VK-CMG2LC HMI How To

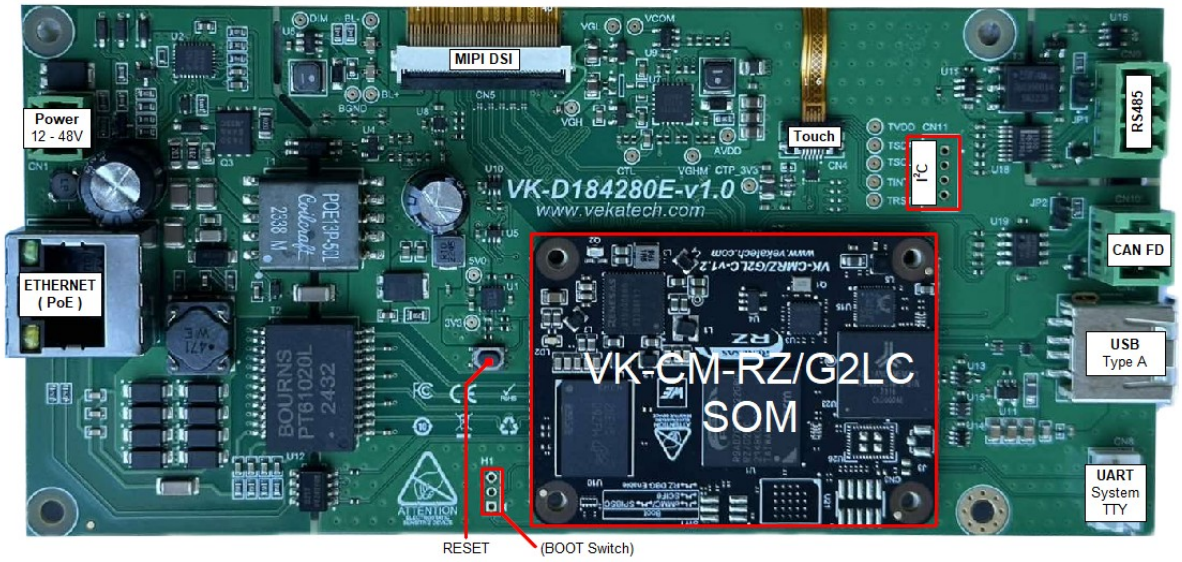
VK-CMG2LC v1.0 HMI Carrier Board

Content:

1. Introduction

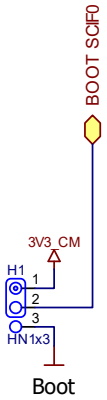
VK-CMG2LC is **H**uman **M**achine **I**nterface carrier board for VK-CM-RZ/G2LC **S**ystem **o**n **M**odule. It is based on Renesas **R9A07G044C22GBG, Dual ARM Cortex-A55 + Cortex-M33 MCU**. The main purpose of this manual is to show how to get started with the module onboard.

1.1 Connectors



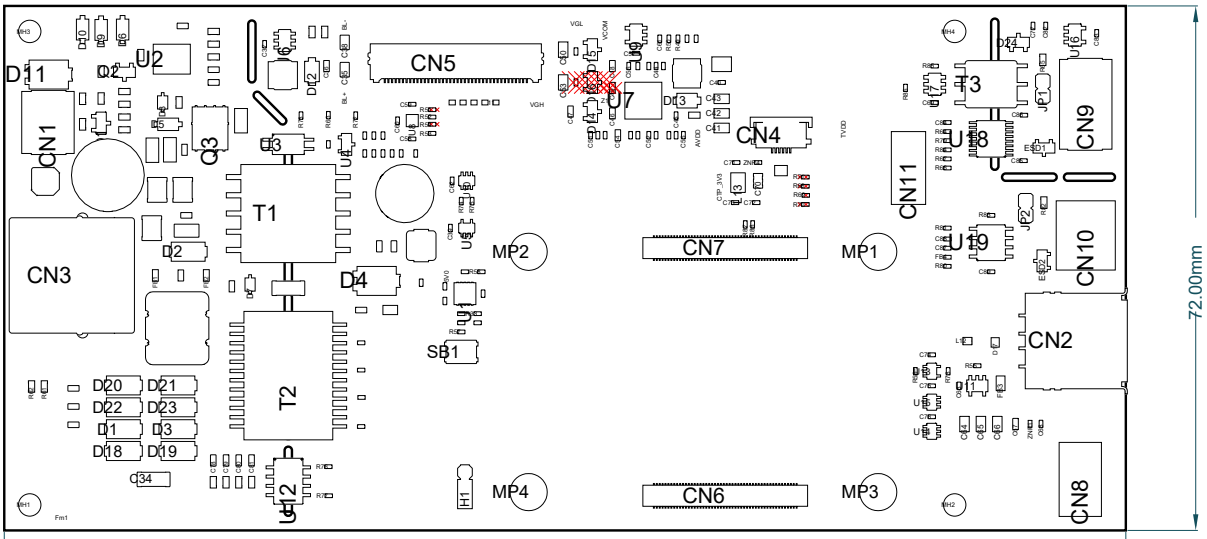
Connectors & Signals

1.2 Boot switch



- **H 1** → Controls the SOM boot source: **1-2** from **SCIF0** / **NC** from **eMMC**.

