Chapter 4

Controller Area Network (CAN)

4.1 Controller Area Network (CAN) Standard

4.1.1 Introduction:

What is CAN?

The Controller Area Network (CAN) is a serial communications protocol that efficiently supports distributed real-time control with a very high level of data integrity. The CAN bus is used to interconnect a network of electronic nodes or modules.

Typically it uses a twisted pair of Wires for communication.

Example: In automotive electronics, engine control units, sensors and anti-skid systems may be connected using CAN, with bit-rates up to 1 Mbit/s. At the same time, it is cost effective to build

History:

It is first introduced in in February of 1986 by Robert Bosch GmbH. It was developed because existing serial buses in the early 1980s were not able to fulfill all the requirements to be used in passenger cars. Then Intel released the first CAN controller chip in 1987 followed by the CAN ISO standard which was published in November 1993.

Basic Features of CAN Protocol

CAN Protocol

- Uses a Differential twisted pair of wires i.e a two wired communication with names as CANH and CANL.
- Messages from different nodes communicate with message Identification with each other. The highest priority message gets through.
- The priority of message is its identifier. The message with least ID number has the highest priority. Always 0 has the highest priority.
- It does not work with Master or Slave concept. It is a Peer to Peer communication protocol.
- All nodes see all messages on network except their own…..
- CAN protocol operates with maximum of 1Mbps speed and the least is 10Kbps.
    - 125, 250 & 500 Kbps common.
    - Longer cable runs mean slower frequency.
    - Better controllers and transceivers make higher speeds easier.
    - Can’t change speed dynamically…ever !
- Logic 0 is Dominant bit and Logic 1 is recessive bit.

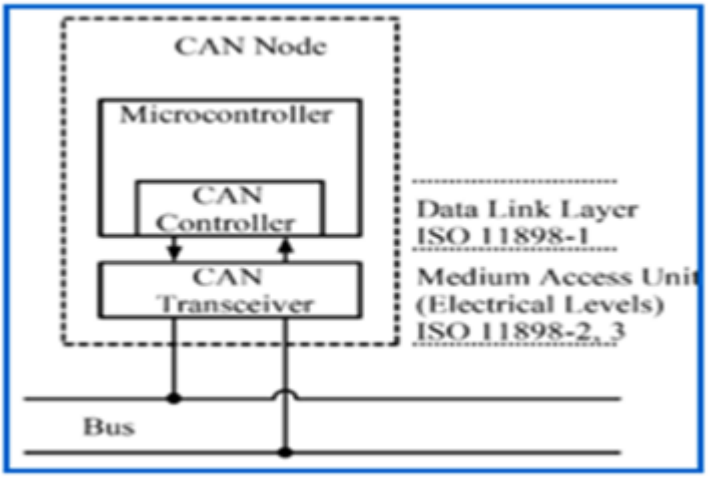
CAN Protocol Specifications

- CAN Specification 2.0 describe the base frame format (using 11-bit CAN-identifier) and the extended frame format (using 29-bit CAN-identifier).
- A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A
- A CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.

Different Levels in CAN Communication Protocol

CAN Controller has three levels

i) Central Processing Unit
ii) CAN Controller
iii) Transceiver



i) **Central processing unit** : It consists of a Microprocessor, or Host processor
Role of Central Processing Unit:
- The host processor decides what the received messages mean and what messages it wants to transmit.
- Sensors, actuators and control devices can be connected to the host processor using appropriate interfacing circuit.

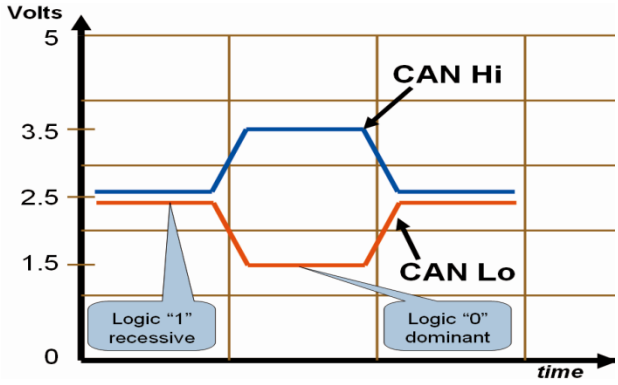
ii) **CAN controller**: It is often an integral part of the microcontroller
Role of CAN Controller:
- **Receiving:** The CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the host processor (usually by the CAN controller triggering an interrupt).
- **Sending:** The host processor sends the transmit message(s) to a CAN controller, which transmits the bits serially onto the bus when the bus is free.

iii) **Transceiver**: Defined by ISO 11898-2/3 Medium Access Unit [MAU] standards

- **Receiving:** It converts the data stream from CAN bus levels to levels that the CAN controller uses.
- It usually has protective circuitry to protect the CAN controller.
- **Transmitting:** It converts the data stream from the CAN controller to CAN bus levels.

CAN Logic Level: A differential Signal

- The transceiver converts TTL logic to CAN Logic. If the bit to be transmitted is Recessive i.e., Logic 1 = 0V in TTL Logic is converted to CAN Logic in which 2.5 volts is available on both CANH and CANL. Difference is 2.5 – 2.5 = 0 volts.
  Dominant i.e Logic 0 = Vcc(3.3V) in TTL Logic is converted to CAN logic in which CANH is at 3.5V and CANL is at 1.5V . Difference is 3.5 – 1.5 = 2.0 volts.



- The signal is a differential signal on CAN H and CANL which leads to cancellation of Noise.

4.2 Message Transfer:

Transmitter/Receiver
Transmitter
A node is said to be Transmitter if it is originating a message and it continues to be Transmitter as long as bus is idle or the node loses Arbitration.
Receiver
A node is said to be Receiver if it is not the TRANSMITTER of that message, and the bus is not idle.
All nodes other than the current Transmitter are Receiver and all nodes can see all messages on network except their own.

Frames and Its Format
CAN standard has two frame format standards as below

- CAN Specification 2.0 describe the base frame format (using 11-bit CAN-identifier) and the extended frame format (using 29-bit CAN-identifier).
- A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A
- A CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.

Message transfer is manifested and controlled by four different frame types:

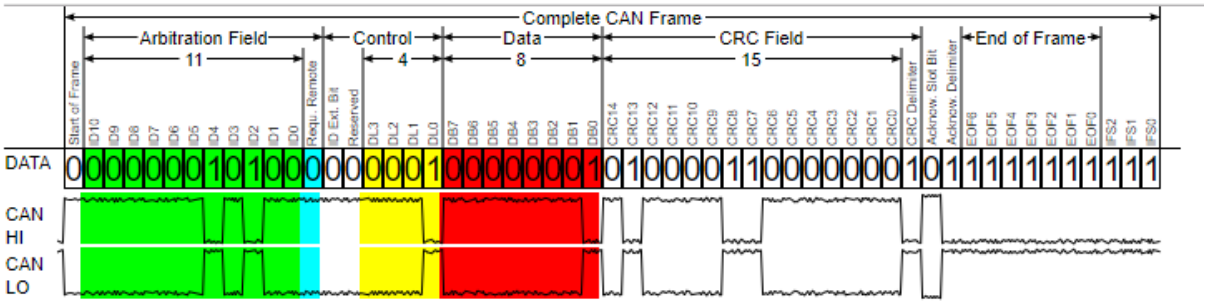
i) A Data frame – This Frame carries data from a Transmitter to the Receivers.
ii) A Remote frame - This is transmitted by a bus node to request the transmission of the Data frame with the same Identifier.
iii) An Error frame - This is transmitted by any node on detecting a bus error.
iv) An Overload frame- This used to provide for an extra delay between the preceding and the succeeding Data or Remote frames.

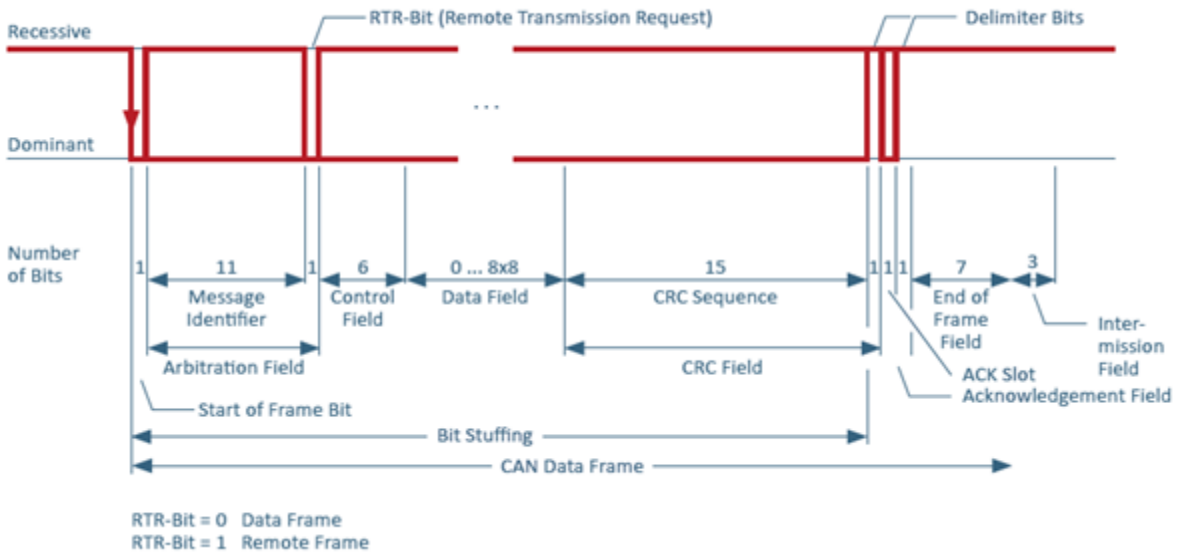
A) Data Frame Formats

Data frame consists of following fields

i) Start of frame (SOF).
ii) Arbitration.
iii) Control.
iv) Data.
v) Cyclical Redundancy Check (CRC).
vi) Acknowledge (ACK)
vii) End of frame (EOF).

