CYW43439 Bluetooth

The CYW4349 wifi and bluetooth chip is used in the Raspberry Pico Pi W. The C SDK includes the required drivers and stacks to run this for bluetooth and WIFI.

This document describes how the chip interface works for blueooth, with the aim to create a Micropython implementation for a HCI interface layer.

Conventions used for numbering

Because this description uses both byte arrays and u32 values, the following is used to clarify which is in use.
All u32 values are used with underscore separators at byte boundaries, so a hex number 0x12345678 would be shown as 12_34_56_78.

When this is stored in a byte array, this may be with different word lengths and little endian or big endian. Byte arrays do not use the underscore, so a little endian array for the value above would be shown as 78 56 34 12, and a big endian representation is 12 34 56 78

HCI interface

The HCI layer has the following format. The header for the SPI packet is the length plus HCI event.

Length is number of bytes in the data in the packet, which excluded the HCI event.

The header for the SPI packet is the length plus HCI event.

Packet header		
Length	**HCI event**	**Data**
3 bytes	1 byte	n bytes

Packet format

Packet header		
Length	**HCI event**	**Data**
06 00 00	04	0e 04 01 03 0c 00

Example packet

CYW4349 interface

The CYW4349 interface uses half-duplex SPI to communicate to registers and memory in the chip.

With SPI every data transfer is 32 bits, so all data is 32 bit aligned and the length rounded up to the next 32 bit value.

Each command to the CYW43439 has a 32 bit command field and then following data values.

Command	Data
32 bits	n x 32 bits

SPI command format

Write	Incr	Function	Address	Size
1 bit	1 bit	2 bits	17 bits	11 bits

SPI command field layout

With this bit layout.

0	0	00	00	00	00	00	00	00	00	00	0	0	00	00	00	00	00
wr	in	fn	address									size					
1	1	0-3	0 00 00 to 1 ff ff									0 00 to 3 ff					

Bit layout of command field

w	i	ff	aaaa	aaaa	aaaa	aaaa	a sss	ssss	ssss

32-bit word layout of command field

Field	Explanation
Write / read	1 Write to CYW43439 0 Read from CYW43439
Increment	
Function	0 BUS 1 BACKPLANE 2 WLAN
Address	Address in memory or of register
Size	Size of data (not rounded to 32 bits)

Explanation of fields

The address depends on which Function is being accessed.

Backplane addresses

Backplane memory and registers are accessed via a 15 bit address range, from 0x0 to 0x7fff. This creates a window into the full memory range and the base address for this window can be set.

As the address field is 17 bits, there are two special bits.

Bit 16 is set to show a 32 bit (4 byte) data transfer, so is always set for SPI transactions (0x8000)

Bit 17 is unknown but is set for the backplane registers (0x1 000a throught 0x1 000c), so may indicate a fixed address regardless of the current backplane settings.

Key register addresses

Function	Address	Field	Explanation
BACKPLANE	18_00_0c_7c	BT CTRL	Controls bluetooth
BACKPLANE	18_00_0d_6c	HOST CTL	Controls host
BACKPLANE	18_00_0d_68	BASE ADDRESS	Base address for WIFI (and buffers)

Key registers for bluetooth

Register address	Field	Mask	Explanation
HOST CTRL	DATA VALID	00_00_00_02	Tell chip data is now valid (toggle value)
HOST CTRL	WAKE BT	00_02_00_00	Wake up the BT chip
HOST CTRL	SW READY	01_00_00_00	Set host as ready
BT CTRL	BT AWAKE	00_00_01_00	Is the BT chip awake
BT CTRL	FW READY	01_00_00_00	Is the BT chip ready

Bit fields for key registers

Setting the backplane base address

Addresses are 32 bit. The chipset base address for the backplane window is 0x18 00 00 00.

The backplane window is set as the top three bytes of the full address.

Any memory access is therefore made up of the command address and the backplane address.

	Byte 3	Byte 2	Byte 1	Byte 0
Backplane address	ff	ff	ff	
Field address		01	ff	ff

Mapping of addresses via backplane and field address

Function	Address	Field	Explanation
BACKPLANE	1_000a	WINDOW LOW	Byte 1 of window address
BACKPLANE	1_000b	WINDOW MID	Byte 2 of window address
BACKPLANE	1_000c	WINDOW HIGH	Byte 3 of window address

Backplane address mask registers

Transmit and receive buffers

To send HCI data to the chip and receive data from the chip, data is stored in transmit and receive circular buffers in the chip memory.