1.3 Dimensions



Board size

2. Power Up

The board can be powered from 2 sources:

- POWER connector (24 – 48 V)
- PoE

In idle state (without plugged periphery) the board consumes ~ **300 mA**. The consumption however, can jump up to ~ **1,2 A** (if Camera, Display, Ethernet, USB Flash, USB Keyboard, Audio jack & SD card are plugged in).

3. Install U-boot (v2021.10)

You can get the U-boot firmware from our site or compile it yourself from our repository, if you prefer manually building it from scratch.

- Connect **VK-CMG2LC** to the PC (through **USB ↔ UART converter**) & see what COM port is assigned by the OS in the Device Manager.
- Place a jumper in position 1-2 of the **H1** to boot from the **SCIF0**.
- Download **vkPyFlasher** rescue tool in folder of your choice.
- Download the U-boot **firmware** and unpack it in: **vkPyFlasher/images**.

3.1 into SPI Flash

- Run the following commands:

```
cd vkPyFlasher.
flash_boot.py --board=vk-d184280e --serial_port=COM<n> --qspi.
```

- Press **reset** button on the **VK-CMG2LC**
- Wait the flashing to complete.

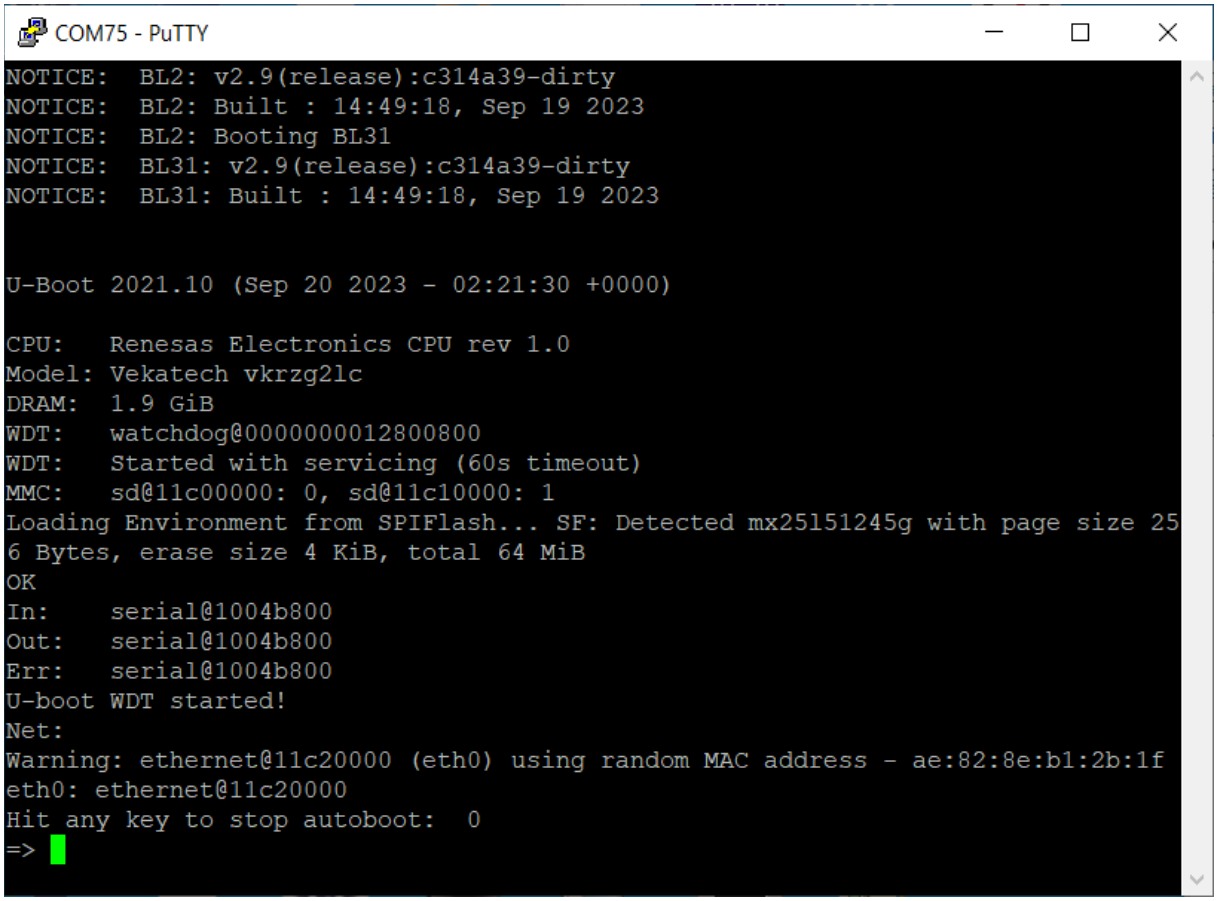
3.2 into eMMC SSD

- Run the following commands:

```
cd vkPyFlasher.
flash_boot.py --board=vk-d184280e --serial_port=COM<n>.
```

- Press **reset** button on the **VK-CMG2LC**
- Wait the flashing to complete.

After download is ready, remove the jumper from **H1** to boot from **eMMC** (by default) and press **reset** => you should now be able to see the boot log of the U-boot.



