**SerialBuffer for orienteering radio transmission**

1. Background
   1. The serial Buffer is a new HW/SW component in the FIF TinyMesh box that interfaces the SRR to the Radio Module.
   2. The Serial Buffer solves a problem with the SRR to Radio Module sata path: the SRR does not respont 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, an data arriving during the pause can be lost.
   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. Dev Project:
   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 to serial channels.
   2. The Pico code is in C for speed, the test program is in Python.
   3. https://www.mathaelectronics.com/raspberry-pi-pico-a-complete-guide/
   4. Based on the uart\_advanced example
   5. Started a new workspace, pico-projects, copy of the pico-examples workspace
   6. Based on the uart\_advanced example, but substituted the interrupt-driven UART R/w with a poll loop.
   7. The RP400 is connected directly to the DBG pins, so no additional Pico as debug interface is required.
3. SerialBuffer structure
   1. Single channel, character based
      1. A FIFO buffer queue between Rx and Tx. The one used is not thread and interrupt safe, so need a better one.
      2. The queue is character based,not analyzing package structure.
      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.
         1. CTS is available on GPIO16.
         2. RTS is available on GPIO17.
         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. See page 102 of [BCM2835 ARM Peripherals](http://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf)
         5. https://raspberrypi.github.io/pico-sdk-doxygen/group\_\_hardware\_\_gpio.html#ga2af81373f9f5764ac1a5bd6097477530
      4. Only enabling the handshake for the Pico Tx, since we assume that the Rx must aways be ready.
   2. Dual input channel, package based
      1. 
      2. Dual FIFO buffers in two layers: a large stream buffer, followed by a small package buffer. The package buffers are multiplexed to the serial output UART, maintaining contiguous punches from both channels.
      3. Poll based, in a single execution loop, unless the test shows that the loop is too
      4. slow, missing input punches from an SRR
      5. Writes into the buffer are not monitored for content
      6. 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.
      7. No control of record length or CRC is done.
      8. 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.

Serial buffer states

1. Connections:



Serial buffer inserted between SRR units and the TinyMesh radio module

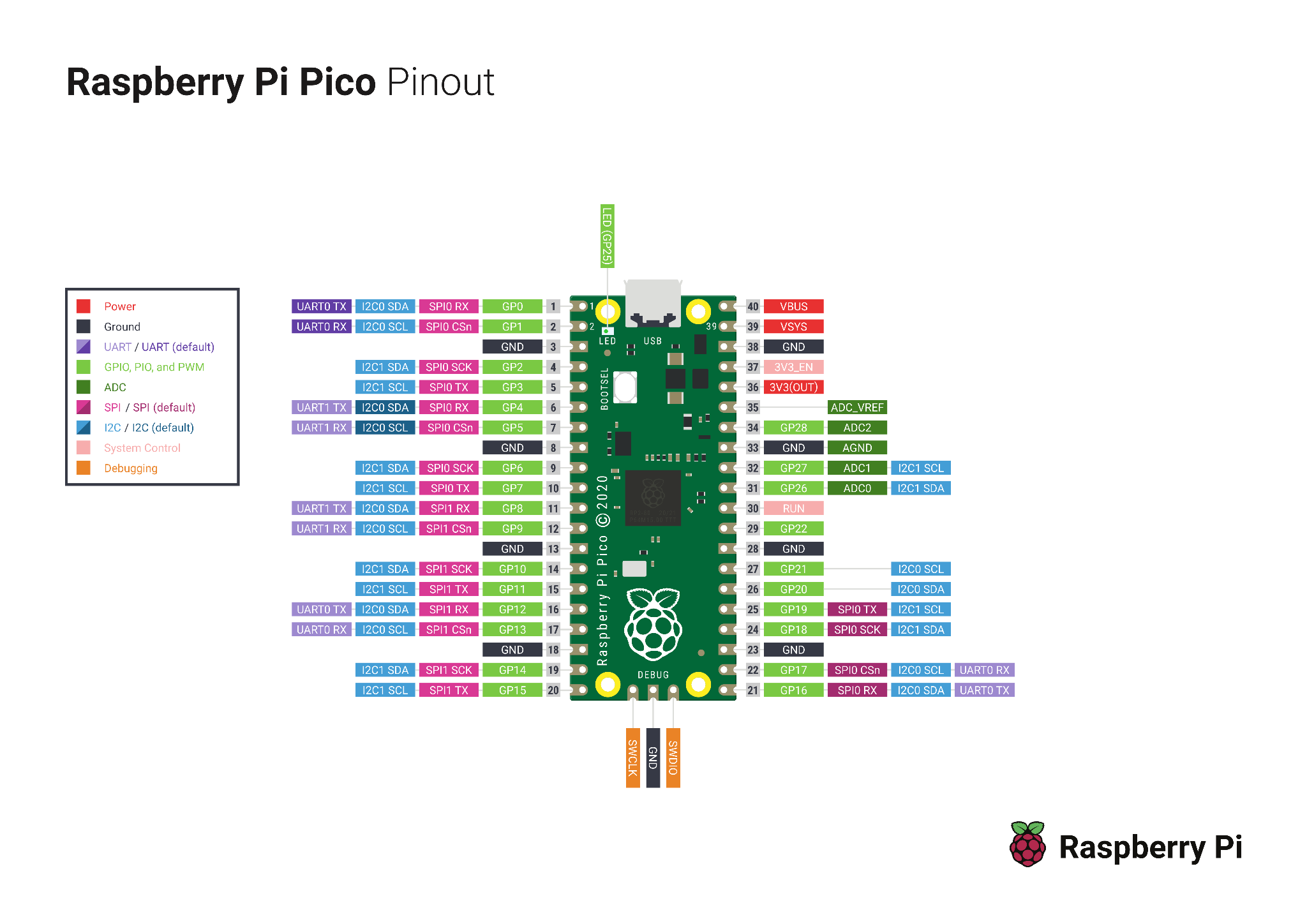
| RP400  function | RP400 pin | Pico function | Pico pin | SRR function | SRR pin | TinyMesh function | TinyMesh  module pin |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DBG SWDIO | 22 | SWDIO | debug - SWDIO | - | - | - | - |
| DBG GND | 20 | GND | debug - GND | - | - | - | - |
| DBG CLK | 18 | CLK | debug - SWCLK | - | - | - | - |
| UART0 Rx | 10 | UART0 Tx (GP0) | 1 | - |  | Tx |  |
| UART0 Tx | 8 | UART0 Rx (GP1) | 2 | (SRR0) Tx |  | Rx | Rx<=>Tx? |
| UART0 RTS | 11 | UART0 CTS, (GP2) | 4 | - |  | RTS |  |
| UART2 Tx | 27 | UART1 Rx (GP4) | 6 |  |  | - |  |
|  | - | UART 1 Tx (GP5) | 7 |  |  | - |  |
| UART2 Rx | 28 | - | (unused) |  |  | Tx |  |
| UART2 RTS | - | - |  |  |  |  |  |
| GND |  | GND (signal) | 3,8,13,18,23,28,33 | GND |  | GND |  |
| - | - | GND (Power) | 38 |  |  | GND |  |
| - | - | Power: VSYS, 2 - 5.5 V | 39 |  |  | VCC |  |

