SAMSUNG

CAN

Deep Dive into Baud Rate and Error Handling Model

Vivek Yadav

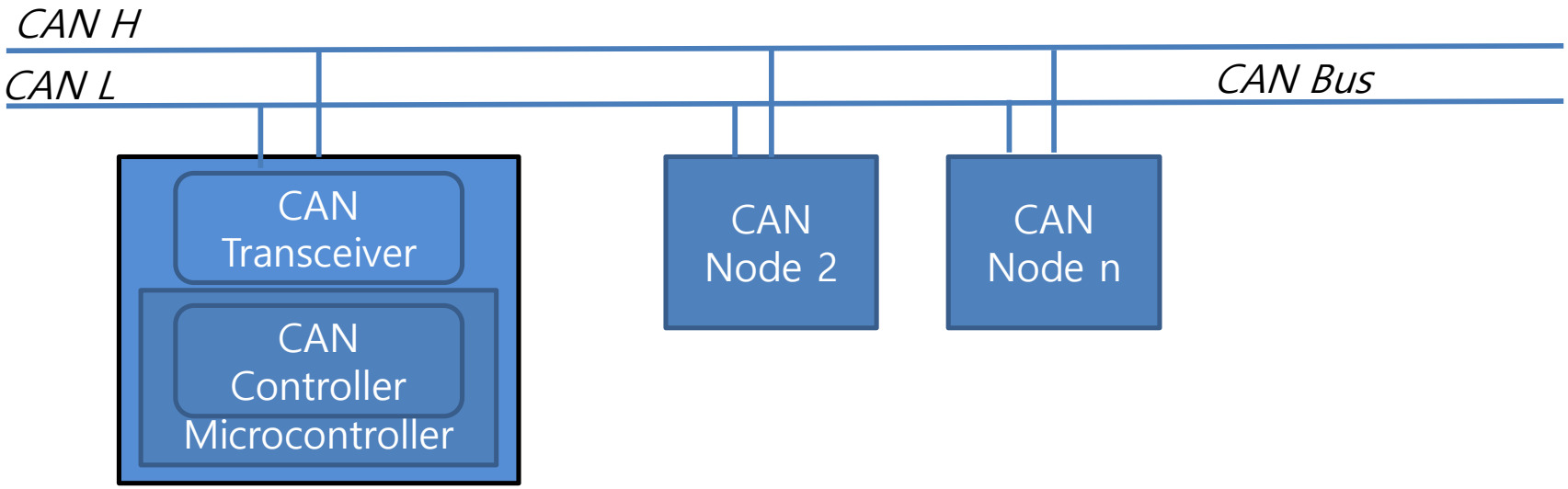
Samsung Semiconductor India R&D Center

Agenda

- What is CAN?
- Application of CAN
- CAN in Automotive and Aerospace Industry
- Linux CAN Subsystem
- User space tools
- Examples.

What is CAN?

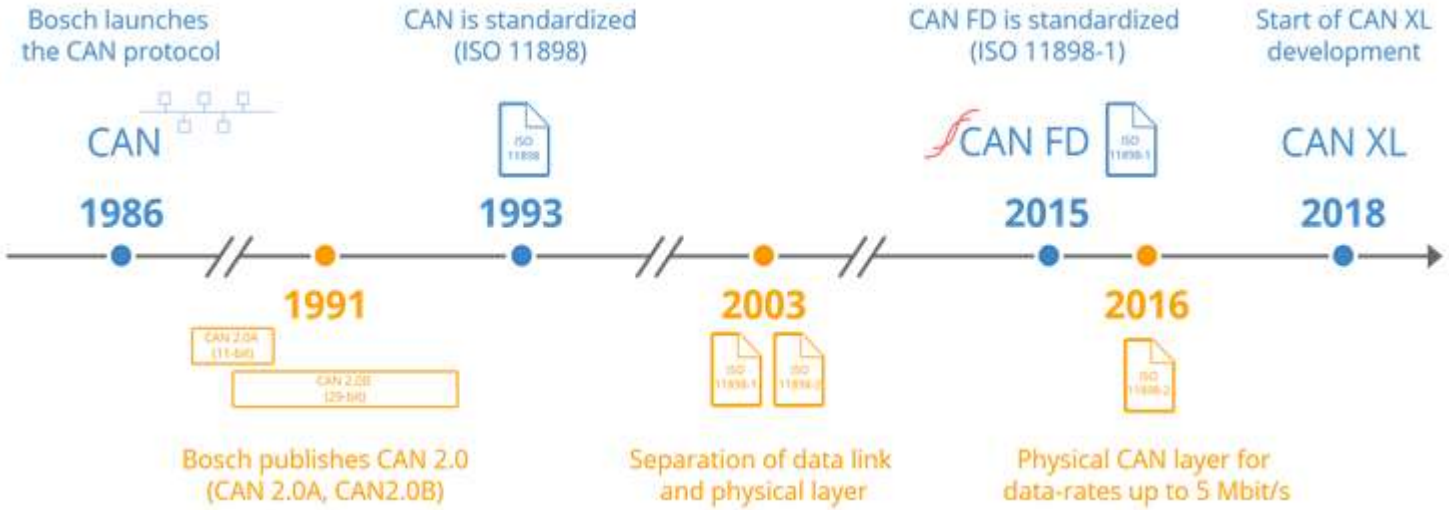
- CAN stands for Controller area network.
- The idea was initiated by Robert Bosch GmbH in 1983 and first released in 1986.
- CAN is called as multi-master serial and broadcast Bus.
- CAN is a message based protocol.
- CAN Provides message filtering so that each node act only on the interesting messages.
- Bus supports Non-Return To Zero (NRZ) with bit-stuffing.
- CAN standard defines four different message types.
- CAN Bus supports bit-wise arbitration to control access to the bus.



CAN Features

CAN Bus Details

- **CAN BUS Details:**
  - **ISO 11898-2**, called high-speed CAN, It is two-wire balanced signaling scheme.
  - **ISO 11898-3**, called low-speed CAN, It is fault tolerant, signal continued even bus wire is shorted or damaged.
  - CAN (ISO 11898-3) speeds up to 125 kbit/s and ISO 11898-2 speeds up to 1 Mbit/s on CAN and 5 Mbit/s on CAN-FD.
  - CAN bus is terminated using a resistor of 120 Ohms.