U-boot log

4. Boot logic

Understanding boot logic is crucial. As you see there are **2** copies of **U-boot** (1 in eMMC & 1 in SPI). Environment parameters of the ones are slightly different from the others. They are configured in such way, that when **H1** is set to boot from eMMC (**no jumper**) it's U-boot searches Linux image in the same that **eMMC**, but when H1 is set to boot from SPI (**1-2**) it's U-boot searches Linux image in **µSD** card. That's why default U-boot environment parameters differs on purpose and you don't need to edit them every time you want to boot from one or other media (you just set H1). That actually explains why booting from SPI, leads to booting from µSD. You can always change those parameters & boot from wherever you want.

5. Install Linux (Kernel v5.10.184)

Now that you have 2 workable U-boot instances, up & running, it is time to load something bigger. You can get Debian Linux image from our site or compile Yocto yourself from our repository, (if you prefer building it from scratch):

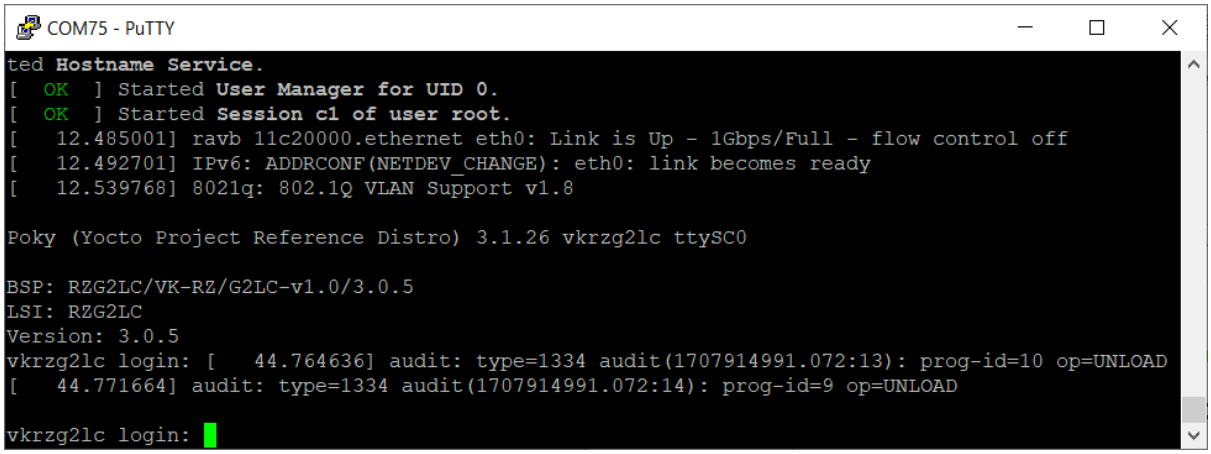
5.1 into eMMC SSD → Yocto v3.1.26 (Dunfell)

Get the **vkPyFlasher** prepared:

- Connect **VK-CMG2LC** to the PC (through **USB ↔ UART converter**) & see what COM port is assigned by the OS in the Device Manager.
- Connect **VK-CMG2LC** to the PC (through the **USB Type A**)
- Remove any jumper from **H1** to boot from the **eMMC**.
- Download the desired Linux image (Yocto / Debian), and place it at the location: **vkPyFlasher/images/vk-d184280e**.

Flash the image:

- `cd vkPyFlasher`.
- In case you want Yocto:
  `flash_img.py --board=vk-d184280e --serial_port=COM<n> --image_rootfs=core-image-weston-vk-d184280e.simg`.
- In case you want Debian:
  `flash_img.py --board=vk-d184280e --serial_port=COM<n> --image_rootfs=debian-bookworm-vk-d184280e.simg`.
- Press **reset** button on the **VK-CMG2LC**.
- Wait the flashing to complete.
- Open the COM port assigned by the OS for the VK-CMG2LC with (115200|B|N|1) settings.
- Press **reset** once again.
- Login to Yocto (user: **root**) or Debian (user: **vkrz** & password: **vkrzg2lc**).
- In case of Debian, you can setup the partition on the fly:
  → Extend the root partition to use the full capacity of the eMMC:
  `sudo growpart /dev/mmcblk0 2 && sudo resize2fs /dev/mmcblk0p2`.
- In case of Yocto, you should now be able to see login screen:



Yocto Login screen

6. Launch the installed Linux images

Thanks to the convenient boot logic, launching is easy, it depends on correct setting of **H1**

- To boot Linux from µSD: place a jumper to **SPIBSC (2-3)**.
- To boot Linux from eMMC: remove the jumper from **eMMC (1-2)**.
- To boot Linux from Ethernet: set the switch to **SPIBSC** and make sure µSD slot is Empty. When U-boot can't find SD card, it will try to boot from its **serverip**, **tftpdir** & **netrootfs** environment parameters. (serverip is the address, where U-boot will look for NFS & TFTP servers, tftpdir shows where the boot directory is on the server and netrootfs → where root directory is on the server) Same server can be used to boot multiple boards, every board can have its own boot and root directories and that's why tftpdir and netrootfs parameters should be filled with the correct paths on the server.