Standard Data Frame Format





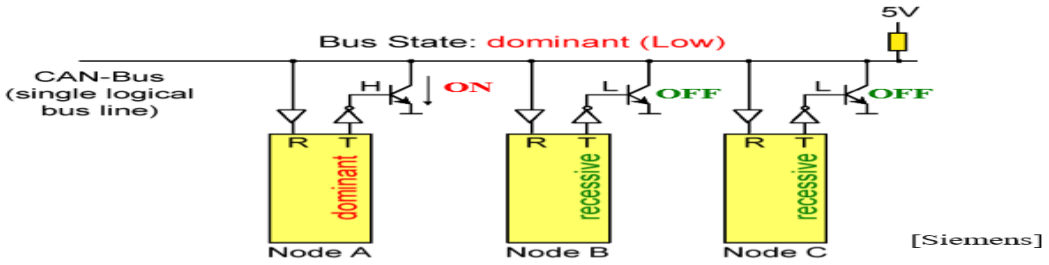
i) **Start of Frame (SOF) [1 bit]:** The SOF field consists of one dominant bit. All network nodes waiting to transmit synchronize with the SOF and begin transmitting at the same time.

ii) **Arbitration Field [11 bit] :** An arbitration scheme determines which of the nodes attempting to transmit will actually control the bus.

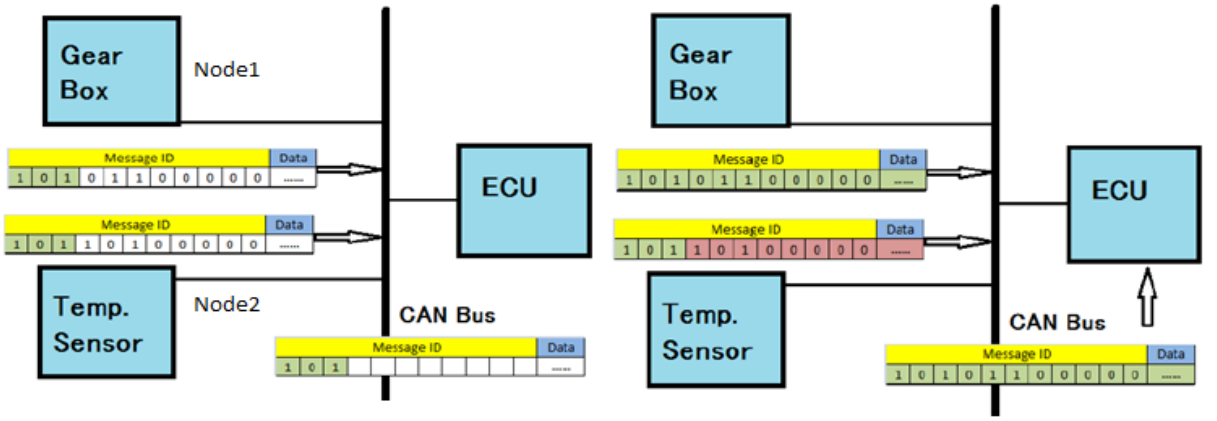
Bus Arbitration

Whenever the bus is free, any node may start to transmit a message. If two or more nodes 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 on the bus. If these levels are equal the node may continue to send. When a recessive level is sent, but a dominant level is monitored, the node has lost arbitration and must withdraw without sending any further bits.

	Dominant 0	Recessive 1
Dominant 0	0	0
Recessive 1	0	1

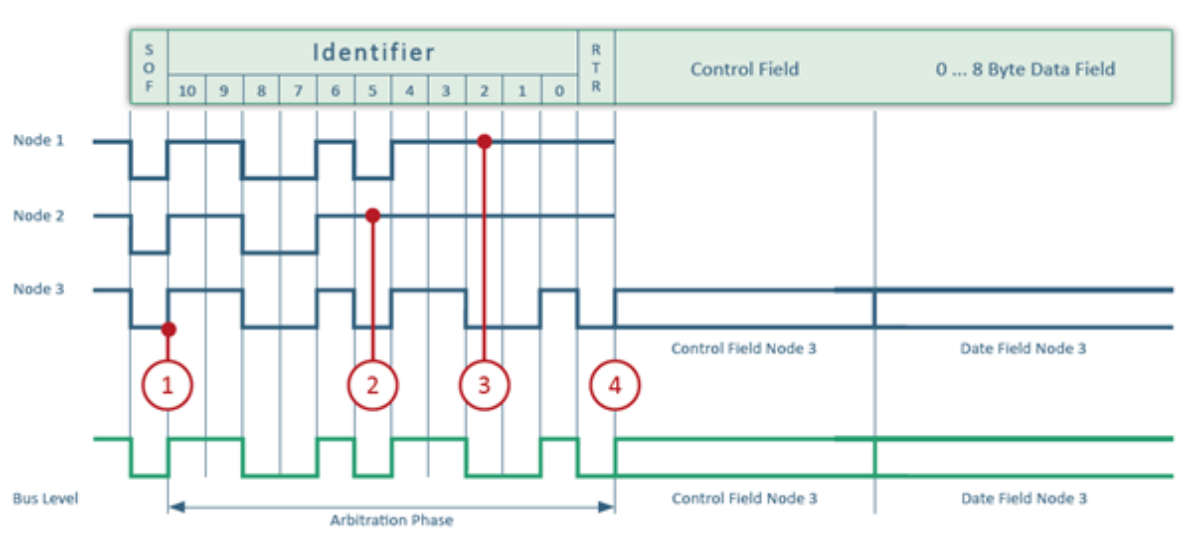


In below example shows how bus arbitration works on buses



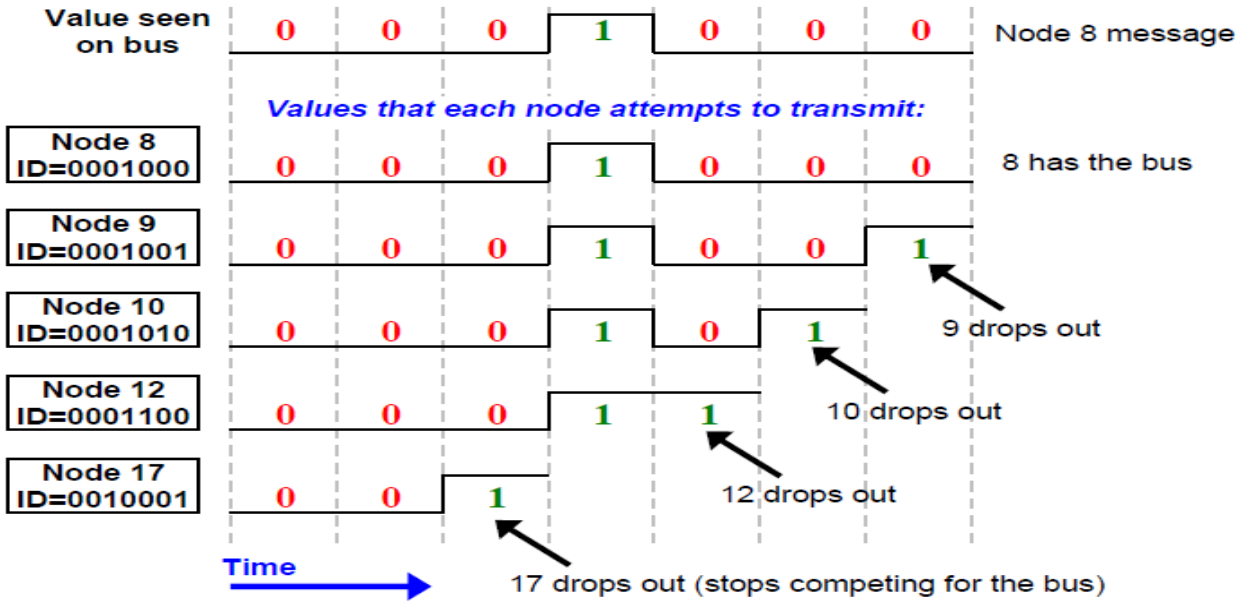
Node1 wins bus arbitration as it has dominanace when 4th bit of message ID is transmitted. Hence Node2 stops and the Node1 takes over the CAN bus

Example1:



Node 1, 2 and 3 start an arbitration process at the very same time. At time 2 node 2 detects that the bus doesn't have the recessive level it has sent and completes its arbitration process and withdraws from bus arbitration process. At time 3 node 1 withdraws. At time 4 (end of the arbitration process) nodes 3 takes control over CAN bus and keeps transmitting the frame and other nodes start receiving the frame sent by Node3.

Example2:



The above example indicates that Node8 wins the bus arbitration as it as the least message Identifier 0001000.

We can see that as long as the every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal the node may continue to send.

Example: In the first two bits all the nodes have same transmit level as a result all continue to transmit. When the 3rd bit is transmitted, it is found that the Node17 loose bus as it has

recessive logic in the ID but, all other node continues as they have dominant bits. In the next cycle all the nodes have recessive bits as a result all continue to transmit as they have the same level as the logic level on bus. In next successive bits transmission we find that at Node12, Node10 and Node 9 successively drop out and finally Node8 wins the bus arbitration and continues to transmit the message.