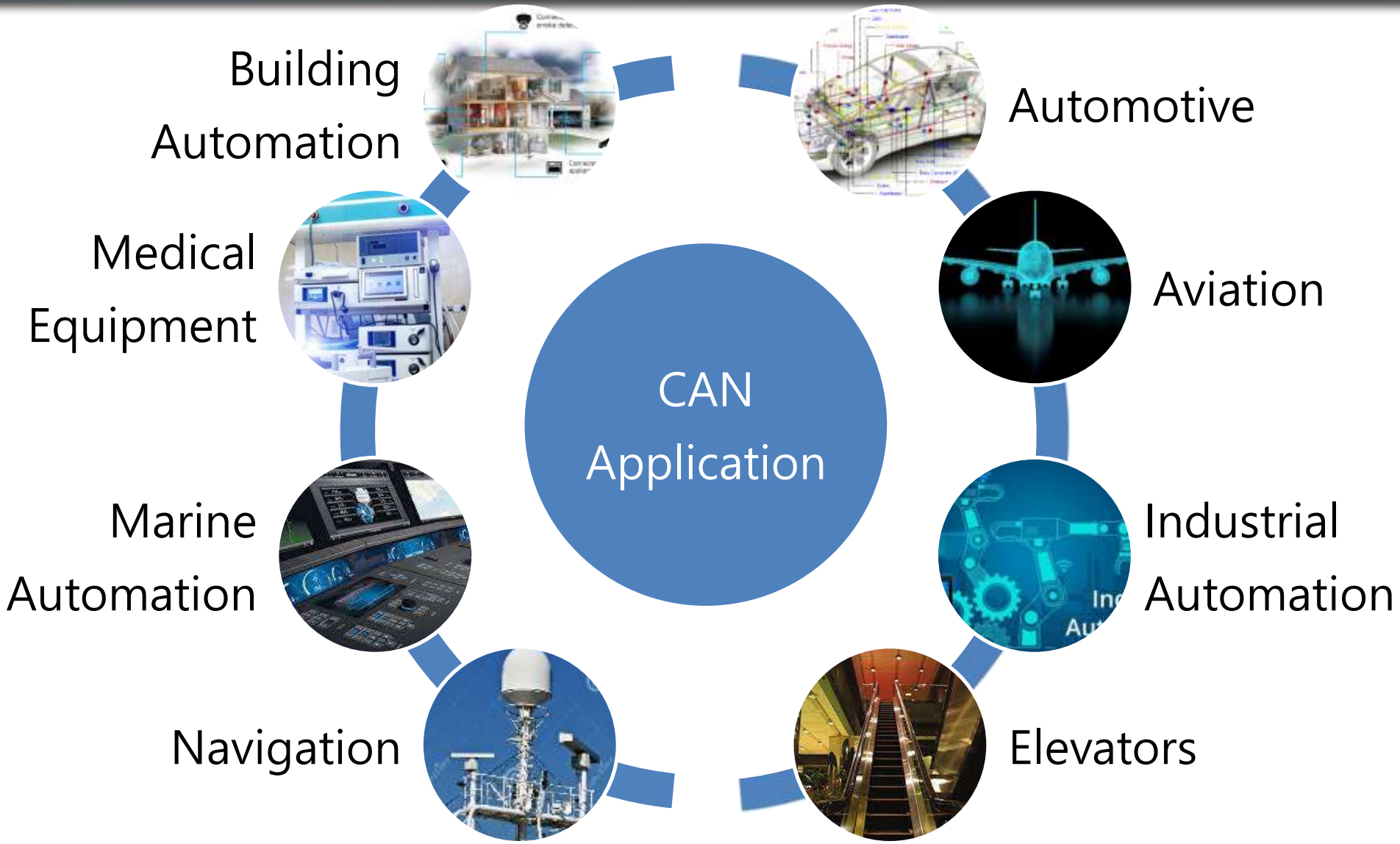


Source: https://cdn.shopify.com/s/files/1/0579/8032/1980/files/can-bus-history-timeline-controller-area-network.svg?v=1633690040

Speed
- 100 meters at 500 kbit/s
- 200 meters at 250 kbit/s

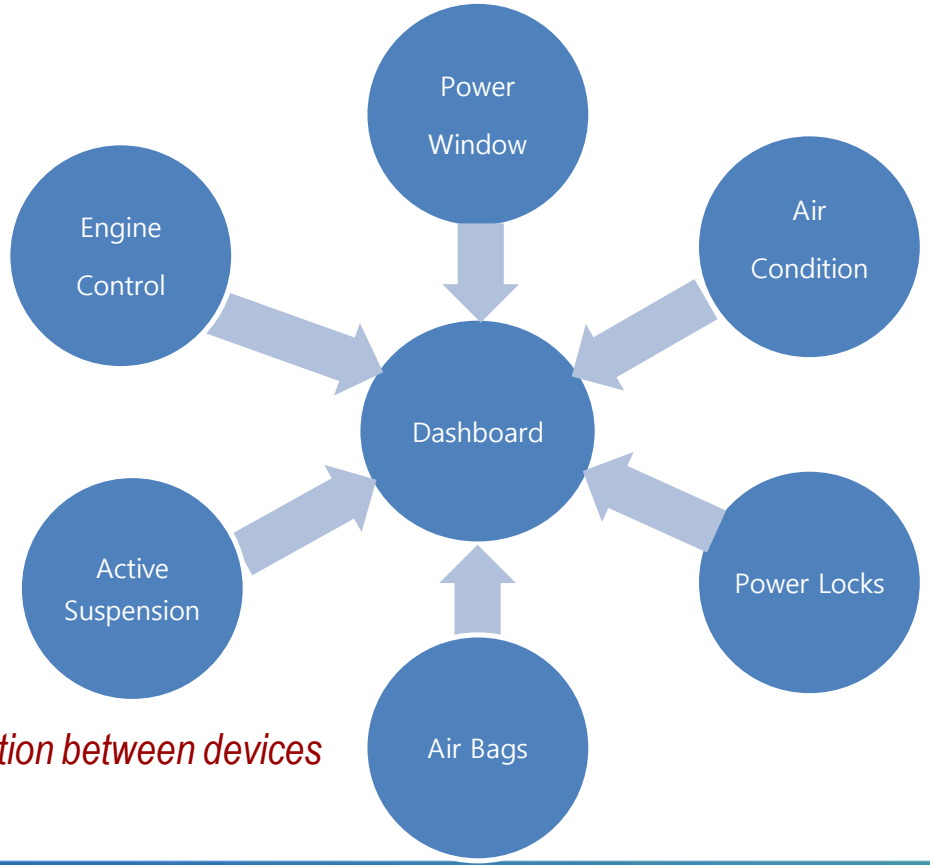
Length
- 500 meters at 125 kbit/s
- 6 kilometers at 10 kbit/s

