CAN Communication

using STM32F4 Discovery Board

Rational software

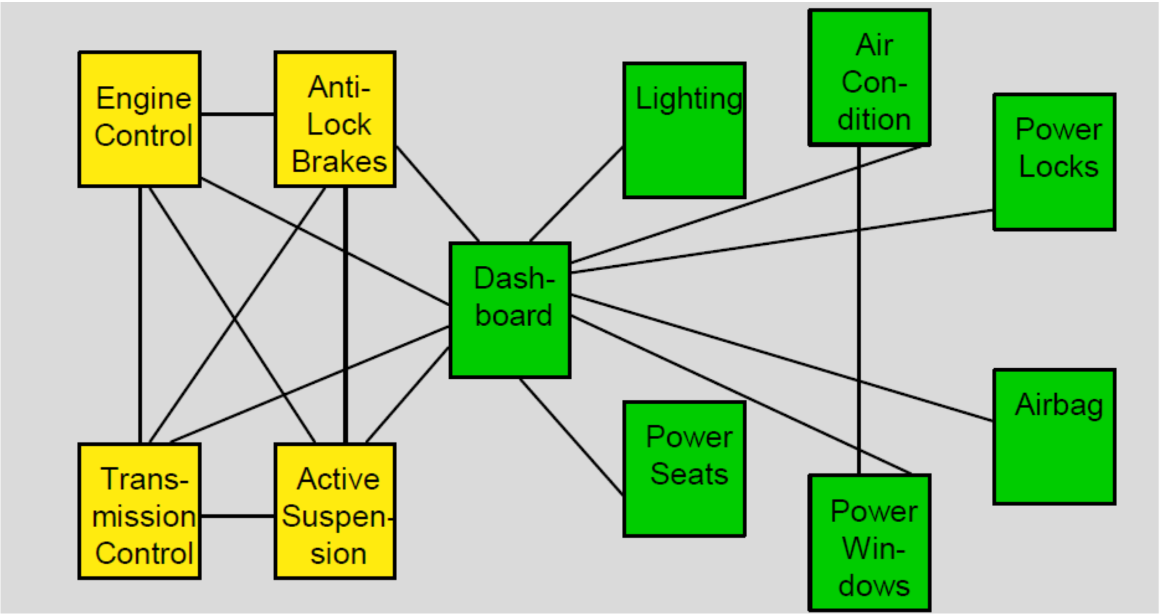
CAN History

1. In **1985** Bosch originally developed CAN, a high-integrity serial bus system for networking intelligent devices, to replace automotive point-to-point wiring systems.
2. As vehicle electronics became pervasive, complex wire harnesses which were heavy, expensive and bulky were replaced with CAN throughout the automotive industry.
3. In **1993** CAN became the international standard known as ISO 11898.
4. Since **1994**, several widely used higher-level protocols have been standardized on top of CAN, such as **CANopen*** and DeviceNet.
5. In **1996** the OnBoard Diagnostics OBD-II standard which incorporates CAN becomes mandatory for all cars and light trucks sold in the United States.
6. **Today** markets including surface transportation, industrial automation, maritime and avionics systems have widely adopted CAN.
7. **Today** CAN is incorporated into many microcontrollers

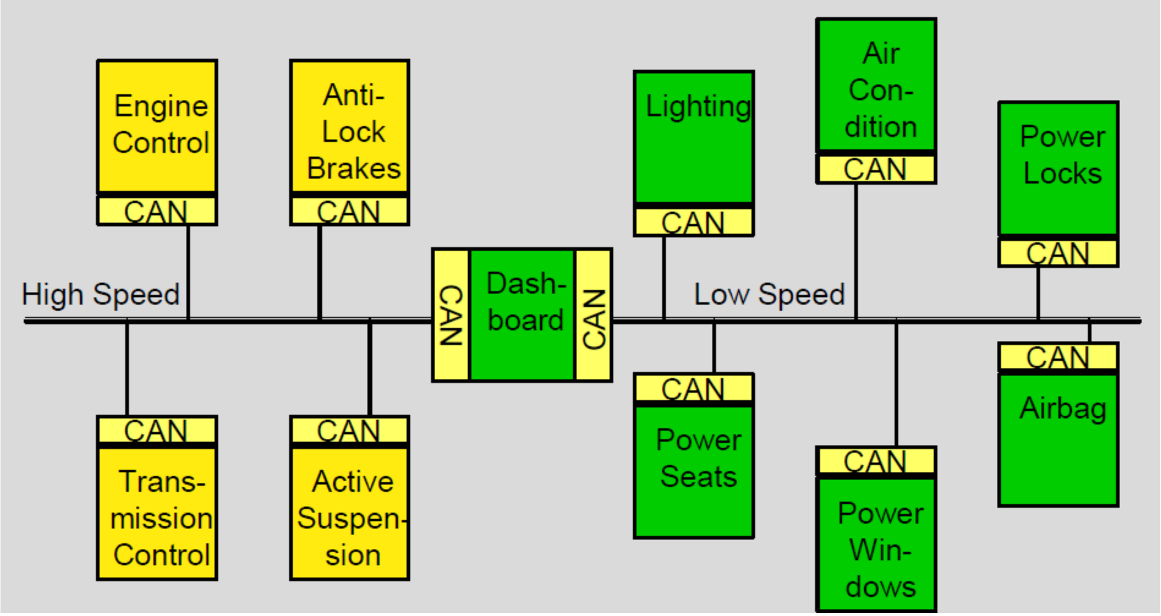
Vehicles Before CAN: Expensive, bulky point to point wiring, wiring harnesses and many connectors.

Vehicles After CAN: Systems of Systems with multiple CAN busses, simplified wiring harnesses and many Fewer connectors

