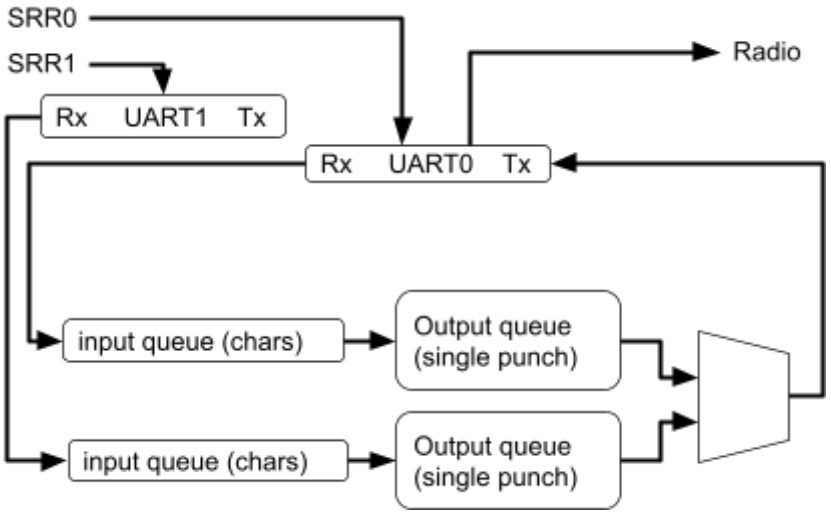
SerialBuffer for orienteering radio transmission

1. TBD:
   1.1.MeOs reports error in punch, but the reconstructs it.
   1.2. Use RTS or CTS from TinyMesh? the doc is not quite clear? Works with CTS (pin 6)
   1.3. Interrupt driven operation, at least for the Rx side, if necessary for speed.
   1.4. Improved LED indications: sine heartbeat when idle, long ON and OFF pulses for Tx/Rx
   1.5. Consider routing the TinyMesh module's Tx to SRR Rx through the Pico module, to facilitate later communication to the Pico or modifying the communication to the SRRs.
   1.6.Debug output on UART1 TXD, copying all input and output data.
2.Background
   2.1.The serial Buffer is a new HW/SW component in the FIF TinyMesh box that interfaces one or two SRRs to the Radio Module.
   2.2.The Serial Buffer solves a problem with the SRR to Radio Module data path: the SRR does not respond to the serial channel handshake signals, so the Radio Module cannot throttle data, and the arrival of two events close in time can cause loss of data. The radio module forces pauses in the data transmission for compliance with regulations, and data arriving during the pause can be lost.
   2.3. The solution is a RaspberryPi Pico module, connectng to multiple SRR modules, buffering and combining the data streams to one TinyMesh radio module.
   2.4. The SerialBuffer in context:



3. Dev Project:
   3.1.The IDE is Visual Studio Code on a RspberryPi RP400 minicomputer. The RP400 is also used as a test station, simulating the SRR and the TinyMesh radio via two serial channels.
   3.2. The Pico code is in C for speed, the test program is in Python.
   3.3.https://www.mathaelectronics.com/raspberry-pi-pico-a-complete-guide/
   3.4.Based on the uart_advanced example
   3.5.Started a new workspace, pico-projects, copy of the pico-examples workspace
   3.6.Based on the uart_advanced example, but substituted the interrupt-driven UART R/W with a poll loop.
   3.7.The RP400 is connected directly to the DBG pins, so no additional Pico as debug interface is required.

4. SerialBuffer structure
   4.1. Single channel, character based
      4.1.1.A FIFO buffer queue between Rx and Tx. The one used is not thread and interrupt safe, so need a better one.
      4.1.2.The queue is character based,not analyzing package structure.
      4.1.3.CTS/RTS: At the very least you will have to enable the CTS and RTS GPIO. You will

then have to connect those GPIO to the external device.

4.1.3.1. CTS is available on GPIO16.

4.1.3.2. RTS is available on GPIO17.

4.1.3.3. For UART 0 (I guess ttyS0) you will have to place those GPIO in mode ALT3. For UART 1 you will have to place those GPIO in mode ALT5.

