Embedding it better...

µTasker Document

µTasker – Wi-Fi

Table of Contents

1. Introduction

Radio connectivity requirements for even the simplest of embedded systems has increased continuously over the last few years and a large amount of inexpensive solutions have become available from a multitude of suppliers.

Wi-Fi is one form of such connectivity that is characterised by the possibility to securely integrate the device into high speed networks (up to about 50Mb/s) that are readily available in many locations. Wi-Fi has not only a fairly high data rate and good range, but also good responsiveness but does consume more power than low power radio communication alternatives (like Bluetooth).

This document discusses the strategy chosen by the μTasker projects in order to integrate a number of Wi-Fi modules for quick and easy use in μTasker based products.

1.1 The Typical Wifi Module

Since Wi-Fi modules represent high technology solutions they are invariably implemented using a powerful (eg. Cortex-m4 based) processor that includes the necessary control, security and network protocols. In addition the high frequency part is also circuitry subject to certification requirements and such modules allow OEM users to benefit from the knowledge that certified modules can be used without additional concerns for such testing and certification.

Often such modules support a low frequency communication interface, such as UART, for a host to control the overall operation and implements large parts of the networking interface – such as web server, TCP sockets, etc. In addition there is usually also a higher speed interface, such as SPI or SDIO, that allows the host to send and receive Ethernet type traffic so that it can be more involved with the overall operation.

In addition to the Wi-Fi functionality the modules often include other features, such as Bluetooth – these additional features are not discussed in any detail in this document.

1.2 Wi-Fi Firmware

As noted above, Wi-Fi modules are complex devices that include firmware and it is important that this firmware can be managed within the overall product. The manufacturer of the module may release updates to the Wi-Fi firmware used in their product in order to fix bugs or security issues that have been identified and some of these may be critical for the overall product performance or security. It is therefore imperative that products using this technology contain the necessary support to ensure that not only its own firmware can be keep up to date but also that the Wi-Fi module's firmware is an inherent part of this overall concept.

1.3 Supported WiFi Modules

There is a large and ever growing range of Wi-Fi modules that are available for integration into embedded products and it is not possible to support all. Therefore a selection of modules have been identified for inclusion into the µTasker that should be found useful in the majority of situations. *Please request directly if you have the requirement for integration of particular solutions for specific product developments since the experience available can be leveraged to quickly and efficiently enable custom solutions if required.*

1. *ESP32-S2-WROOM and ESP32-S2-WROVER*

These modules are very low cost and allow users to integrate their own code into the device's processor core. 2.4GHz. It is also possible to connect the modules via UART which – although not the fastest data throughput - makes adding them to existing systems simple as long as a UART is free. It also allows the µTasker simulator to easily be connected to a module for development work.

2. *Microchip ATWINC1510 and ATWINCC3400*

These modules are moderately priced and their complete firmware contains rich features maintained by Microchip. 2.4GHz.

3. *Telit WL865E4-P*

These modules are professional level parts and their complete firmware contains rich features maintained by Telit. 2.4GHz, 5GHz dual-band.

4. *Telit WL866C6-P + LE910C1.AP*

These modules are professional level parts allowing a combined 2.4GHz, 5GHz dual-band Wi-Fi and 4G cellular module.

1.4 Wi-Fi Firmware Concept

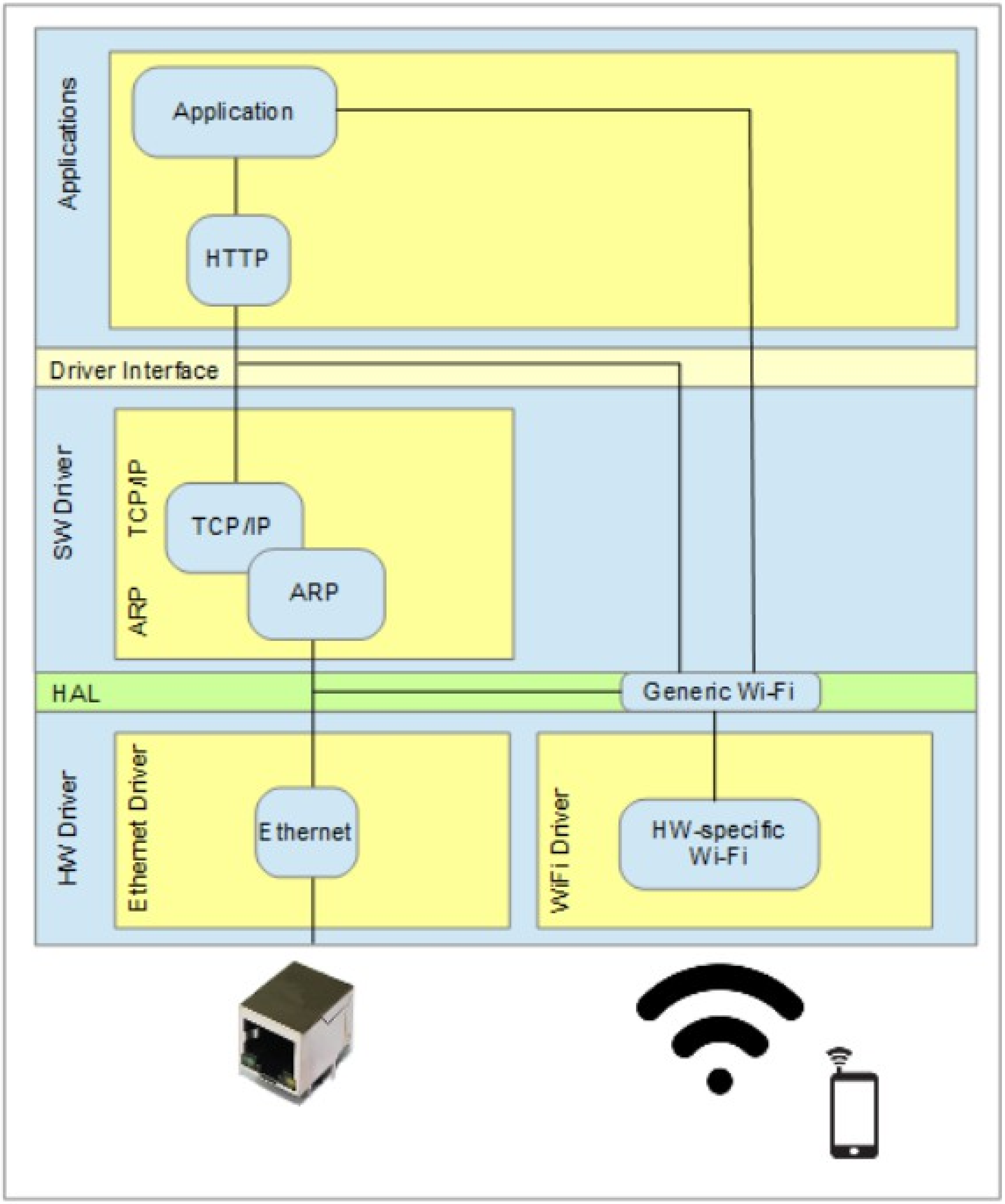
When a Wi-Fi module is integrated into a product it is to serve a particular purpose, which could be (for example) to allow a user to connect to the product (as Access Point) from a smart phone and configure, monitor or control it. The product's firmware will manage details such as the Wi-Fi's configuration (eg. SSID, security and passwords), command it to the appropriate mode and then subsequently interface with the user via the communication channel.

Although the requirement and actions are essentially the same no matter what Wi-Fi module is being used there is no standardised method of actually achieving it – each Wi-Fi module will require its own dedicated initialisation and will have its own specific command set to do the work, which means that typically the product's firmware needs to be developed exclusively for this Wi-Fi module and will have difficulties if another one needs to be moved to at any point.

The µTasker implementation adds a layer in the process which supplies a standardised interface for initialising and controlling the particular model used. This allows a product to be developed with little knowledge of the way that a particular module operates. If the Wi-Fi module is replaced – for example, because one with more performance of capabilities is needed, or the original one is no longer available – there is little or no impact on the firmware that was originally developed since the Wi-Fi interface - as seen by the application - remains the same. The only action usually required is to define that a different Wi-Fi part is in use and rebuild the project with the new setting.

Furthermore, the µTasker implementation allows multiple Wi-Fi interfaces to be enabled in a project and only the ones that are physically detected are then used. It is thus even possible to develop an application that accepts one of several possible Wi-Fi modules or operates with more that one Wi-Fi module connected – each with its own interface.

The following diagram goves an overview of how the Generic Wi-F interface fits into the overall formware concept showing a a situation where there is an optional Ethernet connection as well as the Wi-Fi conection to make the relationshf between Wi-Fi and other IP interfaces clearer.



The Generic Wi-Fi interface sits between the HW-specific driver and the TCP/IP stack so that raw frames can be routed in a similar fashion to an Ethernet interface.

The Generic Wi-Fi interface can also be controlled by the application in a generic manner. For example a Wi-Fi module will generally be inactive when a product starts operating and the application will give the Wi-Fi module instructions as to what mode of operation is should assume and what parameters it should use.

The Generic Wi-Fi interface also has an optional interface to the services above the TCP/IP stack. This is not needed when the Wi-Fi module is used in raw frame mode (also known as Ethernet or bypass mode) but is needed in cases when either the Wi-Fi module doesn't offer this capability or has been intentionally set to this mode by the application configuration. The Wi-Fi module will be handling the TCP/IP layer and so will be handling traffic to the socket layer instead.

2. Basic Wifi Terms

SSID = Service Set Identifier (name of the individual network)

AP = Access-Point, allowing clients to connect to it (sometimes referred to as a hot spot)

L1 = Layer 1 (PHY layer) is the physical layer of the OSI model

L2 = Layer 2 (MAC layer) is the data link layer of the OSI model

L3 = Layer 3 (Network layer) is the data network layer of the OSI model

M2M = machine-to-machine

P2P = peer-to-peer

Station Mode = Client mode that can connect o the Access Points

WEP = Wired Equivalent Privacy , based on 40 or 104 bit keys (WEP-40/WEP-104) – since 2004 depreciated and superseded by WPA

WPA = Wi-Fi Protected Access (with further developed WPA2 and WPA3) which supersedes the weaker WEP security standard

WWAN = Wireless Wide Area Network

2.1 Wi-Fi Frequencies

Various Wi-Fi specifications regulate the use of Wi-Fi at bands of 900MHz, 2.4GHz, 3.65 GHz, 5.0 GHz, 5.9 GHz, 6 GHz and 60GHz. 2.4 GHz seems to be the most common and dual-band Wi-Fis tends to use 2.4 GHz and 5.0 GHz and so these two bands are discussed in more detail here.

2.4GHz (802.11b/g/n/ax) has 14 channels starting at 2.412 GHz for channel 1 and ending at 2.484 GHz for channel 14. Each channel is spaces 5MHz above the previous channel with the exception of channel 14, which is spaced at 12MHz above channel 13's frequency of 2.472 GHz.

1 – 2.412 GHz
2 – 2.417 GHz
3 – 2.422 GHz
4 – 2.427 GHz
5 – 2.432 GHz
6 – 2.437 GHz
7 – 2.442 GHz
8 – 2.447 GHz
9 - 2.452 GHz
10 – 2.457 GHz
11 – 2.462 GHz
12 – 2.467 GHz
13 – 2.472 GHz
14 – 2.284 GHz

One should be aware that the use of channel 14 is not permitted in many countries and the use of channels 12 and 13 are also not permitted in North America, unless operating under low power conditions. *Sticking to channels 1 to 12 thus should ensure conformity around the world*.

5.0 GHz (802.11a/h/j/n/ac/ax) has a larger number of channels than 2.4 GHz. Not all possible channels may be used since their frequency range is reserved for public safety entities and certain channels are restricted in various countries, or require a special registration. For this reason 5.0 GHz Wi-Fi modules may restrict use of the 5.0 GHz range to those that are permitted in most countries, which are
36 – 5.180 GHz
40 – 5.200 GHz
44 – 5.220 GHz
48 – 5.240 GHz
52 – 5.260 GHz
56 – 5.280 GHz
60 – 5.300 GHz
64 – 5.320 GHz
100 – 5.500 GHz
104 – 5.520 GHz
108 – 5.540 GHz
112 – 5.560 GHz
116 – 5.580 GHz
132 – 5.660 GHz
136 – 5.680 GHz
140 – 5.700 GHz

149 – 5.745 GHz
153 – 5.765 GHz
157 – 5.785 GHz
161 – 5.805 GHz
165 – 5.825 GHz

However one should still consult which ones may be used in each country to be sure since those above 64 are still restricted in some regions, such as Russia and Israel, for example. 5.0 GHz Wi-Fi tends to be restricted to indoors use in many countries.

