is copied into SRAM locations. To copy the data to the right location the application has to identify the data by means of the identifier. To avoid this, and to ease the access to the SRAM locations, the CAN controller provides a Filter Match Index. This index is stored in the mailbox together with the message according to the filter priority rules. Thus, each received message has its associated filter match index. The Filter Match index can be used in two ways:

• Compare the Filter Match index with a list of expected values.

• Use the Filter Match Index as an index on an array to access the data destination

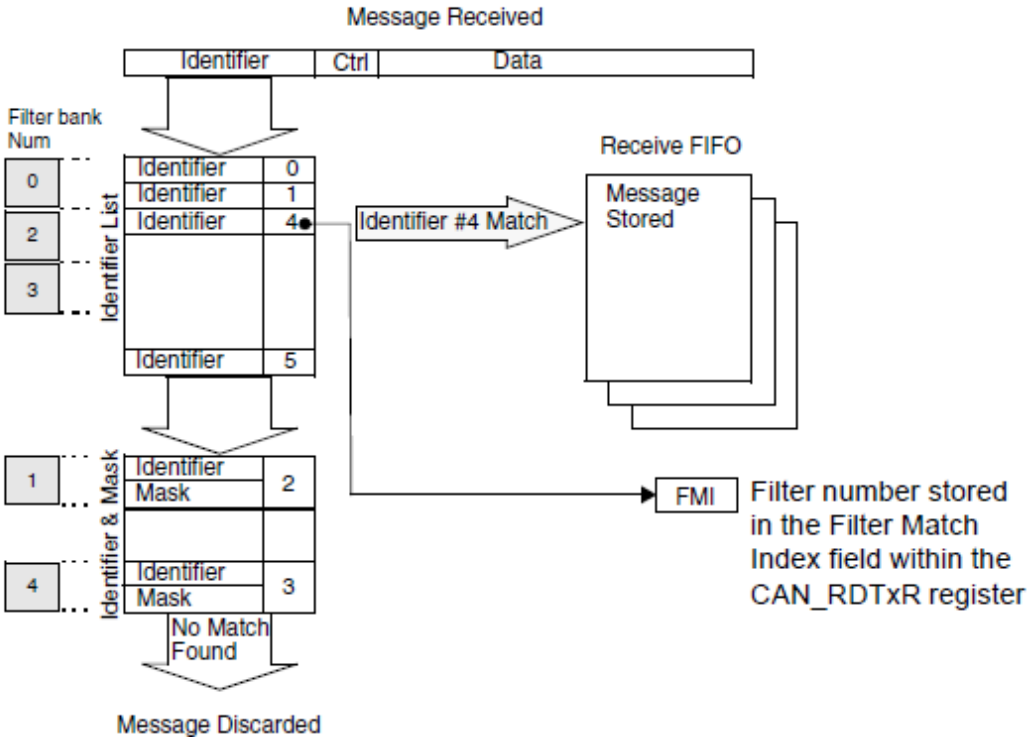
Location .For non masked filters, the software no longer has to compare the identifier. If the filter is masked the software reduces the comparison to the masked bits only .The index value of the filter number does not take into account the activation state of the filter banks. In addition, two independent numbering schemes are used, one for each FIFO.

Filter priority rules

Depending on the filter combination it may occur that an identifier passes successfully through several filters. In this case the filter match value stored in the receive mailbox is chosen according to the following priority rules:

- A 32-bit filter takes priority over a 16-bit filter.
- For filters of equal scale, priority is given to the Identifier List mode over the Identifier Mask mode
- For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority)

Example of 3 filter banks in 32-bit Unidentified List mode and the remaining in 32-bit Identifier Mask mode



Message storage

The interface between the software and the hardware for the CAN messages is implemented by means of mailboxes. A mailbox contains all information related to amessage; identifier, data, control, status and time stamp information.

Transmit mailbox

The software sets up the message to be transmitted in an empty transmit mailbox. The status of the transmission is indicated by hardware in the CAN_TSR register.