Remote transmission request (RTR) [1 bit]– This bit defines whether the frame is Data Frame or Remote frame
RTR=0 – Dominant in Data Frame.
RTR=1 – Recessive in Remote Frame.

Identifier extension bit (IDE)[1 bit] – This bit indicates whether the identifier used is 11 or 29 bit identification
IDE=0 – Indicates 11 bit identification is used.
IDE=1 – Indicates 29 bit identification is used.

Reserved bit (r0) [1 bit] – Set as dominant bit (0).

Data length code (DLC) [4 bit] – It indicates the size of data length being transmitted/ received . It is 4 bit field (0000 to 1000) indicating the length of data as (0-8) bytes

Ex: DLC = 0011 (3) – Three bytes of data is transmitted/ received which must be stores in TDA if getting transmitted. If received it will be in RDA.
Note: If the length of data is more than 4 bytes then the byte[4-8]can be stored in TDB/ RDB based on transmitted / received.

Data Field (DF) [64 bits = 8 bytes] – This field holds the data being transmitted. It can be maximum of 8 bytes length. Data to be transmitted (length in bytes dictated by DLC field).

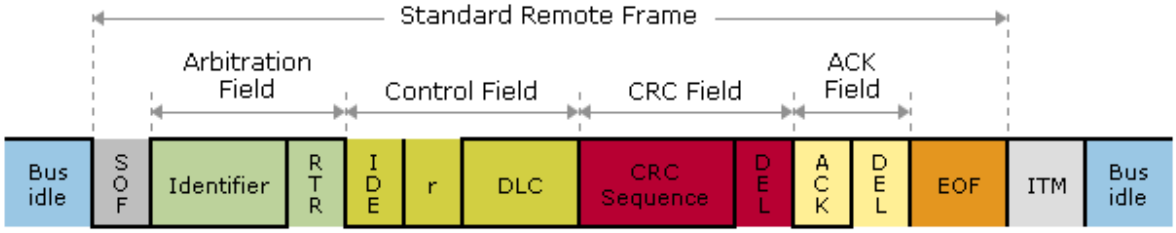
Cyclic Redundancy Code (CRC)[15 bits] -

CAN data frames and remote frames contain a safeguard based on a CRC polynomial: The transmitter calculates a check sum from the transmitted bits and provides the result within the frame in the CRC field. **The receivers use the same polynomial to calculate the check sum from the bits as seen on the bus-lines.** The self-calculated check sum is compared with the received one. If it matches, the frame is regarded as correctly received and the receiving node transmits a dominant state in the ACK slot bit, overwriting the recessive state of the transmitter. **In case of a mismatch, the receiving node sends an Error Frame after the ACK delimiter.**

In Classical CAN, a 15-bit CRC polynomial is used ($x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + x^0$).

Note: Refer to Error Checking

B) Remote Frame:



Remote Frame is a frame type used to request data, i.e. data frames, from any CAN node. Nonetheless, these frames are hardly ever used in automotive applications, since data transmission is not based on requesting, rather it is primarily based on the self-initiative of information producers. Remote frames may be transmitted in either standard or extended format.

Except for the lack of a data field, the layout of a remote frame identical to that of a data frame. Data and remote frames are differentiated by the **RTR bit** (Remote Transmission Request). In the case of a data frame, the RTR bit is sent as dominant. A remote frame is identified by a recessive RTR bit.

C) The 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.

D) The 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.

4.3 Error Checking

The robustness of CAN may be attributed in part to its abundant error-checking procedures. The CAN protocol incorporates five methods of error checking:

i) Three at the message level and

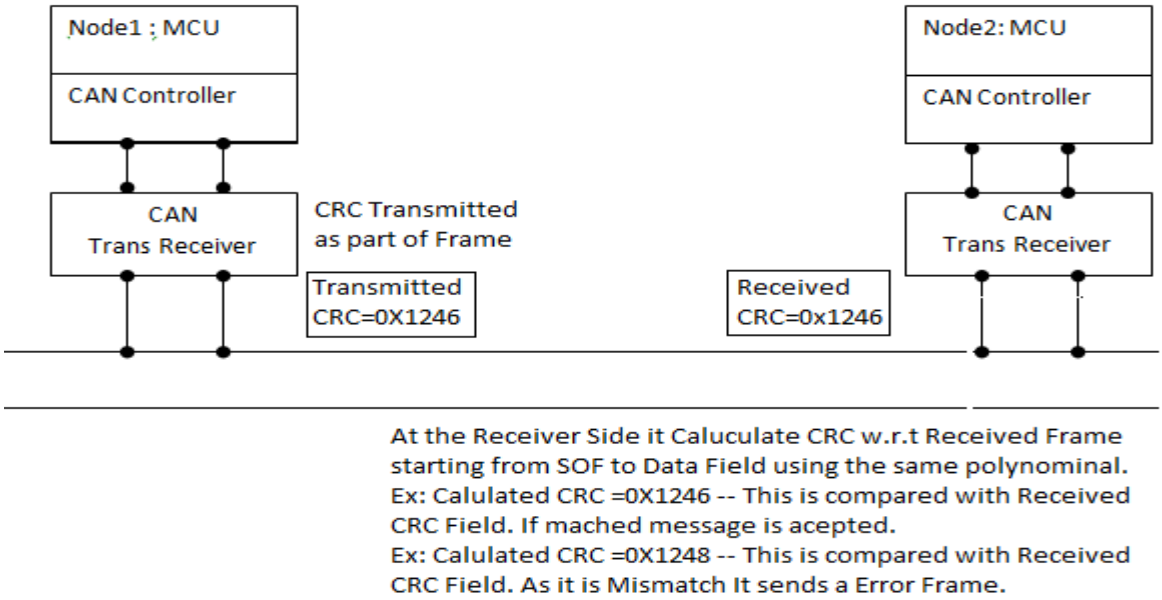
ii) Two at the bit level.

If a message fails any one of these error detection methods, it is not accepted and an error frame is generated from the receiving node. This forces the transmitting node to resend the message until it is received correctly. However, if a faulty node hangs up a bus by continuously repeating an error, its transmit capability is removed by its controller after an error limit is reached.

1. Message Level Error Checking

Message Level Error checking is done by 16 bit CRC field which has 15 bit CRC Code and 1 bit delimiter and 2 bits ACK field in which 1 bit acknowledgment and 1 bit acknowledgment delimiter.

a) Concept of CRC Checking at the Receiver:



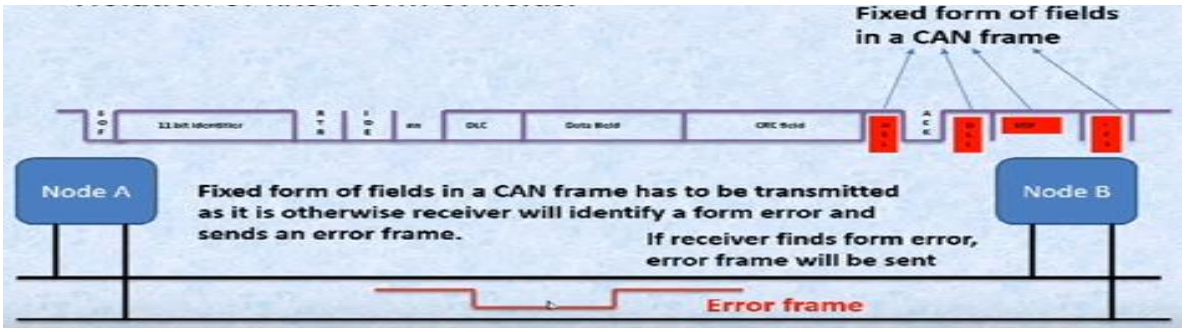
b) ACK ERROR:

For any transmitted Node the Frame must be acknowledged by at least one node otherwise ACK error will be reported in CAN network.

When Transmitter transmits the Frame it transmits the Frame with ACK as a recessive bit (1). When receiver receives the ACK bit which was recessive will be changed to dominant and sent back to Transmitter.

If it the Transmitter finds None of the nodes then ACK bit which was recessive will be sent back to Transmitter as recessive (1). Indicating a ACK Error.

c) FRAME FORMAT Error:



In a Fixed form CRC Delimiter bit, ACK Delimiter and End of Frame bits are Recessive at the time of transmission. If at the receiver any of these bits have changed from recessive to dominant then it will send the Error Frame

2. Bit Level Error Checking

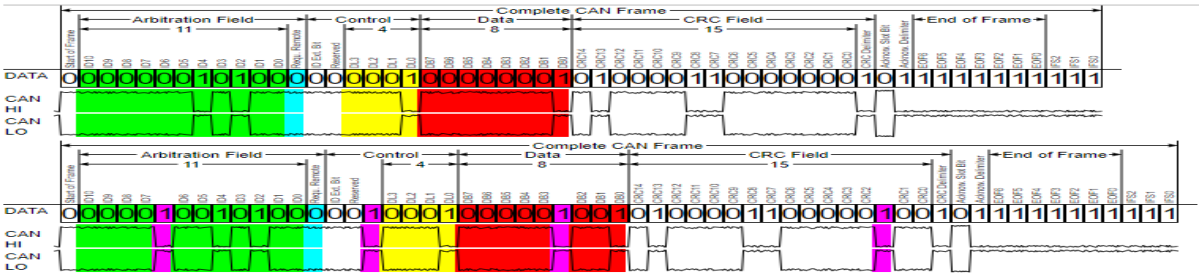
There are two bit level error checking

i) Checking of each bit Transmitted by Transmitter

In this , each bit transmitted is monitored by the transmitter of the message. If a data bit (not arbitration bit) is written onto the bus and its opposite is read, an error is generated. The only exceptions to this are with the message identifier field which is used for arbitration, and the acknowledge slot which requires a recessive bit to be overwritten by a dominant bit.

ii) Bit Stuffing: The final method of error detection is with the bit-stuffing rule where after five consecutive bits of the same logic level, if the next bit is not a complement, an error is generated. Stuffing ensures that rising edges are available for on-going synchronization of the network. Stuffing also ensures that a stream of bits are not mistaken for an error frame, or the seven-bit interframe space that signifies the end of a message. Stuffed bits are removed by a receiving node's controller before the data is forwarded to the application.

