Moter ICEBERG100 Gateway Application Board User Guide

Abstract: This document describes the steps for setup, usage of Moter ICEBERG100 gateway application board and testing procedure

STATUS OF THE DOCUMENT	DOCUMENT REFERENCES				
DRAFT	Project. Reference	Item Number	Doc. Identification	Revision Index	Page
	ICEBERG100_SOM	001	108	0.6	1/51

Document History

Version	Date	By	Changes
V0.1	19-08-2022	Anusha	Creation
V0.2	19-08-2022	Vinay K	Updated with software section 8
V0.3	24-08-2022	Anusha	Updated Section 9
V0.4	30-08-2022	Charan	Updated APIs test in 8.4
V0.5	16-09-2022	Charan	Updated with section 8.3 and 8.4
V0.6	28-09-2022	Vinay	Updated with section 8.2.1.6

Table of contents

List of Figures

1 INTRODUCTION

This SoM is based on a ICEBERG100_SOM high-performance vehicle network processor that combine controller area network (CAN), local interconnect network (LIN), and FlexRay networking with high-data-rate Ethernet networking. It also combines a functional safe-core infrastructure with MPU cores and includes high-level security features.

Purpose

This document brings out the steps for set-up, usage of ICEBERG100_SOM Gateway application board, flashing and booting procedure. Also, it explains detail step by step procedure of Linux test application developed by Tessolve

ICEBERG100_SOM System on module (SOM) is designed according to the specification specified by SGET for SMARC (Smart Mobility Architecture). SMARC standard defines the signals to be carried out through MXM 3.0 connector with pin outs newly defined by SGET group.

Scope

This document describes the below:

- Architecture of SoM
- Architecture of Gateway Board
- Product interfaces and pin outs
- Flashing and booting procedure
- Test application

2 Definitions, Abbreviations and Acronyms

Abbreviations

- **SOM:** System On Module
- **CAN:** Controller Area Network
- **LIN:** Local Interconnect Network
- **USB:** Universal Serial Bus
- **PCIe:** Peripheral Component Interconnect Express
- **IMU:** Inertial Measurement Unit
- **GNSS:** Global navigation satellite system
- **SDIO:** Secure Digital Input Output
- **eMMC:** Embedded Multi-Media Card
- **UART:** Universal asynchronous receiver-transmitter

Acronyms

- **SMARC:** Smart Mobility ARChitecture

3 Architecture of Gateway Application Board

Product Mechanical View

Figure 1 : S32G2 Product Mechanical View

S32G2 SoM Architecture

Below figure shows the high-level block diagram of S32G2 SMARC SoM.

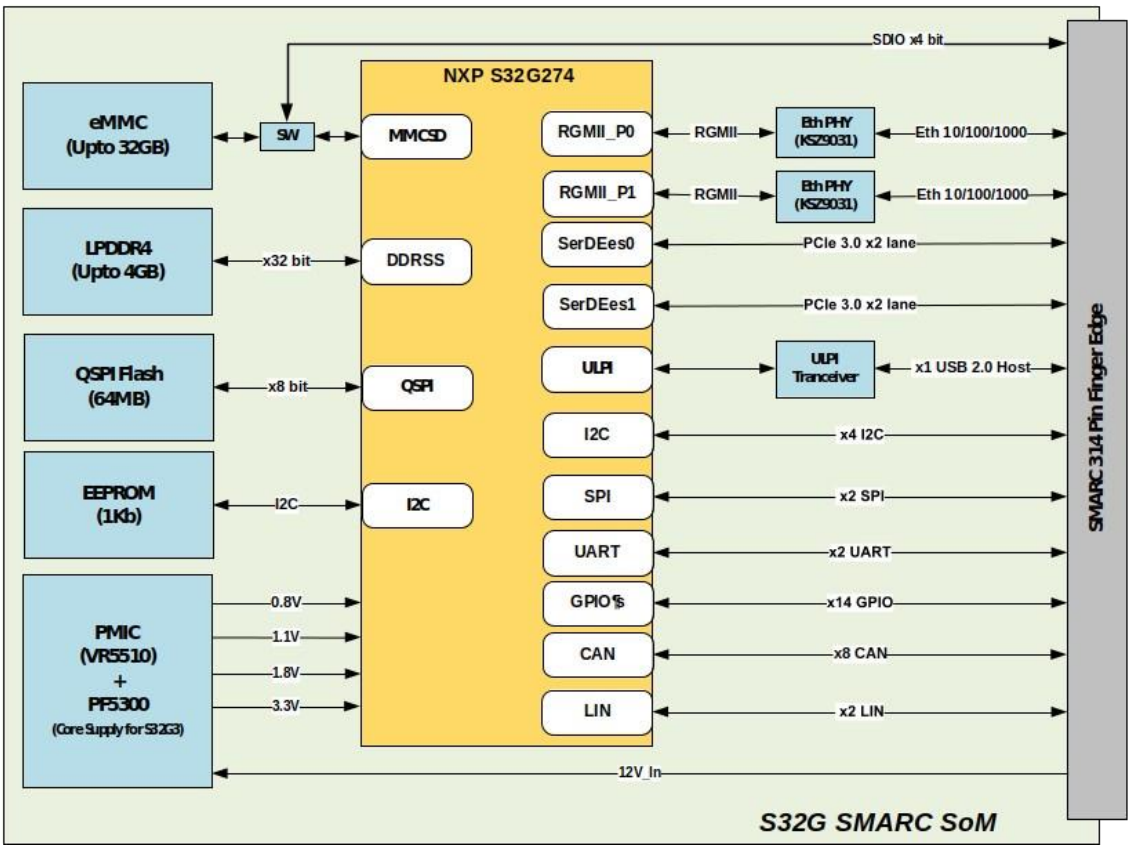

Figure 2 : S32G2 SoM Architecture Block Diagram

Moter Gateway Application board Architecture

The figure below shows the high level block diagram of Moter Gateway Application board

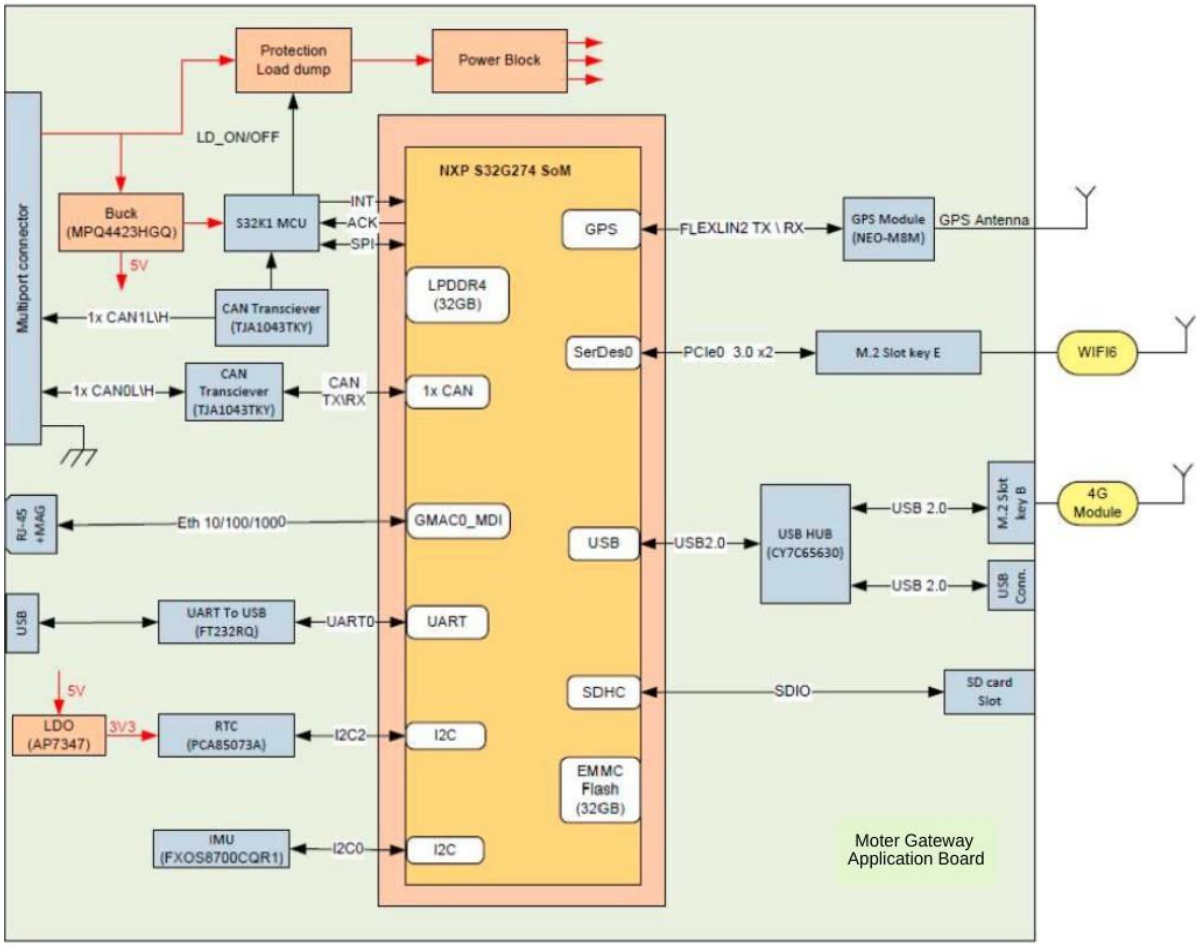

Figure 3: Moter Gateway Application board Architecture Block Diagram

4 Moter Gateway Application board Interfaces

High Level Interfaces

The Table below shows the high level Interfaces of Moter Gateway Application board.

Technical Information	
Module Name	Moter SOM
Processor (CPU APPLICATION)	· NXP S32G2 (S32G274A) · ARMv8 CPU 64 Bit · Quad core · Cluster 0: dual Cortex-A53 core @1 GHz · Cluster 1: dual Cortex-A53 core@1 GHz
Processor (CPU REALTIME)	· 3 Cortex-M7 cores in lockstep · Supports 64 bit addressing with 400 MHz
Memory	· LPDDR4-1866, 2GB/4GB (x64 bits) · Dual Channel high-speed memory
SDIO	· SDIO port can either be used for 1. eMMC 5.1 (Boot & Storage) 2. SD Card (Boot & Storage)
Ethernet	· x1 1Gbps Ethernet Ports
CAN/LIN	· x1 LLCE CAN and x2 FLEX CAN · x1 LLCE LIN and x1 FLEX LIN
USB 2.0	· x1 USB 2.0 Host Ports
PCIe 0	· PCIe 0 port can be used for: 1. WIFI 6 MODULE
USB	· 4G LTE module
GNSS	· The NEO-M8 modules utilize concurrent reception of up to three GNSS systems. GPS/Galileo together with BeiDou or GLONAS
RTC/IMU	· RTC Controller and IMU Sensor
SIM Card Slot	· Micro SIM connector for 4G/5G
DEBUG UART	· x1 USB to UART Debug Port

Table 1 : Moter Gateway Application Board High Level Interfaces

Interface Connection Details

The section shows the details of External interfaces and usage of Moter gateway Application board.