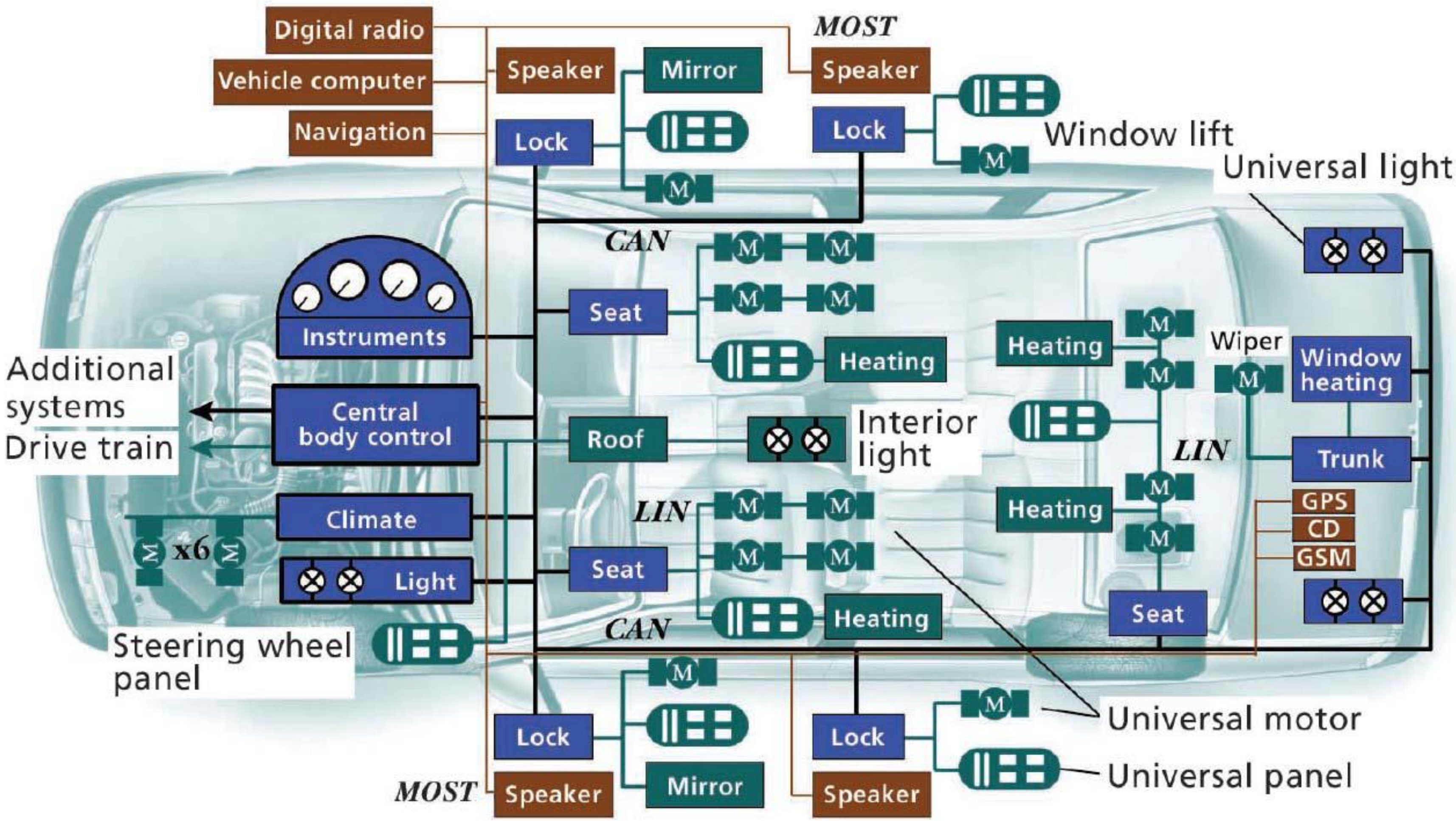
Before CAN



After CAN



CAN is Now Central to Automotive Networks



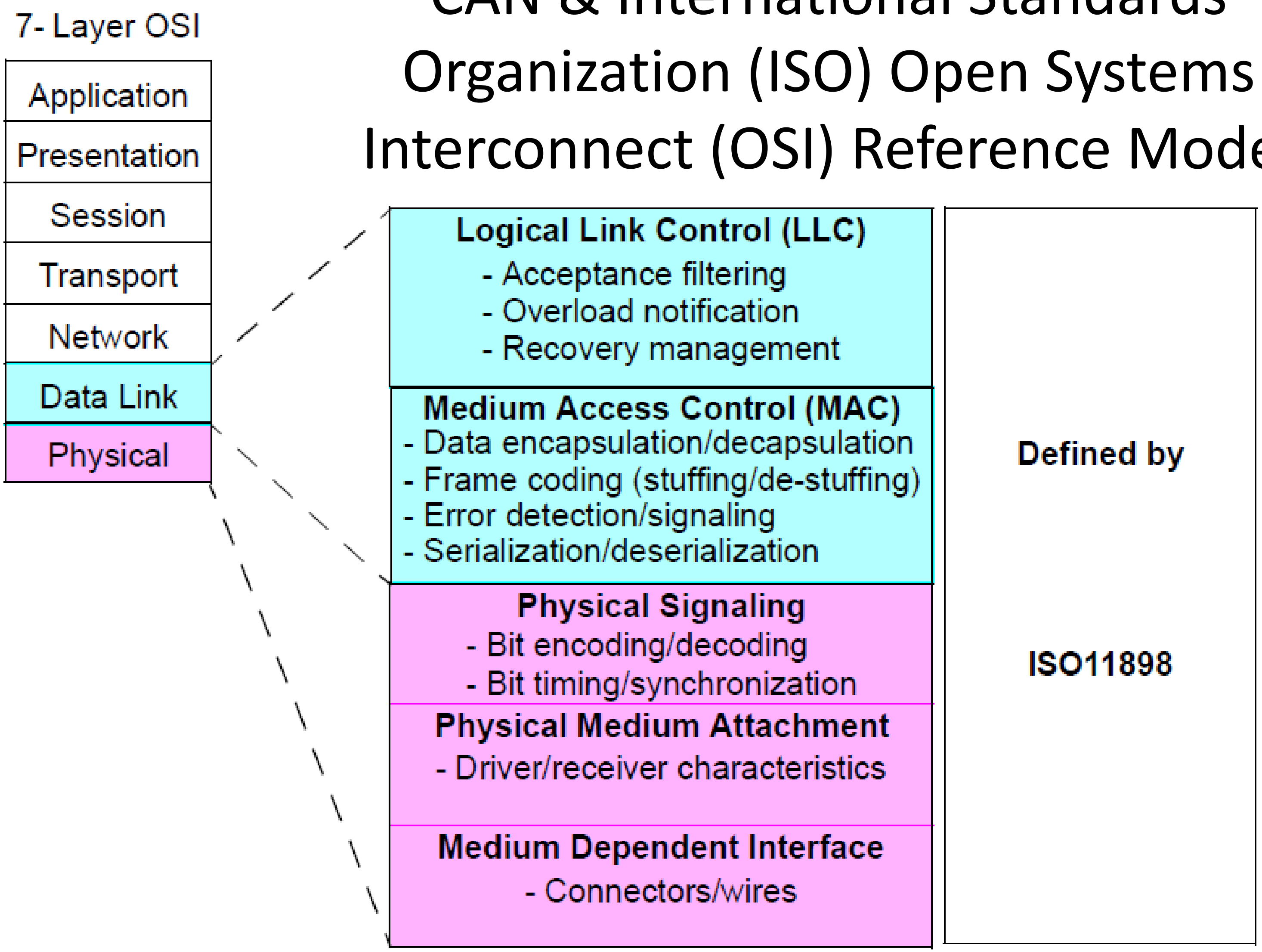
CAN Controller area network
GPS Global Positioning System
GSM Global System for Mobile Communications
LIN Local interconnect network
MOST Media-oriented systems transport

New cars typically contain 50 to 100 microcontrollers

Advantages of CAN

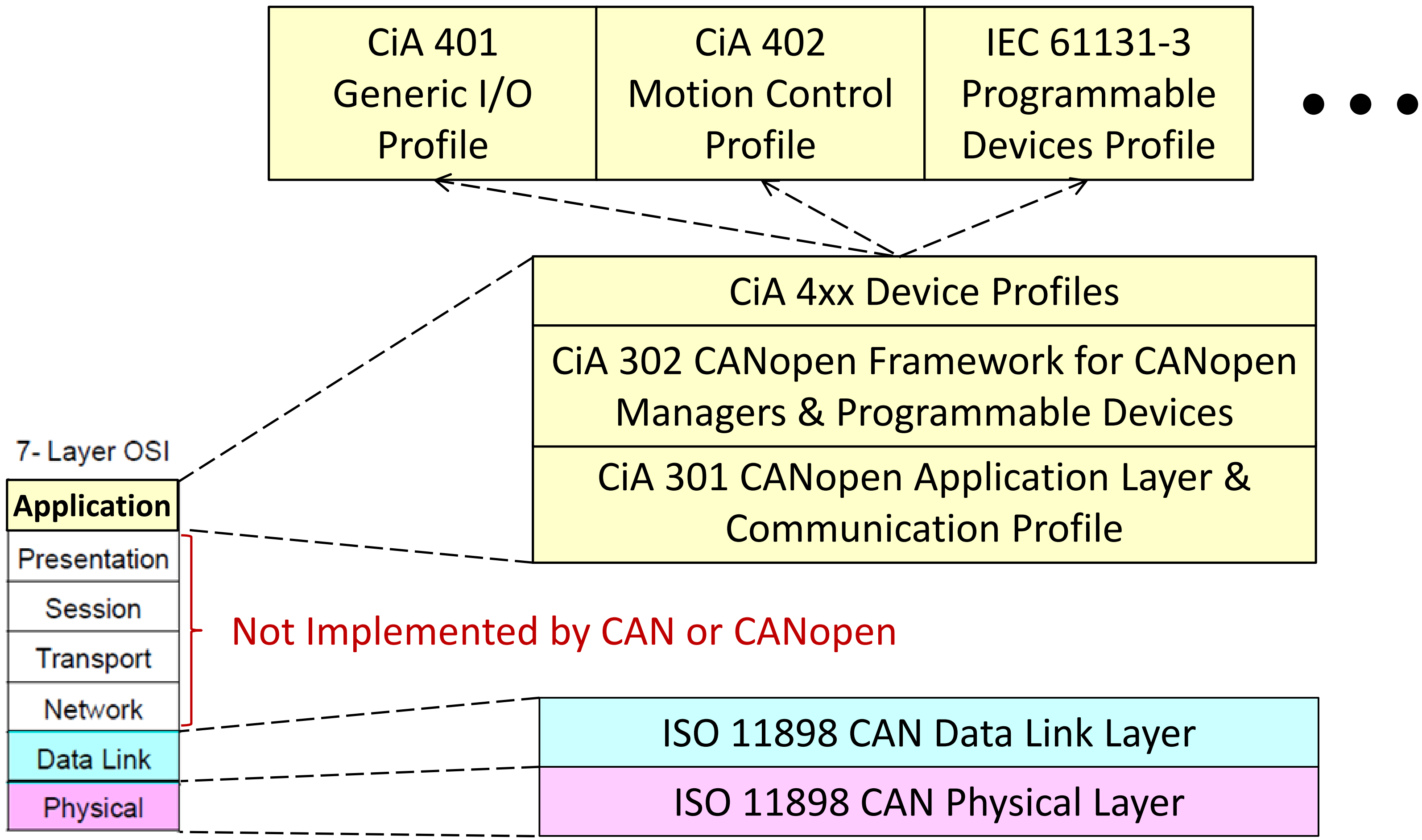
1. Low cost network infrastructure which is often built into microcontrollers.
2. Large market segment with broad availability of hardware, software and systems engineering tools.
3. Light weight, low latency, highly deterministic design specifically for real-time embedded applications.
4. Reliable with strong error detection, fault tolerant versions available.
5. Flexible and highly configurable with various higher level application protocols.
6. Foundation for next generation technology controller area networks.

CAN & International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model



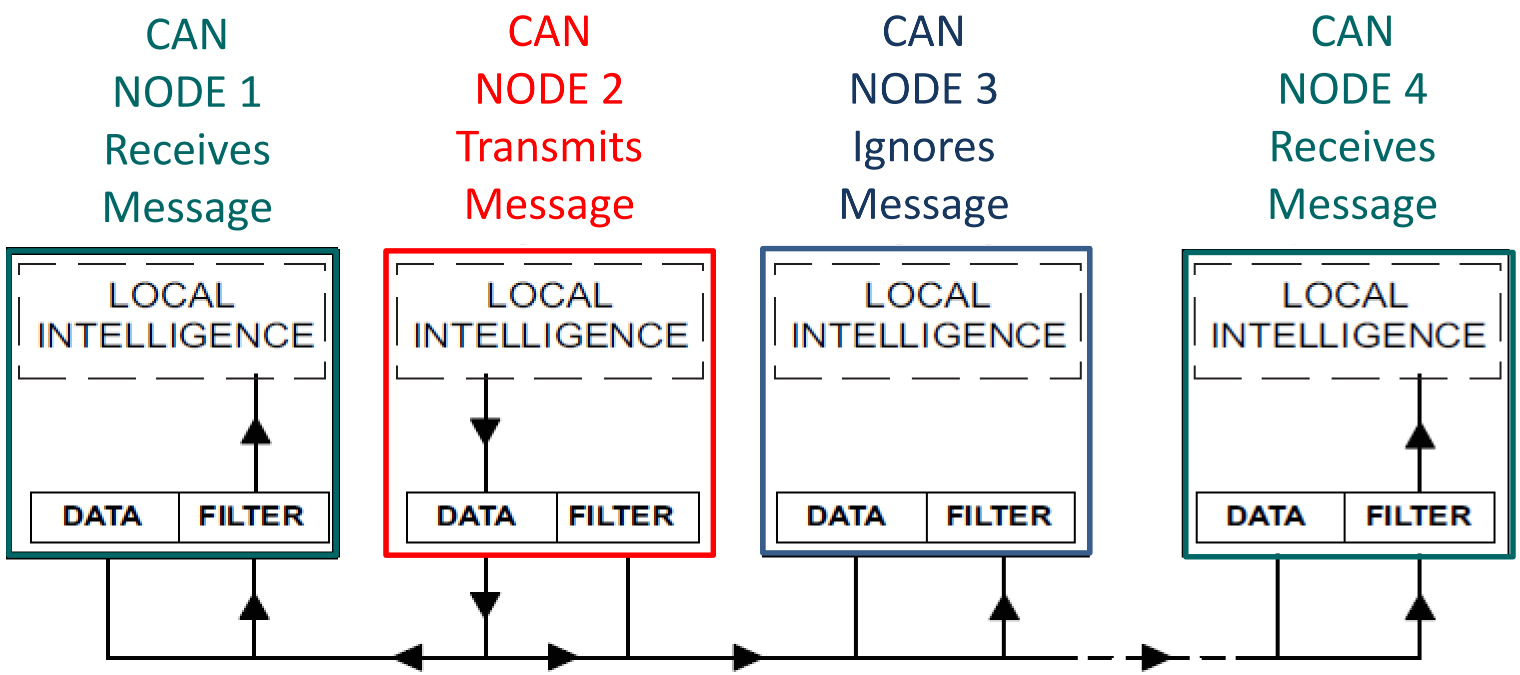
High level CAN Protocols implement Application layer and skip the four intervening layers

The CANopen Application



High level CAN Protocols implement Application layers and skip the four intervening layers