Concept of Bit stuffing



3. Timing Concepts of CAN Protocol

3.1 How CAN node synchronize with each other?

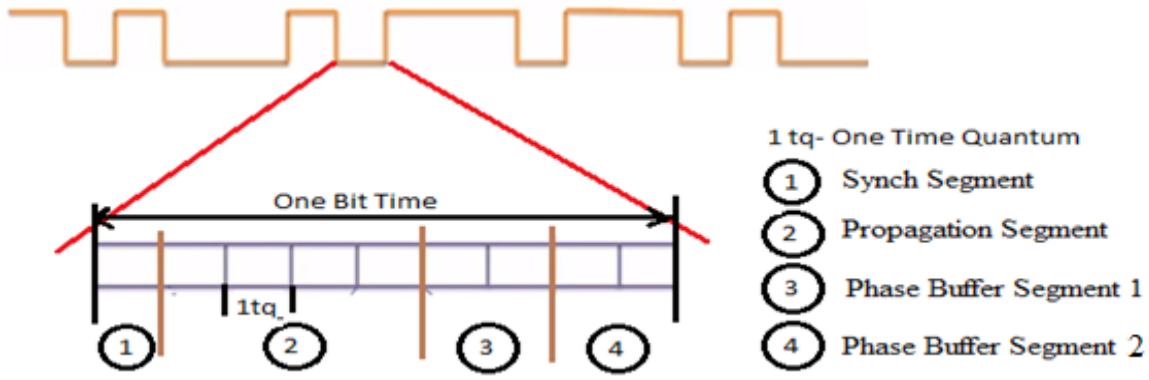
3.1.1 Bit Time: Bit time is the time taken to Transmit single bit (either it may be recessive or dominant)

Bit Time is divided into 4 segments in CAN protocol

1. Synch Segment
2. Propagation Segment
3. Phase Buffer Segment 1
4. Phase Buffer Segment 2

All these segments are divided into fixed number of Time quantum (tq). Quanta is fixed unit of time derived from oscillator period

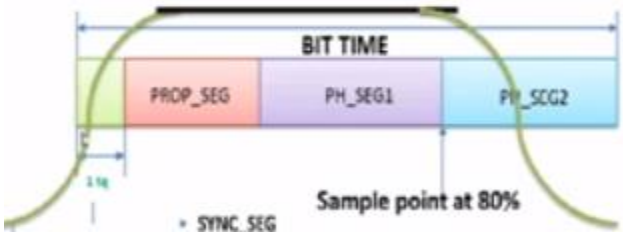
4.4 Bit Construction:



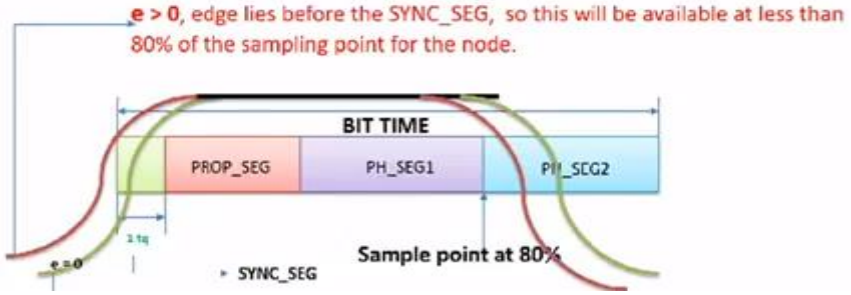
Bit is segregated into Multiple segments are Synch, Propagation, Phase Buffer1 and Phase Buffer2 segments. Each segment is further segregated into fixed number of time quantum.

i) **Synchronization Segment (SYNC_SEG) :** In order to provide synchronization of Transmitter and Receiver, the Transmitted bit is edge is expected to occur in SYNC_SEG field for proper interpretation of bit by receiver.

ii) **What if the edge won't occur in SYNC_SEG field?**
It will lead to edge phase error i.e., position of the edge relative to SYNC_SEG.
Case1: Edge phase Error (e) =0 – When edge lies within the SYNC_SEG , it will available at 80% of sampling rate.

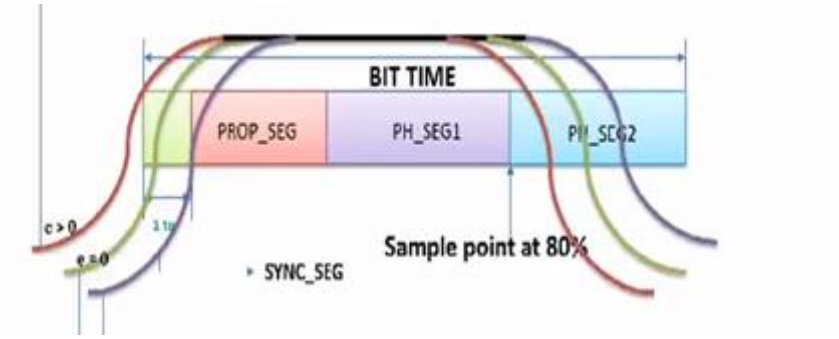


Case2: Edge phase Error (e) > 0 – It means edge lies before the SYNC_SEG. Therefore the transmitted bit will be available at less than 80% of Sample point for the receiver.



This error (e) >0 can be compensated by lengthening the phase buffer Segment 1 (time slot).
How Receiver will come to know what value it should be compensated?

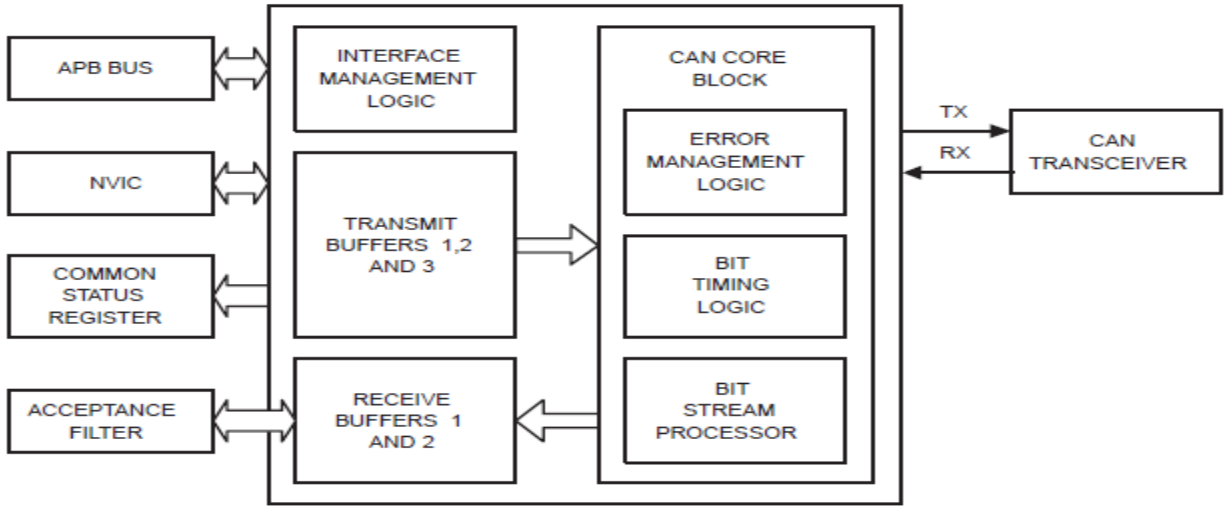
Case3: Edge phase Error (e) < 0 – It means edge lies after the SYNC_SEG. Therefore the transmitted bit will be available at a position beyond 80% of Sample point for the receiver.



This error (e) >0 can be compensated by reducing the phase buffer Segment 1 (time slot).

4.5 CAN Controller of LPC1768:
CAN controller architecture

The CAN Controller is a complete serial interface with both Transmit and Receive Buffers but without Acceptance Filter. CAN Identifier filtering is done for all CAN channels in a separate block (Acceptance Filter).
The CAN Controller Block includes interfaces to the following blocks:
• APB Interface
• Acceptance Filter
• Nested Vectored Interrupt Controller (NVIC)
• CAN Transceiver
• Common Status Registers

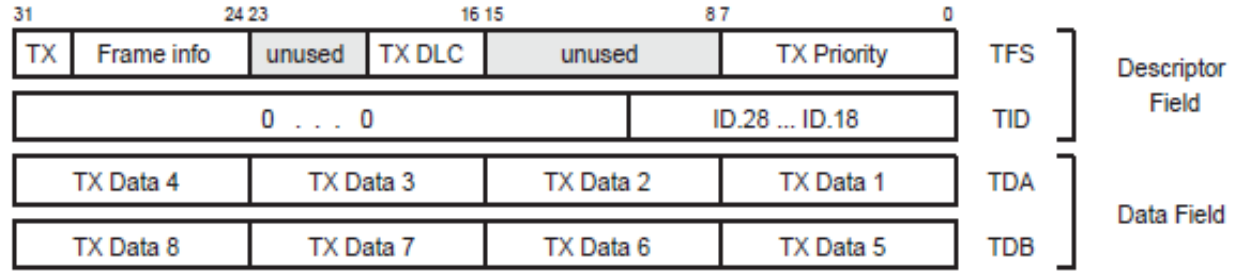


APB Interface Block (AIB)
The APB Interface Block provides access to all CAN Controller registers.

