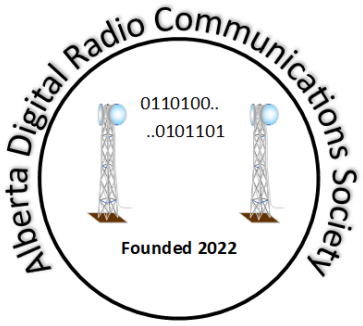

Digital Radio Projects

IP400 Radio Node SPI Protocol Specification

Document Number: IP400-SPI
Revision: 1.0
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].

Revision Status

Revision	Date	Description
1.0	March 24th, 2025	Initial release

Table 1 Revision status

Reference Documents

Author	Issue Date	Description
M. Alcock	Jan 2025	IP400 Physical Layer Specification
M. Alcock	Mar 2025	IP400 Experimenter Nucleo CC2 Radio Node

Table 2 Reference Documents

Intellectual Property Notice

The hardware components and all intellectual property described herein is the exclusive property of the Alberta Digital Radio Communications Society and others (“the owners”), all rights are reserved.

The owners grant licence to any Amateur for personal or club use, on an as is and where is basis under the condition that its use is for non-commercial activities only, all other usages are strictly prohibited. Terms and conditions are governed by the GNU public licence [1].

No warranty, either express or implied or transfer of rights is granted in this licence and the owner is not liable for any outcome whatsoever arising from such usage.

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 rial peripheral interface (SPI) presents the most expedient method to exchange data between a host and the node processor, as it is full duplex, which enables communication is both directions at the same time, and, unlike a traditional serial port, does not increase the word size by adding addition stop and start bits.

On the STM32WL33 processor [2], the SPI can support serial speeds of up to 16 MHz and can make use of DMA (direct memory access) in both the transmit and receive directions, making it ideal for higher speed applications.

This document discusses the SPI protocol used between an IP400 radio node and a host raspberry Pi processor. It is intended for developers who want to develop their own applications, either using the SPI directly or by the interface module, which uses a local IP connection.

A support application has been developed for the Raspberry Pi, which can be used as a gateway between the user datagram protocol (UDP) and a node, or as base for developing further applications.

SPI Data Transfers

SPI data transfers are between a master and slave, with the host processor as the master, and the IP400 node as the slave. Transfers are full duplex, occurring in both directions at the same time. All transfers consist of 8 bits, with the MSB being transmitted first.

The SPI consists of three signals:

Signal	Usage	Purpose
SPI_CLK	Clock	Indicates validity of a bit or sampling time
SPI_MOSI	Master Out, Slave In	Serial data from the master to the slave
SPI_MISO	Master In, Slave Out	Serial data from the slave to the master

Figure 1 illustrates the relationship between the clock and serial data transfers. In the forward direction, the master shifts data out on the negative edge, the slave samples on the positive edge. In the reverse, the slave shifts on the positive edge, and the master samples on the negative edge.

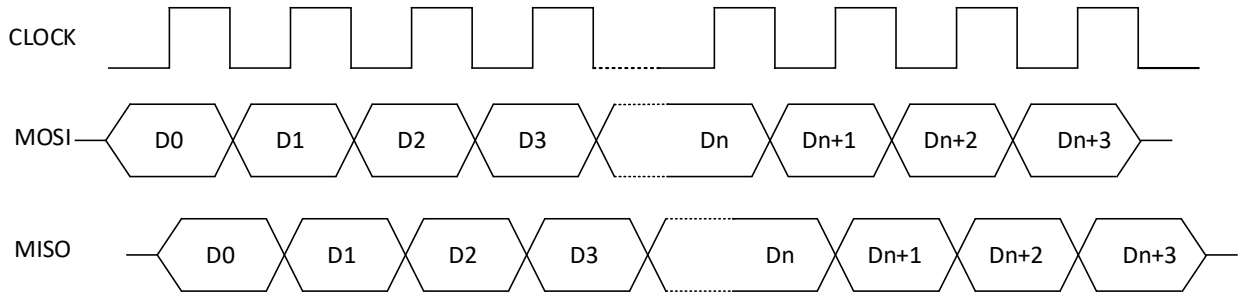

Figure 1 SPI Data transfers

The master controls the transfer by supplying enough clock cycles for the entire transfer. The maximum transfer rate is a function of the number of bytes transferred in each cycle, the speed of the clock, and the interval between transfers.

Clock Rate (KHz)	Bytes/Sec	Transfer (ms)	Transfer Bytes	Transfers/Sec
100	12500	3.2	400	31.25
500	62500	10	400	100
1000	125000	5	400	200
2000	250000	5	400	400

Table 3 SPI Effective transfer Rates

Table 3 illustrates effective data rates that can be achieved.

