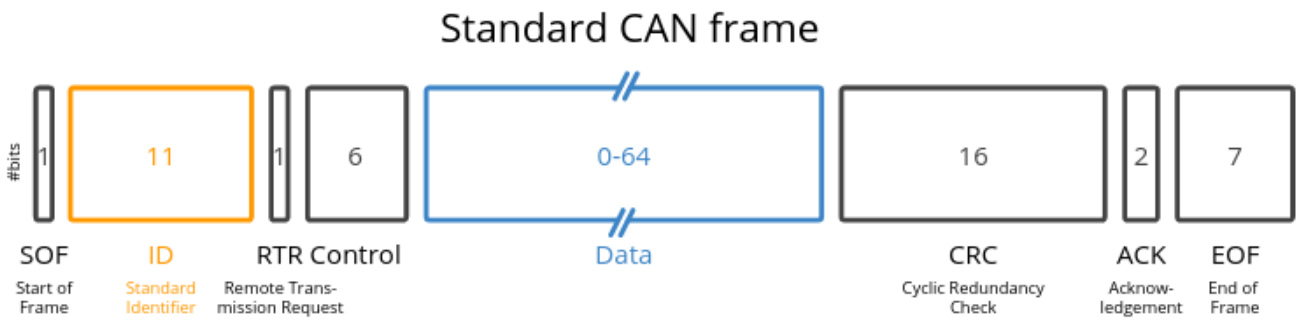
CAN

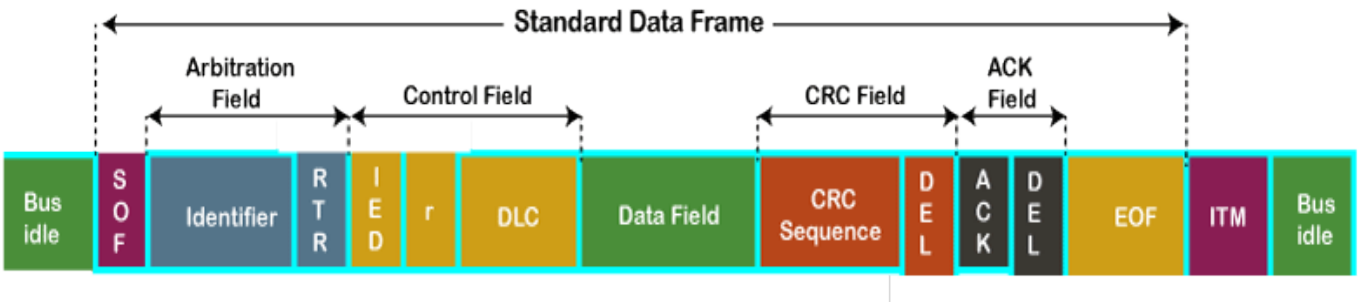
Introduction :

i) CAN stands for Controller Area Network protocol. It is a protocol that was developed by Robert Bosch in around 1986.
ii) The CAN protocol is a standard designed to allow the microcontroller and other devices to communicate with each other without any host computer.
iii) The feature that makes the CAN protocol unique among other communication protocols is the broadcast type of bus. Here, broadcast means that the information is transmitted to all the nodes.
iv) The node can be a sensor, microcontroller, or a gateway that allows the computer to communicate over the network through the USB cable or ethernet port. The CAN is a message-based protocol, which means that message carries the message identifier, and based on the identifier, priority is decided.
v) There is no need for node identification in the CAN network, so it becomes very easy to insert or delete it from the network. It is a serial half-duplex and asynchronous type of communication protocol.
vi) The CAN is a two-wired communication protocol as the CAN network is connected through the two-wired bus. The wires are twisted pair having 120Ω characteristics impedance connected at each end.

CAN Framing:

Let's understand the structure of the CAN frame.





- **SOF:** SOF stands for the start of frame, which indicates that the new frame is entered in a network. It is of 1 bit.
- **Identifier:** A standard data format defined under the CAN 2.0 A specification uses an 11-bit message identifier for arbitration. Basically, this message identifier sets the priority of the data frame.
- **RTR:** RTR stands for Remote Transmission Request, which defines the frame type, whether it is a data frame or a remote frame. It is of 1-bit.
- **Control field:** It has user-defined functions.
    1. **IDE:** An IDE bit in a control field stands for identifier extension. A dominant IDE bit defines the 11-bit standard identifier, whereas recessive IDE bit defines the 29-bit extended identifier.
    2. **DLC:** DLC stands for Data Length Code, which defines the data length in a data field. It is of 4 bits.
    3. **Data field:** The data field can contain upto 8 bytes.
- **CRC field:** The data frame also contains a cyclic redundancy check field of 15 bit, which is used to detect the corruption if it occurs during the transmission time. The sender will compute the CRC before sending the data frame, and the receiver also computes the CRC and then compares the computed CRC with the CRC received from the sender. If the CRC does not match, then the receiver will generate the error.
- **ACK field:** This is the receiver's acknowledgment. In other protocols, a separate packet for an acknowledgment is sent after receiving all the packets, but in case of CAN protocol, no separate packet is sent for an acknowledgment.
- **EOF:** EOF stands for end of frame. It contains 7 consecutive recessive bits known End of frame.

