CAN BUS

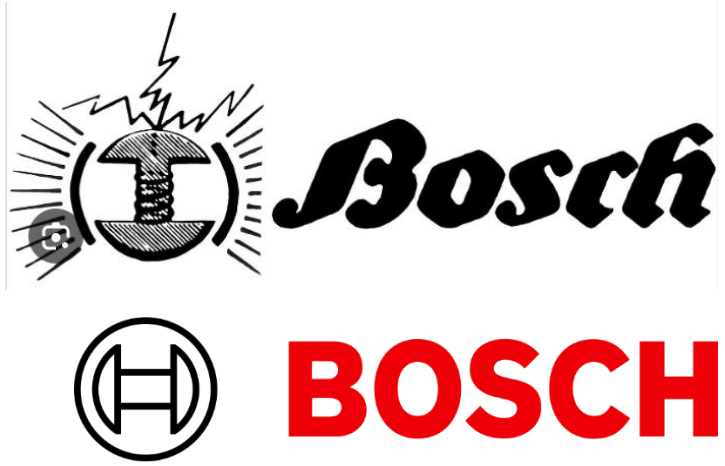
Paolo Bernardi

Controller Area Network - CAN bus

- A Controller Area Network (CAN bus) is a bus standard that allows microcontrollers and devices to communicate with each other's applications without a host computer.
- It is a message-based protocol designed originally for multiplex electrical wiring within automobiles to save on copper, but it can also be used in many other contexts.
- For each device, the data in a frame is transmitted serially but in such a way that if more than one device transmits simultaneously, the highest priority device can continue while the others back off.
- Frames are received by all devices, including the transmitting device.

History[1]

- Development of the CAN bus started in **1983** at Robert Bosch GmbH.
- The first CAN controller chips were introduced by Intel in 1987, and shortly thereafter by Philips.



- Released in 1991, the Mercedes-Benz W140 was the first production vehicle to feature a CAN-based multiplex wiring system.

[1] "CAN History". CAN in Automation. http://www.can-cia.org/can-knowledge/can/can-history/

Documentation[2]

- Bosch published several versions of the CAN specification. The latest is CAN 2.0, published in 1991.
- This specification has two parts.
  - Part A is for the standard format with an 11-bit identifier, and
  - part B is for the extended format with a 29-bit identifier.
- A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A, and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.
- These standards are freely available from Bosch, along with other specifications and white papers.

[2] "Bosch Semiconductor CAN Literature".
https://web.archive.org/web/20170523083421/http:/www.bosch-semiconductors.de/en/automotive_electronics/ip_modules/can_literature_2.html

Applications

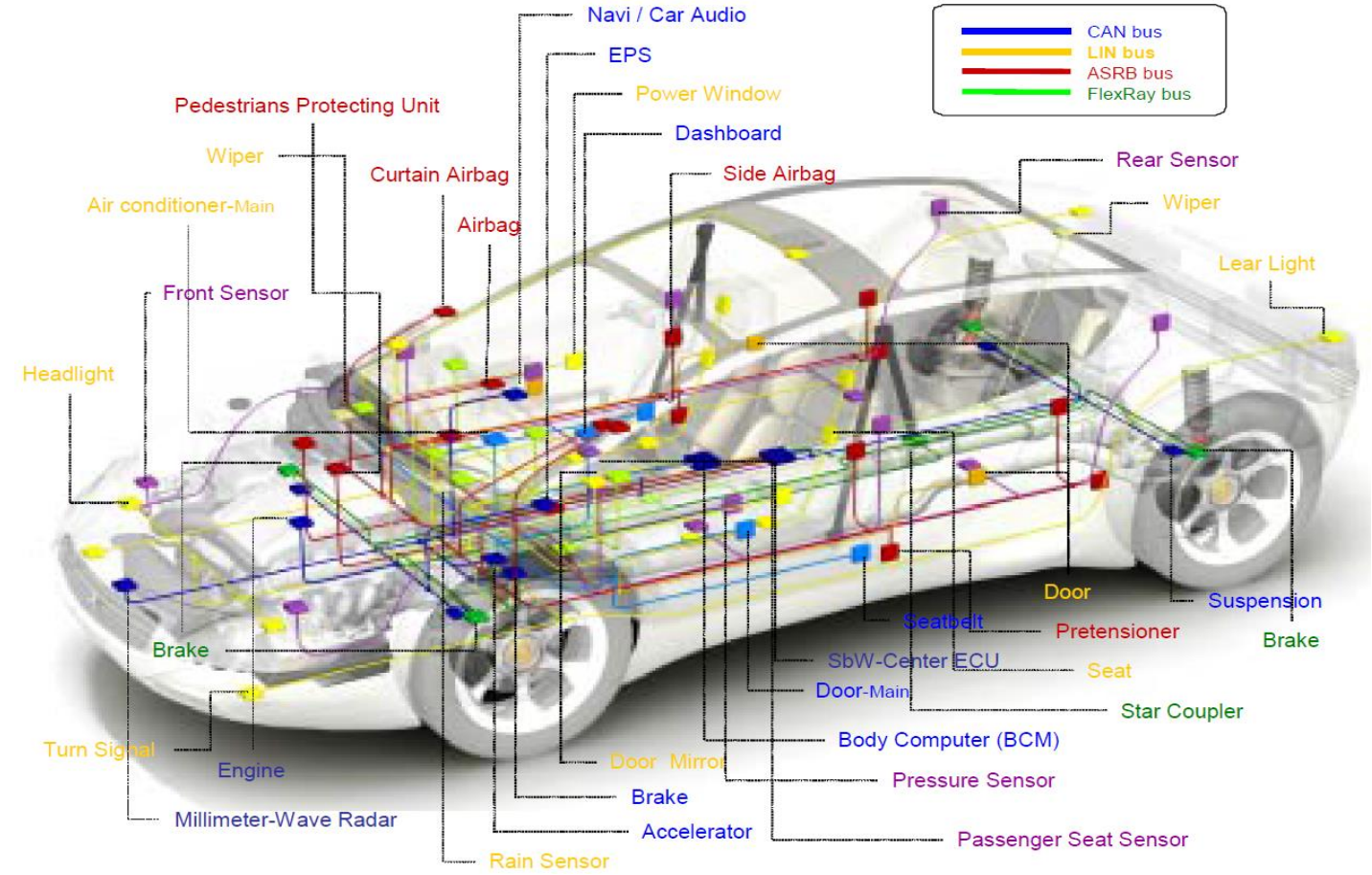
- Passenger vehicles, trucks, buses (combustion vehicles and electric vehicles)
- Agricultural equipment
- Electronic equipment for aviation and navigation
- Industrial automation and mechanical control
- Elevators, escalators
- Building automation
- Medical instruments and equipment
- Pedelecs
- Model railways/railroads
- Ships and other maritime applications
- Lighting control systems
- 3D Printers
- Robotics/Automation

Automotive field

- The modern automobile may have as many as 70 electronic control units (ECU) for various subsystems.
- Traditionally, the biggest processor is the engine control unit.
- Others are used for Autonomous Driving, Advanced Driver Assistance System (ADAS), transmission, airbags, antilock braking/ABS, cruise control, electric power steering, audio systems, power windows, doors, mirror adjustment, battery and recharging systems for hybrid/electric cars, etc.
- Some of these form independent subsystems, but communication among others is essential.
- A subsystem may need to control actuators or receive feedback from sensors.
- The CAN standard was devised to fill this need.

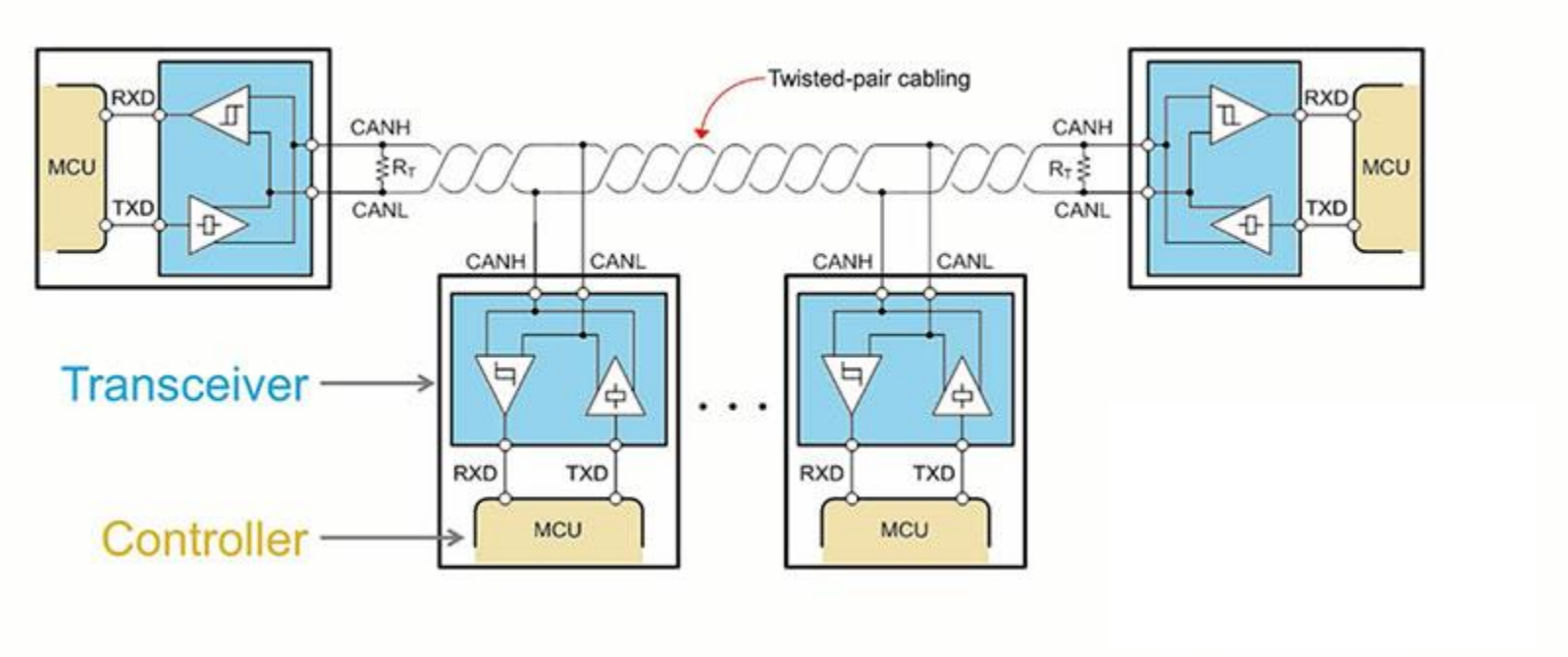
Key advantages

- CAN bus is empowered with **software alone** – functionalities. Cheaper than implemented hard-wired using traditional automotive electrics.
- Main advantages:
  - Low cost
  - Centralized
  - Robust
  - Efficient.



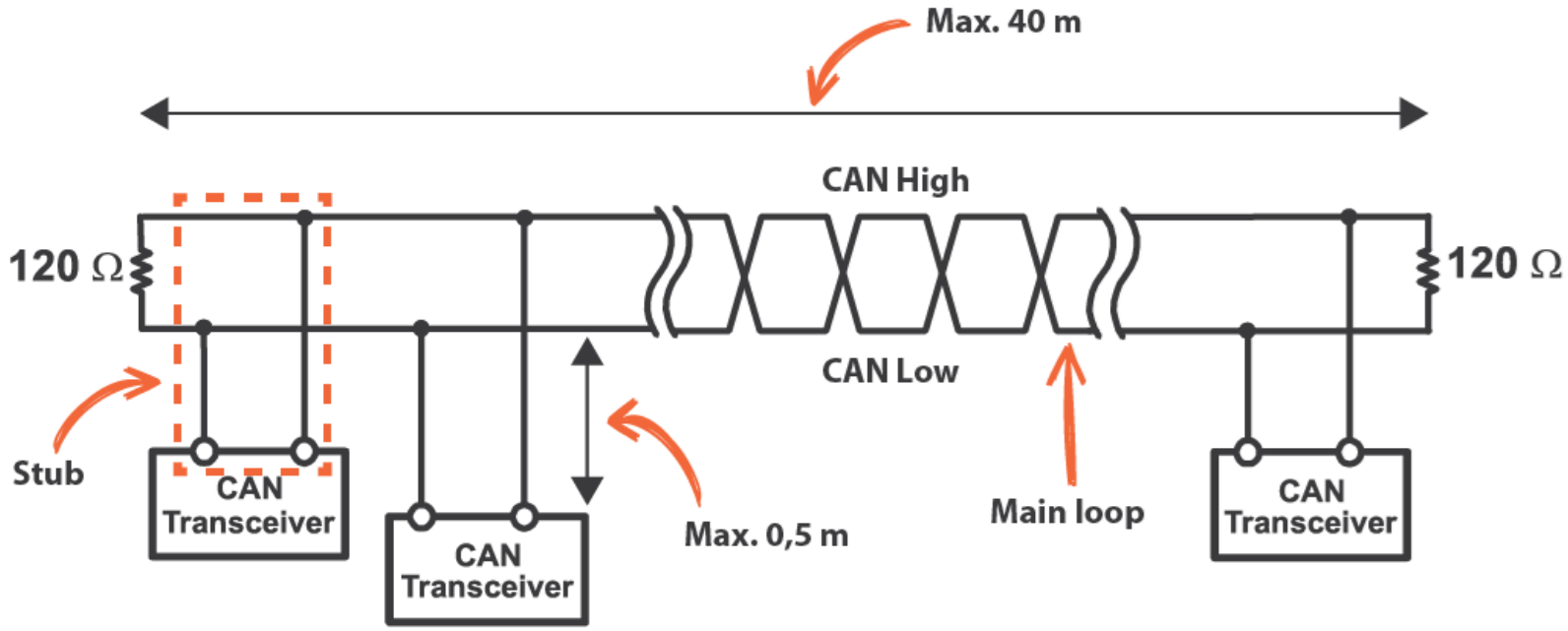
Physical organization

- CAN is a multi-master serial bus standard for connecting electronic control units (ECUs) also known as nodes.



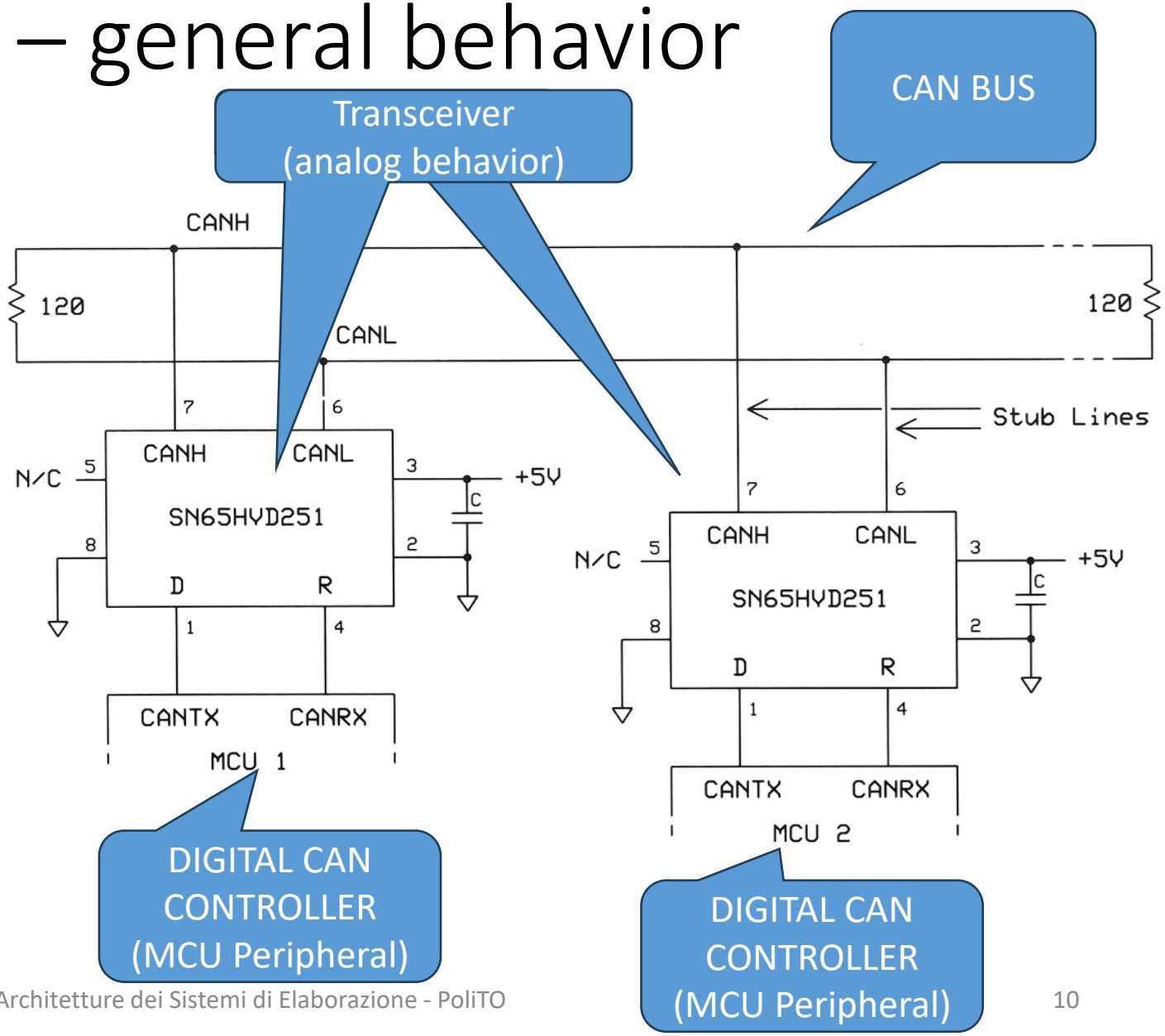
Physical organization (II)

- All nodes are connected to each other through a physically conventional two-wire bus.
- The wires are a twisted pair with a 120 Ω (nominal) characteristic impedance.



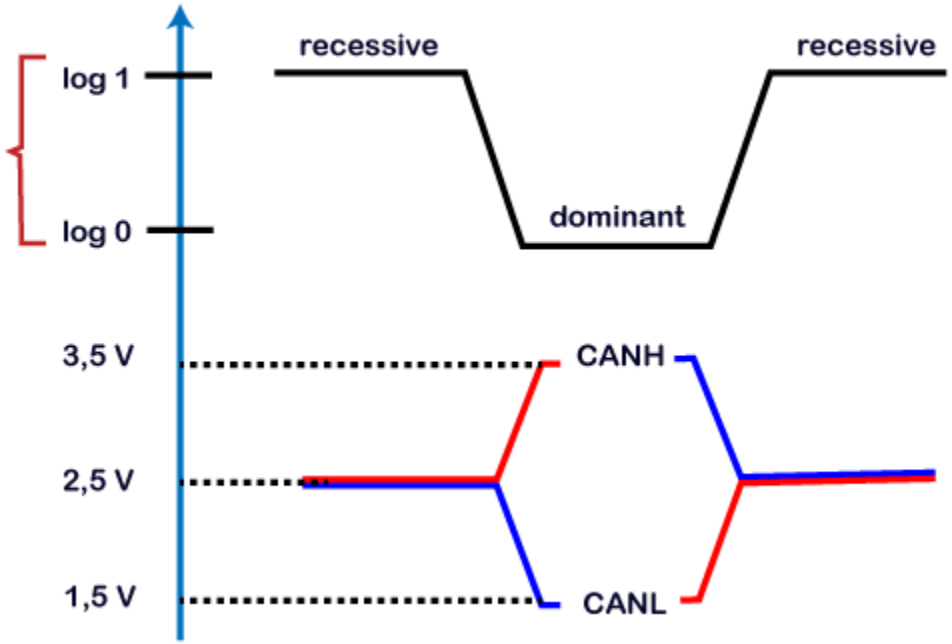
Two-wire CAN bus – general behavior

- MCU CAN Controller communicates with (sends and receives data to and from) a transceiver
- The transceiver
  - converts **a (single) logic value** from the MCU **into a couple of electric signals** to drive to the two-wire bus
  - Reads a couple of values on the two-wire bus and returns a logic value to the MCU



Two-wire CAN bus – electric and logical values

- This bus uses differential wired-AND signals.
- Two signals, CAN high (CANH) and CAN low (CANL) are either driven to
  - A "**dominant**" state "**logic 0**" with CANH > CANL,
  - Or not driven and pulled by passive resistors to a "**recessive**" state "**logic 1**"with CANH ≤ CANL.
- A 0 data bit encodes a dominant state, while a 1 data bit encodes a recessive state, supporting a wired-AND convention.

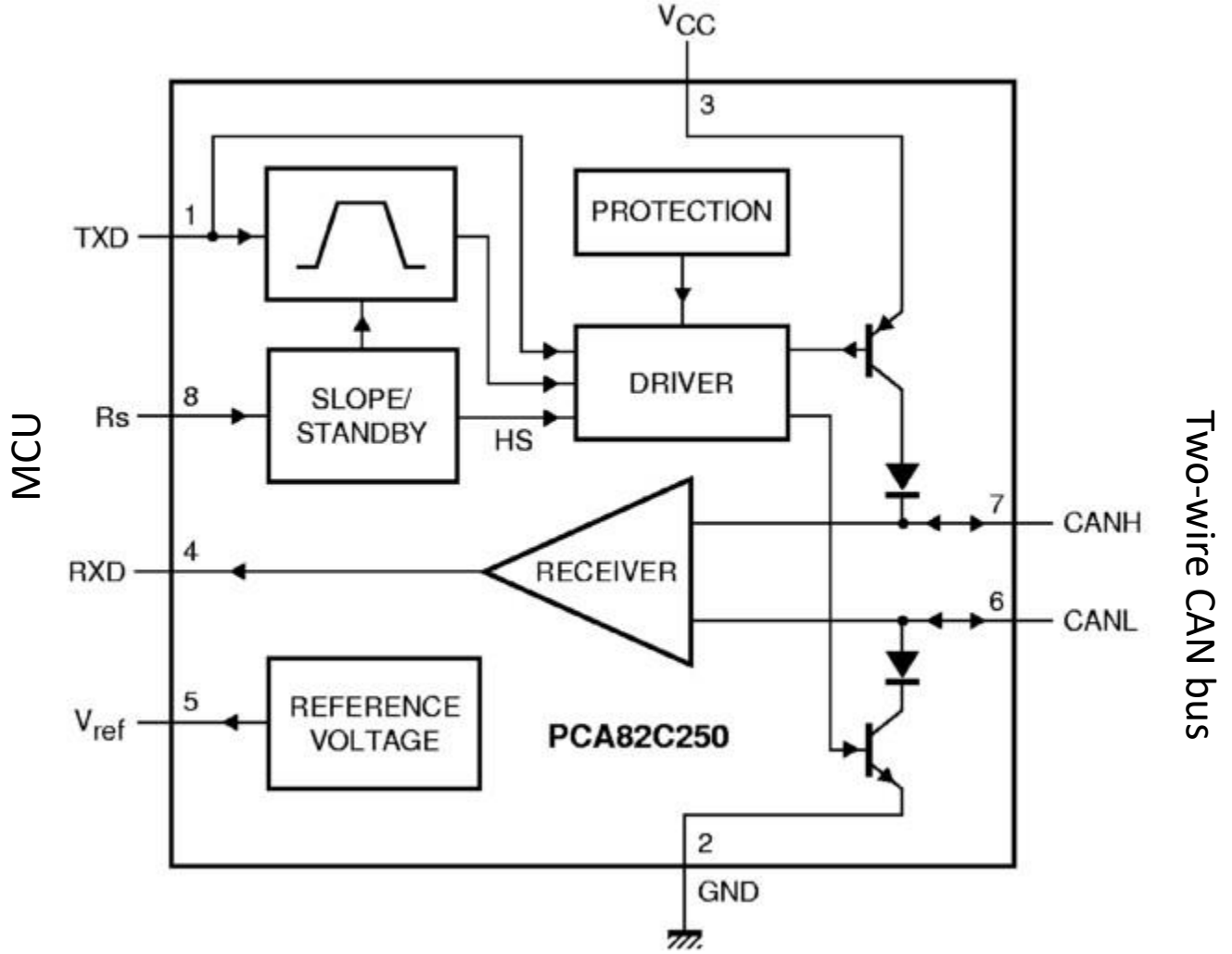


Output of the CAN Transceiver

Simple and mindblowing concept

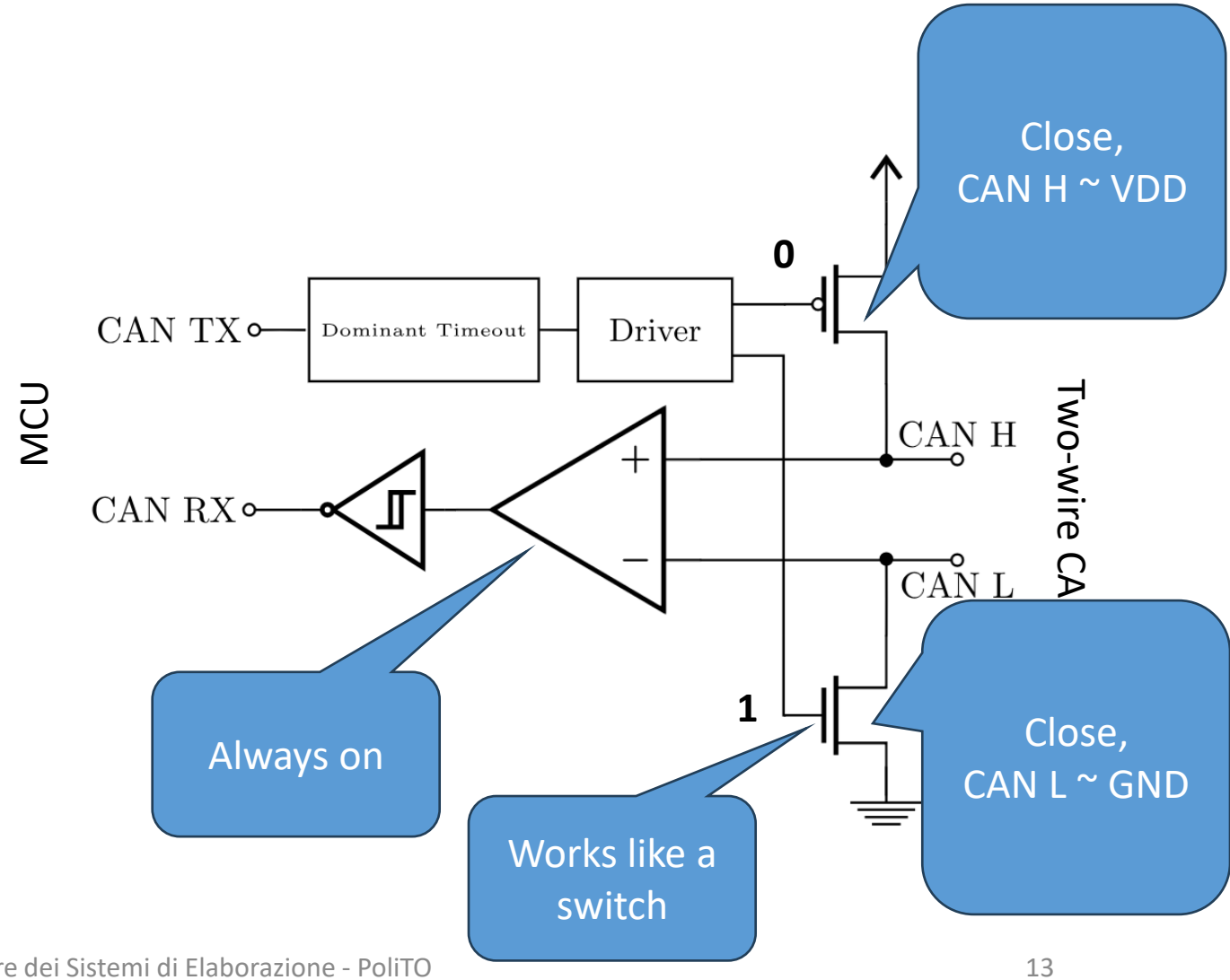
Two-wire CAN bus – transceiver schema

- 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.