Protocol

The data exchange between the host and node consists of three fields:

Field	Size (bytes)	Contents
Header	24	Packet header information. See
Hop Table	0-60	Callsigns of stations repeating the frame
Payload	1-1025	Payload data

Table 4 SPI data exchange

Header Field

The header field contains the data as described in Table 5.

Field	Size (bytes)	Contents
Eye	4	Constant 'IP4C'
Status	1	Frame status. See Table 6.
Offset Hi	1	High byte of payload offset
Offset Lo	1	Low byte of payload offset
Length Hi	1	High byte of payload length
Length Lo	1	Low byte of payload length
From Call	4	Compressed 'from' call sign. See hop table description.
From Port	2	From port number
To Call	4	Compressed 'to' call sign. See hop table description.
To Port	2	To port number
Coding	1	Packet coding (type). See Table 7.
Hop Count	1	Number of times the packet has been repeated (also hop table size)
Flags	1	Flag field. See Table 8.

Table 5 Header field

The frame status can be one of the following values:

Value	Meaning
1	Complete frame, entire payload contained in payload field
2	Frame fragment. Offset and length can be used to reassemble final frame.
3	Reassembled frame. Frame has been reassembled from fragments

Table 6 Frame status field.

The packet type and coding method are shown below. The packet coding is contained in the lower 4 bits, the upper 4 bits are unused.

Coding (3-0)	Packet content
0000	UTF-8 Text Packet
0001	Compressed Audio packet. See Table 9.
0010	H.264 Compressed Video packet. See Table 9.
0011	Data Packet
0100	Beacon Packet
0101	IP encapsulated packet
0110	AX.25 packet
0111	Encoded DTMF Frame
1000	DMR Frame
1001	D-Star Frame
1010	TIA Project 25
1011	NXDN
1100	M17
1101	TBD
1110	
1111	Local command frame

Table 7 Packet coding

The flag field is described in Table 8.

Bits	Content
7-6	Compression method
5	Hop table present. Set to 1 when a hop table is contained in the frame
4	To call extended. The ‘to’ callsign has been extended into the payload field.
3	From call extended. Follows ‘to’ callsign if extended.
2	Command packet, consumed at the node.
1	Connectionless. Packet does not need a connection.
0	Repeatable. Packet can be repeated.

Table 8 Flags field

The compression method is described in Table 9.

Value	Audio Packet		H.264 Video Packet	
	Method	Data Rate	Image Size	Frames/Sec
00	µLaw	64 Kb/Sec	240x180	24
01	Raw PCM-16	128 Kb/s	320x240	24
10	M17 Codec 2	3.6Kb/s	480x360	12
11	AMBE Codec	9.6Kb/s	640x480	6

Table 9 Packet compression method

Hop Table

The hop table is a field from zero to 60 bytes, containing up to 15 entries of compressed callsigns of 4 bytes each. Each entry contains the callsign of the node that repeated the frame, with the latest at the end.

When a repeatable packet (flag byte bit 0) is received, the node adds its callsign to the end of the table and increases the hop count, if it has not reached the maximum, and the packet is repeated. When a packet is originated, it contains the routing information for the packet to reach its destination. If a node finds its callsign at the end of the hop table, it repeats the packet after removing itself from the table.

The callsign fields are compressed using an excess-40 scheme, which can reduce callsigns of up to 6 characters into four bytes, the remaining two are reserved for ports. In cases where an extension is needed, a further characters can be accommodated with additional compressed fields before the payload portion of the packet. The end is signified by an FF_{16} in the first (and only) byte of the last field.

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**
Callsign in Excess 40 format				Port Number	

Table 10 Source and Destination Coding

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**
Callsign extension characters 6-12 in Excess 40 format			
Callsign extension characters 13-18 in Excess 40 format			
FF_{16}	-		

Table 11 Callsign extension

The callsign field shall conform to the code set as illustrated in Table 12.

Character	**Coding**	**Character**	**Coding**	**Character**	**Coding**	**Character**	**Coding**
0	0	Space	10	J	20	T	30
1	1	A	11	K	21	U	31
2	2	B	12	L	22	V	32
3	3	C	13	M	23	W	33
4	4	D	14	N	24	X	34
5	5	E	15	O	25	Y	35
6	6	F	16	P	26	Z	36
7	7	G	17	Q	27	_	37
8	8	H	18	R	28	-	38
9	9	I	19	S	29	@	39

Table 12 Callsign character coding

The field is encoded using the following formula. Unused characters shall be replaced with a space (00).

$$F(3,0) = \sum_{n=1}^{n=6} C(n) + C(n-1) \times 40$$

Where F is the field value in bytes, C is the character and n the index into the callsign field. Each callsign can be up to 6 characters. The maximum value of a legally encoded callsign field is F423FFFF. A value of all 1's (FFFFFFFF) is a broadcast address to all stations, addresses outside these limits are not used.