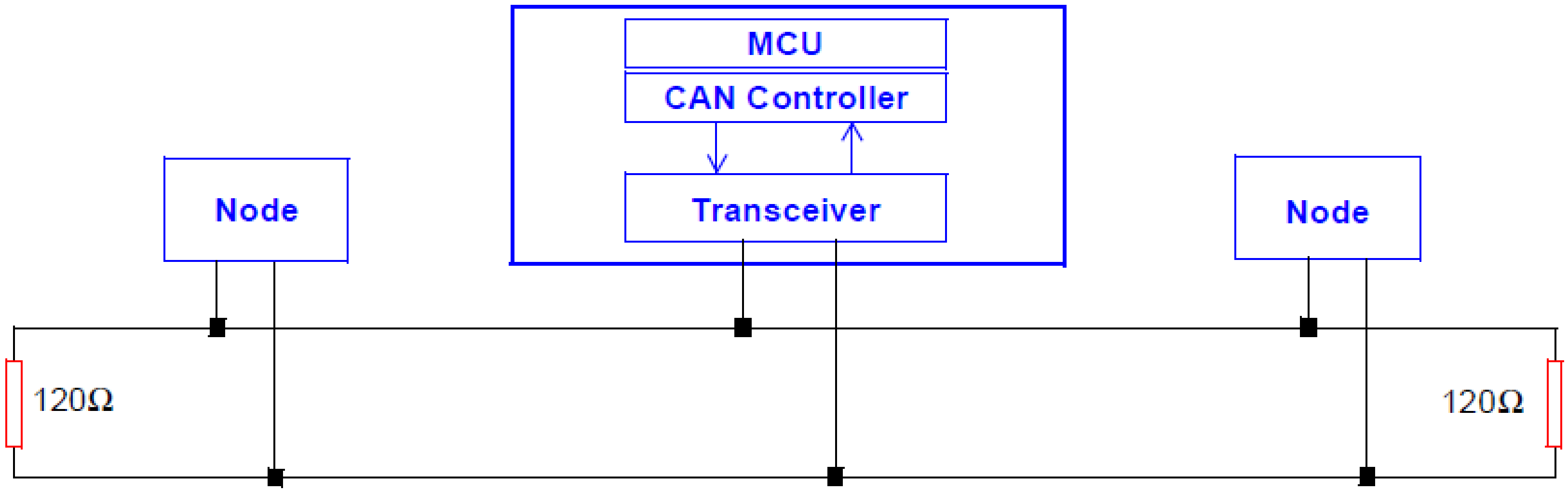
CAN Data-Flow Model



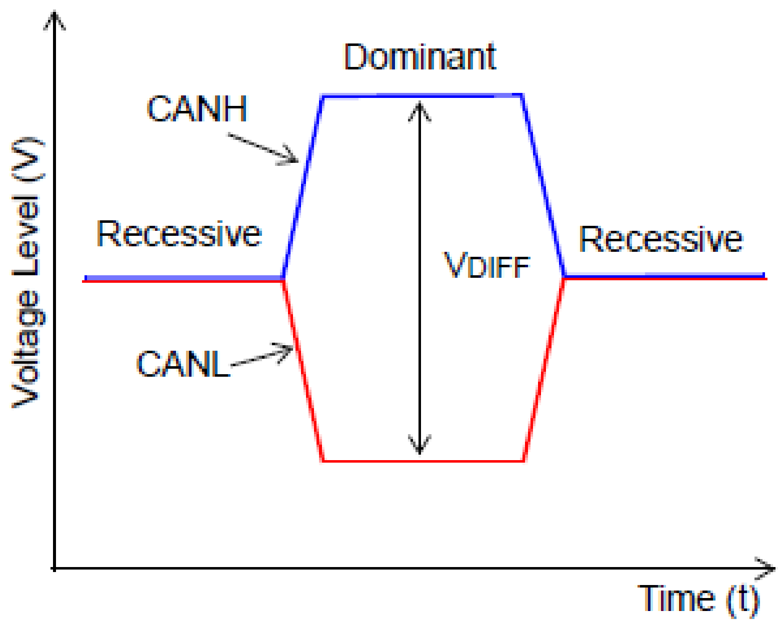
One node transmits, all nodes listen and processor data frames selectively. Message filtering is typically performed in transceiver hardware. This data flow supports a broad range of network communication models:

1. Master / Slave : All communications initialed by master node
2. Peer-to-Peer : Nodes interact with autonomously with equal authority
3. Producer / Consumer : Producer nodes broadcast (push) messages to Consumer nodes
4. Client / Server : Client nodes request (pull) data from Server nodes

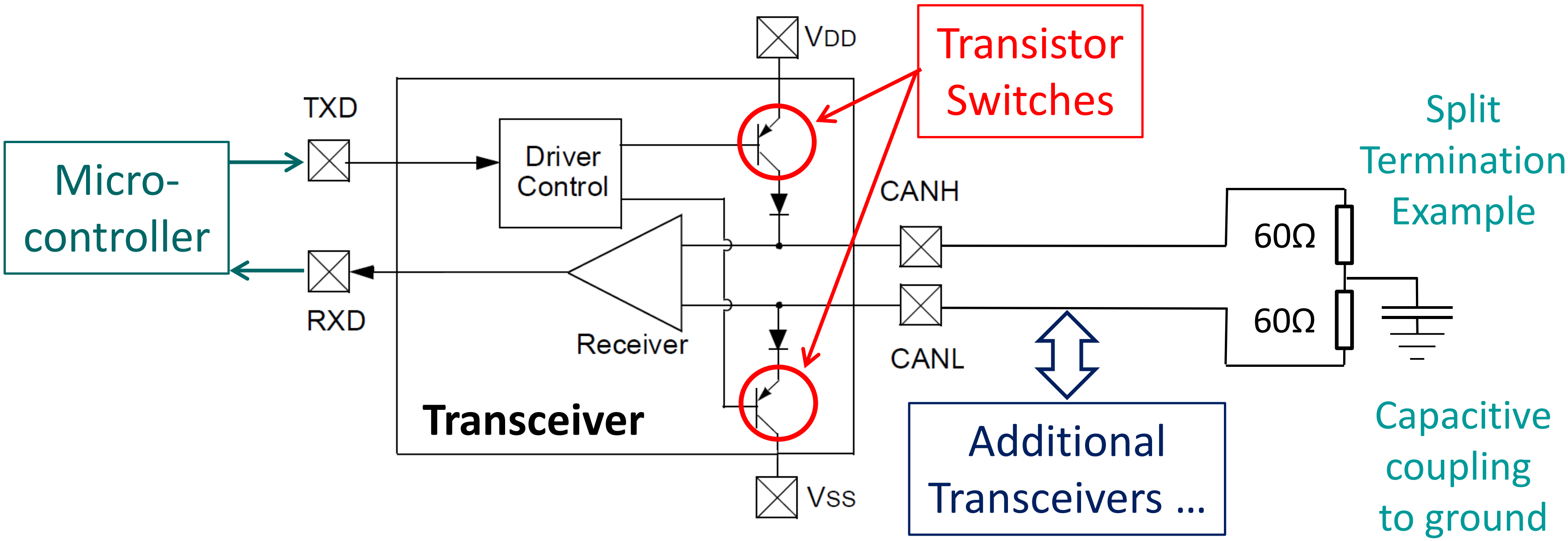
CAN Typical High-Speed Physical Layer



- CAN uses differential signaling to improve signal to noise ratio. Termination resistors reduce signal reflection.
- Idle bus state is **Recessive** with no applied differential signal: $V_{CAN\_H} \approx V_{CAN\_L}$
- **Dominant** state occurs when one or more nodes drive the bus state to: V_{DIFF}

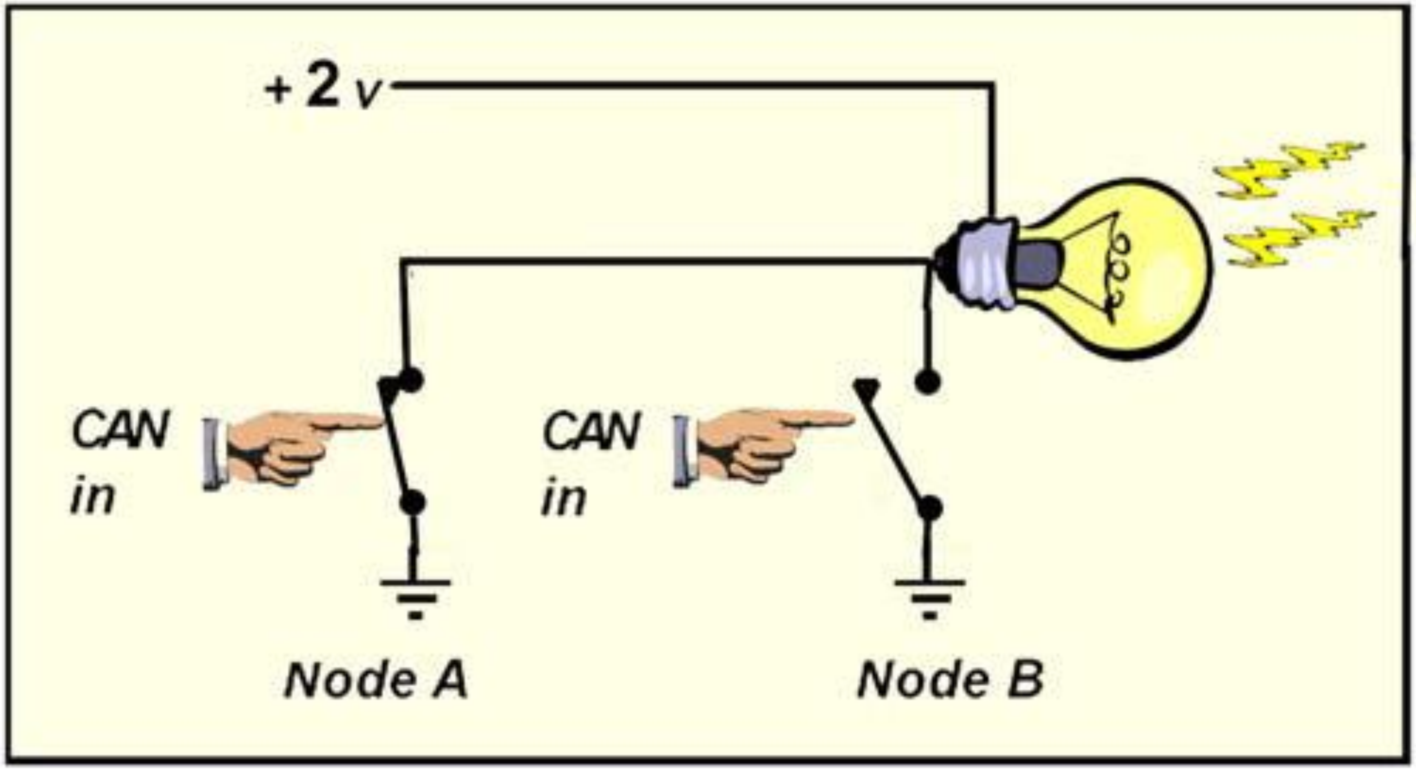


CAN Differential Bus Interface Transceivers



- The CAN idle state presents a recessive state, signaled by a small differential voltage across CANH and CANL. With the indicated split termination, this idle voltage will be halfway between VDD (positive supply) and VSS (ground).
- The CAN dominant state occurs when one or more transceivers simultaneously close the indicated transistor switches driving CANH and CANL toward VDD and VSS, respectively.
- This open collector transistor switch configuration is referred to as a **"wired or"** since any node transmitting a dominant bit always overrides a recessive bit. Since a dominant bit represents a logic 0, this arrangement is sometimes referred to as **"wired and"** since bus a logic "1" state is achieved only if all nodes (node 1 **AND** node 2 **AND** node 3 …) signal logic "1" recessive bits).

