Chapter 1. Introduction

Microcontrollers connect the world of software to the world of hardware. They allow developers to write software which interacts with the physical world in the same deterministic, cycle-accurate manner as digital logic. They occupy the bottom left corner of the price/performance space, outselling their more powerful brethren by a factor of ten to one. They are the workhorses that power the digital transformation of our world.

RP2040 is the debut microcontroller from Raspberry Pi. It brings our signature values of high performance, low cost, and ease of use to the microcontroller space.

With a large on-chip memory, symmetric dual-core processor complex, deterministic bus fabric, and rich peripheral set augmented with our unique Programmable I/O (PIO) subsystem, it provides professional users with unrivalled power and flexibility. With detailed documentation, a polished MicroPython port, and a UF2 bootloader in ROM, it has the lowest possible barrier to entry for beginner and hobbyist users.

RP2040 is a stateless device, with support for cached execute-in-place from external QSPI memory. This design decision allows you to choose the appropriate density of non-volatile storage for your application, and to benefit from the low pricing of commodity Flash parts.

RP2040 is manufactured on a modern 40nm process node, delivering high performance, low dynamic power consumption, and low leakage, with a variety of low-power modes to support extended-duration operation on battery power.

Key features:

- Dual ARM Cortex-M0+ @ 133MHz
- 264kB on-chip SRAM in six independent banks
- Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus
- DMA controller
- Fully-connected AHB crossbar
- Interpolator and integer divider peripherals
- On-chip programmable LDO to generate core voltage
- 2 on-chip PLLs to generate USB and core clocks
- 30 GPIO pins, 4 of which can be used as analogue inputs
- Peripherals
    - 2 UARTs
    - 2 SPI controllers
    - 2 I2C controllers
    - 16 PWM channels
    - USB 1.1 controller and PHY, with host and device support
    - 8 PIO state machines

Whatever your microcontroller application, from machine learning to motor control, from agriculture to audio, RP2040 has the performance, feature set, and support to make your product fly.

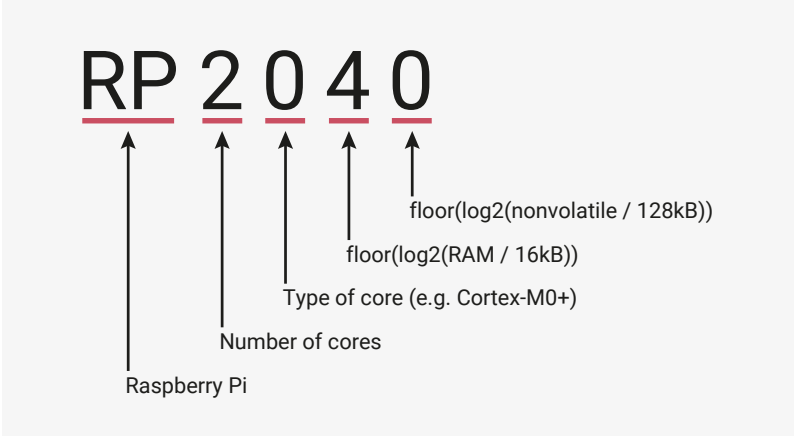
1.1. Why is the chip called RP2040?

The post-fix numeral on RP2040 comes from the following,

1. Number of processor cores (2)
2. Loosely which type of processor (M0+)
3. floor(log2(RAM / 16kB))
4. floor(log2(nonvolatile / 128kB)) or 0 if no onboard nonvolatile storage

see Figure 1.

Figure 1. An explanation for the name of the RP2040 chip.



1.2. Summary

RP2040 is a low-cost, high-performance microcontroller device with flexible digital interfaces. Key features:

- Dual Cortex M0+ processor cores, up to 133MHz (or 200MHz at 1.15V, see Section 2.15.3)
- 264kB of embedded SRAM in 6 banks
- 30 multifunction GPIO
- 6 dedicated IO for SPI Flash (supporting XIP)
- Dedicated hardware for commonly used peripherals
- Programmable IO for extended peripheral support
- 4 channel ADC with internal temperature sensor, 500ksps, 12-bit conversion
- USB 1.1 Host/Device

1.3. The Chip

RP2040 has a dual M0+ processor cores, DMA, internal memory and peripheral blocks connected via AHB/APB bus fabric.