Application of CAN



CAN in Automotive and Aerospace Industry

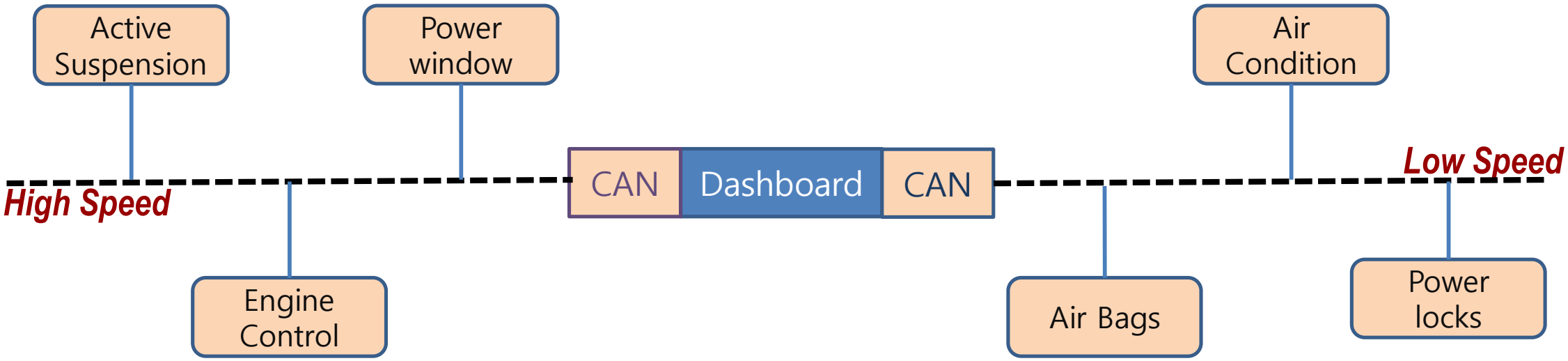
- Before CAN was introduced in Automotive Industry, each electronic device was connected to another via point-to-point wiring.
- Problem for automotive engineers was linking the ECUs of different devices so that real-time information could be exchanged. The CAN protocol was designed to address the above problem.



Point to Point connection between devices

CAN in Automotive and Aerospace Industry

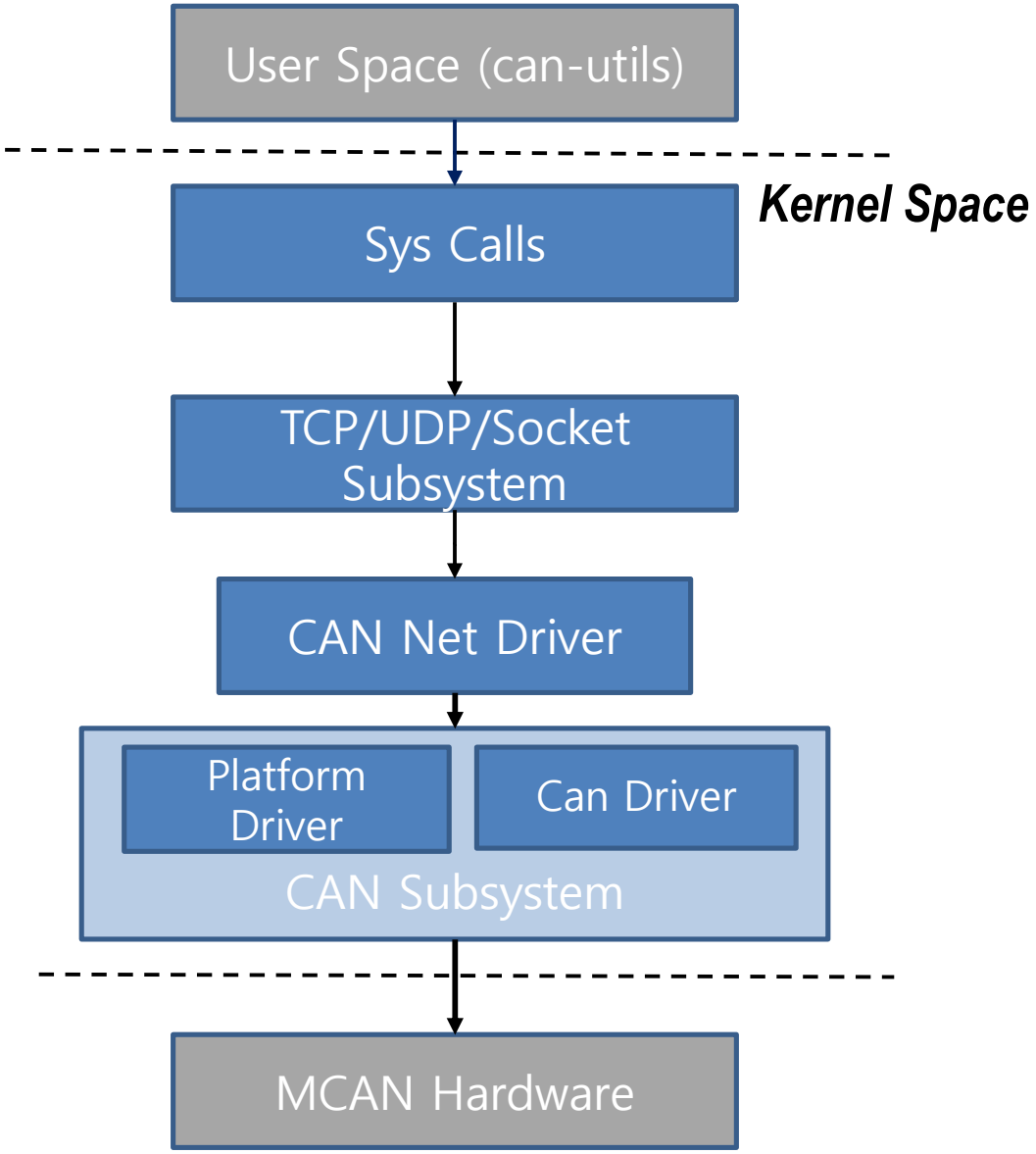
- The CAN protocol helps the electronic devices can exchange information with one another over a common serial bus. It reduced the overall complexity of the system.



