SmartEdge Industrial IoT Gateway User Guide

OVERVIEW

The Avnet SmartEdge Industrial IoT Gateway connects your sensors and other devices to the cloud. This allows you to view status and control connected devices from a customizable web portal from any internet connected location.

SOFTWARE CONFIGURATION

Avnet Image

- Based on Raspbian stretch lite image April 2019
- Enables support for TPM 2.0 for trusted boot
- IoTConnect Cloud solution
- Our github site is https://github.com/Avnet/raspberrypi-industrial-gateway for the linux source code.
- Our github site for u-boot is https://github.com/Avnet/raspberrypi-industrial-gateway-uboot
- Our github site for custom files is https://github.com/Avnet/raspberrypi-industrial-gateway-custom
- Option to enable secure boot to ensure only signed boot files can be executed. Find files and how to information on our github site. (**Important!** Once secure boot is enabled only the AVNET u-boot image can be executed on the gateway. The zImage/linux kernel can be isn't affected by this change).

What is Included with image

- Default user name and password are "avnet" for username and "avnet" for password.
- Normal Raspbian stretch lite utilities such as "raspi-config, and raspi-gpio", etc.
- Various TPM 2.0 compliant utilities can be found in /usr/local/bin.
- U-Boot is Raspberry Pi secure, if using Raspberry Pi secure boot feature. It launches kernel after filling out TPM 2.0 PCR 0 value. This completes the Trusted Platform implementation.
- TPM 2.0 PCR 0 element contains the value of the kernel (/boot/zImage) that is calculated during u-boot.
- Access to github for various software source code and components.
- Default behavior requires a configuration set using the Mobile App "Avnet IoTConnect"
- For a list of preinstall packages, run "dpkg –l" for current debian installed packages. For Python 2.7 installed packages run "pip list".
- Custom utilities are provided for system control and are found in /usr/bin/*.sh or /usr/bin/*.py
- Image startup features are contained in /etc/rc.local.* files and various scripts operate from these files.
- Buttons and REST API for the Mobile App are in the python file "/usr/bin/server.py". For Python SDK information see below.
- NodeRed base line installation, not started. Look at the online NodeRed documentation for Raspberry Pi.
- The IoTConnect SDK will be restarted if it finds a problem, this is controlled by the file "/usr/bin/startup.sh"
- To reboot the gateway use "reboot.sh" from the terminal or your application. This eliminates any hangs do to software reboot, by enabling the onboard microcontroller watchdog feature to remove the cpu power after 30 seconds.
- To reboot the gateway using the RST button, do a short press on the button. This calls "/usr/bin/switch.sh" and the device will reboot in the default mode.
- All other .sh files in /usr/bin are used by others in the system, however some may be helpful to review. A complete list and functional description is in the Appendix.
- An IoTConnect SDK OTA software update method is available.
- Minicom and Tio are included as terminal programs.

Iot Connect

- The Avnet image comes configured with our IoTConnect Cloud solution
- A 30-day free trial is provided
- After the 30-day trial, or when you are ready, you can use the Mobile App to reconfigure your gateway so that it works on your new machine name.
- Please visit the web portal (www.element14.com/gateway) for details on how to continue accessing your device over the cloud beyond the free trial period.

RST Button and upper LED

The gateway will indicate the current status using the following:

- The default mode is blinking RED/GREEN once every second. This indicates that the gateway is in configuration mode (Access Point) and the Mobile App can be used to setup the device.
- Solid GREEN indicates that the configuration was successful and we have a WiFi or Ethernet(SKIP button on Mobile App) connection and are waiting for the SDK to become active
- Blinking GREEN/OFF once every second indicates the SDK is communicating your sensor data to the cloud.
- The RST button can be used to hard reset or put the gateway into default mode. A short press will cause a hard reboot. Use a long press (greater than 10 seconds) to go into default mode with a hard reboot. The LEDS will blink RED/GREEN in default mode.

HOW THE SOFTWARE WORKS

What happens during boot

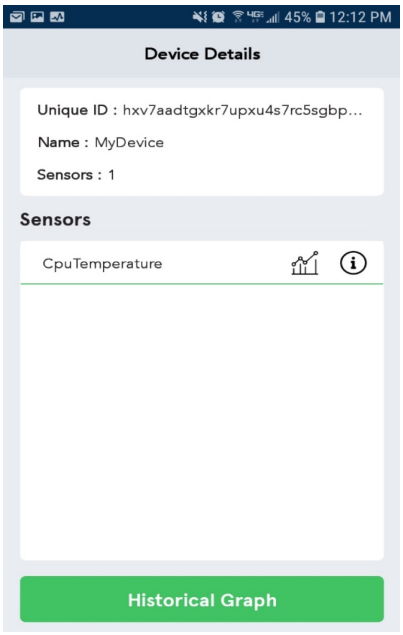
The boot process can be observed on a HDMI screen. A USB keyboard can be used to log into the system and run commands.

The boot process starts with some initial messages from the bootloader indicating the system is powering up. The next set of messages will be coming from the linux kernel. Just before the login prompt you will see a message indicating "My IP address is x.x.x.x" followed by the name of your device "IotGateway xxxxxxxx" where xxxxxxxx is the 8 digit serial number of the device. Once the kernel boots, all the operations for the gateway can be found in /etc/rc.local. These operations are required to use the mobile app and setup the device to talk to the IoTConnect cloud. Since this is a Trusted Platform using TPM 2.0 the boot up PCR list can be utilized, so that your system setup can be verified as trusted. Run the command "tpm2_pcrlist" and see SHA256 values for PCR 0 and 10. For more information on using the TPM tools installed on your system see the following web site https://github.com/tpm2-software/tpm2-tools/wiki/How-to-use-tpm2-tools . Our specific linux kernel version is "Linux raspberrypi 4.14.79-v7+" and can be verified by using the following terminal command "uname –a". The specific date stamp for the image creation can be found using the following terminal command "cat/usr/bin/ImageDate.txt".
You will also be able to use this setup as your linux terminal. When you see the following enter the text below.