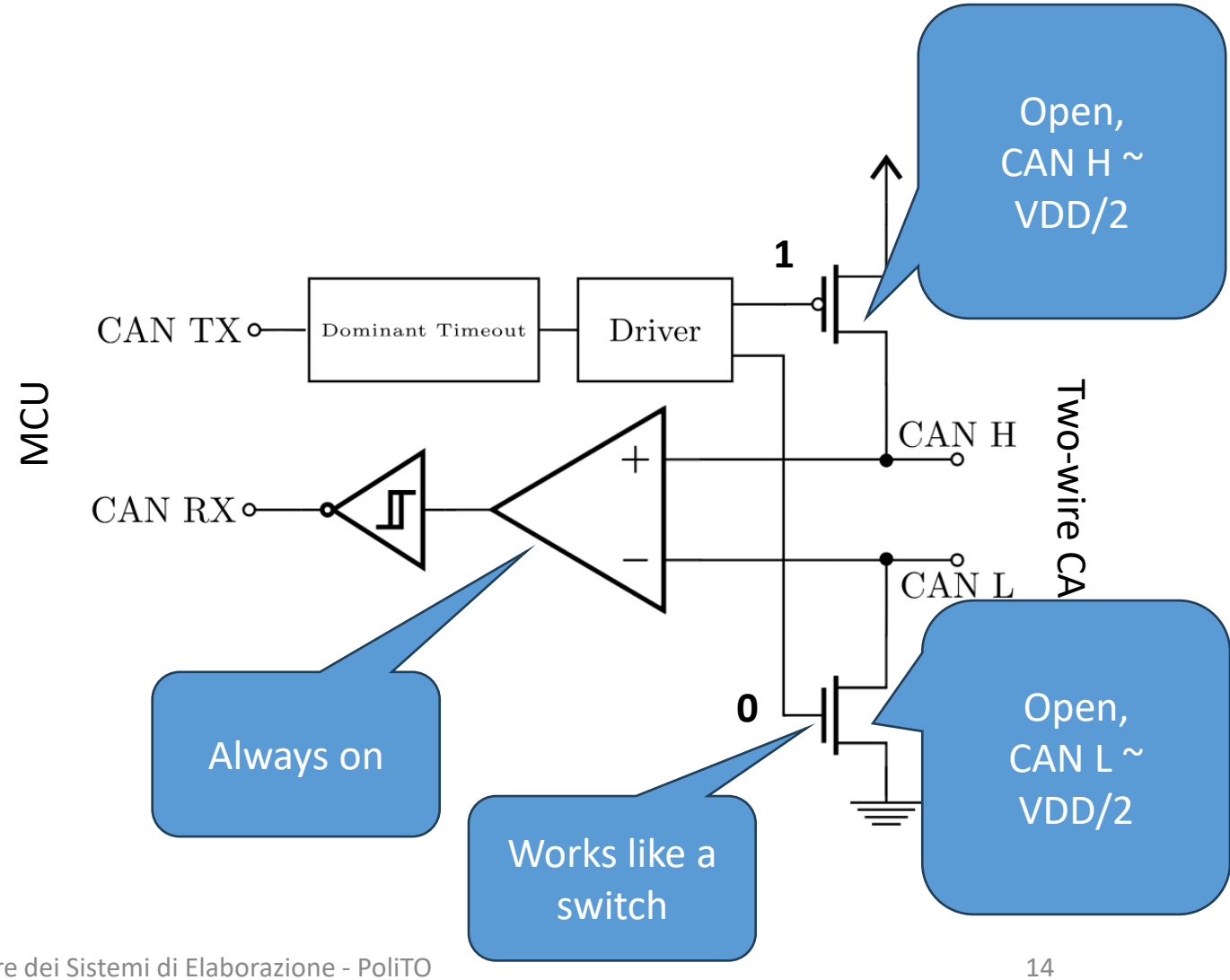
Two-wire CAN bus – **TX a logic 0**

- A "**dominant**" state "**logic 0**" with CANH > CANL
- Both MOSfets are Closed
  - **CAN H →HIGH**
  - **CAN L → LOW**
- The transceiver drives a 0 to the bus.

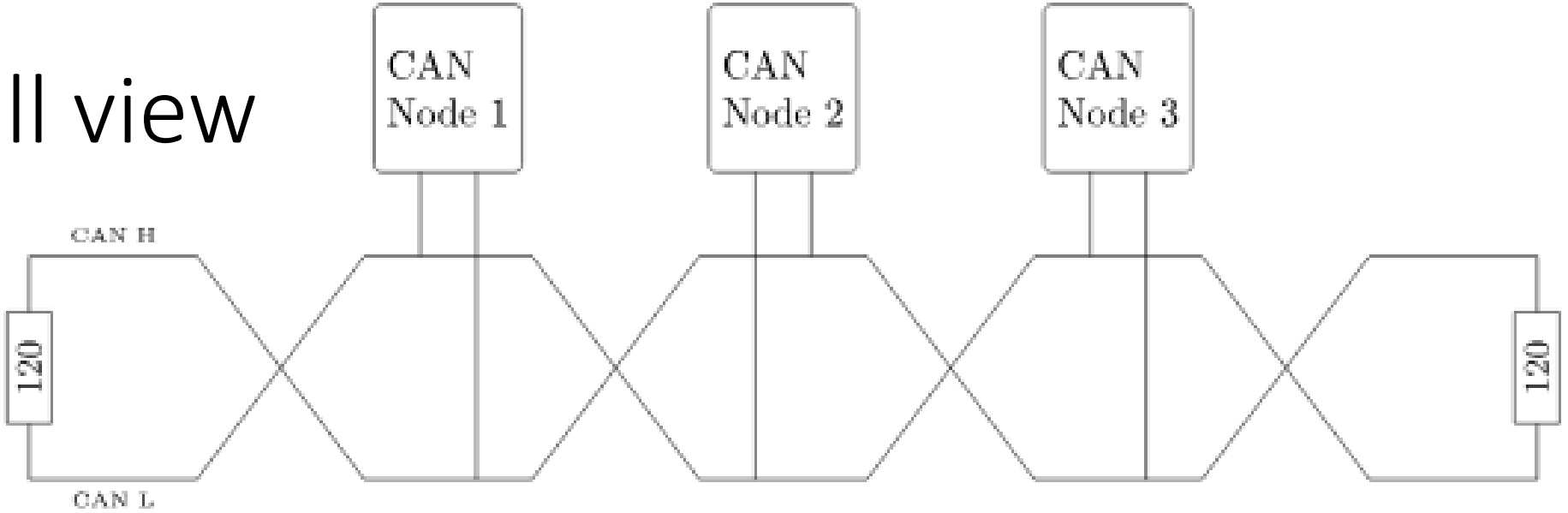


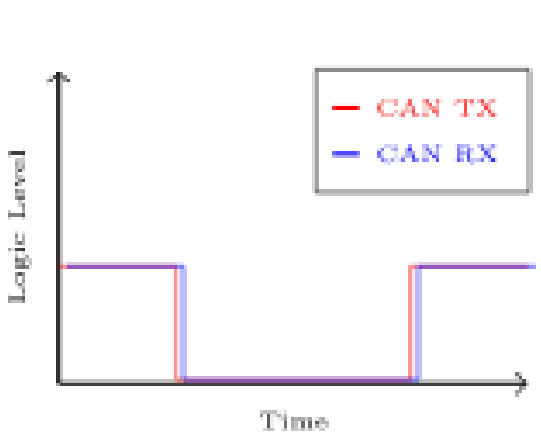
Two-wire CAN bus – **TX a logic 1**

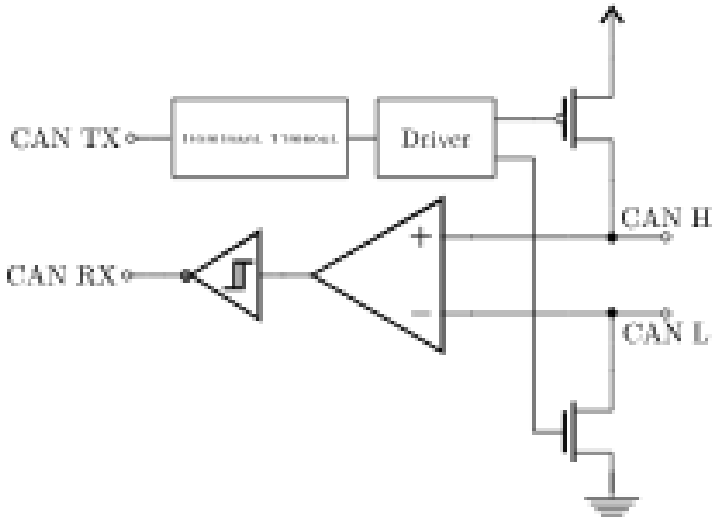
- A “**recessive**" state “**logic 1**” with CANH ≤ CANL,
- Both MOSfets are Open
  - **CAN H** and **CAN L** are driven by the BUS.
- The transceiver drives a 1 to the bus.



Overall view







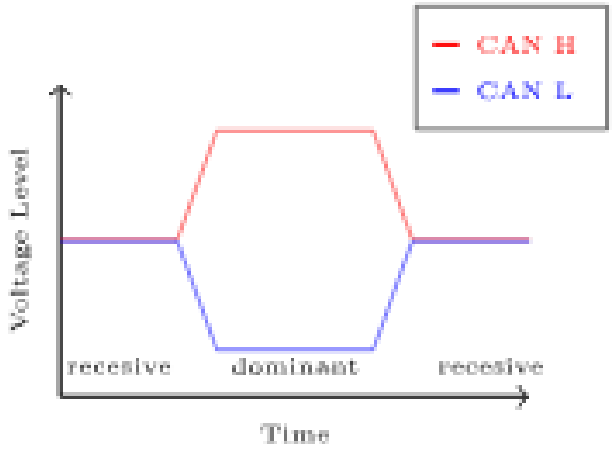
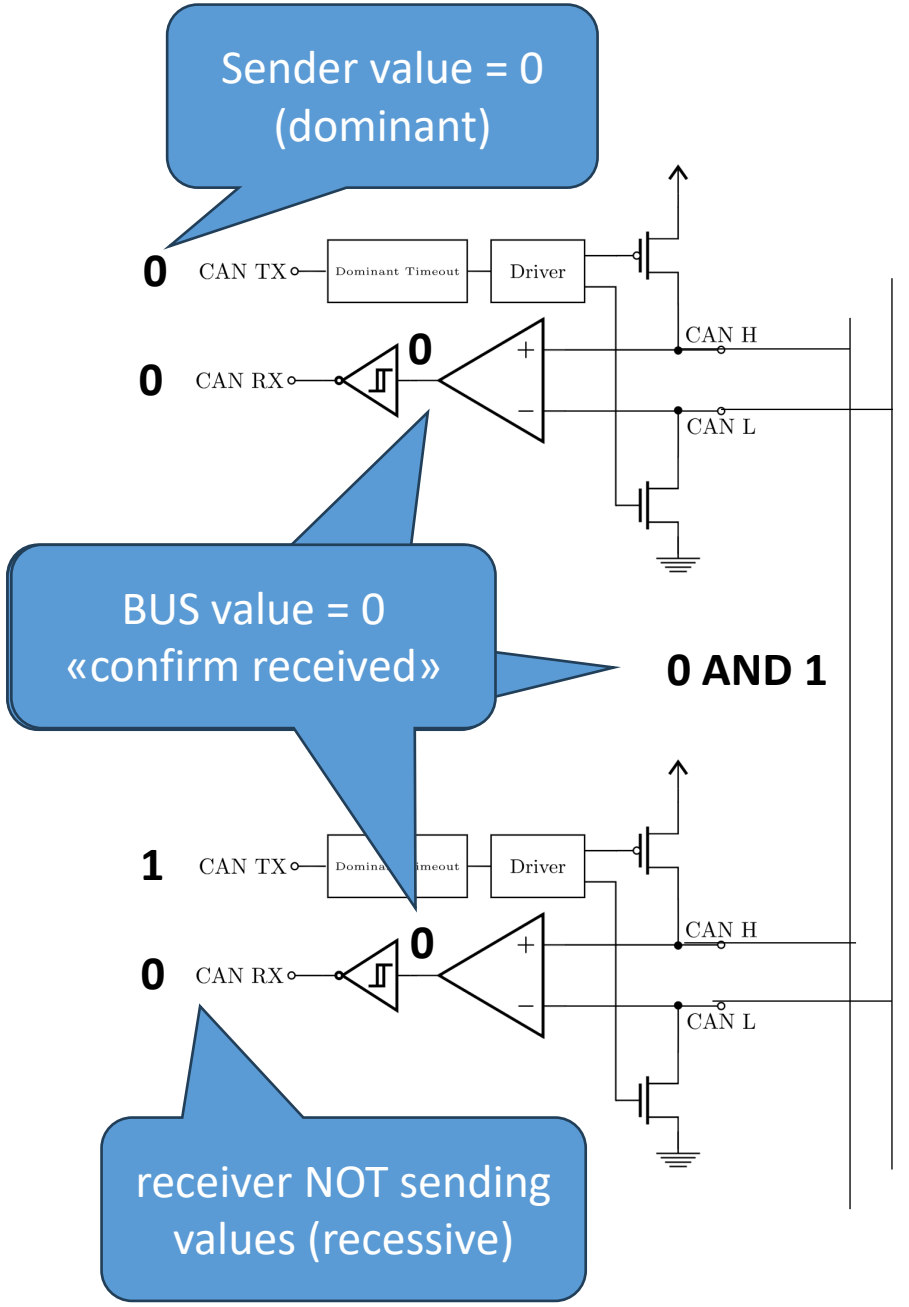


Figure 1: CAN Transceiver, from Logic Levels to Bus Levels. ©Alexander Wachter

Who is sending?
Who is receiving?

- All nodes are sending something,
  - «talking» nodes are sending either dominant or regressing values
  - «listening» nodes are sending only recessive values
- Thus, all CAN nodes in the system, including the sender node, receive the transmitted data, and possibly interact with it
- EXAMPLE:
  - Sender sends dominant (0)
  - Receiver sends recessive (1)
  - Sender receives dominant (0)
  - Receiver receives dominant (0)



Usage modes

- Master-Slave
  - It is the easiest usage mode
  - There is a single node sending data to the others
- Multi-Master
  - Many nodes can send data simultaneously
  - Must deal with a bus contention issue

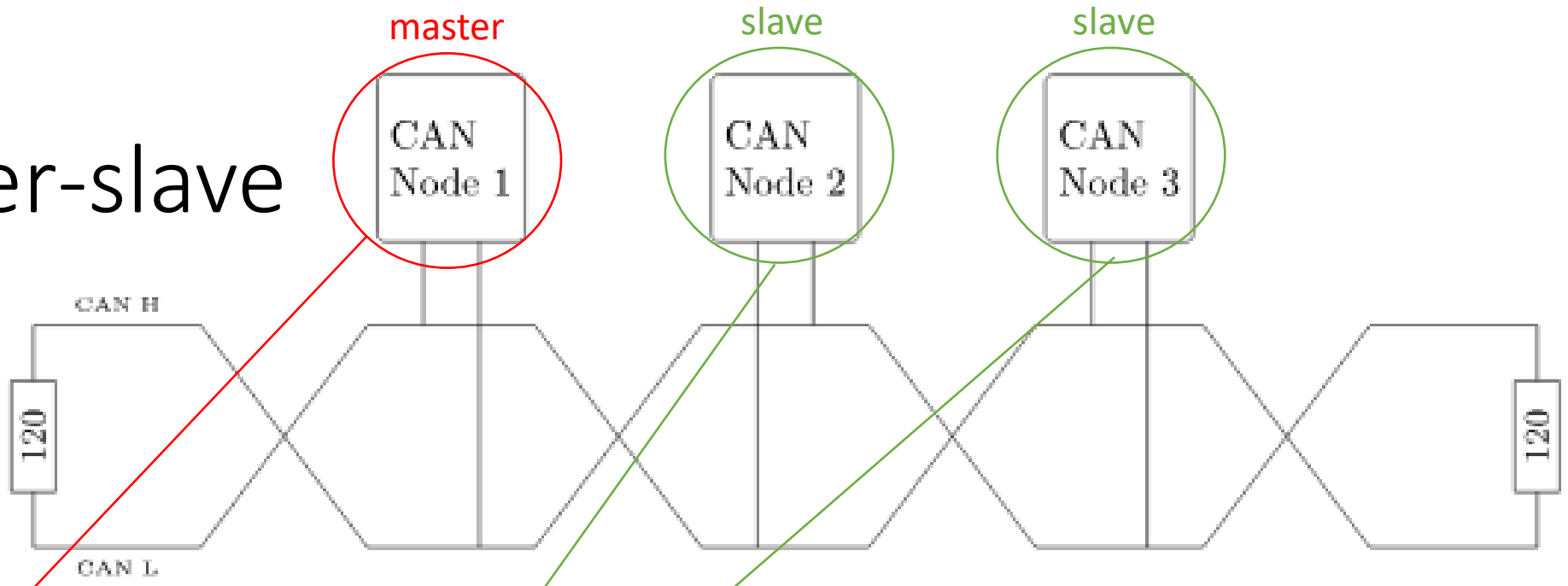
Usage modes

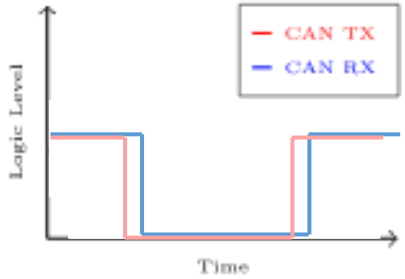
- Master-Slave
  - It is the easiest usage mode
  - There is a single node sending data to the others
- Multi-Master
  - Many nodes can send data simultaneously
  - Must deal with a bus contention issue.

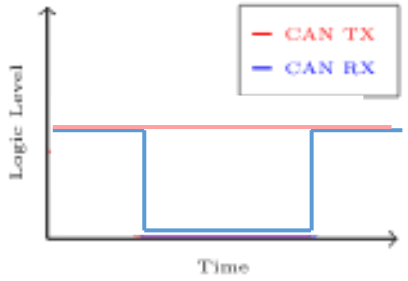
The content of the next slides directly addresses the subject of Master-Slave mode of CAN communication

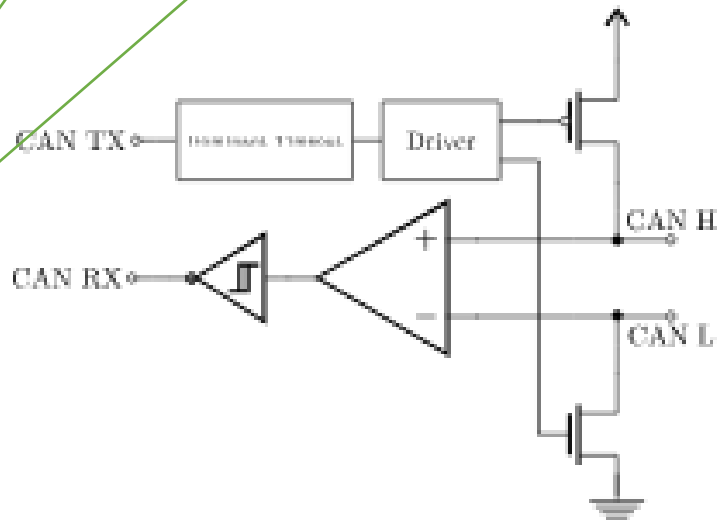
BUS arbitration methods, which are explained later, is base on the next.