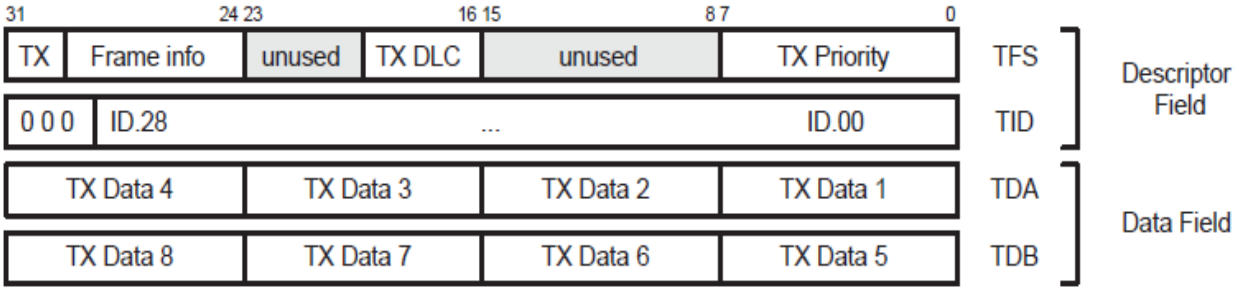
Interface Management Logic (IML)
The Interface Management Logic interprets commands from the CPU, controls internal addressing of the CAN Registers and provides interrupts and status information to the CPU.

Transmit Buffers (TXB)
The TXB represents a Triple Transmit Buffer, which is the interface between the Interface Management Logic (IML) and the Bit Stream Processor (BSP).

- Each Transmit Buffer is able to store a complete message which can be transmitted over the CAN network.



Standard Frame Format (11-bit Identifier)



Extended Frame Format (29-bit Identifier)

The CAN Controller consist of three Transmit Buffers. Each of them has a length of 4 words and is able to store one complete CAN message. Transmit buffer1 has TFS1,TID1,TDA1 and TDB1 , buffer2 has TFS2,TID2,IDA2 and TDB2 and buffer3 has TFS3,TID3,IDA3 and TDB3 ,

The buffer layout is subdivided into

1. Descriptor – TFI and TID and
2. Data Field – TDA and TDB

Depending on the chosen Frame Format, an 11-bit identifier for Standard Frame Format (SFF) or an 29-bit identifier for Extended Frame Format (EFF) follows. Note that unused bits in the TID field have to be defined as 0. The Data Field in TDA and TDB contains up to eight data bytes.

Descriptor Field

TFI- Transmit Frame Information. This Descriptor Field includes the TX Frame Info that describes the

- Frame Format,
- the Data Length and
- whether it is a Remote or Data Frame.
- In addition, a TX Priority register allows the definition of a certain priority for each transmit message.

Table 331. CAN Transmit Frame Information register (CAN1TFI[1/2/3] - address 0x4004 40[30/40/50], CAN2TFI[1/2/3] - 0x4004 80[30/40/50]) bit description

Bit	Symbol	Function	Reset Value	RM Set
7:0	PRIO	If the TPM (Transmit Priority Mode) bit in the CANxMOD register is set to 1, enabled Tx Buffers contend for the right to send their messages based on this field. The buffer with the lowest TX Priority value wins the prioritization and is sent first.		x
15:8	-	Reserved.	0	
19:16	DLC	Data Length Code. This value is sent in the DLC field of the next transmit message. In addition, if RTR = 0, this value controls the number of Data bytes sent in the next transmit message, from the CANxTDA and CANxTDB registers: 0000-0111 = 0-7 bytes 1xxx = 8 bytes	0	X
29:20	-	Reserved.	0	
30	RTR	This value is sent in the RTR bit of the next transmit message. If this bit is 0, the number of data bytes called out by the DLC field are sent from the CANxTDA and CANxTDB registers. If this bit is 1, a Remote Frame is sent, containing a request for that number of bytes.	0	X
31	FF	If this bit is 0, the next transmit message will be sent with an 11-bit Identifier (standard frame format), while if it's 1, the message will be sent with a 29-bit Identifier (extended frame format).	0	X

Transmit Priority [7:0]: Which is an 8 bit field defining the priority. It holds the Index value of message in RAM

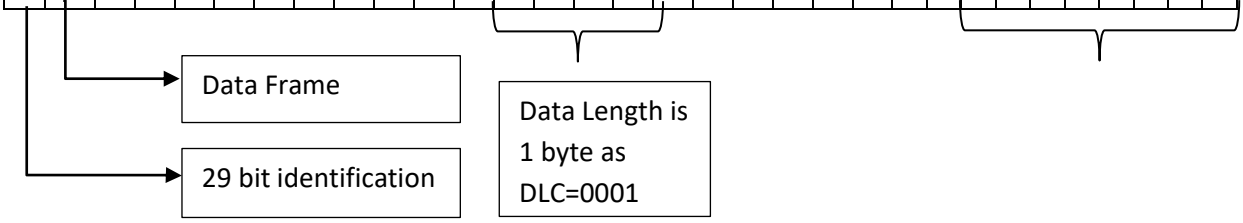
Transmit DLC [19:16]: This field holds Size of Data (message) being transmitted. It may hold values from 0000b to 1000b indicating Size as (1 to 8) bytes.

Frame Information [30]: Remote Transmit Request (RTR) – If RTR=0 – Data Frame RTR=1 – Remote frame.

FF – It indicates whether the next message transmitted is with 11 bit identification or 29 bit identification. If FF=0 – 11 bit identification If FF=1 – 29 bit identification

Example: If TFI =0X8001 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FF	RTR							Unused				TxDLC				Unused								TX Priority							
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



TID – Transmit Identification Number – Bit[10:0] holds 11 bit identification / 29 bit identification.

Ex:0x0000 0021

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			29 bit Identification [28:0]																												
0	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**

Data Field

It has TDA and TDB each holds 4 bytes of message. Therefore TDA and TDB combined will hold 8bytes = 64 bits

TDA[31:24]	TDA[23:16]	TDA[15:8]	TDA[7:0]
Byte3	Byte2	Byte1	Byte0

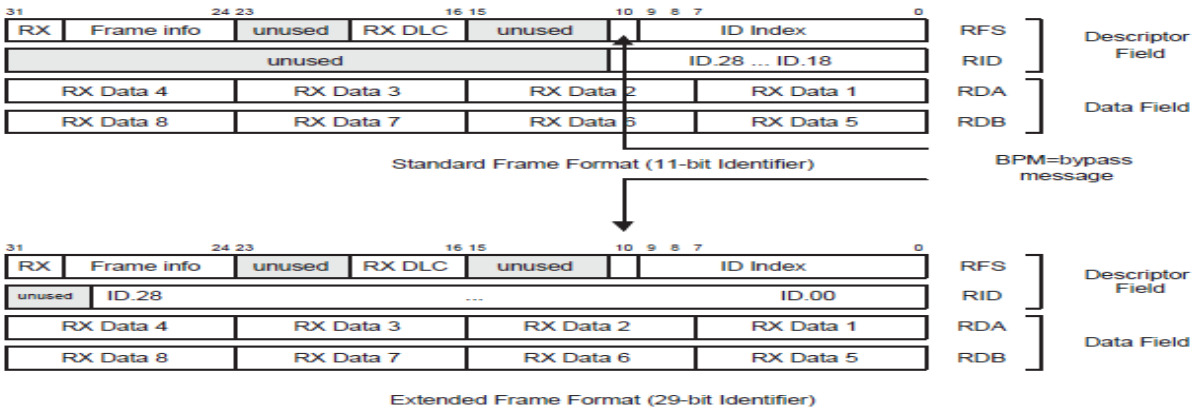
TDB[31:24]	TDB[23:16]	TDB[15:8]	TDB[7:0]
Byte7	Byte6	Byte5	Byte4

Receive Buffer (RXB)

The Receive Buffer (RXB) represents a CPU accessible Double Receive Buffer. It is located between the CAN Controller Core Block and APB Interface Block and stores all received messages from the CAN Bus line. With the help of this Double Receive Buffer concept the CPU is able to process one message while another message is being received.

The global layout of the Receive Buffer is very similar to the Transmit Buffer described earlier. Identifier, Frame Format, Remote Transmission Request bit and Data Length Code have the same meaning as described for the Transmit Buffer.

In addition, the Receive Buffer includes an ID Index field. The received Data Length Code represents the real transmitted Data Length Code, which may be greater than 8 depending on transmitting CAN node. Nevertheless, the maximum number of received data bytes is 8. This should be taken into account by reading a message from the Receive Buffer. **If there is not enough space for a new message within the Receive Buffer, the CAN Controller generates a Data Overrun condition when this message becomes valid and the acceptance test was positive. A message that is partly written into the Receive Buffer (when the Data Overrun situation occurs) is deleted.** This situation is signaled to the CPU via the Status Register and the Data Overrun Interrupt, if enabled.



