Digital Radio Projects

IP400 Phase 2 Pi HAT Radio Node

Document Number: TBD
Revision: 1.1
Status: Preliminary

Written By: Martin C. Alcock, M. Sc, MIEEE, VE6VH

Table of Contents

List of Tables

List of Figures

References

[1] gnu.org, "General Public Licence," [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 25th February 2018].

[2] ST Microelectronics, "STM32WL33CC: Sub-GHz Wireless Microcontrollers.," ST Microelectronics, [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32wl33cc.html. [Accessed 28 1 2025].

[3] ST Microelectronics, "Integrated Development Environment for STM32," [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html. [Accessed 26 1 2025].

Revision Status

Revision	Date	Description
1.0	April 11th, 2025	First Draft
1.1	May 17th, 2025	Corrected minicom command

Table 1 Revision status

Reference Documents

Author	Issue Date	Description
M. Alcock	Jan 2025	IP400 Protocol Specification
M. Alcock	Mar 2025	IP400 module SPI port specification

Table 2 Reference Documents

Intellectual Property Notice

Disclaimer

This document is a preliminary release for a product still in development and may be subject to change in future revisions. The software described herein may be subject to unpredictable behaviour without notice. You are advised to keep a can of RAID™ Ant, Roach and Program Bug killer handy. Spray liberally on the affected area when needed.

If any page in this document is blank, it is completely unintentional.

Introduction

The IP400 project was launched to experiment with digital mesh networking on the 400 MHz band, using commercial devices designed to run in this band.

This document describes the implementation for a module which contains an STM32WL33 microcontroller [2]. This will be superseded with other platforms for the Raspberry Pi and new devices that will include signal processing for higher orders of modulation, and RF components for other bands.

Operation of the node is segmented into a physical layer that runs on the microcontroller, and other layers that run on different host processors. The lower layer code contains the bare minimum to send and receive frames, repeating frames and building a mesh table, and contains a simple application.

The application code includes a simple setup menu, and the ability to change and store station and radio parameters, as well as a simple chat application to demonstrate the capabilities. It periodically sends a 'beacon' frame to build the mesh tables, which contains information about the station, including latitude, longitude and grid square. Provision has also been made to connect to a GPS receiver to update the position information dynamically.

Software for the Pi enables an ethernet connection to be made between the node and other devices, using the UDP protocol. The serial peripheral interface (SPI) is used for this purpose. The code is distributed in source form, so that it can be included in other applications. A dissector has also been developed for the popular ethernet analyzer 'wireshark', which can decode and display IP400 ethernet packets for debugging purposes. For more information, see the SPI specification document.

Roadmap

Figure 1 illustrates the roadmap for the IP400 node development.