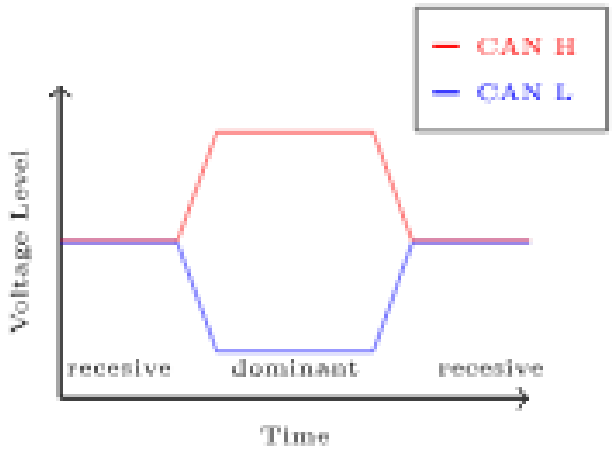
Master-slave









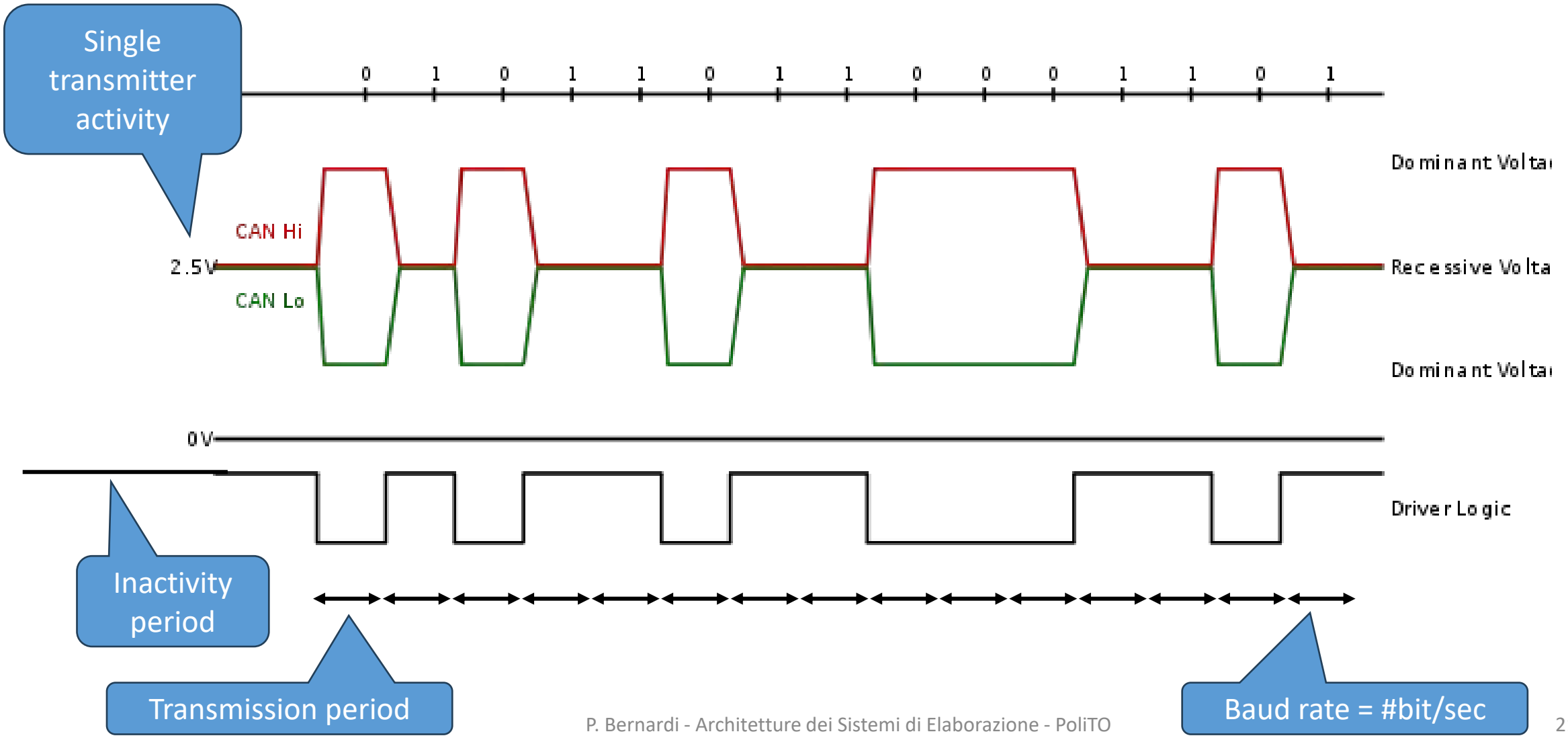


Master-Slave - Single or no transmission

- When any device is transmitting a dominant (logic value 0), high-speed CAN signaling drives the CANH wire towards 3.5 V and the CANL wire towards 1.5 V
- If no device is transmitting a dominant, the **terminating resistors** passively return the two wires to the recessive (logic value 1) state with a nominal differential voltage of 0 V.
  - Receivers consider any differential voltage of less than 0.5 V to be recessive.
  - The dominant differential voltage is a nominal 2 V.

Example of transmission

All nodes need to know the bus speed



Bit timing

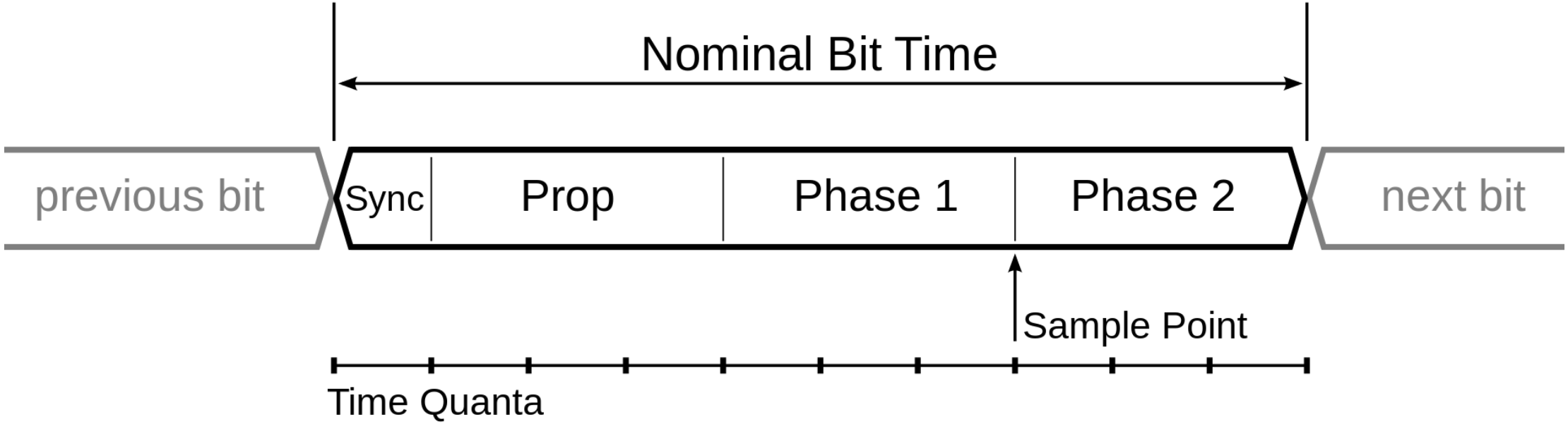
- All nodes on the CAN network **must operate at the same nominal bit rate**, but noise, phase shifts, oscillator tolerance and oscillator drift mean that the actual bit rate might not be the nominal bit rate.
- Since a separate clock signal is not used, a means of synchronizing the nodes is necessary.
- Synchronization is important during arbitration since the nodes in arbitration must be able to see both their transmitted data and the other nodes' transmitted data at the same time.
- Synchronization is also important to ensure that variations in oscillator timing between nodes do not cause errors.

Bit timing (II)

- Synchronization starts with a hard synchronization on the first recessive to dominant transition after a period of bus idle (the start bit).
  - Resynchronization occurs on every recessive to dominant transition during the frame.
  - The CAN controller expects the transition to occur at a multiple of the nominal bit time.
  - If the transition does not occur at the exact time the controller expects it, the controller adjusts the nominal bit time accordingly.
- The adjustment is accomplished by dividing each bit into a number of time slices called quanta and assigning some number of quanta to each of the four segments within the bit: synchronization, propagation, phase segment 1, and phase segment 2.

Bit timing (III)

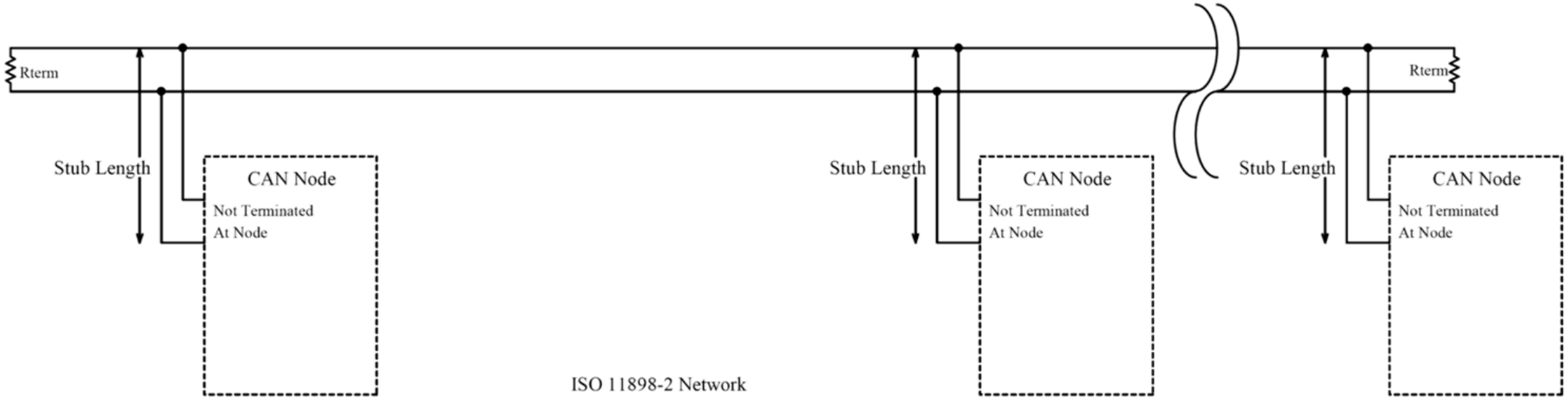
- The number of quanta, the bit is divided into, can vary by controller, and the number of quanta assigned to each segment can be varied depending on bit rate and network conditions.



Bit timing (IV)

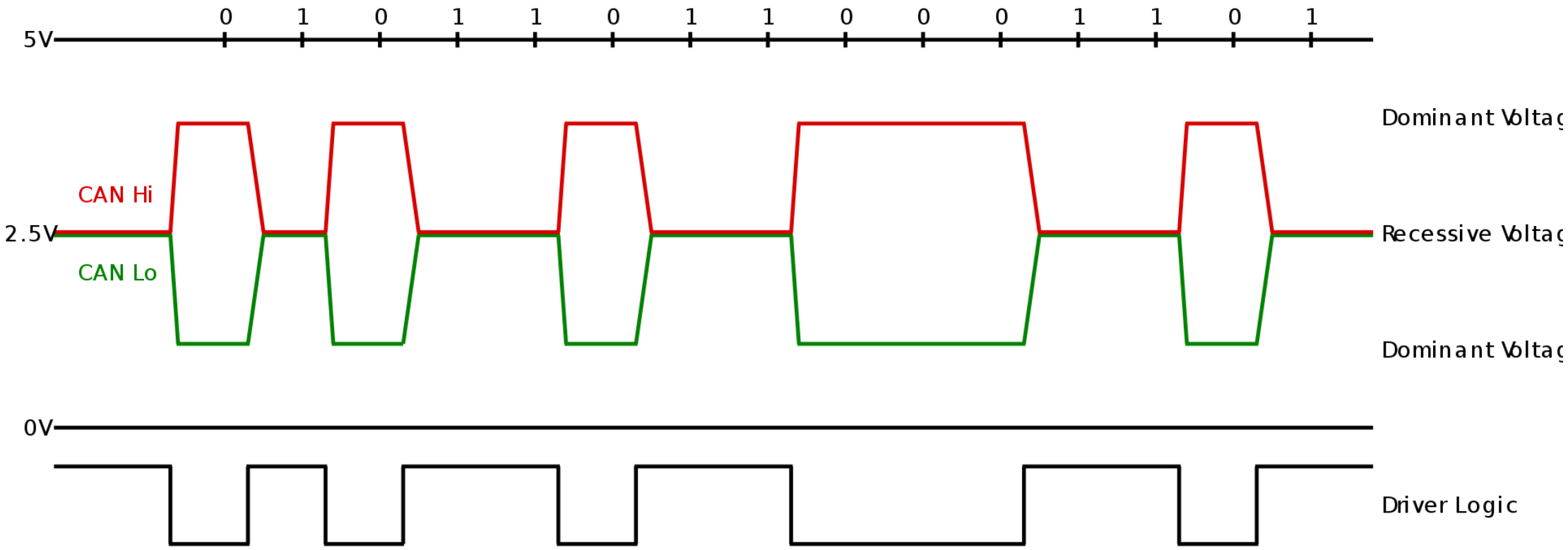
- A transition that occurs before or after it is expected causes the controller to calculate the time difference and lengthen phase segment 1 or shorten phase segment 2 by this time.
- This effectively adjusts the timing of the receiver to the transmitter to synchronize them.
- This resynchronization process continuously occurs at every recessive to dominant transition to ensure the transmitter and receiver stay in sync.
- Continuously resynchronizing reduces errors induced by noise and allows a receiving node that was synchronized to a node that lost arbitration to resynchronize to the node that won arbitration.

High-speed CAN network. ISO 11898-2.

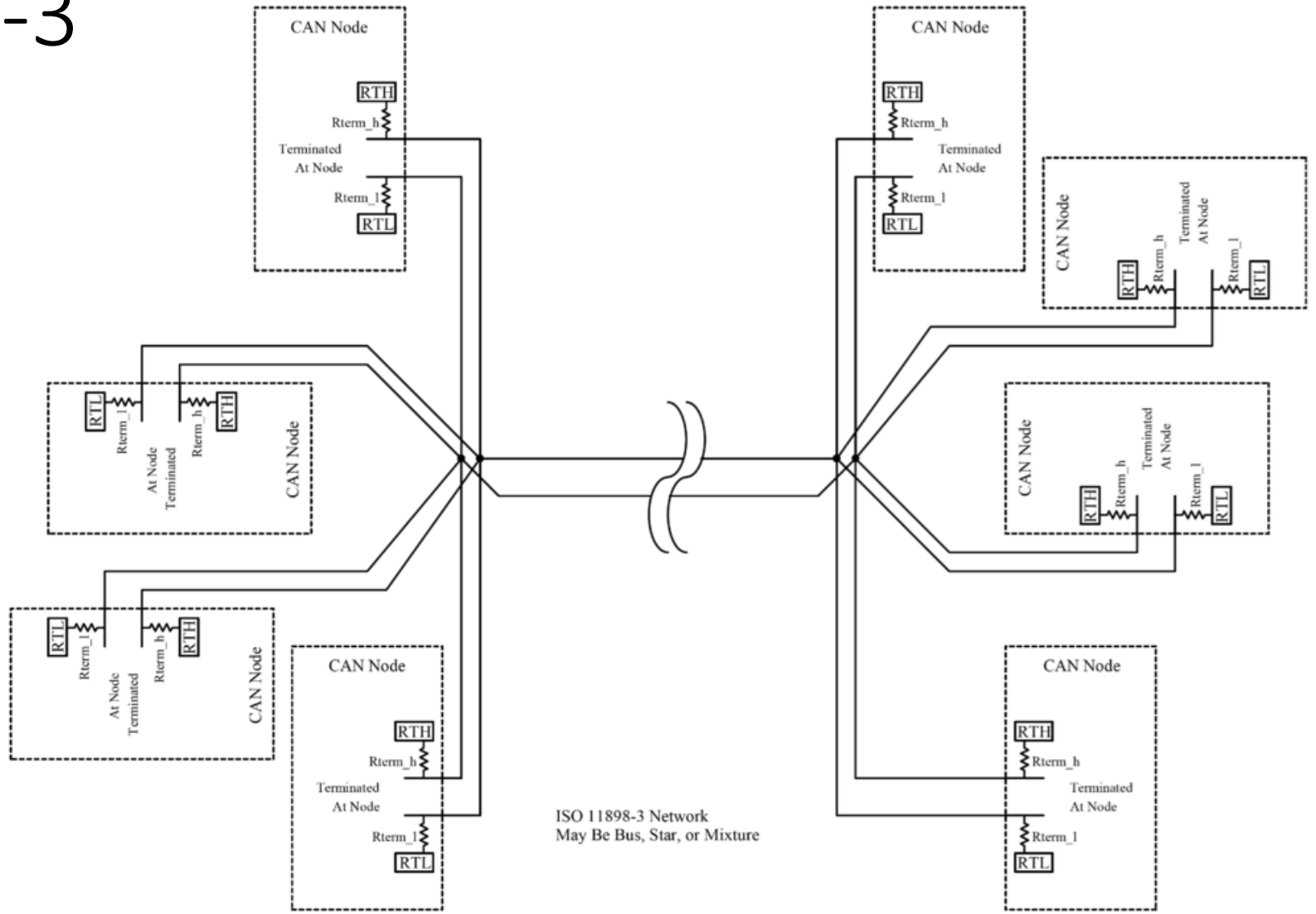


- Bit speeds up to 1 Mbit/s on CAN (5 Mbit/s on CAN-Full Duplex)
- Uses a linear bus terminated at each end with 120 Ω resistors.

High-speed CAN signaling. ISO 11898-2.



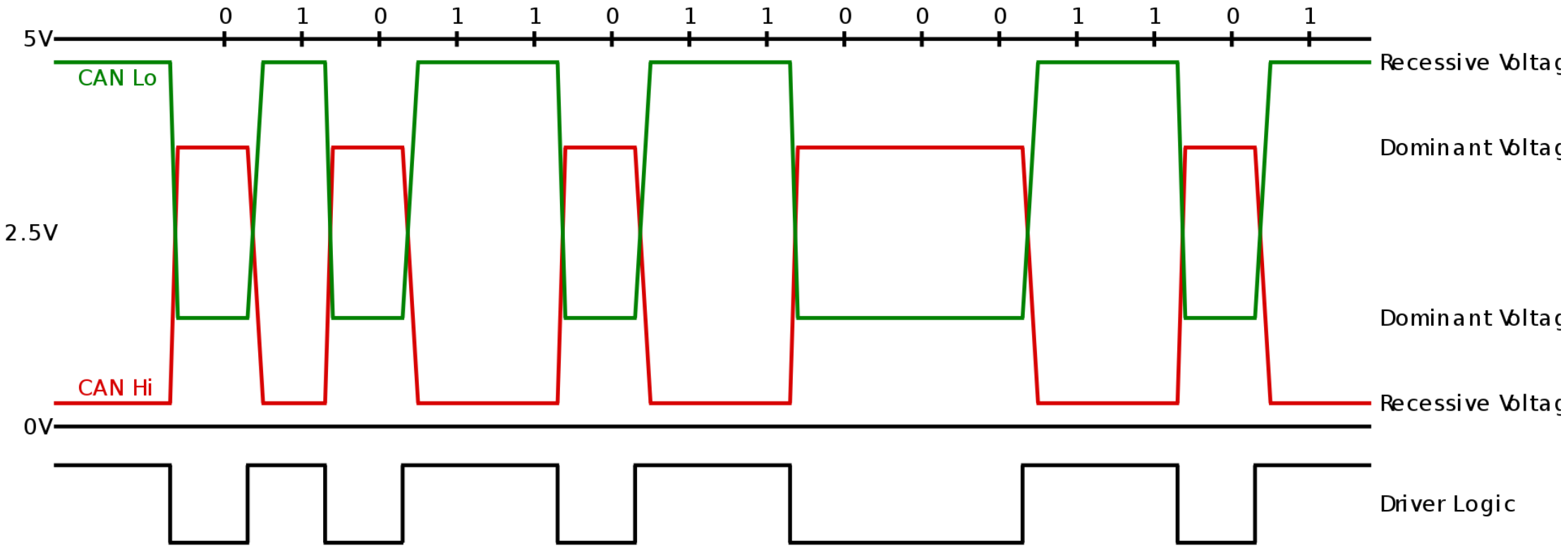
Low-speed fault-tolerant CAN network. ISO 11898-3



Low-speed fault-tolerant CAN network. ISO 11898-3

- **ISO 11898-3**, also called low-speed or fault-tolerant CAN
  - up to 125 kbit/s
  - uses
    - a linear bus
    - star bus
    - multiple star buses
  - connected by a linear bus
  - terminated at each node by a fraction of the overall termination resistance.
- The overall termination resistance should be close to, but not less than, 100 Ω.

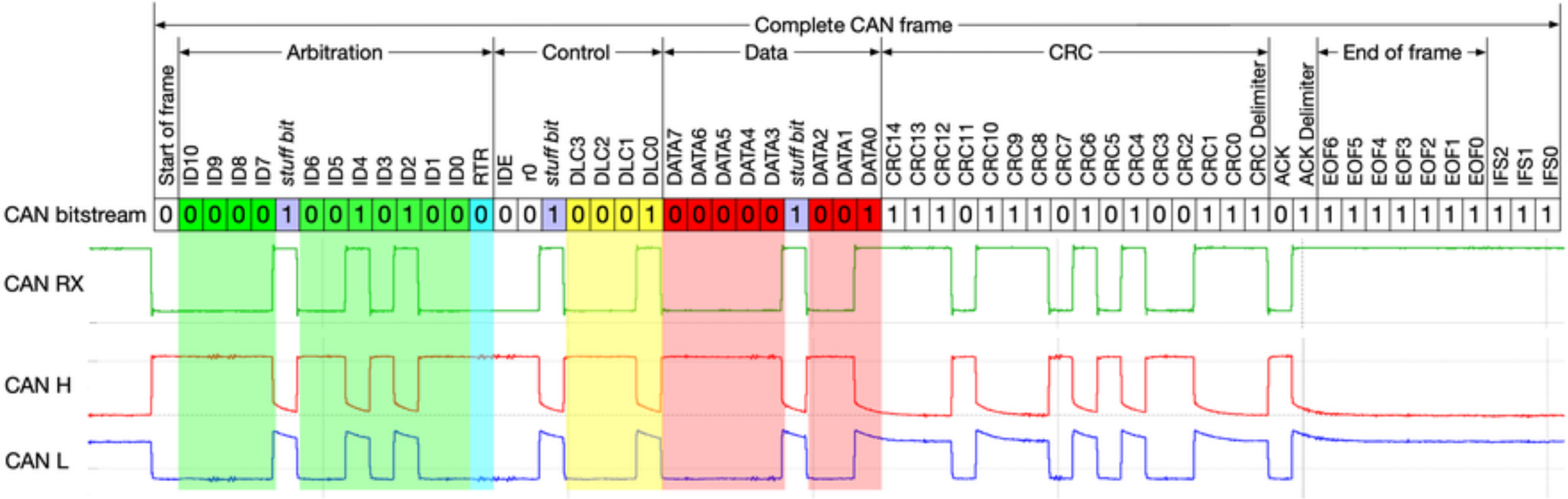
Low-speed CAN signaling. ISO 11898-3.



