**Implementation of Controller Area Network (CAN) using Cortex-M Microcontroller**

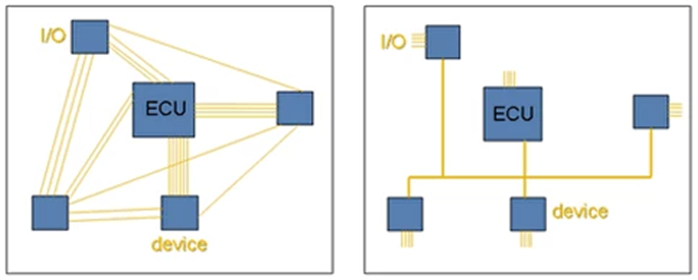
**1. Aim:**

Establish the CAN communication between two microcontrollers for the given bit rate.

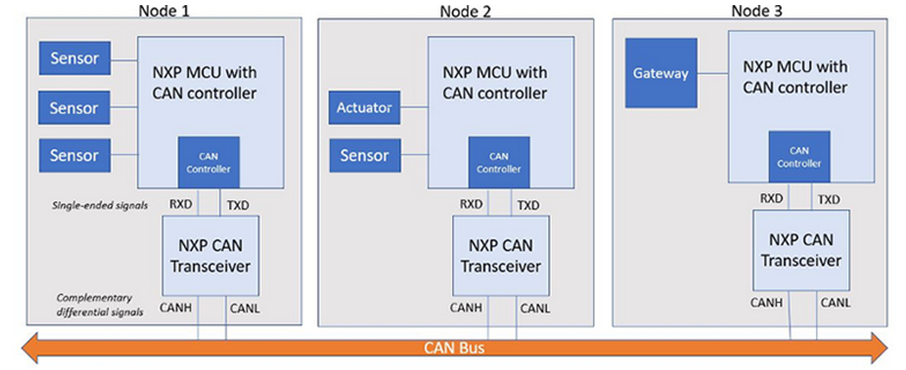
**2. Introduction:**

CAN stands for Controller Area Network. It was first created by the German Automotive Company Robert Bosch in the mid-1980s for automobile applications. CAN is a message-based protocol, which means that communication does not happen based on addresses but rather the CAN message itself contains the priority and the data that needs to be transmitted. Every node on the bus receives this message, and then each node decides whether that information is useful or needs to be discarded. A single message can be destined for a particular receiver or for multiple nodes that are present on the bus. Another important feature of the CAN protocol is its ability to request information from other nodes which is known as "Remote Transmission Request" or "RTR". Since CAN is a message-based protocol a node can be added to the system without the need to reprogram the other nodes for acknowledging the new addition.

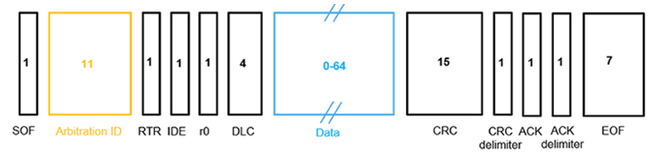
There are four types of CAN frames in the CAN protocol. The most common type is the "Data Frame", which is used when a node transmits information to other nodes. The second one is the "Remote Frame" which is a Data frame with the RTR bit set to signify a Remote Transmission Request. The other two frames are namely the Error Frame and the Overload Frame which are used for handling errors. Error Frames are generated by any one of the nodes that detect any one of the many protocol errors defined in CAN. Overload Frames are generated by nodes that require more time to process the data that has already been received.