Receive mailbox

When a message has been received, it is available to the software in the FIFO output Mail box. Once the software has handled the message (e.g. read it) the software must release the FIFO output mailbox by means of the RFOM bit in the CAN_RFR register to make the next incoming message available. The filter match index is stored in the MFMI field of the CAN_RDTxR register. The 16-bit time stamp value is stored in the TIME[15:0] field of CAN_RDTxR.

Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDHxR

Transmitter mailbox mapping

Offset to receive mailbox base address (bytes)	Register name
0	CAN_RIxR
4	CAN_RDTxR
8	CAN_RDLxR
12	CAN_RDHxR

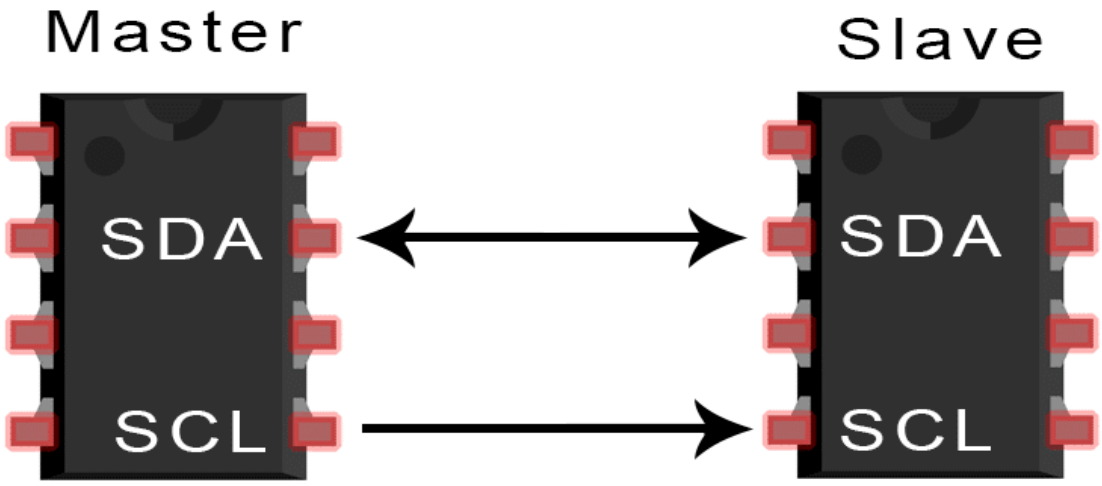
Receive mailbox mapping

The example above shows the filtering principle of the bxCAN. On reception of a message, the identifier is compared first with the filters configured in identifier list mode. If there is a match, the message is stored in the associated FIFO and the index of the matching filter is stored in the Filter Match Index. As shown in the example, the identifier matches with Identifier #2 thus the message content and FMI 2 is stored in the FIFO.

If there is no match, the incoming identifier is then compared with the filters configured in mask mode. If the identifier does not match any of the identifiers configured in the filters, the message is discarded by hardware without disturbing the software.

2.5 I2C Protocol

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address – they must still be obtained from NXP (formerly Philips semiconductors).



I2C bus is popular because it is simple to use, there can be more than one master, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.
Each slave device has a unique address. Transfer from and to master device is serial and it is split into 8-bit packets. All these simple requirements make it very simple to implement I2C interface even with cheap microcontrollers that have no special I2C hardware controller. You only need 2 free I/O pins and few simple i2C routines to send and receive commands.
The initial I2C specifications defined maximum clock frequency of 100 kHz. This was later increased to 400 kHz as Fast mode. There is also a High speed mode which can go up to 3.4 MHz and there is also a 5 MHz ultra-fast mode.

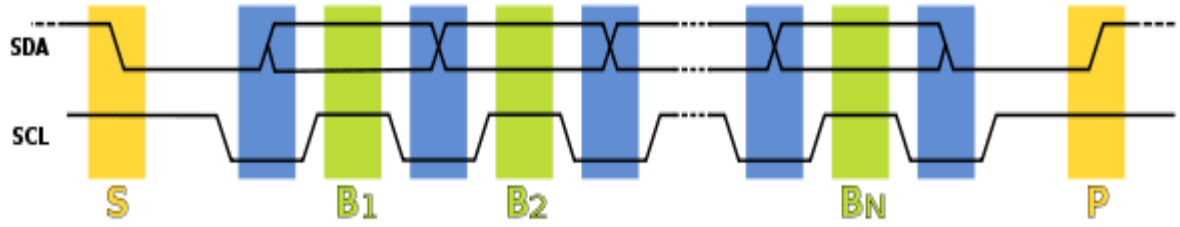
I2C Interface

I2C uses only two wires: SCL (serial clock) and SDA (serial data). Both need to be pulled up with a resistor to +Vdd. There are also I2C level shifters which can be used to connect to two I2C buses with different voltages.