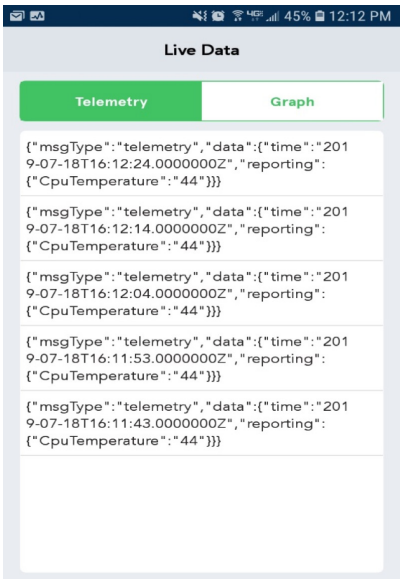
Raspberrypi login: **avnet** Password: **avnet**

HOW TO SETUP THE GATEWAY

Avnet IoTConnect app is available at the Google and Apple app stores. The app is required to setup the Gateway. The app is used to setup your user account for the 30-day trial period. It will also configure the gateway's network connection to the IoTConnect cloud system. Once you complete this process, the device will be listed in the mobile app. Tapping the icon for the device will open a device screen which will look similar to this one:



Tapping on the CpuTemperature icon, will present live Telemetry data from the device. This screen will look similar to this one:



WHAT'S NEXT

Now the device is communicating to the IoTConnect cloud. The SDK can be customized for sensors used, commands sent from cloud, complete the setup of the device on the web portal, etc. The next sections will go over these and other various things.

ADDITIONAL CLOUD SETUP ON WEB PORTAL

Once the mobile app set up is complete, then login into the web portal using a standard browser. The following web address should be used https://avnet.iotconnect.io/login. The same username and password should be used that were used on the mobile app. Once logged in your device can be found on Device tab on the left side. Also documentation is available under the Resource tab. Clicking on the device tab will show a list of the devices that are configured for the logged in user. To accomplish more advanced options like OTA updates, Command Setup, and other options available to an Administrator, the device will need to transition from the 30 day trial period to your own system. Once that's done full access will be provided using the administrative account on the new system to do other important tasks.

The following procedure can be used to do OTA updates on the gateway.

1) On a linux machine make a directory called updates.
2) Put a file in that directory called install.sh. Make sure the permissions are set to 777 on the file. See appendix for an example.
3) Add anything you want do to in that script file.
4) Include supporting files in that directory, this would be the files you want installed.
5) Compress the updates directory contents into a .tar.xz.gz file
6) Rename this file to "install".
7) Setup the web portal for the OTA and point the file for OTA to your install file.
8) Make sure the device is on and sending data to the cloud, this is important as the OTA will not get sent unless the device is connected to the cloud.
9) Issue a FORCED OTA update to your device.
10) The software will be installed. Double check on the gateway that the files were installed.

Also, the install files will be left in the SDK area in case you need to further debug/analyse failures.

CONFIGURING THE SDK

When you first get your Gateway, you should configure it with the Mobile App Avnet IoTConnect, which is available at the App store for IOS and Android. Once the gateway is configured, you should be able to login to the web portal at "avnet.iotconnect.io" and see your device. The factory configuration sends "CpuTemperature" to the cloud. The IoTGateway uses an access point with WiFi configuration:

– SSID: IoTGateway_xxxxxxxx"

– WPA PSK: IoTConnect

Modifying the SDK can be done by following these steps

1) Connect a HDMI screen and USB keyboard to the device, then power the device on. Now reboot the device and log in. Once you have logged , the command line prompt "avnet@raspberrypi:~ $ will be available. The user home directory is /home/avnet.
2) Use the following command to get to the SDK directory. "cd /usr/bin/IoTConnectSDK_Py2.7/Testing/sample"
3) The 2 files you will need to edit for your configuration are "IoTConnectSDK.conf" and "user_functions.py". They are found in the SDK directory.
4) The .conf file contains sections for "CloudSDKConfiguration", "CloudSystemControl", and "CloudSDKDefaultObject<xxx>" entries. For the 30 day trial period you only need to setup the section for "UserName" and "PassWord" in "CloudSystemControl" which will be the same credentials you used to create your account with the Mobile App "Avnet IoTConnect".
5) For the "CloudSDKDefault options, you should add the section names [CloudSDKDefaultObject<xxx>] sequentially numbered and update the "DefaultObjectCount" in the "CloudSystemControl" section.
6) Working examples for Objects are commented out in the "IoTConnectSDK.conf" file.
7) The file "user_functions.py" is used to add any custom python code to the system. There is a requirement that the first two fucntions remain avaialble. They are "def user_callbackMessage(msg):" which is used for COMMANDS sent from the cloud. The second function is "def user_Initialize():" This function gets called when the SDK starts and you can put any initialization code here.
8) There is an example called "def ThisIsMyFunction():" that is commented out and simply prints to the console.
9) Now if you want a simple test do the following:
   a. Add a section to the "IoTConnectSDK.conf" file
   b. [CloudSDKDefaultObject2]
   c. # Predefined objects can go here. NUMBER/STRING/OBJECT
   d. Value=NUMBER
   e. Name=TestUserFunction
   f. UsePythonInterface=ThisIsMyFunction
10) Then uncomment the function in "user_functions.py", for the items can will be sent to the cloud.
11) Now issue the following command "reboot.sh"
12) The new information you just added will show up on the cloud.

Changing the default device parameters on the cloud

To change the device parameters on the cloud, edit the IoTConnectSDK.conf file and modify the parameters for Display_Name and Template_Name to the desired settings. Note: do not use quotes. Then remove the device and template for that device from the cloud . The device will be re-enrolled using the new settings, there is no need to reboot for this change to take effect. After the change, the new template can modify on the cloud. This includes using the OTA feature.

Debugging the SDK

After you get the basic idea and want to be sure your changes are working you can do the following.