To send the CAN data into Arduino or to dump into device from Arduino we follow below steps:---

1. For arduino software downloading steps:

i) wget -c https://downloads.arduino.cc/arduino-1.8.19-linux64.tar.xz
ii) tar -xf arduino-1.8.19-linux64.tar.xz
iii) cd arduino-1.8.19
iv) sudo ./install.sh
v) arduino
vi)Add libraries (i.e.,CAN_BUS_Shield-2.3.3 4.zip file) in sketch/include library/Add .zip Library.

2. Downloading CAN utils:

i) git clone https://github.com/linux-can/can-utils.git
ii) export ARCH=arm
iii) export CROSS_COMPILE=/opt/gcc-linaro-7.4.1-2019.02-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-
iv) ./autogen.sh
v) ./configure
vi) make

Note: This steps helpful to get candump and cansend tool.

3. Steps to flash the YP source code into device and CAN connections.

i) Open the flashtool then add Andriod_scatter.txt in scatter-loading File and keep in Firmware update mode.
ii) Then click on Download option.
iii) While downloading plug-in the power adapter pin and adb cable to device.
iv) After complition of downloading remove power adapter and adb cable connection and again plug-in both cables power ON the device by pressing power button.
v) With the help of adb devices command check whether device is connected or not.
vi) Connect the Arduino board to PC with of usb cable and take two patch wires and connect to Arduino CAN_H and CAN_L pin to DB-9 connector pins then continue with step-4.

4. Steps to dump CAN data from Arduino to device(YP_Board):

i) While sending CAN data to device follow below steps:-
    a) Add **Matter.ino** file in file/open option.
    a) Go to Tools option in Arduino and check Board status whether it is in “Arduino Uno” or not.
    b) Again go to tools option and set port option to /*dev*/usb while arduino is connnected.
    c) Then click on verify option and upload into Arduino.
ii) adb push ~/candump /data
iii) adb shell
iv) su
v) cd /data
vi) chmod +x candump
vii) ifconfig
    --After giving ifconfig command if we don’t find can0 node then following two steps:--
    a) ip link set can0 up type can bitrate 500000
    b) ip link set up can0
    ---To remove can0 we use :--
    a) ip link set can0 down
viii) ./candump can0

ix) check whether the messages id's are getting or not on terminal(i.e.,message id's present in matter.ino file).

5. Steps to send CAN code from device to Arduino Board:

i) While receiving CAN data from device follow below steps:-
    a) Add **receive_check.ino** file which is present in file/examples/CAN_BUS_SHIELD option.
    b) Go to Tools option in Arduino and check Board status whether it is in "Arduino Uno" or not.
    c) Again go to tools option and set port option to /*dev*/usb while arduino is connnected.
    d) Then click on verify option and upload into Arduino.
ii) adb push ~/cansend /data
iii) adb shell
iv) su
v) cd /data
vi) chmod +x cansend
vii) ifconfig
viii) ./cansend can0 350#000000000000001100
    (here 350 is CAN data message id and 000000000000001100 is data)
ix) check whether the messages id's are getting in serial monitor or not(i.e.,serial monitor present in arduino and message id's present in receive_check file).

Steps for loopback test :--

It is possible to configure the SocketCAN in internal Loopback test mode. In that case, the FDCAN treats its own transmitted messages as received messages. This mode can be used for hardware self-test.

Take two terminal :---

-----Short the CAN_H and CAN_L pins in DB-9 connector with the help of one jumper wire.

In 1st terminal:

i) adb push ~/cansend /data
ii) adb shell
iii) su
iv) cd /data
v) chmod +x cansend
vi) ifconfig
    ----ip link set can0 down
    ----ip link set can0 up type can bitrate 500000 loopback on
vii) ./cansend can0 350#000000000000001100

In 2nd terminal:

i) adb push ~/canddump /data
ii) adb shell
iii) su
iv) cd /data
v) chmod +x candump
vi) ifconfig
    ----ip link set can0 down
    ----ip link set can0 up type can bitrate 500000 loopback on
vii) ./candump can0

Note:-- while sending CAN messege on terminal-1 with **./cansend can0 350#0000000000000001100** it will receive the data by **./candump can0** command we can check on terminal-2)