Fachhochschule Münster University of Applied Sciences

Lab for Semiconductor Devices and Bussystems

Exercise Bussystems SS2012/13
Tutor: Dipl. Informatiker Gregor Rebel
MSc. Francisco Estevez

Exercise 4
CAN Bus Management

The Controller Area Network or CAN bus is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer.
CAN bus is a message-based protocol, designed specifically for automotive applications but now also used in other areas such as aerospace, industrial automation and medical equipment.

Interface
The CAN bus specifies four logic signals:

- CAN High (CANH)
- CAN Low (CANL)
- CAN V+ (Power)
- GND (Ground)

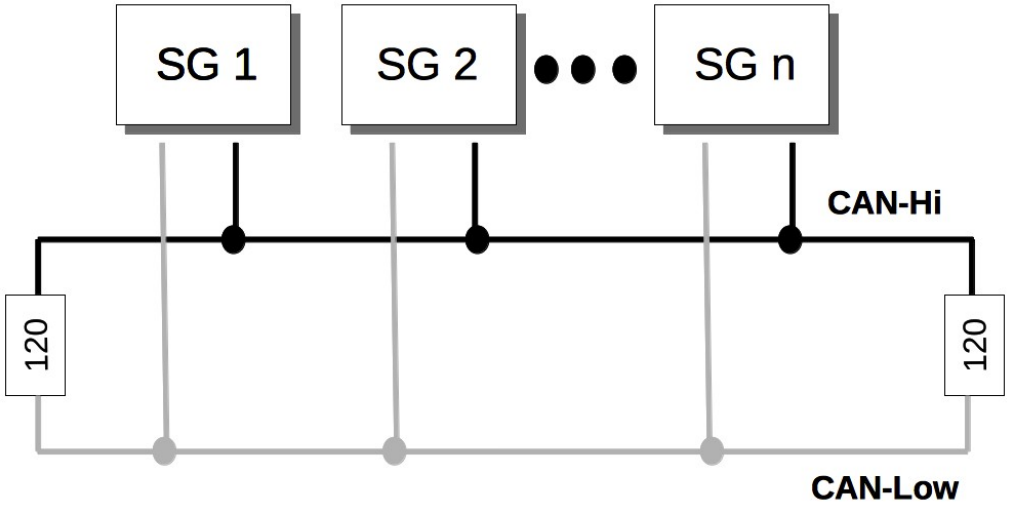


Figure 1: CAN Devices Schema

Technology

CAN is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs). Each node is able to send and receive messages, but not simultaneously. A message consists primarily of an ID (identifier), which represents the priority of the message, and up to eight data bytes. The improved CAN (CAN FD) extends the length of the data section to up to 64 bytes per frame. It is transmitted serially onto the bus. This signal pattern is encoded in non-return-to-zero (NRZ) and is received by all nodes.

The devices are not connected directly to the bus, but through a host processor and a CAN controller. If the bus is free, any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits, i.e., zeroes) will overwrite other nodes' less dominant IDs, so that eventually only the dominant message remains and is received by all nodes. This mechanism is referred to as priority based bus arbitration. Messages with numerically smaller values of IDs have higher priority and are transmitted first.

Each node requires:

- **Host processor**
  - The host processor decides what received messages mean and which messages it wants to transmit itself.
  - Sensors, actuators and control devices can be connected to the host processor.
- **CAN controller** (hardware with a synchronous clock)..
  - *Receiving*: the CAN controller stores received bits serially from the bus until an entire message is available, which can then be fetched by the host processor (usually after the CAN controller has triggered an interrupt).
  - *Sending*: the host processor stores its transmit messages to a CAN controller, which transmits the bits serially onto the bus.
- **Transceiver**
  - *Receiving*: it adapts signal levels from the bus to levels that the CAN controller expects and has protective circuitry that protects the CAN controller.
  - *Transmitting*: it converts the transmit-bit signal received from the CAN controller into a signal that is sent onto the bus.

Bit rates up to 1Mbit/s are possible at network lengths below 40m. Decreasing the bit rate allows longer network distances (e.g., 500 m at 125kbit/s). The improved CAN (CAN FD) extends the speed of the data section by a factor of up to 8 of the arbitration bit rate.