I2C Addresses

Basic I2C communication is using transfers of 8 bits or bytes. Each I2C slave device has a 7-bit address that needs to be unique on the bus. Some devices have fixed I2C address while others have few address lines which determine lower bits of the I2C address. This makes it very easy to have all I2C devices on the bus with unique I2C address. There are also devices which have 10-bit address as allowed by the specification.

7-bit address represents bits 7 to 1 while bit 0 is used to signal reading from or writing to the device. If bit 0 (in the address byte) is set to 1 then the master device will read from the slave I2C device.
Master device needs no address since it generates the clock (via SCL) and addresses individual I2C slave devices.
I2C Protocol



In normal state both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition (S) followed by the address of the slave device (B1). If the bit 0 of the address byte was set to 0 the master device will write to the slave device (B2). Otherwise, the next byte will be read from the slave device. Once all bytes are read or written (Bn) the master device generates Stop condition (P). This signals to other devices on the bus that the communication has ended and another device may use the bus.
Most I2C devices support repeated start condition. This means that before the communication ends with a stop condition, master device can repeat start condition with address byte and change the mode from writing to reading.

Chapter 3

Hardware Requirements

Materials required with Specifications

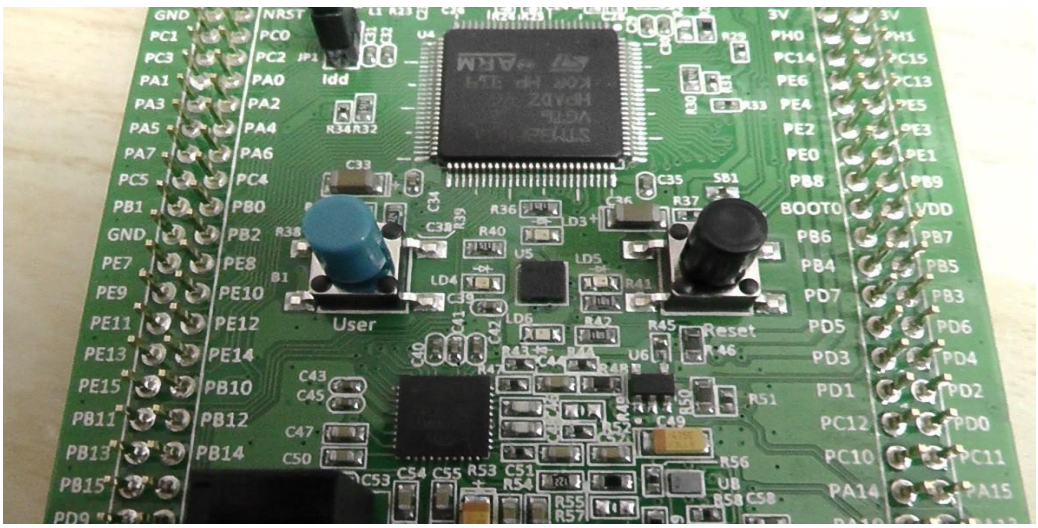
Equipment
Two STM32F4 Discovery Board
CAN Transceivers
Buzzer,LED,OLED,I2C,Ultrasonic Sensor

Table 3.1- Materials Required and Specifications

3.1 Introduction to STM32F407 Discovery Board

The STM32F407xx family is based on the high-performance ARM® Cortex®-M4 32-bit RISC core with FPU operating at a frequency of up to 72 MHz, and embedding a floating point unit (FPU), a memory protection unit (MPU) and an embedded trace macro cell (ETM). The family incorporates high-speed embedded memories (up to 1 Mbytes of Flash memory, up to 192 Kbytes of RAM) and an extensive range of enhanced I/O s and peripherals connected to two APB buses.

