Controller Area Network (CAN) - Implementation using STM32F407VG Microcontroller from STMicroelectronics

SRIVATHS RAMASUBRAMANIAN

January 2024

1 Controller Area Network Fundamentals

The Controller Area Network (CAN), first invented by Bosch-GmbH, is a multi-master message broadcast system that works on the principle of differential signaling. Unlike traditional protocols like USB or ethernet, CAN does not send large blocks of data from one node to another. It however, sends short messages to the entire network, and these messages are made available to all the nodes in the network

The protocol is very popular, and used in applications, where the network to be created spans a large area. Some of the popular applications include automobiles, building automation, industry automation etc.

The CAN network is a two-wire bus system. The wires used are twisted pairs, with a characteristic impedance of 120 ohms at the ends. CAN is a broadcast type of bus, wherein all the nodes in the network will receive the message that is transmitted by a node at a particular point of time. Every message transmitted by a node contains identifiers, which can be used to direct the message to a receiver node in the network and also set the priority of the incoming message.

CAN also works on the principle of differential signaling. With the use of differential signaling, it is possible to transmit messages over a longer distance.

2 About the Microcontroller - STM32F407VG Discovery Board by STMicroelectronics

The STM32F407VG DISC-1 board, produced by STMicroelectronics, is an ARM Cortex-M4 based microcontroller board, equipped with the ST-LINK V2A debugger interface. It supports multiple peripherals and functions, which include GPIOs, SPI, I2C, CAN, UART (USART), timers, Ethernet, DMA, Random Number Generators (RNG) and Serial Audio Interface (I2S). It is also equipped to handle a wide variety of interrupts for each and every peripheral that is interfaced to it.

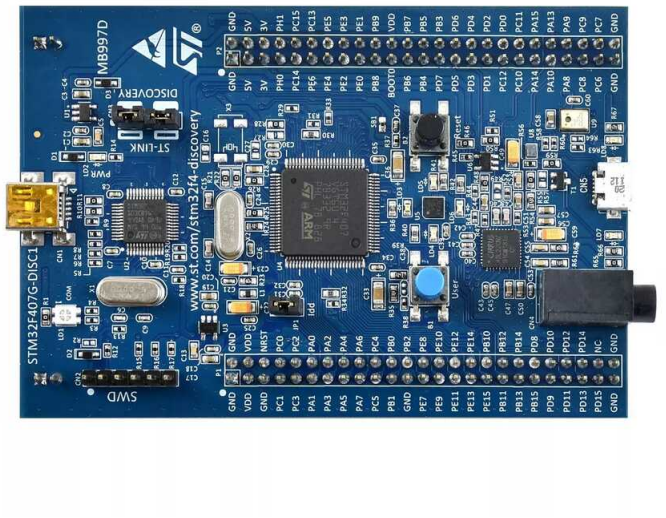

Figure 1: STM32F407VG DISC-1

The microcontroller features the bxCAN peripheral (Basic-Extended CAN), which can be enabled and configured to implement the CAN network, where the microcontroller serves as the network node.

3 STM32 bxCAN Peripheral

The Basic Extended CAN, named bxCAN, interfaces the CAN network and it supports CAN 2.0A and 2.0B versions. It supports the following features

- Supports CAN protocol versions 2.0A and 2.0B
- Bit rates upto 1 Mbps
- Supports time triggered communication options (period-elapsed-callbacks)
- Three dedicated transmit mailboxes
- Configurable transmit priority
- Two receive FIFOs with three stages
- 28 configurable filter banks shared between CAN1 and CAN2
- Identifier list feature
- Auto re-transmission of message

- Maskable interrupts

The bxCAN peripheral shares two CAN interfaces, which include

- CAN1 : Master bxCAN that manages the communication between slave CAN and has direct access to SRAM memory
- CAN2 : Slave bxCAN that has no access to SRAM memory and can only act as a node

The CAN interfaces are mapped to the address 0x40006400 to 0x40006BFF in the memory map of the microcontroller as shown in figure 2

Boundary address	Peripheral	Bus	Register map
0x4000 7400 - 0x4000 77FF	DAC		*Section 14.5.15: DAC register map on page 453*
0x4000 7000 - 0x4000 73FF	PWR		*Section 5.6: PWR register map on page 149*
0x4000 6800 - 0x4000 6BFF	CAN2		*Section 32.9.5: bxCAN register map on page 1118*
0x4000 6400 - 0x4000 67FF	CAN1		
0x4000 5C00 - 0x4000 5FFF	I2C3		*Section 27.6.11: I2C register map on page 872*
0x4000 5800 - 0x4000 5BFF	I2C2		
0x4000 5400 - 0x4000 57FF	I2C1		