CAN Frames

- A CAN network can be configured to work with two different message (or *frame*) formats:
  - the standard or base frame format (described in CAN 2.0 A and CAN 2.0 B), and
  - the extended frame format (described only by CAN 2.0 B).
- 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 or remote frame

CAN Frame content composition

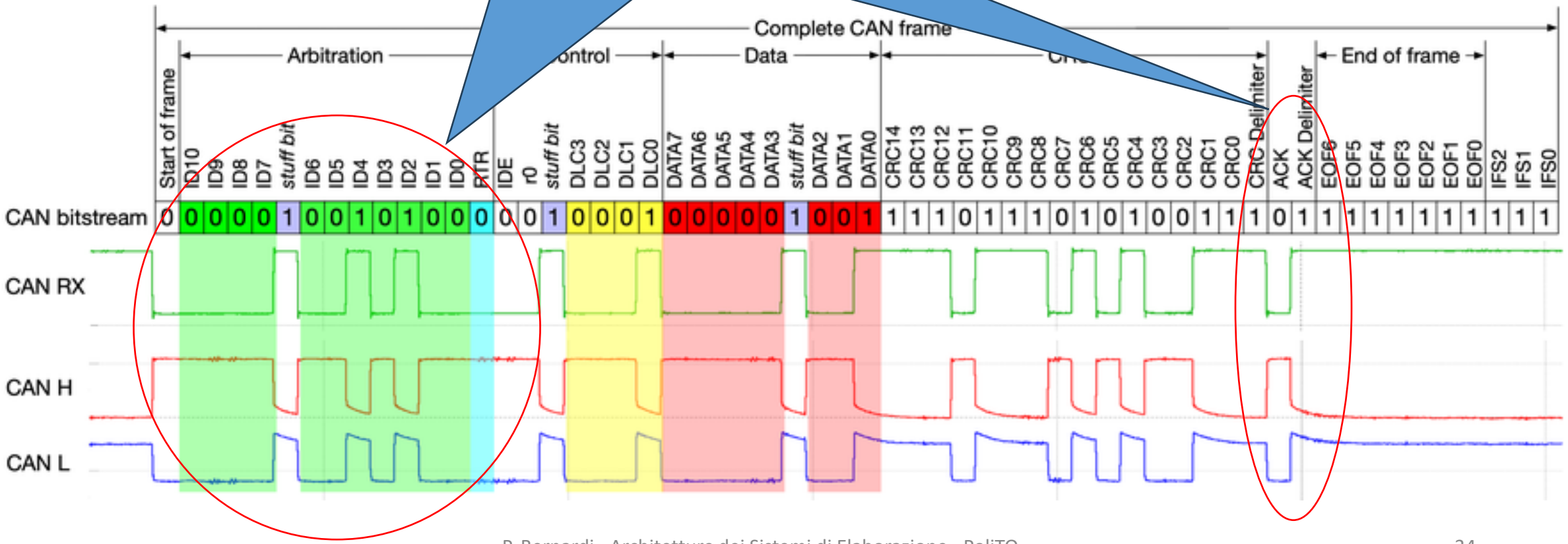


Frame composition and meaning

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier (green)	11	A (unique) identifier which also represents the message priority
Stuff bit	1	A bit of the opposite polarity to maintain synchronization
Remote transmission request (RTR) (blue)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames
Identifier extension bit (IDE)	1	Must be dominant (0) for base frame format with 11-bit identifiers
Reserved bit (r0)	1	Reserved bit. Must be 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)
Inter-frame spacing (IFS)	3	Must be recessive (1)

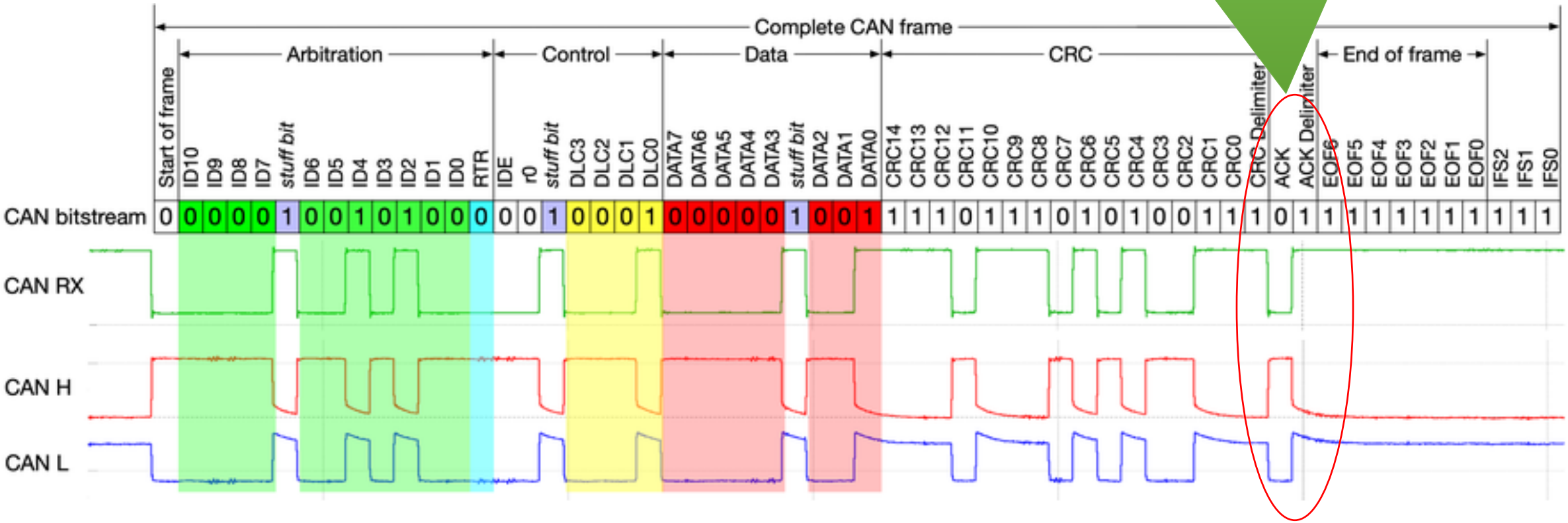
CAN Frame content composition

Along with the explaination, we will go through Arbitration and Acknowledge fields



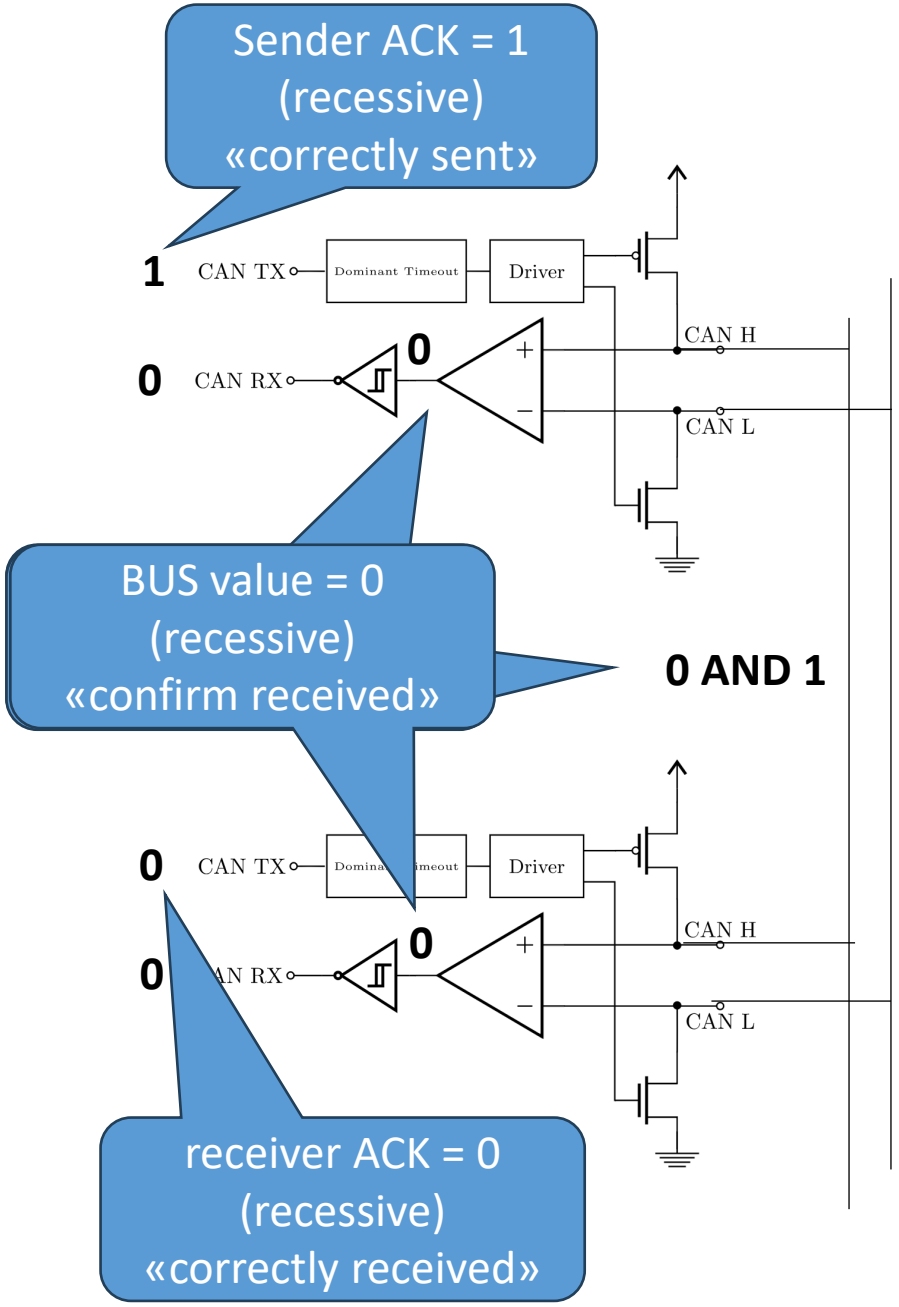
CAN Frame content composition

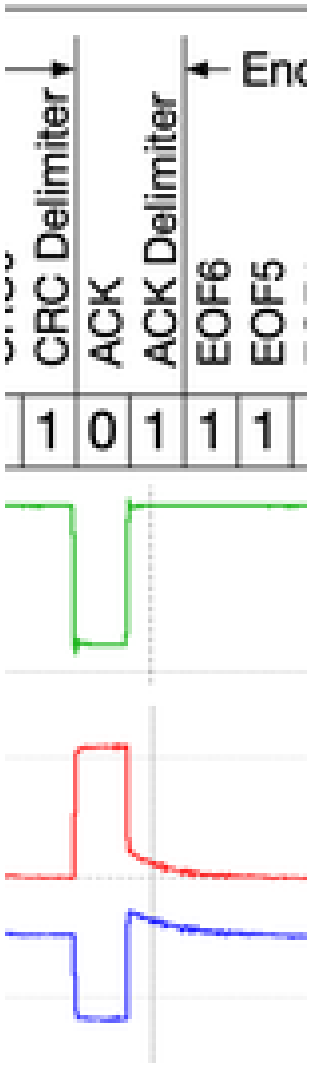
Let's start with the Acknowledge field



Acknowledge (I)

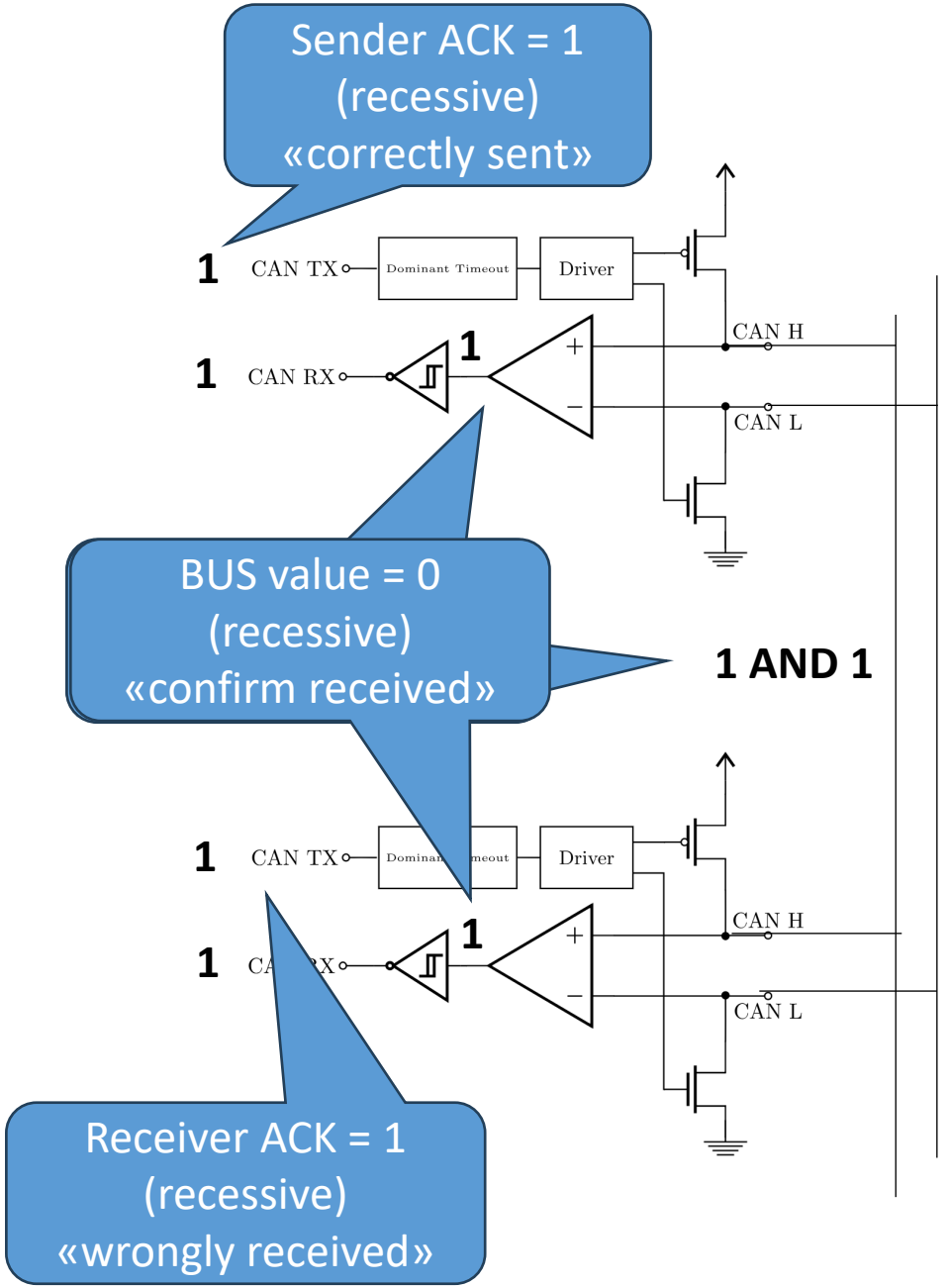
- When ACK bits are scheduled
  1. Transmitter sends recessive (1)
  2. ANY receiver can assert a dominant (0)
  3. Receivers and receiver read bus value and validate the transmission if it is dominant (0)
  4. If different (ACK = 1), retransmission is necessary

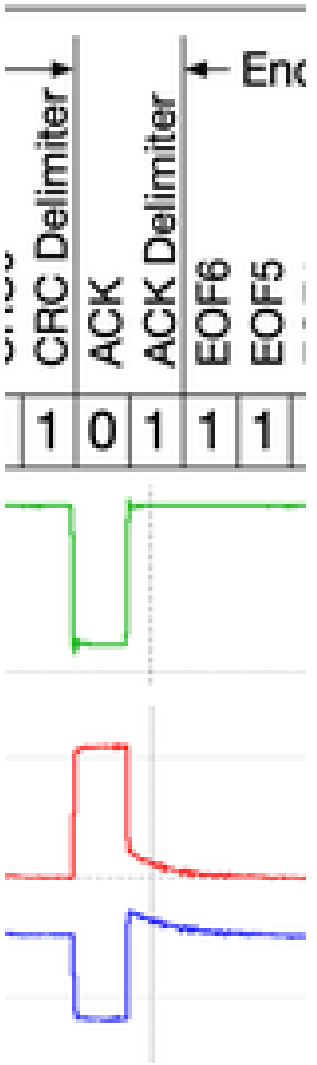




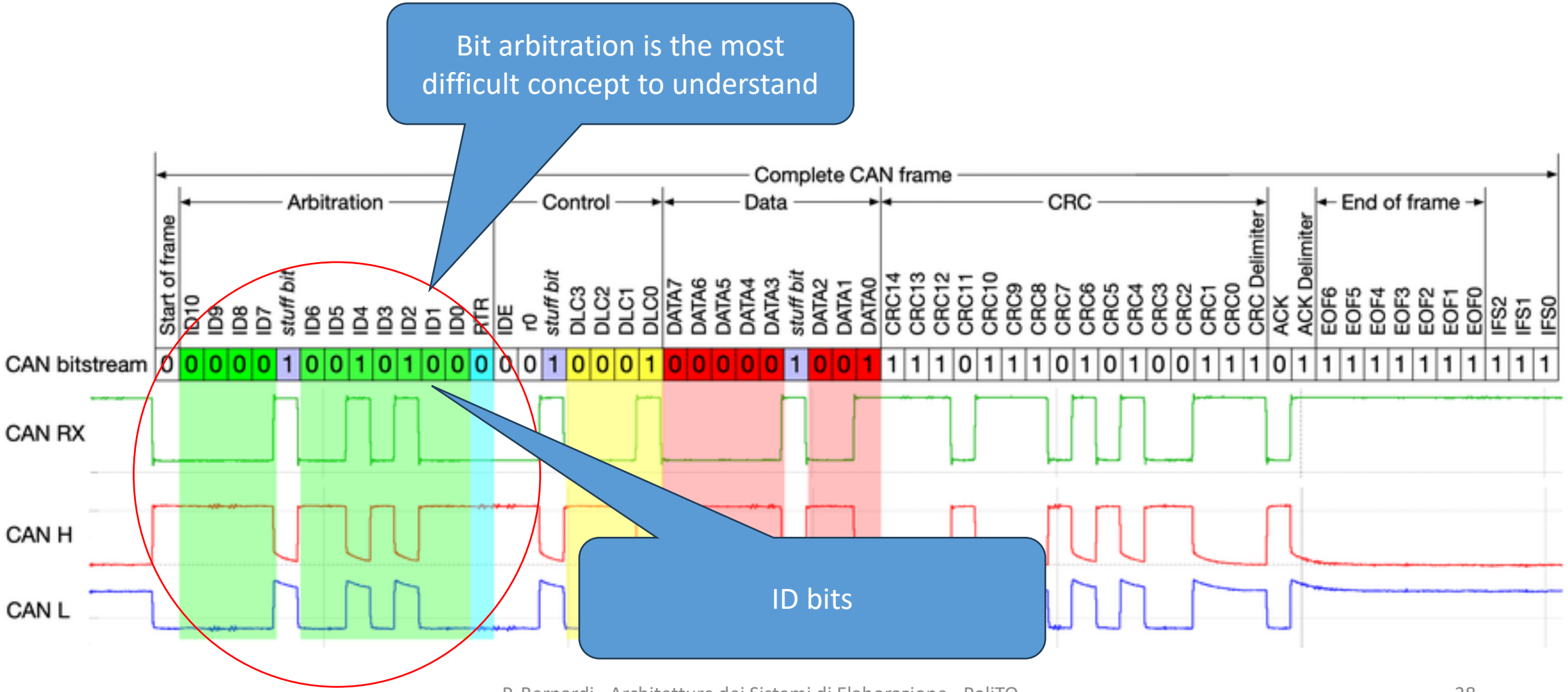
Acknowledge (II)

- If data is NOT received correctly (i.e., wrong CRC due to strong noise on the line), the receiver sends ACK = 1
  - Bus value driven to 1
  - All nodes acknowledged about the transmission issue.





CAN Frame content composition



BUS arbitration – ID Bits

- CAN data transmission uses a lossless bitwise arbitration method of contention resolution.
- This arbitration method requires all nodes on the CAN network to be **synchronized to sample every bit** on the CAN network simultaneously.
- This is why some call CAN synchronous.
  - Unfortunately, the term synchronous is imprecise since the data is transmitted in an asynchronous format without a clock signal.

Multi-master – BUS arbitration

- BUS arbitration is necessary when more than one CAN node needs to send asynchronous messages to the other nodes
- If the communication is not «dense», then it is less probable that they contend the BUS
- Else, many nodes would like to access the BUS at the same time

Multi-master – BUS arbitration

USAGE RULES ARE NEEDED – see the scenario below:

1. Any node has its ID
2. Any node can act as a master
3. If a node currently uses the BUS as a master
   A. ALL connected CAN nodes know it (including the sender) because they got the "start of frame" bit
   B. NONE is allowed to interfere with the CAN BUS till the frame is fully sent, till the "end of frame" bits
4. Along the transmission of the previous frame
   A. Messages may accumulate in the CAN controllers' queues
   B. They may come from different nodes
5. **Who is "taking" the BUS among accumulated requests when it free up?**

Who is “taking” the BUS?

- By using this process, **any node that transmits a logical 1, when another node transmits a logical 0, loses the arbitration and drops out**.
- A node that loses arbitration re-queues its message for later transmission and the CAN frame bit-stream continues without error until only one node is left transmitting.
- This means that the node that transmits the first 1 loses arbitration.
- Since the 11 (or 29 for CAN 2.0B) bit identifier is transmitted by all nodes at the start of the CAN frame, the node with the **lowest identifier** transmits **more zeros** at the start of the frame, and that is the node that wins the arbitration or has the **highest priority**.

Who is “taking” the BUS? Example….

For example, consider an 11-bit ID CAN network, with two nodes with IDs of 15 (binary representation, 00000001111) and 16 (binary representation, 00000010000).

Start bit	ID bits										
	10	9	8	7	6	5	4	3	2	1	0
Node 15	0	0	0	0	0	0	0	0	1	1	1
Node 16	0	0	0	0	0	0	0	1	**Stopped Transmitting**		
BUS value	0	0	0	0	0	0	0	0	1	1	1

Case of study: the CAN works as a Wired-AND logic

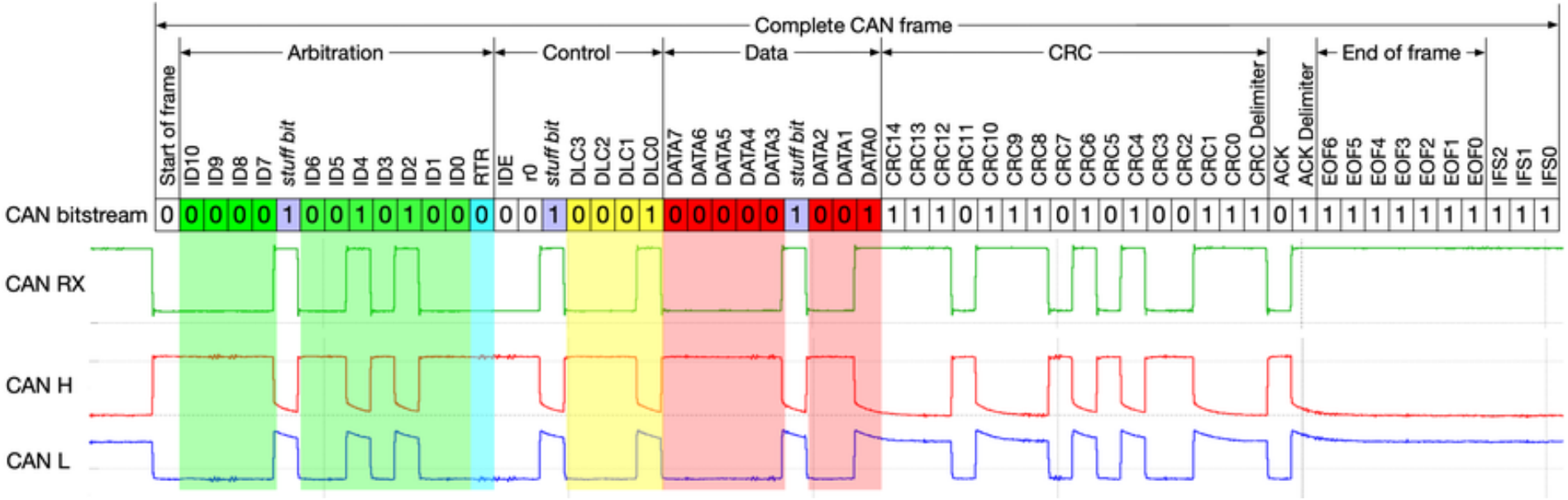
- The basic principle of CAN requires that each node listen to the data on the CAN network including the transmitting node(s) itself (themselves).
- If a logical 1 is transmitted by all transmitting nodes at the same time, then a logical 1 is seen by all of the nodes, including both the transmitting node(s) and receiving node(s).
- If a logical 0 is transmitted by all transmitting node(s) at the same time, then a logical 0 is seen by all nodes.
- If a logical 0 is being transmitted by one or more nodes, and a logical 1 is being transmitted by one or more nodes, then a logical 0 is seen by all nodes including the node(s) transmitting the logical 1.
- When a node transmits a logical 1 but sees a logical 0, it realizes that there is a bus contention, and it quits transmitting.

CAN Frames

- CAN 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 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 node which takes the BUS transmits the following data



Data frame

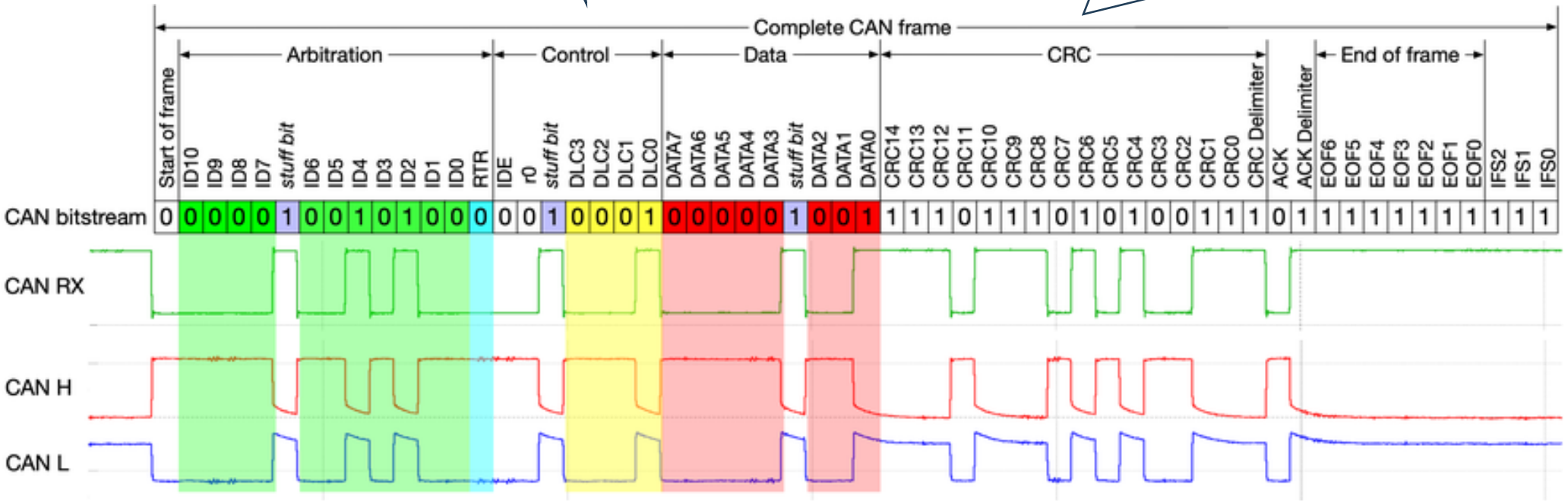
Includes many fields

- CONTROL
- DATA
  - Include stuff bits
- CRC

Including DATA LENGHT field (indicating the number of BYTES sent by the frame)

Must be coherent with the DATA LENGHT field

Cyclic Code Redundancy BITS permits to test the DATA field



CAN Frames

- CAN 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 or remote frame

Remote Frame

- Data transmission is performed on an autonomous basis with the data source node (e.g., a sensor) sending out a data frame. A destination node can request the data from the source by sending a remote frame.
- There are two differences between a data frame and a remote frame:
  - The RTR-bit is transmitted as a dominant bit in the data frame (i.e., 0 value) and in the remote frame there is no data field.
  - RTR = 0 ; DOMINANT in data frame
  - RTR = 1 ; RECESSIVE in remote frame
- The DLC field indicates the data length of the requested message.
- If a data frame and a remote frame with the same identifier start TX contemporary, the data frame wins arbitration due to the dominant RTR bit.