They also feature standard and advanced communication interfaces: up to two I2Cs, up to three SPIs (two SPIs are with multiplexed full-duplex I2Ss), three USARTs, up to two UARTs, CAN and USB. To achieve audio class accuracy, the I2S peripherals can be clocked via an external PLL. The STM32F303xB/STM32F303xC family operates in the -40 to +85 °C and -40 to +105 °C temperature ranges from a 2.0 to 3.6 V power supply. A comprehensive set of power-saving mode allows the design of low-power applications.



3.2 CAN Transceiver (SN65HVD230)

Key Features and pin functions:-

SN65HVD230 can be used at high interference environment. The device has the ability to send and receive good at different rates, its main features are as follows:
Fully compatible with ISO11898 standards
High input impedance, allowing 120 nodes
Low current standby mode, the typical current is 370μA
Signal transfer rate up to 1Mb / s
Thermal protection, open fail-safe function
With anti-moment interference protection function bus
Slope control, reduce radio frequency interference (RFI)
Differential receiver with a wide range of anti-common mode interference, electromagnetic interference (EMI) capabilities.

Operating mode and control logic:-

SN65HVD230 high speed, slope and wait three different modes of operation. Its mode of control can be achieved by control pin Figure 2 is a SN65HVD230 CAN bus system in a typical application diagram. Can be seen from the figure, CAN controller is connected to the output pin Tx SN65HVD230 data input terminal D, this CAN node can send data transferred to the CAN network; CAN controller and receive pins of Rx and SN65HVD230 R data output terminal coupled for receiving data. Options SN65HVD230 Rs port via jumpers and slope resistance grounded at one end, a way can be achieved through hardware choose 3 modes of operation, in which the slope is 0 ~ 100kΩ resistor potentiometer.

3.3 Ultrasonic Sensor HCSR04

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning. Test distance = (high level time × velocity of sound (340M/S)/2.

Wire connecting direct as following:

- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground
- 5V Supply

Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
Measuring Angle	15 degree
Trigger Input Signal	10Us TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

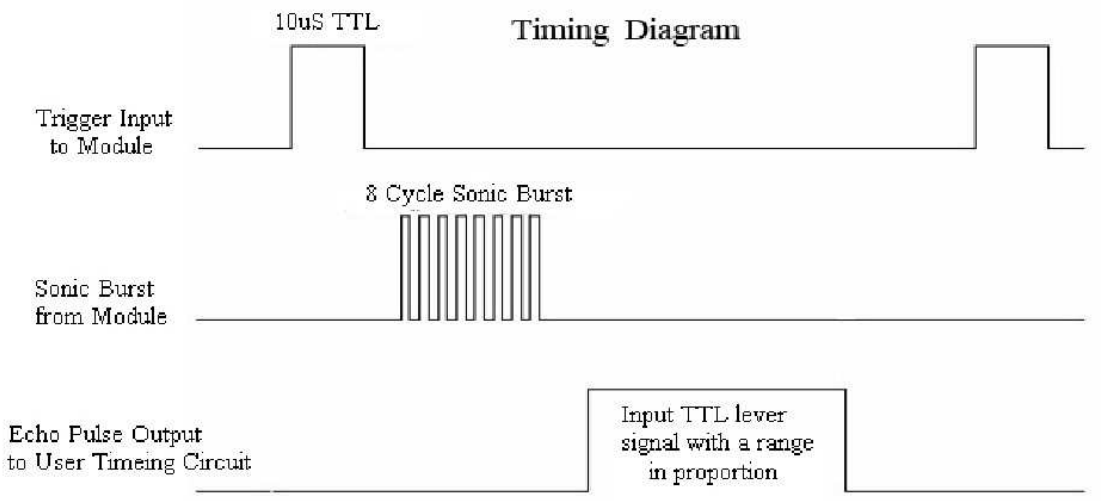
Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Formula: us / 58 = centimeters or us / 148 =inch; or:

The range = high level time * velocity (340M/S) / 2;

we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



3.4 OLED

OLED (Organic Light Emitting Diodes) is a flat light emitting technology, made by placing a series of organic thin films between two conductors. When electrical current is applied, a bright light is emitted. OLEDs are emissive displays that do not require a backlight and so are thinner and more efficient than LCD displays (which do require a white backlight).
OLED displays are not just thin and efficient - they provide the best image quality ever and they can also be made transparent, flexible, foldable and even rollable and stretchable in the future. OLEDs represent the future of display technology

How do OLEDs work?