1) Plug in the HDMI screen and USB keyboard, power off /on, then log on.
2) Go to the SDK area described earlier.
3) Run commands "sudo pkill startup" and "sudo pkill python". Now you are ready
4) Run this command to see the SDK output. "sudo -E python example.py"
5) Now you should be able to see what's going on and track down odd behavior, such as you have a syntax error in "user_functions.py", etc.

Included in the SDK are options for CpuTemperature, CpuFrequency, DigitalIN1, DigitalIN2, DigitalIN3, and DigitalIN4. There are also options for Reboot, FactoryDefault, SetDigitalOutput1, SetDigitalOutput2, SetDigitalOutput3 , SetDigitalOutput4, ClearDIgitalOutput1, ClearDigitalOutput2, ClearDigitialOutput3, ClearDigitalOutput4, that are setup as cloud commands.

SYSTEM OPERATIONS

On boot up, the gateway will look up the TPM unique ID and Endorsment key. It will then construct the BSSID for the gateway access point mode and startup access point mode services. The gateway's BSSID will be broadcast so the mobile app can connect to it and the RED/GREEN leds will be blinking. Then the API is started that communicates with the mobile app for gateway configuration. After the mobile app configures the gateway the contents of /etc/rc.local will change and the gateway operating state will shift from Access Point mode to SDK running. The GREEN/OFF leds will be blinking at this time. The following services will be started at this time. The python SDK will be started and the API for the LEDS/BUTTON/REST will be started. Now date will be send to the cloud. The system will restart the SDK if there is a problem.

You can control the system services to stop the SDK or the LED/BUTTON/REST service. There are also scripts that will put the gateway back to the default state, cleanly reboot with no hangs.The system has access to the on board micro for hardware reset which the reboot.sh command uses. The gateway Broadcom hardware driver on /dev/watchdog0 is also available for any application needs

1) To stop the LED/BUTTON/REST service run command "sudo pkill server"
2) To stop the SDK, run command "sudo pkill startup" then "sudo pkill example"
3) To put the gateway back to the default state run the command "switch_only.sh"
4) To cleanly reset run "reboot.sh"

SYSTEM CUSTOMIZATION

The gateway has the potential to be customized in almost any fashion.

The standard Pi HAT connector can be used to add already available cards, or you can develop and use your own HAT card. Standard HAT cards can be easily added to the system, all you need to do is edit the file in /boot/config.txt and add an overlay for your standard HAT card. If you make your own card and need a custom kernel driver , then the w kernel source files from our github site, are required. A cross compiler for ARM is also required. We use cross-gcc-8.3.0-pi_2-3 tool set. Any toolset with GCC 8.3 or greater that compiles to arm linux gnueabihf- standards will work. Follow your standard developer instructions for adding your driver/device tree overlay to the kernel source tree.

The digital inputs and outputs in any fashion. There are hooks in the SDK for sending the Digital Inputs to the cloud. If you need cloud interface to the digital outputs, you must use the cloud command feature and add hooks in your SDK configuration (IotConnectSDK.conf and user_functions.py). The digital outputs are available on /sys/class/gpio/gpio201, gpio203,gpio205, and gpio207. The inputs are on /sys/class/gpio/gpio200, gpio202, gpio204, and gpio206. They all must be exported in your application to use them. If they are used in the SDK (IoTConnectSDK.conf) then the ones specified are exported automatically.

The RS-232/RS-485 port is pre-configured for RS-485 operation. On board jumpers are used to setup the port for RS-232/485 and half/full duplex in the case of RS-232 or RS-422. This port is available for anything you need, like a serial terminal or an industrial based RS-485 device. The port is found on /dev/ttySC0. See the Appendix for default jumper position and other jumper configurations for this port.

The internal CAN bus is also available but disabled in the image. It can be enabled by removing the comments dealing with the can0 section in /etc/network/interfaces. Utilities "CANSEND and CANDUMP" have been included and the can0 device can be used by the network stacks. Note: there is a CAN bus termination jumper inside the gateway next to the CAN connector. See the Appendix for more information.

The u-boot portion of the code is also available on the github site. If the intent is to generate a custom boot loader, we would recommend that the secure boot feature is never enabled. If it is the u-boot generated will need to be digitally signed by Avnet. Also, a completely different solution can be loaded, however if it replaces any files in the /boot directory and if secure boot is enabled it will not work. If secure boot is enabled it can NEVER be disabled. If a different solution is used Avnet will have to digitally sign the appropriate files before you can enable secure boot.

There is a battery backed up RTC on the gateway. For systems that do not have internet connectivity (I.e. closed system) then this can be set then enabled with a few standard linux commands to keep the date/time even if the power fails. Normally this system is on the internet and the correct time information is provided using network time keeping protocols.

There is a PCIe connector on board and accepts most PCIe USB devices and is primarily used for cellular modem connections. Note: If you decide to use a cellular modem there will be customization needed control data calls. Standard linux tools that are available can be used for dialer customizations.

The gateway has the NodeRed project baseline installed on the system. By default the service is not started. Additional NodeRed modules will be provided for the device, as they are currently being developed by Avnet. For more information on NodeRed and how to use it, use the following web site. https://nodered.org/docs/getting-started/raspberrypi

The ssh terminal service is disabled by default. You can use the raspi-config tool to enable it. The setting is under the interface options section. Also add an empty file called "ssh" to the /boot directory, then reboot.

The raspi-gpio tool can be used to see the default GPIO configuration for the gateway. If custom hardware is used on the HAT connector you should use the device tree overlay to setup and HAT connection GPIO. Then use the raspi-gpio tool to ensure the GPIO's are configured correctly.

DOWNLOAD A NEW IMAGE OR UPDATE FILES ON THE DEVICE

There are several options to update files on the device. Also, the micro usb port near the center of the board can be used to download a new image or update files on the device. It can also be used to save your current gateway image.

To download a new image or update files on the device, get the custom files from our github site. The rpiboot directory contains the needed files to use on your PC linux system for this feature. Note: The rpiboot must be complied for your version of linux. Run "make" to do this. There are 2 scripts that can be used after the code is compiled. The scrip rpi.sh is the non secure method to do the updates. First plug in the micro usb to the PC, then power on the gateway. The gateway will show up as a new /dev/sd* file system and you can mount either the /boot area or the / rootfs area if you intend to update files. To update a new image use the linux "dd" command. To save the current image use the linux "dd" command.