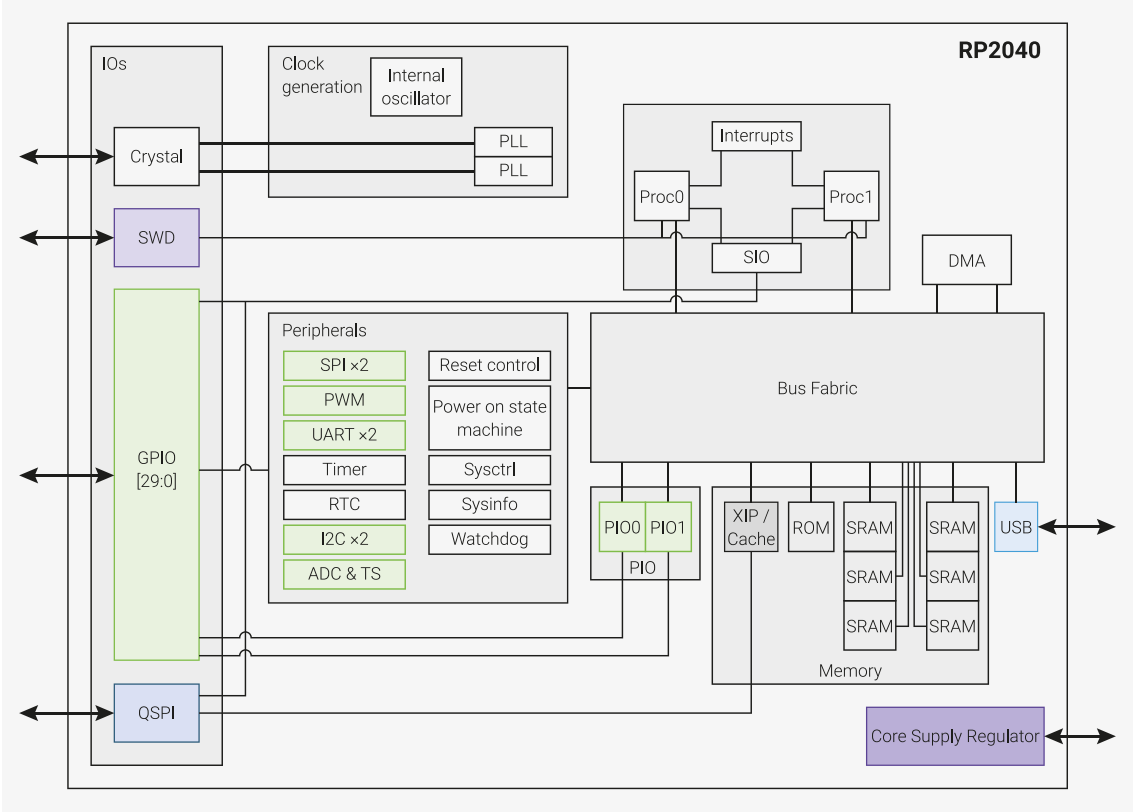


Figure 2. A system overview of the RP2040 chip

Code may be executed directly from external memory through a dedicated SPI, DSPI or QSPI interface. A small cache improves performance for typical applications.

Debug is available via the SWD interface.

Internal SRAM can contain code or data. It is addressed as a single 264 kB region, but physically partitioned into 6 banks to allow simultaneous parallel access from different masters.

DMA bus masters are available to offload repetitive data transfer tasks from the processors.

GPIO pins can be driven directly, or from a variety of dedicated logic functions.

Dedicated hardware for fixed functions such as SPI, I2C, UART.

Flexible configurable PIO controllers can be used to provide a wide variety of IO functions.

A USB controller with embedded PHY can be used to provide FS/LS Host or Device connectivity under software control.

Four ADC inputs which are shared with GPIO pins.

Two PLLs to provide a fixed 48MHz clock for USB or ADC, and a flexible system clock up to 133MHz.