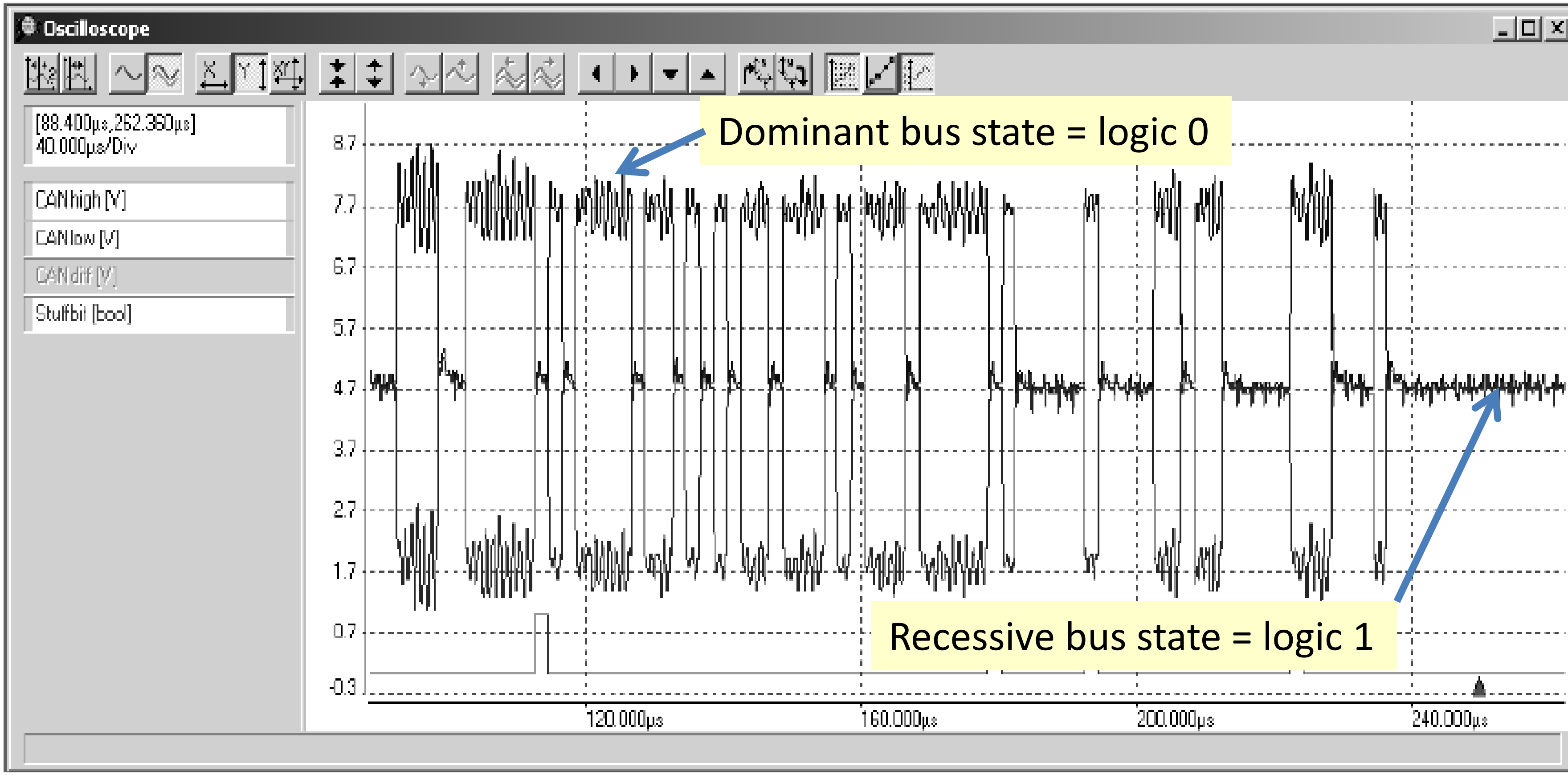
Example of a **“Wired OR”**



Closing Node A switch **OR** closing Node B switch turns on the light.

Conversely, the light is off unless Node A switch is open **AND** Node B switch is also open.

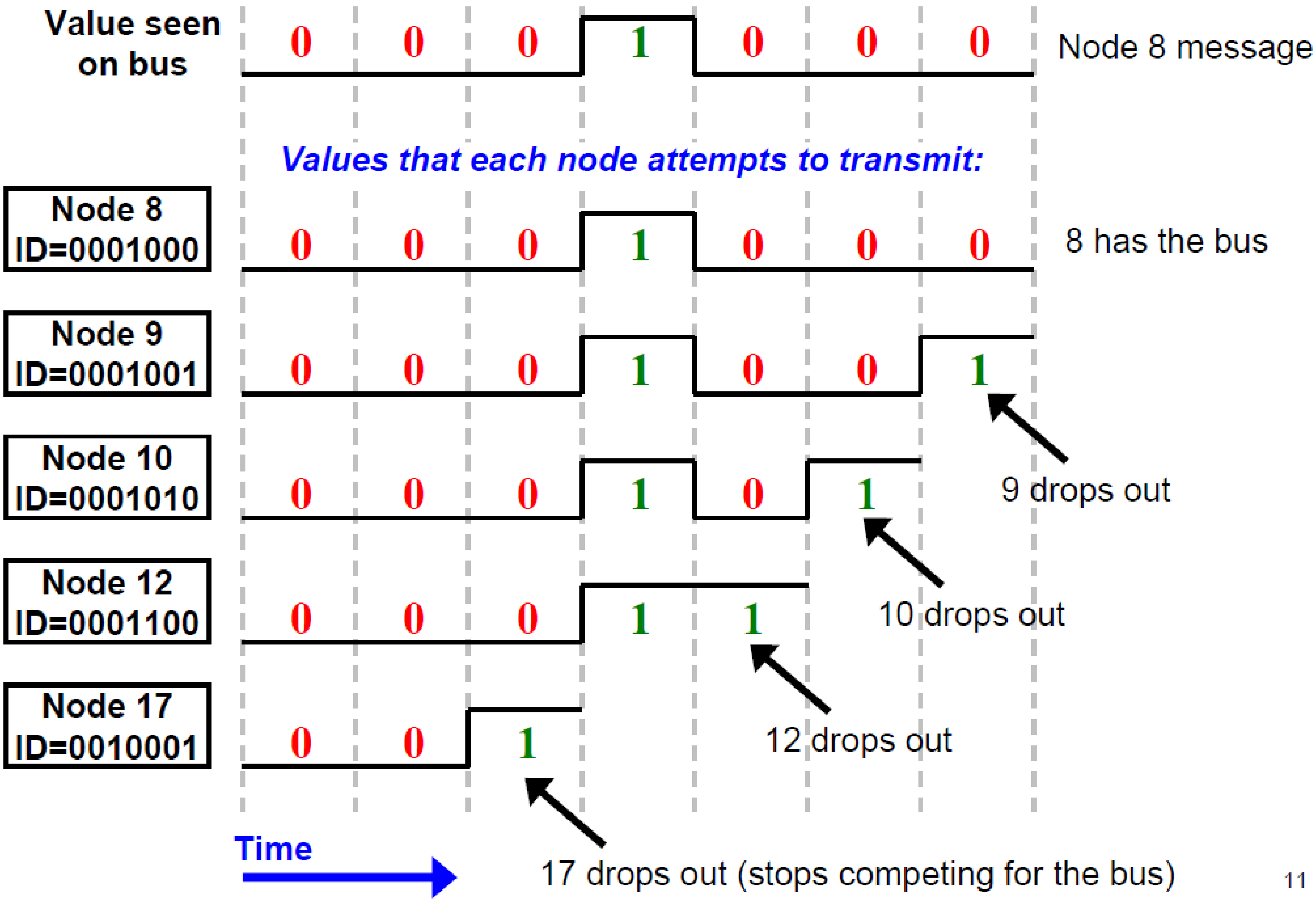
Example CAN Sample Signaling



CAN Logic & Arbitration

1. CAN 2.0A messages begin with an 11-bit message ID which identifies the message type and also establishes the message priority.
2. As with many computer interfaces, the CAN transceivers invert the microcontroller signal. Thus, the **dominant** bus state occurs when a **logic "0"** is transmitted and the **recessive** state occurs when a **logic "1"** is transmitted.
3. CAN uses the message ID to perform bus access arbitration between nodes.
4. Each node waits for an idle bus state then begins to transmit its message ID.
5. Each node also listens to the bus to see if the bus state match its transmission.
6. If a node detects a dominant bus state while transmitting a recessive message ID bit (logic "1"), it drops out of the current arbitration round and will try again the next time the bus is idle

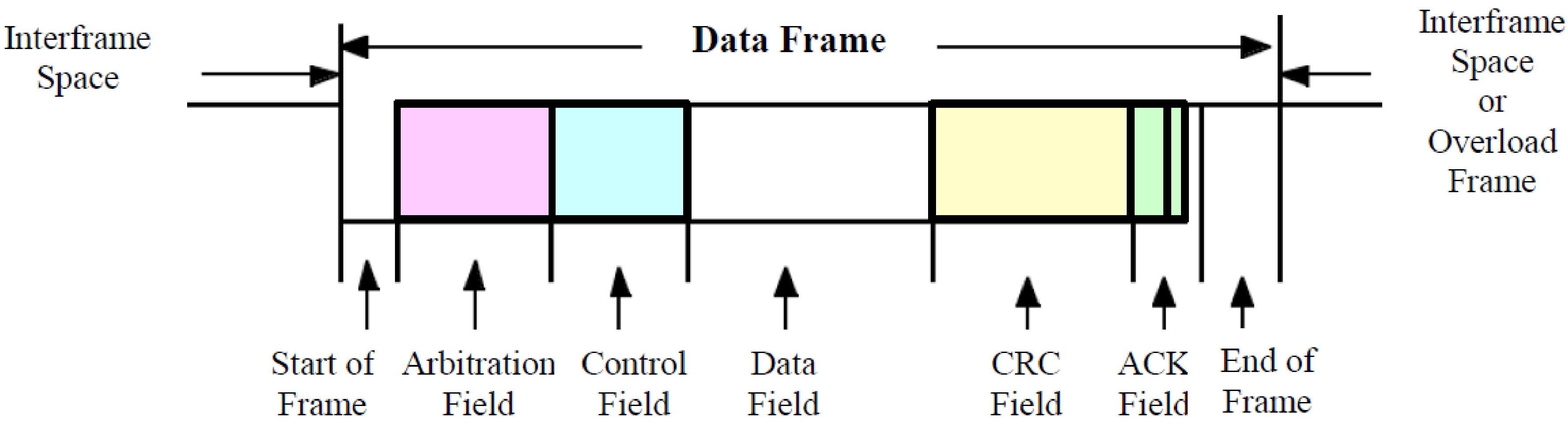
7-bit CANopen Node ID Arbitration Example



Key Advantages of CAN Bus Arbitration

1. Fast & deterministic.
2. Highest priority message gets immediate access once the bus is available.
3. Arbitration is essential “free” since message ID encodes message priority.
4. Unlike Carrier Sense Multiple Access with Collision Detect (CSMA/CD) arbitration propagation delays never cause message collisions.