An OLED is made by placing a series of organic thin films between two conductors. When electrical current is applied, a bright light is emitted.

So what's organic about OLEDs?
OLEDs are organic because they are made from carbon and hydrogen. There's no connection to organic food or farming - although OLEDs are very efficient and do not contain any bad metals - so it's a real green technology.

3.5 LED and Buzzer

LED, in full **light-emitting diode**, in electronics, a semiconductor device that emits infrared or visible light when charged with an electric current. Visible LEDs are used in many electronic devices as indicator lamps, in automobiles as rear-window and brake lights, and on billboards and signs as alphanumeric displays or even full-colour posters. Infrared LEDs are employed in autofocus cameras and television remote controls and also as light sources in fibre-optic telecommunication systems.

The familiar lightbulb gives off light through incandescence, a phenomenon in which the heating of a wire filament by an electric current causes the wire to emit photons, the basic energy packets of light. LEDs operate by electroluminescence, a phenomenon in which the emission of photons is caused by electronic excitation of a material. The material used most often in LEDs is gallium arsenide, though there are many variations on this basic compound, such as aluminum gallium arsenide or aluminum gallium indium phosphide.

LED emission is generally in the visible part of the spectrum (i.e., with wavelengths from 0.4 to 0.7 micrometre) or in the near infrared (with wavelengths between 0.7 and 2.0 micrometres). The brightness of the light observed from an LED depends on the power emitted by the LED and on the relative sensitivity of the eye at the emitted wavelength. Maximum sensitivity occurs at 0.555 micrometre, which is in the yellow-orange and green region. The applied voltage in most LEDs is quite low, in the region of 2.0 volts; the current depends on the application and ranges from a few milliamperes to several hundred milliamperes.

Buzzer

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke.