Each buffer is 0xfff bytes long (so is larger than the backplane window)

These have a specific address range, offset from the WIFI base address.

The buffer for host to chip data is offset at 0x000 and the chip to host buffer is at 0x1000

There are also pointers for the head and tail of each buffer stored in registers.

The key registers and memory locations in use are:

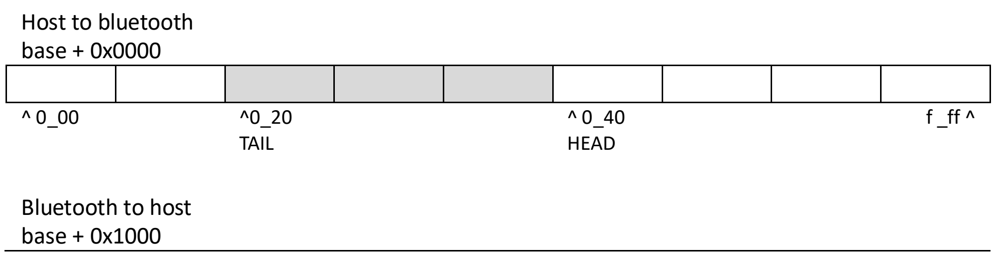
Address		Memory location (offest from WIFI base)
Host to bluetooth buffer		00_00
Bluetooth to host buffer		10_00
Host to bluetooth	Head	20_00
	Tail	20_04
Bluetooth to host	Head	20_08
	Tail	20_0a

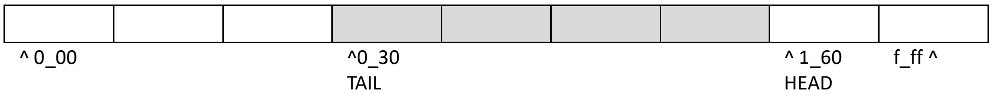
Buffer registers

The example shows the two buffers in use, with the head and tail locations marked. WIFI base is 0x6_861c. To reach this the backplane window is set to 0x00_06_80_00 and offset of 0x06_1c is used (or 0x16_1c or 0x26_1c). When the 0x80 00 SPI mask is added, this becomes 0x86_1c (or 0x96_1c or 0xa6_1c).

			Example	
Address		**Memory location**	**Address**	**Values**
Host to bluetooth buffer		00_00	6_86_1c	
Bluetooth to host buffer		10_00	6_96_1c	
Host to bluetooth	Head	20_00	6_a6_1c	0_40
	Tail	20_04	6_a6_20	0_20
Bluetooth to host	Head	20_08	6_a6_24	1_60
	Tail	20_0a	6_a6_28	0_30

Buffer example





Buffer diagram

SPI interface

The SPI interface is half-duplex, meaning it only has a single data line and can therefore only transmit or receive at one time.

The GPIO pins used are shown below.

Pin	Name	Explanation
23	REG ON	Power on
24	DATA OUT	Data out
24	DATA IN	Data in
24	IRQ	Interrupt line
25	CS	Chip select
29	CLOCK	SPI clock

GPIO pins for SPI interface

Transfers are in units of 32 bits. A 32-bit word is written to the BT chip, then a 32 bit word is read from the chip.

In a simple transfer, a register write, two 32-bits words are written and the read data is discarded.

For a backplane read, a buffer is inserted between the command and the received data. This allows for the chip to process the command and generate the response.

The length of the buffer is stored in register 0x01c (Response delay register F1). By default this is 4 bytes. For the Pico Pi CYW43439 driver this is set at 16 bytes (or 4 32-bit words).

Direction	**Command**	**Value**
Write	Write to 00_06_a6_1c	00_00_00_08
Read	00_00_00_00	00_00_00_00

Simple write to register

Direction	**Command**	**Value**				
Write	Read from 8d_6c	00_00_00_00	00_00_00_00	00_00_00_00	00_00_00_00	00_00_00_00
Read	00_00_00_00	00_00_00_00	00_00_00_00	00_00_00_00	00_00_00_00	00_01_00_01