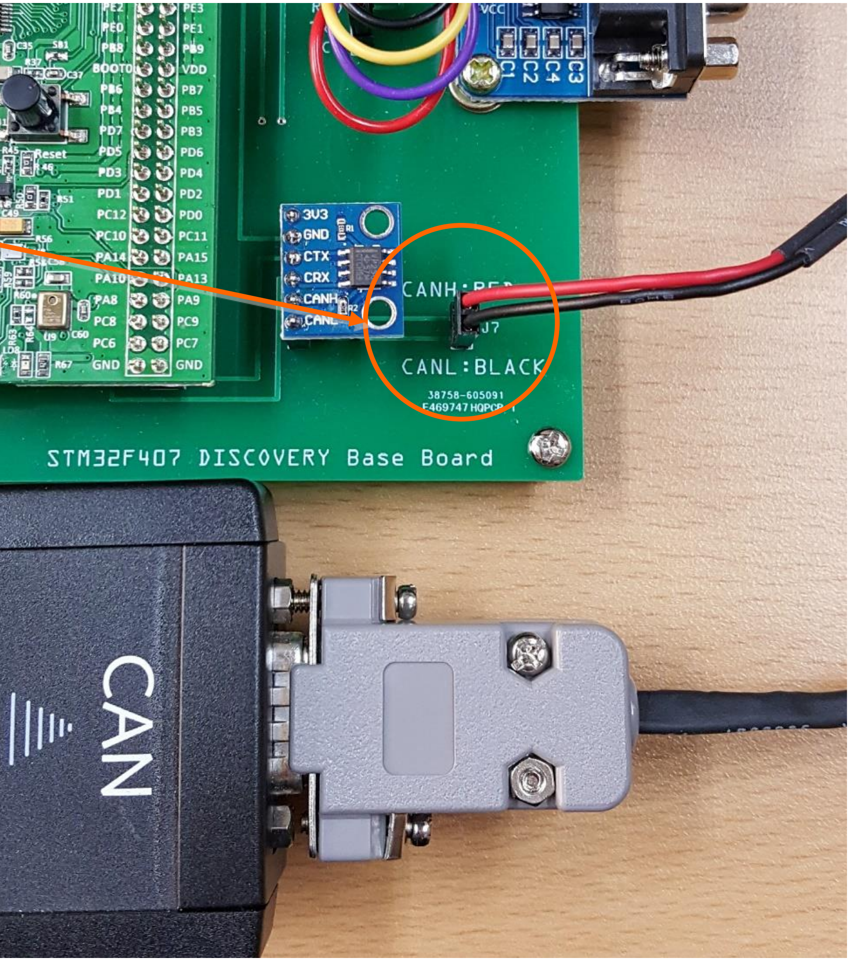
CAN Data Frame Format



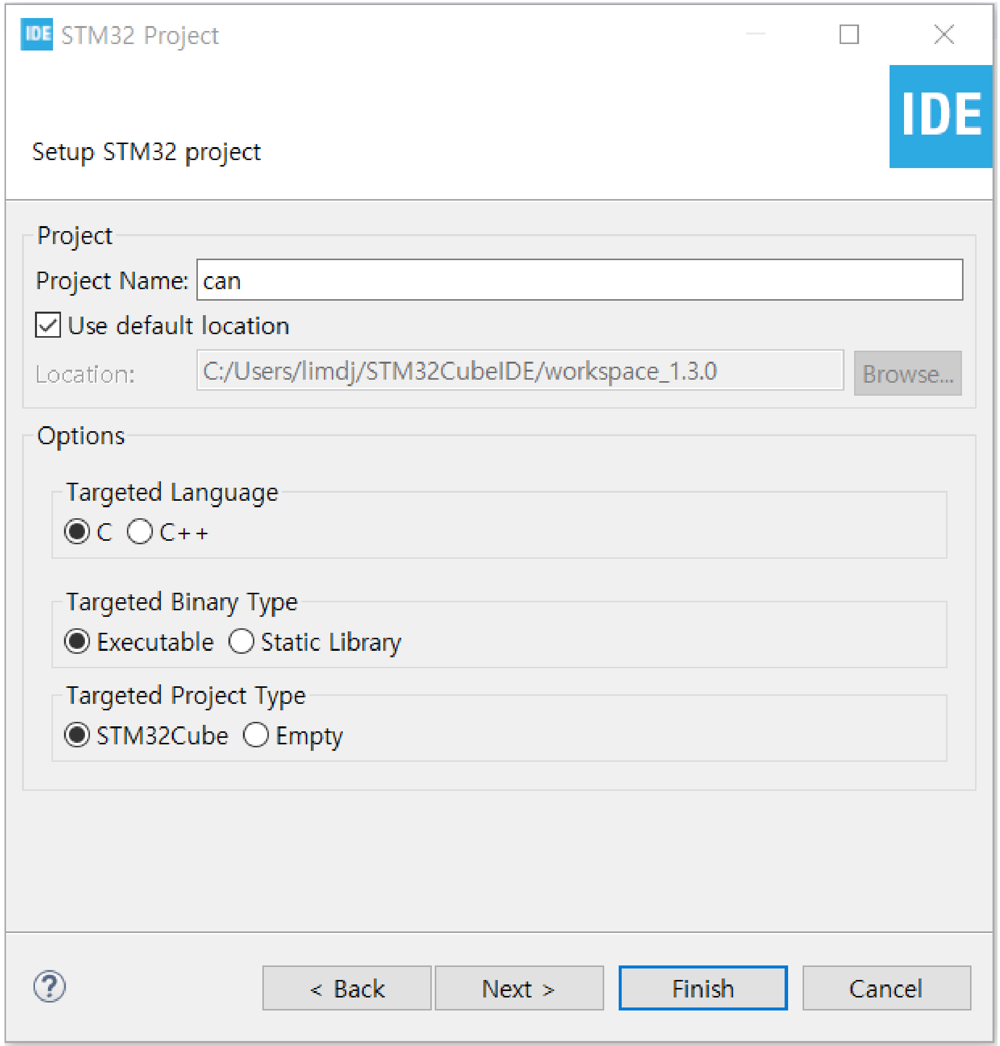
Base Field Name	Length (bits)	Purpose
Start of Frame	1	Denotes the start of frame transmission
Message Identifier / Arbitration Field	11	Message identifier also represents the message priority
Remote Transmission Request (RTR)	1	Dominant (0): Data is included in message Recessive (1): Remote Frame request for data
Identifier extension bit (IDE)	1	Must be dominant (0) for 11 bit message IDs
Reserved bit (r0)	1	Reserved bit should be dominant (0) for 11 bit IDs
Data length code (DLC)	4	Number of bytes of data (0–8 bytes)
Data Field	0–64	0 to 8 bytes of data (length dictated by DLC field)
CRC Filed	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) to acknowledge message
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

Before you begin

- Connect CAN cable to the CAN connector of the board
- Correct polarity

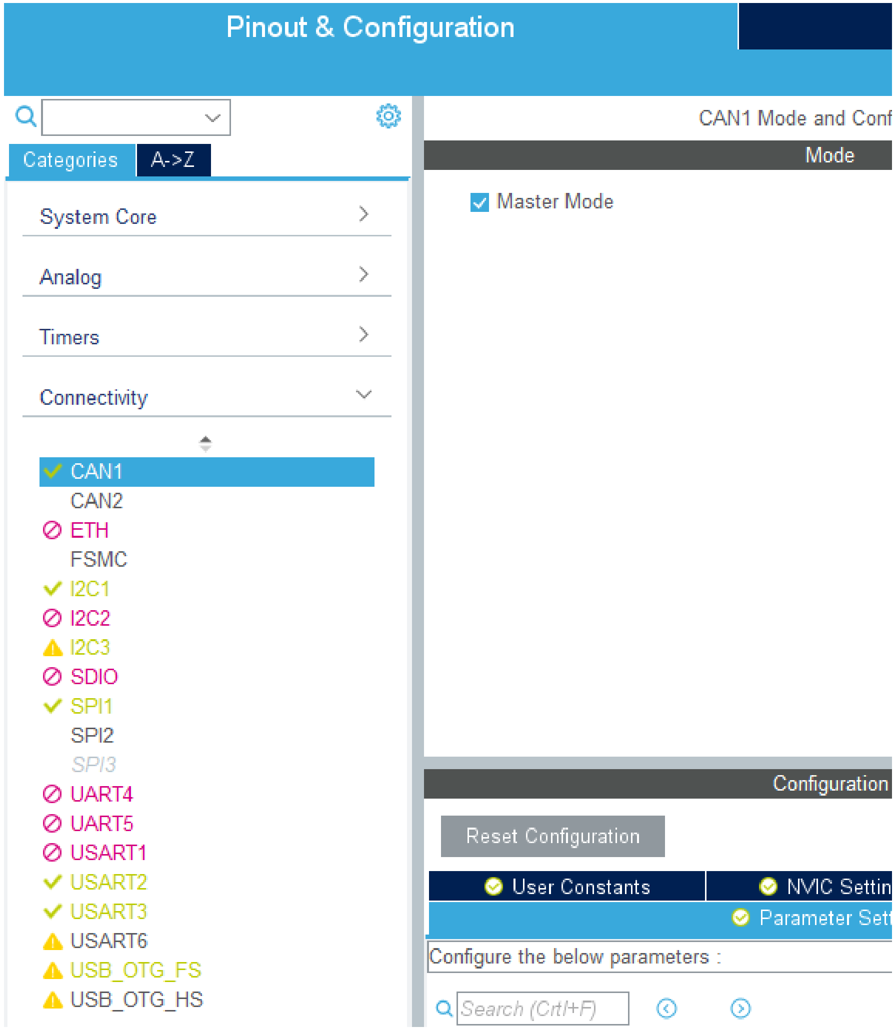


New STM32 project



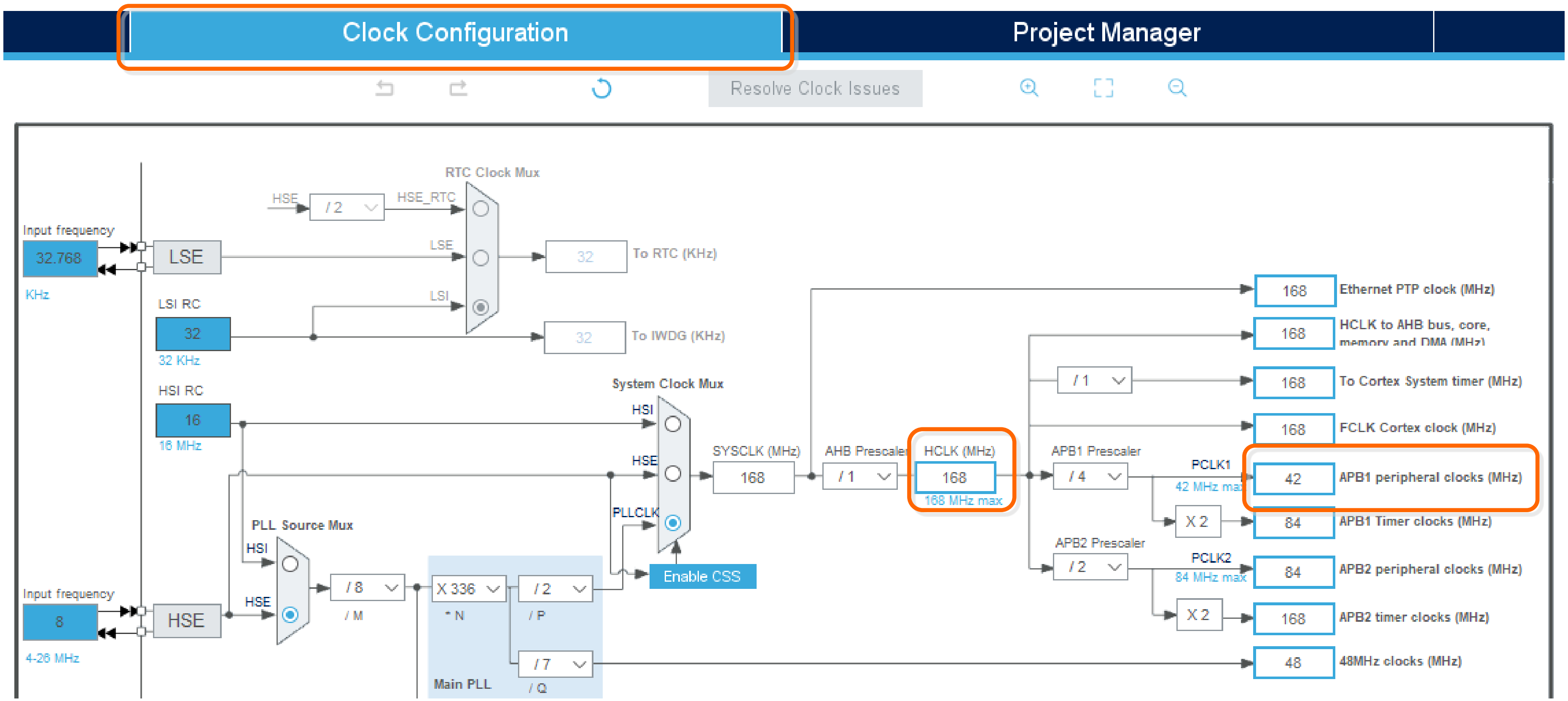
Pinout Selection

- CAN1: Master Mode
- I2C1: I2C
- I2C3: I2C
- USART2: Asynchronous
- USART3: Asynchronous