All the fields of Descriptor Field are similar to transmitter field except few changes as shown in table below of RFS

Table 326. CAN Receive Frame Status register (CAN1RFS - address 0x4004 4020, CAN2RFS - address 0x4004 8020) bit description

Bit	Symbol	Function	Reset Value	RM Set
9:0	ID Index	If the BP bit (below) is 0, this value is the zero-based number of the Lookup Table RAM entry at which the Acceptance Filter matched the received Identifier. Disabled entries in the Standard tables are included in this numbering, but will not be matched. See Section 16.17 "Examples of acceptance filter tables and ID index values" on page 402 for examples of ID Index values.	0	X
10	BP	If this bit is 1, the current message was received in AF Bypass mode, and the ID Index field (above) is meaningless.	0	X
15:11	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	
19:16	DLC	The field contains the Data Length Code (DLC) field of the current received message. When RTR = 0, this is related to the number of data bytes available in the CANRDA and CANRDB registers as follows: 0000-0111 = 0 to 7 bytes1000-1111 = 8 bytes With RTR = 1, this value indicates the number of data bytes requested to be sent back, with the same encoding.	0	X
29:20	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	
30	RTR	This bit contains the Remote Transmission Request bit of the current received message. 0 indicates a Data Frame, in which (if DLC is non-zero) data can be read from the CANRDA and possibly the CANRDB registers. 1 indicates a Remote frame, in which case the DLC value identifies the number of data bytes requested to be sent using the same Identifier.	0	X
31	FF	A 0 in this bit indicates that the current received message included an 11-bit Identifier, while a 1 indicates a 29-bit Identifier. This affects the contents of the CANid register described below.	0	X

The addition bit found is BP

BP =1 – it means current message received in Acceptance Filter(AF) is in Bypass Mode and ID index is field is meaningless. Bypass mode of AF is set by AccBP bit of AFMR (Acceptance Filter Mode Register)

BP=0 –It means message received go through Acceptance filter. In this condition, Upon reception of a message, It checks whether the message ID of received message is present in the acceptance filter or not. If the received message ID is present in the acceptance filter list then message will be accepted else rejected. Upon acceptance, the Index Value of accepted message ID is copied into ID index[9:0] of RFS.

AFMR

Table 343. Acceptance Filter Mode Register (AFMR - address 0x4003 C000) bit description

Bit	Symbol	Value	Description	Reset Value
0	AccOff[2]	1	if AccBP is 0, the Acceptance Filter is not operational. All Rx messages on all CAN buses are ignored.	1
1	AccBP[1]	1	All Rx messages are accepted on enabled CAN controllers. Software must set this bit before modifying the contents of any of the registers described below, and before modifying the contents of Lookup Table RAM in any way other than setting or clearing Disable bits in Standard Identifier entries. When both this bit and AccOff are 0, the Acceptance filter operates to screen received CAN Identifiers.	0
2	eFCAN[3]	0	Software must read all messages for all enabled IDs on all enabled CAN buses, from the receiving CAN controllers.	0
		1	The Acceptance Filter itself will take care of receiving and storing messages for selected Standard ID values on selected CAN buses. See Section 16.16 "FullCAN mode" on page 391.	

[1] **Acceptance Filter Bypass Mode (AccBP):** By setting the AccBP bit in the Acceptance Filter Mode Register, the Acceptance filter is put into the Acceptance Filter Bypass mode. During bypass mode, the internal state machine of the Acceptance Filter is reset and halted. All
Received CAN messages are accepted, and acceptance filtering can be done by software.

[2] Acceptance Filter Off mode (AccOff): After power-up or hardware reset, the Acceptance filter will be in Off mode, the AccOff bit in the Acceptance filter Mode register 0 will be set to 1. The internal state machine of the acceptance filter is reset and halted.
If not in Off mode, setting the AccOff bit, either by hardware or by software, will force the acceptance filter into Off mode.
[3] FullCAN Mode Enhancements: A FullCAN mode for received CAN messages can be enabled by setting the eFCAN bit in the
Acceptance filters mode register.

Algorithm

1. Power: On reset CAN1/2 are disabled. Make sure that corresponding bits in PCONP register are set so that power is enabled.

Table 46. Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description

Bit	Symbol	Description	Reset value
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0

Bit 13 & 14 are Power Control bits for CAN1 and CAN2 respectively.

PCONP = PCONP | 0000 0000 0000 0000 0010 0000 0000 0000b

2. Peripheral clock: Select PCLK bits PCLKSEL registers for CAN1/2 and Acceptance Filter peripheral clock (PCLK_ACC). PCLK_CAN1/2 and PCLK_ACC clocks should be same.
3. Select the PINs for CAN1/2 by choosing alternate function using appropriate PINSEL registers.

Table 80. Pin function select register 0 (PINSEL0 - address 0x4002 C000) bit description

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1	00

4. Interrupts: CAN interrupts are enabled using the CAN1/2IER registers. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register

Table 322. CAN Interrupt Enable Register (CAN1IER - address 0x4004 4010, CAN2IER - address 0x4004 8010) bit description

Bit	Symbol	Function	Reset Value	RM Set
0	RIE	Receiver Interrupt Enable. When the Receive Buffer Status is 'full', the CAN Controller requests the respective interrupt.	0	X
1	TIE1	Transmit Interrupt Enable for Buffer1. When a message has been successfully transmitted out of TXB1 or Transmit Buffer 1 is accessible again (e.g. after an Abort Transmission command), the CAN Controller requests the respective interrupt.	0	X
2	EIE	Error Warning Interrupt Enable. If the Error or Bus Status change (see Status Register), the CAN Controller requests the respective interrupt.	0	X
3	DOIE	Data Overrun Interrupt Enable. If the Data Overrun Status bit is set (see Status Register), the CAN Controller requests the respective interrupt.	0	X

Bit0: RIE =1; Receiver Interrupt is enabled. When Receive Buffer is Full, the CAN controller requests the respective interrupt.
Bit1:TIE1=1; Transmit Interrupt is enabled for Buffer1. When message stored in TXB1 (Transmit Buffer1) is transmitted successfully/ accessible again, then the CAN Controller request for corresponding interrupt.

Note: Refer Table322 of reference manual [1]

5. CAN1MOD Register

It is used to change the behavior of the CAN Controller. Bits may be set or reset by the CPU that uses the Mode Register as a read/write memory.

31 8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved	TM	-	RPM	SM	TPM	STM	LOM	RM

Bit0 - Reset Mode [RM]

RM=0	CAN Controller is in Operational Mode. In this mode certain register cannot be initialized.
RM=1	CAN Controller operation is disabled. Current transmission/ reception of message is aborted. It should be used when certain certain registers need to be initialized such as CMR, GSR, IER, BTR.

Note: Once initialization of registers done it RM bit is made 0 in order make CAN controller operation so that message can be transmitted or received

Bit1 - Transmit priority Mode

TPM=0	The transmit priority for 3 Transmit Buffers depends on the CAN Identifier. The internal transmit message queue is organized such as that the transmit buffer with the lowest CAN Identifier (TID)
TPM=1	The transmit priority for 3 Transmit Buffers depends on the contents of the Tx Priority register within the Transmit Buffer.

If TPM=1

Bit	Symbol	Function	Reset Value	RM Set
7:0	PRIO	If the TPM (Transmit Priority Mode) bit in the CANxMOD register is set to 1, enabled Tx Buffers contend for the right to send their messages based on this field. The buffer with the lowest TX Priority value wins the prioritization and is sent first.		x

Other bits like LOM- Listen Only Mode ,Self-Test Mode(STM), Sleep Mode(SM), Test Mode (TM) and Receive Polarity Mode (RPM) can be configured based on requirement.
Note: For details refere Reference Manual Page No: 361 Table No:318

6. CAN Command Register (CAN1CMR)

31 8	7	6	5	4	3	2	1	0
	STB3	STB2	STB1	SRR	CD0	RRB	AT	TR

TR	STB1	STB2	STB3	Description
0	0	0	0	TR=0- No Transmission Request
1	1	0	0	Transmission Request with Tx Buffer1 The message, previously written to the CANxTFI1, CANxTID1, and optionally the CANxTDA1 and CANxTDB1 registers, is queued for transmission **from the selected Transmit Buffer1**.
1	0	1	0	Transmission Request with Tx Buffer2 The message, previously written to the CANxTFI2, CANxTID2, and optionally the CANxTDA2 and CANxTDB2 registers, is queued for transmission **from the selected Transmit Buffer2**.
1	0	0	1	Transmission Request with Tx Buffer3 The message, previously written to the CANxTFI3, CANxTID3, and optionally the CANxTDA3 and CANxTDB3 registers, is queued for transmission **from the selected Transmit Buffer3**.
1	1	1	1	Transmission Request with Tx Buffer1 If at two or all three of STB1, STB2 and STB3 bits are selected when TR=1 is written, Transmit Buffer will be selected based on the chosen priority scheme **Transmit priority Mode** (TPM) =1 & TFIx[7:0] – Buffer with lower Tx priority value wins prioritization and sent first

AT		Abort Transmission.
	0 (no action)	Do not abort the transmission.
	1 (present)	if not already in progress, a pending Transmission Request for the selected Transmit Buffer is cancelled.
RRB		Release Receive Buffer.
	0 (no action)	Do not release the receive buffer.
	1 (released)	The information in the Receive Buffer (consisting of CANxRFS, CANxRID, and if applicable the CANxRDA and CANxRDB registers) is released, and becomes eligible for replacement by the next received frame. If the next received frame is not available, writing this command clears the RBS bit in the Status Register(s).
CDO		Clear Data Overrun.
	0 (no action)	Do not clear the data overrun bit.
	1 (clear)	The Data Overrun bit in Status Register(s) is cleared.
SRR		Self Reception Request.
	0 (absent)	No self reception request.
	1 (present)	The message, previously written to the CANxTFS, CANxTID, and optionally the CANxTDA and CANxTDB registers, is queued for transmission from the selected Transmit Buffer and received simultaneously. This differs from the TR bit above in that the receiver is not disabled during the transmission, so that it receives the message if its Identifier is recognized by the Acceptance Filter.

