CAN Board-to-Board Communication using RB-A5D2X Boards

What is CAN:-

CAN (Controller Area Network) is a robust communication protocol designed for real-time data exchange between devices in embedded systems. It was originally developed by Bosch for the automotive industry but is now widely used in various industrial applications.

How Does CAN Work:-

1. Bus Architecture:
    - CAN uses a two-wire bus system (CAN-H and CAN-L) for communication.
    - All nodes (devices) connect to the same bus, enabling a multi-master setup.
2. Message-Based Protocol:
    - Instead of addressing specific devices, CAN sends data packets identified by unique message IDs.
    - This makes it efficient and flexible for dynamic systems.
3. Error Handling:
    - CAN includes error detection and fault confinement mechanisms to ensure reliable communication.
4. Arbitration:
    - When multiple nodes transmit simultaneously, CAN resolves conflicts using a priority-based arbitration system.

Objective

To establish communication between two RB-A5D2X boards using the CAN interface and validate the data transmission and reception.

Prerequisites

1. **Hardware Requirements:**
    - 2 RB-A5D2X boards.
    - CAN transceiver modules (if required).
    - Proper wiring setup between the CAN ports of both boards.
2. **Software Requirements:**
    - Boards booted using a preconfigured SD card image.

    - Linux kernel with CAN protocol support enabled.
    - `can-utils` package installed on both boards (optional for debugging).
3. **Setup Requirements:**
    - CAN protocol settings enabled on both boards.
    - Ensure a proper CAN cable or bus wiring between the boards.

CAN Pin Configuration on Rugged Board (RB-A5D2X)

The **Rugged RB-A5D2X** boards typically use a **CAN transceiver** for CAN communication. Here are the key pins related to CAN on such boards:

Pin Names on RB-A5D2X for CAN Communication

- **CAN_H** (CAN High): This is the positive line of the differential pair used for CAN communication.
- **CAN_L** (CAN Low): This is the negative line of the differential pair used for CAN communication.
- **GND**: Ground reference for the CAN bus.
- **Vcc**: Power supply for the CAN transceiver.

2. Wiring the CAN Communication Between Two Rugged Boards

To establish CAN communication between two **RB-A5D2X boards** (one as sender and one as receiver), you will need to wire the **CAN_H** and **CAN_L** pins from one board to the corresponding pins on the other board. Below are the steps to make the connection:

Board 1 (CAN Transmitter) Connections

- CAN_H (Board 1) → CAN_H (Board 2)
- CAN_L (Board 1) → CAN_L (Board 2)
- GND (Board 1) → GND (Board 2) (Ensure a common ground between both boards)

Board 2 (CAN Receiver) Connections

- CAN_H (Board 2) → CAN_H (Board 1)
- CAN_L (Board 2) → CAN_L (Board 1)
- GND (Board 2) → GND (Board 1) (Ensure a common ground between both boards)

These connections ensure that the **CAN_H** and **CAN_L** differential signals are correctly shared between the two boards, enabling proper CAN communication.

Software Configuration for CAN Communication:-

Step 1: CAN Protocol Configuration on Both Boards

Perform the following commands on both boards to configure the CAN interface.

On the First Board:

```
ip link set can0 down
ip link set can0 type can bitrate 50000
ip link set can0 up
```

On the Second Board:

```
ip link set can0 down
ip link set can0 type can bitrate 50000
ip link set can0 up
```

Step 2: Set Up Cross-Compilation Environment

If cross-compilation is required, set up the environment on your host machine:

```
. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
```

Step 3: Compilation of CAN Sender and Receiver Applications

Compile the sender and receiver programs for the RB-A5D2X boards using the following commands:

Compilation Command:

```
${CC} cansender.c -o cansender
${CC} canreceiver.c -o canreceiver
```

Replace `${CC}` with the appropriate cross-compiler (e.g., `arm-poky-linux-gnueabi-gcc`).

Source Code

CAN Sender Application (`cansender.c`)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <string.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>

int main(void) {
        int s, nbytes;
        struct sockaddr_can addr;
        struct can_frame frame;
        struct ifreq ifr;

        char *ifname = "can0";
        printf("CAN Send Test\n");

        if ((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
        perror("Error while opening socket");
        return -1;
        }

        strcpy(ifr.ifr_name, ifname);
        ioctl(s, SIOCGIFINDEX, &ifr);

        addr.can_family = AF_CAN;
        addr.can_ifindex = ifr.ifr_ifindex;

        printf("%s at index %d\n", ifname, ifr.ifr_ifindex);

        if (bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("Error in socket bind");
        return -2;
        }

        frame.can_id = 0x123;
        frame.can_dlc = 2;
        frame.data[0] = 0x11;
        frame.data[1] = 0x22;

        nbytes = write(s, &frame, sizeof(struct can_frame));
        printf("Wrote %d bytes\n", nbytes);
        return 0;
}
```

CAN Receiver Application (`canreceiver.c`)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>

#define BUF_SIZ (255)

int main(void) {
        int s, nbytes;
        struct sockaddr_can addr;
        struct can_frame frame;
        struct ifreq ifr;
        char buf[BUF_SIZ];

        char *ifname = "can0";
        printf("CAN Receive Test\n");

        if ((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
        perror("Error while opening socket");
        return -1;
        }

        strcpy(ifr.ifr_name, ifname);
        ioctl(s, SIOCGIFINDEX, &ifr);

        addr.can_family = AF_CAN;
        addr.can_ifindex = ifr.ifr_ifindex;

        printf("%s at index %d\n", ifname, ifr.ifr_ifindex);

        if (bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("Error in socket bind");
        return -2;
        }
       while (1) {
```

```
        if ((nbytes = read(s, &frame, sizeof(struct can_frame))) < 0) {
        perror("Read Error");
        return 1;
        } else {
        snprintf(buf, BUF_SIZ, "<0x%03x> [%d] %02x %02x",
                frame.can_id, frame.can_dlc, frame.data[0], frame.data[1]);
        printf("Received CAN Frame: %s\n", buf);
        }
        }
            return 0;
}
```

Application Execution

1. **On the First Board (Sender):** Execute the sender application:

   ./cansender

2. **On the Second Board (Receiver):** Execute the receiver application:

   ./canreceiver

Expected Output

On the Sender Board:

When running the `cansender` application, the output will be:

```
CAN Send Test
can0 at index 3
Wrote 16 bytes
```

On the Receiver Board:

When running the `canreceiver` application, the output will be:

```
CAN Receive Test
can0 at index 3
Received CAN Frame: <0x123> [2] 11 22
```

Result

The CAN data transmitted from the sender (`cansender.c`) will be received and displayed on the receiver (`canreceiver.c`).

Sender CAN Frame:

can_id = 0x123
can_dlc = 2
data[0] = 0x11
data[1] = 0x22

Receiver Output:
Received CAN Frame: <0x123> [2] 11 22

Applications of CAN

1. Automotive

- Engine control units (ECUs), airbags, ABS, and infotainment systems.
- Enables efficient communication among various subsystems in vehicles.

2. Industrial Automation

- PLCs and sensors in factories use CAN for real-time process control.

3. Medical Devices

- CAN is used in medical equipment like imaging systems, ventilators, and patient monitors.

4. Robotics

- Used in multi-axis robotic arms for coordinating motors and sensors.

5. Agriculture and Heavy Machinery

- CAN is implemented in tractors, harvesters, and construction machinery for monitoring and control.