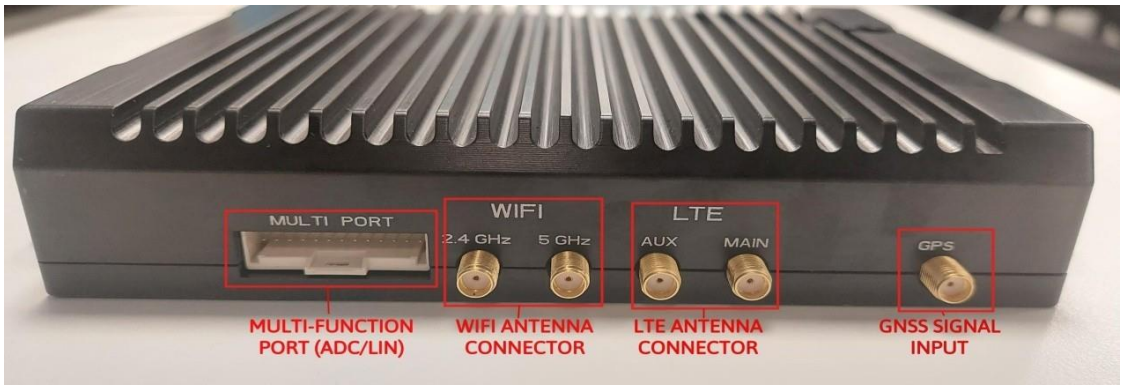

Figure 4 : Moter Gateway External connectors 1

Figure 5 : Moter Gateway External connectors 2

Figure 6 : Moter Gateway External connectors 3

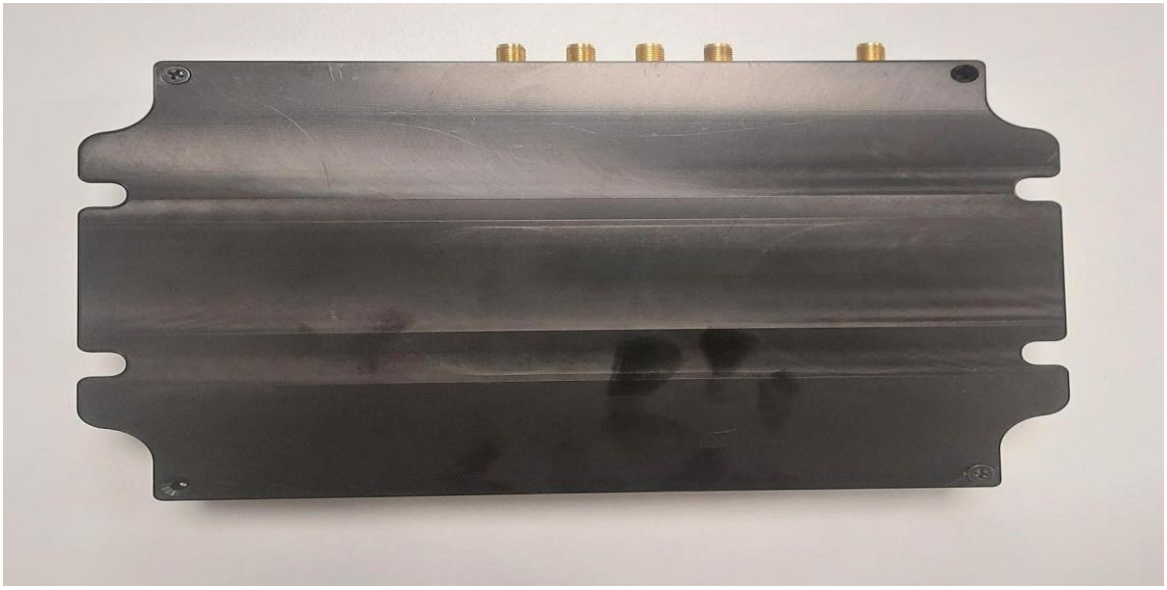

Figure 7 : Moter Gateway External connectors 4

Input Power Connector:

- Power Input Jack (5.5mm OD connector) used to supply power to Gateway board.
- System support 6 to 36V DC power supply. Typical power supply to the board is 12V/3A DC.

Wi-Fi or 4G/5G Antenna SMA connectors:

- Three SMA connectors are used for connect Wi-Fi or 4G/5G module antennas.

GNSS Antenna Connector:

- SMA connector for connect external GPS antenna.

Ethernet RJ45 Jacks:

- Two RJ45 connectors are provided for Ethernet connectivity.
- Both Ethernet ports support up to 1GbE transfer speed.

USB Host Ports

- Three USB 2.0 Host ports are provided for connect external USB devices.

Debug UART

- Micro USB connector is provided to debug Gateway application board.

Status LED

- One dual colour Status LED is provided to indicate operation status of the system.
- One RED LED is provided to indicate Load ON/OFF.

Multi-Function Port

Multi-function port is used for optional power supply to the board and it supports power out for external boards. This Multi-function port support LIN, ADC, I2C and SPI hardware interfaces.

Figure 8 : Multi-function Port connector pin outs

Multi function connector pin outs as below,

MULTI -FUNCTION PORT			
Pin No.	**Pin Name**	**Description**	**Comment**
1	PGND	Power GND	Input Power Supply Ground
2	VDD_PWR_IN	Power	6V to 36V Input Power Supply
3	PGND	Power GND	Input Power Supply Ground
4	VDD_PWR_IN	Power	6V to 36V Input Power Supply
5	PGND	Power GND	Input Power Supply Ground
6	VDD_PWR_IN	Power	6V to 36V Input Power Supply
7	DSPI1_SIN	I	SPI Master Serial Data In
8	VDD_5V0_OUT	Power	5V Power Output
9	DSPI1_PCS0	O	SPI Master Serial Data In
10	VDD_12V0_OUT	Power	12V Power Output
11	GND	DIG GND	Digital Ground
12	VDD_3V3_OUT	Power	3.3V Power Output
13	GND	DIG GND	Digital Ground
14	FLEX_LIN2	I/O	LIN bus line input/output
15	LLCE_LIN1	I/O	LIN bus line input/output
16	LLCE_LIN0	I/O	LIN bus line input/output
17	LLCE_LIN3	I/O	LIN bus line input/output
18	LLCE_LIN2	I/O	LIN bus line input/output
19	I2C2_SDA	I/O	I2C SDA
20	I2C2_SCL	O	I2C SCL
21	GND	DIG GND	Digital Ground
22	DSPI1_SOUT	I	SPI Master Serial Data Out
23	ADC_CH_00	I	SAR ADC Input Channel
24	DSPI1_SCK	O	SPI Master Serial Clock
25	ADC_CH_01	I	SAR ADC Input Channel
26	ADC_CH_02	I	SAR ADC Input Channel

Table 2 : Multi Function connector pin outs

SIM card

- SIM connector is provided for wireless communication through 4G/5G network.

SD Card

- MicroSD card connector is provided for external storage.
- Micro SD card supports for system Boot and Storage purpose.

Reset switch

- Reset switch is provided in bottom side of the mechanical system for complete system reset.

Power ON/OFF switch

- SPDT switch is provided to turn OFF the board and also select the power Input from Power Jack and Multi Port connector.

5 Dip Switch Settings

The section shows the details of DIP switch settings in Moter SoM and Gateway Application board.

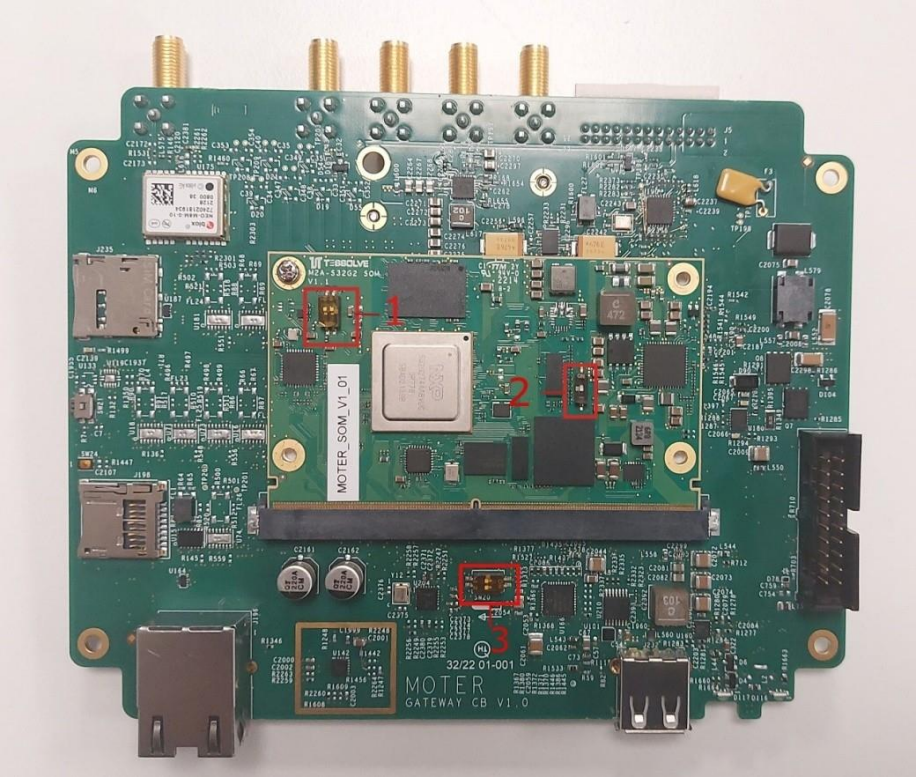

Figure 9 : Gateway Application Board (Front view)