Chapter 4

Software Requirements

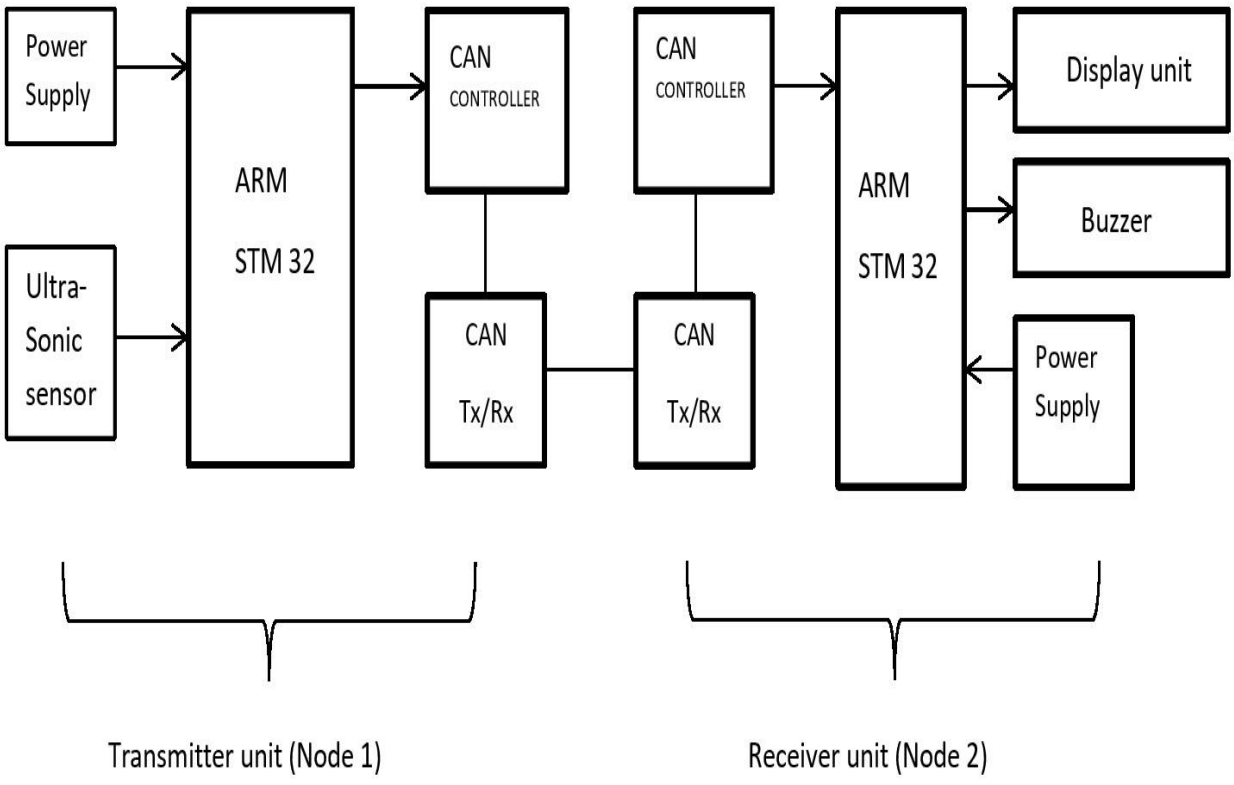
5.1 Keil

Keil development tools support every level of software developer from the professional applications engineer to the student just learning about embedded software development. The Keil Development Tools are designed to solve the complex problems facing embedded software developers. Numerous example programs are included to help you get started with the most popular embedded 8051 devices. The Keil µVision Debugger accurately simulates on-chip peripherals (I C, CAN, UART, SPI, Interrupts, I/O Ports, A/D Converter, D/A Converter, and PWM Modules) of your 8051 device. Simulation helps you understand hardware configurations and avoids time wasted on setup problems. Additionally, with simulation, you can write and test applications before target hardware is available. When you are ready to begin testing your software application with target hardware, use the MON51, MON390, MONADI, or FlashMON51 Target Monitors, the ISD51 In-System Debugger, or the ULINK USB-JTAG Adapter to download and test program code on your target system.