Note that if the define `RESTRICT_WIFI_CHANNELS` is set in the µTasker project only the Wi-Fi channels that are allows in every country will be allowed to be set, which ensures that no breach of regulations can take place

3. ATWINC1500 and ATWINC3400

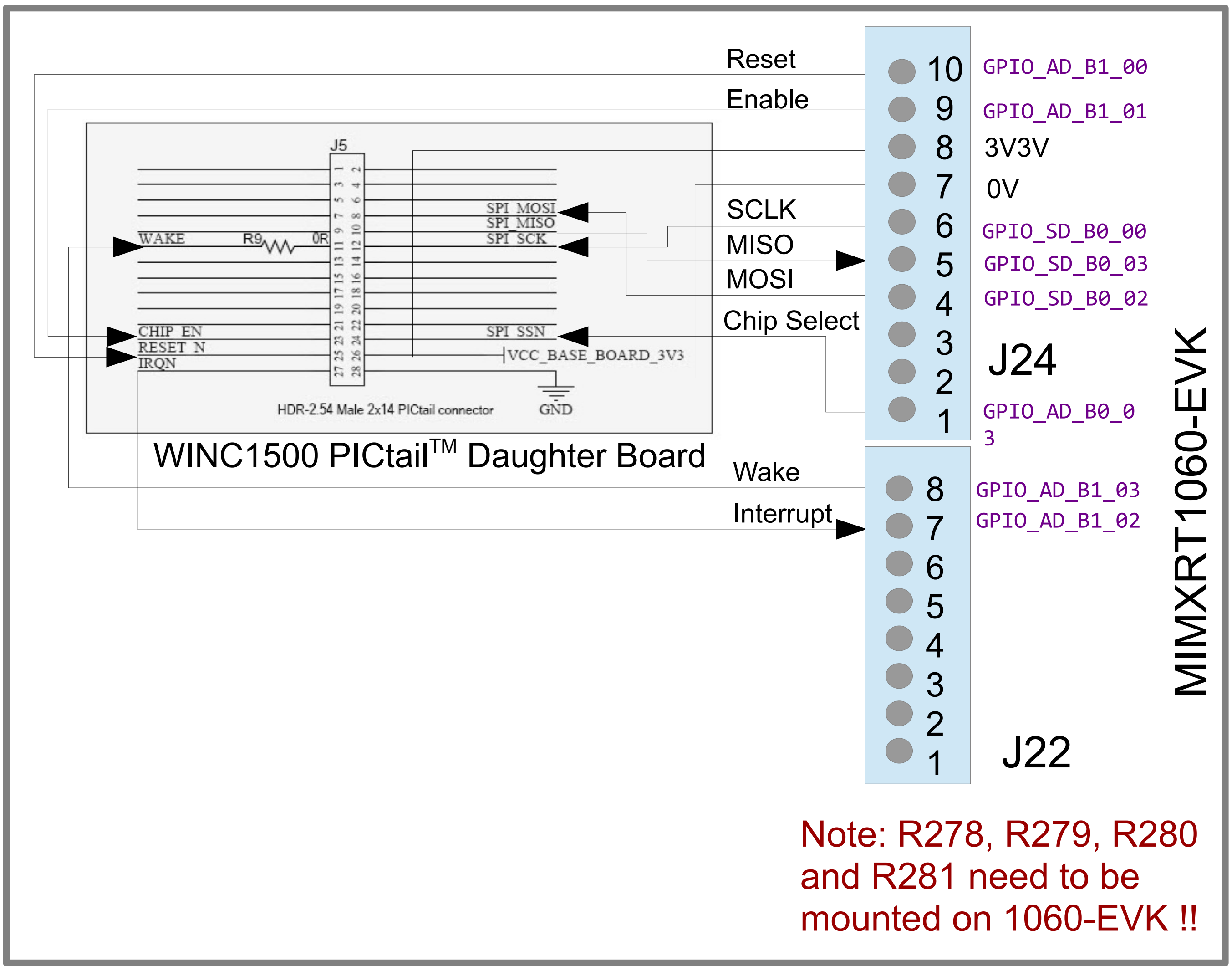
The ATWINC1500/ATWINC3400 can operate as either Access Point, P2P or in Station Mode, with a single connection; they can't perform both AP and stations mode at the same time.

Ethernet (bypass) mode is available making it very flexible if the µTasker TCP/IP stack is to be used instead of its internal socket modes.

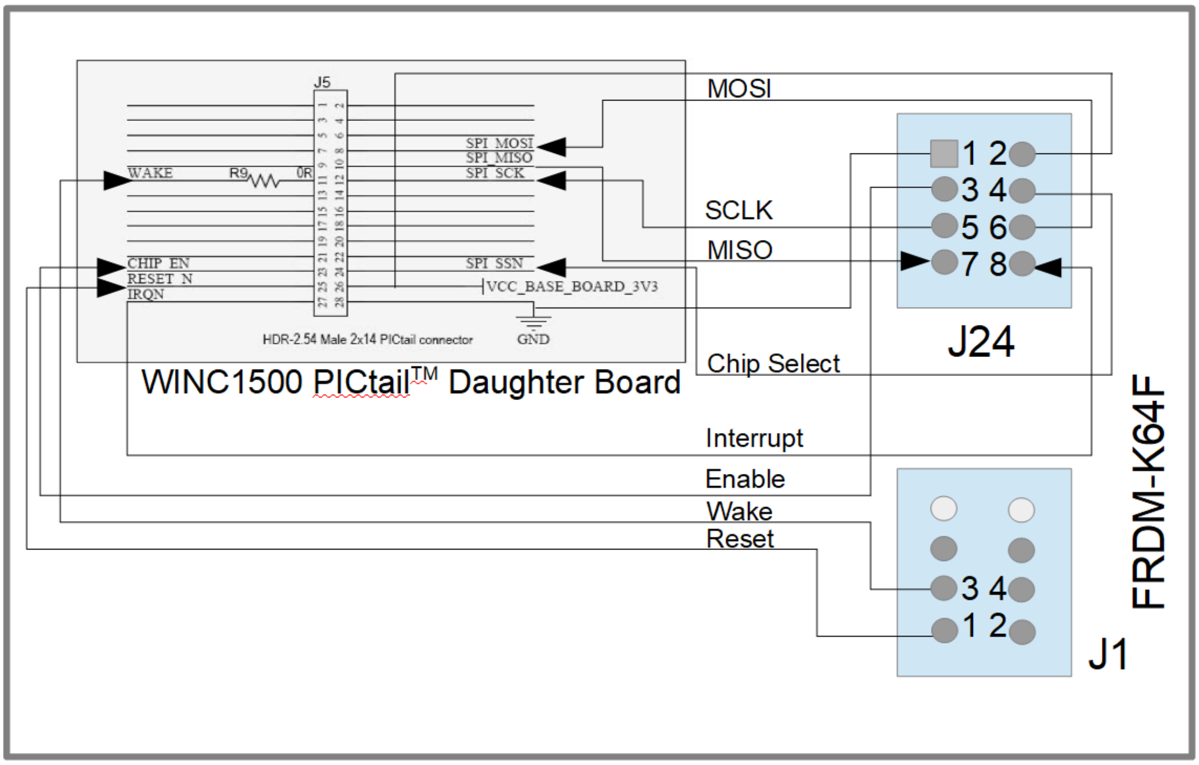
The Wi-Fi interface in the ATWINC3400 is highly compatible to that of the ATWINC1500, but the ATWINC3400 has in addition Bluetooth 5. The Wi-Fi interface library used does however need to be adjusted to suit the device and it is not possible to use the same for both.

Wi-Fi modes supported are IEEE 802.11 b/g/n 20MHz

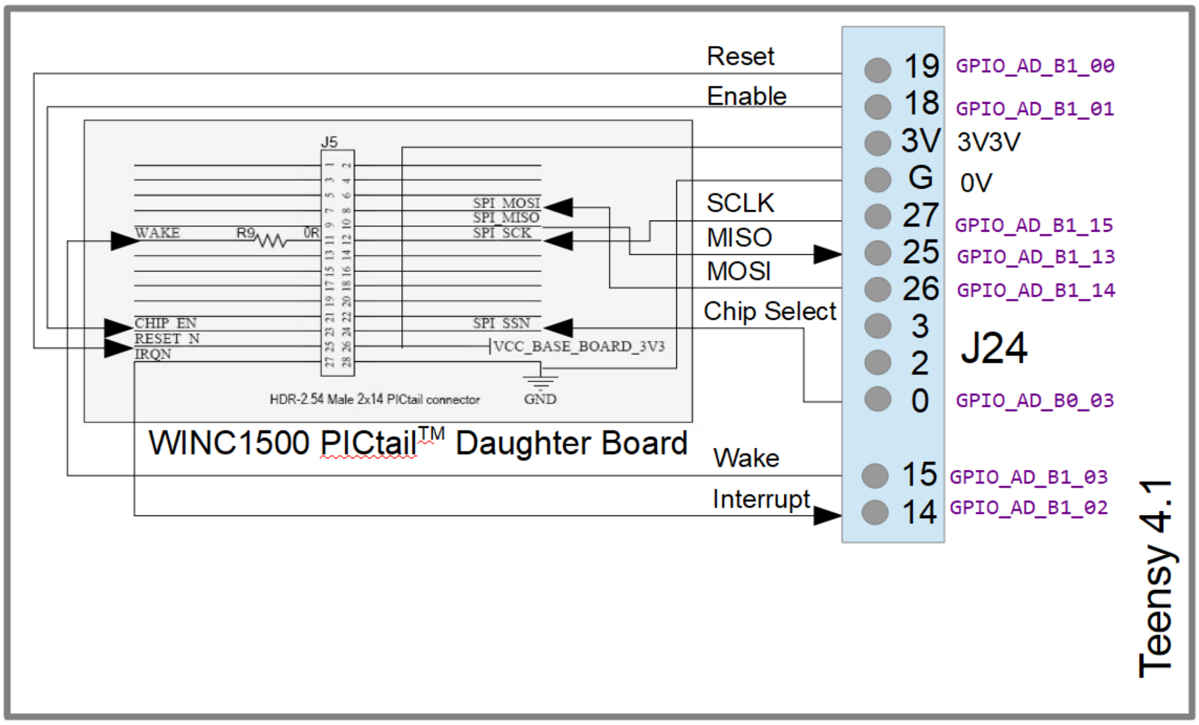
3.1 Connecting an i.MX RT 1060 EVK to a Microchip WINC1500 PICtail^{TM} Daughter Board