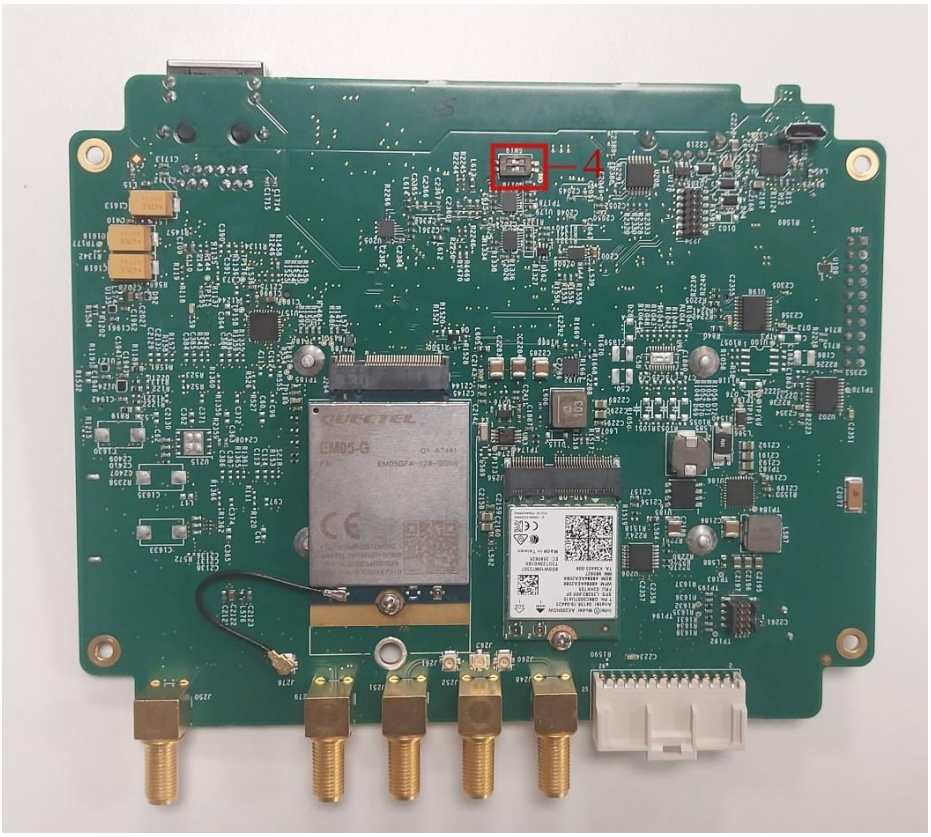

Figure 10 : Gateway Application Board (Back View)

NO	Switch	Function
1	SW10	Boot Device Selection Switch
2	SW3	eMMC/SDIO Selection Switch
3	SW20	Boot mode 1 Switch
4	SW19	Boot mode 0 Switch

Table 3 : Dip switches List

Boot Device Selection Switch (SW10):

This switch is used to select the boot devices like QSPI, eMMC or SD Card. Below truth table shows the switch settings to select preferred boot devices.

SW10		BOOT DEVICES
SW10(2)	SW10(1)	
OFF	OFF	QSPI
OFF	ON	RESERVED
ON	OFF	SD Card
ON	ON	eMMC

Table 4 : Boot device selection switch

eMMC/SDIO Selection Switch (SW3):

This switch must be configured in accordance to SW10. If we choose boot device as SD card then switch must be turned ON if we select eMMC as a boot device then SW3 must be turned off.

SW3	FUNCTION
OFF	eMMC
ON	SD Card

Table 5 : eMMC/SD selection switch

BOOT Mode switch (SW19 and SW20):

Below two switches are used to select different boot modes of processor-based use cases of S32G2 processor.

SW19		BOOTMOD1
SW19(2)	SW19(1)	
OFF	OFF	0
OFF	ON	1
ON	OFF	RESET
ON	ON	INV_RESET

SW20		BOOTMOD2
SW20(1)	SW20(2)	
OFF	OFF	0
OFF	ON	1
ON	OFF	RESET
ON	ON	INV_RESET

Table 6 : Boot mode selection switch

FUSE_SEL	BMODE1	BMODE2	Boot Mode	Available boot interfaces	Use Case
0	0	0	Serial Boot, XOSC configured in differential bypass mode	Serial communication interfaces: LIN (UART) and CAN	Production config for virgin devices
	0	1	Serial Boot, XOSC in Crystal mode or Bypass mode	Serial communication interfaces: LIN (UART) and CAN	
	1	0	Boot from external memory using RCON configurations	Memory interfaces: QSPI, SD card, MMC, eMMC	Development config
1	0	X	Boot from external memory using Fuse configuration	Memory interfaces: QSPI, SD card, MMC, eMMC	Production config for programmed devices
	1	0	Serial Boot	Serial communication interfaces: LIN (UART) and CAN	Debug of programmed device
X	1	1	Reserved		

Table 7 : Boot mode settings

6 Flashing procedure or programming procedure

QSPI Flash (Blank Device)

- Prepare binary image using S32DS IVT configurator. Use S32DS flash tool to program the QSPI flash. The tool is available with S32DS installation at directory
  <installation_directory> \S32DS.3.4\S32DS\tools\S32FlashTool\GUI\s32ft.exe
- Set the device to (serial) flashing mode. Switch status to Flash mode: “SW9” and “SW10” all OFF
- Connect the UART0 to PC via USB type-B cable
- Power ON the device
- Open S32 Flash Tool -> select COM -> enter the proper com number(com3)
- Select the proper Target (ex: S32G2xxx) -> select the proper Algorithm (bootmode for QSPI flash: MX25UW51245G)
- Click on 'Upload target and algorithm to hardware'
- Click on 'Upload file to device' for flashing. Need to mention the start address for flashing
- The device will boot from QSPI after the next reset

QSPI Flash (Already Flashed)

Perform the following steps in the U-Boot console. Binaries involved in a Linux boot (U-Boot, Linux Kernel image, DTB and ramdisk) on a TFTP server:

1. Copy u-boot.s32, Image, dtb file and ramdisk image to TFTP folder.
2. Connect Ethernet to the board
3. Prepare flash environment
   =>run flashbootargs
4. Setup the interface IP and TFTP server IP:
   =>setenvipaddr<board_ip>
   =>setenvserverip<server_ip_addr>
   => ping <server_ip_addr>
5. Load QSPI driver
   =>sf probe 6:0
   => sf erase 0 ${flashsize}
6. Update u-boot image
   =>setenv image u-boot.s32
   =>run loadtftpimage
   =>sf erase ${uboot_flashaddr} +${filesize}
   =>sf write ${loadaddr} ${uboot_flashaddr} ${filesize}
7. Update Linux Kernel Image
   =>setenv image Image
   => run loadtftpimage
   => sf erase ${kernel_flashaddr} +${kernel_maxsize}
   => sf write ${loadaddr} ${kernel_flashaddr} ${kernel_maxsize}
8. Update Linux device tree
   =>setenv image ${fdt_file}
   => run loadtftpimage
   => sf erase ${fdt_flashaddr} +${fdt_maxsize}
   => sf write ${loadaddr} ${fdt_flashaddr} ${fdt_maxsize}
9. Update the ramdisk image

=>setenv image fsl-image-base-<machine>.cpio.gz.u-boot
=> run loadtftpimage
=> sf erase ${ramdisk_flashaddr} +${ramdisk_maxsize}
=> sf write ${loadaddr} ${ramdisk_flashaddr} ${ramdisk_maxsize}

10. Power off the board and after setting the switches to boot from QSPI Flash, perform a power on reset of the board and verify.

Flashing rootfs.sdcard image to eMMC (Already Flashed)

Execute <s32ft> tool from GUI folder, Select the following options as shown in image below:

- Target: S32G2xxx
- Algorithm: EMMC

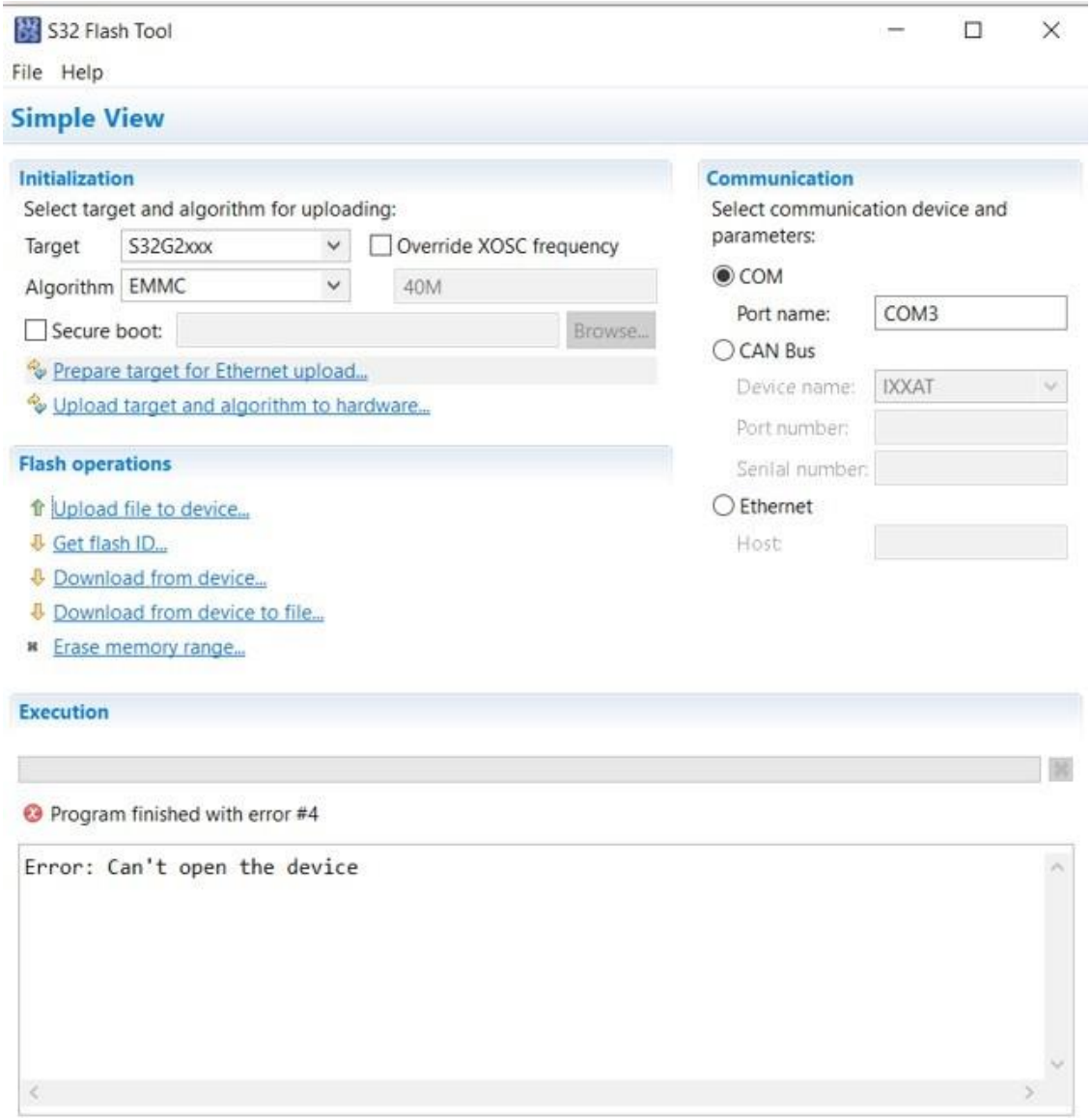

Figure 11 : eMMC Flashing

Flash rootfs.sdcard image by clicking <Upload file to device> option from tool GUI and select image folder as below:

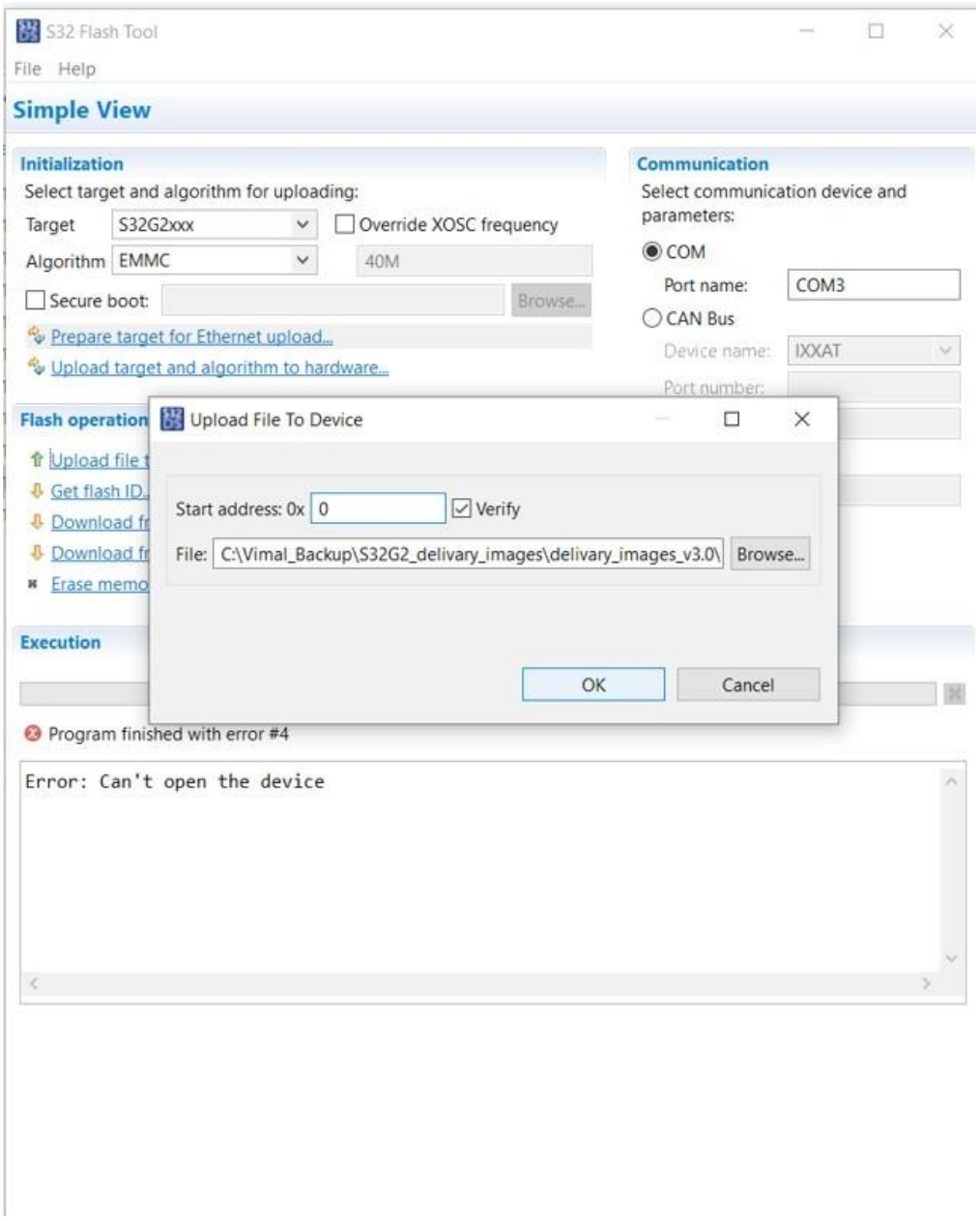

Figure 12 : eMMC Image Folder selection

7 Booting of the device

- After complete flashing, switch status to boot the device: SW10 pin1 as “ON”
- For QSPI mode booting switch: SW4 pin all as “OFF”
- Reset the system (Power supply on button reset)
- Install “Putty.exe” for debug serial COM port.
- Run putty.exe
- From the “Device Manager”, check for COM port number detected on windows PC.
- Select Serial, set COM port number in Serial Line and set baud rate to 115200 as mentioned below

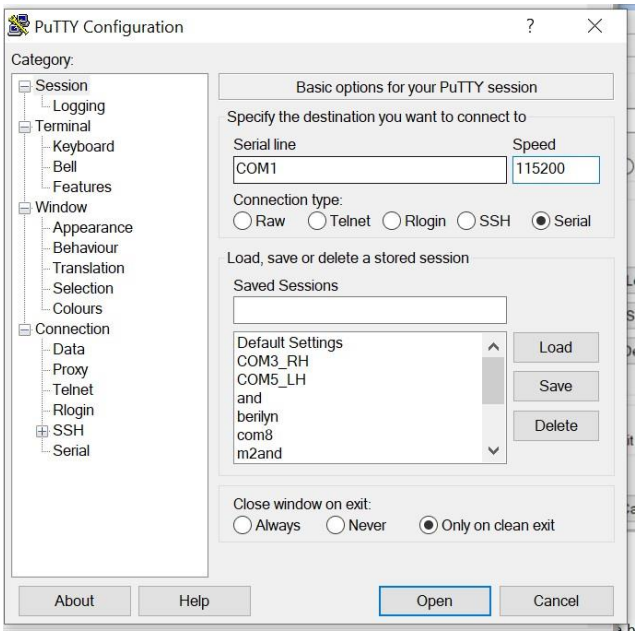

Figure 13 : eMMC Image Folder selection

- User can see the boot logs in COM port. Login as “root” as mentioned below

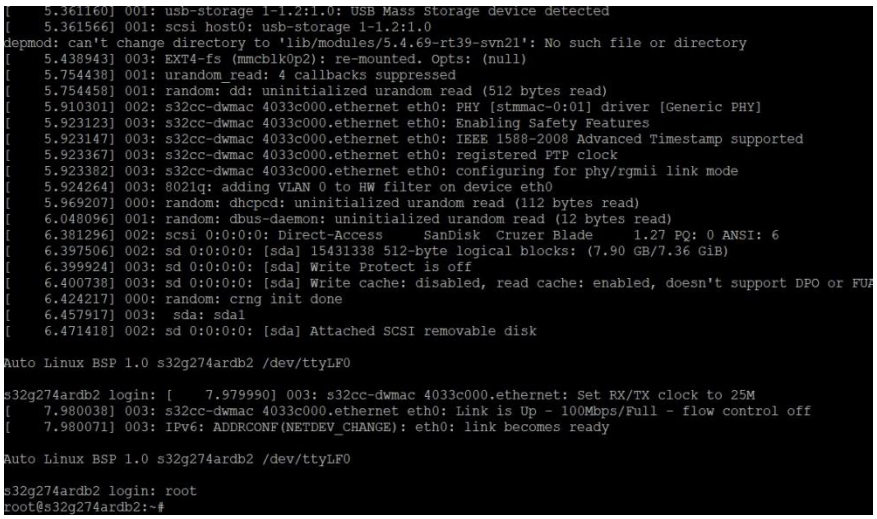

Figure 14 : Booting logs

- User can copy kernel, dtb and filesystem to eMMC after complete boot manually using "fdisk" command as
  <root@s32g274ardb2>:~# fdisk /dev/mmcblk0
  <root@s32g274ardb2>:~# mkfs.vfat /dev/mmcblk0p1
  <root@s32g274ardb2>:~# mount /dev/mmcblk0p1 /mnt/

- Copy kernel images to vfat partition using scp command from server to /mnt partition
- Also copy filesystem tar file through scp command on the board and untar it as below
  <root@s32g274ardb2>:~# umount /mnt/
  <root@s32g274ardb2>:~# mount -t ext3 /dev/mmcblk0p2 /mnt
  <root@s32g274ardb2>:~# cd /mnt/
  <root@s32g274ardb2>:/mnt# tar -xvf /home/root/fsl-image-base-s32g274ardb2 20220127111057.rootfs.tar.gz

- Reboot the device too boot kernel and file system from EMMC

8 MCU Firmware Info

Hardware Connection

Figure 15 : JTAG and Board Cable Connection

- Connect PE Micro JTAG Debugger
  - Connect JTAG Pins to Moter board
  - And USB side of Debugger to your PC(System)

Firmware development

Installation

S32 Design Studio Installation

Download the S32 Design Studio -

Error! Hyperlink reference not valid.