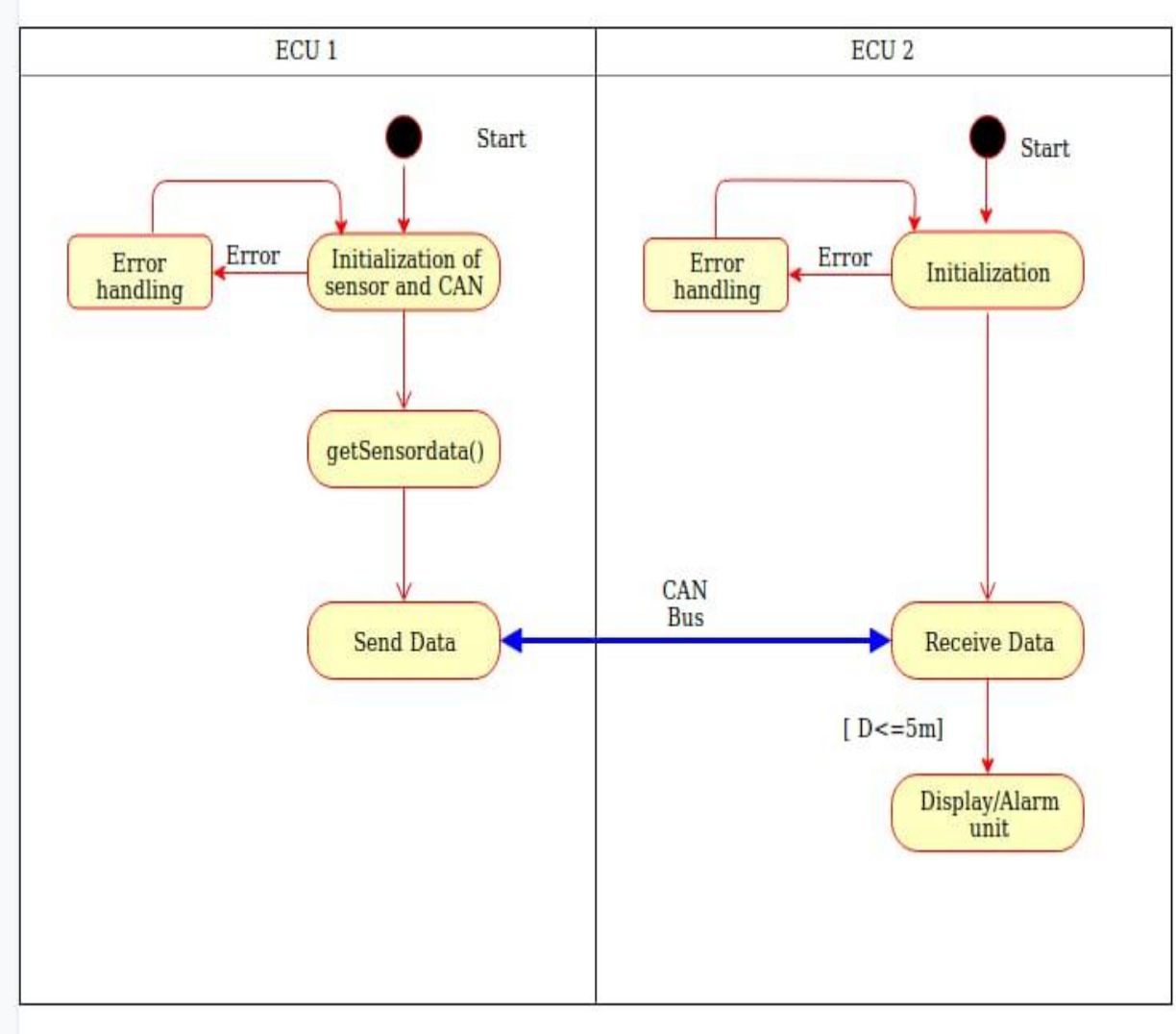
Chapter 5

Block Diagram and State Diagram

5.1 Block Diagram



5.2 State Diagram



5.3 Working

The distance between two moving cars/ car and stationary object is measured by ultrasonic system. Trigger pin is set high for 10us, then whenever sound get reflected from the object in front of car, the echo pin becomes high and in meantime the value of counter of hardware timer is calculated for every 1 us. This time (**Time of flight**) is used to find the distance between the two car, by using following formula

S=d/t

This distance is divided by 2 due to Doppler Effect.