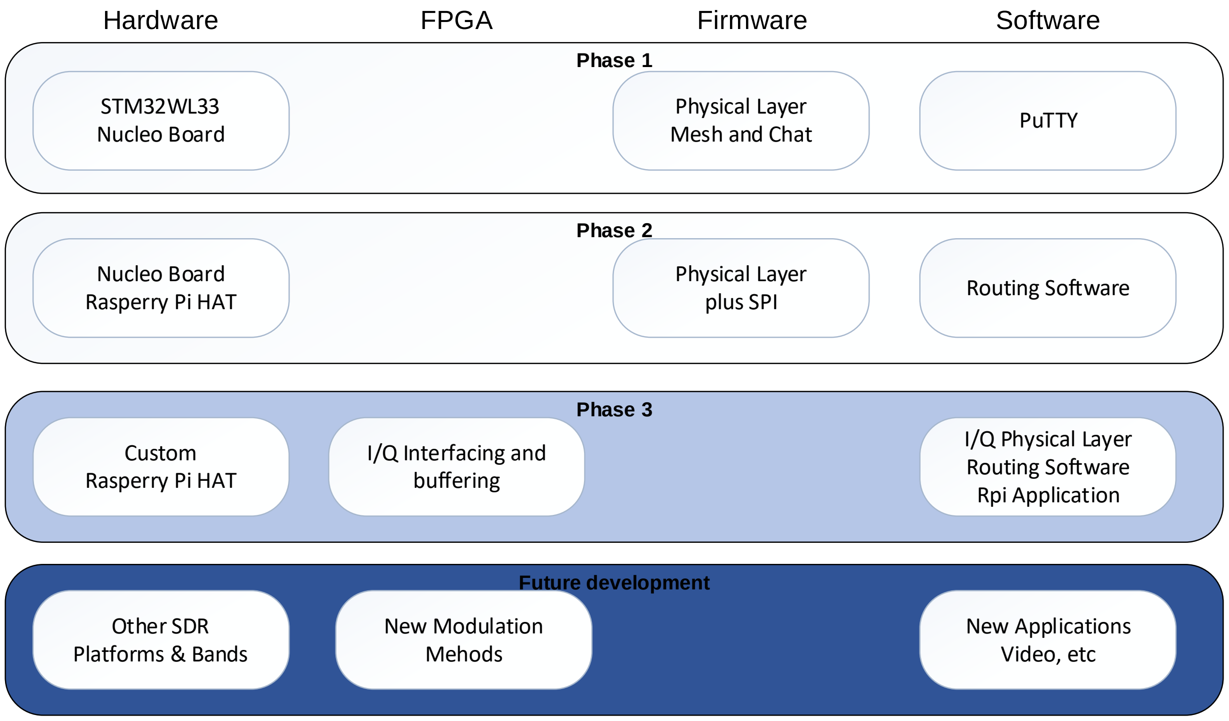

Hardware
FPGA
Firmware
Software
Phase 1
STM32WL33 Nucleo Board
Physical Layer Mesh and Chat
PuTTY
Phase 2
Nucleo Board Rasperry Pi HAT
Physical Layer plus SPI
Routing Software
Phase 3
Custom Rasperry Pi HAT
I/Q Interfacing and buffering
I/Q Physical Layer Routing Software Rpi Application
Future development
Other SDR Platforms & Bands
New Modulation Mehods
New Applications Video, etc

Figure 1 IP400 Node Development Roadmap

There are four phases of development for the IP400 node:

- Phase 1 is the initial evaluation phase using the Nucleo board and a simple chat application and is purely a firmware exercise. Using the PuTTY application, a connection can be made to the USART on the board. The software will implement a basic frame transmitter and receiver and be able to build a mesh table and repeat frames. The firmware can only be loaded using the ST integrated development environment (IDE) [3] and an integrated STLINK Debugger.
- Phase 2 introduces the HAT for the raspberry Pi platform. The firmware has been upgraded to include a high speed SPI, plus a downloader has also been developed to alleviate the absolute requirement for the IDE, but support for an external STLINK debugger has been provided, so the IDE can also be utilized as an option. New applications on the Pi can be developed for routing, audio, file transfers, AX.25 and encapsulated IP.
- Phase 3 will introduce new custom hardware and FPGA-based signal processing for higher modulation methods and add new applications, to be determined.
- The final phase will migrate to other bands and platforms.