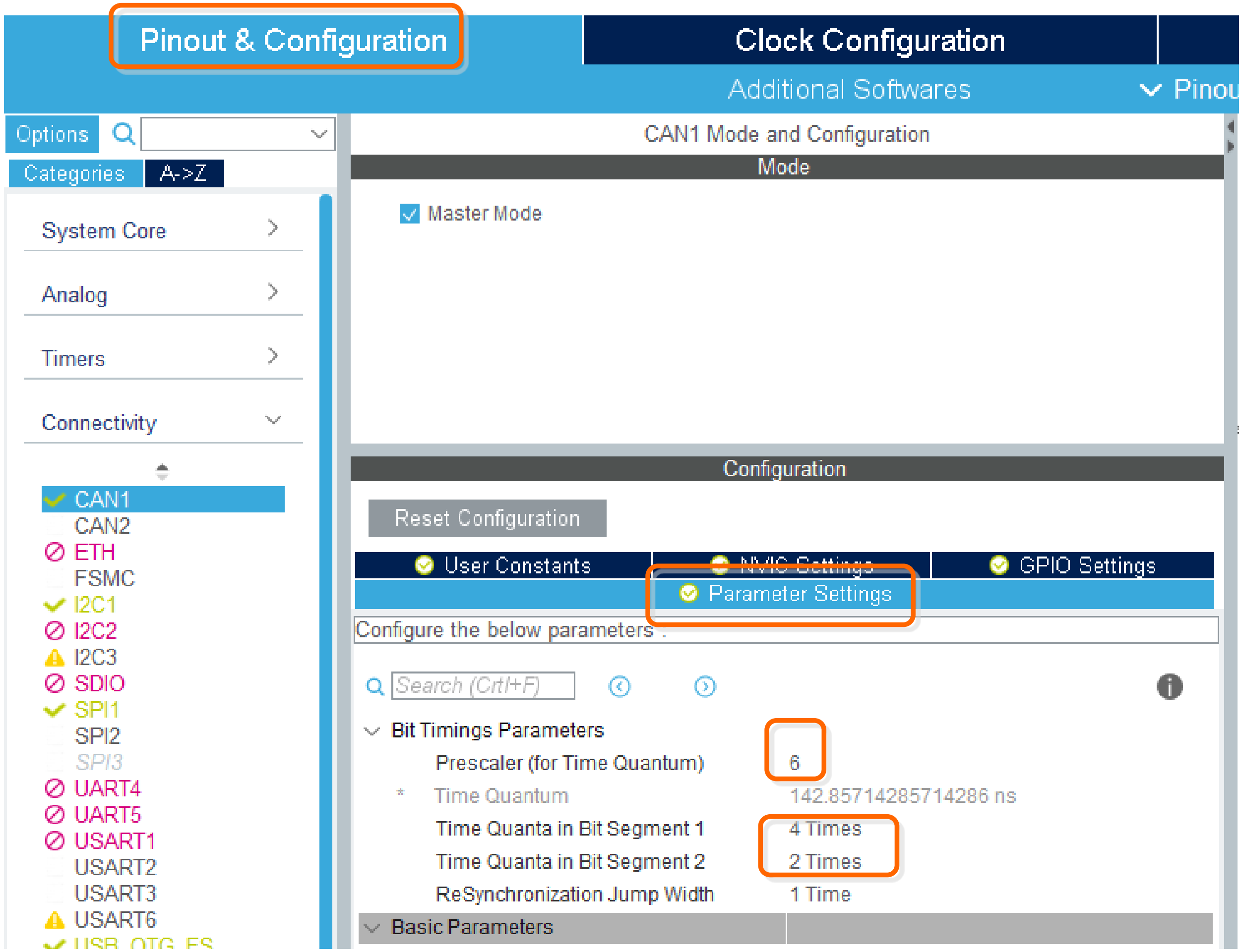
Clock Configuration

- Select Clock Configuration Tab
- Check 42MHz for APB1 peripheral clocks



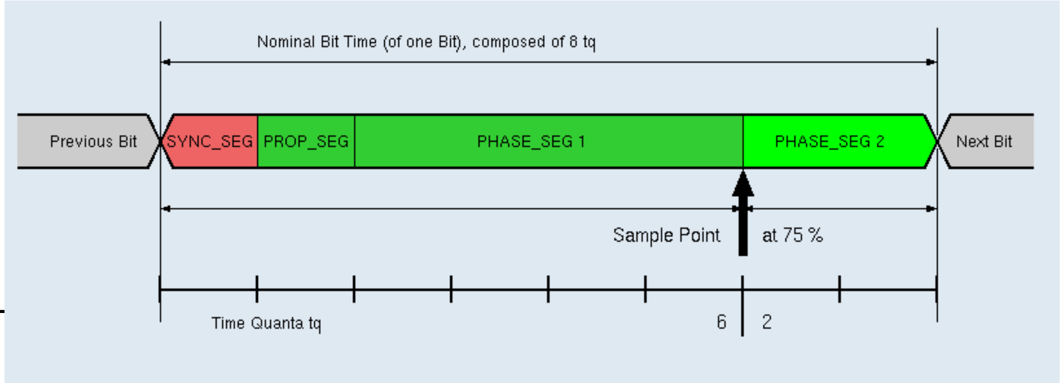
CAN Parameters

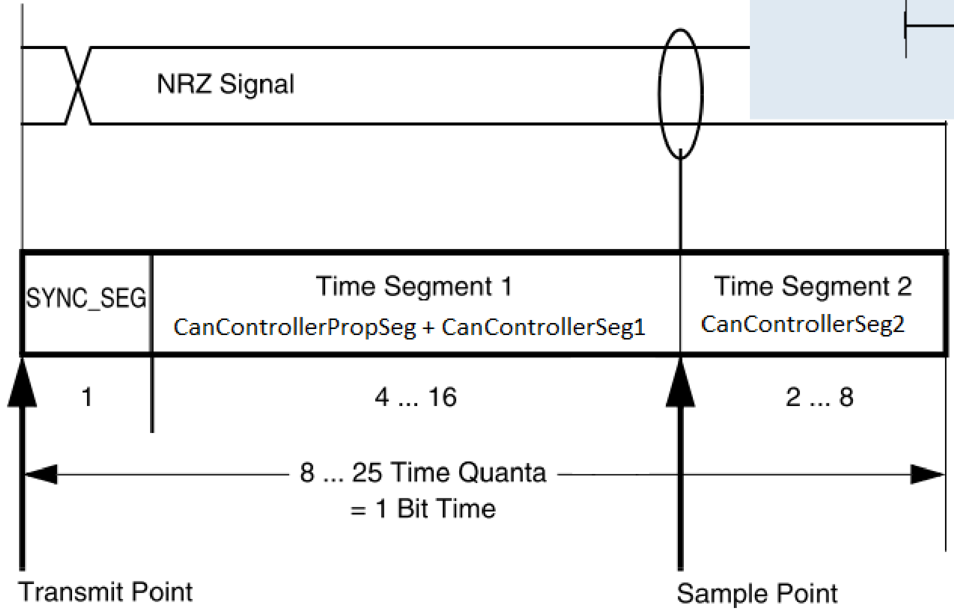
- Select Pinout & Configuration Tab
- Click CAN1 and select Parameter Settings
- Change Time Quanta to 4 Times and 2 Times
- Change Prescaler to 6



CAN bit timing

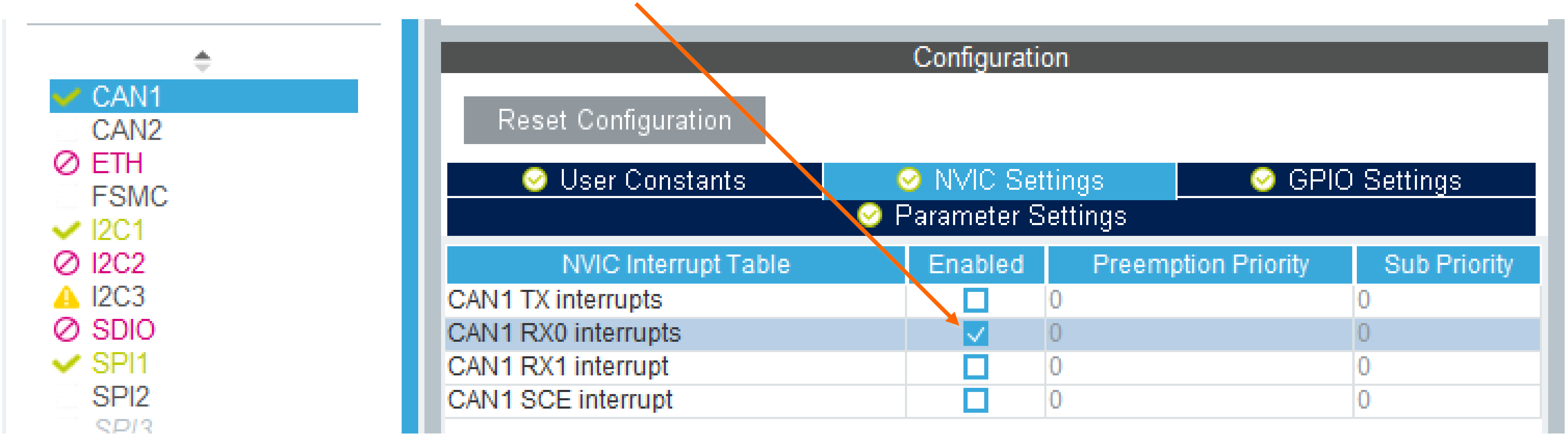
- 42Mhz/6=7MHz
- 1/7MHz=142.851743 nsec
- 1+4+2=7
- 2/7=0.29