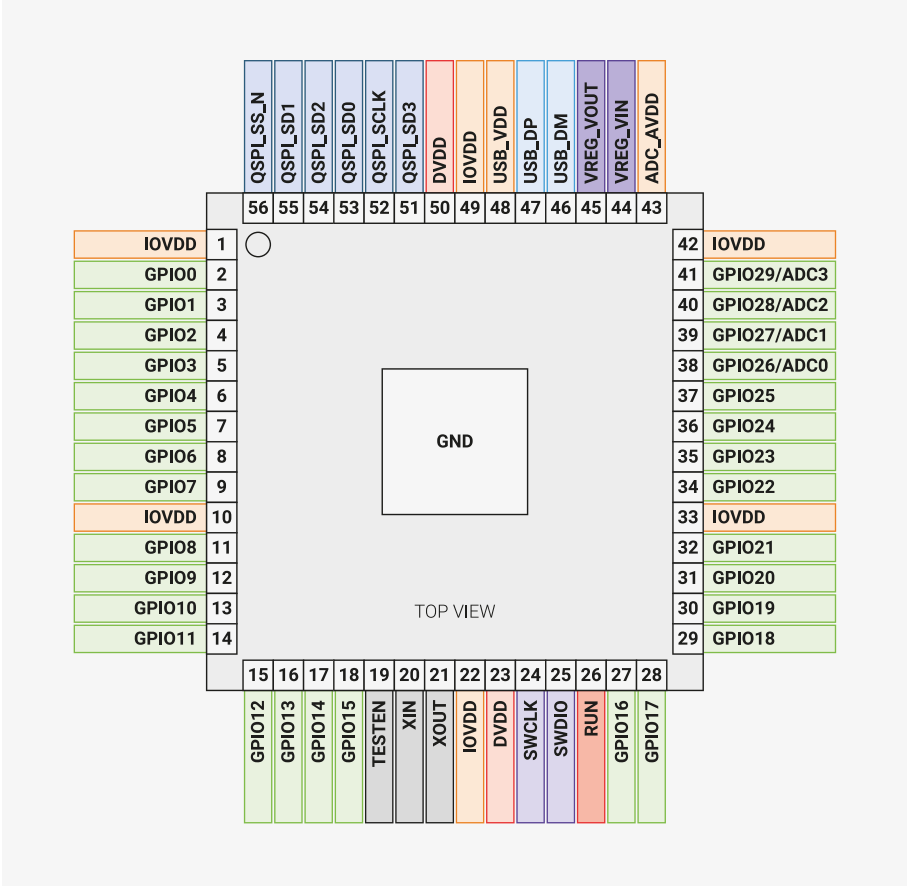
An internal Voltage Regulator to supply the core voltage so the end product only needs supply the IO voltage.

1.4. Pinout Reference

This section provides a quick reference for pinout and pin functions. Full details, including electrical specifications and package drawings, can be found in Chapter 5.

1.4.1. Pin Locations

Figure 3. RP2040 Pinout for QFN-56 7×7mm (reduced ePad size)



1.4.2. Pin Descriptions

Table 1. The function of each pin is briefly described here. Full electrical specifications can be found in Chapter 5.

Name	Description
GPIOx	General-purpose digital input and output. RP2040 can connect one of a number of internal peripherals to each GPIO, or control GPIOs directly from software.
GPIOx/ADCy	General-purpose digital input and output, with analogue-to-digital converter function. The RP2040 ADC has an analogue multiplexer which can select any one of these pins, and sample the voltage.
QSPIx	Interface to a SPI, Dual-SPI or Quad-SPI flash device, with execute-in-place support. These pins can also be used as software-controlled GPIOs, if they are not required for flash access.
USB_DM and USB_DP	USB controller, supporting Full Speed device and Full/Low Speed host. A 27Ω series termination resistor is required on each pin, but bus pull-ups and pull-downs are provided internally.
XIN and XOUT	Connect a crystal to RP2040's crystal oscillator. XIN can also be used as a single-ended CMOS clock input, with XOUT disconnected. The USB bootloader requires a 12MHz crystal or 12MHz clock input. For recommended crystals, see Crystal Oscillator (Section 2.16).
RUN	Global asynchronous reset pin. Reset when driven low, run when driven high. If no external reset is required, this pin can be tied directly to IOVDD.
SWCLK and SWDIO	Access to the internal Serial Wire Debug multi-drop bus. Provides debug access to both processors, and can be used to download code.
TESTEN	Factory test mode pin. Tie to GND.
GND	Single external ground connection, bonded to a number of internal ground pads on the RP2040 die.
IOVDD	Power supply for digital GPIOs, nominal voltage 1.8V to 3.3V