4.1.3.4. See page 102 of BCM2835 ARM Peripherals

4.1.3.5. https://raspberrypi.github.io/pico-sdk-doxygen/group__hardware__gpio.html#ga2af81373f9f5764ac1a5bd6097477530

4.1.4. Only enabling the handshake for the Pico Tx, since we assume that the Rx must aways be ready.

4.2. Dual input channel, package based

4.2.1. Dual FIFO buffers in two layers: a large, byte-based input buffer, followed by a small package based output buffer. The package buffers are multiplexed to the serial output UART, maintaining contiguous punches from both channels.

4.2.2. Poll based, in a single execution loop, unless the test shows that the loop is too slow, missing input punches from an SRR.

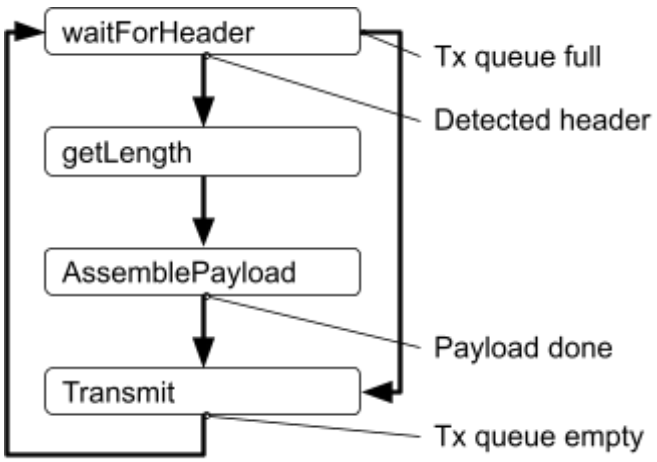
4.2.3. Writes into the input queue are not monitored for content.

4.2.4. Continuous reads from the main large buffers into a secondary single-package buffer. When a package buffer holds a complete punch, it is sent to the UART in a contiguous stream. The punch is defined by the header byte 0xD3 and the length in the LEN field. Any chars before the header are transmitted, in particular, the 0x2 byte that precedes the header in the new record format.

4.2.5. No control of record length or CRC is done.

4.2.6. A state machine keeps track of the punches in each channel, collecting a complete punch in a queue before seizing the output UART and sending a complete punch. Note that the order of the punches is not guaranteed.

4.2.7. The state machne controlling each channel:

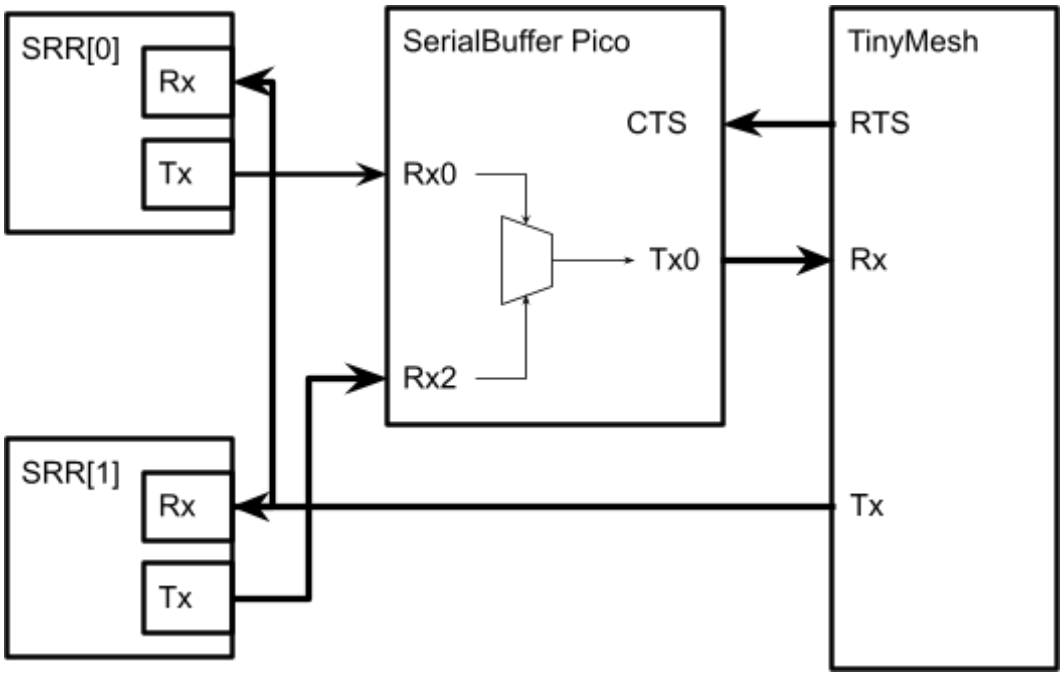


Serial buffer states

4.2.8. Visual indictor: The LED blinks rapidly 10 times on power on. Then it will turn ON on Rx, OFF on Tx

4.2.9. Internal pull-ups are enabled on the Pico RXD inputs, to enable operation with a single SRR.

5.Connections:



Serial buffer inserted between SRR units and the TinyMesh radio module

RP400 function	RP400 pin	SRR function	SRR pin	Pico function	Pico pin	TinyMesh function	TinyMesh pin
DBG SWDIO	18	-	-	SWDIO	debug - SWDIO	-	-
DBG GND	20	-	-	GND	debug - GND	-	-
DBG CLK	22	-	-	CLK	debug - SWCLK	-	-
UART0 Rx	10	-	-	UART0 Tx (GP0)	1	RXD	2
UART0 Tx	8	(SRR0) Tx	2	UART0 Rx (GP1)	2	-	
UART0 RTS	11	-		UART0 CTS, (GP2)	4	RTS (CTS?)	5 (6?)
UART2 Tx	27	(SRR1) Tx	2	UART1 Rx (GP4)	7	-	
	-			UART 1 Tx (GP5)	6	-	
UART2 Rx	28			-	(unused)	-	
UART2 RTS	-						
GND		GND	1	GND (signal)	3,8,13,18,23,28, 33	GND	1
-	-			GND (Power)	38	GND	

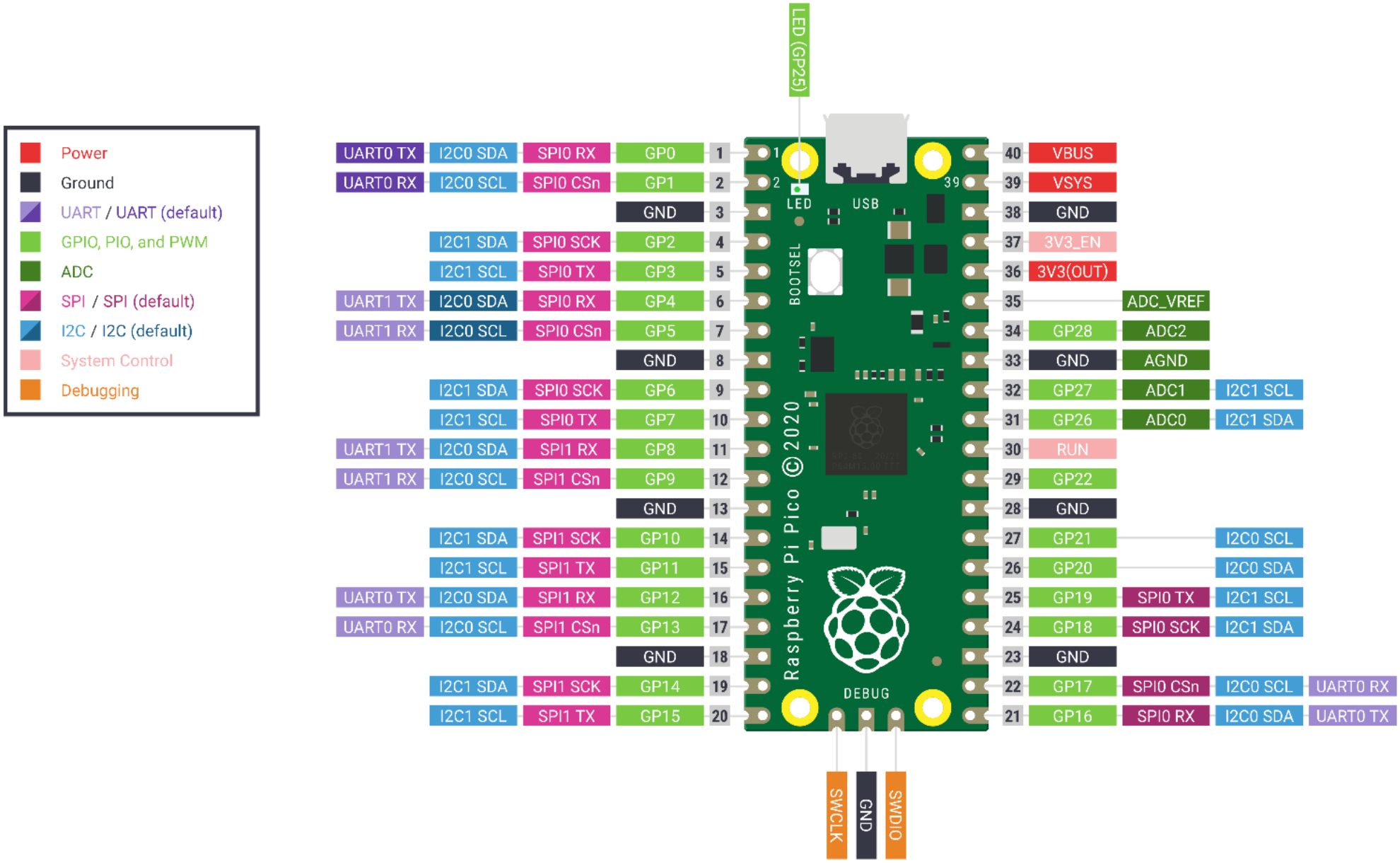
-	-	VCC	7	Power: VSYS, 2 - 5.5 V	39	VCC	7
-	-	-	-	-	-	CONFIG	4
						CTS	6