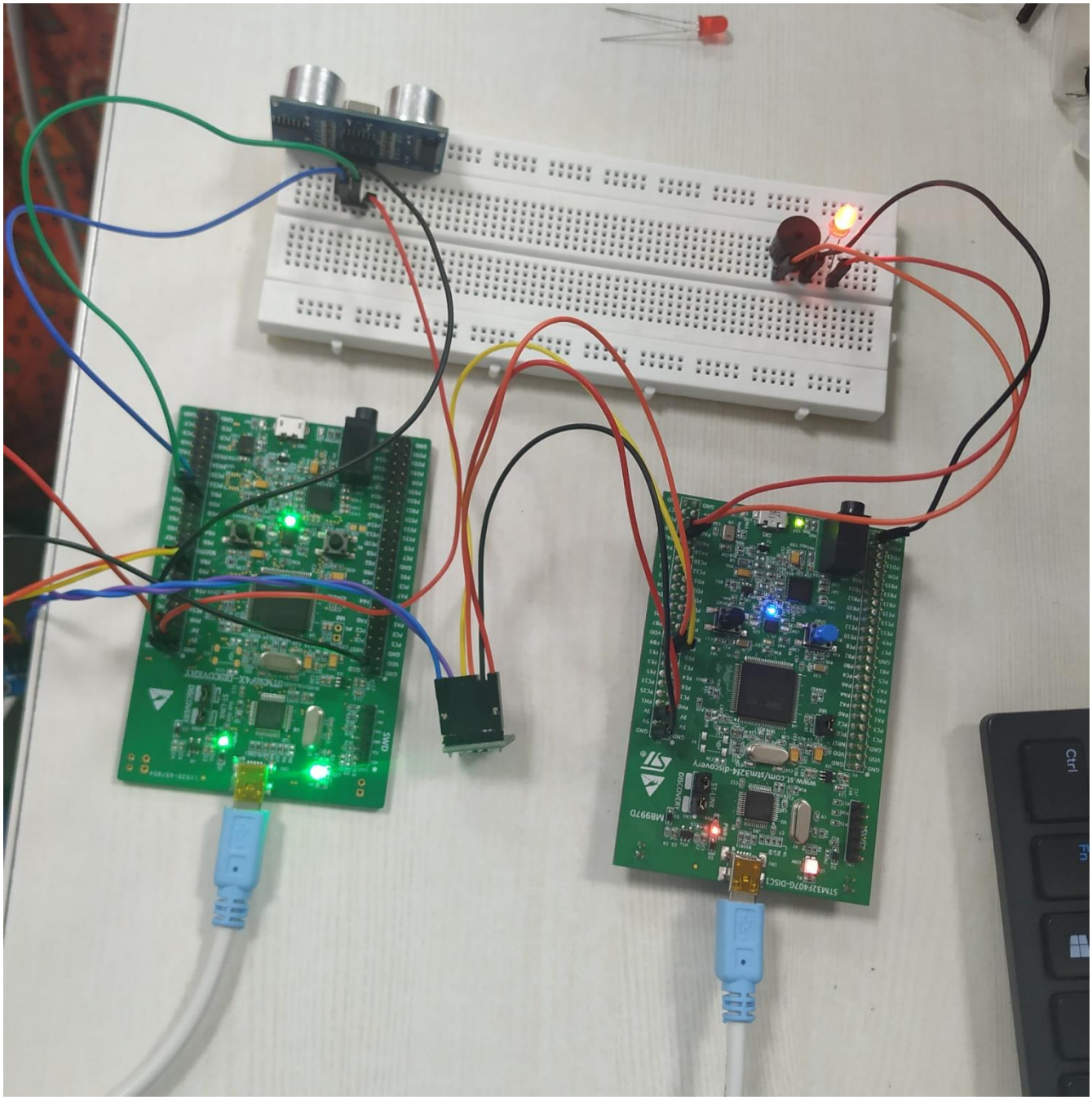
Where, speed of sound is 340 m/s

D= (0.034*t)/2 cm

The distance is calculated in node 1 (ECU 1) through ultrasonic and continuously sent to node 2 (ECU 2) through CAN transciever.

Node 2

This distance is continuously displayed in OLED using I2C bus. Whenever the distance between car and object is less than **3 meter** practically (in our demo it is 10 cm). The buzzer starts beeping and **red LED** starts glowing, So that driver get alert and apply the break.



Chapter 6

Conclusion

FCW systems use sensors to detect slower-moving or stationary vehicles. When the distance between vehicles becomes so short that a crash is imminent, a signal alerts the driver so that the driver can apply the brakes or take evasive action, such as steering, to prevent a potential crash. Vehicles with this technology provide drivers with an audible alert (Buzzer), a visual display (LED), or other warning signals. This project can be implemented in ADAS (Adaptive Driver Assistance System), which is future of automatic car system. Which is going to be a new age of car. Once an impending collision is detected, these systems provide a warning to the driver. When the collision becomes imminent, they take action autonomously without any driver input (by braking or steering or both). Collision avoidance by braking is appropriate at low vehicle speeds (e.g. below 50 km/h (31 mph)), while collision avoidance by steering may be more appropriate at higher vehicle speeds if lanes are clear. Cars with collision avoidance may also be equipped with adaptive cruise control, using the same forward-looking sensors.

Chapter 7
References

- https://elearning.vector.com/vl_can_introduction_en.html
- Renesas CAN pdf
- Serial Bus System pdf
- CAN primer
- STM32F407 Discovery User Manual
- https://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php
- mydigitallife.com: Toyota Develops Automatic Brake System Assisted by GPS Technology for Safety Driving. http://www.mydigitallife.info/2008/02/13/toyota-develops-automatic-brake-system-assisted-by-gps-technology-for-safety-driving/