CAN Frames

- CAN 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 or remote frame

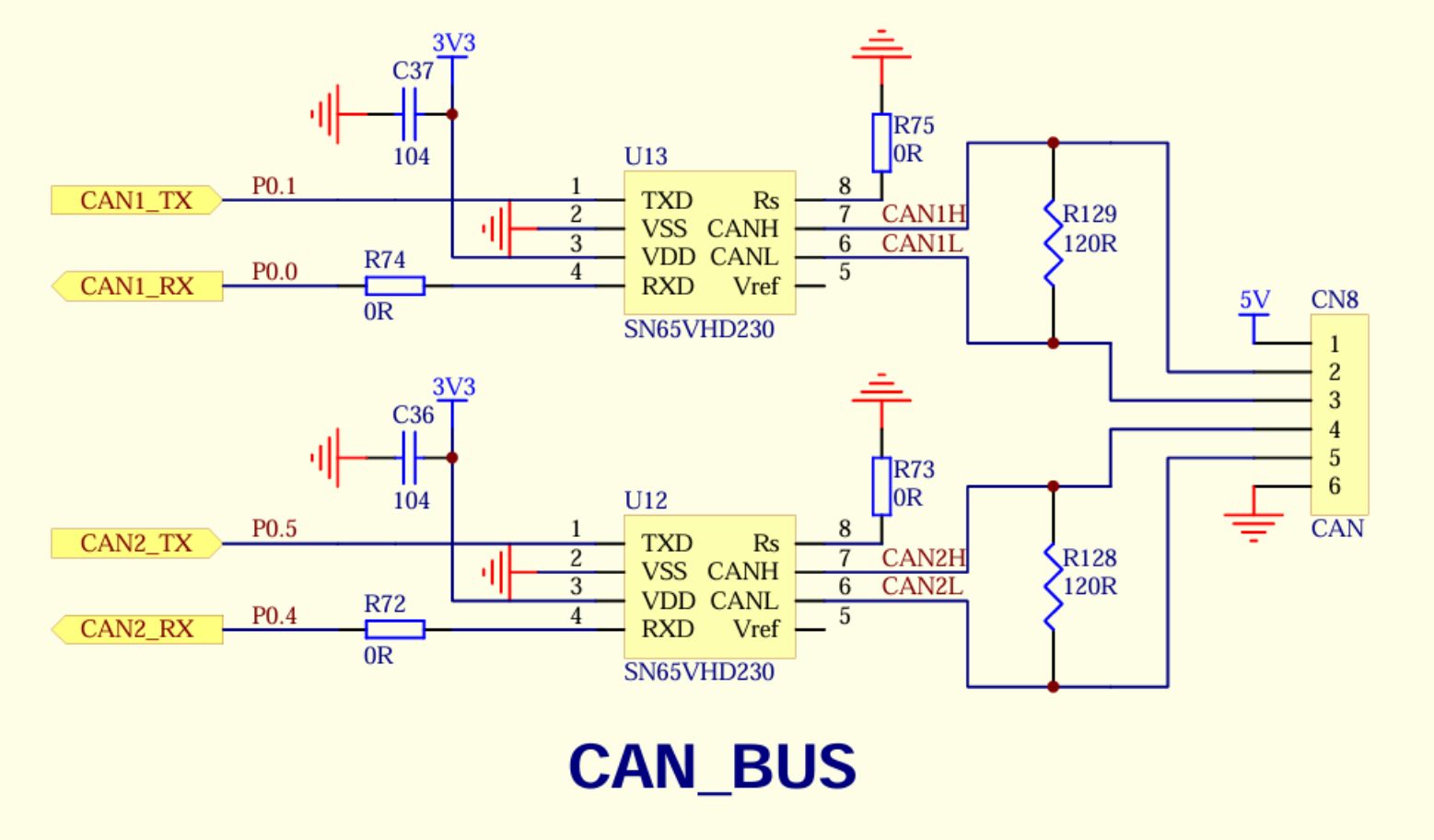
CAN Frames

- CAN 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 or remote frame

LANDTIGER IMPLEMENTATION

LANDTIGER BOARD

LPC1768 CAN BUS and LandTiger



UM10360

Chapter 16: LPC176x/5x CAN1/2

Rev. 3.1 — 2 April 2014 User manual

16.1 Basic configuration

The CAN1/2 peripherals are configured using the following registers:

1. Power: In the PCONP register (Table 46), set bits PCAN1/2.

   Remark: On reset, the CAN1/2 blocks are disabled (PCAN1/2 = 0).

2. Peripheral clock: In the PCLKSEL0 register (Table 40), select PCLK_CAN1, PCLK_CAN2, and, for the acceptance filter, PCLK_ACF. Note that these must all be the same value.

   Remark: If CAN baud rates above 100 kbit/s (see Table 323) are needed, do not select the IRC as the clock source (see Table 17).

3. Wake-up: CAN controllers are able to wake up the microcontroller from Power-down mode, see Section 4.8.8.

4. Pins: Select CAN1/2 pins through the PINSEL registers and their pin modes through the PINMODE registers (Section 8.5).

5. Interrupts: CAN interrupts are enabled using the CAN1/2IER registers (Table 322). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

6. CAN controller initialization: see CANMOD register (Section 16.7.1).

16.2 CAN controllers

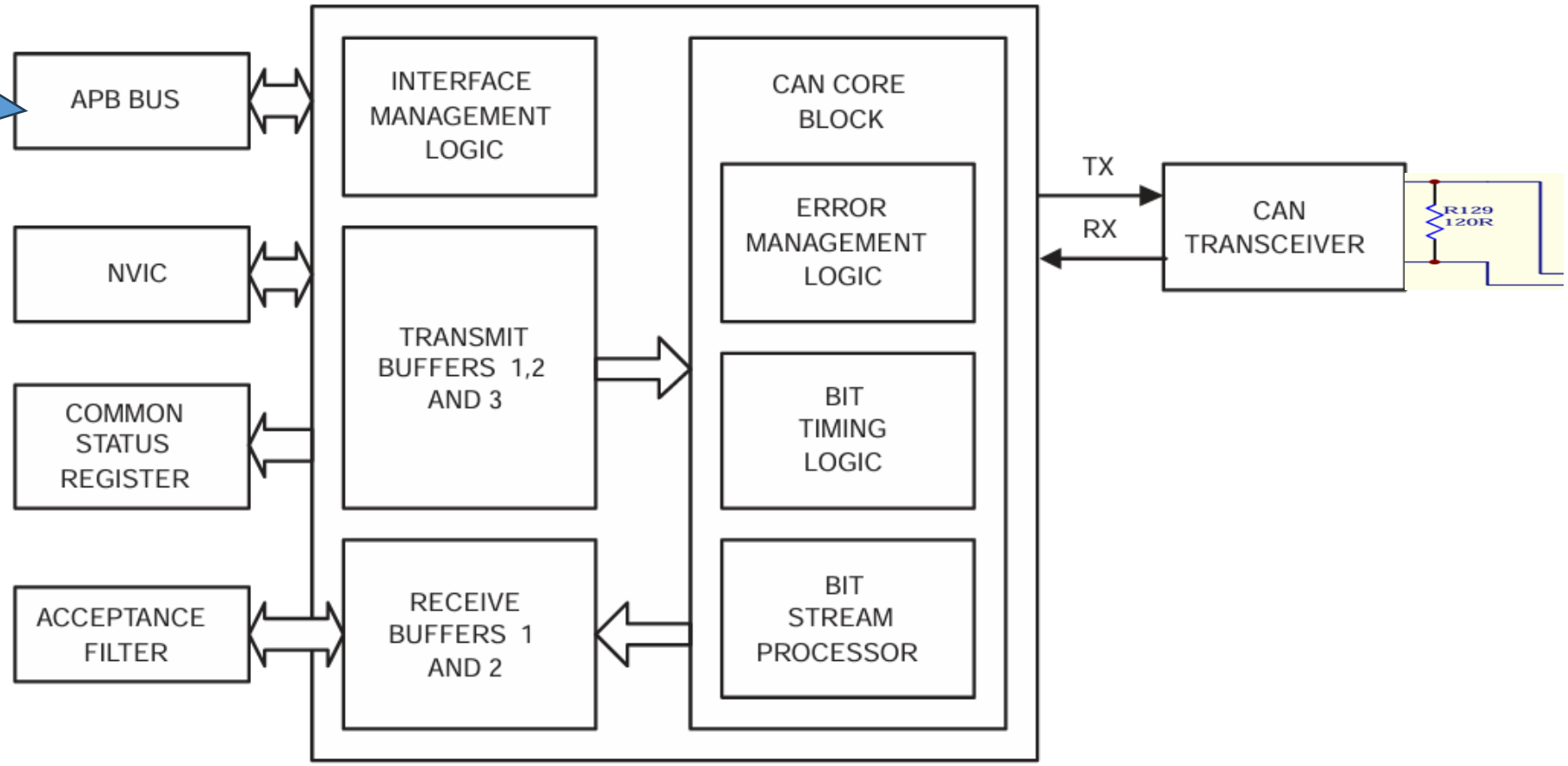
Controller Area Network (CAN) is the definition of a high performance communication protocol for serial data communication. The CAN Controller is designed to provide a full implementation of the CAN-Protocol according to the CAN Specification Version 2.0B. Microcontrollers with this on-chip CAN controller are used to build powerful local networks by supporting distributed real-time control with a very high level of security. The applications are automotive, industrial environments, and high speed networks as well as low cost multiplex wiring. The result is a strongly reduced wiring harness and enhanced diagnostic and supervisory capabilities.

The CAN block is intended to support multiple CAN buses simultaneously, allowing the device to be used as a gateway, switch, or router among a number of CAN buses in various applications.

The CAN module consists of two elements: the controller and the Acceptance Filter. All registers and the RAM are accessed as 32-bit words.

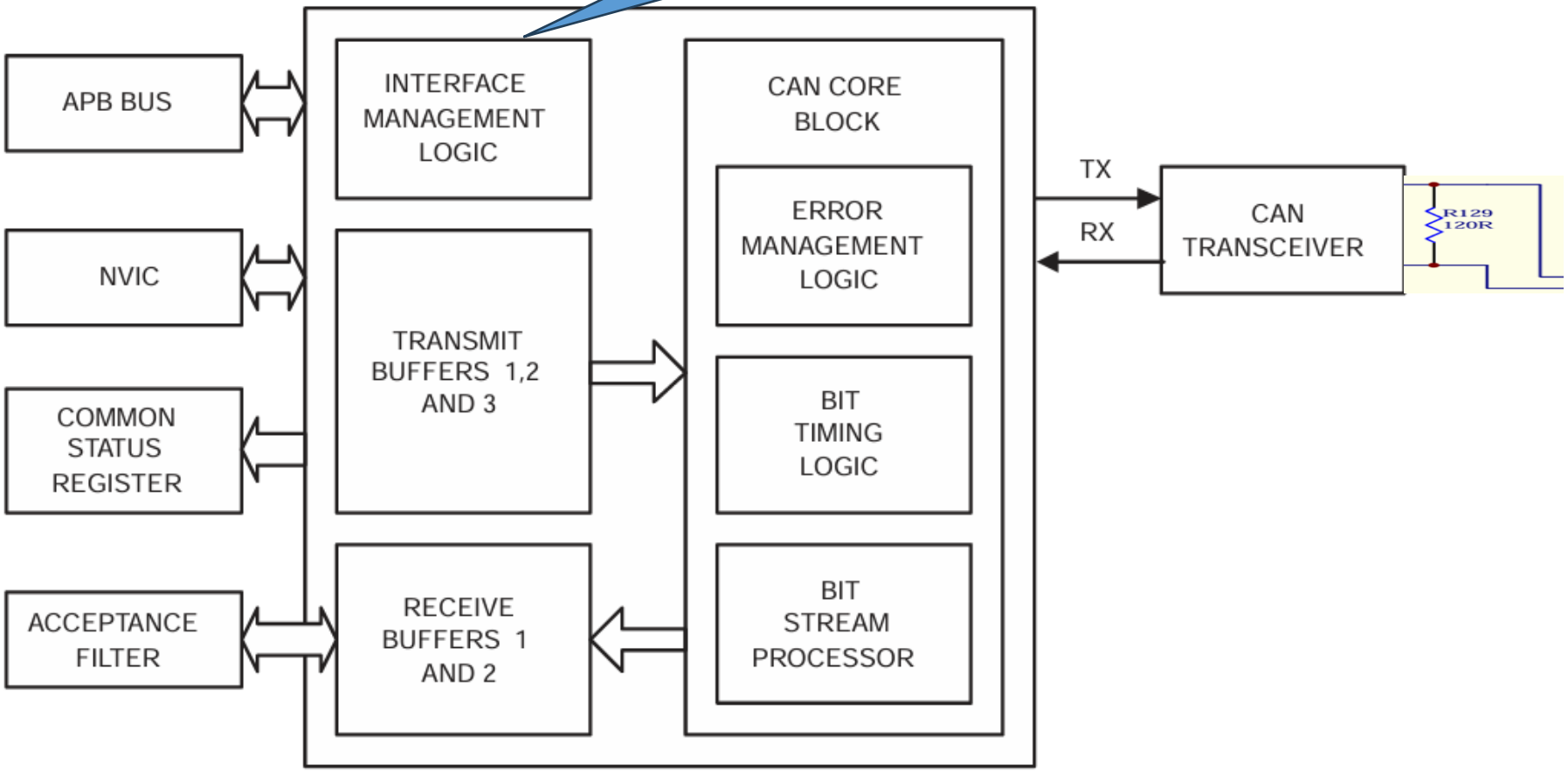
CAN Module schematic

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

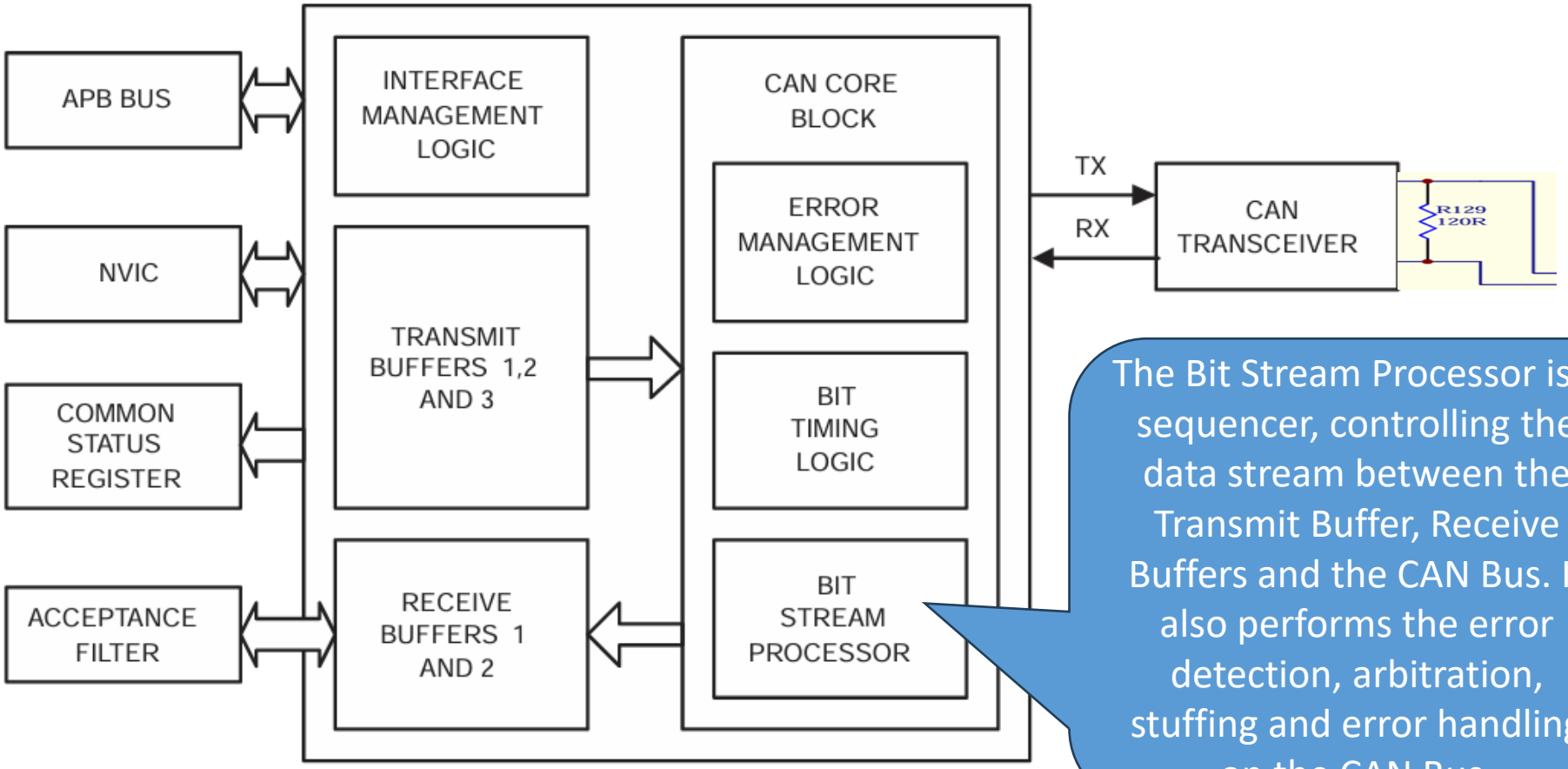


CAN Module schematic

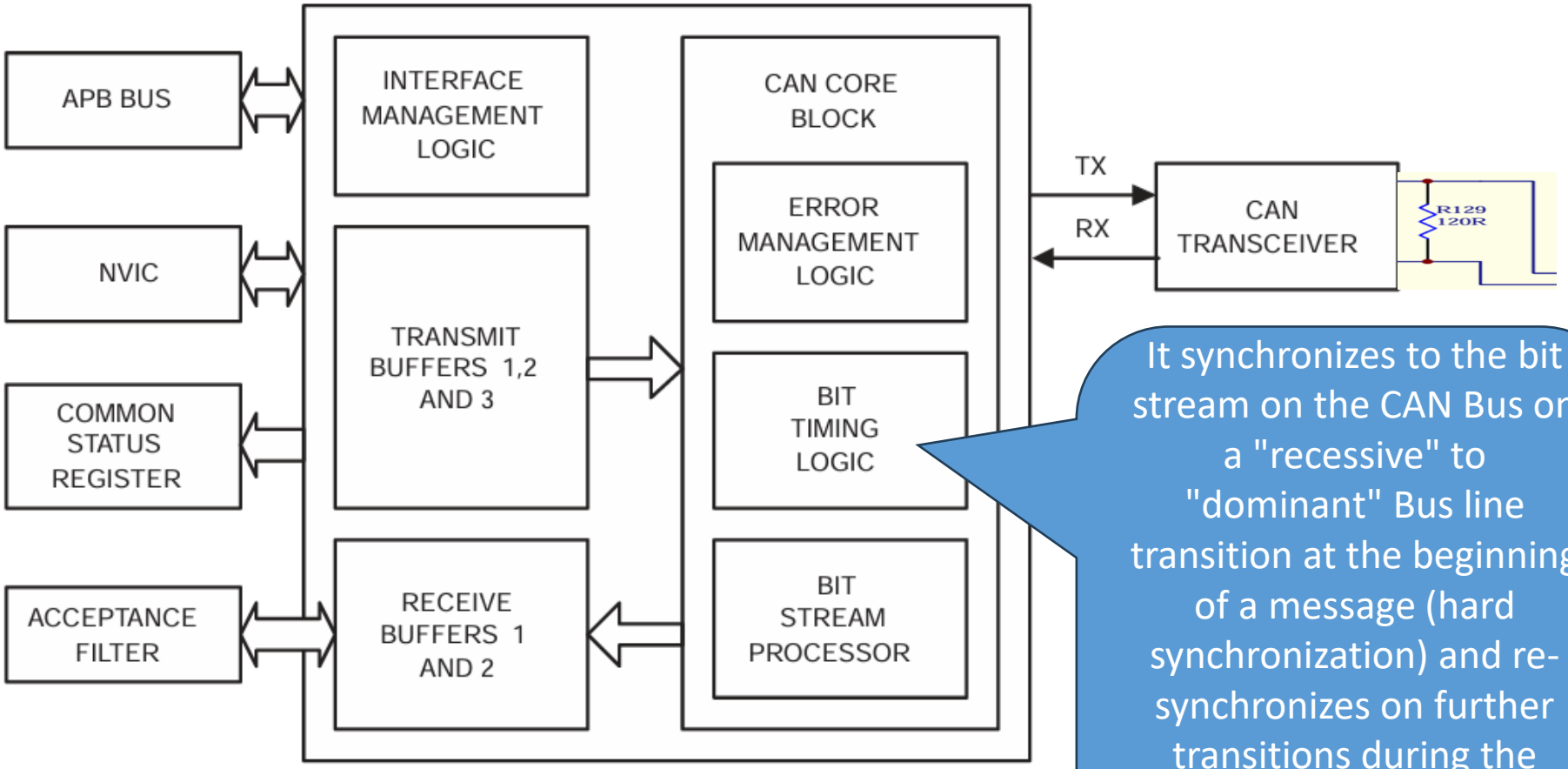
The Interface Management Logic interprets commands from the CPU, controls internal addressing of the CAN Registers and provides interrupts and status information to the CPU.