The gateway USB port can be used to update files on the device. Put your files on a usb stick , then plug it into the gateway. On a gateway terminal you will have to mount the usb device as it does not auto mount for security reasons. Then simply copy the files from the USB stick.

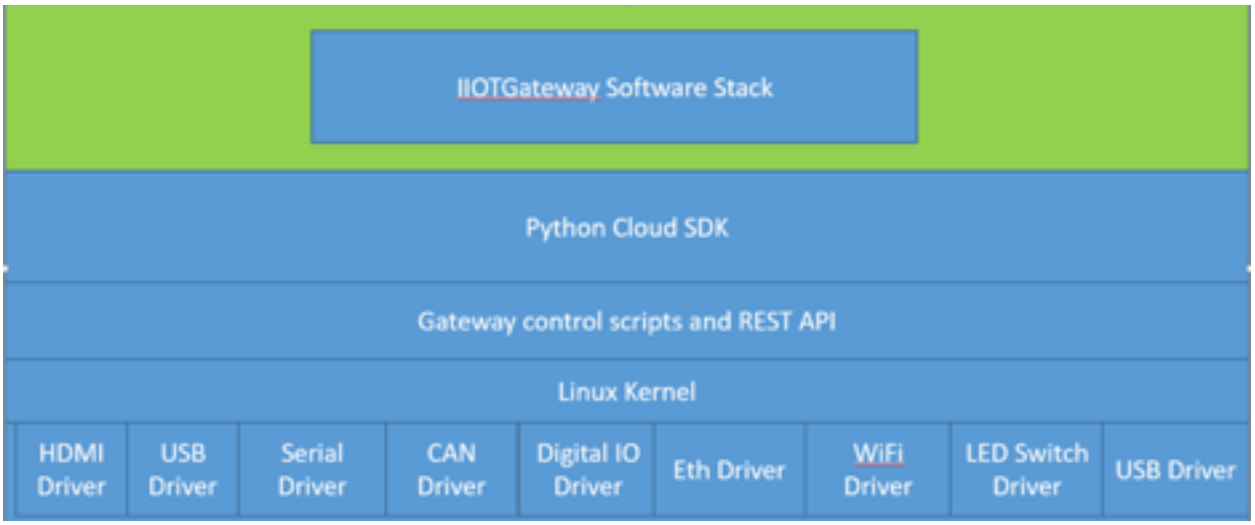
PROVISIONING MULTIPLE DEVICES

Multiple gateways can be provisioned without the Mobile Application. To accomplish this task the gateway should be plugged into the Ethernet. Then the REST API's can be used for this purpose. See appendix for REST API's supported. The process would look something like this. Note: These features will be available in future images, they can be downloaded from the github custom URL specified above.

1) Find the IP addresses for the connected devices.
2) Issue the /DeviceID REST command to get the device serial number. You can use this to select the files needed for that device. If you are unsure just issue the REST command to all IP addresses and if it responds then proceed with that Device ID.
3) Issue the /WiFiGetWPAConf to obtain the current wpa_supplicant.conf file. This can be used if the current settings need to be preserved.
4) Issue the /WiFiIWList to get the currently available access points. This can be used if to determine what access points are available. If the connection settings are already known, they can add them to wpa_supplicant.conf.
5) Issue the /WiFiSetWPAConf to set the new wpa_supplicant.conf file.
6) Issue the /IOTGetIOTConnectSDK.conf command to get the current SDK configuration file.
7) Modify the SDK configuration file, the use the IOTSetIOTConnectSDK.conf file on the gateway.
8) Issue the /CloudAttach command to setup the WiFi connection, if configured. It will also start the SDK.

APPENDIX

GATEWAY SOFTWARE STACK



JUMPER CONFIGURATIONS

The default jumpers for the RS-232/RS-485 are

J22 – pin 1 jumpered to pin 2 . Selects RS-485

J22 – pin 7 not jumpered. Selects half duplex operation.

All other RS-232/RS-485 jumper configurations are found near the beginning of this document.

The default jumper for the CAN bus termination resistor is enabled.

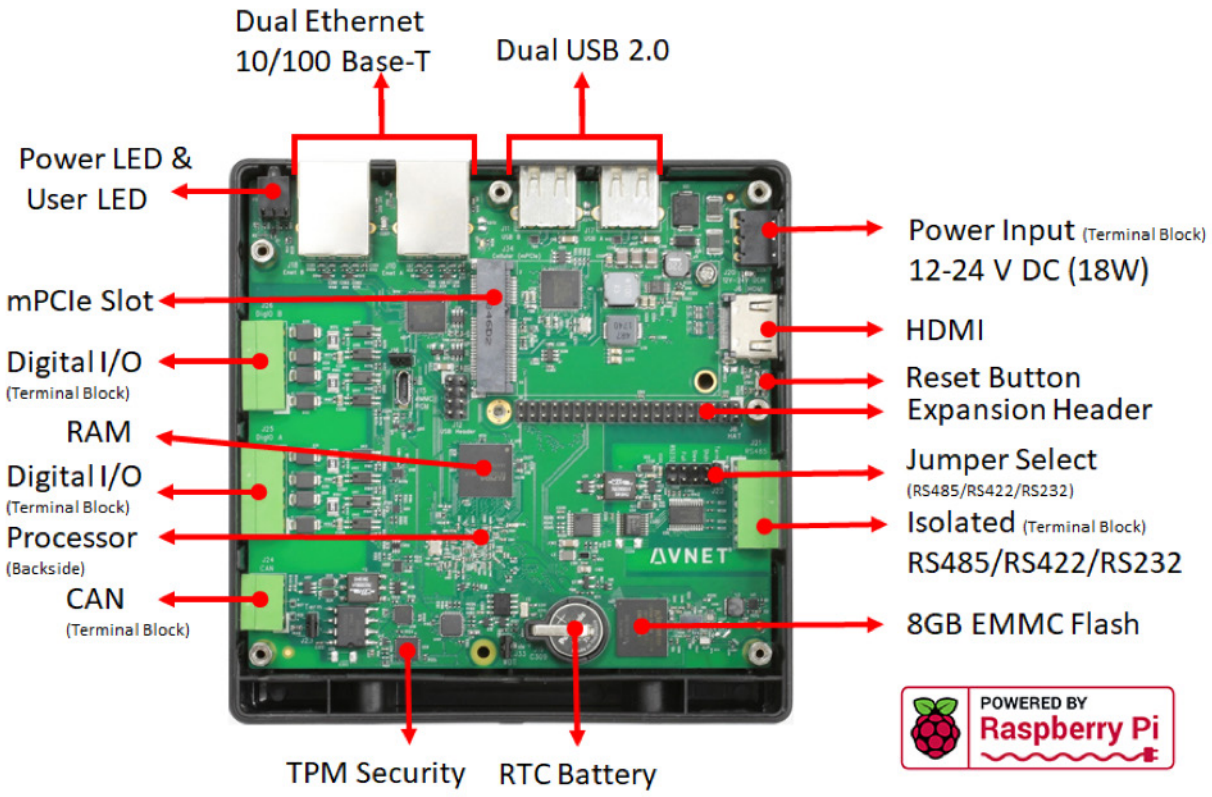
For how to configure the CAN bus termination please reference CAN bus documentation found on https://www.ni.com/en-no/innovations/white-papers/09/can-physical-layer-and-termination-guide.html