Figure 2: Base address of CAN1 and CAN2 - MCU memory map

Both the CAN interfaces share a common 512-byte SRAM memory
The CAN network topology is given in figure 3

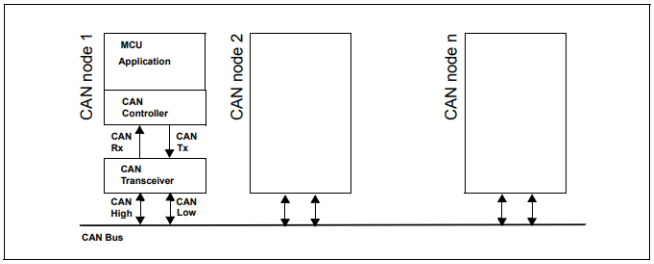

Figure 3: CAN network topology

In figure 3, CAN transceivers are employed to convert the single ended digital signal into a differential signal. This differential signal is then fed into the CAN bus. However, this transceiver block is optional, and the CAN network can be realised using an alternate connection scheme.

4 CAN Differential Signaling

The CAN application consists of the CAN controller and a CAN transceiver tied to the bus. There are hence, two types of signals that are processed by the CAN transceivers, They include -

- Single ended signals : From RXD and TXD pins of the CAN controller
- Differential signals : From CANH and CANL pins of the CAN transceiver

During normal operation of the CAN controller, it converts the single ended output signal from the CAN controller to differential signal and also converts the differential signal back to the single ended signal for input to the CAN controller. Hence, the CAN transceiver provides facility for transmission and reception of differential signals.

The block diagram of signalling is given in figure 4

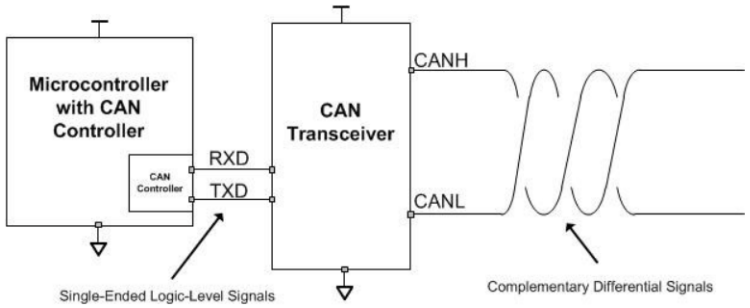

Figure 4: CAN differential signaling

The CAN bus features two states. They include-

- Dominant state : When TXD pin is single ended logic-0
- Recessive state : When TXD pin is single ended logic-1

The CAN bus signal states are depicted in figure 5

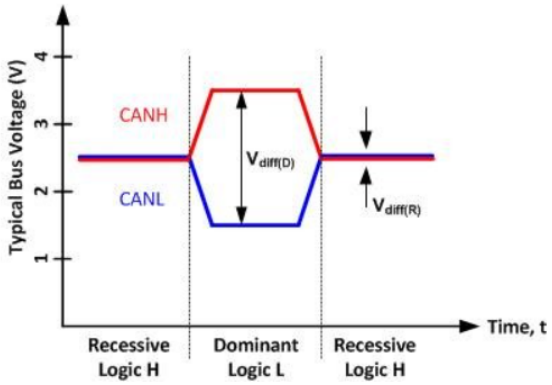

Figure 5: Differential signaling states

It can be noticed from figure 4 that both CANH and CANL are equally biased at 2.5V in the recessive state. When the TXD pin transmits logic-0, the voltage at CANH approximately shifts to 3.5V and the voltage of CANL approximately shifts to 1.5V. Upon subtraction, the bus state can be determined, as given by the equation below-

$$V_{diff} = V_{CANH} - V_{CANL} \tag{1}$$

If the value of V_{diff} is less than 0.5V, then the bus is in recessive state. When the value of V_{diff} is greater than 0.9V, then the bus is in dominant state. If the value of V_{diff} is between 0.5V and 0.9V, then the bus is in indeterminate state.

It should be noted that the bus exists in recessive state only if all the nodes connected to it are transmitting logic-1. Even if one node in the bus transmits a logic-0, then, the bus enters into the dominant state.

Due to the implementation of differential signalling, the CAN network is extremely immune to noise. This signaling scheme helps to cancel the noise in the network, due to cancelling of common mode signals in the bus.

5 CAN Message Types and Frame Format

The CAN protocol features 4 types of messages. They include-

- Data frame
- Remote frame
- Error frame
- Overload frame

The CAN data frame sends the data from one node to numerous other nodes in the network while the remote frame is sent from one node to another to request data. A remote frame is ideally followed by the transmission of a data frame.