CAN Module schematic

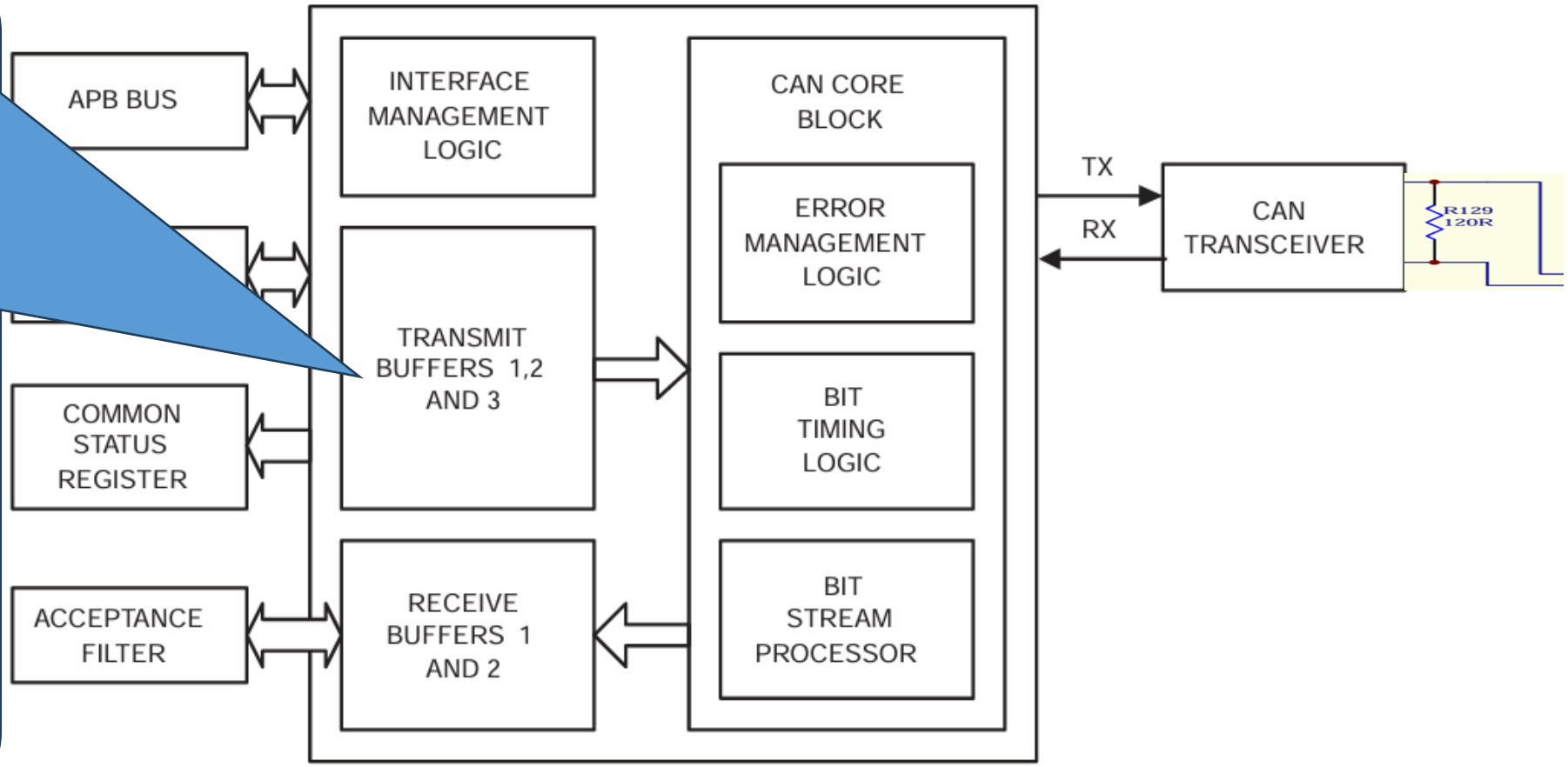


CAN Module schematic



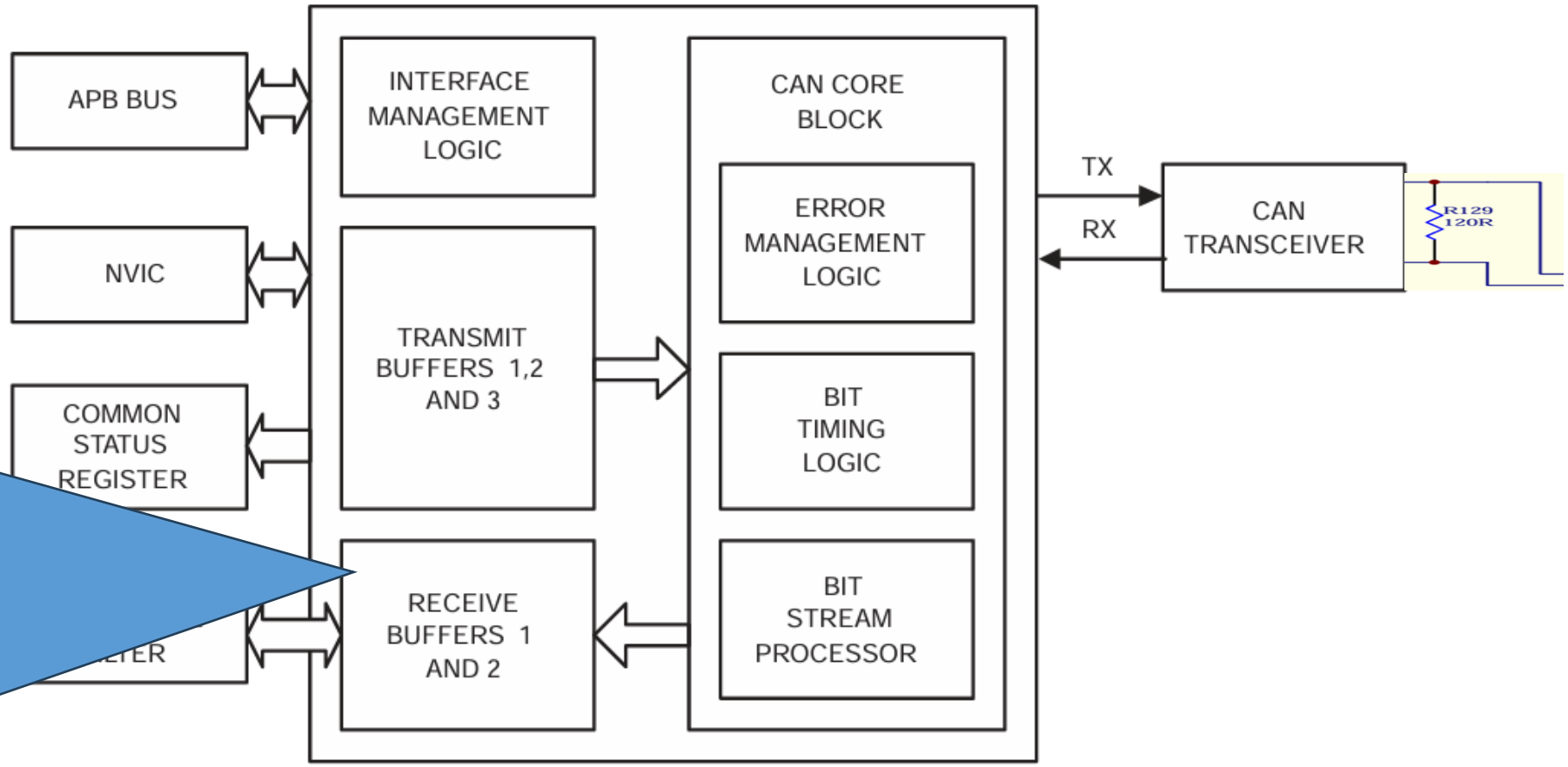
CAN Module schematic

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.
This buffer is written by the CPU and read out by the BSP



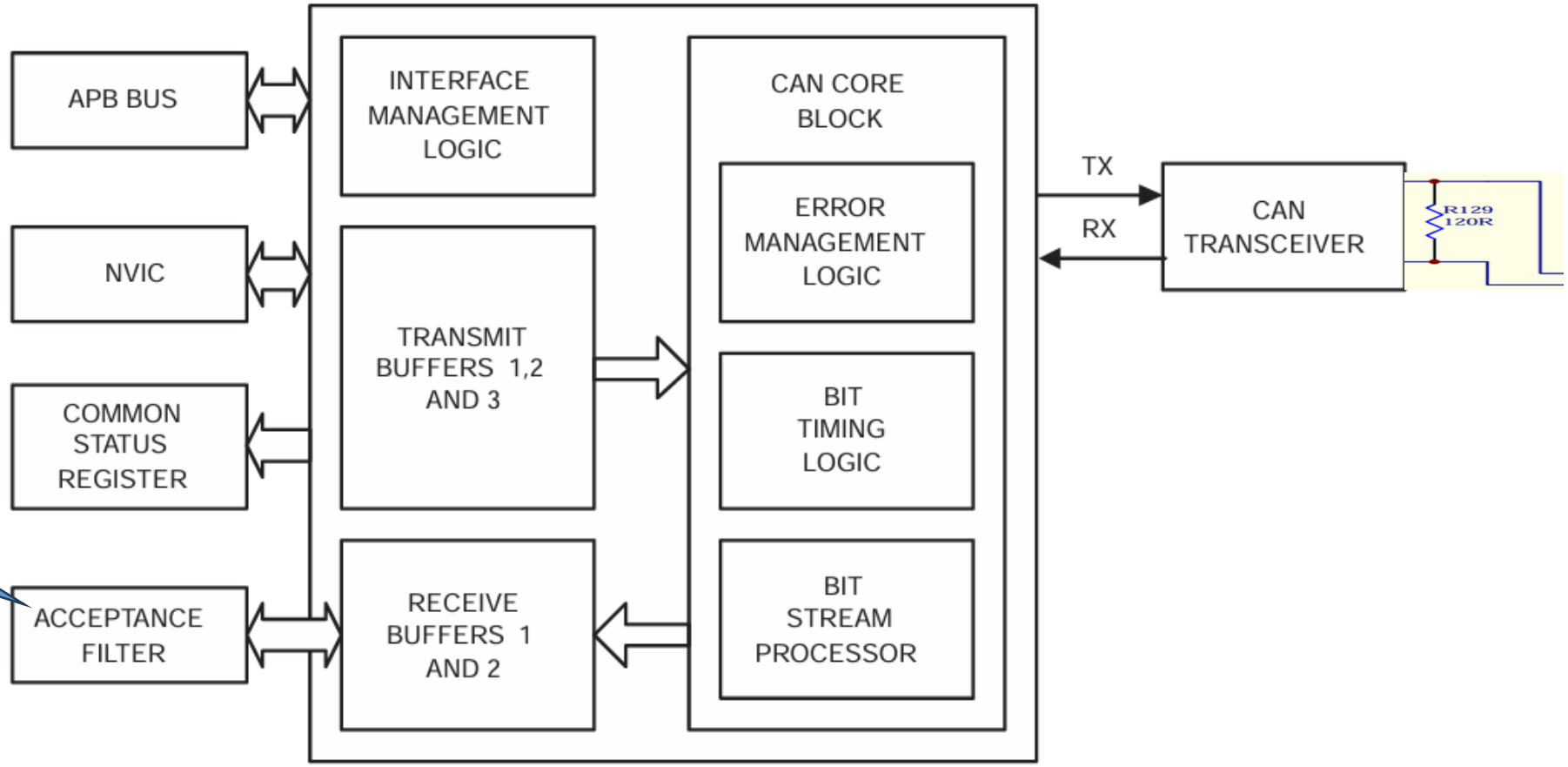
CAN Module schematic

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.



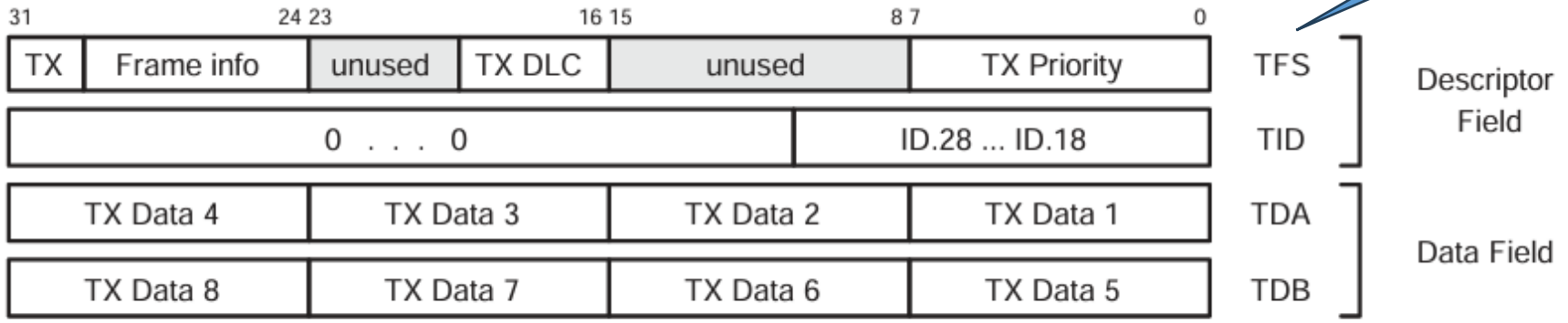
CAN Module schematic

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).

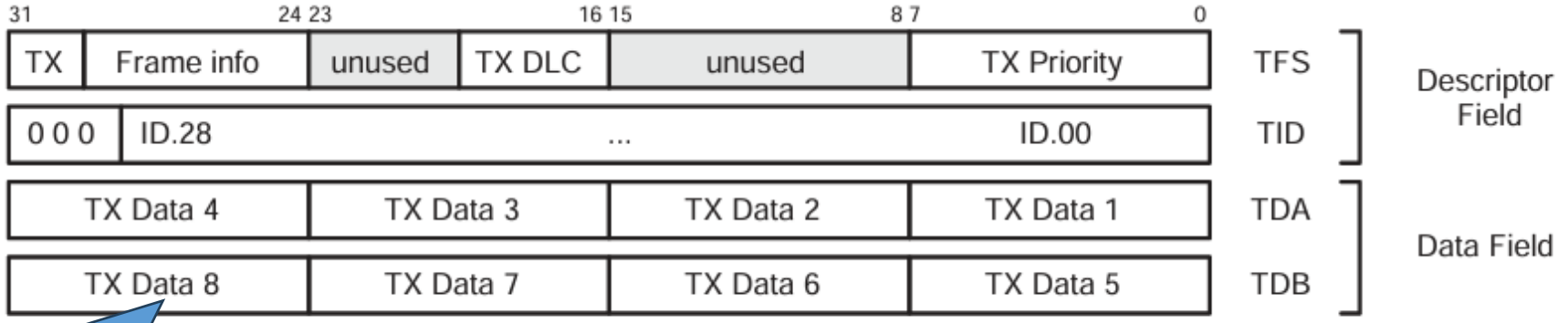


Transmit buffer layout

Register names to use when programming



Standard Frame Format (11-bit Identifier)



Extended Frame Format (29-bit Identifier)

DATA Transmitted over maximum 8 bytes

Memory map of the CAN block

- The CAN Controllers and Acceptance Filter occupy a number of APB slots

Address Range	Used for
0x4003 8000 - 0x4003 87FF	Acceptance Filter RAM.
0x4003 C000 - 0x4003 C017	Acceptance Filter Registers.
0x4004 0000 - 0x4004 000B	Central CAN Registers.
0x4004 4000 - 0x4004 405F	CAN Controller 1 Registers.
0x4004 8000 - 0x4004 805F	CAN Controller 2 Registers.
0x400F C110 - 0x400F C114	CAN Wake and Sleep Registers.

CAN Controller Registers

Not including acceptance filter regs

Table 316. CAN1 and CAN2 controller register summary

Generic Name	Operating Mode		Reset Mode	
	Read	Write	Read	Write
MOD	Mode	Mode	Mode	Mode
CMR	0x00	Command	0x00	Command
GSR	Global Status and Error Counters	-	Global Status and Error Counters	Error Counters only
ICR	Interrupt and Capture	-	Interrupt and Capture	-
IER	Interrupt Enable	Interrupt Enable	Interrupt Enable	Interrupt Enable
BTR	Bus Timing	-	Bus Timing	Bus Timing
EWL	Error Warning Limit	-	Error Warning Limit	Error Warning Limit
SR	Status	-	Status	-
RFS	Rx Info and Index	-	Rx Info and Index	Rx Info and Index
RID	Rx Identifier	-	Rx Identifier	Rx Identifier
RDA	Rx Data	-	Rx Data	Rx Data
RDB	Rx Info and Index	-	Rx Info and Index	Rx Info and Index
TFI1	Tx Info1	Tx Info	Tx Info	Tx Info
TID1	Tx Identifier	Tx Identifier	Tx Identifier	Tx Identifier
TDA1	Tx Data	Tx Data	Tx Data	Tx Data
TDB1	Tx Data	Tx Data	Tx Data	Tx Data

CAN Controller Registers (II)

Table 315. CAN1 and CAN2 controller register map

Generic Name	Description	Access	Reset value	CAN1 & 2 Register Name & Address
MOD	Controls the operating mode of the CAN Controller.	R/W	1	CAN1MOD - 0x4004 4000 CAN2MOD - 0x4004 8000
CMR	Command bits that affect the state of the CAN Controller	WO	0	CAN1CMR - 0x4004 4004 CAN2CMR - 0x4004 8004
GSR	Global Controller Status and Error Counters	RO[1]	0x3C	CAN1GSR - 0x4004 4008 CAN2GSR - 0x4004 8008
ICR	Interrupt status, Arbitration Lost Capture, Error Code Capture	RO	0	CAN1ICR - 0x4004 400C CAN2ICR - 0x4004 800C
IER	Interrupt Enable	R/W	0	CAN1IER - 0x4004 4010 CAN2IER - 0x4004 8010

Table 315. CAN1 and CAN2 controller register map *...continued*

Generic Name	Description	Access	Reset value	CAN1 & 2 Register Name & Address
BTR	Bus Timing	R/W[2]	0x1C0000	CAN1BTR - 0x4004 4014 CAN2BTR - 0x4004 8014
EWL	Error Warning Limit	R/W[2]	0x60	CAN1EWL - 0x4004 4018 CAN2EWL - 0x4004 8018
SR	Status Register	RO	0x3C3C3C	CAN1SR - 0x4004 401C CAN2SR - 0x4004 801C

CAN Controller Registers (III)

RFS	Receive frame status	R/W[2]	0	CAN1RFS - 0x4004 4020 CAN2RFS - 0x4004 8020
RID	Received Identifier	R/W[2]	0	CAN1RID - 0x4004 4024 CAN2RID - 0x4004 8024
RDA	Received data bytes 1-4	R/W[2]	0	CAN1RDA - 0x4004 4028 CAN2RDA - 0x4004 8028
RDB	Received data bytes 5-8	R/W[2]	0	CAN1RDB - 0x4004 402C CAN2RDB - 0x4004 802C
TFI1	Transmit frame info (Tx Buffer 1)	R/W	0	CAN1TFI1 - 0x4004 4030 CAN2TFI1 - 0x4004 8030
TID1	Transmit Identifier (Tx Buffer 1)	R/W	0	CAN1TID1 - 0x4004 4034 CAN2TID1 - 0x4004 8034
TDA1	Transmit data bytes 1-4 (Tx Buffer 1)	R/W	0	CAN1TDA1 - 0x4004 4038 CAN2TDA1 - 0x4004 8038
TDB1	Transmit data bytes 5-8 (Tx Buffer 1)	R/W	0	CAN1TDB1- 0x4004 403C CAN2TDB1 - 0x4004 803C
TFI2	Transmit frame info (Tx Buffer 2)	R/W	0	CAN1TFI2 - 0x4004 4040 CAN2TFI2 - 0x4004 8040
TID2	Transmit Identifier (Tx Buffer 2)	R/W	0	CAN1TID2 - 0x4004 4044 CAN2TID2 - 0x4004 8044
TDA2	Transmit data bytes 1-4 (Tx Buffer 2)	R/W	0	CAN1TDA2 - 0x4004 4048 CAN2TDA2 - 0x4004 8048
TDB2	Transmit data bytes 5-8 (Tx Buffer 2)	R/W	0	CAN1TDB2 - 0x4004 404C CAN2TDB2 - 0x4004 804C
TFI3	Transmit frame info (Tx Buffer 3)	R/W	0	CAN1TFI3 - 0x4004 4050 CAN2TFI3 - 0x4004 8050
TID3	Transmit Identifier (Tx Buffer 3)	R/W	0	CAN1TID3 - 0x4004 4054 CAN2TID3 - 0x4004 8054
TDA3	Transmit data bytes 1-4 (Tx Buffer 3)	R/W	0	CAN1TDA3 - 0x4004 4058 CAN2TDA3 - 0x4004 8058
TDB3	Transmit data bytes 5-8 (Tx Buffer 3)	R/W	0	CAN1TDB3 - 0x4004 405C CAN2TDB3 - 0x4004 805C

CAN Mode register

- The contents of the Mode Register are 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.

Table 318. CAN Mode register (CAN1MOD - address 0x4004 4000, CAN2MOD - address 0x4004 8000) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RM[1][6]		Reset Mode.	1	1
		0 (normal)	The CAN Controller is in the Operating Mode, and certain registers can not be written.		
		1 (reset)	CAN operation is disabled, writable registers can be written and the current transmission/reception of a message is aborted.		
1	LOM[3][2][6]		Listen Only Mode.	0	x
		0 (normal)	The CAN controller acknowledges a successfully received message on the CAN bus. The error counters are stopped at the current value.		
		1 (listen only)	The controller gives no acknowledgment, even if a message is successfully received. Messages cannot be sent, and the controller operates in "error passive" mode. This mode is intended for software bit rate detection and "hot plugging".		
2	STM[3][6]		Self Test Mode.	0	x
		0 (normal)	A transmitted message must be acknowledged to be considered successful.		
		1 (self test)	The controller will consider a Tx message successful even if there is no acknowledgment received. In this mode a full node test is possible without any other active node on the bus using the SRR bit in CANxCMR.		
3	TPM[4]		Transmit Priority Mode.	0	x
		0 (CAN ID)	The transmit priority for 3 Transmit Buffers depends on the CAN Identifier.		
		1 (local prio)	The transmit priority for 3 Transmit Buffers depends on the contents of the Tx Priority register within the Transmit Buffer.		
4	SM[5]		Sleep Mode.	0	0
		0 (wake-up)	Normal operation.		
		1 (sleep)	The CAN controller enters Sleep Mode if no CAN interrupt is pending and there is no bus activity. See the Sleep Mode description Section 16.8.2 on page 379.		
5	RPM		Receive Polarity Mode.	0	x
		0 (low active)	RD input is active Low (dominant bit = 0).		
		1 (high active)	RD input is active High (dominant bit = 1) -- reverse polarity.		
6	-	-	Reserved, user software should not write ones to reserved bits.	0	0
7	TM		Test Mode.	0	x
		0 (disabled)	Normal operation.		
		1 (enabled)	The TD pin will reflect the bit, detected on RD pin, with the next positive edge of the system clock.		
31:8	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	

CAN Command Register

Table 319. CAN Command Register (CAN1CMR - address 0x4004 4004, CAN2CMR - address 0x4004 8004) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0[1][2]	TR		Transmission Request.	0	0
		0 (absent)	No transmission request.		
		1 (present)	The message, previously written to the CANxTFI, CANxTID, and optionally the CANxTDA and CANxTDB registers, is queued for transmission from the selected Transmit Buffer. 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 (for details see Section 16.5.3 "Transmit Buffers (TXB)")		
1[1][3]	AT		Abort Transmission.	0	0
		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.		
2[4]	RRB		Release Receive Buffer.	0	0
		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).		
3[5]	CDO		Clear Data Overrun.	0	0
		0 (no action)	Do not clear the data overrun bit.		
		1 (clear)	The Data Overrun bit in Status Register(s) is cleared.		
4[1][6]	SRR		Self Reception Request.	0	0
		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.		

CAN Command Register (II)

Table 319. CAN Command Register (CAN1CMR - address 0x4004 4004, CAN2CMR - address 0x4004 8004) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
5	STB1		Select Tx Buffer 1.	0	0
		0 (not selected)	Tx Buffer 1 is not selected for transmission.		
		1 (selected)	Tx Buffer 1 is selected for transmission.		
6	STB2		Select Tx Buffer 2.	0	0
		0 (not selected)	Tx Buffer 2 is not selected for transmission.		
		1 (selected)	Tx Buffer 2 is selected for transmission.		
7	STB3		Select Tx Buffer 3.	0	0
		0 (not selected)	Tx Buffer 3 is not selected for transmission.		
		1 (selected)	Tx Buffer 3 is selected for transmission.		
31:8	-		Reserved, user software should not write ones to reserved bits.	NA	

CAN Global Status Register

Table 320. CAN Global Status Register (CAN1GSR - address 0x4004 4008, CAN2GSR - address 0x4004 8008) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RBS[1]		Receive Buffer Status.	0	0
		0 (empty)	No message is available.		
		1 (full)	At least one complete message is received by the Double Receive Buffer and available in the CANxRFS, CANxRID, and if applicable the CANxRDA and CANxRDB registers. This bit is cleared by the Release Receive Buffer command in CANxCMR, if no subsequent received message is available.		
1	DOS[2]		Data Overrun Status.	0	0
		0 (absent)	No data overrun has occurred since the last Clear Data Overrun command was given/written to CANxCMR (or since Reset).		
		1 (overrun)	A message was lost because the preceding message to this CAN controller was not read and released quickly enough (there was not enough space for a new message in the Double Receive Buffer).		
2	TBS		Transmit Buffer Status.	1	1
		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).		
		1 (released)	All three Transmit Buffers are available for the CPU. No transmit message is pending for this CAN controller (in any of the 3 Tx buffers), and software may write to any of the CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.		

Table 320. CAN Global Status Register (CAN1GSR - address 0x4004 4008, CAN2GSR - address 0x4004 8008) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RBS[1]		Receive Buffer Status.	0	0
		0 (empty)	No message is available.		
		1 (full)	At least one complete message is received by the Double Receive Buffer and available in the CANxRFS, CANxRID, and if applicable the CANxRDA and CANxRDB registers. This bit is cleared by the Release Receive Buffer command in CANxCMR, if no subsequent received message is available.		
1	DOS[2]		Data Overrun Status.	0	0
		0 (absent)	No data overrun has occurred since the last Clear Data Overrun command was given/written to CANxCMR (or since Reset).		
			run)	A message was lost because the preceding message to this CAN controller was not read and released quickly enough (there was not enough space for a new message in the Double Receive Buffer).	
				Transmit Buffer Status.	1	1
			ed)	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).	
			sed)	All three Transmit Buffers are available for the CPU. No transmit message is pending for this CAN controller (in any of the 3 Tx buffers), and software may write to any of the CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.	
				Transmit Complete Status.	1	x
			mplete)	At least one requested transmission has not been successfully completed yet.	
			plete)	All requested transmission(s) has (have) been successfully completed.	
				Receive Status.	1	0
				The CAN controller is idle.		
			ive)	The CAN controller is receiving a message.		
				Transmit Status.	1	0
				The CAN controller is idle.		
			smit)	The CAN controller is sending a message.		
				Error Status.	0	0
				Both error counters are below the Error Warning Limit.		
			)	One or both of the Transmit and Receive Error Counters has reached the limit set in the Error Warning Limit register.		
				Bus Status.	0	0
			On)	The CAN Controller is involved in bus activities		
			Off)	The CAN controller is currently not involved/prohibited from bus activity because the Transmit Error Counter reached its limiting value of 255.		
				Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	
				The current value of the Rx Error Counter (an 8-bit value).	0	X
31:24	TXERR	-		The current value of the Tx Error Counter (an 8-bit value).	0	X

[1] After reading all messages and releasing their memory space with the command 'Release Receive Buffer,' this bit is cleared.