The default WDT jumper is not enabled. This feature is currently not enabled and should never be jumpered.

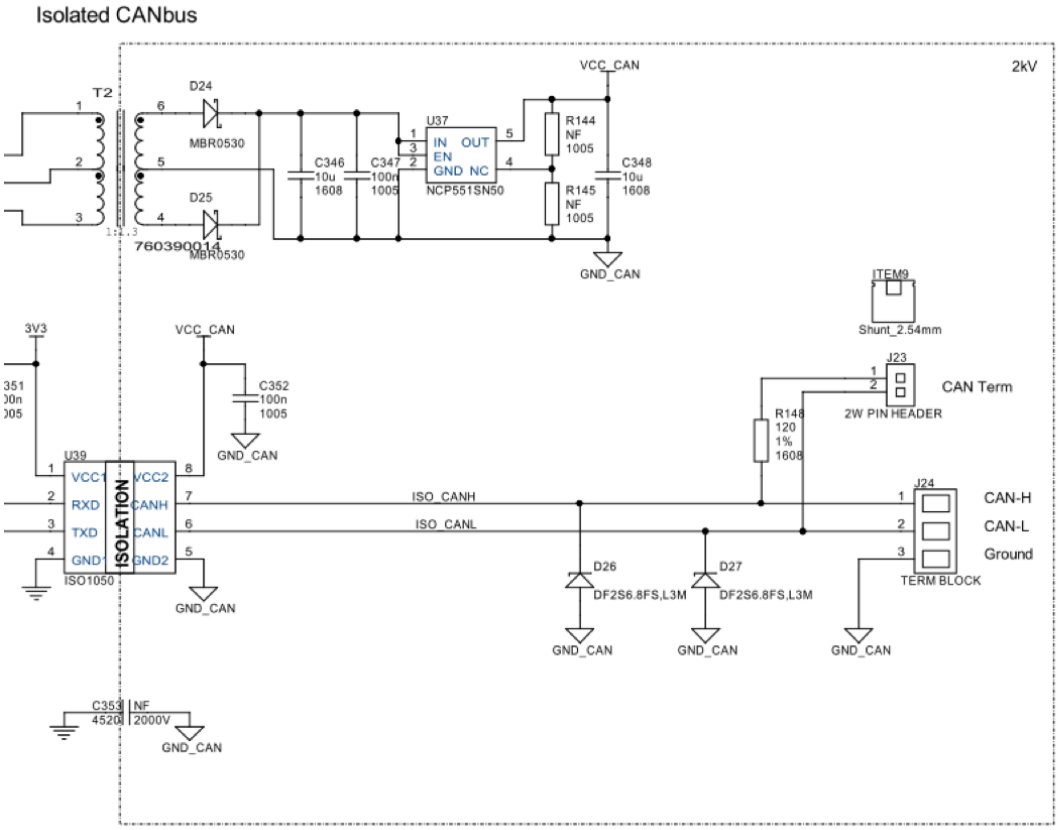
The jumper near the onboard USB micro connector J16 should be left in the default shipping position. This jumper is used with the usb port for complete image updates using the rpiboot utility. More information can be found later in this appendix.