6. Test:
    6.1. The RP400 used as test machine
    6.2. A Python script "SerialTest" sends data packages, blocking the Rx channel to force filling the buffer. Then It reads the whole buffer, comparing the Rx data to an internal test buffer.
    6.3. various buffer levels are tested, ending with an overfilled buffer to force the test to fail, validating the test method.
    6.4. By default, the primary UART (UART0) is assigned to the Linux console. If you wish to use the primary UART for other purposes, you must reconfigure Raspberry Pi OS. This can be done by using raspi-config
    6.5. For 2-channel test, the RP400 UART2 is used, since it is identical to UART1, whereas UART1 is a reduced functionality type. UART2 is used to transmit half of the punches (randomised).
        6.5.1. In order to use UART2, add the following line to /boot/config.txt: *dtoverlay=uart2*. Then it can be accessed as /dev/ttyAMA1

RP400 pinout:

GPI #	Pin #	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0		SDA0	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
1		SCL0	SA4	DE	SPI3_MISO	RXD2	SCL6
2		SDA1	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
3		SCL1	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3
4		GPCLK0	SA1	DPI_D0	SPI4_CE0_N	TXD3	SDA3
5		GPCLK1	SAO	DPI_D1	SPI4_MISO	RXD3	SCL3
6		GPCLK2	SOE_N	DPI_D2	SPI4_MOSI	CTS3	SDA4
7		SPI0_CE1_N	SWE_N	DPI_D3	SPI4_SCLK	RTS3	SCL4
8		SPI0_CE0_N	SDO	DPI_D4	_	TXD4	SDA4
9		SPI0_MISO	SD1	DPI_D5	_	RXD4	SCL4
10		SPI0_MOSI	SD2	DPI_D6	_	CTS4	SDA5
11		SPI0_SCLK	SD3	DPI_D7	_	RTS4	SCL5
12		PWM0	SD4	DPI_D8	SPI5_CE0_N	TXD5	SDA5
13		PWM1	SD5	DPI_D9	SPI5_MISO	RXD5	SCL5
14		TXD0	SD6	DPI_D10	SPI5_MOSI	CTS5	TXD1
15		RXD0	SD7	DPI_D11	SPI5_SCLK	RTS5	RXD1
16		FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
17		FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
18		PCM_CLK	SD10	DPI_D14	SPI6_CE0_N	SPI1_CE0_N	PWM0
19		PCM_FS	SD11	DPI_D15	SPI6_MISO	SPI1_MISO	PWM1
20		PCM_DIN	SD12	DPI_D16	SPI6_MOSI	SPIl_MOSI	GPCLK0
21		PCM_DOUT	SD13	DPI_D17	SPI6_SCLK	SPI1_SCLK	GPCLK1
22		SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	SDA6
23		SD0_XMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	SCL6
24		SD0_DATO	SD16	DPI_D20	SD1_DAT0	ARM_TDO	SPI3_CE1_N
25		SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	SPI4_CE1_N
26		SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	SPI5_CE1_N
27		SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	SPI6_CE1_N