1. Follow the installation instruction

S32 Design Studio User Guide

1. New project path

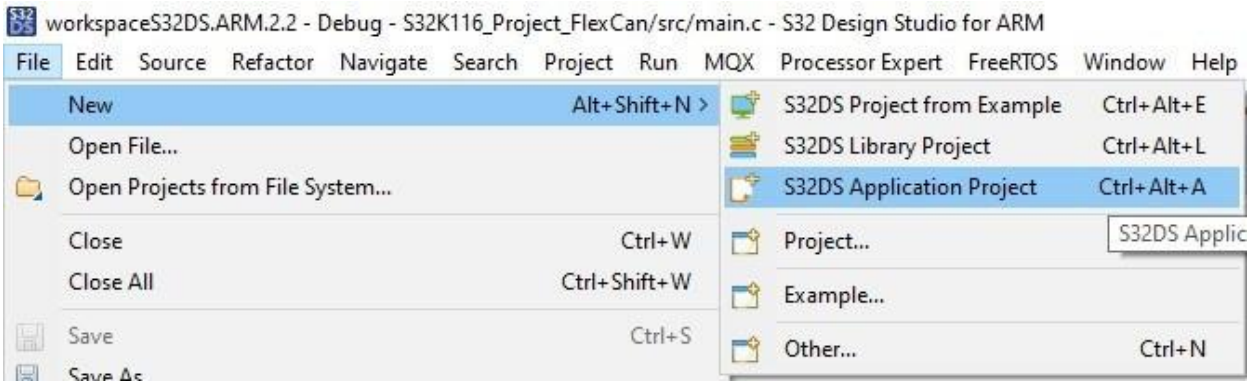

Figure 16 : Design Studio: New project

2. Name the project and click on the board version name what you are using

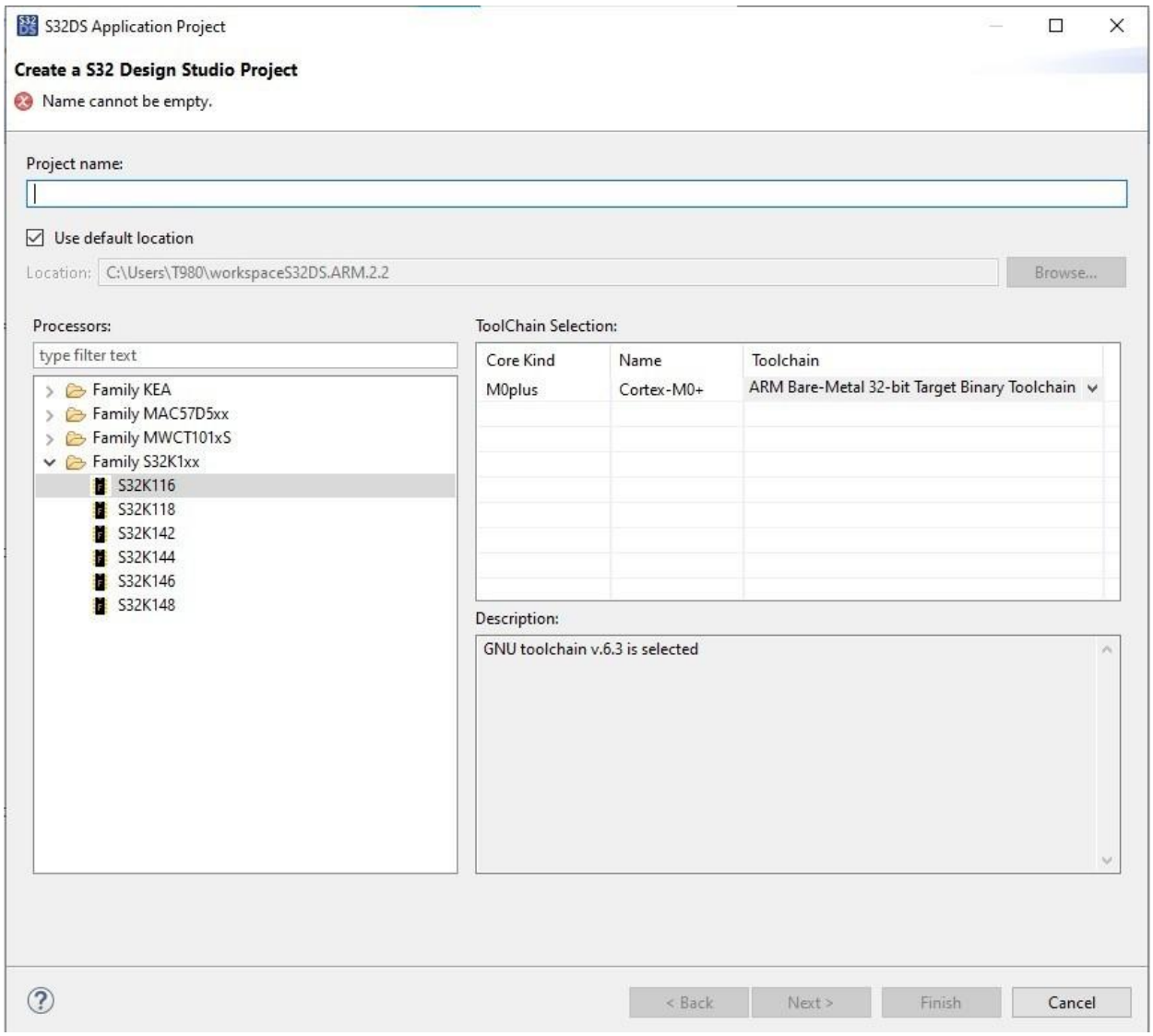

Figure 17 : Design Studio: Project and Controller selection

3. SDK selection

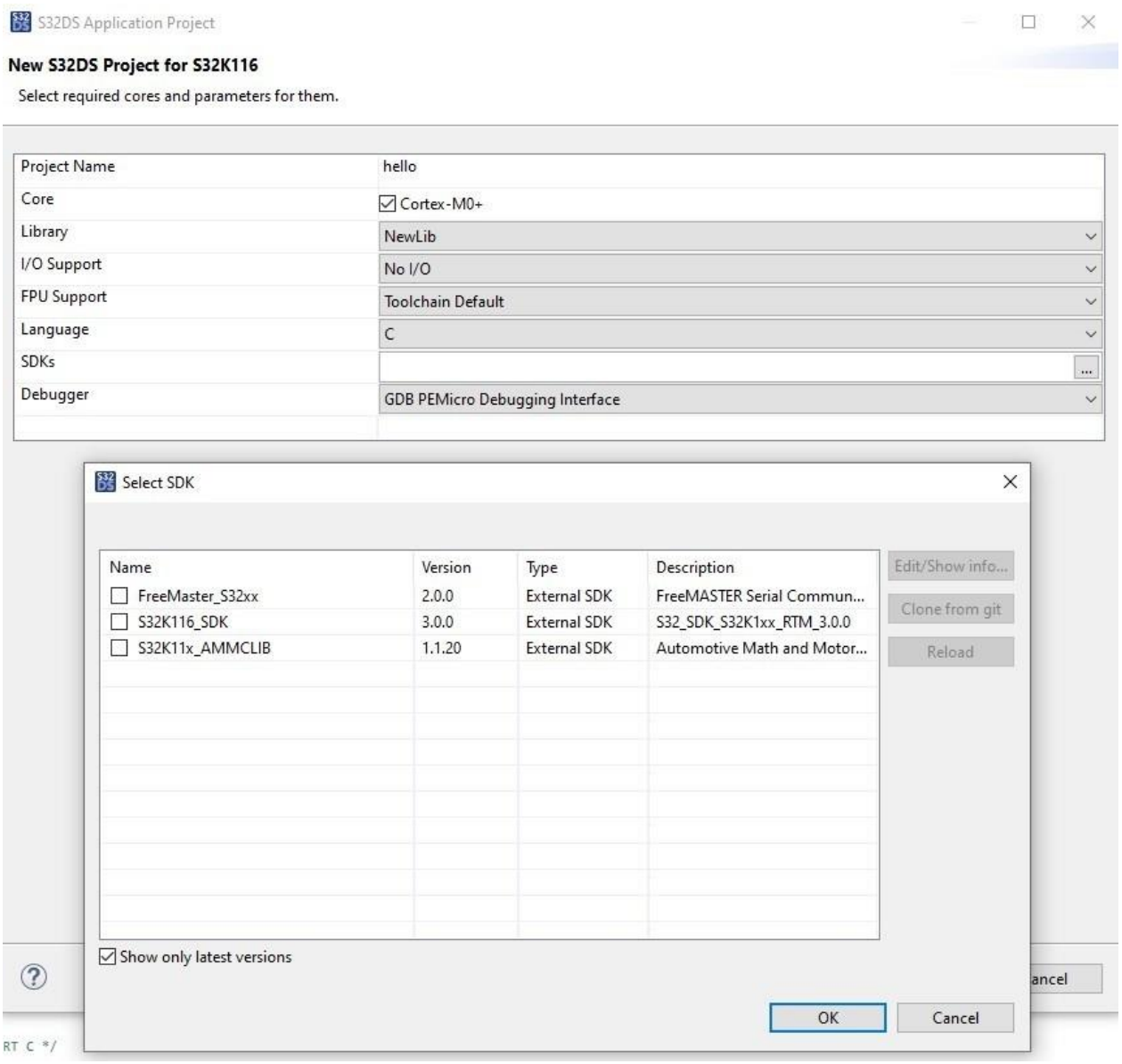

Figure 18 : Design Studio: SDK Selection

4. Connect the power cable and Power source 12 V DC

5. Connect the PE Micro debugger to carrier board and PC

Accessing project file

- Open S32 design studio
- File -> Open Project from File System
- Select your project which was downloaded and click open

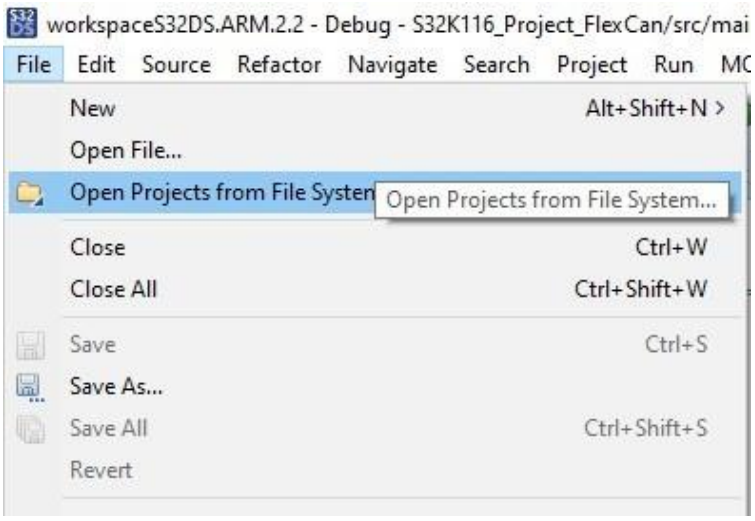

Figure 19 : Design Studio: Project file Selection

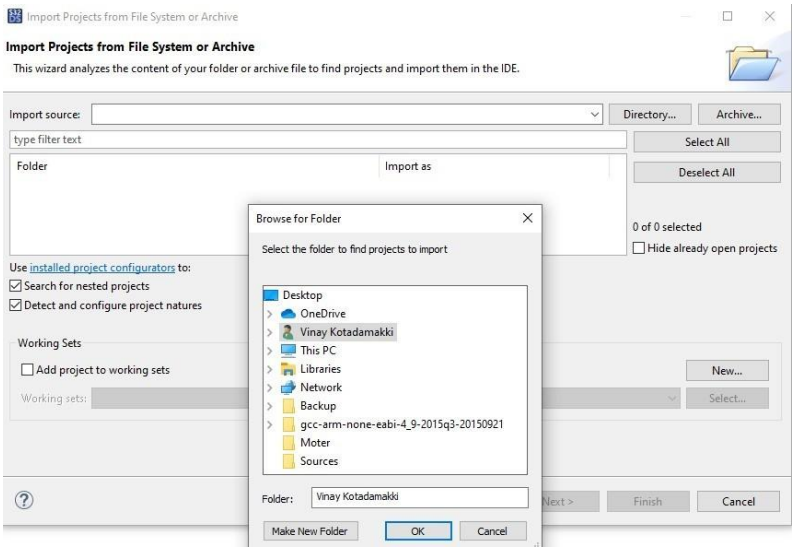

Figure 20 : Design Studio: Path selection

- In “src” folder we can see “main.c” file

- Open it to see the code and access it.
- Read the codes and comments so to know what that code will do or what that function does.

Flashing from file

- Connect the PE micro debugger to Motor board and PC
- Select the project you want to flash
- Press on flash icon . So it will start to flash the elf file to the hardware.
- Once it is completed the hardware ready to work without debugger.