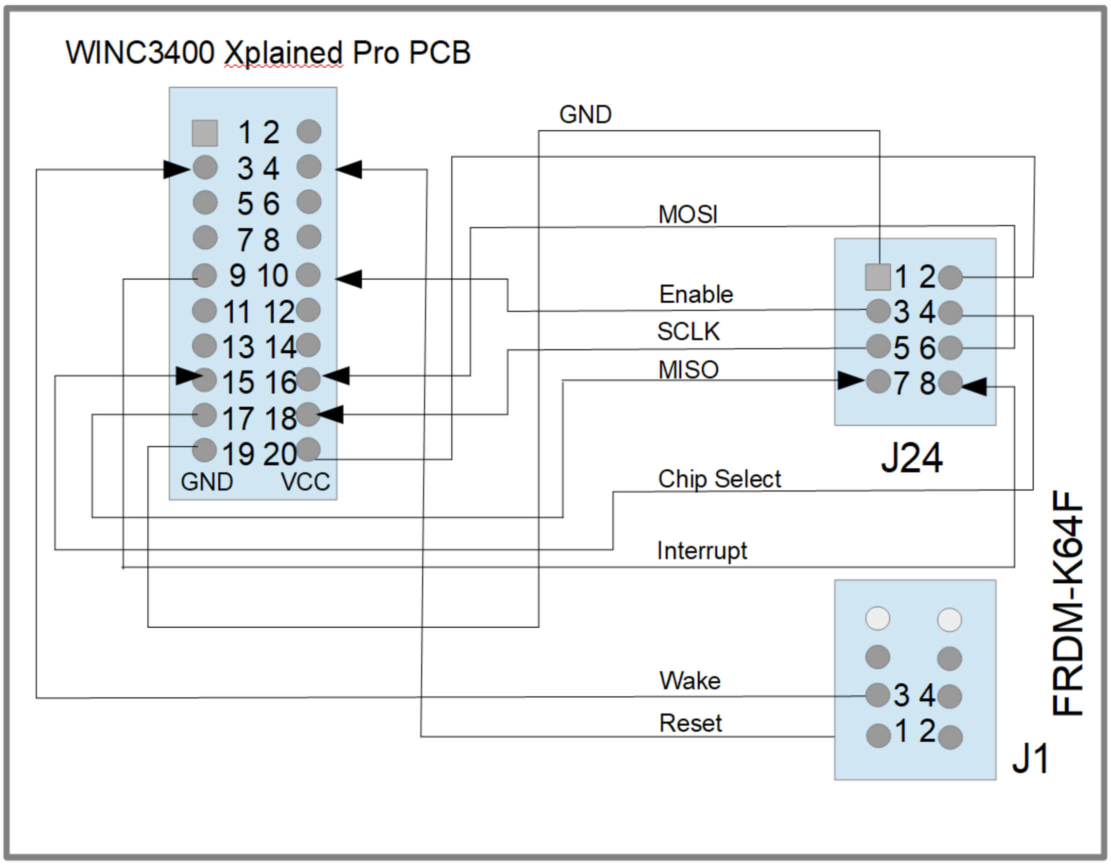
3.2 Connecting a Kinetis FRDM-K64F to a Microchip WINC1500 PICtail^{TM} Daughter Board



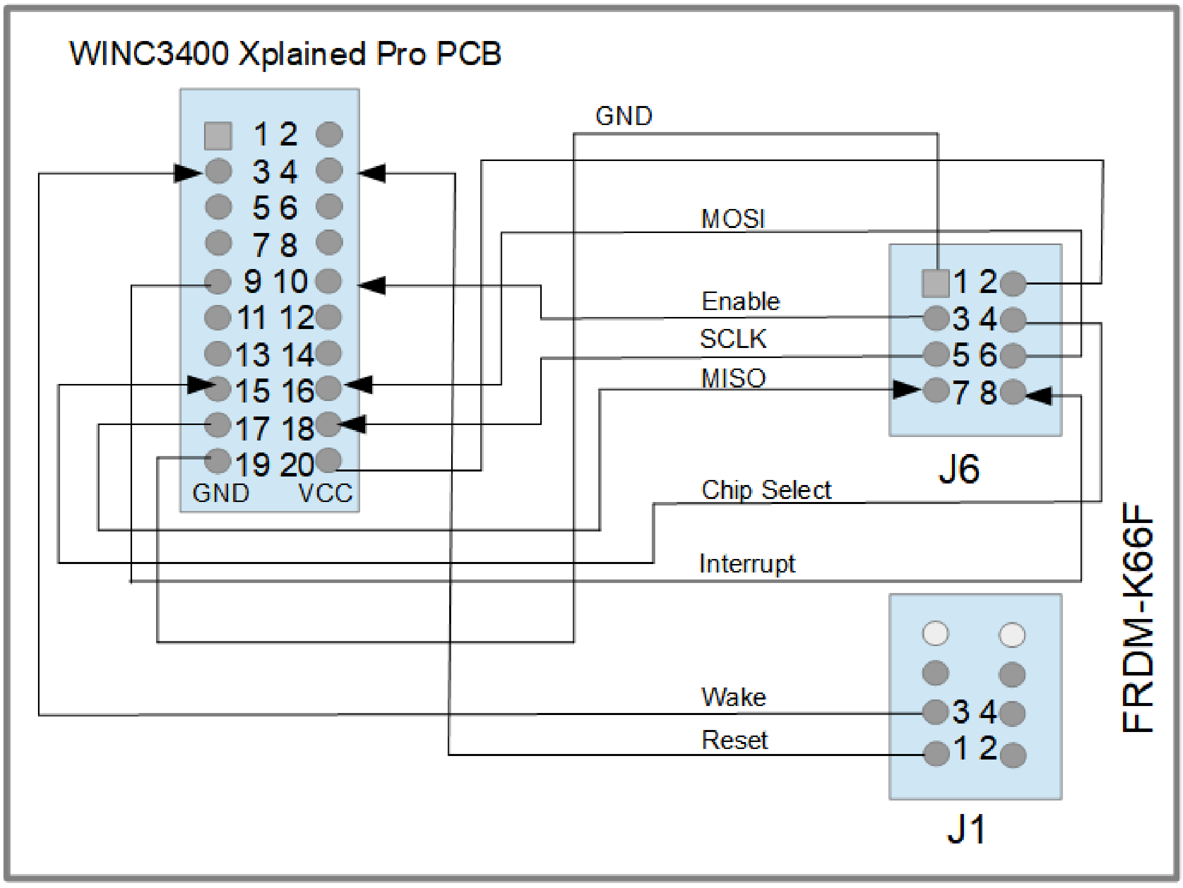
3.3 Connecting a TEENSY 4.1 to a Microchip WINC1500 PICtail^{TM} Daughter Board



3.4 Connecting a Kinetis FRDM-K64F to a Microchip 3400 Xplained Pro PCB Daughter Board



3.5 Connecting a Kinetis FRDM-K66F to a Microchip 3400 Xplained Pro PCB Daughter Board



3.6 Firmware Updating

The ATWINC1500 supports firmware updates via SPI or UART via its integrated serial flash memory download procedure. The ATWINC3400 supports firmware updates via SPI via its integrated serial flash memory download procedure – *at the time of writing the ATWINC3400 does not support loading using the UART method*.

Microchip integrates the uploading methods and accompanying binary files and utilities in its Atmel Studio which requires this to be installed and appropriate ASF projects to be used to obtain these files. The ATWINC15xx can then be programmed with any USB to serial cable but it is generally necessary to also purchase a reference development board in order to be able be update firmware to an ATWINC34xx. The SAMD21 Xplained Pro development board and WINC3400 Xplained Pro module are used as reference for the following, where it is ATWINC34xx specific.

The general official guide can be found at:

https://ww1.microchip.com/downloads/en/DeviceDoc/00002378B.pdf

As detailed in the guide Atmel Studio must first be installed in order to be able to start, whereby this is now called Microchip Studio, and the version used for the following reference was 7.0.2542 from 01.11.2020. The following attempts to clear up various steps that may be otherwise confusing to users without prior experience with the Microchip environment – *see the following section for details of how the µTasker can allow ATWINC34xx updating in the circuit*:

1. Download and install Microchip studio, which can be easily found on the Microchip website. During the installation you can select the processors that are to be supported so either choose all or the one on any evaluation board that will be used. Ensure that the ASF (Advanced Software Framework) and Example Projects extensions are also selected when installing. A reboot may be required during the installation process.
2. Start a new example project in the menu `File | New | Example Project...` and once this has opened expand the Atmel menu. Beware that this may take some time to open so be patient if it looks initially that it is not working. Then search for the appropriate WINCxx firmware update project, which can be found either in "All Projects" or in a specific evaluation board's project list. If no evaluation board is to be used (UART loading only) select any processor for the project (eg. "`WINC3400 Firmware Update Project (v1.3.1) - SAMD21 Xplained Pro`").
   This will cause a project solution to be created containing the required files and any other resources.
   Build the solution (F7 can be used) to ensure that anything that needed to be compiled or generated is available.
3. If using an evaluation board that acts as a serial bridge from the PC to the ATWINC module the easiest way to ensure the bridge is correctly loaded to the evaluation board is to simple run the project (F5) in Microchip Studio, which loads the code to the evaluation board's processor and allows it to execute, from which point the loading mechanism can be started.
4. Open a DOS (command) window and move to the location of the project which can be easily found by opening the bat file in the solution explorer (in SRC folder) and looking where its source is located. It will be something like
   `C:\Users\Name\Documents\Atmel Studio\7.0\WINC3400_FIRMWARE_UPDATE_PROJECT1\WINC3400_FIRMWARE_UPDATE_PROJECT1\src`
   where the bat file will be found (eg.

samd21_xplained_pro_firmware_update.bat).
In the case of UART loading the bat file to be used is download_all.bat in the lower directory "firmware".
The update is performed by executing the bat file.

5. Don't panic if there is an initial error stating that Python is not found, eg.
```
Checking for Python support.
Python 3.7.3
Require Python v2.x
```

   Also not if Python is not found or not installed at all, with similar error message.

   This is because there is a newer version found than expected (and the used Python syntax is not compatible) (or nothing found). This part of the process is using Python to check the Microchip Studio version and location and, since it has just been installed, this step is not really required.
   To work around this typical problem, open the .bat file from the solution explorer window in the project and delete the lines between
   `echo Checking for Python support.`
   And
   `:HASAS`
   (including the first line but not including the last line)

   Now create a text file atmelstudiopath.txt in the directory where the bat file is located and add a path to it where the Atmel Studio is installed, eg. "C:\Program Files (x86)\Atmel\Studio\7.0\AtmelStudio.exe" (note that the path needs to be in " ").

   Then make an identical copy this file in the sub-directory „firmware" so that it is at both locations.

   Finally edit the file „firmware\download_all_sb.bat" in the same way as the first bat file so that it also doesn't need to do its checking using python.

   Now it will work (on first use it may ask whether a program called atprogram.exe is allowed to execute, whereby this can be acepted), but some patience is still needed since it may take a few minutes to do all of the work of loading and checking various data files.

For ATWINC1500 users who wish to load without an intermediate evaluation board via its UART interface the following guide, taken from a forum, has proven to work, whereby the control inputs can be controlled by hand as there is no specific timing to be respected:

```
Firmware Upload without MCU or directly with PCs UART interface:
--------------------------------------------------------------------
----------------------
Please follow the steps to download the firmware without any MCU:
1. ATINC1500 UART interface pins Rx, TX, and GND to be connected
to PC. Since ATWINC1500 UART is TTL level output, please take care
about the RS232 converter to connect to the PCs COM port.
```

```
2. RESET_N and CHIP_EN pins to connected to GND to make it low
while ATWINC1500 power up.

3. Power up the ATWINC1500 module with VCC 3.3 V and GND.

4. After the power up, CHIP_EN pin to be connected VCC 3.3V to
make it high.

5. RESET_N pin to be connected to be VCC 3.3 V to make it high.

6. Once above mentioned procedure is done, Open the command prompt
with folder path "WINC1500_FIRMWARE_UPDATE\src\firmware" folder.

7. run the command "download_all.bat UART", you can see the
firmware update process in the window. First it will detect the
COM port and will begin the download of the firmware. [Actually
the command requires additional parameters in order for it to be
accepted: „download_all.bat UART SAMD21 3A0 0 0‟ can be used
generally]
8. After completion of firmware download, open
"WINC1500_FIRMWARE_UPDATE\src\firmware\Tools\root_certificate_down
loader\debug_uart" from cmd prompt, to download the root
certificate to the ATWINC1500.
Note: RESET and CHIP_EN pins, low and high process important in
this procedure.

Perform the following power up sequence on CHIPEN and RESETN:
- pio_set_pin_low(CONF_WINC_PIN_CHIP_ENABLE);
- pio_set_pin_low(CONF_WINC_PIN_RESET);
- nm_bsp_sleep(100);
- Power up the board (Vcc-Gnd)
- pio_set_pin_high(CONF_WINC_PIN_CHIP_ENABLE);
- nm_bsp_sleep(100);
- pio_set_pin_high(CONF_WINC_PIN_RESET);
- nm_bsp_sleep(100);
- Run the "download_all.bat UART" command in the firmware folder
```

Note that this is only possible with the ATWINC15xx (at the time of writing) since the ATWINC34xx doesn't support loading via its UART interface.