CAN Interrupt Setting

- Select NVIC Settings
- Check CAN1 RX0 interrupts

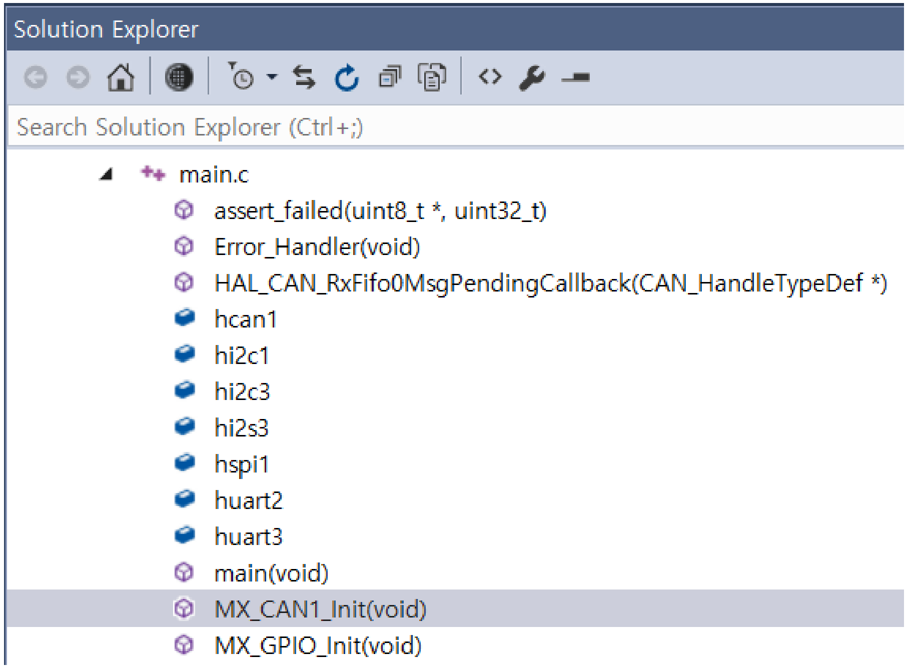


main.c(1)

```
/* USER CODE BEGIN PV */
CAN_HandleTypeDef hcan1;
CAN_TxHeaderTypeDef   TxHeader;
CAN_RxHeaderTypeDef   RxHeader;
uint8_t               TxData[8];
uint8_t               RxData[8];
uint32_t              TxMailbox;
/* USER CODE END PV */

 /* USER CODE BEGIN 2 */
        /* Test CAN data transmission */
        TxData[0] = 0x12;
        TxData[1] = 0x34;
        if (HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox) != HAL_OK)
        {
                /* Transmission request Error */
                Error_Handler();
        }
 /* USER CODE END 2 */
```

main.c(2)

```
static void MX_CAN1_Init(void)
{

  /* USER CODE BEGIN CAN1_Init 0 */
        CAN_FilterTypeDef  sFilterConfig;
  /* USER CODE END CAN1_Init 0 */
```



main.c(3)

```
/* USER CODE BEGIN CAN1_Init 2 */
 /*##-2- Configure the CAN Filter ###########################################*/
		sFilterConfig.FilterBank = 0;
		sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
		sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
		sFilterConfig.FilterIdHigh = 0x0000;
		sFilterConfig.FilterIdLow = 0x0000;
		sFilterConfig.FilterMaskIdHigh = 0x0000;
		sFilterConfig.FilterMaskIdLow = 0x0000;
		sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
		sFilterConfig.FilterActivation = ENABLE;
		sFilterConfig.SlaveStartFilterBank = 14;

		if (HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig) != HAL_OK)
		{
				/* Filter configuration Error */
				Error_Handler();
		}
```

main.c(4)

```
/*##-3- Start the CAN peripheral #############################################*/
if (HAL_CAN_Start(&hcan1) != HAL_OK)
{
	/* Start Error */
	Error_Handler();
}

/*##-4- Activate CAN RX notification #########################################*/
if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
{
	/* Notification Error */
	Error_Handler();
}
```

main.c(5)

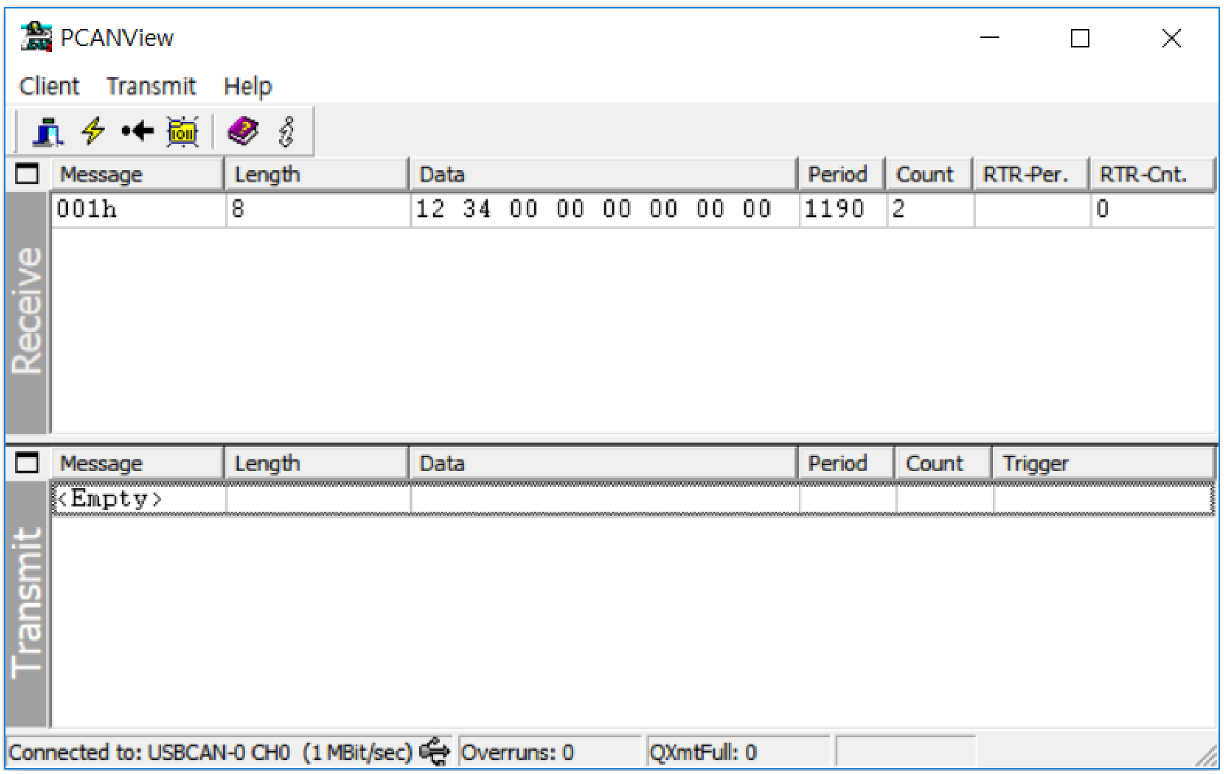
```
        /*##-5- Configure Transmission process #####################################*/
        TxHeader.StdId = 0x001;
        //TxHeader.ExtId = 0x01;
        TxHeader.RTR = CAN_RTR_DATA;
        TxHeader.IDE = CAN_ID_STD;
        TxHeader.DLC = 8;
        TxHeader.TransmitGlobalTime = DISABLE;
/* USER CODE END CAN1_Init 2 */
```

main.c(6)

```
/* USER CODE BEGIN 4 */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
        /* Get RX message */
        if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData) != HAL_OK)
        {
                /* Reception Error */
                Error_Handler();
        }
        TxData[0] = RxData[0];
        TxData[1] = RxData[1];
        if (HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox) != HAL_OK)
        {
                /* Transmission request Error */
                Error_Handler();
        }
}
/* USER CODE END 4 */
```