SCHEMATIC REFERENCES

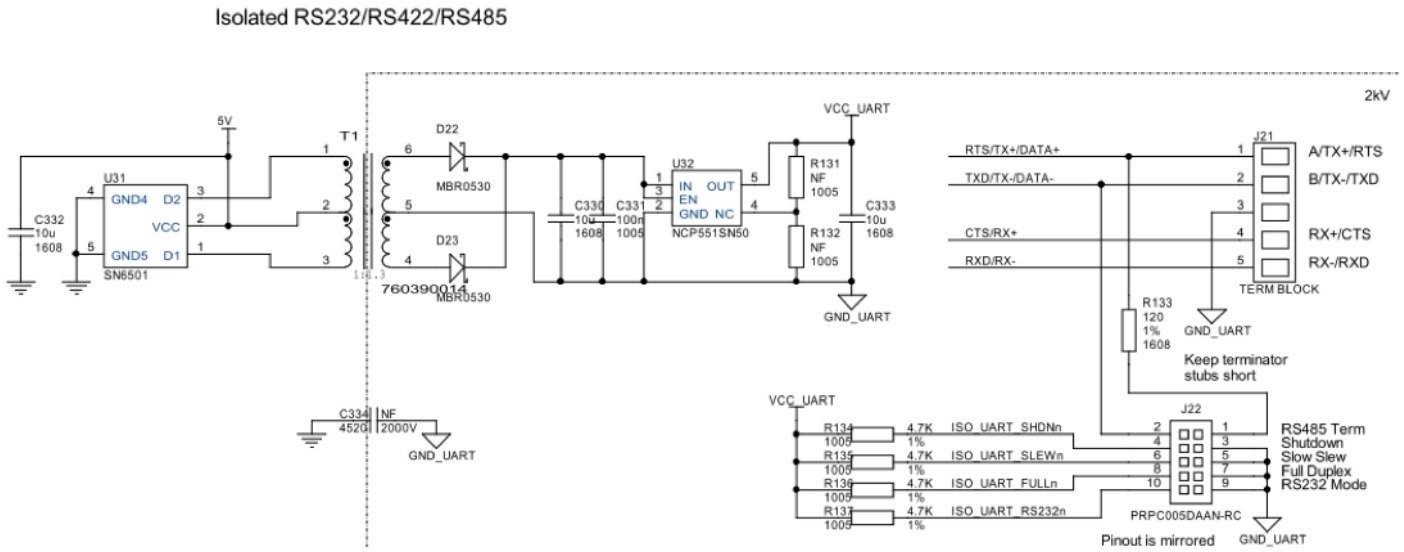
Here is the CAN bus schematic for the gateway



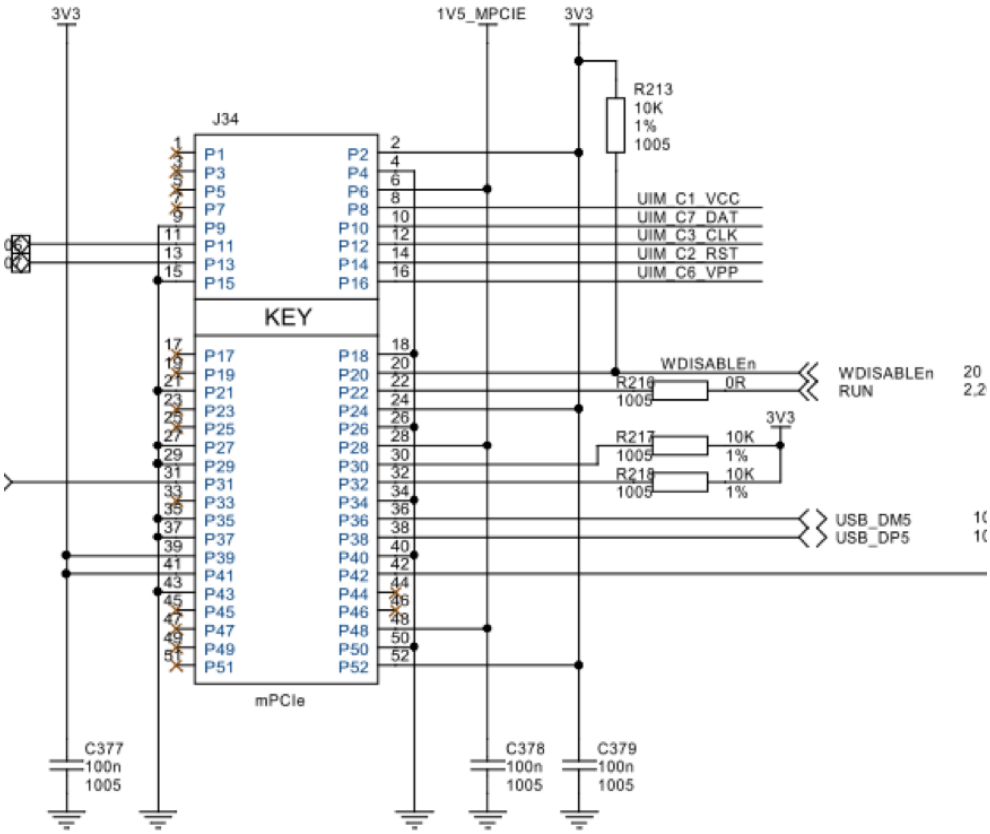
Here is the RS-232/RS-485 schematic for the gateway