3.7 In-Circuit Updating of ATWINC34xx via Bridge

As detailed in the previous section, the ATWINC34xx can be updated in the circuit via its SPI interface as log as the controlling processor can perform a bridging function to a PC running Microchip's updating software. Such a bridging function is included in the µTaske project when the ATWINC34xx is used when the option `WIFI_ATWINC34_USB_LOADER` is enabled.

This can operate over any serial connection (like UART, Telnet or USB-CDC) whereas in the USB device case an example is to use two VCOM connections whereby the first is the command line interface and the second is a dedicated ATWINC34xx bridge – this is descried here are reference.

When the processor is connected as a device to a PC host it appears as two COM ports. The first is used as the traditional debug interface. The second can be connected to b the PC host as a VCOM port and will respond with "`ATWINC34xx Loader`" each time the enter key is pressed. This is useful for identifying the connect COM port that is in use since this is needed to be known during the actual programming step.

The latest version of the recommended ATWINC34xx firmware is contained in the uTasker repository in the directory `\stack\WiFi\ATWINC15x0`. The recommended version at the time of writing is V1.3.1 and therefore a zip file called **`ATWINC3400_V1.1.3.zip`** is located there. When required this can be extracted and a DOS windows opened at the extracted location – eg. `\stack\WiFi\ATWINC15x0\ATWINC3400_V1.1.3`

In the DOS command line shell, knowing the COM port that the loading interface is on (eg. COM7) the update can be started by commanding

`"ATWINC_Prog COM18"`

which executes the local bat file, which calls Microchip's utility `winc_programmer_UART.exe`.

This program opens COM18 (must be connected of course) and detects that the processor supports the bridging protocol. The processor itself sets the ATWINC34xx into its programming mode.

The Microchip utility then loads a small programming binary file (2k in size) to the Wi-Fi module (`programmer_firmware.bin`) and then programs the module's firmware (`m2m_image_3400_V1.3.1.bin` – 353k in size).

The loading take abut 35s to complete when used on a HS USB interface and the complete output from the Microchip utility is shown below (also showing the complete command that is controlled by the bat file, remembering that the COM port reference is passed as a parameter when the bat file is commanded):

```
winc_programmer_uart.exe  -p \\.\COM18 -d winc3400 -i "m2m_image_3400_V1.3.1.bin"
-if prog -w -r -pfw programmer_firmware.bin
WINC Programming Tool 1.0.2 [r671] (Mar 25 2019)
Copyright (C) Microchip Technology Inc. 2019

software WINC serial bridge found, baud rate changes supported
chip ID is 0x003400d2
programming firmware file: programmer_firmware.bin
starting device
reinitialise serial bridge to 500000
waiting for firmware to run
flash ID 0x001440ef
flash size is 8 Mb

begin write operation
```

```
 0x000000:[ww.wwww.] 0x008000:[ew.wwwww] 0x010000:[wwwwwwww] 0x018000:[wwwwwwww]
 0x020000:[wwwwwwww] 0x028000:[wwwwwwww] 0x030000:[wwwwwwww] 0x038000:[wwwwwwww]
 0x040000:[w.......] 0x048000:[........] 0x050000:[.....www] 0x058000:[wwwwwwww]
 0x060000:[wwwwwwww] 0x068000:[wwwwwwww] 0x070000:[wwwwwww.] 0x078000:[.......e]
 0x080000:[eeeeeeee] 0x088000:[eeeeeeee] 0x090000:[eeeeeeee] 0x098000:[eeeeeeee]
 0x0a0000:[eeeeeeee] 0x0a8000:[eeeeeeee] 0x0b0000:[eeeeeeee] 0x0b8000:[eeeeeeee]
 0x0c0000:[eeeeeeee] 0x0c8000:[eeeeeeee] 0x0d0000:[eeeeeeee] 0x0d8000:[eeeeeeee]
 0x0e0000:[eeeeeeee] 0x0e8000:[eeeeeeee] 0x0f0000:[eeeeeeee] 0x0f8000:[eeeeeeee]

begin read operation

 0x000000:[rrrrrrrr] 0x008000:[rrrrrrrr] 0x010000:[rrrrrrrr] 0x018000:[rrrrrrrr]
 0x020000:[rrrrrrrr] 0x028000:[rrrrrrrr] 0x030000:[rrrrrrrr] 0x038000:[rrrrrrrr]
 0x040000:[rrrrrrrr] 0x048000:[rrrrrrrr] 0x050000:[rrrrrrrr] 0x058000:[rrrrrrrr]
 0x060000:[rrrrrrrr] 0x068000:[rrrrrrrr] 0x070000:[rrrrrrrr] 0x078000:[rrrrrrrr]
 0x080000:[rrrrrrrr] 0x088000:[rrrrrrrr] 0x090000:[rrrrrrrr] 0x098000:[rrrrrrrr]
 0x0a0000:[rrrrrrrr] 0x0a8000:[rrrrrrrr] 0x0b0000:[rrrrrrrr] 0x0b8000:[rrrrrrrr]
 0x0c0000:[rrrrrrrr] 0x0c8000:[rrrrrrrr] 0x0d0000:[rrrrrrrr] 0x0d8000:[rrrrrrrr]
 0x0e0000:[rrrrrrrr] 0x0e8000:[rrrrrrrr] 0x0f0000:[rrrrrrrr] 0x0f8000:[rrrrrrrr]

verify range 0x000000 to 0x100000
begin verify operation

 0x000000:[pp.vppp.] 0x008000:[.p.vvvvv] 0x010000:[vvvvvvvv] 0x018000:[vvvvvvvv]
 0x020000:[vvvvvvvv] 0x028000:[vvvvvvvv] 0x030000:[vvvvvvvv] 0x038000:[vvvvvvvv]
 0x040000:[p.......] 0x048000:[........] 0x050000:[.....vpv] 0x058000:[vvvvvvvv]
 0x060000:[vvvvvvvv] 0x068000:[vvvvvvvv] 0x070000:[vvvvvvp.] 0x078000:[........]
 0x080000:[........] 0x088000:[........] 0x090000:[........] 0x098000:[........]
 0x0a0000:[........] 0x0a8000:[........] 0x0b0000:[........] 0x0b8000:[........]
 0x0c0000:[........] 0x0c8000:[........] 0x0d0000:[........] 0x0d8000:[........]
 0x0e0000:[........] 0x0e8000:[........] 0x0f0000:[........] 0x0f8000:[........]

verify passed
```

If the ATWINC34x had been delivered with an older version it may have announced itself on the µTasker debug interface as:

ATWINC34x0: Chip ID 0x003400d2
Firmware 1.2.2
Driver 1.2.2
Oct 5 2017 13:22:37

It will subsequently announce itself (it is best to reset the board after loading) as:

ATWINC34x0: Chip ID 0x003400d2
Firmware 1.3.1
Driver 1.3.1
Jun 28 2019 13:46:26

The µTasker command line interface will display

```
ATWINC34xx Update Starting
ATWINC34xx Update Finished
```

The output in the DOS window gives adequate information concerning the present activity and in case of errors.

4. ESP32-S2

Since the ESP32-S2 uses an open programming concept it can be used in various ways depending on which firmware is loaded to it. Since anyone can configure and change this firmware it is not limited to presently available methods but can also be extended.

Due to this fact it is the most flexible solution when its radio frequency hardware fulfils the product's requirements but may have higher maintenance (and learning curve) when its firmware also needs to be modified and managed.

This module has an integrated boot loader which supports loading new code over the serial interface.

Some basic use choices are:

1. Use its ESP-AT mode of operation, which means installing the "standard" ESP-AT firmware and control it via AT commands. *Users sill have the ability to extend the AT command set if needed.*
2. Use a reference application (either from ESP examples or from an ESP user who makes it available) which corresponds to the product's requirements and allows things to be done in a more efficient or specific manner than the ESP-AT methods. This may allow functions to be realised that cannot be achieved using the standard ESP-AT command set.
3. Develop a custom version of the Wi-Fi implementation in the ESP in order to achieve the exact control needed or to integrate parts or all of the application in the ESP module as well. *Since the ESP32-S2 is a single core module (there are others that have multi-cores) it is recommended that this module is primarily used for Wi-Fi interface offloading and not for full application integration.*

4.1 ESP-AT Mode of Operation

When the standard ESP-AT firmware is installed UART1 of the ESP32-S2 is used for communication in AT mode. The default Baud rate is 115200 Baud but it can be modified and saved to operate up to 5 MBaud.

The AT command set is constructed from three main types of commands:
- Basic AT command: used to control and configure the general operation
- Wi-Fi commands: used to configure and control general Wi-Fi operations
- TCP/IP commands: used to configure and control TCP/IP operations that are performed over the Wi-Fi connection

Settings are saved either automatically or on command in the ESP32-S2 module itself.

Station mode, softAP mode and station and softAP modes are supported. In softAP mode up to 10 clients can be connected and the encryption methods WPA-PSK, WPA2-PSK and WPA-WPA2-PSK (or none) are supported (but not WEP).

Protocols IEEE 802.11b, IEEE 802.11bg or IEEE 802.11bgn protocol standards in station mode are supported.

The ESP-AT firmware package is available on GITHUB at https://github.com/espressif/esp-at

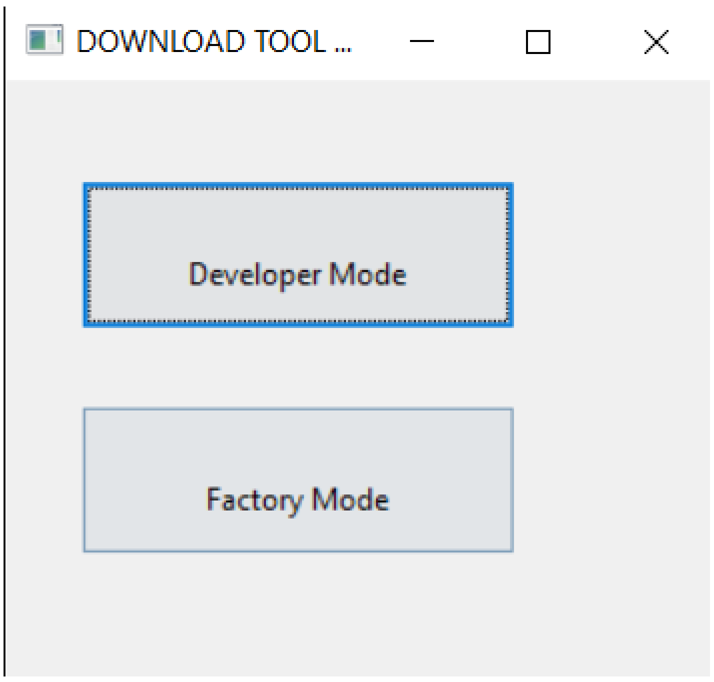
Pre-build ESP32-S2-AT binary files can be obtained from https://docs.espressif.com/projects/esp-at/en/latest/AT_Binary_Lists/ESP32-S2_AT_binaries.html

whereby the version V2.1.0.0 is was used when developing this document.