Data transmission

CAN features an automatic arbitration-free transmission. A CAN message that is transmitted with highest priority will succeed, and the node transmitting the lower priority message will sense this and back off and wait.

This is achieved by CAN transmitting data through a binary model of "dominant" bits and "recessive" bits where dominant is a logical 0 and recessive is a logical 1, this means open collector configuration. If one node transmits a dominant bit and another node transmits a recessive bit then the dominant bit "wins" (a logical AND between both).

Tables for dominant/recessive, logical or, and logical and (for comparison)

Bus state with two nodes transmitting:

	Dominant	**Recessive**
Dominant	Dominant	Dominant
Recessive	Dominant	Recessive

Logical or

	0	**1**
0	0	1
1	1	1

Logical and

	0	**1**
0	0	0
1	0	1

So, if a recessive bit is being transmitted while a dominant bit is sent, the dominant bit is displayed, evidence of a collision. A dominant bit is asserted by creating a voltage across the wires while a recessive bit is simply not asserted on the bus. If any node sets a voltage difference, all nodes will see it. Thus there is no delay to the higher priority messages, and the node transmitting the lower priority message automatically attempts to re-transmit six bit clocks after the end of the dominant message.

When used with a differential bus, a carrier sense multiple access/bitwise arbitration (CSMA/BA) scheme is often implemented: if two or more devices start transmitting at the same time, there is a priority based arbitration scheme to decide which one will be granted permission to continue transmitting.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting. Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID with dominant as binary 0, starting from the high bit. As soon as their ID is a larger number (lower priority) they will be

sending 1 (recessive) and see 0 (dominant), so they back off. At the end of ID transmission, all nodes but one have backed off, and the highest priority message gets through unimpeded.

For example, consider an 11-bit ID CAN network, with two nodes with IDs of 15 (binary representation, 00000001111) and 16 (binary representation, 00000010000). If these two nodes transmit at the same time, each will transmit the first six zeros of their ID with no arbitration decision being made. When the 7th bit is transmitted, the node with the ID of 16 transmits a 1 (recessive) for its ID, and the node with the ID of 15 transmits a 0 (dominant) for its ID. When this happens, the node with the ID of 16 will realize that it lost its arbitration, and allow the node with ID of 15 to continue its transmission. This ensures that the node with the lower bit value will always win the arbitration. The ID with the smaller number will win the right to use.

Message IDs must be unique on a single CAN bus, otherwise two nodes would continue transmission beyond the end of the arbitration field (ID) causing an error.

Bit timing

Each node in a CAN network has its own clock, and no clock is sent during data transmission. Synchronization is done by dividing each bit of the frame into a number of segments: synchronization, propagation, phase 1 and phase 2. The length of each *phase* segment can be adjusted based on network and node conditions. The sample point falls between phase buffer segment 1 and phase buffer segment 2, which helps facilitate continuous synchronization. Continuous synchronization in turn enables the receiver to be able to properly read the messages.

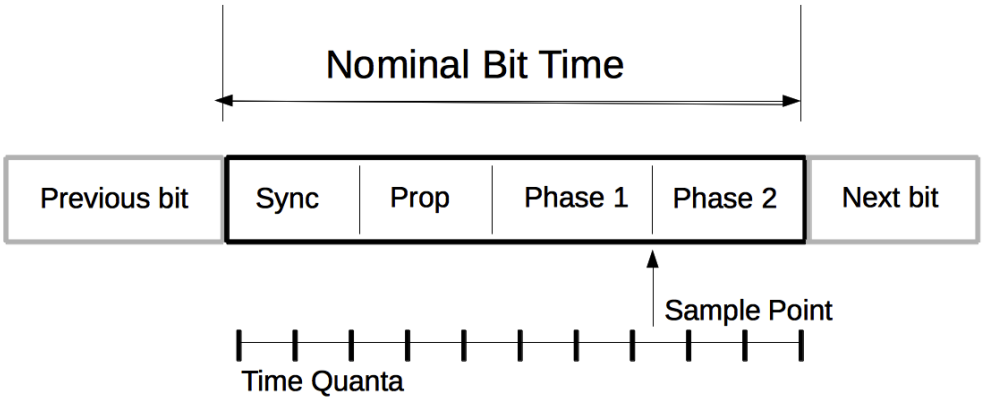


Figure 2: CAN bit timing example with 10 time quanta per bit

Layers

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

Application Layer
Object Layer
Transfer Layer
Physical 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 signalling, and fault confinement. It performs:

- Fault Confinement
- Error Detection
- Message Validation
- Acknowledgement
- Arbitration
- Message Framing
- Transfer Rate and Timing
- Information Routing

The absence of a complete physical layer specification (mechanical in addition to electrical) freed the CAN bus specification from the constraints and complexity of physical implementation. However it left CAN bus implementations open to inter-interoperability issues due to mechanical incompatibility.

Frames

A CAN network can be configured to work with two different message formats: the standard or base frame format and the extended frame format. The only difference between the two formats is that the *CAN base frame* supports a length of 11 bits for the identifier, and the *CAN extended frame* supports a length of 29 bits for the identifier, made up of the 11-bit identifier and an 18-bit extension. The distinction between CAN base frame format and CAN extended frame format is made by using the IDE bit, which is transmitted as dominant in case of an 11-bit frame, and transmitted as recessive in case of a 29-bit frame. CAN controllers that support extended frame format messages are also able to send and receive messages in CAN base frame format. All frames begin with a start-of-frame (SOF) bit that denotes the start of the frame transmission.

CAN has four frame types:

- Data frame: a frame containing node data for transmission
- Remote frame: a frame requesting the transmission of a specific identifier
- Error frame: a frame transmitted by any node detecting an error
- Overload frame: a frame to inject a delay between data and/or remote frame

Data frame

The data frame is the only frame for actual data transmission. There are two message formats:

- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

The CAN standard requires the implementation must accept the base frame format and may accept the extended frame format, but must tolerate the extended frame format.

Base frame format

CAN-Frame in base format with electrical levels without stuffbits

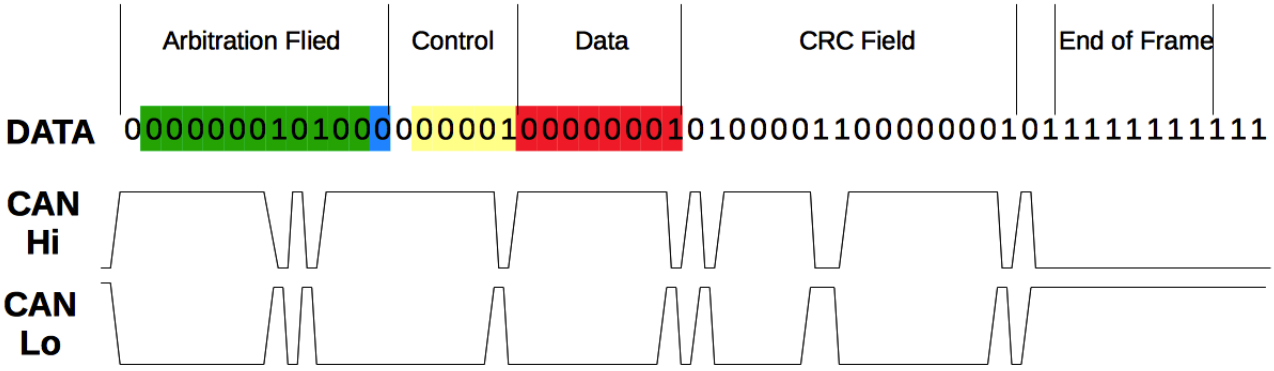


Figure 3: Can frame example

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier (green)	11	A (unique) identifier for the data which also represents the message priority
Remote transmission request (RTR)	1	Dominant (0)
Identifier extension bit (IDE)	1	Must be dominant (0). Optional
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)* (yellow)	4	Number of bytes of data (0–8 bytes)
Data field (red)	0–64 (0-8 bytes)	Data to be transmitted (length in bytes dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

Extended frame format

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier A	11	First part of the (unique) identifier for the data which also represents the message priority
Substitute remote request (SRR)	1	Must be recessive (1). Optional
Identifier extension bit (IDE)	1	Must be recessive (1). Optional

Identifier B	18	Second part of the (unique) identifier for the data which also represents the message priority
Remote transmission request (RTR)	1	Must be dominant (0)
Reserved bits (r0, r1)	2	Reserved bits (it must be set dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)*	4	Number of bytes of data (0–8 bytes)
Data field	0–64 (0-8 bytes)	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

The two identifier fields (A & B) combine to form a 29-bit identifier.