CAN Global Status Register

Table 320. CAN Global Status Register (CAN1GSR - address 0x4004 4008, CAN2GSR - address 0x4004 8008) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RBS[1]		Receive Buffer Status.	0	0
		0 (empty)	No message is available.		
		1 (full)	At least one complete message is received by the Double Receive Buffer and available in the CANxRFS, CANxRID, and if applicable the CANxRDA and CANxRDB registers. This bit is cleared by the Release Receive Buffer command in CANxCMR, if no subsequent received message is available.		
1	DOS[2]		Data Overrun Status.	0	0
		0 (absent)	No data overrun has occurred since the last Clear Data Overrun command was given/written to CANxCMR (or since Reset).		
			A message was lost because the preceding message to this CAN controller was not read and released quickly enough (there was not enough space for a new message in the Double Receive Buffer).		
			Transmit Buffer Status.	1	1
			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).		
			All three Transmit Buffers are available for the CPU. No transmit message is pending for this CAN controller (in any of the 3 Tx buffers), and software may write to any of the CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.		
			Transmit Complete Status.	1	x
			At least one requested transmission has not been successfully completed yet.		
			All requested transmission(s) has (have) been successfully completed.		
			Receive Status.	1	0
			The CAN controller is idle.		
			The CAN controller is receiving a message.		
			Transmit Status.	1	0
			The CAN controller is idle.		
			The CAN controller is sending a message.		
			Error Status.	0	0
			Both error counters are below the Error Warning Limit.		
			One or both of the Transmit and Receive Error Counters has reached the limit set in the Error Warning Limit register.		
			Bus Status.	0	0
			The CAN Controller is involved in bus activities		
			The CAN controller is currently not involved/prohibited from bus activity because the Transmit Error Counter reached its limiting value of 255.		
			Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	
			The current value of the Rx Error Counter (an 8-bit value).	0	X
			The current value of the Tx Error Counter (an 8-bit value).	0	X

[1] After reading all messages and releasing their memory space with the command 'Release Receive Buffer,' this bit is cleared.

3	TCS[3]		Transmit Complete Status.	1	x
		0 (incomplete)	At least one requested transmission has not been successfully completed yet.		
		1 (complete)	All requested transmission(s) has (have) been successfully completed.		
4	RS[4]		Receive Status.	1	0
		0 (idle)	The CAN controller is idle.		
		1 (receive)	The CAN controller is receiving a message.		
5	TS[4]		Transmit Status.	1	0
		0 (idle)	The CAN controller is idle.		
		1 (transmit)	The CAN controller is sending a message.		
6	ES[5]		Error Status.	0	0
		0 (ok)	Both error counters are below the Error Warning Limit.		
		1 (error)	One or both of the Transmit and Receive Error Counters has reached the limit set in the Error Warning Limit register.		
7	BS[6]		Bus Status.	0	0
		0 (Bus-On)	The CAN Controller is involved in bus activities		
		1 (Bus-Off)	The CAN controller is currently not involved/prohibited from bus activity because the Transmit Error Counter reached its limiting value of 255.		
15:8	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	
23:16	RXERR	-	The current value of the Rx Error Counter (an 8-bit value).	0	X
31:24	TXERR	-	The current value of the Tx Error Counter (an 8-bit value).	0	X

CAN Interrupt 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

CAN Interrupt Enable Register (II)

Bit	Symbol	Function	Reset Value	RM Set
4	WUIE	Wake-Up Interrupt Enable. If the sleeping CAN controller wakes up, the respective interrupt is requested.	0	X
5	EPIE	Error Passive Interrupt Enable. If the error status of the CAN Controller changes from error active to error passive or vice versa, the respective interrupt is requested.	0	X
6	ALIE	Arbitration Lost Interrupt Enable. If the CAN Controller has lost arbitration, the respective interrupt is requested.	0	X
7	BEIE	Bus Error Interrupt Enable. If a bus error has been detected, the CAN Controller requests the respective interrupt.	0	X
8	IDIE	ID Ready Interrupt Enable. When a CAN identifier has been received, the CAN Controller requests the respective interrupt.	0	X
9	TIE2	Transmit Interrupt Enable for Buffer2. When a message has been successfully transmitted out of TXB2 or Transmit Buffer 2 is accessible again (e.g. after an Abort Transmission command), the CAN Controller requests the respective interrupt.	0	X
10	TIE3	Transmit Interrupt Enable for Buffer3. When a message has been successfully transmitted out of TXB3 or Transmit Buffer 3 is accessible again (e.g. after an Abort Transmission command), the CAN Controller requests the respective interrupt.	0	X
31:11	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA	

CAN Interrupt and Capture Register

Table 321. CAN Interrupt and Capture Register (CAN1ICR - address 0x4004 400C, CAN2ICR - address 0x4004 800C) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RI[1]	0 (reset) 1 (set)	Receive Interrupt. This bit is set whenever the RBS bit in CANxSR and the RIE bit in CANxIER are both 1, indicating that a new message was received and stored in the Receive Buffer.	0	0
1	TI1	0 (reset) 1 (set)	Transmit Interrupt 1. This bit is set when the TBS1 bit in CANxSR goes from 0 to 1 (whenever a message out of TXB1 was successfully transmitted or aborted), indicating that Transmit buffer 1 is available, and the TIE1 bit in CANxIER is 1.	0	0
2	EI	0 (reset) 1 (set)	Error Warning Interrupt. This bit is set on every change (set or clear) of either the Error Status or Bus Status bit in CANxSR and the EIE bit bit is set within the Interrupt Enable Register at the time of the change.	0	X
3	DOI	0 (reset) 1 (set)	Data Overrun Interrupt. This bit is set when the DOS bit in CANxSR goes from 0 to 1 and the DOIE bit in CANxIER is 1.	0	0
4	WUI[2]	0 (reset) 1 (set)	Wake-Up Interrupt. This bit is set if the CAN controller is sleeping and bus activity is detected and the WUIE bit in CANxIER is 1.	0	0

CAN Interrupt and Capture Register

5	EPI	0 (reset) 1 (set)	Error Passive Interrupt. This bit is set if the EPIE bit in CANxIER is 1, and the CAN controller switches between Error Passive and Error Active mode in either direction. This is the case when the CAN Controller has reached the Error Passive Status (at least one error counter exceeds the CAN protocol defined level of 127) or if the CAN Controller is in Error Passive Status and enters the Error Active Status again.	0	0
6	ALI	0 (reset) 1 (set)	Arbitration Lost Interrupt. This bit is set if the ALIE bit in CANxIER is 1, and the CAN controller loses arbitration while attempting to transmit. In this case the CAN node becomes a receiver.	0	0
7	BEI	0 (reset) 1 (set)	Bus Error Interrupt -- this bit is set if the BEIE bit in CANxIER is 1, and the CAN controller detects an error on the bus.	0	X
8	IDI	0 (reset) 1 (set)	ID Ready Interrupt -- this bit is set if the IDIE bit in CANxIER is 1, and a CAN Identifier has been received (a message was successfully transmitted or aborted). This bit is set whenever a message was successfully transmitted or aborted and the IDIE bit is set in the IER register.	0	0
9	TI2	0 (reset) 1 (set)	Transmit Interrupt 2. This bit is set when the TBS2 bit in CANxSR goes from 0 to 1 (whenever a message out of TXB2 was successfully transmitted or aborted), indicating that Transmit buffer 2 is available, and the TIE2 bit in CANxIER is 1.	0	0
10	TI3	0 (reset) 1 (set)	Transmit Interrupt 3. This bit is set when the TBS3 bit in CANxSR goes from 0 to 1 (whenever a message out of TXB3 was successfully transmitted or aborted), indicating that Transmit buffer 3 is available, and the TIE3 bit in CANxIER is 1.	0	0
15:11	-	-	Reserved, user software should not write ones to reserved bits.	0	0

Acceptance filter

- This block provides a lookup table for received Identifiers (called Acceptance Filtering in CAN terminology) for all the CAN Controllers.
- It includes a 512 x 32 (2kB) RAM in which software maintains one to five tables of Identifiers.
  - This RAM can contain up to 1024 Standard Identifiers or 512 Extended Identifiers, or a mixture of both types
  - Fast hardware implemented search algorithm supporting a large number of CAN identifiers.
  - Global Acceptance Filter recognizes 11-bit and 29-bit Rx Identifiers for all CAN buses.
  - Allows definition of explicit and groups for 11-bit and 29-bit CAN identifiers.
- Acceptance Filter can provide **FullCAN-style** automatic reception for selected Standard Identifiers.

Acceptance filter II

- If the Acceptance Filter (AF) does not find a match in the appropriate individual Identifier table, it then searches the Identifier Range table for the size of Identifier signalled by the CAN controller. If the AF finds a match to a range in the table, it similarly signals the CAN controller to retain the message, and provides it with an ID Index value to store in its Receive Frame Status register.
- If the Acceptance Filter does not find a match in either the individual or Range table for the size of Identifier received, it signals the CAN controller to discard/ignore the received message.

Developer Community

Anonymous | Not applicable

Mar 19, 2014 07:35 AM

Version: **

Question: What is the difference between Basic CAN and Full CAN mailbox?

Answer:

- A Full CAN communication can be easily set up with the help of a GUI with a very limited amount of programming involved, whereas Basic CAN requires all the parameters to be set in the firmware.
- Full CAN uses hardware for message filtering. Basic CAN requires the CPU to be interrupted every time a message is received to determine whether it should be accepted.
- Full CAN can be used only for receiving a single type of message per mailbox, whereas Basic CAN configuration can accept messages with a range of identifiers per mailbox.
- The random transaction rate (RTR) feature is available only for the mailbox that is set as Full CAN. Basic CAN does not have the RTR feature enabled.

Acceptance Filter modes

Table 341. Acceptance filter modes and access control

Acceptance filter mode	Bit AccOff	Bit AccBP	Acceptance filter state	ID Look-up table RAM[1]	Acceptance filter config. registers	CAN controller message receive interrupt
Off Mode	1	0	reset & halted	r/w access from CPU	r/w access from CPU	no messages accepted
Bypass Mode	X	1	reset & halted	r/w access from CPU	r/w access from CPU	all messages accepted
Operating Mode and FullCAN Mode	0	0	running	read-only from CPU[2]	access from Acceptance filter only	hardware acceptance filtering

[1] The whole ID Look-up Table RAM is only word accessible.

[2] During the Operating Mode of the Acceptance Filter the Look-up Table can be accessed only to disable or enable Messages.

Configuration Register

Table 342. Section configuration register settings

ID-Look up Table Section	Register	Value	Section status
FullCAN (Standard Frame Format) Identifier Section	SFF_sa	= 0x000	disabled
		> 0x000	enabled
Explicit Standard Frame Format Identifier Section	SFF_GRP_sa	= SFF_sa	disabled
		> SFF_sa	enabled
Group of Standard Frame Format Identifier Section	EFF_sa	= SFF_GRP_sa	disabled
		> SFF_GRP_sa	enabled
Explicit Extended Frame Format Identifier Section	EFF_GRP_sa	= EFF_sa	disabled
		> EFF_sa	enabled
Group of Extended Frame Format Identifier Section	ENDofTable	= EFF_GRP_sa	disabled
		> EFF_GRP_sa	enabled

ID look-up table RAM

- The disable bits in Standard entries provide a means to turn response, to particular CAN Identifiers or ranges of Identifiers, on and off dynamically. When the Acceptance Filter function is enabled, only the disable bits in Acceptance Filter RAM can be changed by software.

Fig 59. Entry in FullCAN and individual standard identifier tables

Acceptance Filter Mode

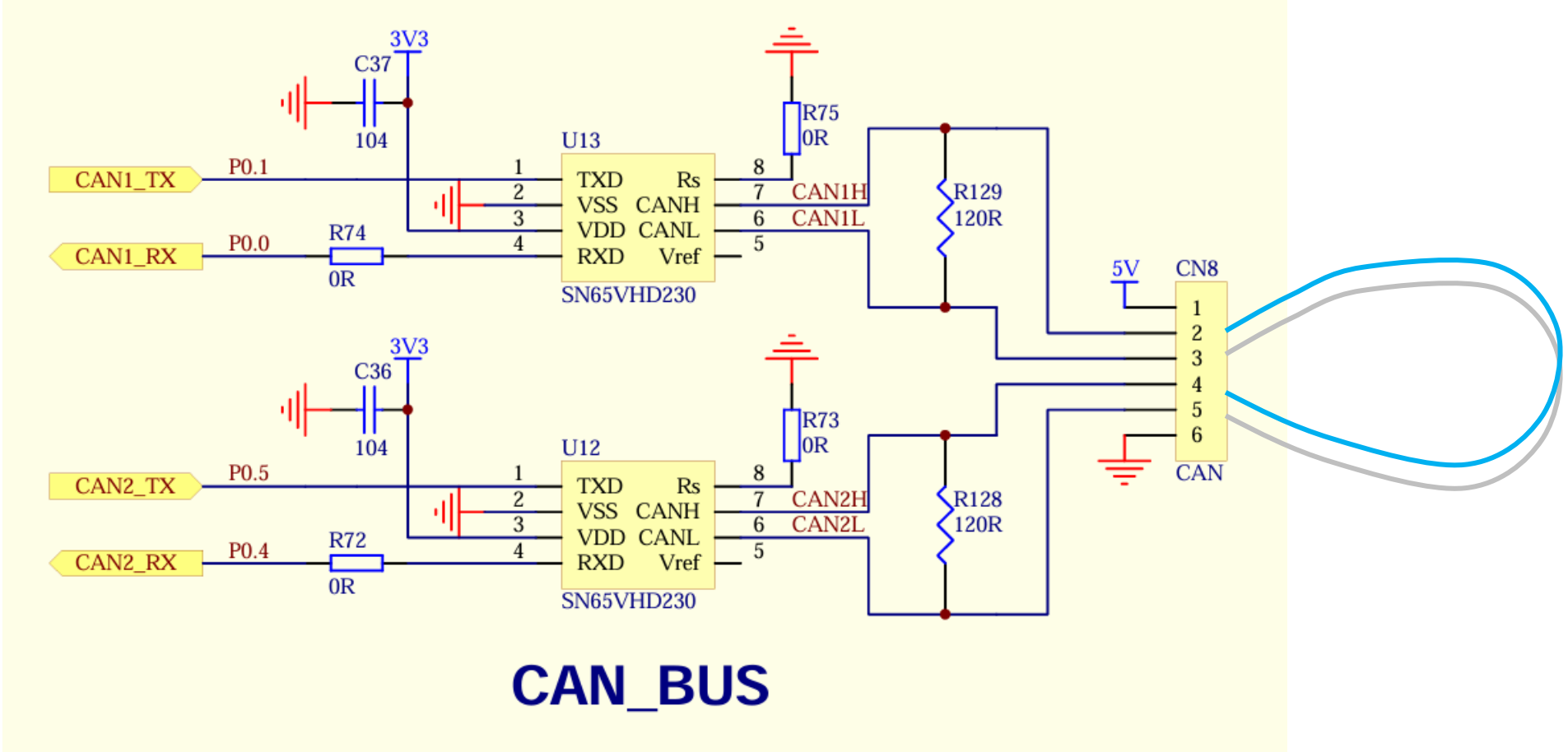
Table 343. Acceptance Filter Mode Register (AFMR - address 0x4003 C000) bit description

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.	
31:3	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

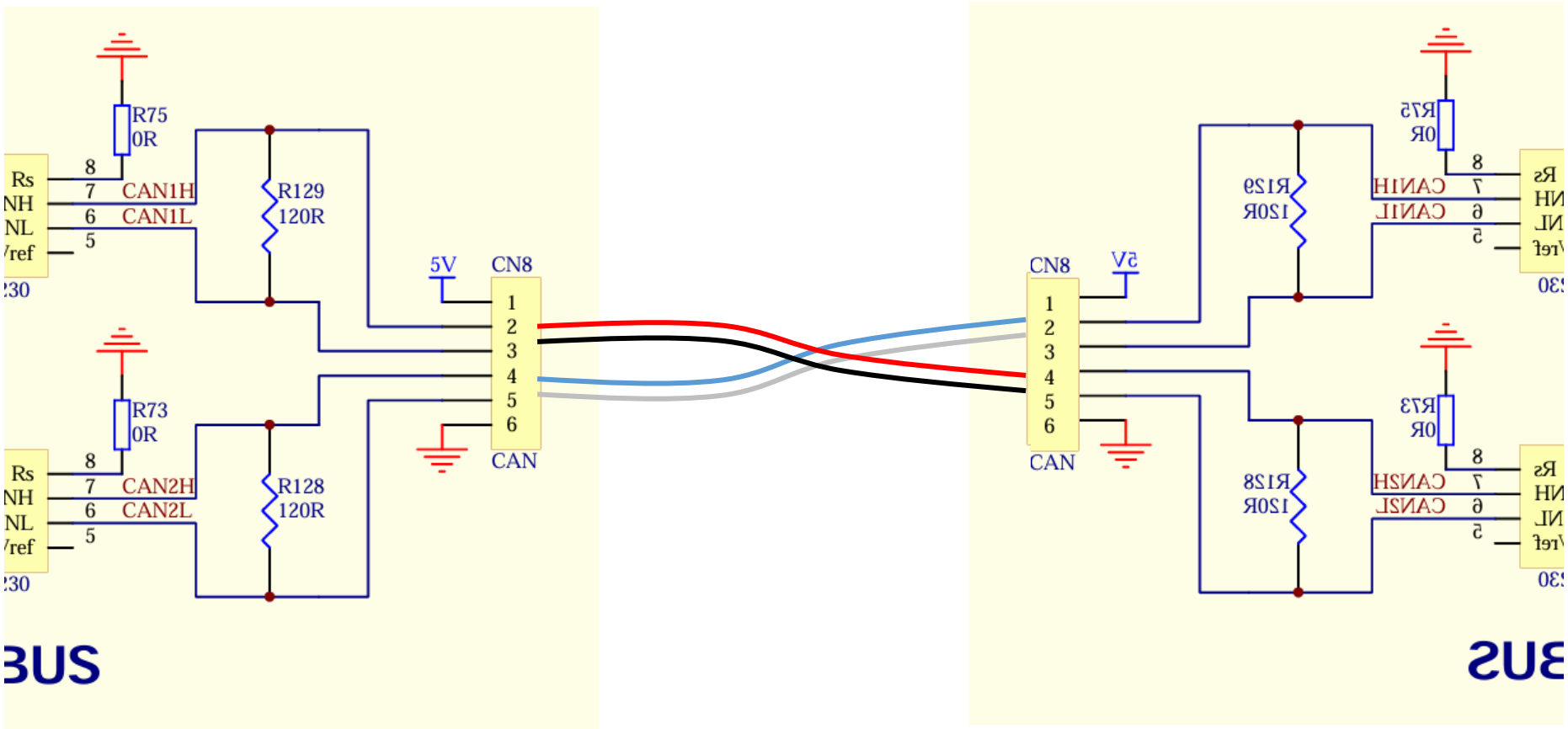
EXAMPLE - CAN Bus-support splatter-mirror visual effect

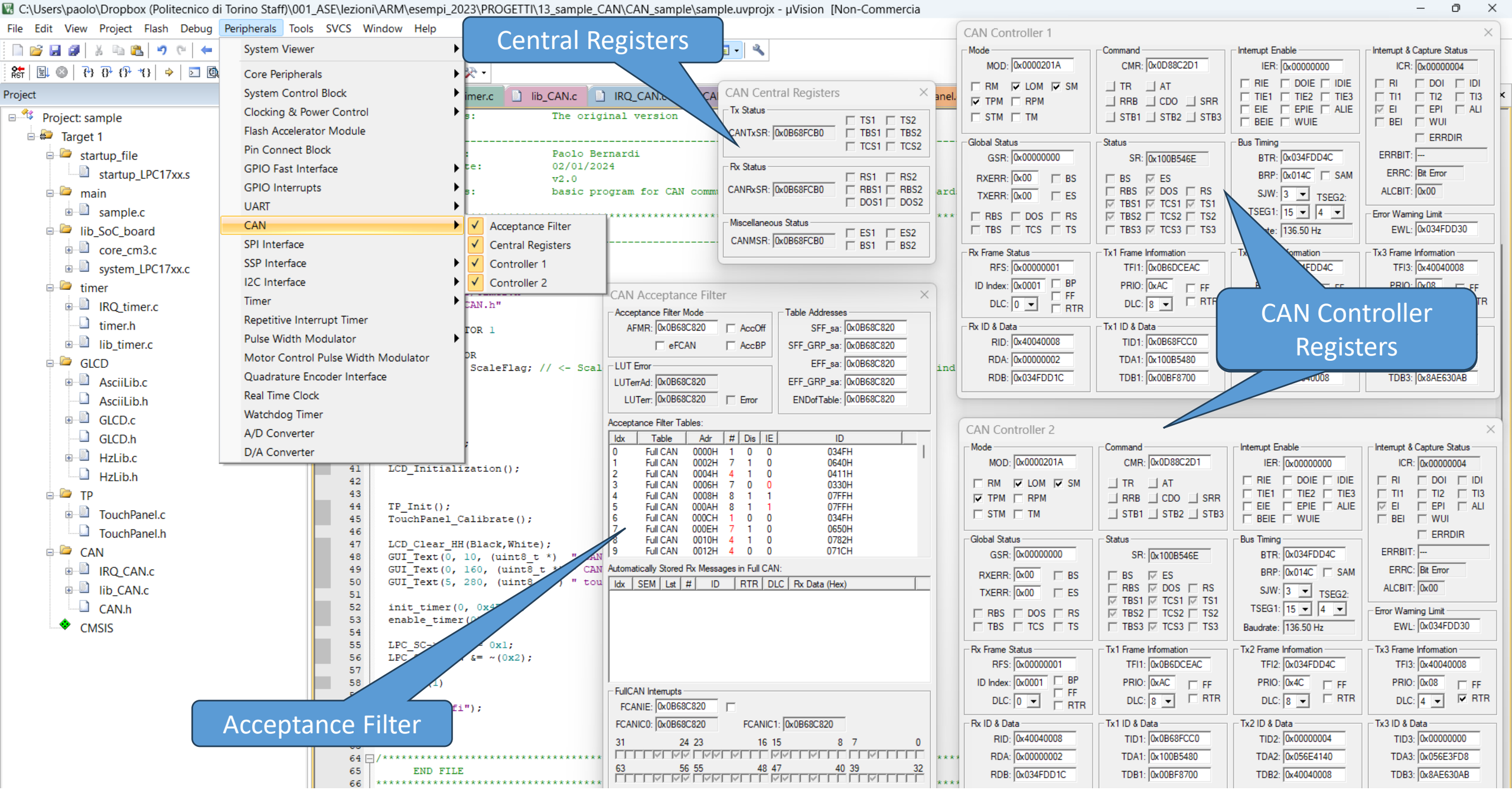
- Works with a single board with CAN channel 1 and 2 connected in loopback
- Works with two boards by a cross connection of CAN Channels
  - Board 1 – Can1 ←→ Board 2 – Can2
  - Board 2 – Can1 ←→ Board 1 – Can2

Loopback


3V3
C37
104
R75
0R
U13
CAN1_TX
P0.1
TXD
Rs
VSS
CANH
VDD
CANL
RXD
Vref
SN65VHD230
CAN1H
CAN1L
R74
0R
P0.0
CAN1_RX
R129
120R
5V
CN8
CAN
3V3
C36
104
R73
0R
U12
CAN2_TX
P0.5
CAN2H
CAN2L
R72
0R
P0.4
CAN2_RX
R128
120R
CAN_BUS