Figure 21 : Design Studio: Debug Select

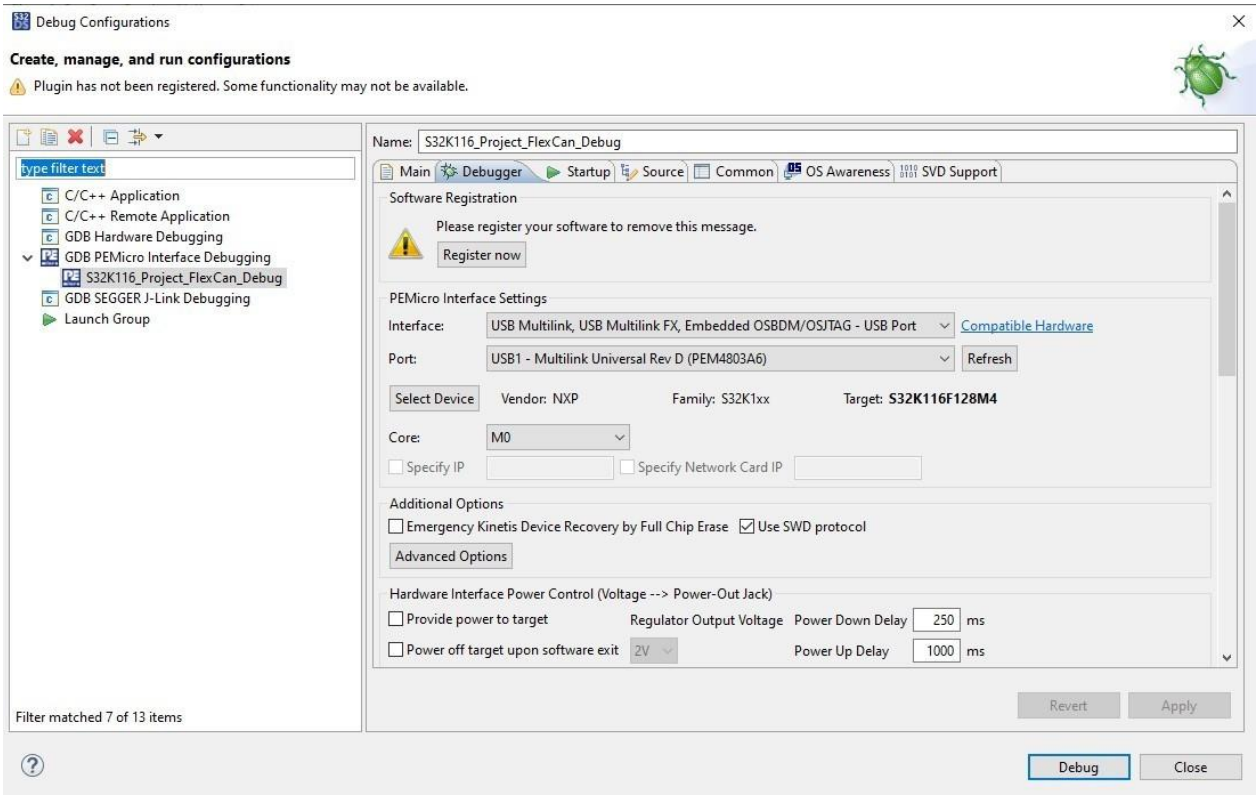

Figure 22 : Design Studio: Debug Configuration

CAN Communication

- Load control is enabled on power cycle
- This load control can be tested using CAN commands as follows

Figure 23 : CAN Analyzer: Connections

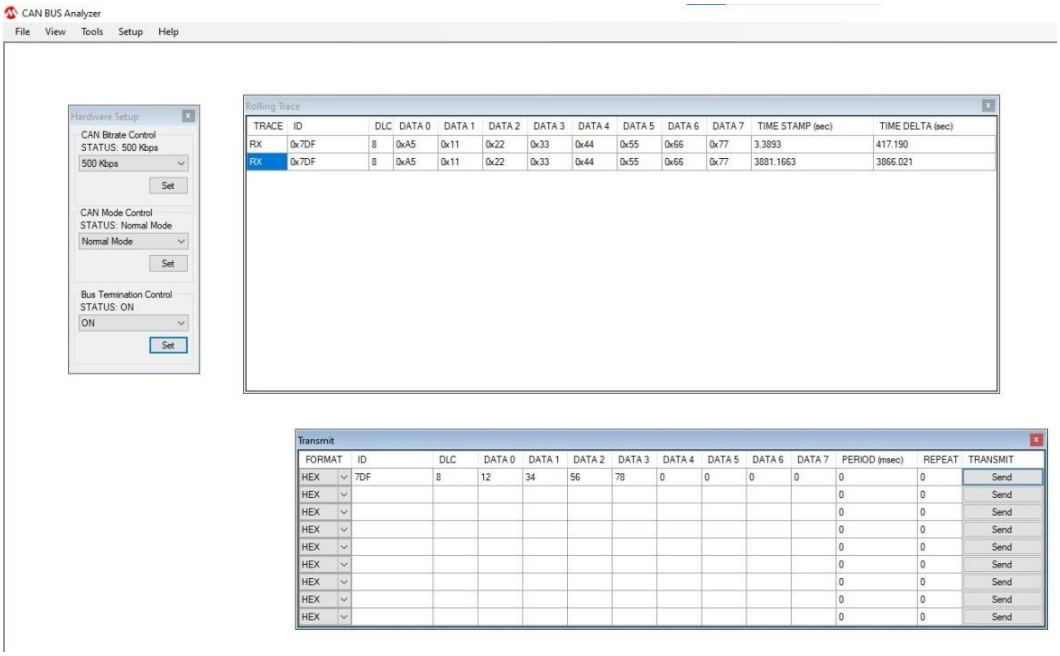

Figure 24 : CAN Analyzer: Data and CAN Frame

The PID engine speed / RPM used for identifying the Engine ignition ON and OFF
The PID is service 01 in UDS CAN frame

0C	12	2	Engine speed	0	16,383.75	rpm	$\frac{256A+B}{4}$

The PID is tested the with following Manufactures:
1. Toyota.
2. Hyundai.
3. Maruti Suzuki.
4. Tata.

Following are the simulated can frame
To switch on the load, send CAN with following message and ID
CAN ID: 0x7E8
CAN DLC: 8
CAN DATA 0.7: 0x04, 0x41,0x0C,0x02,0x00,0x00,0x00,0x00
To switch off the load send CAN with following message and ID
CAN ID: 0x7E8
CAN DLC: 8
CAN DATA 0.7: 0x04, 0x41,0x0C,0x00,0x00,0x00,0x00,0x00

Continues Frame Capturing in CLI Mode

1. Configure the network and system parameters with “Save Data” options
2. Click “Save CLI Mode file” and select the path to save .cli file. Refer: Figure no 51
3. Execute Visualizer in command line mode with below commands:

   SRIR16V2NMCU1_RZA2M_visualizer.exe cli_file.cli “Script.txt" –c

Notice the files being saved in given path in “*.cli” file

S32G Application:

1. This application is used to receive interrupt and send various ack to mcu (from s32g)
2. copy the service file mcu_s32g_ctrl.service to /etc/systemd/system/
3. copy the application mcu_s32g_ctrl to /usr/bin/ and provide exec permissions
4. enable service and reboot
   #systemctl enable mcu_s32g_ctrl.service
5. The application will send the acknowledgments periodically according to the LTE and GPS health status and waits for Shutdown interrupt from the MCU.

API Library

EEPROM

- Open Device

```
/*
 * open eeprom device,
 * return -ve on error, 0 on success
 */
```

int eeprom_open(void);

- Close Device

```
/*
 * close eeprom device
 */
```

void eeprom_close(void);

- Read bytes

```
/*
 * read bytes from EEPROM device
 * offset : address offset
 * buf : buffer pointer
 * len : number of bytes to read
 * returns number of bytes red, -ve on error
 */
```

int eeprom_read(unsigned short offset, unsigned char *buf, unsigned int len);

- Read Mac addresss

```
/*
 * read bytes from EEPROM device
 * buf : buffer pointer
 * returns number of bytes red, -ve on error
 */
```

int eeprom_read_mac(unsigned char *buf);

- Write bytes

```
/*
 * eeprom write
 * offset : address offset
 * buf : buffer pointer
 * len : number of bytes to write
 * returns number of bytes written, -ve on error
 */
```

int eeprom_write(unsigned short offset, unsigned char *buf, unsigned int len);

CAN

- Socket close

```
/*
 * close can device
 */
void can_socket_close(void);
```

- Socket Create

```
/*
 * opens and inittializes can device
 * returns -ve on error , 0 on success
 */
int  can_socket_create(void);
```

- Read

```
/*
 * read from can device
 * buf : buffer pointer to fill data
 * returns number of bytes red , -ve on error
 */
int can_read(unsigned char *buf);
```

- Write

```
/*
 * write to can device
 * buf : buffer pointer with data to write (max 8 bytes)
 * returns number of bytes written , -ve on error
 */
int can_write(unsigned char *buf, int size);
```

RTC

- Set Time

```
/*
 * Will set new date and time provided
 * return 0 on success , -ve on fail
 */
int rtc_set_time(struct rtc_info *info);
```

- Get Time

```
/*
 * Will fill structure with current date and time
 * return 0 on success , -ve on fail
 */
int rtc_get_time(struct rtc_info *info);
```

IMU

- Open Device

```
int imu_open (void);
```

- Close Device

int imu_close (void);

- Read raw data

int imu_read_accel_data_raw(void *data);
int imu_read_mag_data_raw(void *data);

Testing apis :

- Testapp will be provided along with APIs library. The procedure to test APIs is as follows:

EEPROM:

- EEPROM read and write can be tested by following commands:
  ./test_app eeprom w/r offset buffer

  Arguments for testing eeprom :
  1. w/r: select w for write and r for read
  2. offset: give the Offset to write to or read from in Hex values.
  3. Buffer: buffer to be written on the offset (Needed only for write)

  Example:
  ./test_app eeprom w 0x40 hello -> for writing to eeprom module
  ./test_app eeprom r 0x40 -> for rtc module data
- Read MAC Address from EEPROM using following command:

./test_app MAC

RTC

- RTC set time and get time can be tested by using following command:
  ./Testapp rtc w/r sec min hour day month year
  Arguments for testing RTC:
  1. w/r: select w to set time and r to get time and date
  2. set the time and date in the following order (Needed only for write):

  Sec Min Hour Day Month year
  Example :
  ./test_app rtc w 40 30 19 25 08 2022 -> for writing to rtc module
  ./test_app rtc r -> for reading rtc module data

CAN

- Can read can be tested by following commands:

./Testapp can r

IMU

- IMU data can be read by using th following command:

./Testapp imu r

Applying Patch

EEPROM

- Copy the patch to this[s32g/trunk/test_app/moter/apps] folder and run the below command

$svn patch eeprom.patch

- Copy the patch to this[s32g/trunk/bsp_source_code/kernel/linux-bsp31.0-5.10.410-rt/linux/drivers/misc/eeprom] folder and run the below command

$svn patch eeprom_addr_block.patch

9. Linux Test application

Ethernet Test:

Read the ethernet ip address by ifconfig as below

```
root@s32g274ardb2:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.20.1.143  netmask 255.255.255.0  broadcast 10.20.1.255
        inet6 fe80::3eb5:725:e11d:7e15  prefixlen 64  scopeid 0x20<link>
        ether 00:11:12:13:14:15  txqueuelen 1000  (Ethernet)
        RX packets 2342  bytes 202010 (197.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 37  bytes 4338 (4.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 63  base 0x6000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 25 : Reading Ethernet IP

MCU test:

Execute mcu_s32g_ctrl : ./mcu_s32g_ctrl