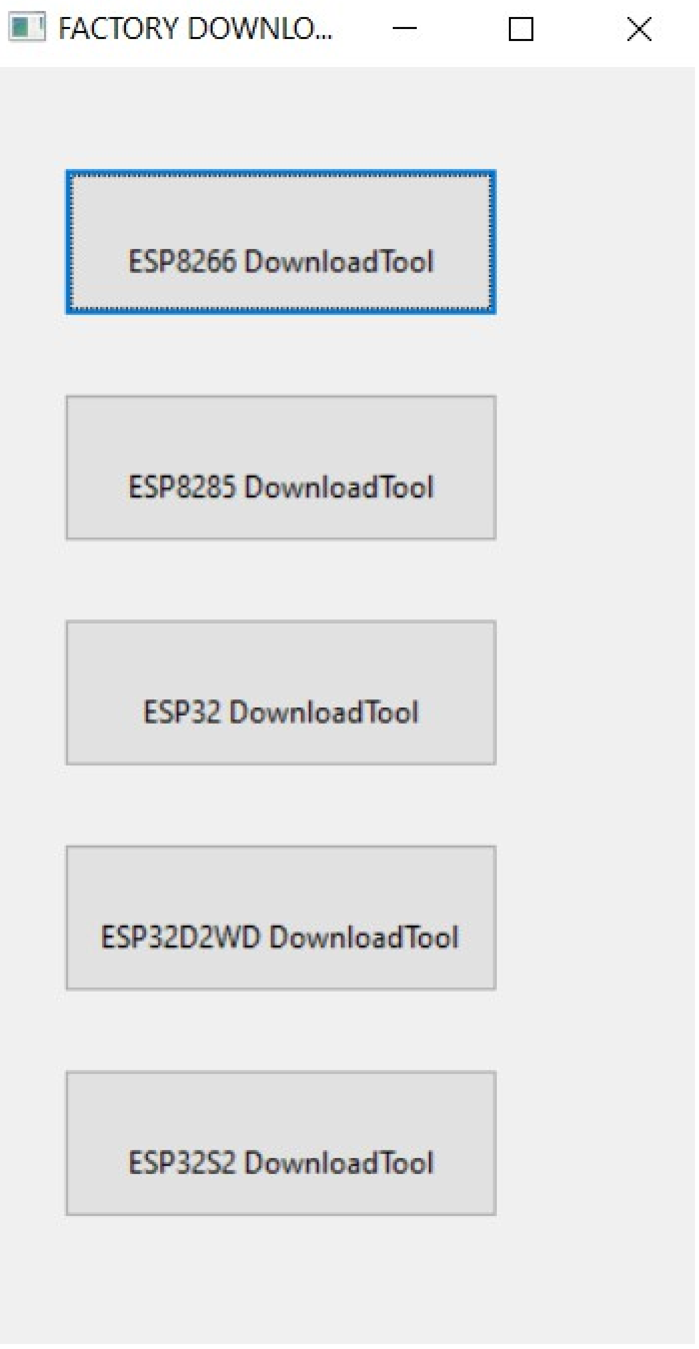
Generally the complete on-line guide and ESP-AT documents can be found at https://docs.espressif.com/projects/esp-at/en/latest/

Tools required for flashing the binary to the device can be downloaded from this site and the "`flash_downoad_tool_3.8.5.exe`" was used as follows to ensure that the latest version was programmed to the module:

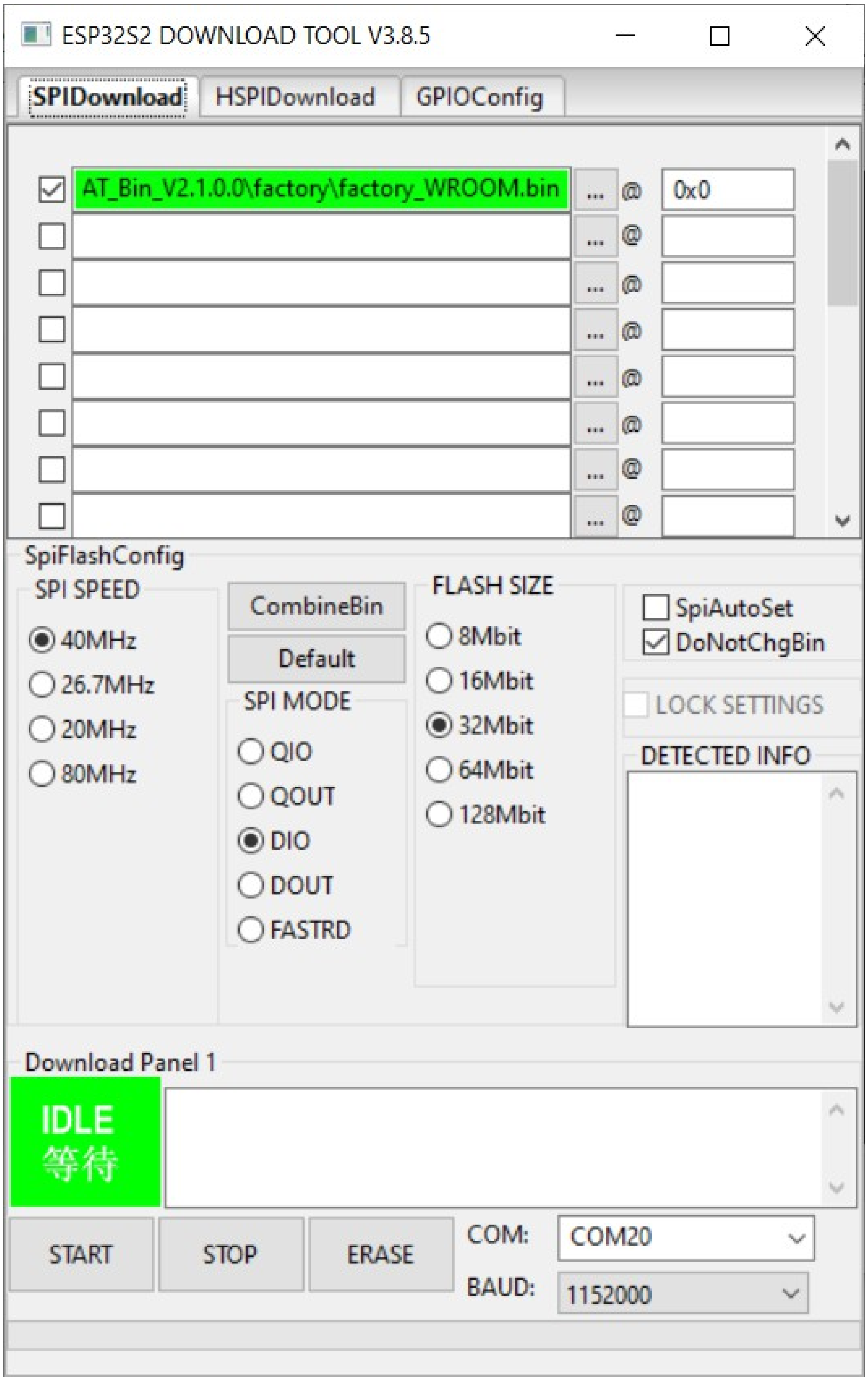
1. Start the utility (it can take a little time to open), after which the following is seen:



2. Select "Developer Mode"
and the "ESP32S2 Download Tool"



3. Leave the SPI Download Mode and select the binary file to be loaded, whereby the `factory_WROOM.bin` is being used since it contains all binary parts that are needed in one go. The load address is 0x0.
Make sure that the file has its Tick box to the left of it so that it is actually programmed during the process!


ESP32S2 DOWNLOAD TOOL V3.8.5

SPIDownload | HSPIDownload | GPIOConfig

☑ AT_Bin_V2.1.0.0\factory\factory_WROOM.bin ... @ 0x0

SpiFlashConfig

SPI SPEED: [✓ 40MHz] 26.7MHz 20MHz 80MHz

CombineBin | Default

SPI MODE: QIO QOUT [✓ DIO] DOUT FASTRD

FLASH SIZE: 8Mbit 16Mbit [✓ 32Mbit] 64Mbit 128Mbit

☐ SpiAutoSet
☑ DoNotChgBin
☐ LOCK SETTINGS

DETECTED INFO

Download Panel 1

IDLE 等待

START | STOP | ERASE

COM: COM20

BAUD: 1152000

Make sure that the COM port matches that on which the connected ESP32-S2 module enumerates as ("Silicon Labs CP210x USB to UART Bridge) and start the loading process by clicking on "START".

In the shell that is also started by the programming utility there are a few messages:

```
test offset :  0 0x0
case ok
test offset :  0 0x0
case ok
======
CONNECT BAUD: 115200
============
.Uploading stub...
Running stub...
Stub running...
Changing baud rate to 1152000
Changed.

 is stub and send flash finish
```

and the programming is completed after about 1 minute.

5. Telit WL865E4-P

The dual-band WL865E4 can support both AP and station modes at the same time. Its Wi-Fi protocol supports 802.11a/b/g/n.

At the time of writing this Wi-Fi module doesn't offer a RAW Ethernet mode of operation.

The Telit module operates essentially always via AT commands (whether connected via UART, SPI or SDHC interface) but has two AT style commands; *GainSpan or Telit*

The µTasker implementation uses the the **Telit style AT** commands – to ensure that this mode is configured the command

`AT+YLC=0` can be used. If it responds with `ERROR` it means that it is already in this mode. If it was previously in the GainSpan mode it will respond with `OK` after a delay of about 4s and then restart in the new mode, with start-up message "`Serial2Wireless APP`". This same start-up message is received after every reset of the module.

The firmware version loaded in the module can be requested using the AT command `AT+YVER`
`+YVER:"36.07.002-B008"` is the response from the module that was used for integration. The version delivered in the module on the board (`"36.07.000")` would originally not respond to any SPI commands.

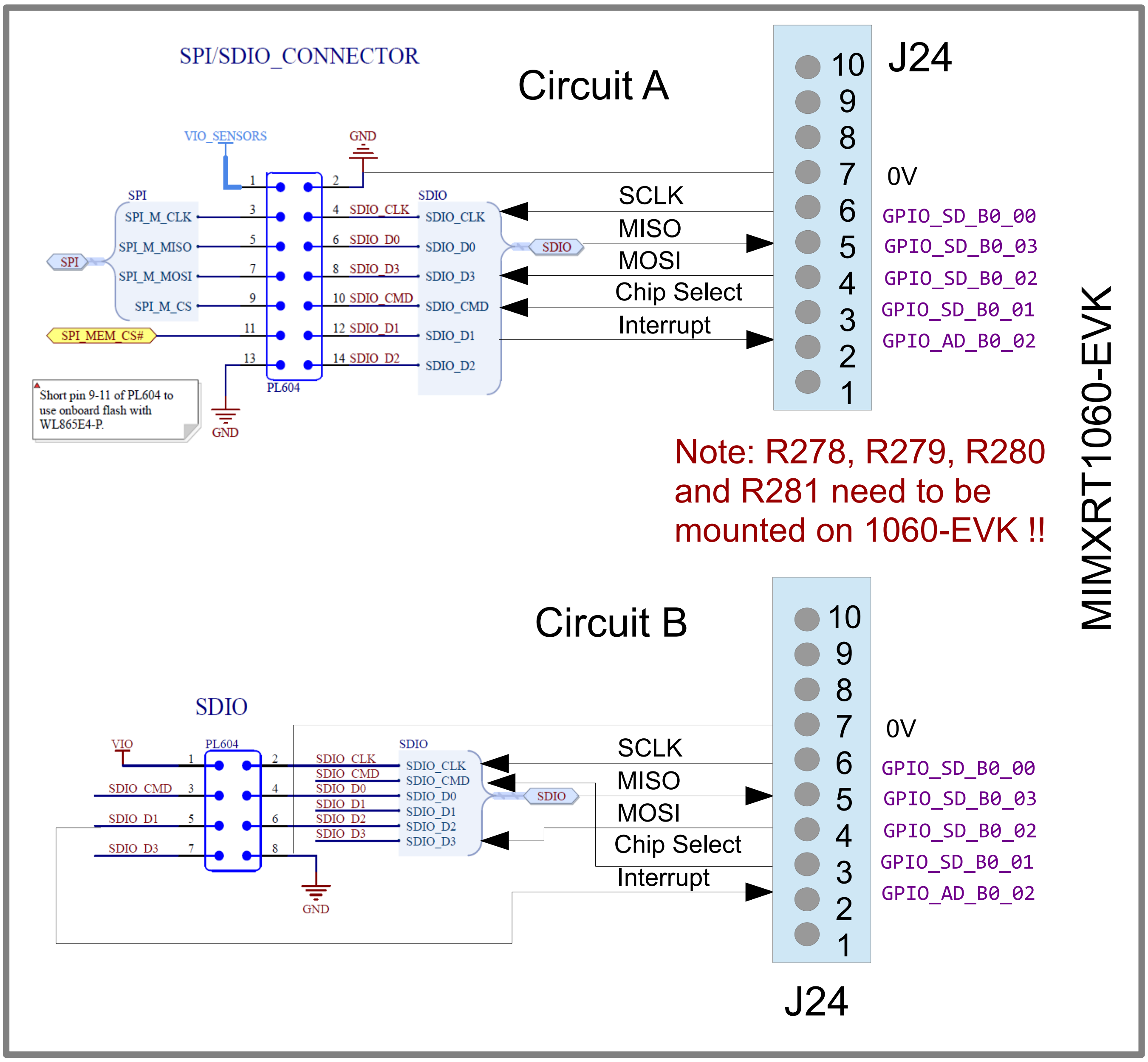
If new firmware needs to be loaded it is performed via the USB interface of the module, once the module has been placed in its loader mode of operation. The Telit document "**Flashing new Firmware in WL865E4 Module**" can be consulted for a guide to programming using their loader program "`WL865E4 Loader.exe`". New firmware versions can be obtained from Telit.