Connectivity between devices using the CAN protocol.

Linux CAN Subsystem

- In Linux, CAN subsystem is designed in such a way that the system running Linux is always an CAN master.
- There will be an CAN platform driver in the kernel, which has routines to read and write onto CAN bus.
- The CAN Platform driver is the medium through which the kernel interacts with the device connected to the system.



Linux CAN Device Node

- The first step for writing a CAN platform driver is to fill the below structure

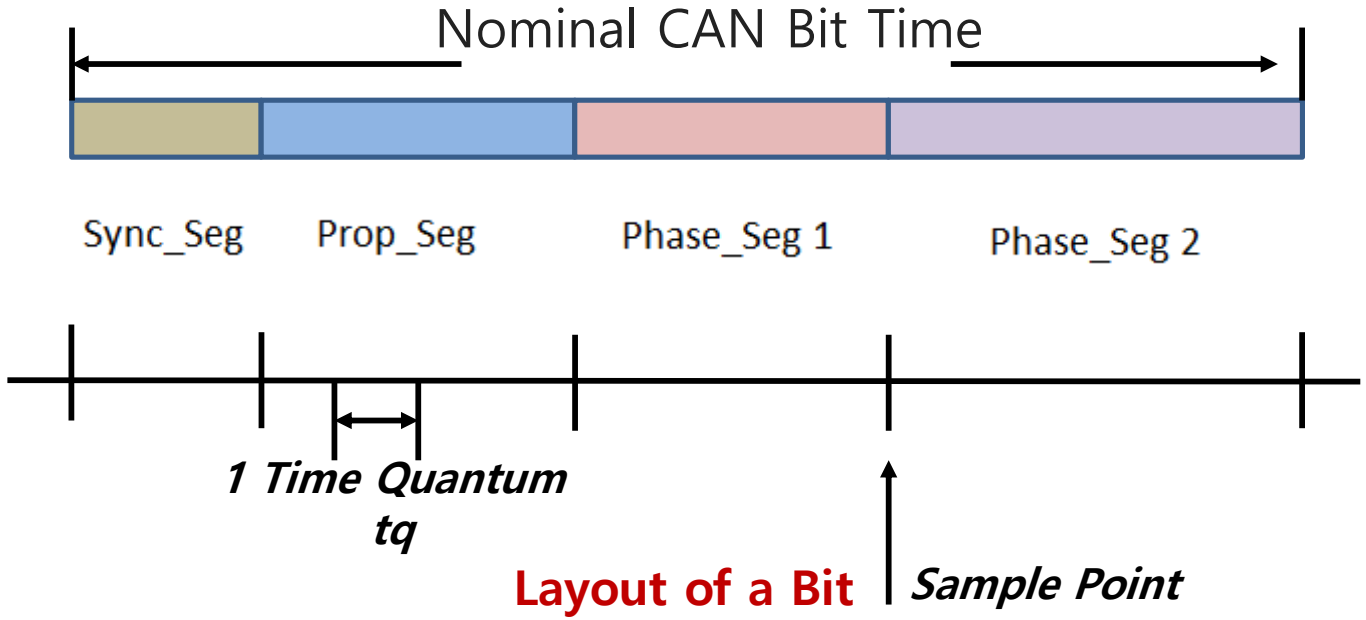
```
m_can0: can@20e8000  {
        compatible = "bosch,m_can";
        reg = <0x020e8000 0x4000>, <0x02298000 0x4000>;
        reg-names = "m_can", "message_ram";
        interrupts = <GIC_SPI 159 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 160 IRQ_TYPE_LEVEL_HIGH>;
        interrupt-names = "int0", "int1";
        clocks = <&clks IMX6SX_CLK_CANFD>,
                 <&clks IMX6SX_CLK_CANFD>;
        clock-names = "hclk", "cclk";
        bosch, mram-cfg = <0x0 128 64 64 64 64 32 32>;
        can-transceiver {
                max-bitrate = <5000000>;
        };
};
```

```
static struct platform_driver
m_can_plat_driver = {
        .driver = {
                .name = KBUILD_MODNAME,
                .of_match_table =
m_can_of_table,
                .pm     = &m_can_pmops,
        },
        .probe = m_can_plat_probe,
        .remove = m_can_plat_remove,
};
```

Source: https://elixir.bootlin.com/linux/v5.16.20/source/Documentation/devicetree/bindings/net/can/bosch,m_can.yaml

CAN Bit Timing

- CAN Bit Timing: Configure the bit segments to achieve the desired baud rate.
- The Nominal bit is logically divided into four groups or segments.
  - Synchronization Segment
  - Propagation Segment
  - Phase Segment 1
  - Phase Segment 2



- Define Layout of a Bit:
  - Baud Rate = 1/Nominal Bit Time
  - Nominal Bit Time = [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] * tq.
  - Tq (time quanta) = (BRP + 1) * (1/PCLK)
  - Total number of time quanta = Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2

Bit Timing Register Configuration

- **Clock Synchronization:** The number of time quanta adjustments required to achieve on chip clock synchronization are termed as the Synchronization Jump Width, SJW
  - Hard synchronization
  - Resynchronization
- Bit Timing Register Calculation:
  - clock Pre scaler value(BRP)
  - Number of quanta before the sampling point (Pseg-1)
  - Number of quanta after the sampling point (Pseg-2)
  - Number of quanta in the Synchronization Jump Width (SJW)

Example: Baudrate == 500k and PCLK : 42 Mhz

Number of time quanta's = 14

Pseg-1 = Prop_Seg + Phase_Seg1 = 11

Pseg-2 = Phase_Seg2 = 2

BRP = 6

tq (time quanta) = (BRP + 1) * 1/PCLK

$$\text{Baud Rate} = \frac{42\text{MHz}}{(\text{BRP}) * \text{total time quanta}} = \frac{42\text{Mhz}}{6*14} = 500{,}000$$