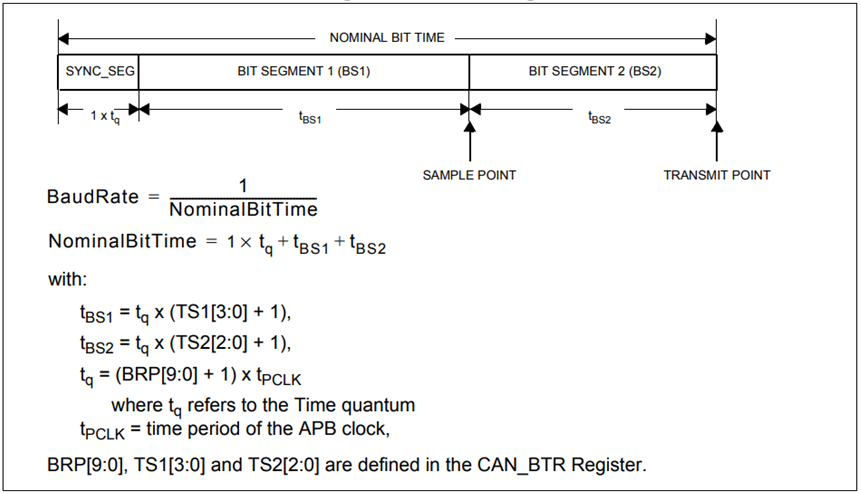
**3.1 CAN Network Topology:**



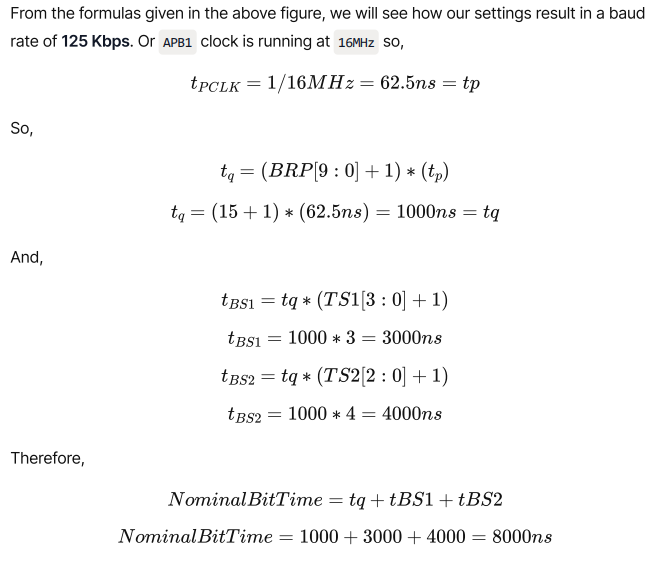
**3.2 CAN Normal and Extended Data Frames:**



**3.3 CAN Timing Configuration for the Given Bit Rate:**



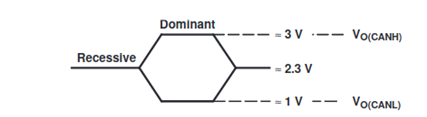
**3.4 Illustration:**





**3.5 CAN Signaling:**

Logic 1 is a recessive state. To transmit 1 on CAN bus, both CAN high and CAN low should be applied with 2.5V. Logic 0 is a dominant state. To transmit 0 on CAN bus, CAN high should be applied at 3.5V and CAN low should be applied at 1.5V. The ideal state of the bus is recessive. If the node reaches the dominant state, it cannot move back to the recessive state by any other node



**4. Components Required:**

1. 2 Cortex-M4 STM32F407 Discovery boards
2. CAN bus setup
3. 2 CAN transceivers
4. Analog Discovery or logic analyzer for CAN frame visualization
5. Connecting wires

**5. Embedded C Codes:**

**5.1 Problem Statement:**

1. Push button connected in PA0 is monitored continuously.
2. Whenever the user presses the button, corresponding count is transmitted.
3. Receiver turns ON the count number of LEDs connected in PD12, P13, PD14, and PD15.
4. Count variable is reset when it reaches 4.

**5.2 CAN Transmit Code:**

#include <stdint.h>

#include <stm32f4xx.h>

uint8\_t value;

uint8\_t arr[5] = {0x0,0x1,0x3,0x7,0xF};

void Set\_Pin(void)

{

RCC->AHB1ENR |= (1<<1)|(1<<3); // GPIOB, GPIOD are to be set

GPIOD->MODER |= (0x55 << 24); // PD12, PD13, PD14, PD15 will be turned on

GPIOB->MODER |= (0xA << 16); // Alternate functionality mode

GPIOB->AFR[1] |= (0x9 | (0x9<<4)); // AF9 for PB8 , PB9 pins

}

uint8\_t CAN\_Rx(void)

{

while(!(CAN1->RF0R & 3)); // waiting for atleast one message

uint8\_t data = (CAN1->sFIFOMailBox[0].RDLR) & 0xFF; // only 1 byte

CAN1->RF0R |= (1<<5); // Release FIFO MailBox

return data;

}

void CAN\_Init(void)

{

RCC->APB1ENR |= 1<<25; // CAN1 Initialize

CAN1->MCR |= (1<<0); // Enter into Initialization mode

while(!(CAN1->MSR & 0x1)); // wait until INAK bit is set

/\* Exit Sleep Mode \*/

CAN1->MCR &= ~(1<<1);

while(CAN1->MSR & 0X2); // SLAK Sleep ACK

CAN1->BTR &= ~(3<<24); // SWJ 1 Time Quantum

CAN1->BTR &= ~(0x7F << 16);