Data frames and remote frames follow the same data format. A remote frame is a data frame without data in its data field. The frame format for both data and remote frames are given in figure 6

Figure 6: CAN Frame Format - brief

A more detailed view of the CAN frame format used in STM32 boards is given in figure 7

Both data frames and remote frames are identical in structure. The only way to distinguish between the data frame and remote frame is by checking the RTR(Request to Remote) bit in the frame. For data frames, RTR bit is

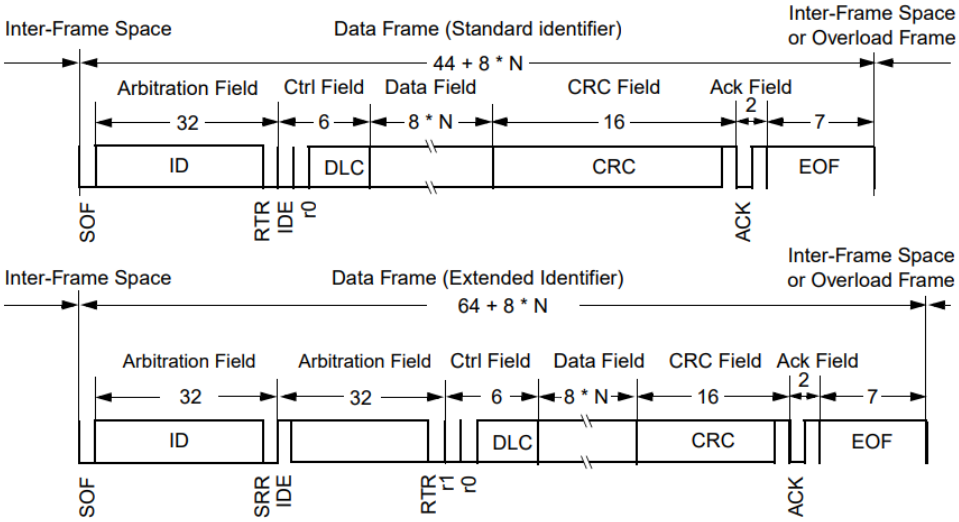

Figure 7: STM32 CAN dataframe format

cleared while for remote frames, the RTR bit is set. Further, the data field in the remote frame is empty.

The SOF bit indicates the Start of Frame bit. It is always a dominant bit. The SOF bit also changes the state of the bus from idle to active.

The message identifier field comprises of a 11 bit identifier. The extended CAN features 29 bit identifier for the message field. This is the field that can be used for message arbitration. Arbitration is a process by which if there are two or more nodes competing to transmit a message over the bus, that message which wins arbitration based on the value encoded into the identifier will get transmitted into the bus, and the remaining messages will be discarded, or set to pending state in the transmit mailbox. Lower the identifier, higher the priority of the message.

The RTR bit in the frame is Request to Remote. It is the bit that is used to distinguish between a data frame and a remote frame.

- If RTR = 0 then the frame is a data frame
- If RTR = 1 then the frame is a remote frame

The DLC field is the Data-Length-Code. It is a 4 bit field that is used to specify the length of the data payload that is transmitted. The length of the payload can be a maximum of 8 bytes.

The data field stores the data payload to be transmitted. It has a maximum length of 8 bytes. Upon reception by a node, the information stored in this field is extracted and processed for further tasks.

The frames also feature the CRC(Cyclic Redundancy Check) field. This field is used to check the integrity of the transmitted message at the receiver node.

Before the transmission of the message, the CAN node transmitting the frame calculates the CRC field. Upon reception by the receiver node, the receiver node recalculates the CRC and tries to match it with the CRC in the data / remote frame. If the values of CRC do not match, then the message has been corrupted during transmission, and the receiver node can demand re-transmission of the message.

The ACK bit (Acknowledge bit) is a very important bit, and it helps to ascertain if a message is transmitted correctly. If the receiver node senses that the message is not transmitted correctly based on violation of CRC, it then sets the ACK bit to recessive. When the transmitter then senses that the ACK bit in the bus is recessive, it then understands that the message is not transmitted correctly, and hence re-transmits the message again. If the CRC at the receiver end is not violated, then, the receiver sets the ACK bit to dominant, hence, indicating that the message has been successfully transmitted.

The EOF stands for End of Frame, which is a set of 7 bits which are all recessive. The IFS bits stands for Inter-Frame Spacing, which is a set of 3 recessive bits. Hence, at the end of the frame, there exists a set of 10 recessive bits. After the transmission of the EOF and IFS bits, the bus enters into the idle state. Hence, during the idle state of the bus, it transmits only recessive bits.

The SOF bit is the first dominant bit. Hence, when the bus is idle, the SOF bit being the dominant bit is transmitted, and puts the bus into active state.

The CAN protocol also features two additional frame formats. They include

- Error frame
- Overrun frame

The Error frame is a special message that is sent when a particular node detects violation in a CAN message. This forces all the nodes to send the error frame and the transmitter re-transmits the message.