7. CAN Global Status Register (CAN1GSR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								BS	ES	TS	RS	TCS	TBS	DOS	RBS
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TxERR								RxERR							

Receive Buffer Status (RBS)

RBS = 0 – No message available

RBS = 1 – At least 1 message is received by double receive buffer and available in RFS, RID, RDA and RDB.

Note:

1) The buffers need to be cleared after copying in received message in variables by using Release Receive Buffer (RRB) of CMR.
2) This is the bit to be monitored to decide whether message is received or not in order to copy the received messaged sitting in RFS, RFD, RDA and RDB.

Transmit Buffer Status (TBS)

TBS = 0 (locked) - At least one of the Transmit Buffers is not available for the CPU, i.e. at least
one previously queued message for this CAN controller has not yet been sent, and therefore software should not write to the CANxTFI, CANxTID,CANxTDA, nor CANxTDB registers of that (those) Tx buffer(s).
TBS = 1 (released) All three Transmit Buffers are available for the CPU

Baud rate Generation for CAN in LPC1768

CAN Bus Timing Register:

This register controls how various CAN timings are derived from the APB clock. It defines the

- Values of the Baud Rate Prescaler (BRP) and
- Synchronization Jump Width (SJW).
- It defines the length of the bit period, the location of the sample point and
- The number of samples to be taken at each sample point.

Note: It can be read at any time but can only be written if the RM bit in CANmod is 1.

CAN Bus Timing Register (CAN1BTR - address 0x4004 4014, CAN2BTR - address 0x4004 8014) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
9:0	BRP		Baud Rate Prescaler. The APB clock is divided by (this value plus one) to produce the CAN clock.	0	X
13:10	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	
15:14	SJW		The Synchronization Jump Width is (this value plus one) CAN clocks.	0	X
19:16	TESG1		The delay from the nominal Sync point to the sample point is (this value plus one) CAN clocks.	1100	X
22:20	TESG2		The delay from the sample point to the next nominal sync point is (this value plus one) CAN clocks. The nominal CAN bit time is (this value plus the value in TSEG1 plus 3) CAN clocks.	001	X
23	SAM		Sampling		
		0	The bus is sampled once (recommended for high speed buses)	0	X
		1	The bus is sampled 3 times (recommended for low to medium speed buses to filter spikes on the bus-line)		
31:24	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	

Baud rate Pre scaler (BRP)

The period of the CAN system clock tSCL is programmable and determines the individual bit timing. The CAN system clock tSCL is calculated using the following equation:

$t_{SCL} = tCANsuppliedClock\ x\ (BRP + 1)$ ---------- (1)

$t_{SCL} = tCANsuppliedClock \times (512 \times BRP.9 + 256 \times BRP.8 + 128 \times BRP.7 + 64 \times BRP.6 + 32 \times BRP.5 + 16 \times BRP.4 + 8 \times BRP.3 + 4 \times BRP.2 + 2 \times BRP.1 + BRP.0 + 1)$

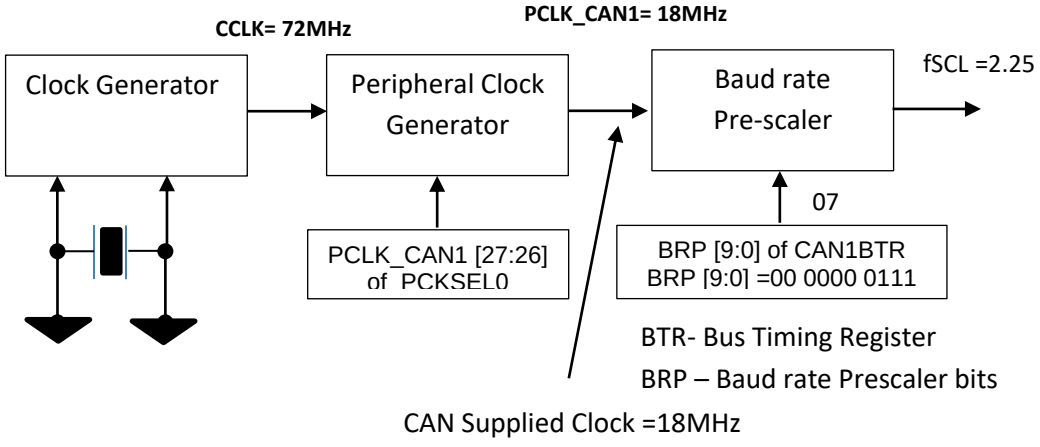
Example:

BRP[9:0] = 00 0000 0111

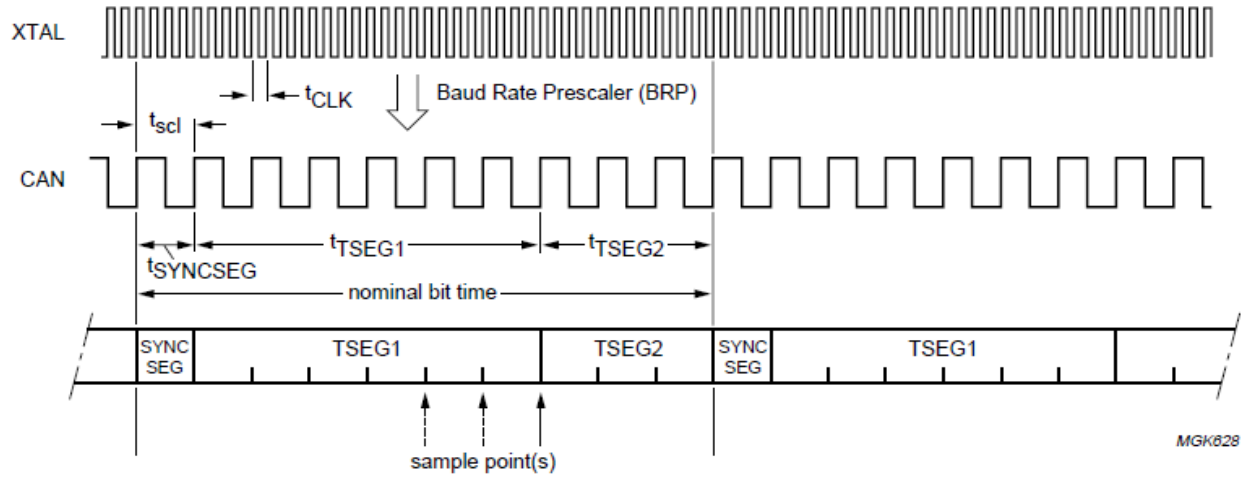
f_{SCL} = f CAN supplied clock / (BRP+1) = 18MHz / (7+1) = 2.25 MHz

$t_{SCL} = \frac{1}{18\ MHz} \times (7 + 1) = 1/\ 2.25\ MHz$ ---------- (2)

Block Diagram of Clock Generator:



Timing Diagram



void CAN_Init(void)

{

LPC_SC->PCLKSEL0|=0X00000000;

LPC_PINCON->PINSEL0|=0X00000005;

LPC_CAN1->MOD=0x00000001

LPC_CAN1->CMR=0X00000000;

LPC_CAN1->GSR=0x00000000;

LPC_CAN1->IER=0x00000001;

LPC_CAN1->BTR=0X001C0007;

LPC_CAN1->MOD=0x00000000;

}

Acceptance Filter in CAN Protocol

Acceptance Filter Register	RAM Location	Memory Location & Memory ID Stored in binary		Index Value(10Bits)
SFF_sa	0x0000 0000	**0x21(Message ID Lower Byte)**	**11 bit ID**	**0x000**
	0x0000 0001	**0x00(Message ID Higher Byte)**		
	0x0000 0002	0x22	11 bit ID	0x001
	0x0000 0003	0x00		
	0x0000 0004	0x23	11 bit ID	0x002
	0x0000 0005	0x00		
	0x0000 0006	0x24	11 bit ID	0x003
	0x0000 0007	0x00		
SFF_GRP_sa	0x0000 0008	**0x25**	Group of 4, 11 bit IDs	**Whole standard Frame Group will have same Index 0x004**
	0x0000 0009	**0x00**		
	0x0000 000A	0x26		
	0x0000 000B	0x00		
	0x0000 000C	0x27		
	0x0000 000D	0x00		
	0x0000 000E	0x28		
	0x0000 000F	0x00		
EFF_sa	0x0000 0010	0x29	**29 bit ID**	**0x005**
	0x0000 0011	0x00		
	0x0000 0012	0x00		
	0x0000 0013	0x00		
	0x0000 0014	**0x2A**	29 bit ID	0x006
	0x0000 0015	**0x00**		
	0x0000 0016	**0x00**		
	0x0000 0017	**0x00**		
	0x0000 0018	0x2B	29 bit ID	0x007
	0x0000 0019	0x00		
	0x0000 001A	0x00		
	0x0000 001B	0x00		
EFF_GRP_sa	0x0000 001C	0x2C	Group of 6,29 bit IDs	**Whole Extended Frame Group will have same Index 0x008**
	0x0000 001D	0x00		
	0x0000 001E	0x00		
	0x0000 001F	0x00		
	⋮	⋮		
	0x0000 0030	0x31		
	0x0000 0031	0x00		
	0x0000 0032	0x00		
	0x0000 0033	0x00		
ENDofTable	0x0000 0034			