```
root@s32g274ardb2:~# ./mcu_s32g_ctrl

Opening Drive[  296.908022] mcu_dev File Opened...!!!
r[  296.908194] GPIO_irqNumber = 93
 [  296.908395] mcu_dev read..blocked
NOW

 read blocked
press any key to continue
1
send LTE_ACK
[  304.624952] mcu_dev IOCTL..
[  304.624978] mcu_dev: Executing Workqueue Function
press any key to continue
1
send GPS_ACK
press any key to continue
[  308.488909] mcu_dev IOCTL..
[  308.488955] mcu_dev: Executing Workqueue Function
1
send POWERON_ACK
press any key to continue
[  310.880901] mcu_dev IOCTL..
[  310.880943] mcu_dev: Executing Workqueue Function
1
send GPS_REC_ACK
press any key to continue
[  312.536892] mcu_dev IOCTL..
[  312.536931] mcu_dev: Executing Workqueue Function
1
send LTE_REC_ACK
[  314.284886] mcu_dev IOCTL..
[  314.285538] mcu_dev: Executing Workqueue Function
1

^C
root@s32g274ardb2:~# [  319.436862] mcu_dev read..un-blocked
[  319.437220] mcu_dev File Closed...!!!
^C
```

Figure 26 : MCU Test

USB Read/Write test:

1. Check if SD card is detecting:
   fdisk –l
2. Make a directory in /mnt
   mkdir /mnt/usb
3. mount /dev/sd1 /mnt/usb
4. Copy the test.txt to mounted USB device
   cp $(pwd)/test.txt /mnt/usb
5. cat /mnt/usb/test.txt
6. umount /mnt/usb

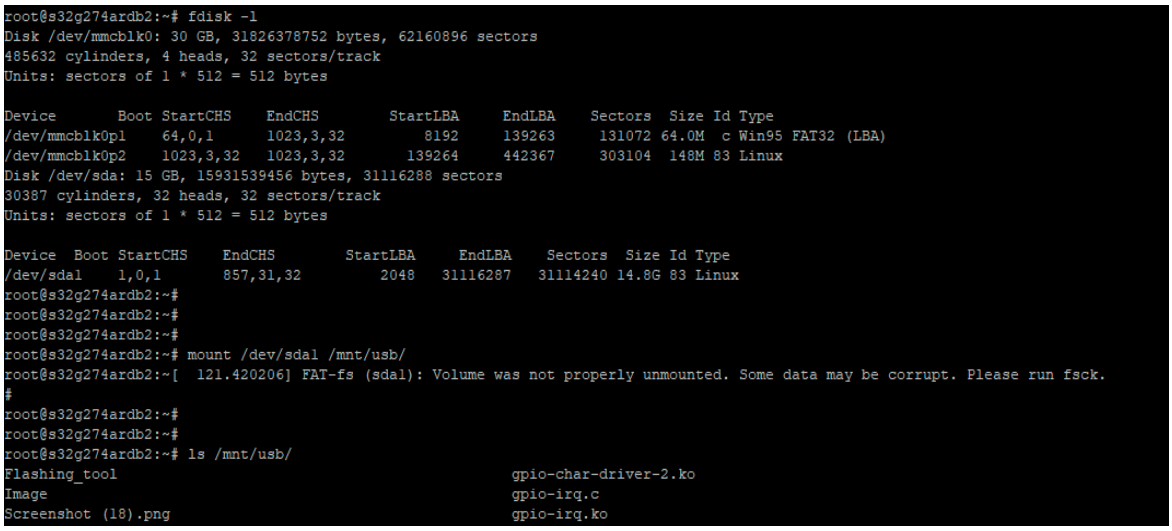

Figure 27 : MCU Test

CAN test:

Give the commands as below:

=>ifconfig can0 down

=>ip link set can0 type can bitrate 125000

=>ifconfig can0 up

Open microchip CAN bus analyser on windows PC and set hardware setup as shown below

- Set CAN Bit-rate control to 125Kbps.
- Set CAN Mode control to “Normal mode”.

- Set CAN Bus Termination status as “ON” condition.

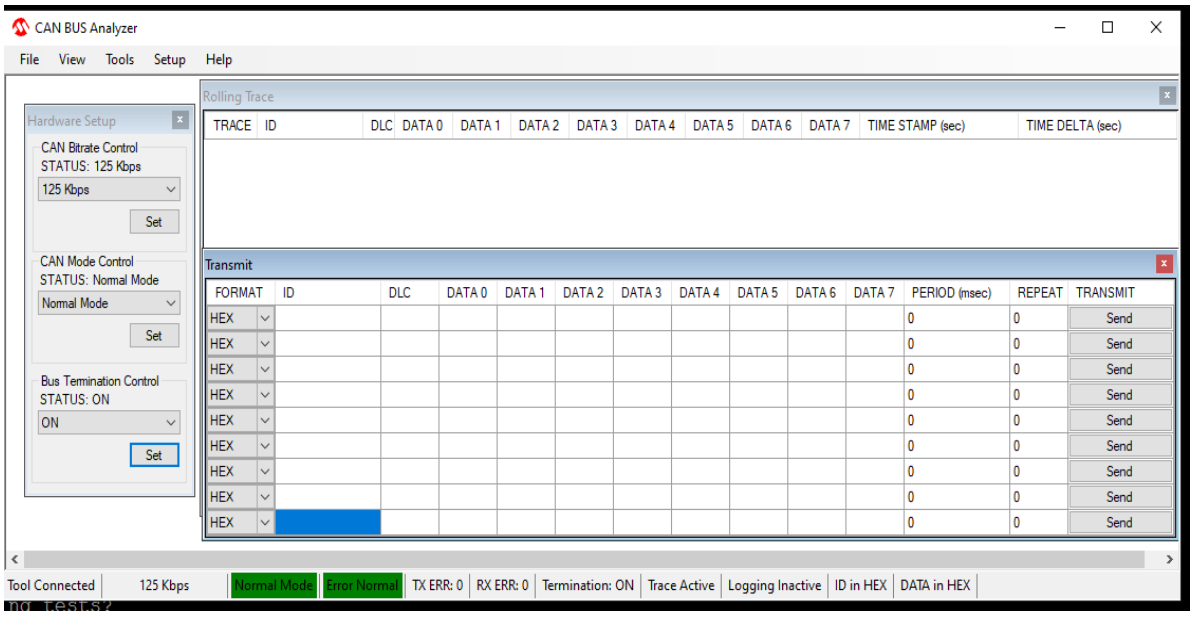

Figure 28 : Microchip CAN bus Analyser

=> cansend can0 5A1#11.22.33.44.55.66.77.99
=> candump -T 30000 can0
Test is passed if user is able to send data from device to CAN bus analyser as shown below

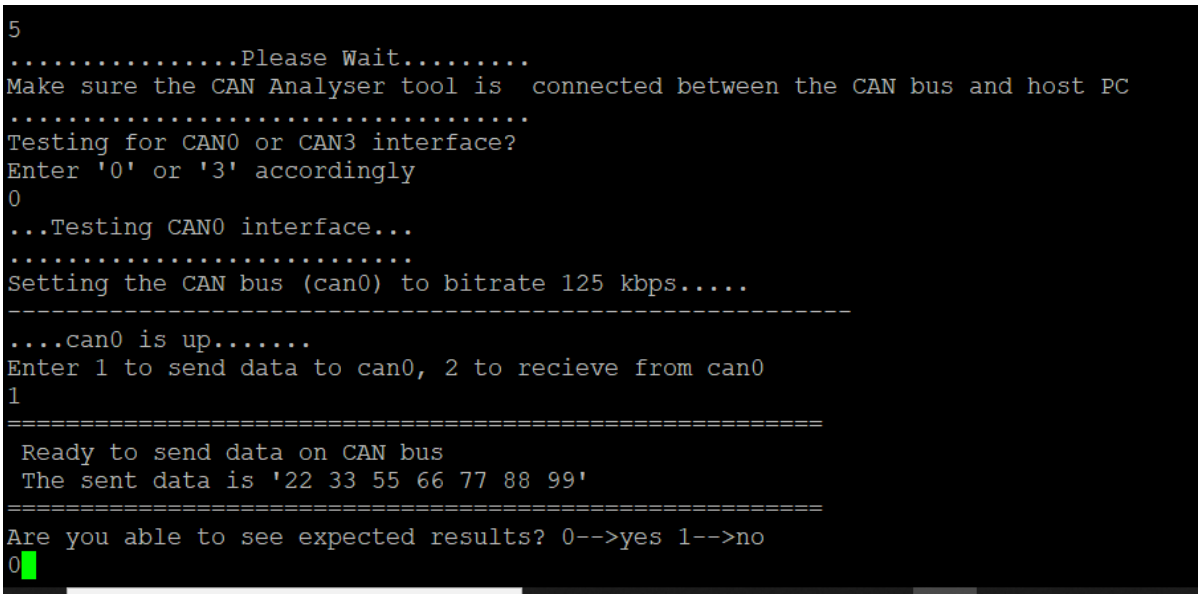

Figure 29 : CAN transmitter (device)

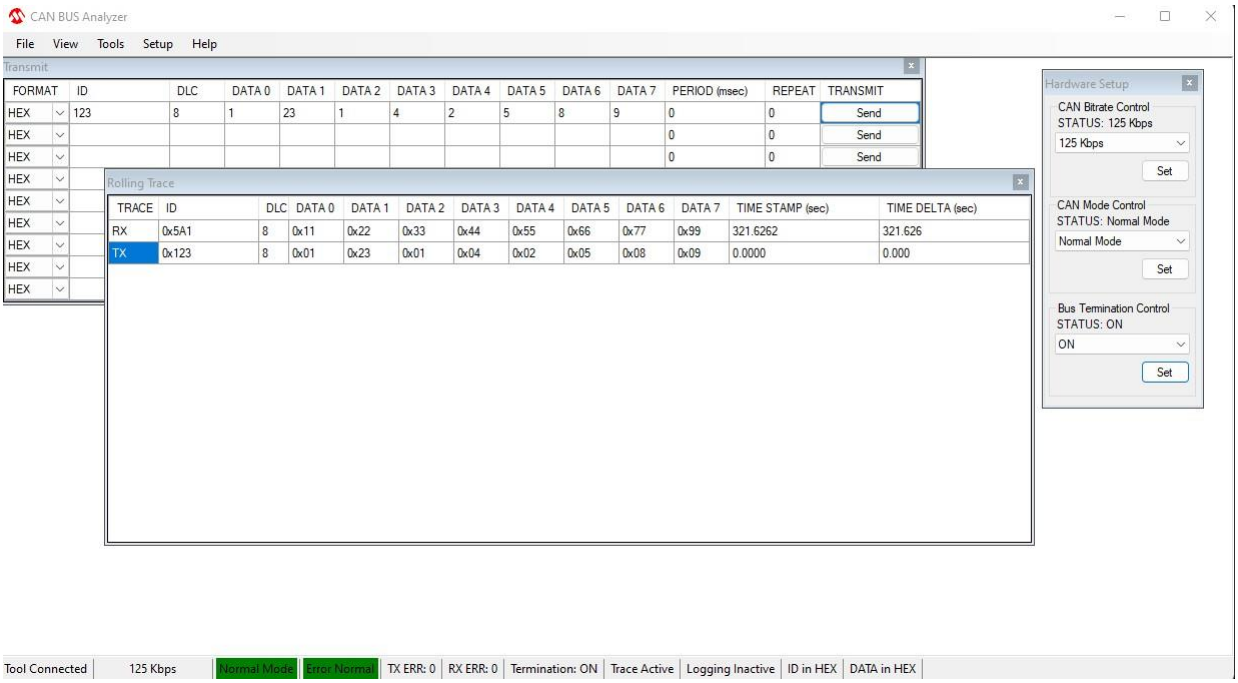

Figure 30 : CAN receiver and transmitter (Microchip CAN bus)

Test is passed if user is able to receive data from device to CAN bus analyser on windows to the device as shown below