The µTasker implementation uses the SPI interface and this is configured once by commanding `AT+YSIF=1`. If the mode is already valid it will respond with `ERROR`, else with OK.

Since the modules may require the initial configurations to be performed over the default UART interface it is advised that they are configured once when received so that they are ready for subsequent use via the SPI interface.

5.1 Connecting an i.MX RT 1060 EVK to a Telit WL865E4 EVK

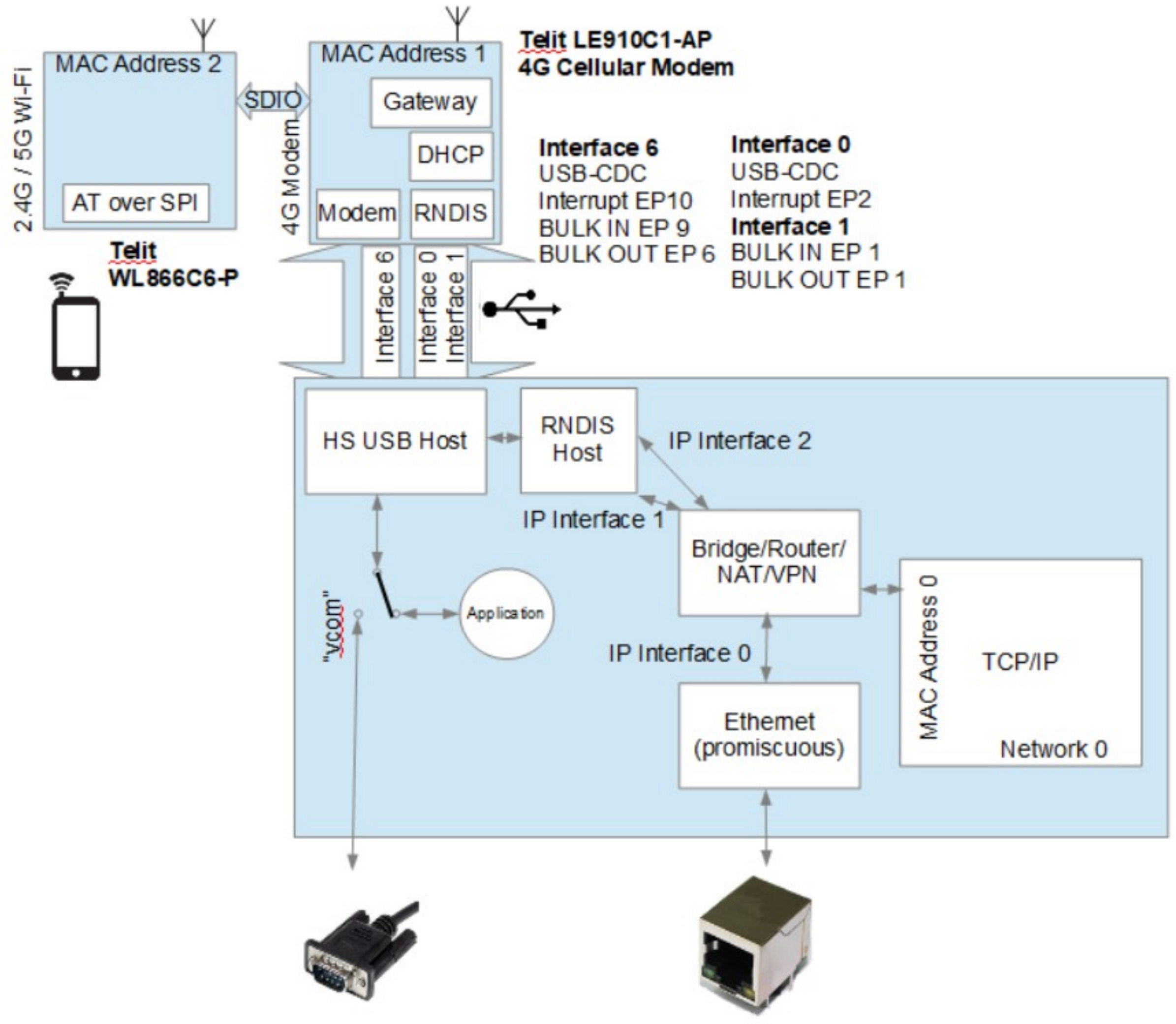
Beware that there are more than one version of the Telit EVK and the have very different layouts and pin connections. Two types have been used and both connections are shown in the hope that one will match the board in use. Furthermore it is to be pointed out that the board have connectors with SPI connections marked on then which have nothing to do with the SPI connection used for communicating with the module – the SPI interface used for communication is in fact the SPI mode of the SDIO interface!!



6. Telit WL866C6-P with Cellular Modem

The dual-band WL866C6-P Wi-Fi protocol supports 802.11a/b/g/n/ac. This is not a stand-alone module but instead is bundled with a Linux companion module such as the LE910C1-AP 4G cellular modem, which is described here.

The embedded processor communicates with the cellular/Wi-Fi combination via USB whereby the embedded processor is a USB host.



As show in the diagram the LE910C1-AP is the master of the bundled modules and interfaces with the USB host. The USB connection incorporates an RNDIS interface for IP traffic and a modem connection for control, using AT commands. The Wi-Fi module is connected to the cellular module via an SDIO and Wi-Fi and cellular commands are possible via the single modem interface. Usually the cellular modem performs a DHCP server function to supply the IP settings for the local network, whereby its own IP address is the network's gateway, allowing Wi-Fi clients and also the embedded system (supplying the USB RNDIS host connection) to connect to the Internet via the cellular connection. When the embedded system is acting as an Ethernet bridge also the nodes on the Ethernet network can access the Internet via the cellular connection.

7. Working with the Wi-Fi Interface

The Wi-Fi interface in the project is enabled by enabling the define (in `config.h`)

```
#define WIFI_INTERFACE                                      // enable wifi interface(s)
```

One or more Wi-Fi module types can be enabled (in `config.h`)

```
#define WIFI_ATWINC15                                // use Microchip ATWINC15x0 module
#define WIFI_WL865E4_P    // use Telit WL865E4-P dual-band (2.6GHz/5GHz) wifi module
#define WIFI_ESP32_S2_WROVER                         // use ESP32-S2_WROVER wifi module
#define WIFI_ESP32_S2_WROOM                          // use ESP32-S2_WROOM wifi module
#if defined TELIT_LE910C1
    #define WIFI_WL866C6_P                // use Telit WL866C6-P dual-band (2.6GHz/5GHz)
                                               wifi module bundled with LE910C1 cellular
#endif
```

The Wi-Fi interface (or multiple) will be initialised when the Ethernet task starts (or by the USB task if they communicate via RNDIS), whereby each will have a handle for subsequent Ethernet type communication if the interface could be successfully detected and configured.

Initially the Wi-Fi mode will be in an idle state waiting for instructions which can executed by using a set of general routines:

```
extern int fnAccessPointWifi(unsigned char ucInterface);

extern int fnStationModeWifi(unsigned char ucInterface);

extern int fnConnectWifi(unsigned char ucInterface);

extern int fnScanWifi(unsigned char ucInterface, unsigned char ucChannel);
```

As can be seen there is always an interface parameter for one of potentially multiple Wi-Fi interfaces that are possible, or that are in operation. Parameters are not required in most cases due to the fact that the operation is based on a product configuration (*uParameterSystem*) where each WiFi interface has its own set of parameters that can be set and saved according to the exact requirements of the project or the user. For example, when the Wi-Fi is used in Access point mode each interface will have its own parameters for its SSID, security type and password which will normally be configured once and then used automatically each time the mode is entered.

7.1 Controlling the Wi-Fi on the Command Line Interface

The Wi-Fi menu is available on the command line interface (on UART, USB-CDC device or Telnet) and looks something like the following reference (*exact commands may change or be extended over time and operations not supported by certain modules may be removed*):

```
   Wi-Fi menu
===================
up             go to main menu
int            Set wi-fi interface reference
show_config    Show Wi-Fi
ap             Start AP
ad             Disconnect AP
scan           Scan WiFi [channel]
st             Station Mode
sd             Disconnect
dhcp           Start DHCP
at             AT command mode
raw_on         enable raw mode
raw_off        disable
dhcp_on        enable DHCP server
dhcp_off       disable
bridge_on      enable bridge
bridge_off     disable
show_ssid      SSID visible
hide_ssid      invisible
set_chan       WiFi [channel]
set_SSID       Set our SSID
set_PASS       Set our pass phrase
set_ssid       Set remote SSID
set_pass       Set remote pass phrase
debug_level    Set debug level [level]
save           Save configuration to FLASH
help           Display menu specific help
quit           Leave command mode
```

The „`int`" (interface) command is only availabel when muliple Wi-Fi's are used and allows selecting each individual Wi-Fi's set of parameters and controls.

There are generic commands to view and change each Wi-Fi module's configuration and command modes and operation.

Wi-Fi modules that support AT commands can be commanded by these AT commands manually (eg. for testing or experimenting purposes) by using the „`at`" command mode. Once in this mode all input is sent directly to the module in its RAW format. A dummy „`ATQ`" command is used to exit this mode.

This interface allows the user to work with all supported Wi-Fi modules in an identical manner – for example, the various settings, names and passwords are configured and saved in the exact same fashion for each and then the „`ap`" command sets the particular Wi-Fi module to its access point mode so that Wi-Fi clients can connect to it. Should different Wi-Fi modules be used in different projects the interface and control doesn't need to be leared again as it remains generic.

8. Typical Configurations

This section concentrates on actually using the Wi-Fi module in projects. It is recommended to review this chapter in the order of its sub-sections since the build up on each other in order – starting with the simplest configuration and moving on to more complex ones.

8.1 Processor with just a single Wi-Fi Interface acting as a Web Server to Mobile Devices

Just about any processor can be used for this configuration. It requires the Wi-Fi connection (eg. via SPI) and the µTasker TCP/IP stack, whereby it is assumed that the Wi-Fi module is operating in RAW Ethernet frame mode (also known as bypass mode). The Wi-Fi is set to its Access Point (AP) mode and a DHCP server enabled on the Wi-Fi interface in the µTasker TCP/IP stack.

After the Wi-Fi client has connected to the AP it will request IP settings and the DHCP server in the µTasker TCP/IP stack will allocate settings, including the client's IP address, suitable for its network. *An example would be that the µTasker TCP/IP stack's single network has an IP address of 192.168.1.3 and allocates the Wi-Fi client its dynamic IP address of 192.168.1.102.*

Once the network settings have been established the Wi-Fi client can communicate with the µTasker TCP/IP stack on the address 192.168.1.3, which means that it can ping it and browse its web server content.

This is a simple one IP interface and one IP network configuration and is more or less identical to a standard processor with Ethernet except for the fact that the IP interface is Wi-Fi and not Ethernet. For the web server, TCP/IP stack and other services the complete operation is identical at the layer above the physical interface layer.

This is the main, basic setup of such a configuration in the µTasker project, assuming an ATWINC15x0 is used as Wi-Fi interface:

```
#define WIFI_INTERFACE                    // enable wifi interface(s)
#define IP_WIFI_INTERFACE_COUNT   1     // set to the number of wifi modules attached

#define IP_NETWORK_COUNT          1     // number of networks

#define USE_IP           // enable IP and ARP, required for possible tcp/ipv4 services
#define ARP_TABLE_ENTRIES        8     // the maximum entries in ARP table
#define USE_ICMP                // enable ICMP (for ping and destination unreachable)
#define ICMP_PING                         // allow PING reply
#define ICMP_SEND_PING                    // support PING transmission
#define USE_UDP                           // enable UDP over IP
#define USE_DHCP_SERVER                   // enable DHCP server
#define MAX_DHCP_CLIENTS          4
#define USE_TCP                           // enable TCP over IP
#define USE_HTTP        // support embedded Web server - needs TCP (and a file system)
```

Various other options can be found in `config.h`, which can be used as full reference.