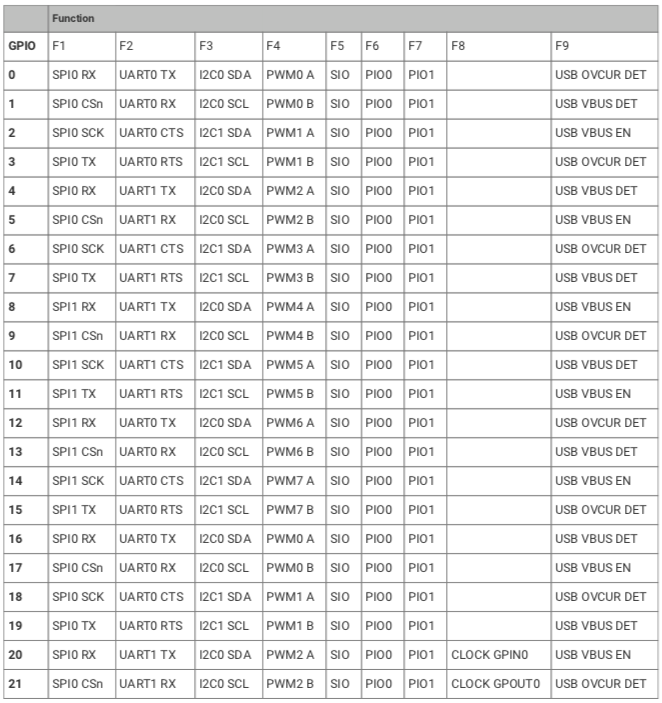
1. Test:
   1. The RP400 used as test machine
   2. A Python script "SerialTest" sends data packages, blocking the Rx channel to force filling the buffer. Then It reads the whole buffer, comparing the Rx data to an internal test buffer.
   3. various buffer levels are tested, ending with an overfilled buffer to force the test to fail, validating the test method.
   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](https://www.raspberrypi.com/documentation/computers/configuration.html#raspi-config)
   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).
      1. In order to use UART2, add the following line to /boot/config.txt: *dtoverlay=uart2*. Then it can be accessed as /dev/ttyAMA1
2. TBD:
   1. Interrupt driven operation, at least for the Rx side, if necessary for speed.
   2. Improved LED indications: sine heartbeat when idle, long ON and OFF pulses for Tx/Rx
   3. 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.

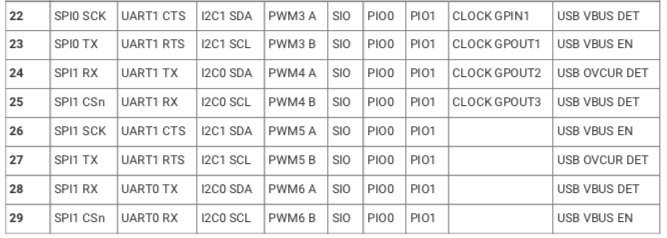
RP400 pinout:

| 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







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? |