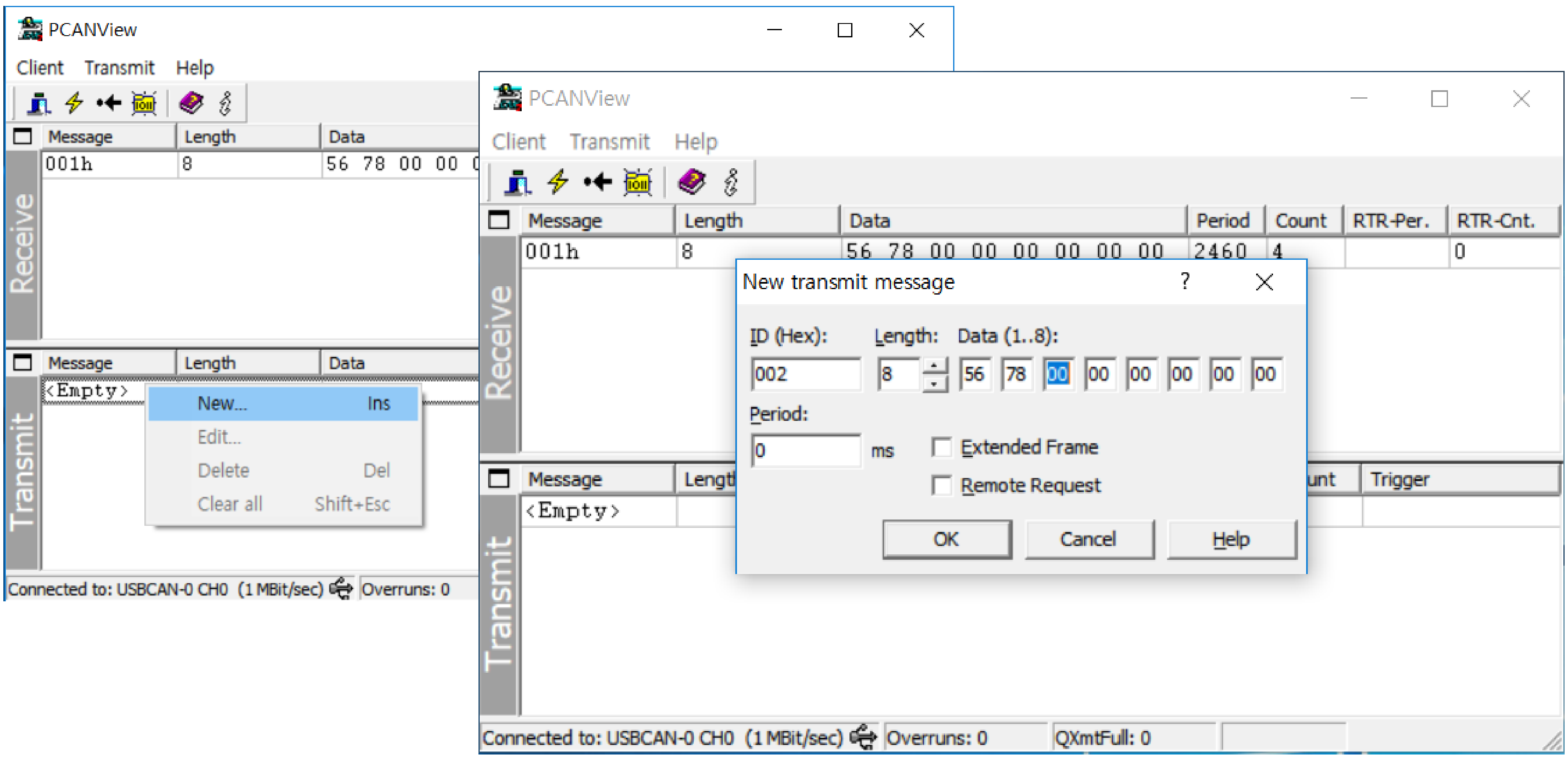
PCANView

- Get ready PCANView and run the program



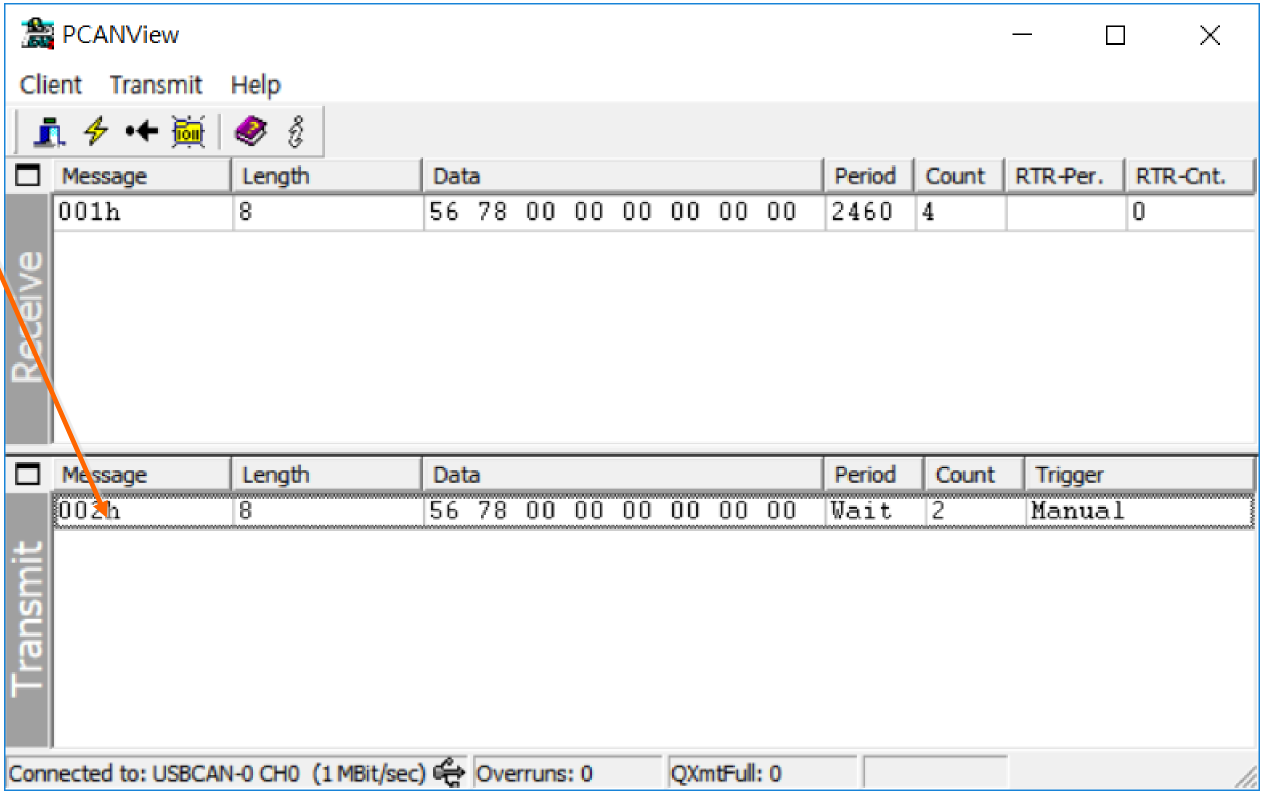
PCANView

- Right click to create a New CAN message
- Manual transmission: 0ms Period



PCANView

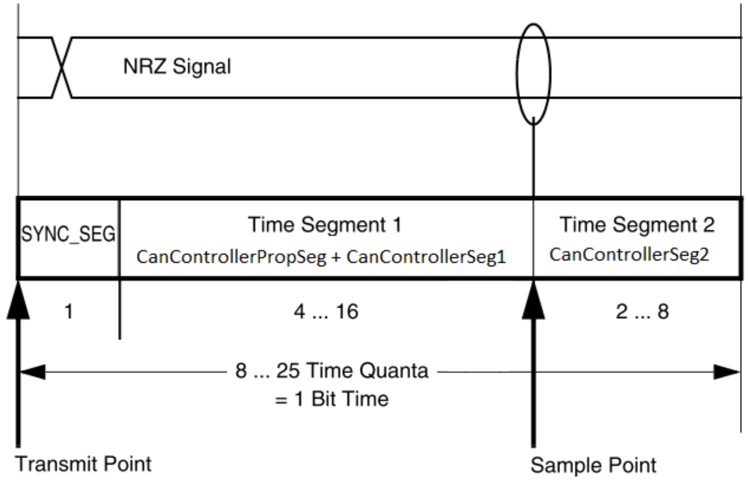
- Double click the message to transmit CAN data from PC to the board

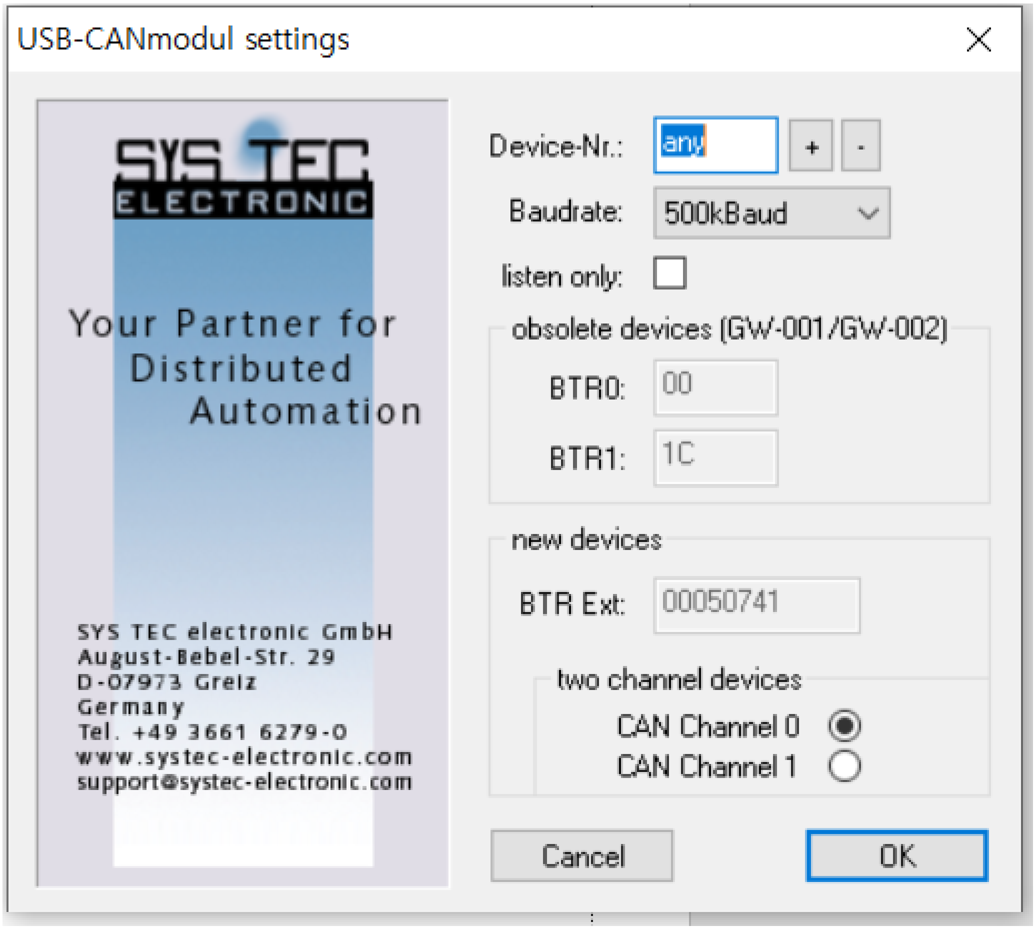


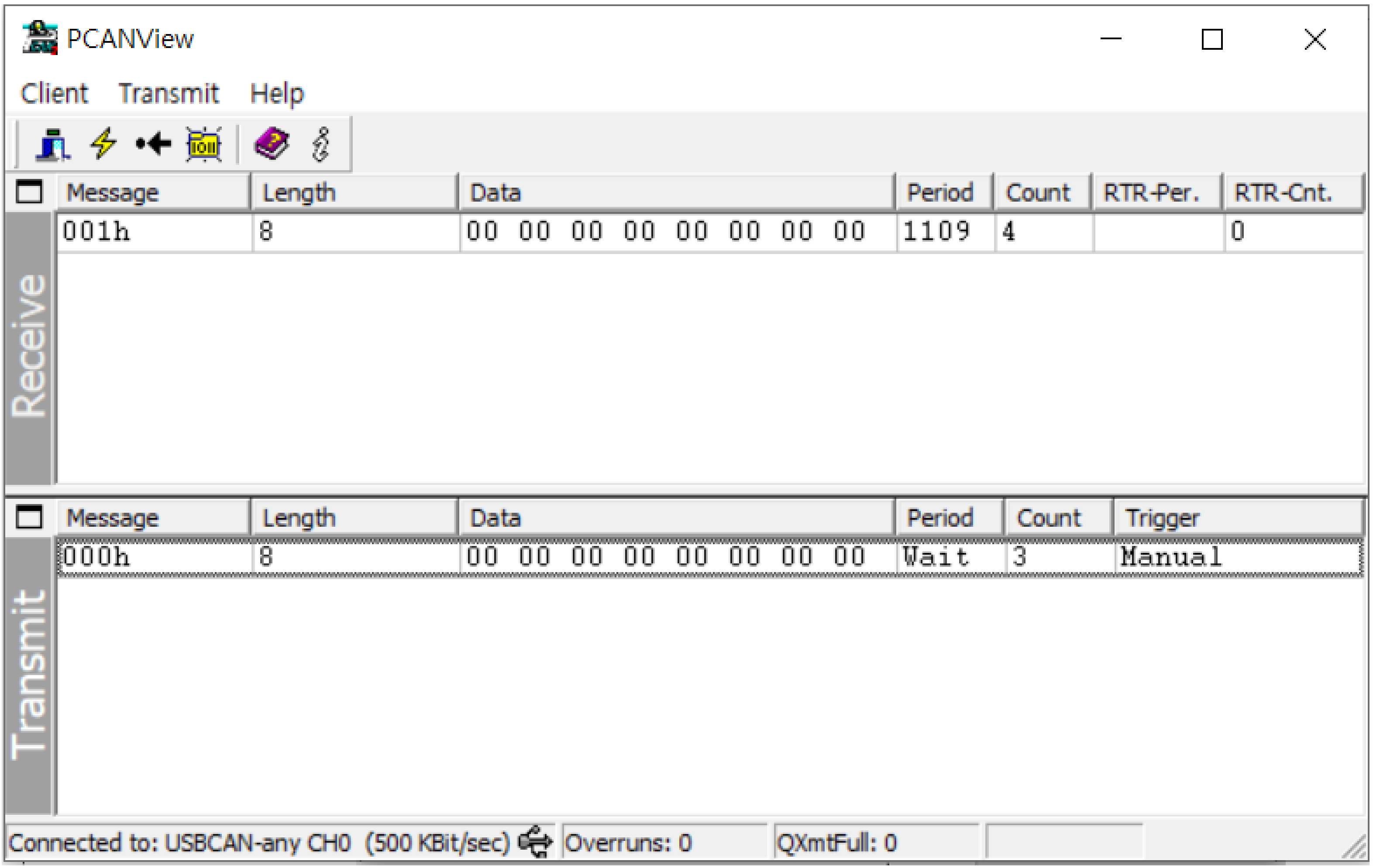
Change to 500 Kbit/sec

- 1+10+3=14
- 3/14=0.21

```
static void MX_CAN1_Init(void)
{

  /* USER CODE BEGIN CAN1_Init 0 */
    CAN_FilterTypeDef  sFilterConfig;
  /* USER CODE END CAN1_Init 0 */

  /* USER CODE BEGIN CAN1_Init 1 */

  /* USER CODE END CAN1_Init 1 */
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 6;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_10TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_3TQ;
```




USB-CANmodul settings
Your Partner for Distributed Automation
SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
Germany
Tel. +49 3661 6279-0
www.systec-electronic.com
support@systec-electronic.com
Device-Nr.: any
+
-
Baudrate: 500kBaud
listen only:
obsolete devices (GW-001/GW-002)
BTR0: 00
BTR1: 1C
new devices
BTR Ext: 00050741
two channel devices
CAN Channel 0
CAN Channel 1
Cancel
OK


PCANView
Client
Transmit
Help
Message
Length
Data
Period
Count
RTR-Per.
RTR-Cnt.
001h
8
00 00 00 00 00 00 00 00
1109
4
0
Receive
Message
Length
Data
Period
Count
Trigger
000h
8
00 00 00 00 00 00 00 00
Wait
3
Manual
Transmit
Connected to: USBCAN-any CH0 (500 KBit/sec)
Overruns: 0
QXmtFull: 0