```
5
.............Please Wait.........
Make sure the CAN Analyser tool is  connected between the CAN bus and host PC
.....................................
Testing for CAN0 or CAN3 interface?
Enter '0' or '3' accordingly
0
...Testing CAN0 interface...
.............................
Setting the CAN bus (can0) to bitrate 125 kbps.....
------------------------------------------------------------
....can0 is up.......
Enter 1 to send data to can0, 2 to recieve from can0
2
============================================================
Ready to receive data on CAN bus
============================================================
  can0  012   [2]  55 66
root@s32g274ardb2:~#
```

Figure 31 : CAN receiver(device)

EEPROM test:

Read and write the eeprom node as below:

echo "TEST" > /sys/bus/i2c/devices/0-0050/eeprom
cat /sys/bus/i2c/devices/0-0050/eeprom

Figure 32 : EEPROM test

IMU Test:

Read the raw accel and gyro data below:

cat /sys/bus/i2c/devices/0-001e/iio:device2/in_accel_x_raw
cat /sys/bus/i2c/devices/0-001e/iio:device2/in_magn_x_raw

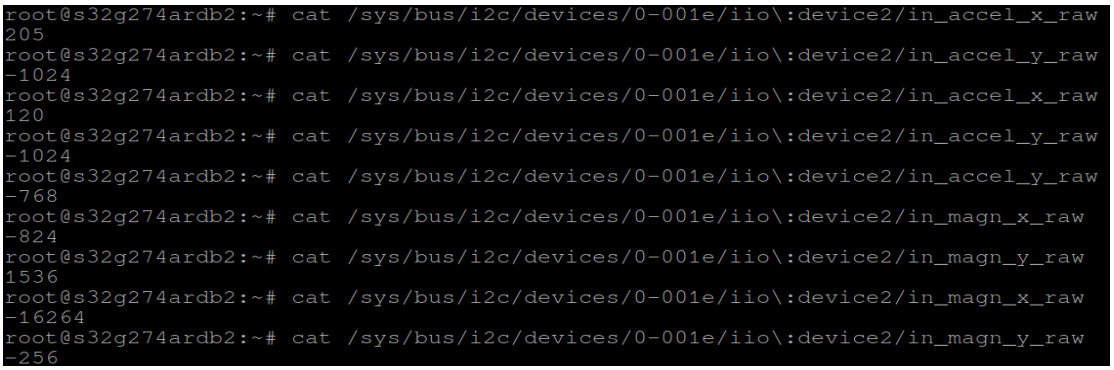

Figure 33 : IMU test

RTC1 test:

Follow the below commands:

=>cd /dev

=>rm rtc

=>ln -s rtc1 rtc

=>ls -l /dev/rtc*

=>date -R xxxxxxxxxxxx

=>hwclock -w -f /dev/rtc1

=>hwclock -r -f /dev/rtc1

=>i2cdump -f -y -a 2 0x51

```
root@s32g274ardb2:~# cd /dev/
root@s32g274ardb2:/dev#
root@s32g274ardb2:/dev# rm rtc
root@s32g274ardb2:/dev#
root@s32g274ardb2:/dev# ln -s rtc1 rtc
root@s32g274ardb2:/dev#
root@s32g274ardb2:/dev# ls -l /dev/rtc*
lrwxrwxrwx    1 root     root             4 Jan  1 01:06 /dev/rtc -> rtc1
crw-------    1 root     root      253,   0 Jan  1  1970 /dev/rtc0
crw-------    1 root     root      253,   1 Jan  1  1970 /dev/rtc1
root@s32g274ardb2:/dev#
root@s32g274ardb2:/dev# date -R 010112122022
Sat, 01 Jan 2022 12:12:00 +0000
root@s32g274ardb2:/dev#
root@s32g274ardb2:/dev# hwclock -w
root@s32g274ardb2:/dev#
root@s32g274ardb2:/dev# hwclock -r -f /dev/rtc1
Sat Jan  1 12:12:29 2022  0.000000 seconds
```

Figure 34 : RTC1 test

WiFi Test:

Follow the below commands in sequence :

=> ifconfig wlan0 up

=>vi /etc/wpa_supplicant.conf

=>add wifi ssid and password as below,

```
   network=
{
   scan_ssid=1
   ssid="Wifi_ssid_NAME"
   psk="password"
}
```

=> sudo wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant.conf -D nl80211

=>ifconfig wlan0

=>udhcpc -i wlan0

=>ping 8.8.8.8

```
root@s32g274ardb2:~# ./wifi_test.sh Lakshmi Shanua@1998
make sure iwlwifi-cc-a0-46.ucode  file available in /lib/firmware/
enabling wifi wlan0....
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
[  453.886545] wlan0: authenticate with d6:63:90:e3:69:21
[  453.888699] wlan0: send auth to d6:63:90:e3:69:21 (try 1/3)
[  453.915291] wlan0: authenticated
[  453.916828] wlan0: associate with d6:63:90:e3:69:21 (try 1/3)
[  453.947869] wlan0: RX AssocResp from d6:63:90:e3:69:21 (capab=0x1431 status=0 aid=1)
[  453.950280] wlan0: associated
[  453.999917] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
udhcpc: started, v1.32.0
udhcpc: sending discover
udhcpc: sending select for 192.168.116.223
udhcpc: lease of 192.168.116.223 obtained, lease time 3599
/etc/udhcpc.d/50default: Adding DNS 192.168.116.160
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
  [  456.687237] s32cc-dwmac 4033c000.ethernet eth0: Link is Down
  [  456.691251] s32cc-dwmac 4033c000.ethernet eth0: Timeout accessing MAC_VLAN_Tag_Filter
 [  456.691275] s32cc-dwmac 4033c000.ethernet eth0: failed to kill vid 0081/0
   inet 10.20.1.143  netmask 255.255.255.0  broadcast 10.20.1.255
        inet6 fe80::3eb5:725:e11d:7e15  prefixlen 64  scopeid 0x20<link>
        ether 00:11:12:13:14:15  txqueuelen 1000  (Ethernet)
        RX packets 33578  bytes 2883032 (2.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 2530 (2.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 63  base 0x6000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.116.223  netmask 255.255.255.0  broadcast 192.168.116.255
        inet6 2409:4071:6e17:64cd:8238:fbff:fe30:5f9d  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::8238:fbff:fe30:5f9d  prefixlen 64  scopeid 0x20<link>
        ether 80:38:fb:30:5f:9d  txqueuelen 1000  (Ethernet)
        RX packets 5  bytes 1128 (1.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 1754 (1.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 35 : WiFi test

```
root@s32g274ardb2:~# echo "nameserver 8.8.8.8" > /etc/resolv.conf
root@s32g274ardb2:~#
root@s32g274ardb2:~#
root@s32g274ardb2:~# ping google.com -I wlan0
PING google.com(bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e)) from 2409:4071:6e17:64cd:8238:fbff:fe30:5f9d wlan0: 56 data bytes
64 bytes from bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e): icmp_seq=1 ttl=58 time=82.5 ms
64 bytes from bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e): icmp_seq=2 ttl=58 time=74.4 ms
64 bytes from bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e): icmp_seq=3 ttl=58 time=75.3 ms
64 bytes from bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e): icmp_seq=4 ttl=58 time=77.0 ms
64 bytes from bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e): icmp_seq=5 ttl=58 time=73.0 ms
64 bytes from bom07s45-in-x0e.1e100.net (2404:6800:4009:832::200e): icmp_seq=6 ttl=58 time=81.5 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5006ms
rtt min/avg/max/mdev = 73.016/77.276/82.521/3.543 ms
```

Figure 36 : WiFi ping test

4G LTE Test:

Follow the below procedures:

=>Make sure the following files are added in /etc/ppp/peers

quectel-chat-connect
quectel-chat-disconnect
quectel-ppp

=>Add your APN in quectel-chat-connect file
AT+CGDCONT=1,"IP","web.vodafone.in","0.0.0.0",0,0 //APN for vodafone

=> Add the required username and password in quectel-ppp

=> sudo echo "2c7c 030a" > /sys/bus/usb-serial/drivers/option1/new_id

=> Check in /dev/ and Add the required port for QUECTEL GSM in quectel-ppp
/dev/ttyUSB1

=> pppd call quectel-ppp

=> Check if ppp0 is created :
ifconfig -a

Figure 37 : 4G LTE test

```
root@ubuntu-s32g274ardb2:/home/bluebox# ping 8.8.8.8 -I ppp0
PING 8.8.8.8 (8.8.8.8) from 100.78.93.209 ppp0: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=59 time=195 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=59 time=39.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=59 time=22.9 ms
^C
```

Figure 38 : 4G LTE ping test

GPS Test:

Follow the below command:

=> cat /dev/ttyLF2

```
bluebox@ubuntu-s32g274ardb2:~$ cat /dev/ttyLF2
cat: /dev/ttyLF2: Permission denied
bluebox@ubuntu-s32g274ardb2:~$ sudo cat /dev/ttyLF2
$GNRMC,,V,,,,,,,,,N*4D

$GNVTG,,,,,,,,,N*2E

$GNGGA,,,,,,0,00,99.99,,,,,,*56

$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E

$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E

$GPGSV,1,1,00*79

$GLGSV,1,1,00*65

$GNGLL,,,,,,,V,N*7A

$GNRMC,,V,,,,,,,,,N*4D

$GNVTG,,,,,,,,,N*2E

$GNGGA,,,,,,0,00,99.99,,,,,,*56

$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E

$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E

$GPGSV,1,1,00*79

$GLGSV,1,1,00*65
```

Figure 39 : GPS test

FCC Compliance Statement (USA)

FCC ID: 2A6PTMT2V1001

Compliance Statements: This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference.
2. This device must accept any interference received, including, an interference that may cause undesired operation.

Caution Statements:

• Any changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate this equipment.

• This equipment should be installed and operated with a minimum distance of 20 cm between the radiator and your body.

INFORMATION TO THE USER

For Class A and Class B digital devices, information to the user is required to include the following statements (Section 15.105):

For a Class A digital device or peripheral, the instructions furnished to the user shall include the following or similar statement, placed in a prominent location in the text of the manual:

NOTE: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

For a Class B digital device or peripheral, the instructions furnished to the user shall include the following or similar statement, placed in a prominent location in the text of the manual:

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

END OF DOCUMENT