Pi Zero HAT Hardware

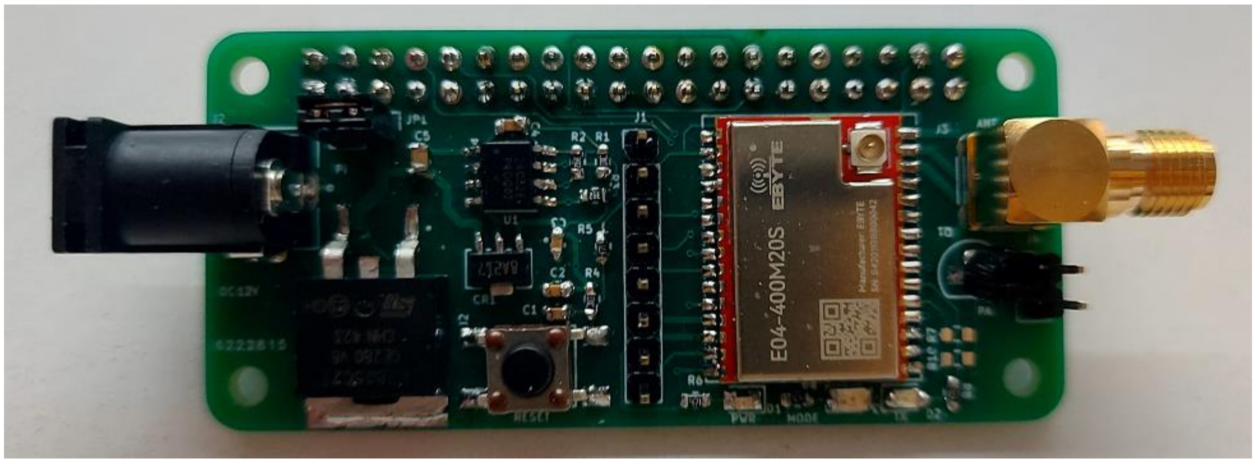


Figure 2 IP400 E04 Pi Zero HAT

The HAT for IP400 consists of a single PCB that holds a module, which contains the RF components. It has the following features:

- EEPROM for Pi HAT standard identification
- STMWL33 processor with 400 MHz coupler
- PA control for external amplifier
- Analog regulator from 7-14V to power the Pi and module
- Reset Switch
- 3 LED's: one power, one bi-colour, and one transmit.
- ST Link connector with VCOM port (LPUART)
- Support for external GPS receiver.
- TTYama0 connection to USART1
- SPI connection for high speed data
- Reset/Boot from the Pi.

Getting Started

Starting with the preconfigured O/S image

Install the HAT on the Pi and connect an external power supply and antenna. The input voltage can be between 7 and 14V. Using a higher voltage could cause the linear regulator to overheat, and it will shut down. It is advisable to install a heatsink on the L7085 regulator if it is getting too warm.

Download the Image

Visit the IP400 project site, www.ip400.adrcs.org, then go to the Hardware page. Scroll to the bottom to find the ‘getting started’ link. From that page, you can download the raspberry Pi O/S. Copy it to an SD card and boot up the pi.

Logging in

The login username is ‘**ip400’** and the password is ‘**ip4004pi’**.

Accessing the Menu

The menu can be launched directly from the pi either by connecting a screen and keyboard, or using a remote SSH session, if you have it connected to a local area network. The recommended method is to use PuTTY to launch a remote session.

From the command prompt, launch minicom with the command:

```
minicom -b 115200 -D /dev/serial0
```

Connecting other devices

Connecting a GPS Receiver

A GPS receiver can be connected to the module using the following pins on the debugger connector.

Pin	Function	GPS Receiver
2	NRST	Reset input
3	VCP_TX	Transmit data from the GPS
4	VCP_RX	Receive data to the GPS
6	GND	Ground
7	VCC	3.3V supply

Table 3 Connecting a GPS Reciever

Connecting a power amplifier

The Pi module supports a connection to a power amplifier that is buffered with an FET transistor. The signal can be found on J4, just below the antenna connector. The top pin is the PTT key, the bottom is a ground. The FET can sink up to 60mA.

Programming the HAT

These steps are only necessary when not using the precompiled image, or for a new board.

Programming the ID EEPROM

If you have built the HAT board yourself, or have replaced the CAT24C32 EEPROM, it should be programmed before operating the node to correctly identify the hardware, if not, you can skip this step.

From the home directory:

```
cd eeprom
make eeprom
make flash
```

Then reboot the Pi. To verify that it was read correctly:

```
cd /proc/device-tree/hat
cat product
```

You should see “IP400 E04 HAT Rev X”, where x is the current rev number.

Flashing the IP400 Module

There are two methods to flash the software on the Pi HAT:

1. Using the IDE and debugger.
2. Using the Flash utility.

In all cases, obtain the latest code from the repository, https://github.com/adrcs/ip400.