Name	Description	Access	Reset Value	Address
SFF_sa	Standard Frame Individual Start Address Register	R/W	0	0x4003 C004
SFF_GRP_sa	Standard Frame Group Start Address Register	R/W	0	0x4003 C008
EFF_sa	Extended Frame Start Address Register	R/W	0	0x4003 C00C
EFF_GRP_sa	Extended Frame Group Start Address Register	R/W	0	0x4003 C010
ENDofTable	End of AF Tables register	R/W	0	0x4003 C014

The above Table indicates that

Registers	Location	Description
SFF_sa	0x00000000	It indicates that Standard Frame (11 bit ID) start address starts at 0x0000 0000 and ends at 0x0000 0007. These address location hold Individual Message ID with Individual Index value of 10 bit size. As Message ID is of 11 bit, each id uses 2 locations. **Hence there 4, 11 bit IDs stored with index values as 0x000,0x001,0x002,0x003.**
SFF_GRP_sa	0x00000008	It indicates that Standard Frame (11 bit ID) Group start address starts at 0x0000 0008 and ends at 0x0000 000F. These address location **hold Group of Message ID with common Index value of 10 bit size.** As Message ID is of 11 bit, each id uses 2 locations. **Hence it is group of 4, 11 bit IDs stored with index value as 0x004.**
EFF_sa	0x00000010	It indicates that Extended Frame (29 bit ID) start address starts at 0x0000 0010 and ends at 0x0000 001B. These address location hold Individual Message ID with Individual Index value of 10 bit size. As Message ID is of 29 bit, each id uses 4 locations. **Hence there 3, 29 bit IDs stored with index values as 0x005,0x006,0x007.**
EFF_GRP_sa	0x0000001C	It indicates that Extended Frame (29 bit ID) Group start address starts at 0x0000 001C and ends at 0x0000 0033. These address location **hold Group of Message ID with common Index value of 10 bit size.** As Message ID is of 29 bit, each id uses 4 locations. **Hence it is group of 6, 29 bit IDs stored with index value as 0x007.**
ENDofTable	0x00000034	

Initialization according to above

```
void CAN_ACC(void)
{
LPC_CANAF->AFMR=0x00000001;
LPC_CANAF->SFF_sa=0x00000000;
LPC_CANAF->SFF_GRP_sa=0x00000008;
LPC_CANAF->EFF_sa=0x00000010;
LPC_CANAF->EFF_GRP_sa=0x0000001C;//
LPC_CANAF->ENDofTable=0x00000034;//
LPC_CANAF->AFMR=0x00000000;
}
```

```
void looktable(void)
{
 LPC_CANAF->AFMR=0x00000001;
 LPC_CANAF_RAM->mask[0]=0x00000021;
 LPC_CANAF_RAM->mask[1]=0x00000022;
 LPC_CANAF_RAM->mask[2]=0x00000023;
 LPC_CANAF_RAM->mask[3]=0x00000024;
 LPC_CANAF_RAM->mask[4]=0x00000025;
 LPC_CANAF_RAM->mask[4]=0x00000026;
 LPC_CANAF_RAM->mask[4]=0x00000027;
 LPC_CANAF_RAM->mask[4]=0x00000028;
 LPC_CANAF_RAM->mask[5]=0x00000029;
 LPC_CANAF_RAM->mask[6]=0x0000002A;
 LPC_CANAF_RAM->mask[7]=0x0000002B;
 LPC_CANAF_RAM->mask[8]=0x0000002C;
 LPC_CANAF_RAM->mask[8]=0x0000002D;
 LPC_CANAF_RAM->mask[8]=0x0000002E;
 LPC_CANAF_RAM->mask[8]=0x0000002F;
 LPC_CANAF_RAM->mask[8]=0x00000030;
 LPC_CANAF_RAM->mask[8]=0x00000031;
 LPC_CANAF->AFMR=0x00000000;
}
```

Example2:

```
void CAN_ACC(void)
{
LPC_CANAF->AFMR=0x00000001;
LPC_CANAF->SFF_sa=0x00000000;
LPC_CANAF->SFF_GRP_sa=0x00000000;
LPC_CANAF->EFF_sa=0x00000000;
LPC_CANAF->EFF_GRP_sa=0x00000008;//
LPC_CANAF->ENDofTable=0x00000008;//
LPC_CANAF->AFMR=0x00000000;
}
```

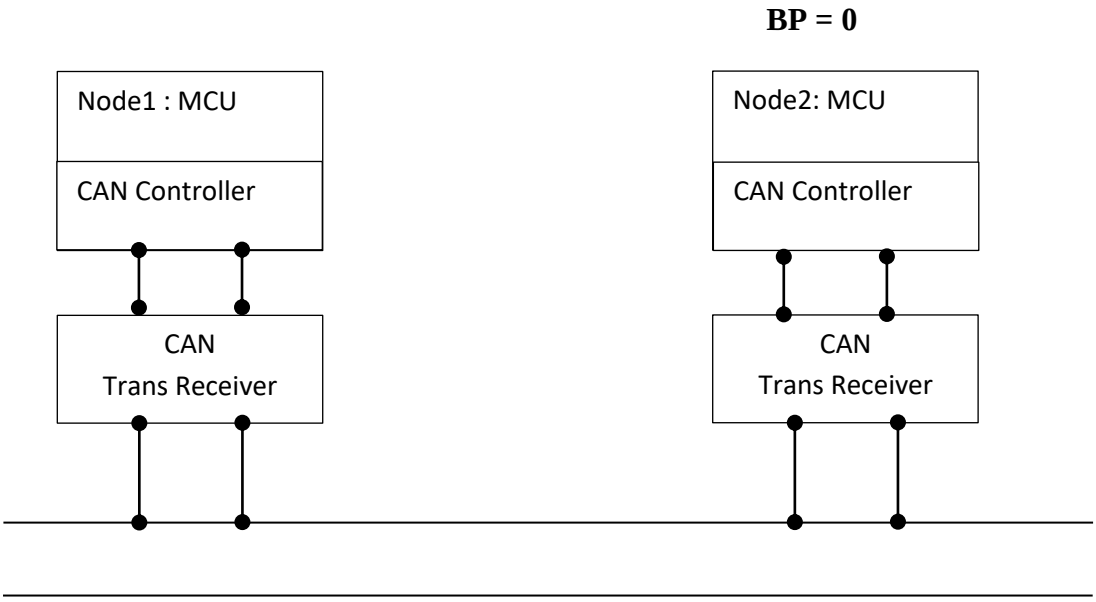
```
void looktable(void)
{
 LPC_CANAF->AFMR=0x00000001;
 LPC_CANAF_RAM->mask[0]=0x00000020;
 LPC_CANAF_RAM->mask[1]=0x00000021;
 LPC_CANAF->AFMR=0x00000000;
}
```

Acceptance Filter Register	**RAM Location 0x4003 8000 - 0x4003 87FF**	Memory Location & Memory ID Stored in binary		Index Value(10Bits)
SFF_sa ,SFF_GRP_sa EFF_sa	0x4003 8000	0x20	29	0X000 mask[0]
	0x4003 8001	0x00		
	0x4003 8002	0x00		
	0x4003 8003	0x00		
	0x4003 8004	0x21	29	0X001 mask[1]
	0x4003 8005	0x00		
	0x4003 8006	0x00		
	0x4003 8007	0x00		
EFF_GRP_sa ENDofTable	0x4003 8008			

In the above example 29 bit identification is used. It indicates there is No Memory allocated for Standard Frame (Individual and Group address) and Extended Frame (Group address). There are teo 29 bit ID stored in index 0 and 1 as above.

This means Rx accepts Message ID 0x0000 0020 and ID=0x0000 0021 and when receive a message with Message ID 0x0000 0020, Its Index value 0x000 is copies into RFS[9:0] as Rx Priority bits. Similarly for ID=0x0000 0021 Index 0x001 will be copied into RFS[9:0] as Rx Priority bits.

Basic concept of Acceptance Filter



	Index	Message ID
AF_RAM	mask[0]	0X0000 0014
AF_RAM	mask[1]	0X0000 0015
AF_RAM	mask[2]	0X0000 0016
AF_RAM	mask[3]	0X0000 0017

	Index	Message ID
AF_RAM	mask[0]	0X0000 0020
AF_RAM	mask[1]	0X0000 0021
AF_RAM	mask[2]	0X0000 0022
AF_RAM	mask[3]	0X0000 0023

Example: If in the receiver lookup table is defined as follows
LPC_CANAF_RAM-> mask[0]=0x00000020;
LPC_CANAF_RAM-> mask[1]=0x00000021;
At the receiver (Node2), Upon reception of a message, It checks whether the message ID of received message is present in the acceptance filter or not. If the received message ID is present in the acceptance filter list then message will be accepted else rejected. Upon acceptance, the Index Value of accepted message ID is copied into ID index[9:0] of RFS.
.Therefore when Message with ID=0x00000021 is received,
In the above lookup table message with ID=0x00000020 has Index as 0x00 and message with ID=0x00000021 has index as 0x01.
Similarly, If node2 transmits and Node1 receives a message, then it will check whether the received message ID is present in look up table or not. Above example shown in block diagram indicates the message with ID 0x0000 0014, 0x0000 0015, 0x0000 0016 and 0x0000 0017 will be accepted.