Pico pinout


Power
Ground
UART / UART (default)
GPIO, PIO, and PWM
ADC
SPI / SPI (default)
I2C / I2C (default)
System Control
Debugging
LED (GP25)
LED
USB
BOOTSEL
DEBUG
Raspberry Pi Pico ©2020
SWCLK
GND
SWDIO
1 UART0 TX I2C0 SDA SPI0 RX GP0
2 UART0 RX I2C0 SCL SPI0 CSn GP1
3 GND
4 I2C1 SDA SPI0 SCK GP2
5 I2C1 SCL SPI0 TX GP3
6 UART1 TX I2C0 SDA SPI0 RX GP4
7 UART1 RX I2C0 SCL SPI0 CSn GP5
8 GND
9 I2C1 SDA SPI0 SCK GP6
10 I2C1 SCL SPI0 TX GP7
11 UART1 TX I2C0 SDA SPI1 RX GP8
12 UART1 RX I2C0 SCL SPI1 CSn GP9
13 GND
14 I2C1 SDA SPI1 SCK GP10
15 I2C1 SCL SPI1 TX GP11
16 UART0 TX I2C0 SDA SPI1 RX GP12
17 UART0 RX I2C0 SCL SPI1 CSn GP13
18 GND
19 I2C1 SDA SPI1 SCK GP14
20 I2C1 SCL SPI1 TX GP15
40 VBUS
39 VSYS
38 GND
37 3V3_EN
36 3V3(OUT)
35 ADC_VREF
34 GP28 ADC2
33 GND AGND
32 GP27 ADC1 I2C1 SCL
31 GP26 ADC0 I2C1 SDA
30 RUN
29 GP22
28 GND
27 GP21 I2C0 SCL
26 GP20 I2C0 SDA
25 GP19 SPI0 TX I2C1 SCL
24 GP18 SPI0 SCK I2C1 SDA
23 GND
22 GP17 SPI0 CSn I2C0 SCL UART0 RX
21 GP16 SPI0 RX I2C0 SDA UART0 TX

Pico I/O

	Function								
GPIO	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1		USB OVCUR DET
1	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1		USB VBUS DET
2	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1		USB VBUS EN
3	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1		USB OVCUR DET
4	SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1		USB VBUS DET
5	SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1		USB VBUS EN
6	SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1		USB OVCUR DET
7	SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1		USB VBUS DET
8	SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1		USB VBUS EN
9	SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1		USB OVCUR DET
10	SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1		USB VBUS DET
11	SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1		USB VBUS EN
12	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1		USB OVCUR DET
13	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1		USB VBUS DET
14	SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1		USB VBUS EN
15	SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1		USB OVCUR DET
16	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1		USB VBUS DET
17	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1		USB VBUS EN
18	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1		USB OVCUR DET
19	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1		USB VBUS DET
20	SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	CLOCK GPIN0	USB VBUS EN
21	SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	CLOCK GPOUT0	USB OVCUR DET
22	SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	CLOCK GPIN1	USB VBUS DET
23	SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	CLOCK GPOUT1	USB VBUS EN
24	SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	CLOCK GPOUT2	USB OVCUR DET
25	SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	CLOCK GPOUT3	USB VBUS DET
26	SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1		USB VBUS EN
27	SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1		USB OVCUR DET
28	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1		USB VBUS DET
29	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1		USB VBUS EN

SRR Protocol
Message structure:

SIAC: 0xD3 - LEN - CNH - CNL - SIID3 - SIID2 - SIID1 - SIID0 - CNH/AM/PM -TH - TL - TSS - MODE - CNTR - CNTT - CRC1 - CRC0

Oskar: "\x02\xD3\x0D\x00\x2A\x00\x1F\x87\x68\x0A\x81\x03\xD0\x00\x01\x10\xF3\x65\x03"

Nr	Oskar	Siac	
0	0x02		Only in new formats
1	0xD3	D3	Header byte, constant
2	LEN	LEN	Always 0x0D = 13
3	0x00	CNH	Station ID
4	0x27	CNL	
5	0x00	SIID3	Chip ID
6	0x00	SIID2	
7	SI1	SIID1	
8	SI0	SIID0	
9	0x05	CNH/AM/PM	Time
10	0x5E	TH	
11	0xB2	TL	
12	0x59	TSS	
13	0x00	MODE	
14	0x0E	CNTR	
15	0x68	CNTT	
16	CRC1	CRC1	CRC
17	CRC0	CRC0	CRC
	0x03		Only in new format?