It is to be noted that the local DHCP server is started with the following code when the Wi-Fi interface configuration is set to enable it and this configuration is required if for this mode to operate correctly (otherwise the Wi-Fi client will not receive its IP settings and therefore will not be able to communicate):

```
if ((temp_pars->temp_parameters.wifi_pars[WIFI_ATWINC15_PARAMETER_REFERENCE].usWifi_config &
                                                                     WIFI_CONFIG_DHCP_SERVER) != 0) {
    fnStartDHCP(OWN_TASK,
               (DHCP_SERVER_OPERATION | defineNetwork(DEFAUT_NETWORK) | WIFI_ATWINC15_INTERFACE));
                                                          // activate DHCP server on the wifi interface only
}
```

The code shows the DHCP server being conditionally started on the `WIFI_CONFIG_DHCP_SERVER` flag, which is configured in the Wi-Fi menu. The DHCP server is started on the default network and listens only on the Wi-Fi interface. The interface detail is in fact superfluous in the case of this first simple configuration but serves to limit its activity to this single interface when multiple ones exist and so is important in following setups.

For details about how such services are defined to operate on particular networks and interfaces see the networking document https://www.utasker.com/docs/uTasker/uTaskerNetworking.pdf

The application can start the Wi-Fi module in AP mode by using the command

```
fnAccessPointDisconnectWifi(WIFI_ATWINC15_INTERFACE);
```

which will allow a Wi-Fi client to connect to it. Once the connection has been made the µTasker TCP/IP stack will handle traffic so that the processor's web server operates on the interface.

The µTasker command line interface allows all settings to be controlled and saved and the AP mode to be commanded. The network can then be monitors on the command line interface by viewing IP statistics on the network, the ARP table and DHCP client table.

Network Menu:

```
show_config
IPV4 address = 192.168.1.4
MAC address = f8-f0-05-f4-1c-8a
Subnet mask = 255.255.255.0
Default gateway = 192.168.1.1
DNS server = 192.168.1.1
```

(Wi-Fi MAC address)

Wi-Fi menu:

```
show_config

Wi-Fi 0 AP config:
In bypass (RAW) mode
DHCP server on
Bridging off
SSID visible
Security: WEP 40
Channel 1
SSID "uTaskerWIFI1"
Pass phrase: "1234567890"
```

Command to AP mode:

```
#ap
Starting AP...OK

#ATWINC state CONNECTED
RSSI -100dBm
DHCP assigned 192.168.1.102 to 44-c7-fc-ed-22-4d
```

Check IP activity on the network:

#ipstat

```
Ethernet Statistics

Total Rx frames = 136
Rx overruns = 0
Rx ARP = 3
Rx IGMP = 0
Rx ICMP = 0
Rx UDP = 115
Rx TCP = 14
Rx checksum errors = 0
Rx other protocols = 0
```

```
Foreign Rx ARP = 4
Foreign Rx ICMP = 0
Foreign Rx UDP = 0
Foreign Rx TCP = 0
Total Tx frames = 19
ARP sent = 3
IGMP sent = 0
ICMP sent = 0
UDP sent = 2
TCP sent = 14
Other events = 0
```

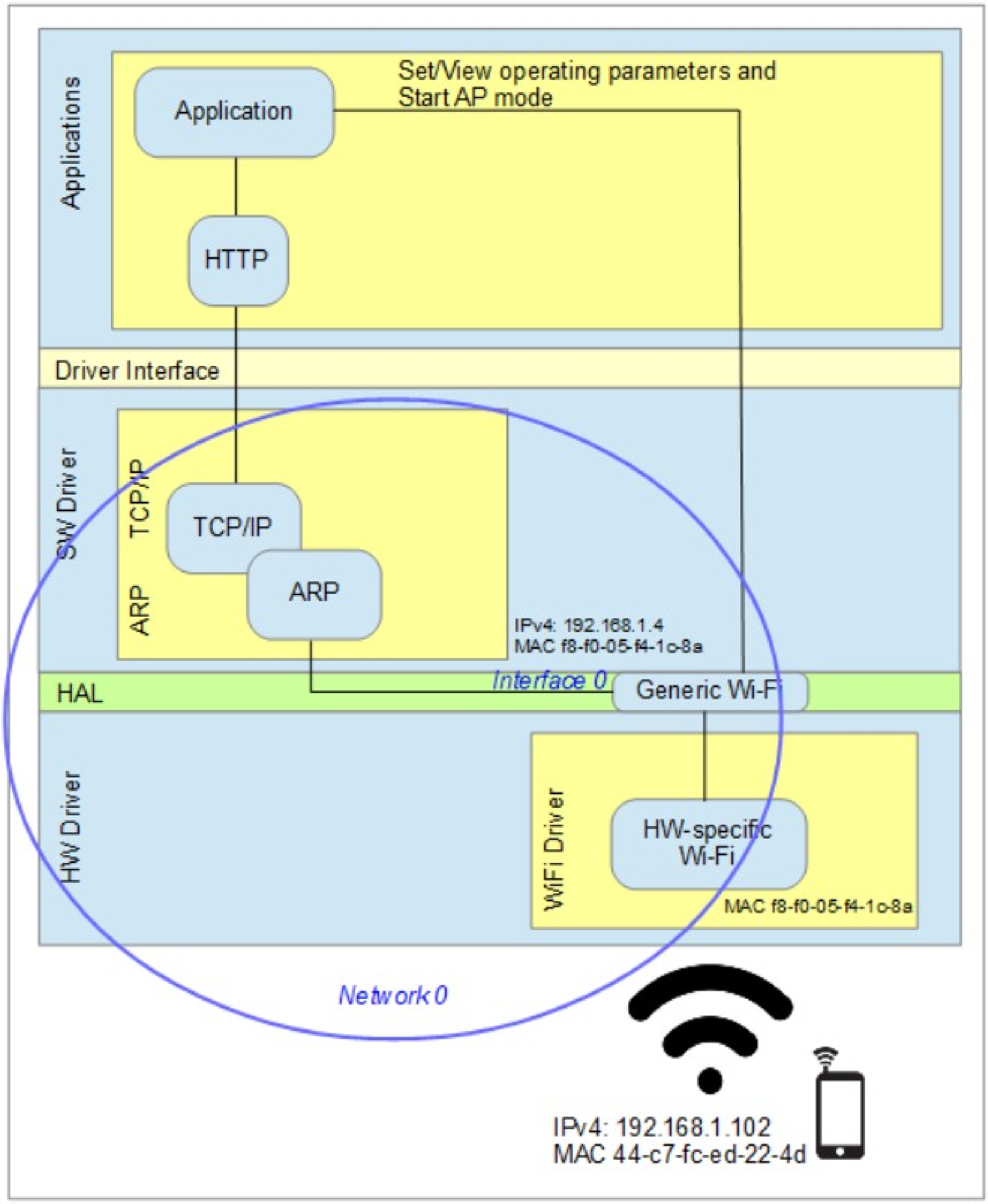
Check the TCP/IP stack's ARP table:

```
#arp -a

ARP Table contents:
IP address: 192.168.1.102 MAC address: 44-c7-fc-ed-22-4d
End
```

Check the network's DHCP client table:

```
#dhcpc

DHCP client Table:
IP address: 192.168.1.102 MAC address: 44-c7-fc-ed-22-4d
End
```

The following diagram shows this configuration with the MAC and IP addresses for this example use case. Note that the Wi-Fi module always supplies the MAC address of the interface and this is assigned to the network that it is on.



8.2 Processor with an Ethernet and a Wi-Fi Interface acting as a Web Server to Mobile Devices and LAN based devices but on separate Networks

Assuming that the processor has an Ethernet connection this can be enabled in parallel to the Wi-Fi interface.

```
#define ETH_INTERFACE                      // enable Ethernet interface driver
#define WIFI_INTERFACE                     // enable wi-fi interface(s)
#define IP_WIFI_INTERFACE_COUNT   1        // set to the number of wi-fi modules attached

#define IP_NETWORK_COUNT          2        // number of networks

#define SEPARATE_NETWORKS              // each interface is on its own separate network

#define ETHERNET0_NETWORK_REFERENCE      DEFAULT_NETWORK
#define WIFI_ATWINC15_NETWORK_REFERENCE  SECOND_NETWORK

#define USE_IP           // enable IP and ARP, required for possible tcp/ipv4 services
#define ARP_TABLE_ENTRIES         8        // the maximum entries in ARP table
#define USE_ICMP                   // enable ICMP (for ping and destination unreachable)
#define ICMP_PING                          // allow PING reply
#define ICMP_SEND_PING                     // support PING transmission
#define USE_UDP                            // enable UDP over IP
#define USE_DHCP_SERVER                    // enable DHCP server
#define MAX_DHCP_CLIENTS          4
#define USE_TCP                            // enable TCP over IP
#define USE_HTTP        // support embedded Web server - needs TCP (and a file system)
```

Both Ethernet and Wi-Fi interfaces have been enabled in the project and, since they should each use independent networks `IP_NETWORK_COUNT` has been set to 2 and the define `SEPARTE_NETWORKS` set, which automatically ensures that traffic form the two interfaces are restricted to their respective network without any intervention by the appliation.

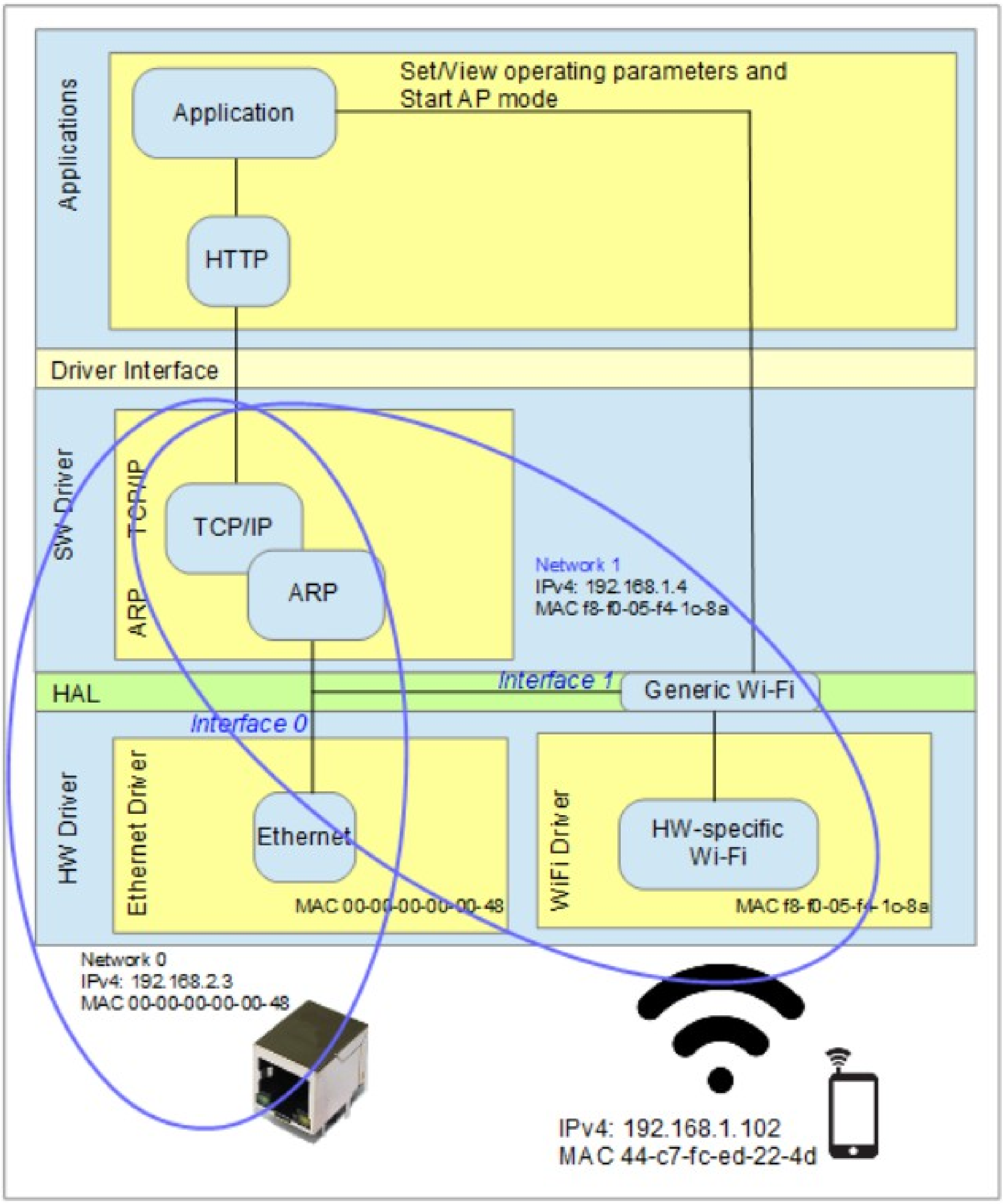
The Ethernet interface is assigned to the first network settings and the Wi-Fi is assigned to the second set, whereby these two sets will normally be non-conflicting network configurations.