Remote frame

- Generally data transmission is performed on an autonomous basis with the data source node sending out a Data Frame. It is also possible, however, for a destination node to request the data from the source by sending a Remote Frame.
- There are two differences between a Data Frame and a Remote Frame. Firstly the RTR-bit is transmitted as a dominant bit in the Data Frame and secondly in the Remote Frame there is no Data Field.

  In the very unlikely event of a Data Frame and a Remote Frame with the same identifier being transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the desired data immediately.

Error frame

The error frame consists of two different fields:

- The first field is given by the superposition of *ERROR FLAGS* (6–12 dominant/recessive bits) contributed from different stations.
- The following second field is the *ERROR DELIMITER* (8 recessive bits).

There are two types of error flags:

- Active Error Flag (6 dominant bits): Transmitted by a node detecting an error on the network that is in error state "error active".
- Passive Error Flag (6 recessive bits): Transmitted by a node detecting an active error frame on the network that is in error state "error passive".

Overload frame

The overload frame contains the two bit fields Overload Flag and Overload Delimiter. There are two kinds of overload conditions that can lead to the transmission of an overload flag:

1. The internal conditions of a receiver, which requires a delay of the next data frame or remote frame.

2. Detection of a dominant bit during intermission.

Overload Flag consists of six dominant bits. The overall form corresponds to that of the active error flag. The overload flag's form destroys the fixed form of the intermission field. As a consequence, all other stations also detect an overload condition and on their part start transmission of an overload flag.

Overload Delimiter consists of eight recessive bits. The overload delimiter is of the same form as the error delimiter.

CAN Exercises:
In order to study the CAN interface, we will start with the examples in the ToolChain, Later you should complete exercises depicted below. Files where CAN interface is defined are:

- ttc_can.h
- ttc_can_types.h
- ttc_can.c

You should activate the CAN library, via *activate_project.sh* script, as described in the previous sessions. You can start using the CAN example in the ToolChain, but later you need to write your own source code to complete the exercises.

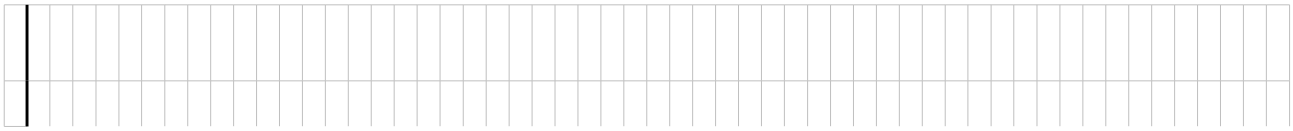
To connect to the CAN port of Olimex P103 (or with an adapter in Olimex LCD and Olimex P107), you need to locate the CAN interface. You should review the schematic from Olimex, and identify the mapping of these pins (you can find these schematics at Olimex website.
Please use the following parameters to configure your UART interface on the virtual terminal: 115200 bps, no parity, 8 bits, 1 stop bit and no flow control.
You will need to connect the interface and finally you should receive signals from the board on the host PC (compare the following Figure).

To manage several interfaces at same time, it is necessary to use several tasks. If you remember, we did it in the last session. If not, you can practice this with *example_threads_queues*, this example can give you the keys to work with multiple interfaces and resources. Read it and try to understand it.

1. In this exercise you should manage two interfaces at same time. You need to manage USART and CAN interfaces. You should configure USART interface like main system output, in order to print everything on your virtual terminal. At same time you should configure your CAN interface and send exactly the same message via CAN. To pass this exercise you need to recognize your message on the oscilloscope (explaining the CAN signals). The message that you send is arbitrary (not more than 8 bytes), but you should not copy from other groups.