Baud Rate = 500 k

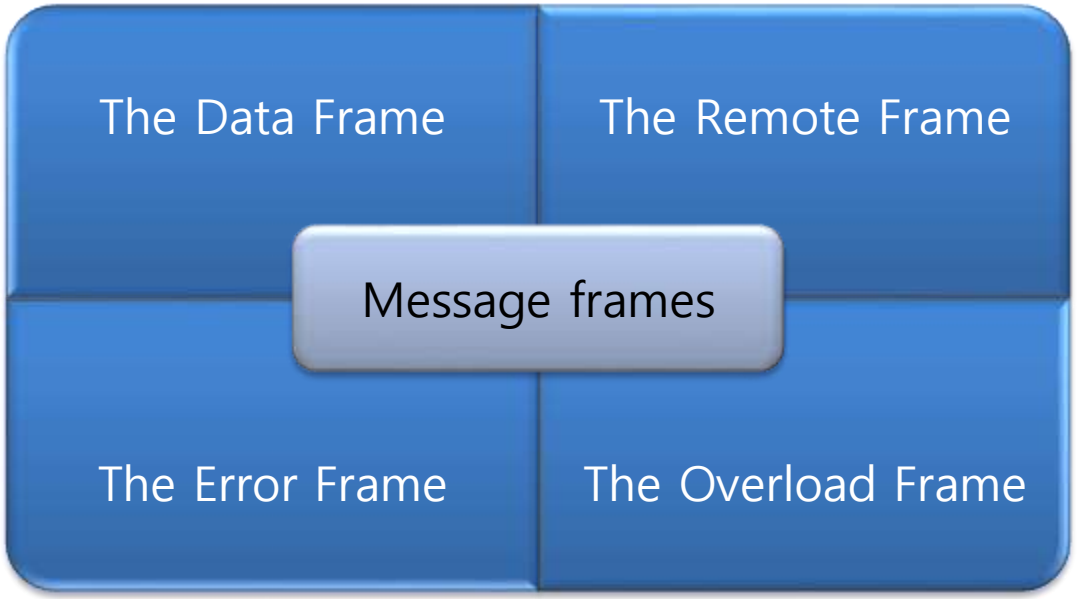
Linux CAN Bit Timing Structure

- **Linux CAN Bit Timing Structure Example:**

```
struct can_bittiming {
        __u32 bitrate;          /* Bit-rate in bits/second */
        __u32 sample_point;     /* Sample point in one-tenth of a percent */
        __u32 tq;               /* Time quanta (TQ) in nanoseconds */
        __u32 prop_seg;         /* Propagation segment in TQs */
        __u32 phase_seg1;       /* Phase buffer segment 1 in TQs */
        __u32 phase_seg2;       /* Phase buffer segment 2 in TQs */
        __u32 sjw;              /* Synchronisation jump width in TQs */
        __u32 brp;              /* Bit-rate prescaler */
};
```

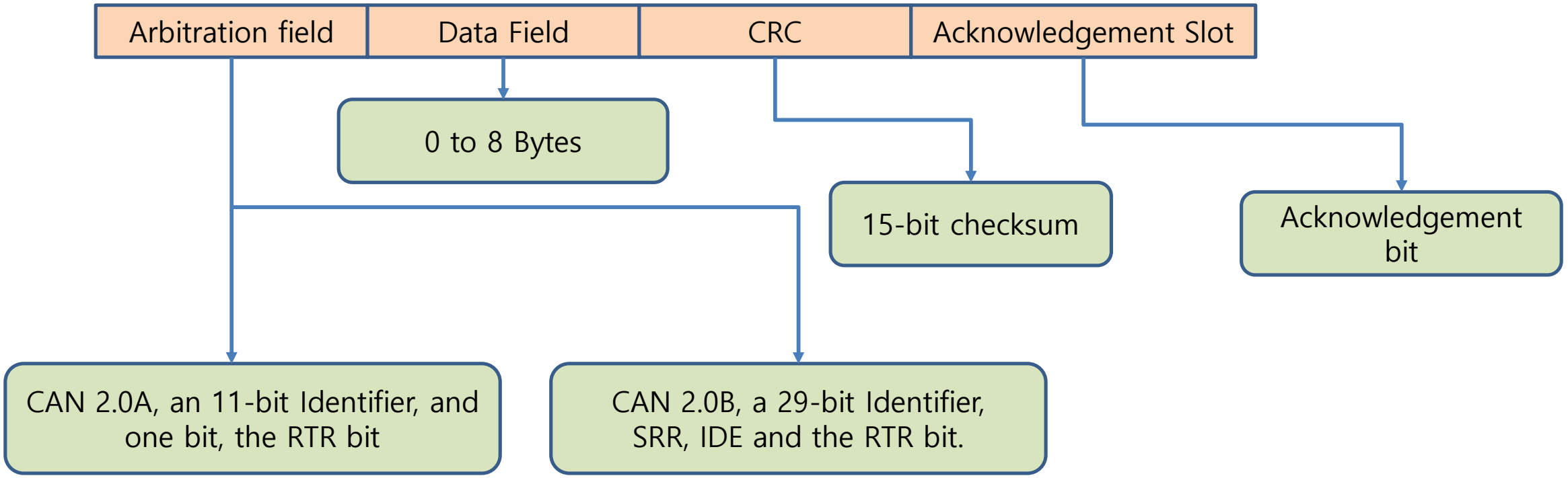
CAN Message Frame

- CAN Messages
  - CAN bus is a broadcast type of bus, means all nodes can ‘hear’ all transmissions.
  - Nodes will pick up all CAN Bus traffic and the CAN hardware provides local filtering onto the interesting messages.
  - The frame said to be Identifier-addressed and there is no explicit address includes in the messages.
  - There are four different types of message frames on a CAN bus.