7. Apply Overlays

Overlays are way to tell Linux to use or not a given periphery. They can even point a specific hardware for a same periphery and form configurations for different version of a board. For example in version 1, the board can use **Realtek** audio driver, in other **someone's else**, in third no audio at all. Management of the overlays is done through **uEnv.txt**, located in **/boot** directory of the particular linux image. This file is analyzed during boot of the Linux image, so every change in that file takes effect after boot. To apply overlay, you need to line it up in the following row:

```
fdt_extra_overlays=
```

All available overlays that can be applied are listed in **/boot/overlays**, but simultaneously only these can be activated at the same time.

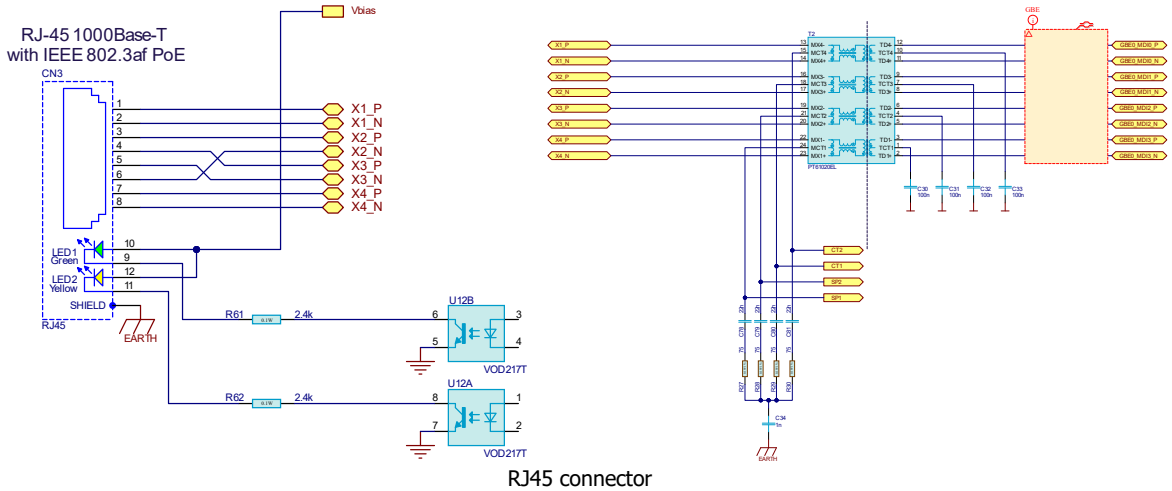
- vko-cm33.dtbo
- vko-i2c2.dtbo
- vko-rs485.dtbo
- vko-can.dtbo
- vko-qspi-mx25l51245g.dtbo
- vko-udma.dtbo

8. Use various periphery in Linux

To demonstrate using different periphery, we are going to use the Debian image, for other images (Yocto for example) the commands could be not available or could be slightly different, so don't worry if some of them do not work, just google how to do it for your particular distribution.

8.1 Ethernet

Ethernet is enabled by default and does not need overlay. Plug the **cat** cable into **RJ45** connector and in a couple of seconds the board should get an **IP** from the local **DHCP** server.

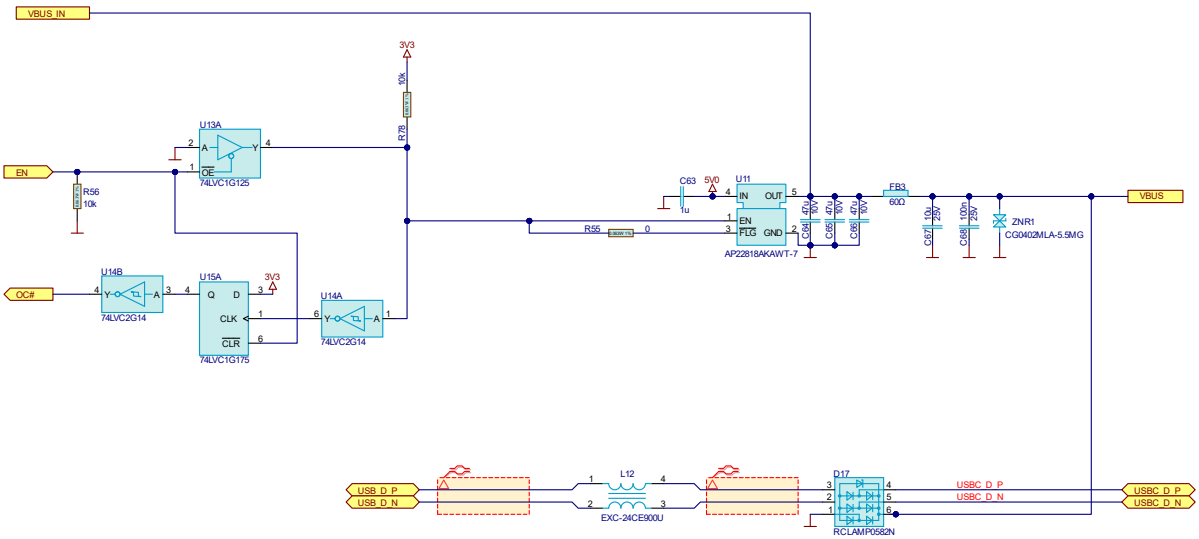


RJ45 connector

- You can check what IP is assigned: `sudo ifconfig`.
- You can probe if Host is reachable: `ping vekatech.com`.

8.2 USB

USB is enabled by default and does not need overlay. To test it, just plug a device such as: Mouse, Keyboard, USB Flash, USB Camera & whatever other USB device you can think of.