2. For the second exercise you need to receive from other boards via CAN interface. It is necessary to configure a filter and receive messages from different groups. These messages should be sent using the program that we did it before. It means that they will type a message on a virtual terminal (in other PC) and you should be able to receive that message via CAN, later you should send it to your PC, via USART interface.

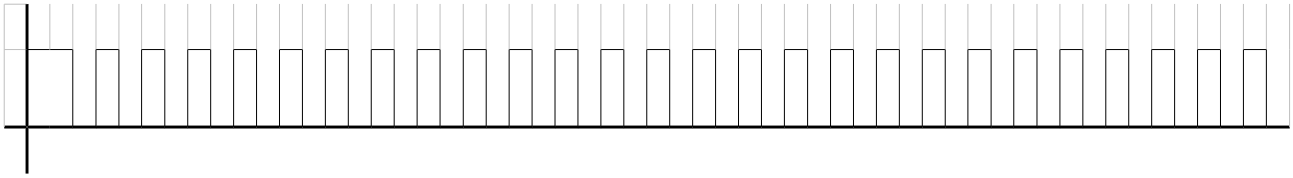
Time division value: _____ Data line: _____



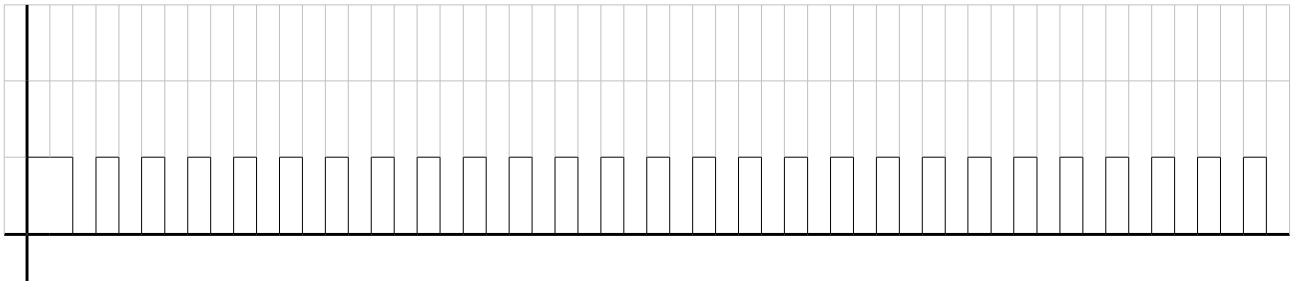
Fachhochschule Münster University of Applied Sciences

Lab for Semiconductor Devices and Bussystems

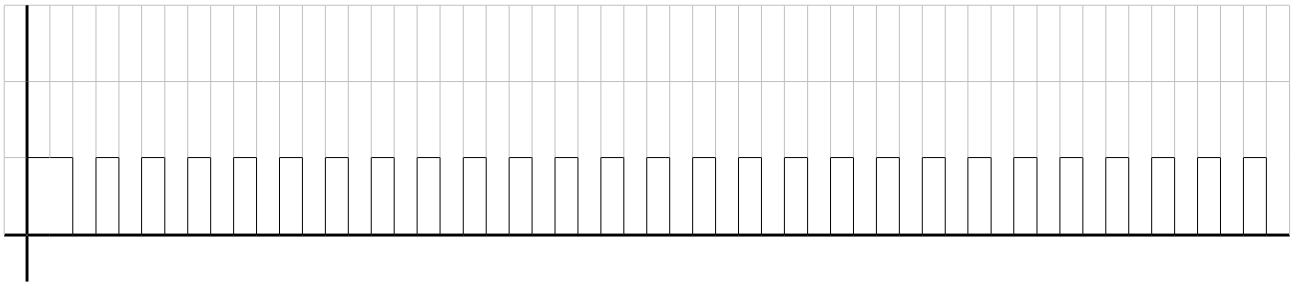
Exercise Bussystems SS2012/13
Tutor: Dipl. Informatiker Gregor Rebel
MSc. Francisco Estevez



Time division value: _____ Data line: _____



Time division value: _____ Data line: _____



String sent: _____ Computer Number: _____

Information from www.en.wikipedia.org Group: A | B | C

Name: _____ Name: _____

Sign: _____ Sign: _____