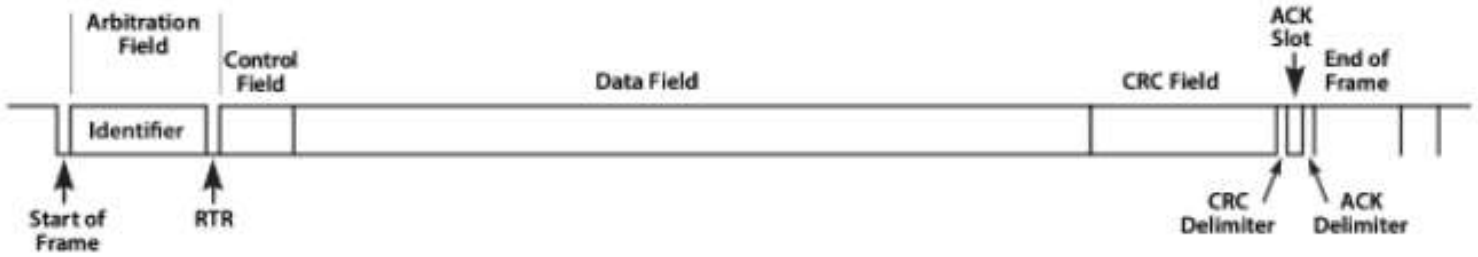


CAN Message Frame

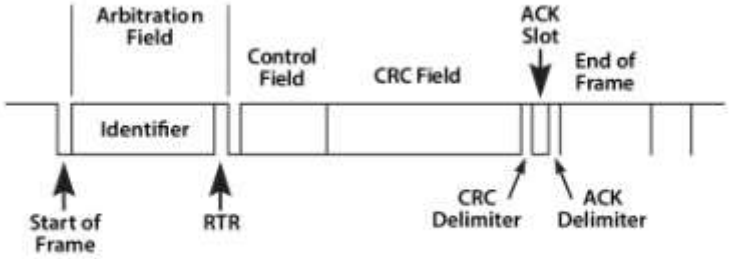
- CAN Frame:
  - The CAN Data Frame comprises into the following major parts:



CAN Message Frame

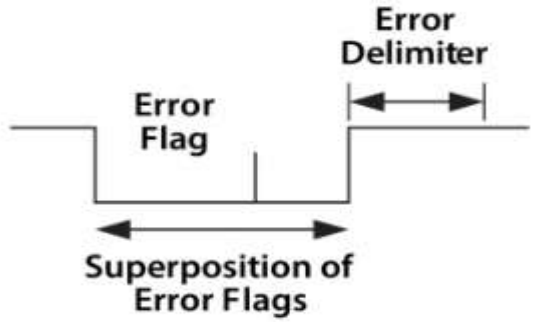


A CAN 2.0A ("standard CAN") Data Frame.



A Remote Frame (2.0A type)

A CAN 2.0B ("extended CAN") Data Frame.



An Error Frame

Source: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

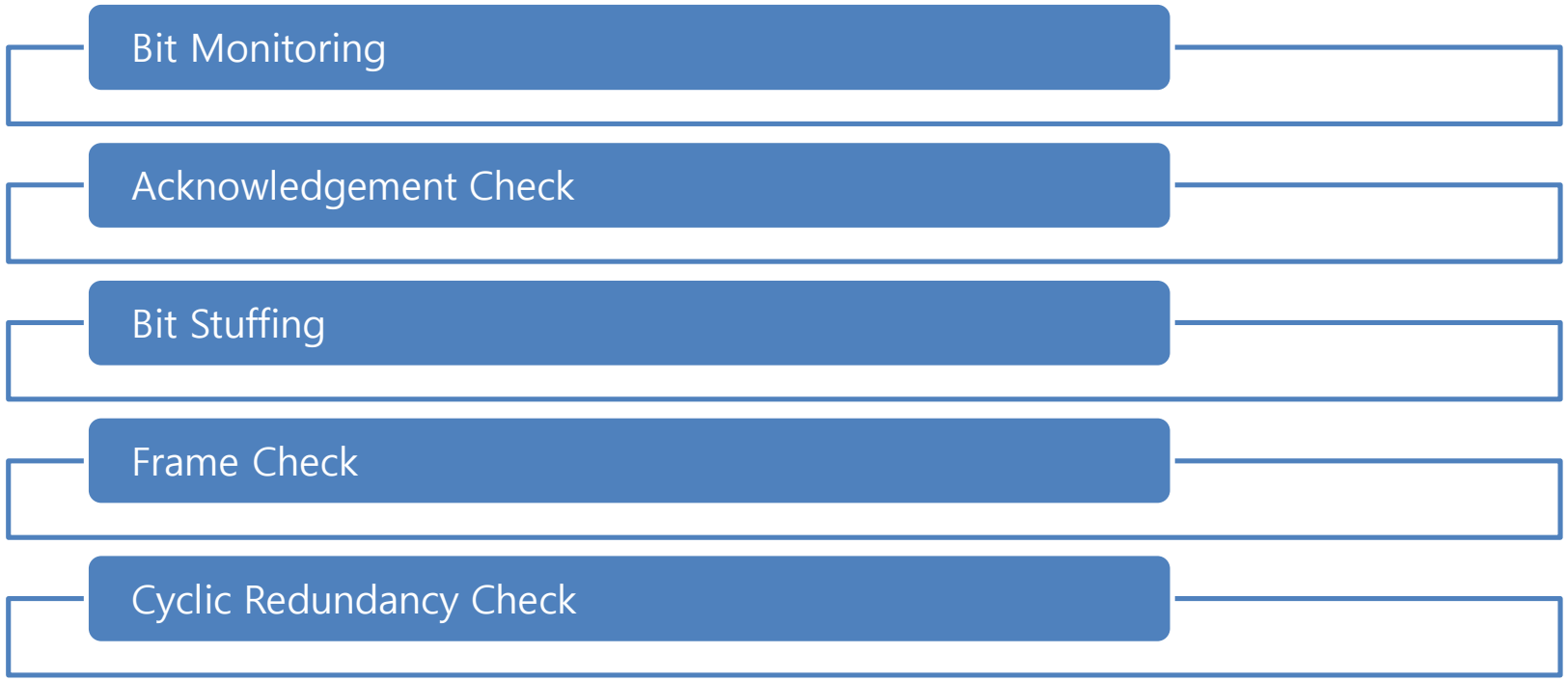
Linux CAN Frame Structure


```
struct can_frame {
        canid_t can_id;             /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        union {
                __u8 len;
                __u8 can_dlc;       /* deprecated */
        } __attribute__((packed));  /* disable padding added in some ABIs */
        __u8 __pad;                 /* padding */
        __u8 __res0;                /* reserved / padding */
        __u8 len8_dlc;              /* optional DLC for 8 byte payload length (9 .. 15)
*/
        __u8 data[CAN_MAX_DLEN] __attribute__((aligned(8)));
};
```

```
 struct canfd_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    len;     /* frame payload length in byte */
        __u8    flags;   /* additional flags for CAN FD */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[CANFD_MAX_DLEN] __attribute__((aligned(8)));
 };
```