USB Type A connector

- Mouse: It is Plug & Play device, you don't need to do anything.
- Keyboard: Also Plug & Play device, you don't need to do anything.
- Flash Drive: most of the latest Linux distributions mounts the drive automatically, (Debian 12 is not an exception), so it is again another Plug & Play device.
- Camera: usually you need software to see the stream, the example below is with VLC, but ffmpeg or others are also a great choice.

  → Open VLC through the GUI: go to **Start/Sound & Video/VLC media player**.

  → Go to **Media** → **Open Capture Device …** → **Capture Devie** Tab

  for **Video device name** select **/dev/video0** and press **Play**.

  → You should now be able to see the USB video stream.
- GSM Modem: Also Plug & Play device, but to set a connection it needs some setup:

→ Make sure `sudo apt-get install network-manager modemmanager` are installed

→ Start their services: `systemctl start NetworkManager ModemManager`.

→ If autostart on boot is a must: `systemctl enable NetworkManager ModemManager`.

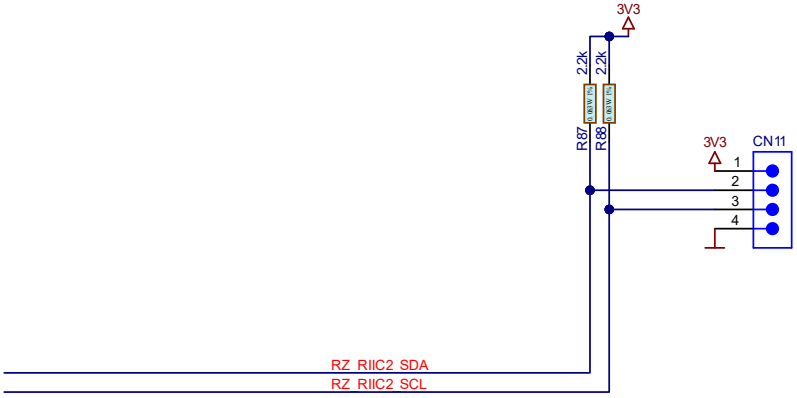
→ Configure GSM connection : `sudo nmcli connection add type gsm ifname '*' con-name '<name>' apn '<AP>' connection.autoconnect yes`.

→ A new network interface, type **gsm** should appear: `nmcli device status`.

8.3 I2C

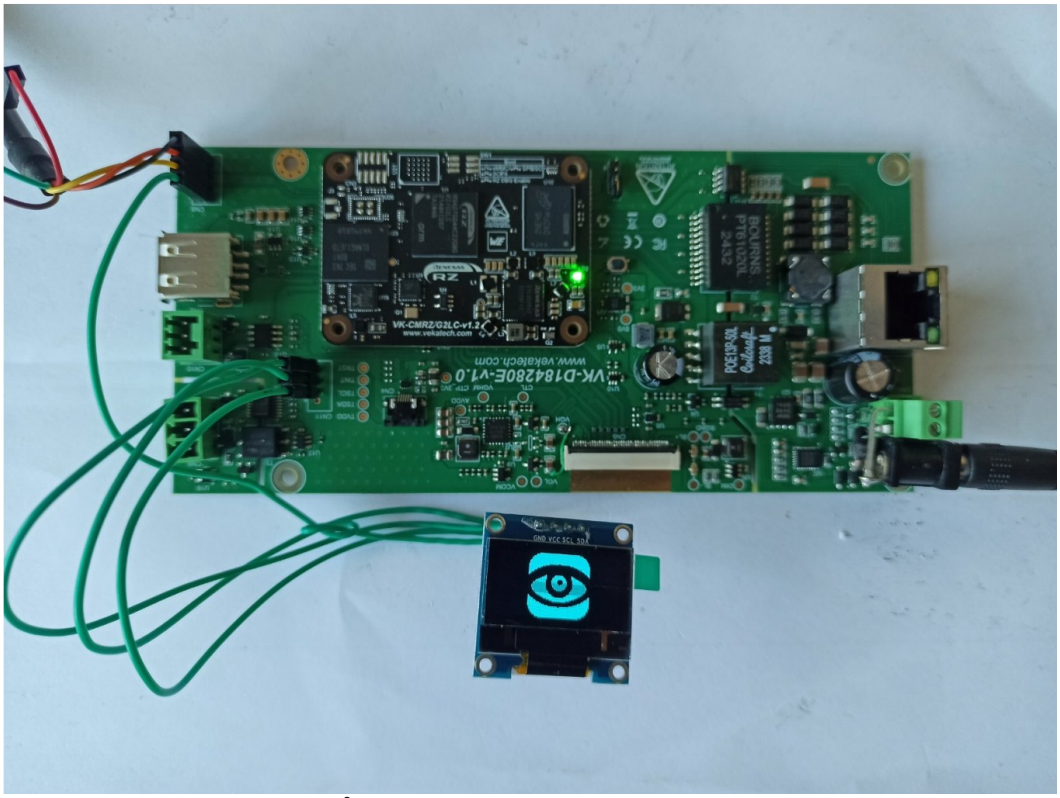
Make sure you are applied **vko-i2c2.dtbo** in **/boot/uEnv.txt**.

```
fdt_extra_overlays=vko-i2c2.dtbo.
```



I²C connector

I²C[2] pins go to Pin_2 (**SDA**) & Pin_3 (**SCL**) of CN11 connector.