Name	Description
USB_VDD	Power supply for internal USB Full Speed PHY, nominal voltage 3.3V
ADC_AVDD	Power supply for analogue-to-digital converter, nominal voltage 3.3V
VREG_VIN	Power input for the internal core voltage regulator, nominal voltage 1.8V to 3.3V
VREG_VOUT	Power output for the internal core voltage regulator, nominal voltage 1.1V, 100mA max current
DVDD	Digital core power supply, nominal voltage 1.1V. Can be connected to VREG_VOUT, or to some other board-level power supply.

1.4.3. GPIO Functions

Each individual GPIO pin can be connected to an internal peripheral via the GPIO functions defined below. Some internal peripheral connections appear in multiple places to allow some system level flexibility. SIO, PIO0 and PIO1 can connect to all GPIO pins and are controlled by software (or software controlled state machines) so can be used to implement many functions.

Table 2. General Purpose Input/Output (GPIO) Bank 0 Functions

	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

	Function								
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

Table 3. GPIO bank 0 function descriptions

Function Name	Description
SPIx	Connect one of the internal PL022 SPI peripherals to GPIO
UARTx	Connect one of the internal PL011 UART peripherals to GPIO
I2Cx	Connect one of the internal DW I2C peripherals to GPIO
PWMx A/B	Connect a PWM slice to GPIO. There are eight PWM slices, each with two output channels (A/B). The B pin can also be used as an input, for frequency and duty cycle measurement.
SIO	Software control of GPIO, from the single-cycle IO (SIO) block. The SIO function (F5) must be selected for the processors to *drive* a GPIO, but the input is always connected, so software can check the state of GPIOs at any time.
PIOx	Connect one of the programmable IO blocks (PIO) to GPIO. PIO can implement a *wide* variety of interfaces, and has its own internal pin mapping hardware, allowing flexible placement of digital interfaces on bank 0 GPIOs. The PIO function (F6, F7) must be selected for PIO to *drive* a GPIO, but the input is always connected, so the PIOs can always see the state of all pins.
CLOCK GPINx	General purpose clock inputs. Can be routed to a number of internal clock domains on RP2040, e.g. to provide a 1Hz clock for the RTC, or can be connected to an internal frequency counter.
CLOCK GPOUTx	General purpose clock outputs. Can drive a number of internal clocks (including PLL outputs) onto GPIOs, with optional integer divide.
USB OVCUR DET/VBUS DET/VBUS EN	USB power control signals to/from the internal USB controller

Chapter 2. System Description

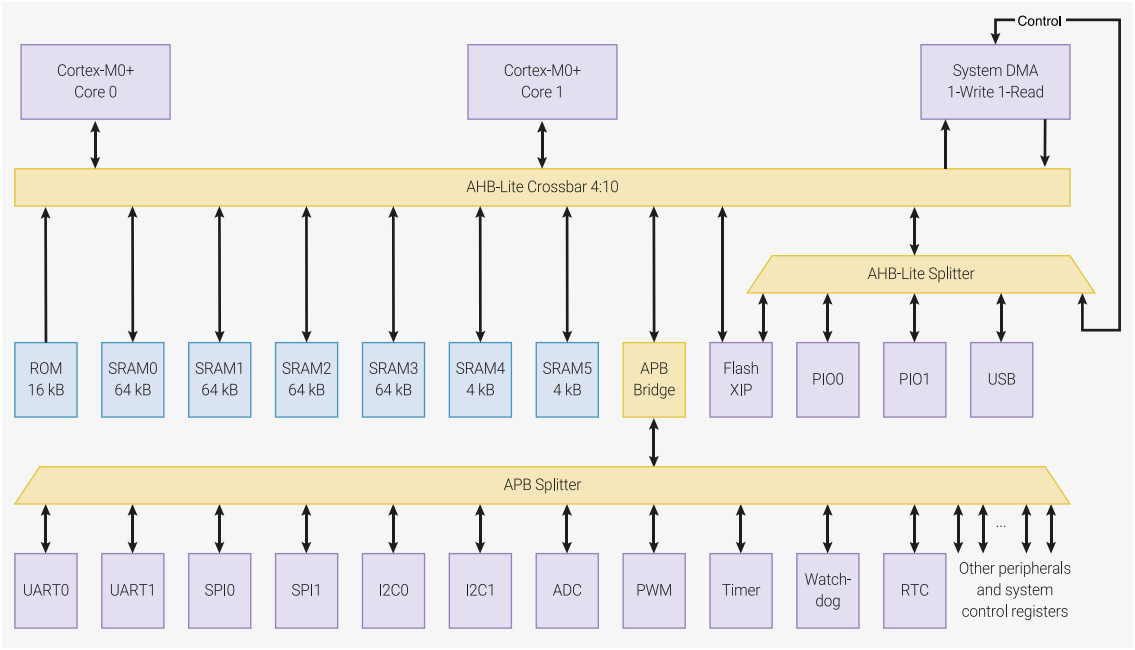
This chapter describes the RP2040 key system features including processor, memory, how blocks are connected, clocks, resets, power, and IO. Refer to Figure 2 for an overview diagram.