6 CAN Operational Modes

The CAN controller can be interfaced in different operational modes. They include-

- Initialization mode
- Normal mode
- Sleep mode

In the Initialization mode, the software can be initialized when the hardware is in initialization mode. When the node is in initialization mode, the message transfers to and from the node are stopped and the state of the CAN bus is recessive.

Once initialization has been complete, the software must request hardware enter into normal mode so that the node can synchronize on the CAN bus and can start reception and transmission of messages.

To reduce the power consumption, the bxCAN also features the sleep mode. In this mode, the bxCAN clock is stopped, but however, the mailboxes can still be accessed by the software.

A flow diagram showing the relationship between all the three operational modes is given in figure 8

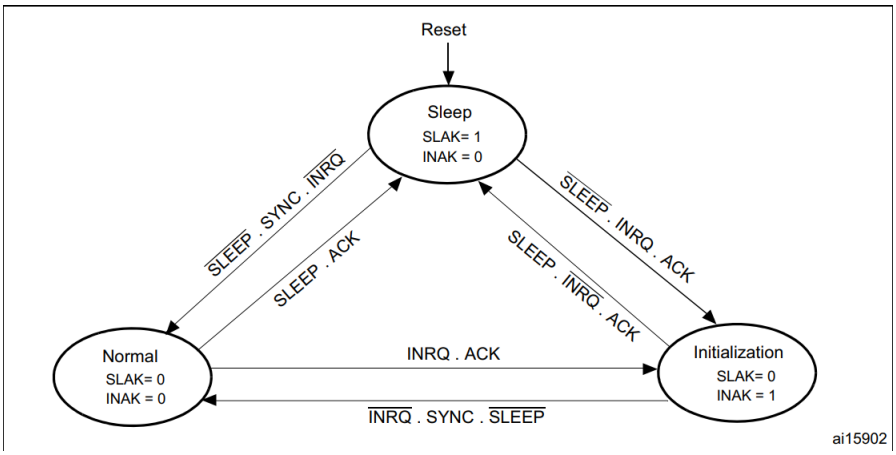

Figure 8: bxCAN Operating Modes

The bxCAN peripheral also features three test modes for testing bxCAN peripheral. The test modes are given by-

- Silent mode
- Loopback mode
- Loopback combined with silent mode

In the silent mode, the bxCAN peripheral can receive messages from the bus. It however, cannot transmit messages over the bus and it sends only recessive bits on the bus. If the node has to send a dominant bit (ACK bit or overload flag), the bit is rerouted internally so that the CAN core can monitor the dominant bit even though the CAN bus will remain in recessive state. The silent mode can be used to monitor traffic on the CAN bus without affecting the transmission of dominant bits. Figure 9 illustrates connection to interface CAN node in silent mode.

In the loopback mode, the CAN node treats its own transmitted messages as received messages and stores them in the received mailbox. The loopback mode is used for self-testing the node. In this mode, the node performs internal feedback from its Tx output to its Rx input. Figure 10 illustrates the connection to interface CAN node in loopback mode.

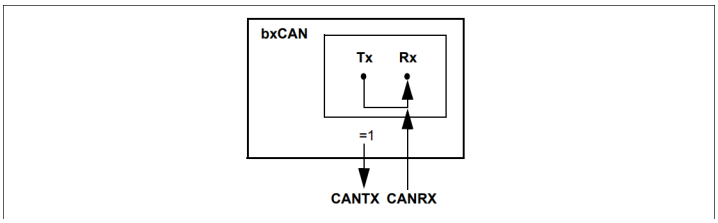

Figure 9: bxCAN - Silent mode

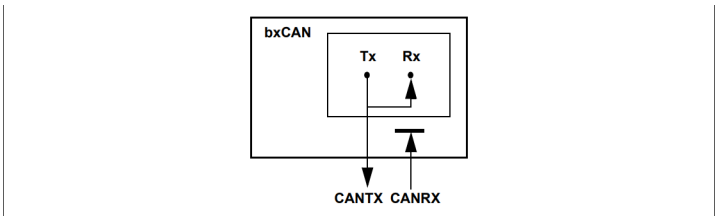

Figure 10: bxCAN - Loopback mode

It is also possible to combine the loopback mode and silent mode. This can be used for host-self-test, where the transmission and reception of messages can be monitored and at the same time, the state of the bus can be held at recessive. Figure 11 illustrates connection to interface CAN node in loopback-silent mode.

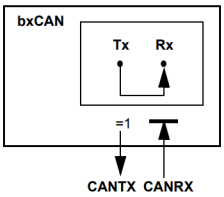

Figure 11: bxCAN - Combined mode