I²C device (SSD1306 based OLED display)

- Install I²C software: `sudo apt-get install i2c-tools`.
- Now we can see all I²C devices on the bus 2: `sudo i2cdetect -y -a -r 2`.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

- Device **0x3C** is happen to be an OLED **SSD1306** based display, so to be controlled, an auxiliary python module is needed to be installed on the system, so called `smbus2`.
  `sudo apt-get install python3-smbus2`.
- Create a new python file and fill it with the following script:
  `nano ~/SBC-OLED01.py`.

```
from smbus2 import SMBus

# Constants
I2C_ADDRESS = 0x3C
I2C_BUS = 2

# OLED commands
COMMAND_MODE = 0x00
DATA_MODE = 0x40

Logo = bytearray(b'\x00\x00\x80\xe0\xf0\xf8\xf8\xfc\xfe\xfe\xfe\xff\xff\xff\xff\xff' + \
                 b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff' + \
                 b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff' + \
                 b'\xff\xff\xff\xff\xff\xfe\xfe\xfe\xfc\xf8\xf8\xf0\xe0\x80\x00\x00' + \
                 b'\xf8\xff\xff\xff\xff\xff\xff\xff\xff\x7f\x7f\x7f\x3f\x3f\x1f\x1f' + \
                 b'\x1f\x0f\x0f\x0f\x0f\x07\x07\x07\x07\x07\x07\x07\x07\x07\x07\x03' + \
                 b'\x03\x07\x07\x07\x07\x07\x07\x07\x07\x07\x07\x0f\x0f\x0f\x0f\x1f' + \
                 b'\x1f\x1f\x3f\x3f\x7f\x7f\x7f\xff\xff\xff\xff\xff\xff\xff\xff\xf8' + \
                 b'\x3f\x1f\x0f\x07\x07\x03\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00' + \
                 b'\x80\x80\xc0\x00\x00\x00\x00\x00\x00\x00\x00\x80\xc0\xc0\xc0\xc0' + \
                 b'\xc0\xc0\xc0\xc0\x80\x00\x00\x00\x00\x00\x00\x00\x00\xc0\x80\x80' + \
                 b'\x00\x00\x00\x00\x00\x00\x00\x00\x01\x01\x03\x07\x07\x0f\x1f\x3f' + \
                 b'\x00\x00\x00\x00\x80\xe0\xe0\xf0\xf8\xf8\xfc\xfc\xfe\xfe\xff\xff' + \
                 b'\xff\xff\x03\x00\x00\x00\x00\x00\xf8\xfe\xff\xff\xff\xff\xc7\x83' + \
                 b'\x03\x83\x83\xc7\xff\xff\xfe\xf8\x00\x00\x00\x00\x00\x03\xff\xff' + \
```

```
                b'\xff\xff\xfe\xfe\xfc\xfc\xf8\xf8\xf0\xe0\xe0\x80\x00\x00\x00\x00' + \
                b'\x00\x00\x00\x07\x1f\x3f\x7f\xff\xff\xff\xff\xff\xff\xff\xff\xff' + \
                b'\xff\xff\xf8\xe0\xc0\x00\x00\x00\x00\x03\x07\x0f\x0f\x1f\x1f\x1f' + \
                b'\x1f\x1f\x1f\x0f\x0f\x07\x03\x00\x00\x00\x00\xc0\xe0\xf8\xff\xff' + \
                b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\x7f\x3f\x1f\x07\x00\x00\x00' + \
                b'\xfc\xf8\xf0\xe0\xe0\xc0\x80\x80\x01\x01\x03\x03\x07\x07\x0f\x0f' + \
                b'\x1f\x1f\x1f\x3f\x3f\x3f\x7e\x7e\x7c\xf8\xf8\xf0\xf0\xf0\xf0\xf0' + \
                b'\xf0\xf0\xf0\xf0\xf0\xf8\xf8\x7c\x7e\x7e\x7f\x3f\x3f\x1f\x1f\x1f' + \
                b'\x0f\x0f\x07\x07\x03\x03\x01\x01\x80\x80\xc0\xe0\xe0\xf0\xf8\xfc' + \
                b'\x1f\xff\xff\xff\xff\xff\xff\xff\xff\xfe\xfe\xfe\xfc\xfc\xf8\xf8' + \
                b'\xf8\xf0\xf0\xf0\xf0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xc0' + \
                b'\xc0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xe0\xf0\xf0\xf0\xf0\xf8' + \
                b'\xf8\xf8\xfc\xfc\xfe\xfe\xfe\xff\xff\xff\xff\xff\xff\xff\xff\x1f' + \
                b'\x00\x00\x01\x07\x0f\x1f\x1f\x3f\x7f\x7f\x7f\xff\xff\xff\xff\xff' + \
                b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff' + \
                b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff' + \
                b'\xff\xff\xff\xff\xff\x7f\x7f\x7f\x3f\x1f\x1f\x0f\x07\x01\x00\x00')

# Initialize display
def init_display(bus):
    commands = [
        0xAE,  # Display off
        0x20, 0x00,  # Memory addressing mode: Horizontal
        0x40,  # Set display start line
        0xA1,  # Set segment re-map (col address 127 is mapped to SEG0)
        0xC8,  # COM output scan direction (remapped mode)
        0xA6,  # Set normal display
        0xA8, 0x3F,  # Set multiplex ratio (1/64 duty)
        0xD3, 0x00,  # Display offset
        0xD5, 0x80,  # Set display clock divide ratio/oscillator frequency
        0xD9, 0xF1,  # Set pre-charge period
        0xDA, 0x12,  # Set COM pins hardware configuration
        0xDB, 0x20,  # Set VCOMH deselect level
        0x8D, 0x14,  # Charge pump setting (enable)
        0xAF,  # Display on
    ]
    for cmd in commands:
        bus.write_i2c_block_data(I2C_ADDRESS, COMMAND_MODE, [cmd])

# Write a single command
def write_command(bus, cmd):
    bus.write_i2c_block_data(I2C_ADDRESS, COMMAND_MODE, [cmd])

# Send data to the display
def write_data(bus, data):
    max_chunk = 32  # SMBus can only handle up to 32 bytes per transaction
    for i in range(0, len(data), max_chunk):
        chunk = data[i:i+max_chunk]
        bus.write_i2c_block_data(I2C_ADDRESS, DATA_MODE, chunk)
```