CAN1->BTR |= (12 << 16); // Time Segment 1

CAN1->BTR |= (1 << 20); // Time Segment 2

CAN1->BTR |= (7 << 0); // Baud-Rate Prescaler

/\* Exit Initialization mode \*/

CAN1->MCR &= ~(1<<0);

while(CAN1->MSR & 0x1);

/\* Filter Configuration \*/

CAN1->FMR |= 1<<0; // Initialise filter mode

CAN1->FMR |= 14<<8; // they define start bank for CAN2

CAN1->FS1R |= 1<<13; // 32 bit scale configuration Filter

// Filtering it with respect to the ID

CAN1->FM1R |= (1<<13); // Identifier List-Mode

CAN1->sFilterRegister[13].FR1 = 254 << 21; // STD ID

CAN1->FA1R |= (1<<13);

CAN1->FMR &= ~(0x1);

}

void delay(uint32\_t d)

{

for(uint32\_t i=0;i<d\*1000;i++);

}

int main(void)

{

/\* Loop forever \*/

Set\_Pin();

CAN\_Init();

while(1)

{

value = CAN\_Rx();

GPIOD->ODR &= ~(0xF << 12);

GPIOD->ODR |= arr[value] << 12;

delay(100);

}

}

**5.3 CAN Receive Code:**

#include <stdint.h>

#include <stm32f4xx.h>

uint8\_t value,count=0;

//PB8 is CRX, PB9 is CTX

void Set\_Pin(void)

{

RCC->AHB1ENR |= (1<<1)|(1<<0); // GPIOA, GPIOB

GPIOA->MODER &= ~(3<<0); // Input

GPIOB->MODER |= (0xA << 16); // Alternate functionality mode

GPIOB->AFR[1] |= (0x9 | (0x9<<4)); // AF9 for PB8 , PB9 pins

}

void CAN\_Tx(void)

{

CAN1->sTxMailBox[0].TDLR = count;

/\* Request for transmission \*/

CAN1->sTxMailBox[0].TIR |= 1;

}

void CAN\_Init(void)

{

RCC->APB1ENR |= 1<<25; // CAN1 Initialize

CAN1->MCR |= (1<<0); // Enter into Initialization mode

while(!(CAN1->MSR & 0x1)); // wait until INAK bit is set

/\* Exit Sleep Mode \*/

CAN1->MCR &= ~(1<<1);

while(CAN1->MSR & 0X2); // SLAK Sleep ACK

CAN1->BTR &= ~(3<<24); // SWJ 1 Time Quantum

CAN1->BTR &= ~(0x7F << 16);

CAN1->BTR |= (12 << 16); // Time Segment 1

CAN1->BTR |= (1 << 20); // Time Segment 2

CAN1->BTR |= (7 << 0); // Baud-Rate Prescaler

/\* Exit Initialization mode \*/

CAN1->MCR &= ~(1<<0);

while(CAN1->MSR & 0x1);

/\* Setting Up Transmission \*/

CAN1->sTxMailBox[0].TIR = 0;

CAN1->sTxMailBox[0].TIR |= (254 << 21); // STD ID : 254

CAN1->sTxMailBox[0].TDHR = 0; // Data byte 4, 5, 6, 7

CAN1->sTxMailBox[0].TDLR = 0;

CAN1->sTxMailBox[0].TDTR = 1; // Sending only 4 bytes of data

/\* Filter Configuration \*/

CAN1->FMR |= 1<<0; // Initialise filter mode

CAN1->FMR |= 14<<8; // they define start bank for CAN2

CAN1->FS1R |= 1<<13; // 32 bit scale configuration Filter

// Filtering it with respect to the ID

CAN1->FM1R |= (1<<13); // Identifier List-Mode

CAN1->sFilterRegister[13].FR1 = 138 << 21; // STD ID

CAN1->FA1R |= (1<<13);

CAN1->FMR &= ~(0x1);

}

void delay(uint32\_t d)

{

for(uint32\_t i=0;i<d\*1000;i++);

}

int main(void)

{

/\* Loop forever \*/

Set\_Pin();

CAN\_Init();

while(1)

{

if(GPIOA->IDR & (1<<0))

{

count++;

CAN\_Tx();

if(count == 4)

count = 0;

}

delay(200);

}

}

**6. Logic Analyzer Output:**

\*Insert images from logic analyzer here\*

**7. Conclusion:**

In this experiment, CAN message is successfully transmitted and received by using two STM32F407 Discovery boards. The CAN message is visualized using the logic analyzer.