Crossed CAN Channels




C:\Users\paolo\Dropbox (Politecnico di Torino Staff)\001_ASE\lezioni\ARM\esempi_2023\PROGETTI\13_sample_CAN\CAN_sample\sample.uvprojx - µVision [Non-Commercia
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Central Registers
CAN Controller Registers
Acceptance Filter
System Viewer
Core Peripherals
System Control Block
Clocking & Power Control
Flash Accelerator Module
Pin Connect Block
GPIO Fast Interface
GPIO Interrupts
UART
CAN
SPI Interface
SSP Interface
I2C Interface
Timer
Repetitive Interrupt Timer
Pulse Width Modulator
Motor Control Pulse Width Modulator
Quadrature Encoder Interface
Real Time Clock
Watchdog Timer
A/D Converter
D/A Converter
Acceptance Filter
Central Registers
Controller 1
Controller 2
Project
Project: sample
Target 1
startup_file
startup_LPC17xx.s
main
sample.c
lib_SoC_board
core_cm3.c
system_LPC17xx.c
timer
IRQ_timer.c
timer.h
lib_timer.c
GLCD
AsciiLib.c
AsciiLib.h
GLCD.c
GLCD.h
HzLib.c
HzLib.h
TP
TouchPanel.c
TouchPanel.h
CAN
IRQ_CAN.c
lib_CAN.c
CAN.h
CMSIS
CAN Central Registers
Tx Status
CANTxSR: 0x0B68FCB0
Rx Status
CANRxSR: 0x0B68FCB0
Miscellaneous Status
CANMSR: 0x0B68FCB0
CAN Acceptance Filter
Acceptance Filter Mode
AFMR: 0x0B68C820
Table Addresses
SFF_sa: 0x0B68C820
SFF_GRP_sa: 0x0B68C820
EFF_sa: 0x0B68C820
EFF_GRP_sa: 0x0B68C820
ENDofTable: 0x0B68C820
LUT Error
LUTerrAd: 0x0B68C820
LUTerr: 0x0B68C820
Acceptance Filter Tables:
Automatically Stored Rx Messages in Full CAN:
FullCAN Interrupts
FCANIE: 0x0B68C820
FCANIC0: 0x0B68C820
FCANIC1: 0x0B68C820
CAN Controller 1
Mode
MOD: 0x0000201A
Command
CMR: 0x0D88C2D1
Interrupt Enable
IER: 0x00000000
Interrupt & Capture Status
ICR: 0x00000004
Global Status
GSR: 0x00000000
Status
SR: 0x100B546E
Bus Timing
BTR: 0x034FDD4C
Error Warning Limit
EWL: 0x034FDD30
Rx Frame Status
RFS: 0x00000001
Tx1 Frame Information
TFI1: 0x0B6DCEAC
Tx3 Frame Information
TFI3: 0x40040008
Rx ID & Data
RID: 0x40040008
RDA: 0x00000002
RDB: 0x034FDD1C
Tx1 ID & Data
TID1: 0x0B68FCC0
TDA1: 0x100B5480
TDB1: 0x00BF8700
TDB3: 0x8AE630AB
CAN Controller 2
Tx2 Frame Information
TFI2: 0x034FDD4C
Tx2 ID & Data
TID2: 0x00000004
TDA2: 0x056E4140
TDB2: 0x40040008
TID3: 0x00000000
TDA3: 0x056E3FD8

```
35
36
37  int main(void)
38  {
39    SystemInit();
40    CAN_Init();
41    LCD_Initialization();
42
43
44    TP_Init();
45    TouchPanel_Calibrate();
46
47    LCD_Clear_HH(Black,White);
48    GUI_Text(0, 10, (uint8_t *)   "
49    GUI_Text(0, 160, (uint8_t *) "
50    GUI_Text(5, 280, (uint8_t *) "
51
52    init_timer(0, 0x4E2 );
53    enable_timer(0);
54
55    LPC_SC->PCON |= 0x1;
56    LPC_SC->PCON &= ~(0x2);
57
58    while (1)
59    {
60      __ASM("wfi");
61    }
62  }
```

Random values are read from memory elements at reset

CAN Acceptance Filter

Acceptance Filter Mode
AFMR: 0x0B68C820 AccOff
eFCAN AccBP

Table Addresses
SFF_sa: 0x0B68C820
SFF_GRP_sa: 0x0B68C820
EFF_sa: 0x0B68C820
EFF_GRP_sa: 0x0B68C820
ENDofTable: 0x0B68C820

LUT Error
LUTerrAd: 0x0B68C820
LUTerr: 0x0B68C820 Error

Acceptance Filter Tables:

Idx	Table	Adr	#	Dis	IE	ID
0	Full CAN	0000H	1	0	0	034FH
1	Full CAN	0002H	7	1	0	0640H
2	Full CAN	0004H	4	1	0	0411H
3	Full CAN	0006H	7	0	0	0330H
4	Full CAN	0008H	8	1	1	07FFH
5	Full CAN	000AH	8	1	1	07FFH
6	Full CAN	000CH	1	0	0	034FH
7	Full CAN	000EH	7	1	0	0650H
8	Full CAN	0010H	4	1	0	0782H
9	Full CAN	0012H	4	0	0	071CH

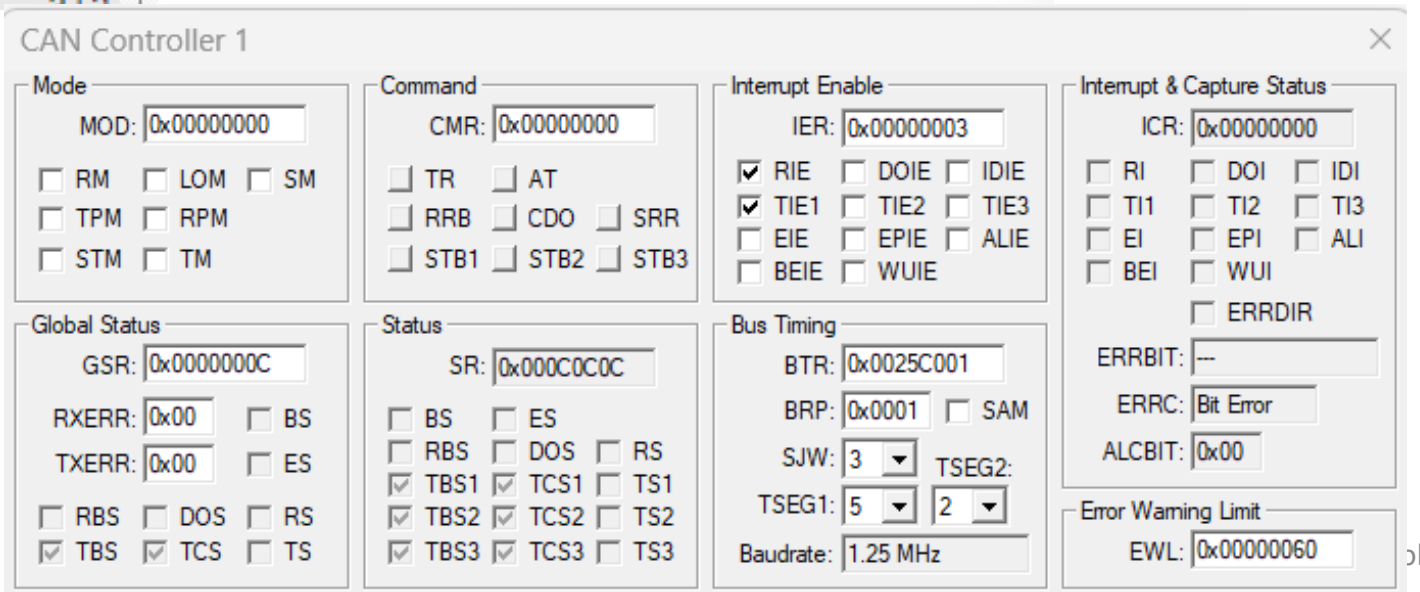
Automatically Stored Rx Messages in Full CAN:

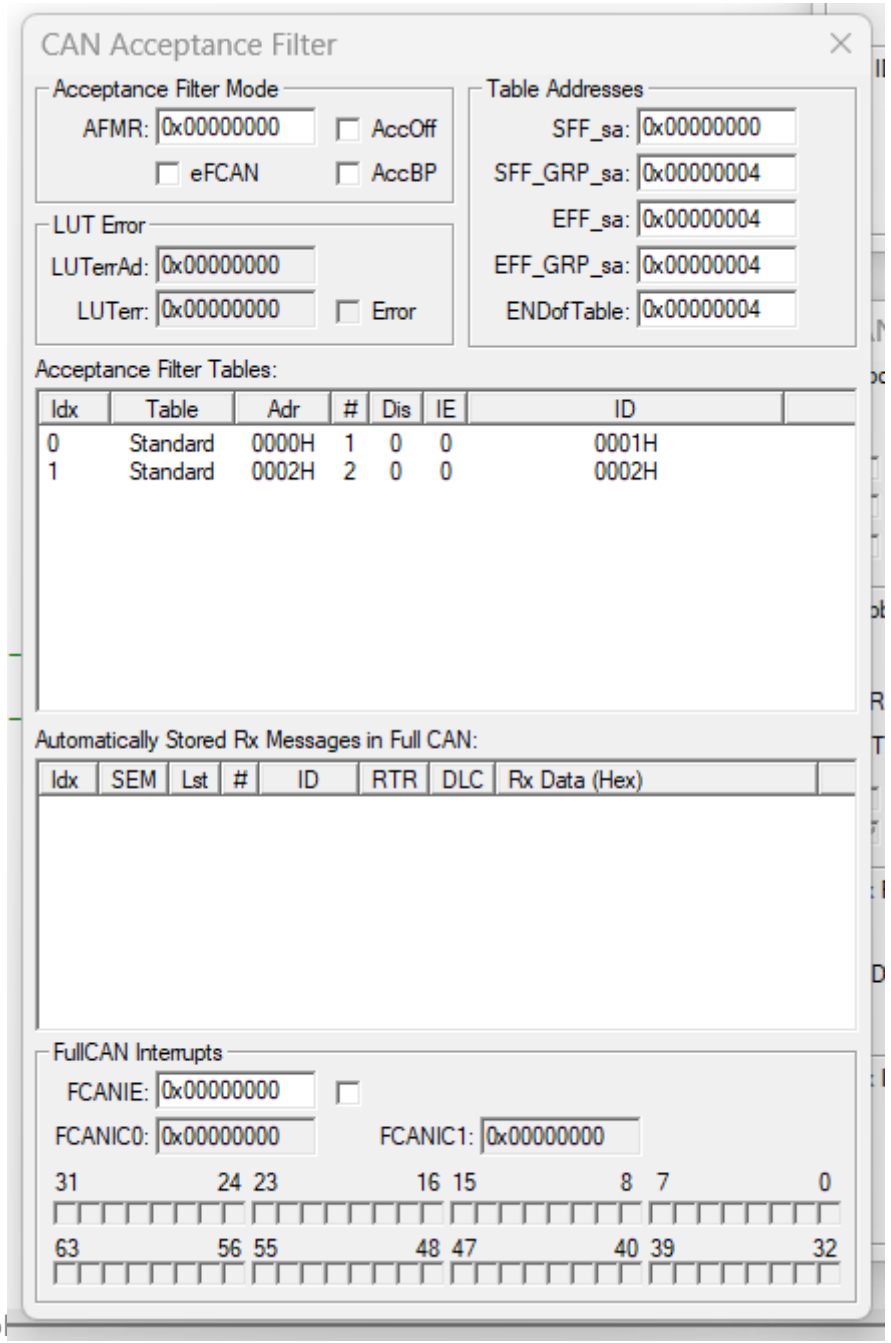
Idx	SEM	Lst	#	ID	RTR	DLC	Rx Data (Hex)

FullCAN Interrupts
FCANIE: 0x0B68C820
FCANIC0: 0x0B68C820 FCANIC1: 0x0B68C820

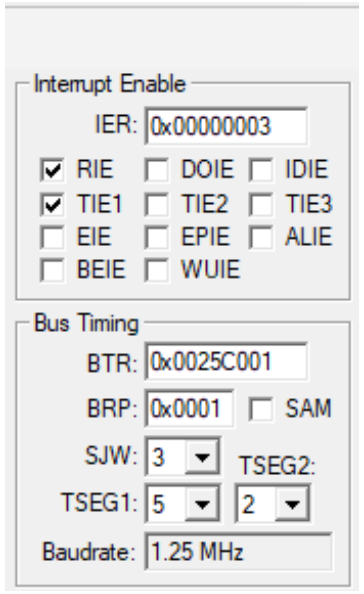
31 24 23 16 15 8 7 0
63 56 55 48 47 40 39 32

```
295 /*----------------------------------------
296   initialize CAN interface
297  *----------------------------------------*/
298 void CAN_Init (void) {
299
300   CAN_setup (1);
301   CAN_setup (2);
302
303   CAN_wrFilter (1, 1, STANDARD_FORMAT);
304   CAN_wrFilter (2, 2, STANDARD_FORMAT);
305
306   CAN_start (1);
307   CAN_start (2);
308
309   CAN_waitReady (1);
310   CAN_waitReady (2);
311 }
```

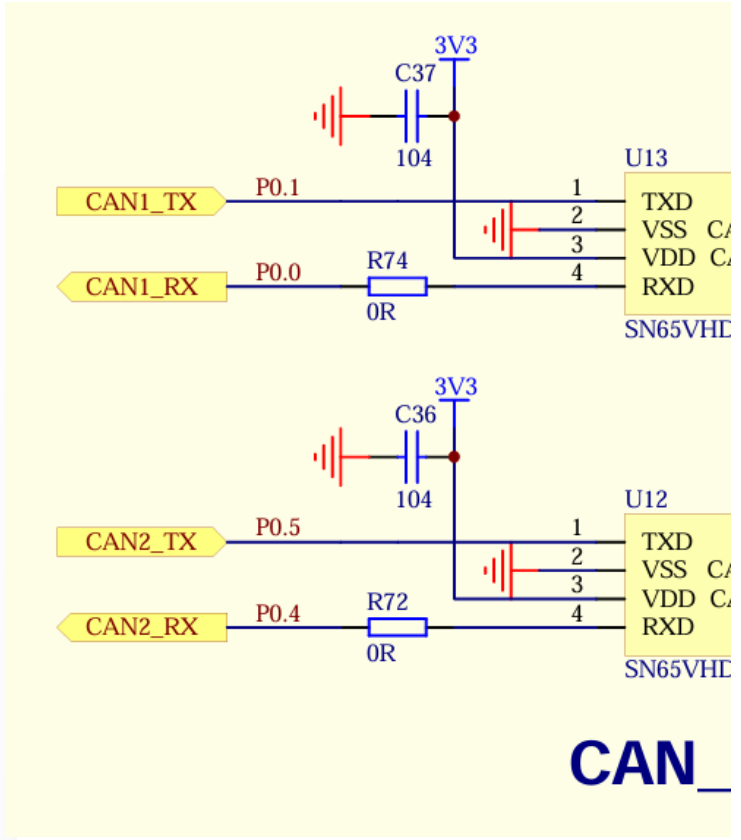


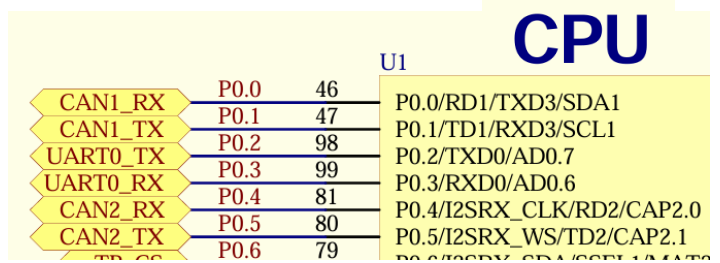


CAN_setup & CAN_start functions




```
/*----------------------------------------------------------------------------
  setup CAN interface.  CAN controller (1..2)
 *----------------------------------------------------------------------------*/
void CAN_setup (uint32_t ctrl)  {
  LPC_CAN_TypeDef *pCAN = (ctrl == 1) ? LPC_CAN1 : LPC_CAN2;

  if (ctrl == 1) {
    LPC_SC->PCONP       |=  (1 << 13);          /* Enable power to CAN1 block */
    LPC_PINCON->PINSEL0 |=  (1 <<  0);          /* Pin P0.0 used as RD1 (CAN1) */
    LPC_PINCON->PINSEL0 |=  (1 <<  2);          /* Pin P0.1 used as TD1 (CAN1) */

    NVIC_EnableIRQ(CAN_IRQn);                   /* Enable CAN interrupt */
  } else {
    LPC_SC->PCONP       |=  (1 << 14);          /* Enable power to CAN2 block */
    LPC_PINCON->PINSEL0 |=  (1 <<  9);          /* Pin P0.4 used as RD2 (CAN2) */
    LPC_PINCON->PINSEL0 |=  (1 << 11);          /* Pin P0.5 used as TD2 (CAN2) */

    NVIC_EnableIRQ(CAN_IRQn);                   /* Enable CAN interrupt */
  }

  LPC_CANAF->AFMR = 2;                          /* By default filter is not used */
  pCAN->MOD   = 1;                              /* Enter reset mode */
  pCAN->IER   = 0;                              /* Disable all interrupts */
  pCAN->GSR   = 0;                              /* Clear status register */
  CAN_cfgBaudrate(ctrl, 1000000);               /* Set bit timing */
  pCAN->IER   = 0x0003;                         /* Enable Tx and Rx interrupt */
}

/*----------------------------------------------------------------------------
  leave initialisation mode.  CAN controller (1..2)
 *----------------------------------------------------------------------------*/
void CAN_start (uint32_t ctrl)  {
  LPC_CAN_TypeDef *pCAN = (ctrl == 1) ? LPC_CAN1 : LPC_CAN2;

  pCAN->MOD = 0;                                /* Enter normal operating mode */
}
```





CAN.h structure

```
21  #define STANDARD_FORMAT  0
22  #define EXTENDED_FORMAT  1
23
24  #define DATA_FRAME       0
25  #define REMOTE_FRAME     1
26  extern uint32_t result;
27  extern uint8_t icr;
28
29  typedef struct  {
30    unsigned int   id;                    /* 29 bit identifier */
31    unsigned char  data[8];               /* Data field */
32    unsigned char  len;                   /* Length of data field in bytes */
33    unsigned char  format;                /* 0 - STANDARD, 1- EXTENDED IDENTIFIER */
34    unsigned char  type;                  /* 0 - DATA FRAME, 1 - REMOTE FRAME */
35  } CAN_msg;
36
37  /* Functions defined in module CAN.c */
38  void CAN_setup          (uint32_t ctrl);
39  void CAN_start          (uint32_t ctrl);
40  void CAN_waitReady      (uint32_t ctrl);
41  void CAN_wrMsg          (uint32_t ctrl, CAN_msg *msg);
42  void CAN_rdMsg          (uint32_t ctrl, CAN_msg *msg);
43  void CAN_wrFilter       (uint32_t ctrl, uint32_t id, uint8_t filter_type);
44  void CAN_Init           (void);
```

Transmit or Receive messages

Message is prepared

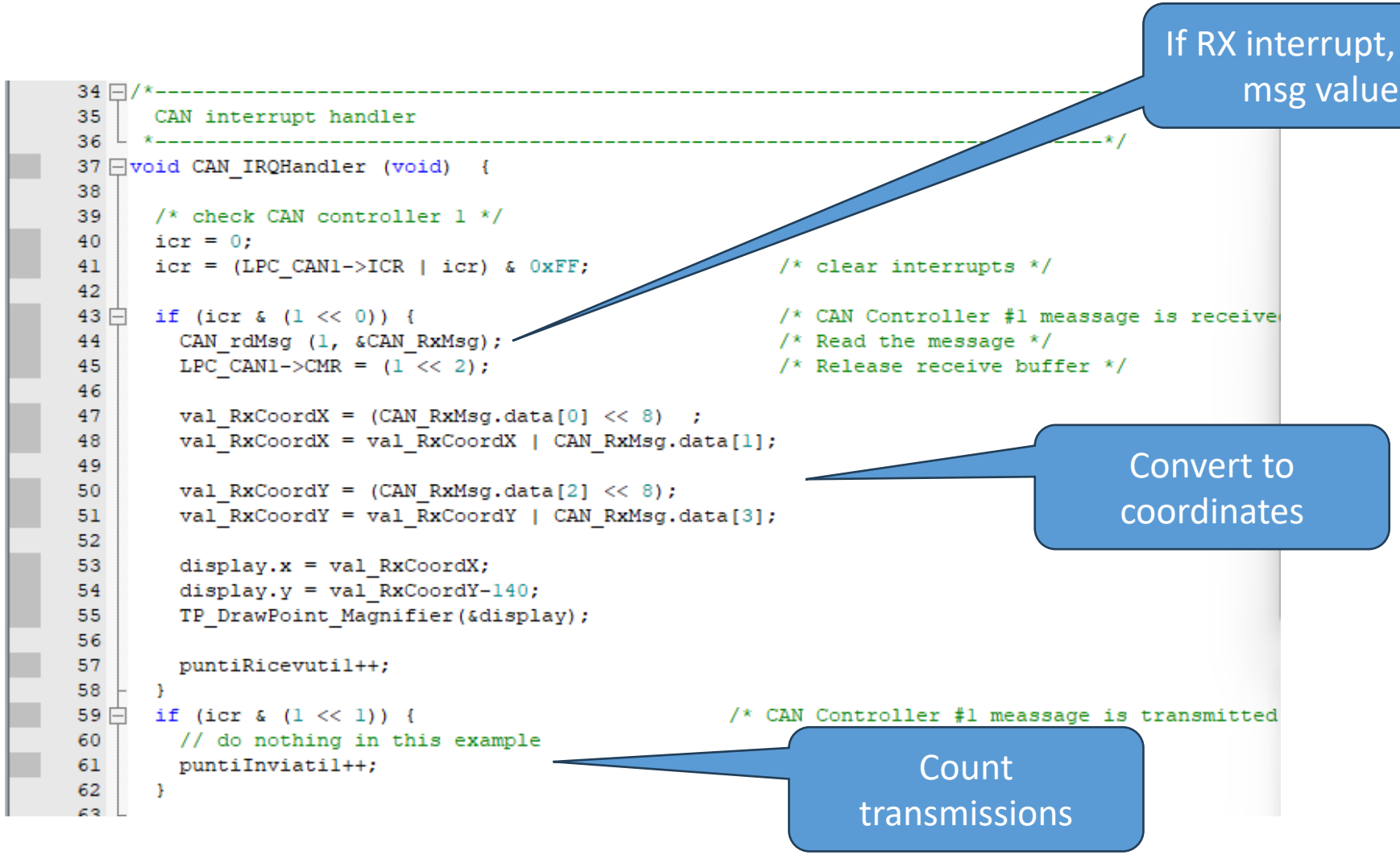
Message is sent

Message is prepared

All information are collected

```
124 /*----------------------------------------------------------------------------
125   write a message to CAN peripheral and transmit it.  CAN controller (1..2)
126  *----------------------------------------------------------------------------*/
127 void CAN_wrMsg (uint32_t ctrl, CAN_msg *msg)  {
128   LPC_CAN_TypeDef *pCAN = (ctrl == 1) ? LPC_CAN1 : LPC_CAN2;
129   uint32_t CANData;
130
131   CANData = ((((uint32_t) msg->len) << 16)     & 0x000F0000) |
132             (msg->format == EXTENDED_FORMAT ) * 0x80000000 |
133             (msg->type   == REMOTE_FRAME)     * 0x40000000;
134
135   if (pCAN->SR & (1<<2))  {                          /* Transmit buffer 1 free */
136     pCAN->TFI1  = CANData;                           /* Write frame informations */
137     pCAN->TID1 = msg->id;                            /* Write CAN message identifier */
138     pCAN->TDA1 = *(uint32_t *) &msg->data[0];        /* Write first 4 data bytes */
139     pCAN->TDB1 = *(uint32_t *) &msg->data[4];        /* Write second 4 data bytes */
140     pCAN->CMR  = 0x21;                               /* Start transmission without loop-back */
141   }
142 }
143
144 /*----------------------------------------------------------------------------
145   read a message from CAN peripheral and release it.  CAN controller (1..2)
146  *----------------------------------------------------------------------------*/
147 void CAN_rdMsg (uint32_t ctrl, CAN_msg *msg)  {
148   LPC_CAN_TypeDef *pCAN = (ctrl == 1) ? LPC_CAN1 : LPC_CAN2;
149   uint32_t CANData;
150
151                                                      /* Read frame informations */
152   CANData = pCAN->RFS;
153   msg->format   = (CANData & 0x80000000) == 0x80000000;
154   msg->type     = (CANData & 0x40000000) == 0x40000000;
155   msg->len      = ((uint8_t)(CANData >> 16)) & 0x0F;
156
157   msg->id = pCAN->RID;                               /* Read CAN message identifier */
158
159   if (msg->type == DATA_FRAME)  {                    /* Read the data if received message was DA
160     *(uint32_t *) &msg->data[0] = pCAN->RDA;
161     *(uint32_t *) &msg->data[4] = pCAN->RDB;
162   }
163 }
```

CAN IRQ




```
34 /*----------------------------------------------------------------------------
35   CAN interrupt handler
36  *----------------------------------------------------------------------------*/
37 void CAN_IRQHandler (void)  {
38
39   /* check CAN controller 1 */
40   icr = 0;
41   icr = (LPC_CAN1->ICR | icr) & 0xFF;               /* clear interrupts */
42
43   if (icr & (1 << 0)) {                             /* CAN Controller #1 meassage is receive
44     CAN_rdMsg (1, &CAN_RxMsg);                      /* Read the message */
45     LPC_CAN1->CMR = (1 << 2);                       /* Release receive buffer */
46
47     val_RxCoordX = (CAN_RxMsg.data[0] << 8)  ;
48     val_RxCoordX = val_RxCoordX | CAN_RxMsg.data[1];
49
50     val_RxCoordY = (CAN_RxMsg.data[2] << 8);
51     val_RxCoordY = val_RxCoordY | CAN_RxMsg.data[3];
52
53     display.x = val_RxCoordX;
54     display.y = val_RxCoordY-140;
55     TP_DrawPoint_Magnifier(&display);
56
57     puntiRicevutil++;
58   }
59   if (icr & (1 << 1)) {                           /* CAN Controller #1 meassage is transmitted
60     // do nothing in this example
61     puntiInviatil++;
62   }
```