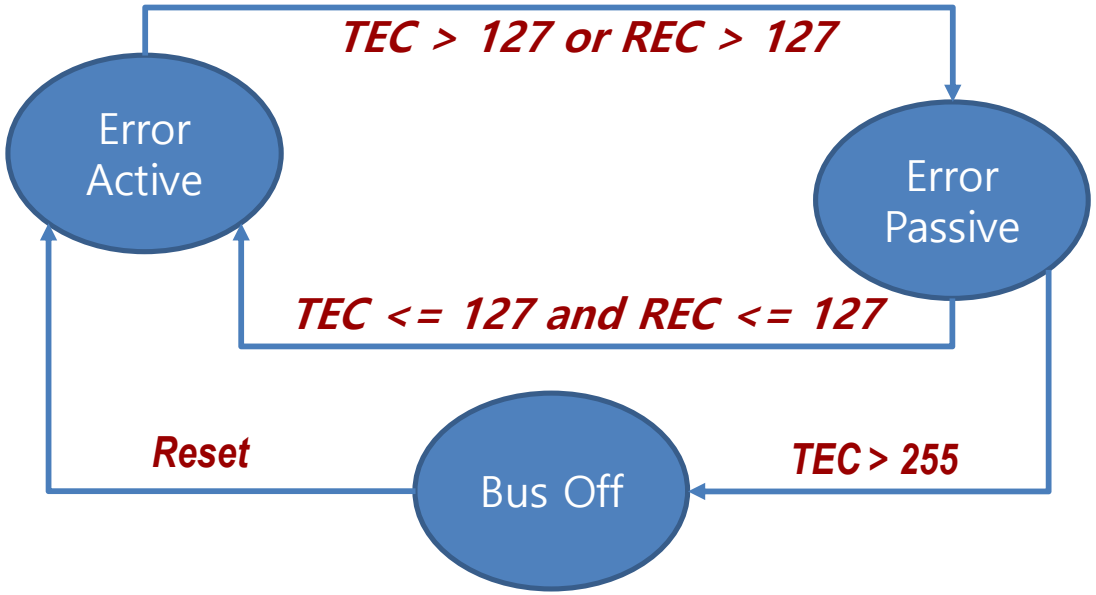
CAN Error Framework

- **CAN Error:** Error handling is built into the CAN protocol is a great impact on the performance of CAN.
  - Every CAN Nodes along a bus detect errors within a message.
  - If an error is found, the discovering node will transmit an Error frame with Error Flag enabled.
  - Nodes detects the error and take appropriate action, i.e. discard the current message.
- The CAN protocol defines five different ways of detecting errors



Error Confinement Mechanisms

- A CAN node start its error state in Error Active mode. When the Error Counters raises above 127, the node will enter into Error Passive and when the Transmit Error Counter raises above 255, the node will enter the Bus Off state.
  - When an Error Passive node detects errors will transmit Passive error Flags.
  - When an Error Active node detects errors will transmit Active error Flags.
  - A node which enters Bus Off will not Participate in any of the Communication.
- Most CAN controllers will provide status bits for two states
  - **BO:** Bus Off Status
    - *0* = The M_CAN is not Bus Off
    - *1* = The M_CAN is in Bus Off state
  - **EW:** Warning Status
    - *0* = Both error counters are below the Error Warning limit of 96
    - *1* = At least one of error counter has reached the Error Warning limi t of 96



Linux CAN Error Structures

- Linux CAN error state Structure Examples:

```
switch (lec_type) {
case LEC_STUFF_ERROR:
        netdev_dbg(dev, "stuff error\n");
        cf->data[2] |= CAN_ERR_PROT_STUFF;
        break;
case LEC_FORM_ERROR:
        netdev_dbg(dev, "form error\n");
        cf->data[2] |= CAN_ERR_PROT_FORM;
        break;
case LEC_ACK_ERROR:
        netdev_dbg(dev, "ack error\n");
        cf->data[3] = CAN_ERR_PROT_LOC_ACK;
        break;
case LEC_BIT1_ERROR:
        netdev_dbg(dev, "bit1 error\n");
        cf->data[2] |= CAN_ERR_PROT_BIT1;
        break;
case LEC_BIT0_ERROR:
        netdev_dbg(dev, "bit0 error\n");
        cf->data[2] |= CAN_ERR_PROT_BIT0;
        break;
case LEC_CRC_ERROR:
        netdev_dbg(dev, "CRC error\n");
        cf->data[3] = CAN_ERR_PROT_LOC_CRC_SEQ;
        break;
default:
        break;
}
```

Last error code occur on CAN BUS

```
struct can_device_stats {
        __u32 bus_error;        /* Bus errors */
        __u32 error_warning;    /* Changes to error warning state */
        __u32 error_passive;    /* Changes to error passive state */
        __u32 bus_off;          /* Changes to bus off state */
        __u32 arbitration_lost; /* Arbitration lost errors */
        __u32 restarts;         /* CAN controller re-starts */
};
```

CAN Bus State

CAN BUS Failures

- Bus Failure Modes:
  - CAN_H interrupted
  - CAN_L interrupted
  - CAN_H shorted to battery voltage
  - CAN_L shorted to ground
  - CAN_H shorted to ground
  - CAN_L shorted to battery voltage
  - CAN_L shorted to CAN_H wire
  - CAN_H and CAN_L interrupted at the same location
  - Loss of connection to termination network

User Space Tools

- **Linux can-utils** : *can-utils* is a command line utility that contains basic tools. CAN dump, can send, Display, record, generate and replay CAN traffic.
- To install can-utils in your working space, use the following command
  - sudo apt-get install can-utils –y
- Linux can-utils Commands:

  - ip link set canX type can help

  - ip link set canX type can bitrate 960000 loopback on
  - ip link set canX type can bitrate 1800000 dbitrate 3420000 fd on fd-non-iso on
  - ip link set canX up type can bitrate 500000 berr-reporting on one-shot on