SCHEMATIC REFERENCES

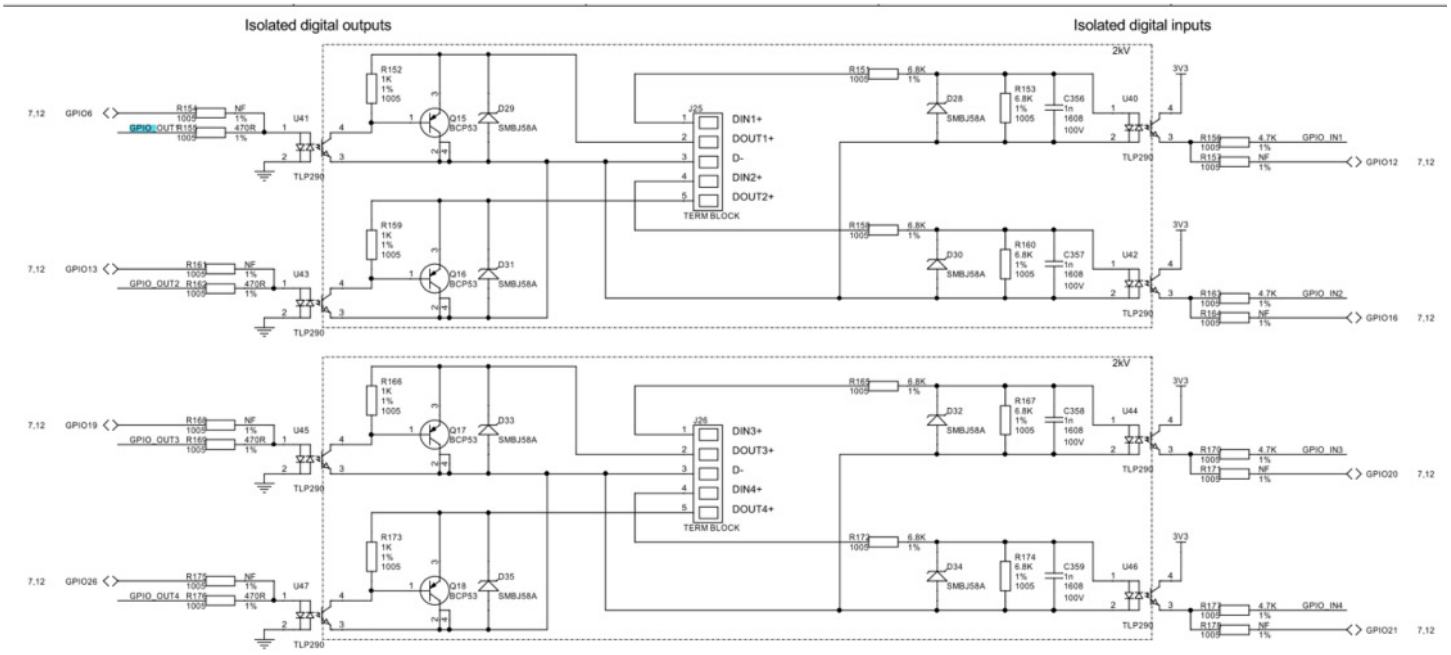


Here is the PCIe schematic for the gateway



SCHEMATIC REFERENCES

Here is the digital IO schematic for the gateway



OTA update example install.sh

install.sh

echo "Installing software updates"

sudo cp rc.local /etc/rc.local

ehco "Installation Complete!"

NOTE: must include new rc.local and install.sh in the compressed installation file. See above text for description.

1 REST API DOCUMENTATION

- Get Device ID
- Get Network Status
- Get Device Activation Status
- Get WiFi Access Point List
- Put WiFi Client SSID/PSK.
- Get WiFi Client Connection Status. (I'm testing this now but is seems likely I can do both STA can AP at the same time)
- Put WiFi Access Point Disable.
- Put NewCPID Set new cloud system identifier/username/password
- Put CloudAttach Start SDK and configure wifi
- Get IOTGetIOTConnectConf get the current SDK configuration file
- Put IOTSetIOTConnectConf set the current SDK configuration file
- Get WiFiGetWPAConf get the current wpa_supplicant.conf file
- Put WiFiPutWPAConf set the current wpa_supplicant.conf file.

1. Title: Get Device ID

Description: Using this API we should get device id in response.

Command: curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET http://192.168.2.1:8080/DeviceId

Possible Response:

```
{
        "DeviceID": {deviceid}
}
```

2. Title: Get Network Status

Description: Using this API we should get network (internet) status, it is connected or not.

Command: curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET http://192.168.2.1:8080/NetworkStatus

Possible Response:

```
{
        "IsConnected": {true/false}
}
```

3. Title: Get Device Activation Status

Description: Using this API we will know the device is activated into IoT connect or not. When SDK done with initialization process this flag needs to be set it true.

Command: curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET http://192.168.2.1:8080/DeviceActivationStatus

Possible Response:

```
{
        "IsActive": {true/false}
}
```

4.Title: Get WiFi Access Point List

Description: Using this API we should get WiFi Access Point List

Command: curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET http://192.168.2.1:8080/WiFiAccessPointList

Possible Response:

```
        {
    "AccessPointList": [
{
"SSID": "value",
"SignalStrength": "value"
},
{...}
]
}
```

5.Title: Put Client WiFi SSID/PSK

Description: Using this API we should get the response from the REST command after SSID/PSK setup is completed.

Command:

curl -X PUT -H "Content-Type: application/json" -d '{"SSID":"PSK"}' http://192.168.2.1:8080/WiFiClientSSID_PSK

Possible Response:

```
{
        "Response": { true/false}
}
```

6.Title: Get WiFi Client Connection Status

Description: Using this API we should get WiFi client connection status. Note: Wait at least 30 second after posting SSID/PSK for this to become valid.

Command: curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET http://192.168.2.1:8080/WiFiClientConnectionStatus

Possible Response:

{

"IsActive": { true/false}

}

7.Title: Put WiFi Access Point Disable.

Description: Using this API we should disable the WiFi Access Point Mode. Note: Response will be given before we disconnect 30 seconds later.

Command:

curl -X PUT -H "Content-Type: application/json" -d '{"SSID":"1", "PSK":"2"}' http://0.0.0.0:8080/WiFiAccessPointDisable

Possible Response:

{

"IsActive": { true/false}

}

8.Title: Put IOTNewCPID Set new cloud system identifier.

Description: Using this API you can set your system identifier/username/password. Note this will restart the example python code if it is running.

Command:

curl -X PUT -H "Content-Type: application/json" -d '{"cpid":"123dddd4567","username":"crash1xxxx0@gmail.com","password":"1xxxx0"}' http://192.168.1.134:8080/IOTNewCPID

Possible Response:

{

"Response": { 1, -1}

}

9.Title: Put CloudAttach

Description: Using this API will cause the gateway to configure the wifi connection and start the SDK.

Command:

curl -X PUT -H "Content-Type: application/json" -d '{""}' http://192.168.2.1:8080/CloudAttach

Possible Response:

{

"Response": { 1/0}

}

10.Title: Get the current IoTConnectSDK.conf file

Description: Using this API returns the current IOTConnectSDK.conf file

curl -X GET -H "Content-Type: application/json" -d '{""}' http://192.168.2.1:8080/IOTGetIOTConnectSDKConf

Possible Response:

{

"IOTConnectSDKConf": { 1/0}

}

11.Title: Set the current IoTConnectSDK.conf file

Description: Using this API sets the current IOTConnectSDK.conf file

curl -X PUT -H "Content-Type: application/json" -d '{"<contents of file"}' http://192.168.1.134:8080/IOTSetIOTConnectConf

Possible Response:

{

"IOTConnectSDKConf": {1/0}

}

12.Title: Get the current wpa_supplicant.conf file

Description: Using this API returns the current wpa_supplicant.conf file

curl -X GET -H "Content-Type: application/json" -d '{""}' http://192.168.2.1:8080/