In this case we want the DHCP server to be active only on the Wi-Fi interface and only on the Wi-Fi network so the DHCP server is started with

```
fnStartDHCP(OWN_TASK, (DHCP_SERVER_OPERATION |
         defineNetwork(WIFI_ATWINC15_NETWORK_REFERENCE) |
         WIFI_ATWINC15_INTERFACE));
                          // activate DHCP server on the wi-fi network/interface only
```

In this configuration the operation of the Wi-Fi interface in AP mode will be equivalent to that in the non-Ethernet case. However the Ethernet interface will also have access to the µTasker TCP/IP stack and web server but on its own network. The two networks are fully independent. This new configuration is illustrated in the following diagram showing that two independent networks and two IP interfaces are in operation.

No application level changes are required between the Ethernet and non-Ethernet cases beyond ensuring that the Wi-Fi interface's DHCP server is enabled accordingly.


Applications
Application
Set/View operating parameters and
Start AP mode
HTTP
Driver Interface
SW Driver
TCP/IP
ARP
TCP/IP
ARP
Network 1
IPv4: 192.168.1.4
MAC f8-f0-05-f4-1c-8a
HAL
Interface 1
Generic Wi-Fi
Interface 0
HW Driver
Ethernet Driver
Ethernet
MAC 00-00-00-00-00-48
WiFi Driver
HW-specific
Wi-Fi
MAC f8-f0-05-f4-1c-8a
Network 0
IPv4: 192.168.2.3
MAC 00-00-00-00-00-48
IPv4: 192.168.1.102
MAC 44-c7-fc-ed-22-4d

8.3 Processor with an Ethernet and a Wi-Fi Interface allowing Mobile Devices to Communicate with LAN Devices and use the LAN router of Internet Access

….

9. RAW Mode and Socket Mode

Wi-Fi modules operate either in RAW mode (also known as bypass mode) or in socket mode.

RAW mode means that they pass raw Ethernet frames between the controlling processor and the Wi-Fi network and so the controlling processor has to perform the complete TCP/IP stack operation. *The controlling processor has more overhead but also full flexibility*.

Socket mode means that the TCP/IP stack is performed by the the Wi-Fi module. This offloads work from the controlling processor but requires the controlling processor to configure specific sockets (usually UDP and/or TCP sockets) at the Wi-Fi module and anything not configured (eg. other TCP ports or various IP protocols) will be consumed or rejected by the Wi-Fi module without the controlling processor having the opportunity to handle them. Often the TCP sockets in the Wi-Fi module can perform SSL/TLS security which further offloads overhead from the controlling processor when secure TCP is required.

Since sockets consume resources in the Wi-Fi module it is possible that they are limited in the number of sockets or ports that can be used at the same time.

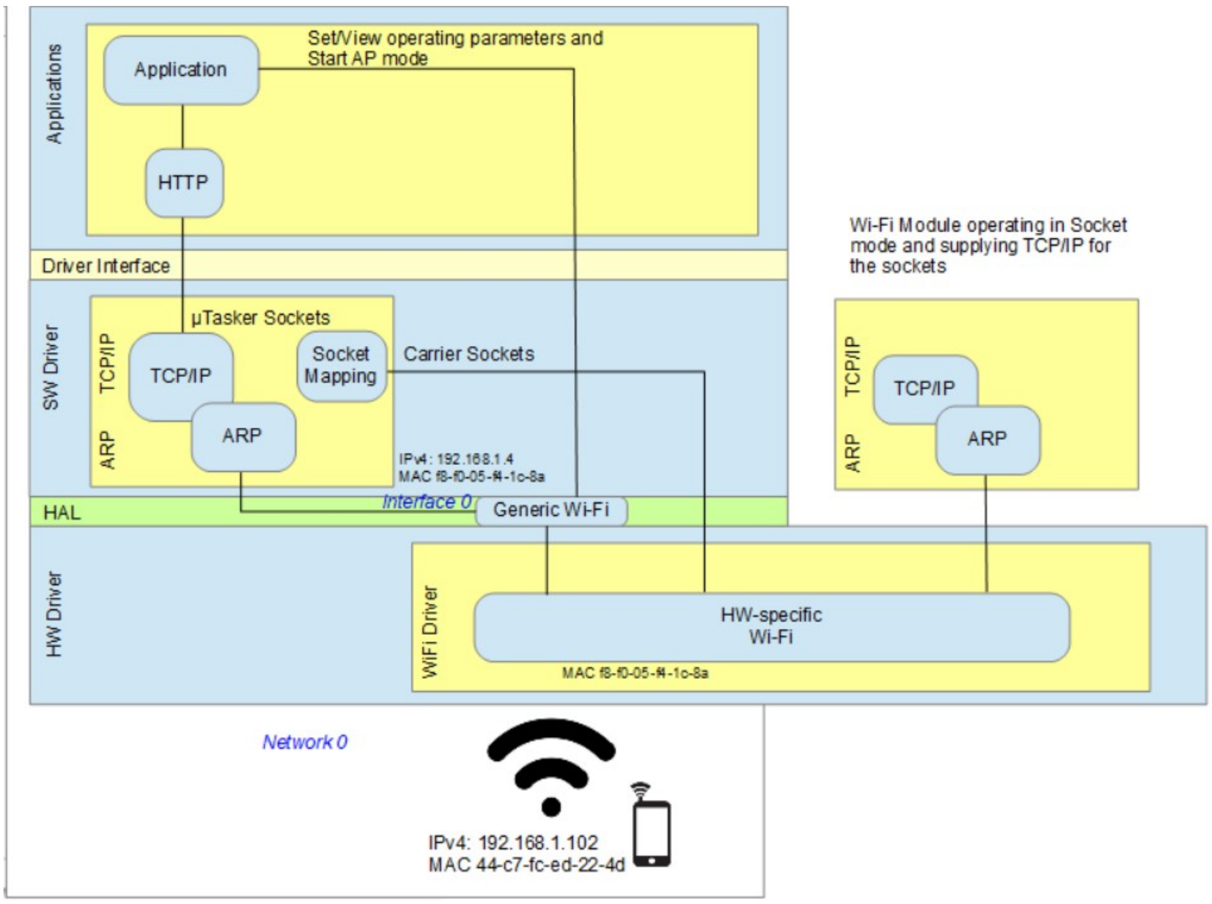
In socket mode the controlling processor receives socket events (such as when a client connects to a TCP server) and payload data. The controller doesn't need to be concerned with the transport layers or repetitions in case of data loss.

Practically the most flexible Wi-Fi modules offer either RAW or sockets mode and ideally allow secure sockets too. Optimally a Wi-Fi module would provide RAW mode operation but allow secure or unsecure sockets to be specified on specific ports to allow the controlling processor maximum flexibility but still the capability of offloading specific (secure) sockets when this makes sense.

In both socket and RAW modes the controlling processor is responsible for the application layer. A good example of the application layer is a web server using the HTTP protocol. The TCP protocol is above the TCP (and SSL/TLS) layer and so generally will have to be handled by the controlling processor, although HTTP itself may be considered to be a part of the overall TCP/IP stack in typical libraries.

The µTasker solution solves both RAW and Socket modes by providing a mechanism for applications (and higher level protocols in the TCP/IP stack) to be able to be developed and used compatibility, whether the specific Wi-Fi module requires socket mode or not. The advantage of this is that applications and higher level protocols need no modification or design changes in order to operate with Wi-Fi modules of any type as long as socket based ones support the quantity of sockets and the variety of ports that the overall application needs.

9.1 µTasker and Carrying Sockets



This diagram illustrates the situation when a Wi-Fi module is operating in socket mode and handling one or more TCP sockets for use by HTTP and an application (eg. a web server).

The difference to the RAW mode case, which is also equivalent to when Ethernet is used, is that the TCP/IP traffic doesn't arrive at Interface 0 and therfore much of the internal TCP/IP stack becomes redundent (at least with regards to the Wi-Fi traffic related to the TCP socket in question). Instead only events (such as when a client establishes a TCP connection) are reported and the payload data is exchanged.

In both cases the application layer works with µTasker sockets and the appication can be oblivious to the fact of how the underlying transport layers actually operate. In the case of a Wi-Fi module using socket mode these sockets, known as „carrying sockets“ since they carry the payload in an alternative fashion but with the same interface from the point of view of the application, are mapped to µTasker sockets (see the additional „Socket Mapping“ module).

Before illustrating the operation it is worth pointing out that the µTasker sockets may still be in used by other interfaces, such as Ethernet or RNDIS, and the application layer shared between both or all.

In the case of a Wi-Fi client establishing a TCP connection to the Wi-Fi socket on the appropriate TCP port the Wi-Fi module reports this as an event (eg. a connection event, possibly with details about the client) and the socket mapper searches for a free µTasker TCP listening socket on that port to *map* the „carrier socket“ to. If there is none it would command the Wi-Fi module to immediately close the TCP connection or else set a link between the µTasker socket and the „Carrier Socket“ for further operations related to this unique TCP connection.

When the Wi-Fi client sends payload data this payload data can be passed to the mapped µTasker socket so that the higher layers receives it in an idenical manner as in the case when it were arriving natively via the µTaster socket.

Application payload sent on µTasker sockets is diverted to the mapped „Carrier Socket“ and thus passed to the Wi-Fi's sockets for transmission.

For each application layer command - *for example the application wants to close the connection* - the application performs the standard command to the µTasker socket, which knows to which „Carrier Socket“ it is being mapped to and thus translates the command into the appropriate Wi-Fi socket command to perfom the requested action.

Likewise, Wi-Fi socket events arriving on „Carrier Sockets“ are translated to equivelmt µTasker socket events so that the application interface remains compatible in all circumstances.

This strategy allows the same application to work with Wi-Fi modules in RAW and Socket modes wthout being aware of the lower level details. It also allows application to work with both Wi.Fi modules in socket mode and other interfaces (such as Etheret or RNDIS) in RAW mode *at the same time*.

9.2 Web Server using ESP32-AT mode

The ESP32-AT mode allows control and communication via a UART, using AT commands. This may not be the most practical method due to the fact that AT commands are quite clusy for data transmission but it is workable and this section shows how it is performed using the µTasker generic Wi-Fi interface layers, carrying sockets and the specific ESP32-AT command set.

10. Conclusion

Document in development – additional Wi-Fi modules, modes and technical details in progress.

Modifications:

V0.02 7.2.2021 Preliminary version in progress
V0.03 18.4.2021 Preliminary version in progress – added Telit cellular modem and CLI Wi-Fi command details
V0.04 23.4.2021 Preliminary version in progress – add details of Wi-Fi socket mode and "Carrier Sockets"
V0.07 27.8.2021 Adds ATWINC34xx and integrated firmware loading bridge

Appendix A

a) ….