Payload Field

The payload field is dependent on the frame type and occupies the remainder of the frame up to the maximum size. If a source frame is too long to fit into the maximum, it is fragmented and sent in several frames, using the status, length and offset fields in the header to indicate how to reassemble the original packet.

UTF-8 Text Packet

This packet type is used by the built-in chat application and is not sent to the host processor. This application can be accessed by connecting to the node with a terminal program and accessing the internal menu.

Beacon Packet

This packet is both consumed internally to build a table of mesh entries and passed on to the host to create routing information. The packet consists of a beacon header and position information. Table 13 illustrates the beacon header.

Byte	Bit(s)	Field	Contents
1	7-5	Data Rate	Current data rate. See
	4	Extended Call	Primary station callsign is extended
	3	Repeat	Station can repeat frames
	2	AREDN	Node has a connection to the AREDN network
	1	OFDM	Node is OFDM capable
	0	FSK	Node is FSK capable
2	7-0	Tx Power	Transmit power in dBm
3	7-0	Firmware Major	Firmware version major digit
4	7-0	Firmware Minor	Firmware version minor digit

Table 13 Beacon packet header

The payload field of the beacon packet contains a position fix separated by commas. The position fix is either specified in the setup data or derived from a GPS receiver. The fields in the payload are described in Table 14.

Field	Purpose
Source	Source of the fix. 'FXD' when specified from setup data, 'GPS' from a receiver.
Latitude	In GPS format, DDMM.SSSS N/S
Longitude	In GPS format, DDMM.SSSS E/W
Speed	Speed across terrain. GPS receiver mode only.
Fix Time	Fix time in UTC, HHMMSS
Grid Square	Home grid square, in the format AANNmm

Table 14 Beacon Frame Conten

Command Packet

The command packet is consumed by the node and is intended to update the setup information from a remote application. It is currently unimplemented but will be added in future software releases. It is currently ignored.

Pi Communication Software

The Pi communication software (ip400spi) has been developed to provide a mechanism to send and receive packets to the node using the IP user datagram (UDP) protocol. The software is invoked on the command line, using the parameters shown in Table 15.

Parameter	Argument	Meaning
-B	None	Sends a single test frame, built into the software in frame.c
-s	/dev/spiX.Y	Specifies the SPI port. Usually /dev/spi0.0
-n	Name or IP	Name or IP address of the host receiving the packet
-p	nnnn	Port address of the host
-m	nnnn	Sending port address
-d	x	Sets the debug level to x.
-h	None	Prints the help message

Table 15 Pi utility software command parameters

The program is terminated by typing Control/C at the keyboard. It can also be run in the background, or as service.

The debug level is a bit field that has the following values:

Bit	Value	Debug
0	1	Enables logging of debug messages
1	2	Enables SPI debugging. Removes real time performance

Table 16 Pi Utility debug levels

Wireshark Dissector

A Wireshark dissector, using the generic dissector, has been developed to aid in debugging data transmissions between a Pi and other hosts. To install it, first download and install the generic dissector according to the instructions which can be found at http://wsgd.free.fr. Pick the correct version of the dissector for you Wireshark installation and place it in the plugins directory. This can be found on the Help->About Wireshark menu, at the 'Folders' tab.

Place the dissector script, ip400.wsgd, in the same directory and capture some UDP frames. If they show as a different protocol, go to Analyze->Enabled Protocols, and disable the one showing.

A sample dissected frame is shown in Figure 2.

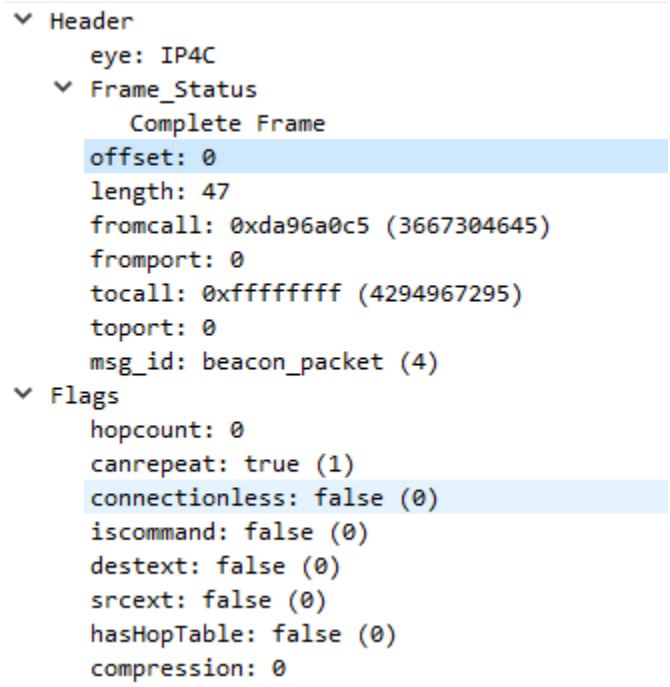

Figure 2 Sample Dissected Frame