To use the IDE, unzip the PI Zero project from the Platforms directory, and import it into the IDE. Copy the latest code from the IP400 directory to the IDE workspace and build the project. Connect an STLInk-V3SET debugger to the board using the pinout in Table 1Table 4, then compile and download the code using the debugger.

Pin	Function	Debugger
1	SPI SYNC	Toggles when SPI addressed
2	NRST	Reset input
3	VCP_RX	Receive data to the LPUART
4	VCP_TX	Transmit data from the LPUART
5	SWCLK	Debugger serial clock
6	GND	Ground
7	VCC	3.3V supply
8	SWDIO	Debugger serial data

Table 4 RPi connections to STLink

To flash the module from the Pi, a precompiled binary and a shell script are in the ‘code’ directory. Starting in the home directory:

```
cd code
./ip400flash.sh PIZero_Ichiban_xxx.bin
```

Where xxx refers to model number of the module.

The flash utility will run and upload the code to the Pi. When complete, the bi-colour LED should be green, indicating that the node is in the receive mode.

If you want to recompile the code in STM32Cube IDE and do not have a STLink-V3SET debugger, you can copy the output of the build to the Pi and flash it with the utility.

Installing the terminal program

To access the menu, you will need a terminal program. The recommended utility is called ‘minicom’, which requires the following steps to install it:

1. Run the configuration program ‘raspi-config’, and choose Interface Options, then Serial Port (I6). Turn off the login shell by answering ‘no’ to the first question. Enable the serial port by answering ‘Yes’ to the second question. Then re-boot the Pi.
2. Download and install minicom if you do not have it, with ‘sudo apt-get install minicom’.

Compiling using Cube IDE

Globally used variables for conditional compilation for the are all contained in the include file config.h, Table 5 lists them and their effect on compilation.

Conditional	Value	Effect
_BOARD_TYPE	PI_BOARD	Compiles code for the raspberry pi board
	NUCLEO_BOARD	Compiles code for the Nucleo board
__ENABLE_GPS	0	Omits GPS code
	1	Includes GPS code for LPUART

Table 5 Conditional Compilation

Other modules may contain conditionals for debug purposes, which are local to that module only.

Import the correct project for your hardware platform and then add the most recent code to it. The steps are shown below.

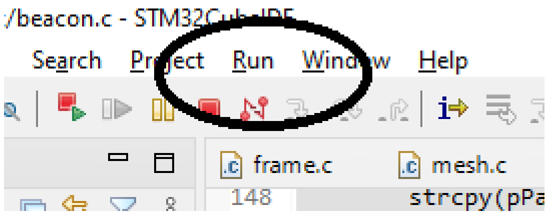
1. Step 1: Go to the File->Import menu, then choose General->File System, then click next.
2. Step2: Using the browse button, navigate to the IP400 directory that you downloaded from GitHub. Click on the ‘Src’ directory.
3. Step 3: Click the box in the left pane, and make sure the “Into Folder” is pointing to your project folder, then IP400/Src.
4. Step 4: Click Finish.
5. Step5: Repeat steps 2-4 for the ‘Inc’ directory.

Then use the ‘build project’ to build it.

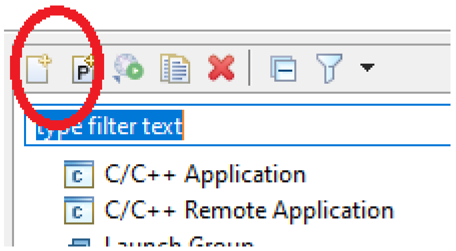
Using the debugger

There are seven steps involved in running from the IDE:

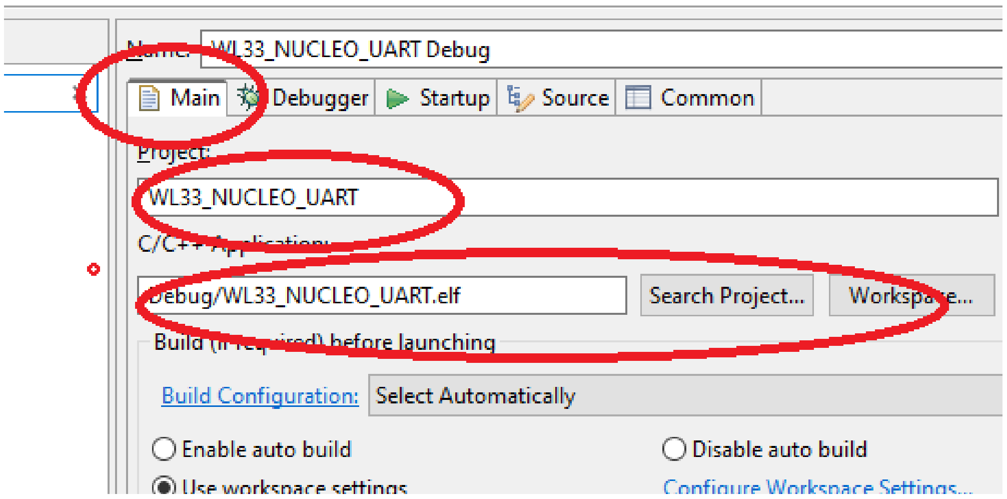
Step 1: Choose the RUN menu item, the Debug Configurations:



Step 2: A dialog box will launch. Click the 'new configuration' icon:



Step 3: on the Main tab, select your project and the .elf object file:



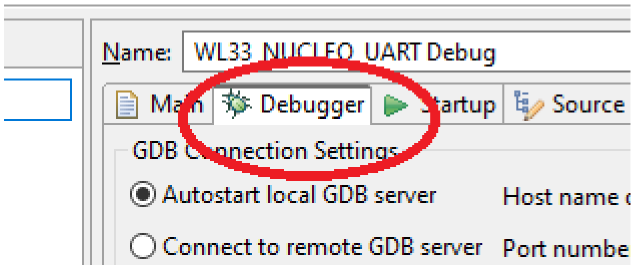
Name: WL33_NUCLEO_UART Debug

Project:
WL33_NUCLEO_UART

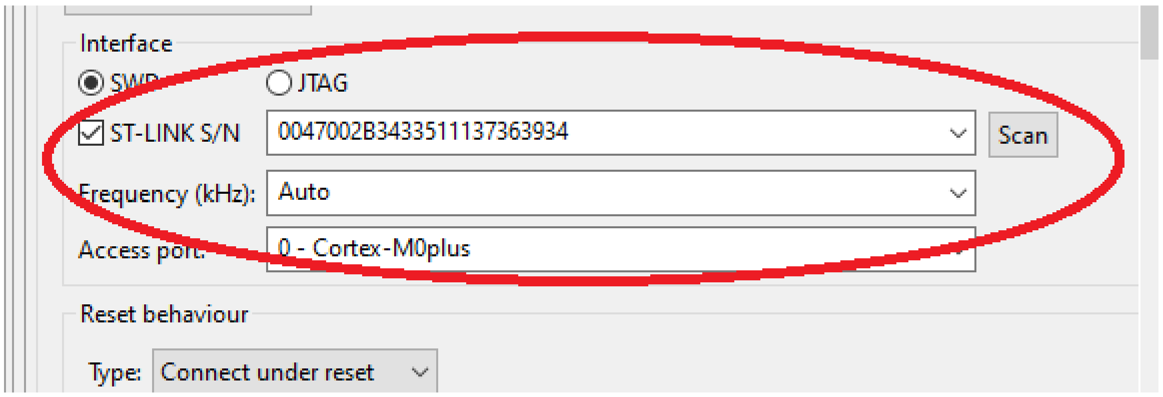
C/C++ Application:
Debug/WL33_NUCLEO_UART.elf

Build (if required) before launching
Build Configuration: Select Automatically
( ) Enable auto build
( ) Disable auto build
(●) Use workspace settings