  - ip link set canX up
  - candump canX&
  - cansend canX 16A#1122334455667788
  - cansend canX 1F334455#1122334455667788

  - candump canX
  - ip -details link show canX
  - ip -details -statistics link show canX

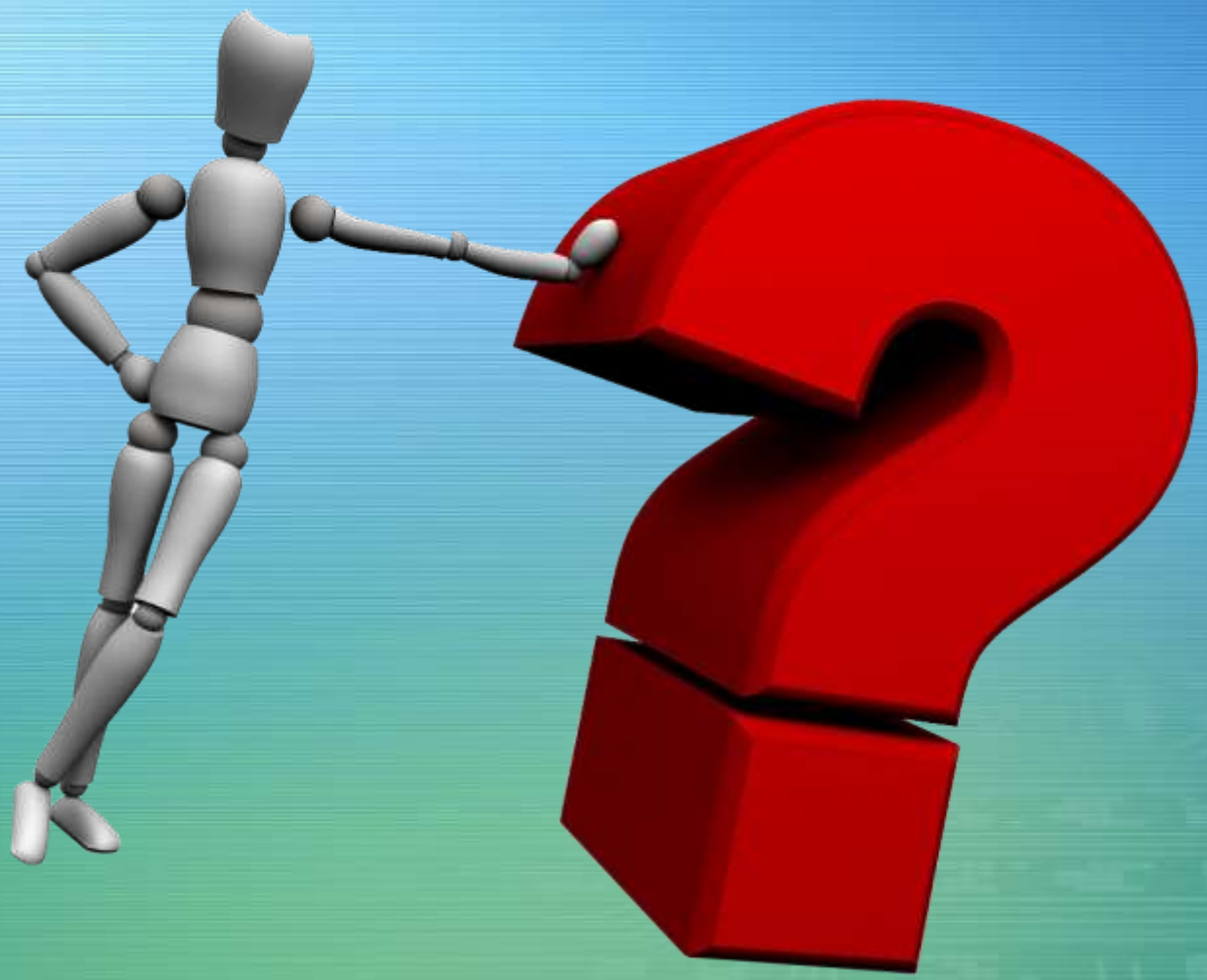
Linux *can-utils* Dump


```
# ip link set can0 type can help
Usage: ip link set DEVICE type can
      [ bitrate BITRATE [ sample-point SAMPLE-POINT] ] |
      [ tq TQ prop-seg PROP_SEG phase-seg1 PHASE-SEG1
        phase-seg2 PHASE-SEG2 [ sjw SJW ] ]

                  [ dbitrate BITRATE [ dsample-point SAMPLE-POINT] ] |
                   [ dtq TQ dprop-seg PROP_SEG dphase-seg1 PHASE-SEG1
                   dphase-seg2 PHASE-SEG2 [ dsjw SJW ] ]

      [ loopback { on | off } ]
      [ listen-only { on | off } ]
      [ triple-sampling { on | off } ]
      [ one-shot { on | off } ]
      [ berr-reporting { on | off } ]
                  [ fd { on | off } ]
      [ fd-non-iso { on | off } ]
      [ presume-ack { on | off } ]

      [ restart-ms TIME-MS ]
      [ restart ]
```

```
# ip link set can0 type can bitrate 960000 loopback on
[  318.736629] m_can_platform 25128000.can can0: bitrate error 0.7%
# ip link set can0 up
[  328.984922] [MCAN] Message RAM initialised
# candump can0&
# cansend can0 16A#1122334455667788
#   can0  16A   [8]  11 22 33 44 55 66 77 88
  can0  16A   [8]  11 22 33 44 55 66 77 88
```

```
# ip -details -statistics link show can0
2: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode
DEFAULT group default qlen 10
    link/can  promiscuity 0
    can <LOOPBACK> state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
        bitrate 952380 sample-point 0.738
        tq 25 prop-seg 15 phase-seg1 15 phase-seg2 11 sjw 1
        m_can: tseg1 2..256 tseg2 2..128 sjw 1..128 brp 1..512 brp-inc 1
        m_can: dtseg1 1..32 dtseg2 1..16 dsjw 1..16 dbrp 1..32 dbrp-inc 1
        clock 40000000
        re-started bus-errors arbit-lost error-warn error-pass bus-off
        0       0       0       0       0       0
    RX: bytes  packets  errors  dropped overrun mcast
    8       1       0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    8       1       0       0       0       0
```

Any Questions ?



THANK YOU