Simple read from register

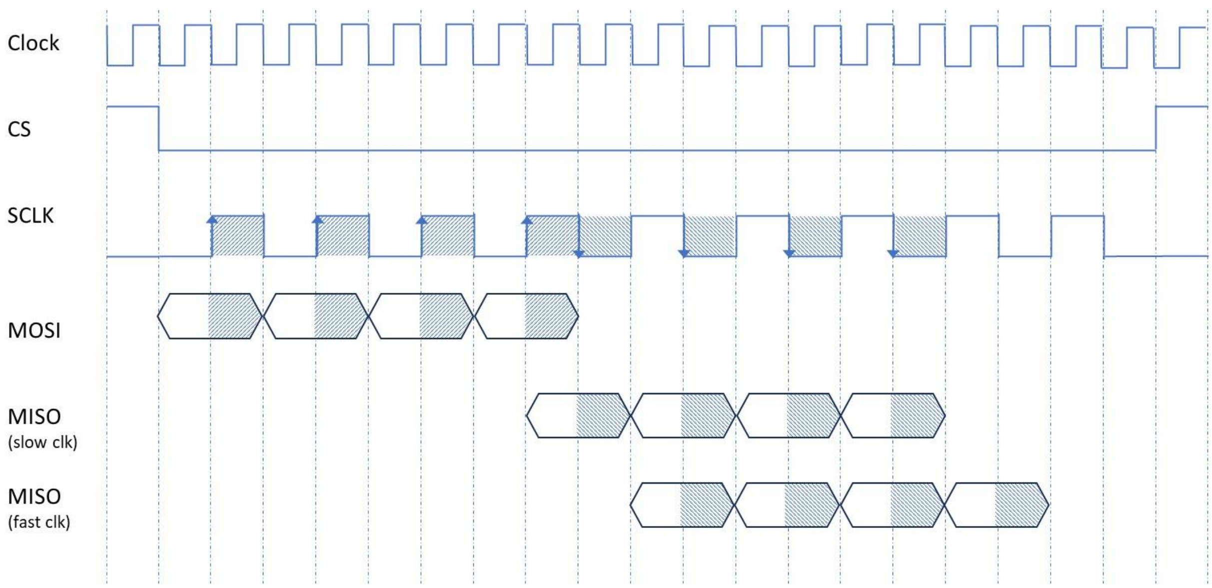
SPI physical interface

SPI from the Pico to the CYW43439 is half-duplex, which is implemented by sharing that data line between MISO, MOSI and the interrupt pin (GPIO 24).

It is also slightly complex because the write is performed before the rising edge of the clock, and the read after the falling edge of the clock. The gap between write and read seems to vary depending on the clock speed, so a slow clock allows the data to be presented to GPIO24 before the falling edge, but a fast clock then requires another cycle before the data is ready.

This is referred to as High Speed mode in the CYW specification (but not reflected in the SPI timing diagrams in that document).

Data transfers are 32 bit words and big endian, which is selected with an option on register 0x00. Before this is set the CYW defaults to 16 bit little-endian, so the command and returned data must be swapped until these options are set.



Endian settings

The CW43439 can be in one of four word length and endian settings. The chips boots in 16 bit little endian.

Command to read the test register is

wr	in	fn	address	size
0	1	00	0 00 14	0 04

w	i	ff	aaaa	aaaa	aaaa	aaaa	a sss	ssss	ssss
0	1	00	0000	0000	0000	1010	0 000	0000	0004

The command value (u32) is 40_00_A0_04 and the test register value (u32) is FE_ED_BE_AD – which corresponded to 32 bit little endian in the table.

Type	Command	Test data
u32 value	40_00_A0_04	FE_ED_BE_AD
Big endian 32 bit word	40 00 A0 04	FE ED BE AD
Little endian 32 bit word	04 A0 00 40	AD BE ED FE
Little endian 16 bit word	A0 04 40 00	BE AD FE ED

Endian	**Word size**	**Register x00 (endian / word size)**	**How this maps to bytes b3 b2 b1 b0**	**Command example bytes 40_00_A0_04**	**FEEDBEAD example bytes FE_ED_BE_AD**
Big endian	32 bit	11	b0 b1 b2 b3	04 A0 00 40	AD BE ED FE
Big endian	16 bit	10	b0 b1 b2 b3	04 A0 00 40	AD BE ED FE
Little endian	32 bit	01	b3 b2 b1 b0	40 00 A0 04	FE ED BE AD
Little endian	16 bit	00 (default)	b1 b0 b3 b2	A0 04 40 00	BE AD FE ED

Simple read from register

Note that 32 bit big endian is actually the same as 16 bit big endian.

In default mode the command sent is A0 04 40 00 and the data received is BE AD FE ED. This is word swapped to AD BE ED FE, which is now little endian for FE_ED_BE_AD.

So the default setting is little endian 16 bit as per the table above, with byte order b1 b0 b3 b2.

Setting big endian 32 bit, though, appears to be byte order b0 b1 b2 b3 which is little endian.