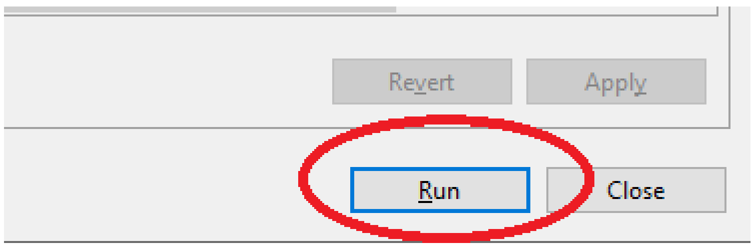
Step 4: Ensure your Nucleo is connected to your PC, the click the 'Debugger' tab.


Name: WL33_NUCLEO_UART Debug

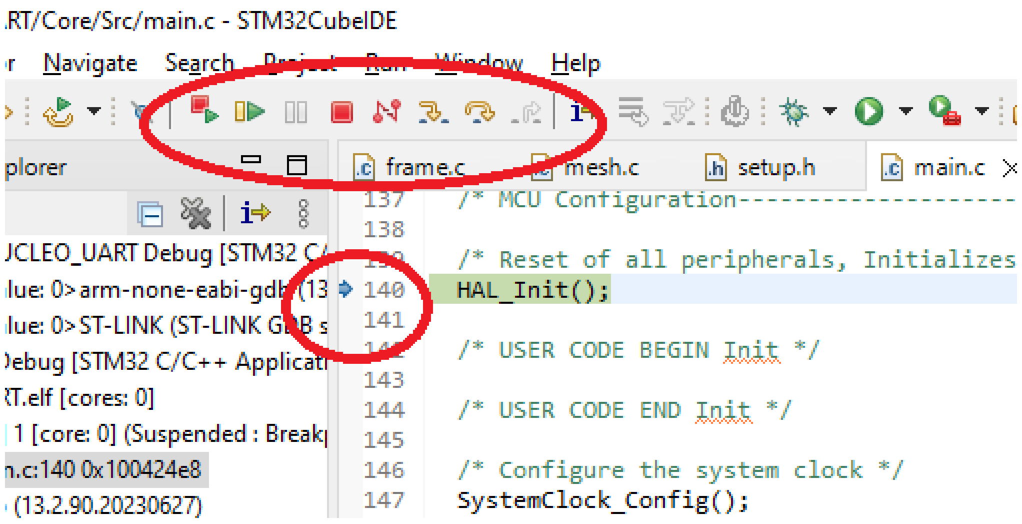
Step 5: Select the debugger tab, click the ST-LINK check box, and scan. Then the serial number of your Debugger.


Interface: SWD [✓] / JTAG
ST-LINK S/N [✓]: 0047002B3433511137363934
Frequency (kHz): Auto
Access port: 0 - Cortex-M0plus
Reset behaviour
Type: Connect under reset

Step 6: Click the 'Run' button.



Step 7: An editor will open and there will be an arrow next to the first line of executable code.


.RT/Core/Src/main.c - STM32CubeIDE
r Navigate Search Project Run Window Help
plorer
frame.c
mesh.c
setup.h
main.c

JCLEO_UART Debug [STM32 C/
lue: 0>arm-none-eabi-gdb (13
lue: 0>ST-LINK (ST-LINK GDB s
Debug [STM32 C/C++ Applicati
RT.elf [cores: 0]
1 [core: 0] (Suspended : Break
n.c:140 0x100424e8
(13.2.90.20230627)

```
137  /* MCU Configuration--------------------
138
139  /* Reset of all peripherals, Initializes
140  HAL_Init();
141
142  /* USER CODE BEGIN Init */
143
144  /* USER CODE END Init */
145
146  /* Configure the system clock */
147  SystemClock_Config();
```

Click the green arrow to run it. The Red square will terminate it, and the combination. Will reload the code. Under the run menu you will find a restart that issues a reset to the device.