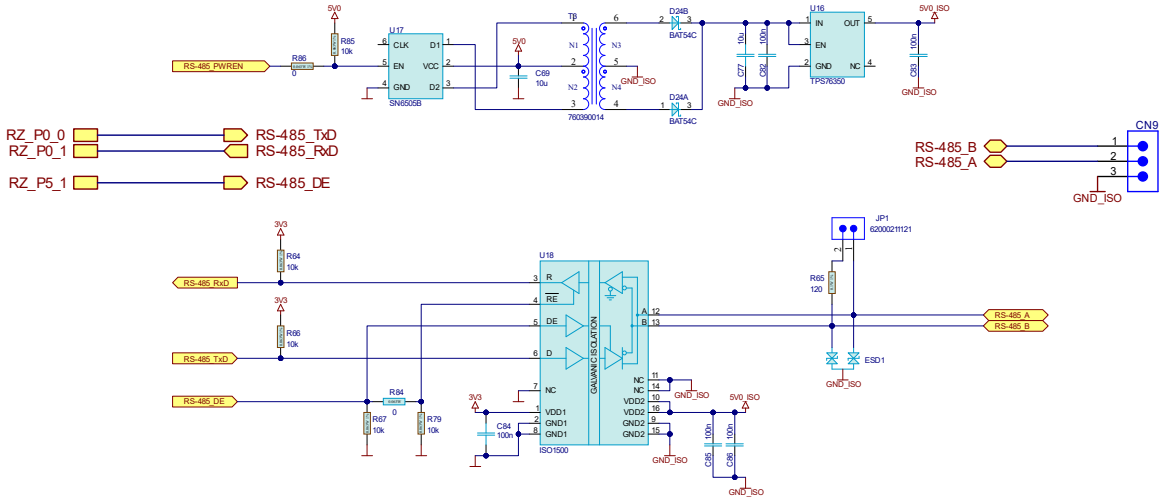
```
# Draw logo
def display_logo(bus):
    logo_width = 64
    logo_height = 64
    page_count = logo_height // 8

    # First, clear the entire display (all 128x64 pixels)
    for page in range(page_count):
        write_command(bus, 0xB0 + page)  # Set page address
        write_command(bus, 0x00)  # Set column address lower part to 0
        write_command(bus, 0x10)  # Set column address higher part to 0
        write_data(bus, [0x00] * 128)  # Clear all 128 columns for each page

    # Now draw the logo centered (with 32 pixels blank space on each side)
    x_offset = 32  # Start drawing logo from column 32 (leaving 32 blank pixels on left)

    for page in range(page_count):
        write_command(bus, 0xB0 + page)  # Set page address
        write_command(bus, (x_offset & 0x0F))  # Set lower column address (32)
        write_command(bus, 0x10 | (x_offset >> 4))  # Set higher column address (32)
        start = page * logo_width
        end = start + logo_width
        write_data(bus, Logo[start:end])  # Send the logo data

# Main logic
def main():
    with SMBus(I2C_BUS) as bus:
        init_display(bus)
        display_logo(bus)

if __name__ == "__main__":
    main()
```

- Execute the script from that file: **`sudo python3 ~/SBC-OLED01.py`**.
- You should see the logo on the OLED screen, which confirms the I^2C bus is working.

8.4 RS485

Make sure you are applied **vko-rs485.dtbo** in **/boot/uEnv.txt**.

```
fdt_extra_overlays=vko-rs485.dtbo.
```



RS485 connector

UART[3] pins go to Pin_2 (**A**) & Pin_1 (**B**) of CN9 connector.

Set RS485 module to 19200 8 N 1 → `stty -F /dev/ttySC<n> 19200 cs8 -cstopb -parenb`.

- Init DE pin. RS485 is actually UART with **D**ata **E**nable pin, which goes **Hi** before **TX** transaction and goes back to **Low** after the transaction, so you need to manually control that pin, this means the internal pin should be **P5_1**, which is (**5** x 8) + **1** + 120 = **161**.

  `sudo su` && `echo 161 > /sys/class/gpio/export`.