Possible Response:

{

"WiFiWAPConf": {<contents of file>}

}

13.Title: Set the current wpa_supplicant.conf file

Description: Using this API sets the current wpa_supplicant.conf file

curl -X PUT -H "Content-Type: application/json" -d '{"<contents of file"}' http://192.168.1.134:8080/WiFiSetWPAConf

Possible Response:

{

"Response": {1/0}

}

SDK DEFAULT FILES ON DEVICE

IotConnectSDK.conf

++

[CloudSDKConfiguration]

scopeid = One0005A911

env = avnetpoc

cpid = B3A7D54220AD4397ABF35D7EC539FBA6

[CloudSystemControl]

http_auth_token = https://avnetauth.iotconnect.io/api/v1.1

http_auth_login = https://avnetauth.iotconnect.io/api/v1.1

http_device_create = https://avnetdevice.iotconnect.io/api/v1.1

http_device_template = https://avnetdevice.iotconnect.io/api/v1.1

template_guid =

template_name =

display_name =

entity_guid = 12F60889-BF0E-402B-A3B4-FFD38BA62D62

username =

password =

solution-key = QkQ2OTg1MUMtM0M4Ni00NzZGLUJDODctOUJDRThDQ0E4QkU4LWFjY2Vzc0tFWS1pazZhbW8wbmJp

defaultobjectcount = 1

defaultcommandcount = 2

defaultsenddelayseconds = 10

#

Command definitions

#

[CloudSDKDefaultCommand1]

commandname = Reboot

command = Reboot

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = RebootNow

[CloudSDKDefaultCommand2]

commandname = FactoryDefault

command = FactoryDefault

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = FactoryDefaultNow

#[CloudSDKDefaultCommand3]

#commandname = SetDigitalOutput1

#command = SetDigitalOutput1

#hasparameter = 0

#requiresack = 0

#isiotcommand = 0

```
#usepythoninterface = SetDigitalOutput1Now

#[CloudSDKDefaultCommand4]
#commandname = ClearDigitalOutput1
#command = ClearDigitalOutput1
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = ClearDigitalOutput1Now

#[CloudSDKDefaultCommand5]
#commandname = SetDigitalOutput2
#command = SetDigitalOutput2
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = SetDigitalOutput2Now

#[CloudSDKDefaultCommand6]
#commandname = ClearDigitalOutput2
#command = ClearDigitalOutput2
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = ClearDigitalOutput2Now

#[CloudSDKDefaultCommand7]
#commandname = SetDigitalOutput3
#command = SetDigitalOutput3
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = SetDigitalOutput3Now

#[CloudSDKDefaultCommand8]
#commandname = ClearDigitalOutput3
#command = ClearDigitalOutput3
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = ClearDigitalOutput3Now

#[CloudSDKDefaultCommand9]
#commandname = SetDigitalOutput4
#command = SetDigitalOutput4
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = SetDigitalOutput4Now

#[CloudSDKDefaultCommand10]
```

```
#commandname = ClearDigitalOutput4
#command = ClearDigitalOutput4
#hasparameter = 0
#requiresack = 0
#isiotcommand = 0
#usepythoninterface = ClearDigitalOutput4Now

#
# Sensor definitions
#

[CloudSDKDefaultObject1]
value = NUMBER
name = CpuTemperature
usepythoninterface = GetTheTemp

#
# Predefined yet disabled for shipping.
#
# Uncomment out the section you want to add to your cloud Attributes
#
#[CloudSDKDefaultObject2]
# Predefined objects can go here.
#Value=NUMBER
#Name=CpuFrequency
#UsePythonInterface=GetTheFreq

#[CloudSDKDefaultObject3]
# Predefined objects can go here.
#Value=NUMBER
#Name=DigitalIN1
#UsePythonInterface=GetDigitalInput1

#[CloudSDKDefaultObject4]
# Predefined objects can go here.
#Value=NUMERIC
#Name=DigitalIN2
#UsePythonInterface=GetDigitalInput2

#[CloudSDKDefaultObject5]
# Predefined objects can go here.
#Value=NUMBER
#Name=DigitalIN3
#UsePythonInterface=GetDigitalInput3

#[CloudSDKDefaultObject6]
# Predefined objects can go here.
#Value=NUMBER
#Name=DigitalIN4
#UsePythonInterface=GetDigitalInput4
```

User_functions.py

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# this function must always exist it also allows you to add hooks for
# commands sent from cloud.  It can be empty but the example prints out
# some important information.
def user_callbackMessage(msg):
    if msg != None and len(msg.items()) != 0:
        cmdType = msg["cmdType"]
        #print(cmdType)
        #if msg["cmdType"] != None else None
        data = msg["data"]
        if data != None:
            print("\n--- User Command Received ---")
                print("Ack " + str(msg[`data'][`ack']))
                print("AckID " + str(msg[`data'][`ackId']))
            print("Command " + str(msg[`data'][`command']))
            print("UniqueID " + str(msg[`data'][`uniqueId']))
            print("CmdType " + cmdType)

# this function must always exist it gets called on SDK startup so that you
# can setup anything custom for your user functions.
def user_Initialize():
    print("User Initialization Called")

# Put your user functions starting here.
#def ThisIsMyFunction():
#    print("This is my function")
#    return 0
```

ERRATA

For the initial release, the file /usr/bin/ImageDate.txt is missing. The release date was 07/19/2019

1) If you wish to use the feature to create your own template name and device name then first edit the file wifi_client.sh and change this line

```
cat provisioning.txt | tr -d "\r\n" | cut -d `:' -f2 | cut -b 1-424 | tr -d "\r\n" >tmp3.txt
```

To

```
cat provisioning.txt | tr -d "\r\n" | cut -d `:' -f3 | cut -b 1-424 | tr -d "\r\n" >tmp3.txt
```


Avnet
2211 S. 47th Street
Phoenix, AZ 85034
1-800-332-8638
avnet.com