  `echo out > /sys/class/gpio/P5_1/direction`.

  `echo 0 > /sys/class/gpio/P5_1/value`.

- TX some data: `echo 1 > /sys/class/gpio/P5_1/value`.

  `echo "Helow world" > /dev/ttySC3`.

  Wait data to be fully transmited & exec. `echo 0 > /sys/class/gpio/P5_1/value`.

- RX some data: `cat /dev/ttySC3`.

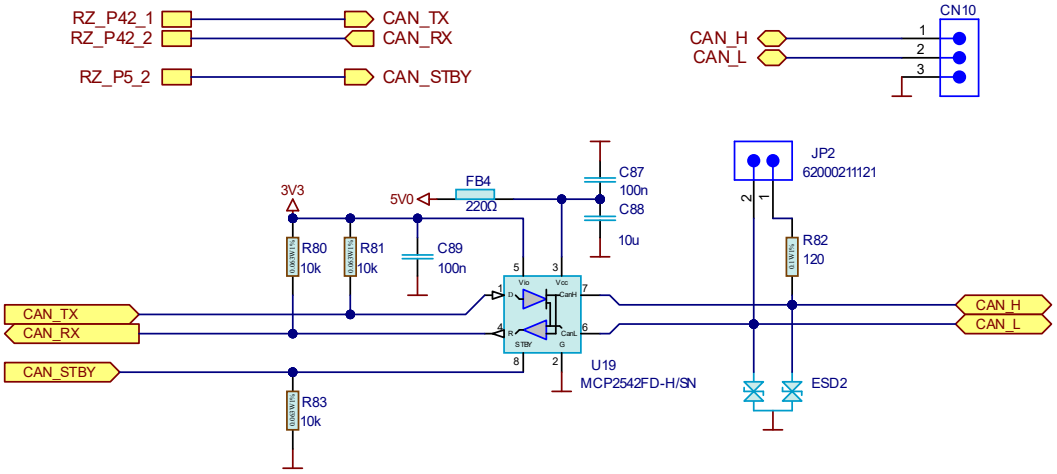
  Note that the receiving terminal can have buffering enabled, so the data read might not be displayed immediately. The buffer is flushed once enough data is entered, or when enough newlines are encountered.

- Release DE pin resource: `echo 161 > /sys/class/gpio/unexport`.

8.5 CAN

Make sure you are applied **vko-can.dtbo** overlay in **/boot/uEnv.txt**.

```
fdt_extra_overlays=vko-can.dtbo.
```



CAN connector.

CAN[x] pins go to Pin_xx (**H**) & Pin_yy (**L**) of CN10 connector.

Use the CAN module:

- Install CAN software: `sudo apt-get install can-utils`.
- Turn off the can module: `ip link set can0 down`.
- Set the config you need: `ip link set can0 type can bitrate 2000000 dbitrate 2000000 fd on`.
- Turn on the can module: `ip link set can0 up`.
- Here a Loopback test is possible if only a special mode exist in the can module, but this is not the case here, so you will need a second can device to talk to.
- If you want to receive messages type: `candump can0`.
- If you want to send message type: `cansend can0 123#0102030405060708`.
- Use `cansend -help` for more info for the format of the can messages (in this case, 11bit address mode is used → [addr: **123**] & full msg len [data: **0102030405060708**])

9. Install Mali GPU driver on Debian

Renesas RZ/G2LC MCU is equipped with **Mali-G31** GPU on chip. Debian has a built in support for that GPU through Mesa/X.org project. The productivity of that driver, however, is quite disappointing, especially when you try to use the browser. You can use the evaluation version of the driver, optimized for that GPU, from Renesas. The downside of it, however, is it's evaluation, i.e. it only works for a couple of hours. For the full version you have to contact with Renesas. The driver supplied by Renesas is only compatible with the Wayland display server protocol, keep that in mind if you consider using it.

- You can **Install** the driver directly by executing the script:
  In short the script installs **Weston**, configures a Weston startup service, and finally installs the driver itself.
  → On the target board, get the script for installation:
  `wget https://vekatech.com/os/debian/get_mali.sh`.
  → On the target board, make it executable:
  `chmod +x get_mali.sh`.
  → On the target board, Install the driver:
  `./get_mali.sh`.

10. Launch App on OS start up

The way to do it, is to create a user service, i.e write a text file, with **.service** extension, located at **/usr/lib/systemd/user**.

Let's say the application that needs to be launched is the **Chromium** browser, and the service is named **browser.service** . The contents of that file should look like this:

```
[Unit]
Description=Chromium (Kiosk  Mode)
After=graphical-session.target

[Service]
Restart=on-failure
RestartSec=5
Environment="WAYLAND_DISPLAY=wayland-1"

ExecStart=/usr/bin/chromium -enable-features=UseOzonePlatform
--ozone-platform=wayland --kiosk https://vekatech.com

[Install]
WantedBy=default.target
```

- To allow the service to be launched on OS start up execute this:
  `systemctl --user enable browser.service`.

- To try if the service works as expected, reboot the system:
  `sudo reboot`.

Revision overview list

Revision number	Description changes
0.1	Initial

Vekatech Ltd.

63, Nestor Abadzhiev st.
4023 Plovdiv
Bulgaria
Tel.: +359 (0) 32 262362
info@vekatech.com

www.vekatech.com