2.1. Bus Fabric

The RP2040 bus fabric routes addresses and data across the chip.

Figure 4 shows the high-level structure of the bus fabric. The main AHB-Lite crossbar routes addresses and data between its 4 upstream ports and 10 downstream ports: up to four bus transfers can take place each cycle. All data paths are 32 bits wide. Memory devices have dedicated ports on the main crossbar, to satisfy their high bandwidth requirements. High-bandwidth AHB-Lite peripherals have a shared port on the crossbar, and an APB bridge provides bus access to system control registers and lower-bandwidth peripherals.

Figure 4. RP2040 bus fabric overview.



The bus fabric connects 4 AHB-Lite masters, i.e. devices which generate addresses:

- Processor core 0
- Processor core 1
- DMA controller Read port
- DMA controller Write port

These are routed through to 10 downstream ports on the main crossbar:

- ROM
- Flash XIP
- SRAM 0 to 5 (one port each)
- Fast AHB-Lite peripherals: PIO0, PIO1, USB, DMA control registers, XIP aux (one shared port)
- Bridge to all APB peripherals, and system control registers

The four bus masters can access any four *different* crossbar ports simultaneously, the bus fabric does not add wait

states to any AHB-Lite slave access. So at a system clock of 125MHz the maximum sustained bus bandwidth is 2.0GBps. The system address map has been arranged to make this parallel bandwidth available to as many software use cases as possible — for example, the striped SRAM alias (Section 2.6.2) scatters main memory accesses across four crossbar ports (SRAM0…3), so that more memory accesses can proceed in parallel.

2.1.1. AHB-Lite Crossbar

At the centre of the RP2040 bus fabric is a 4:10 fully-connected crossbar. Its 4 upstream ports are connected to the 4 system bus masters, and the 10 downstream ports connect to the highest-bandwidth AHB-Lite slaves (namely the memory interfaces) and to lower layers of the fabric. Figure 5 shows the structure of a 2:3 AHB-Lite crossbar, arranged identically to the 4:10 crossbar on RP2040, but easier to show in the diagram.

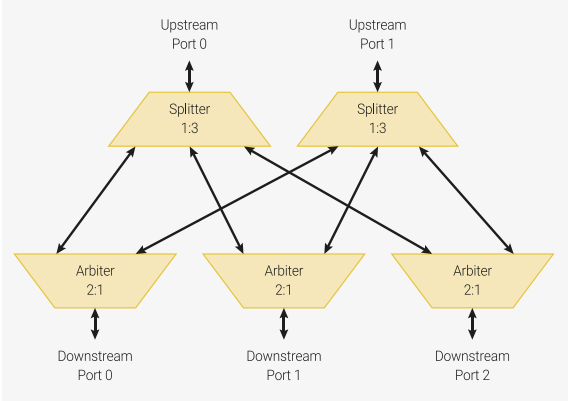


Figure 5. A 2:3 AHB-Lite crossbar. Each upstream port connects to a splitter, which routes bus requests toward one of the 3 downstream ports, and routes responses back. Each downstream port connects to an arbiter, which safely manages concurrent access to the port.

The crossbar is built from two components:

- Splitters
  - Perform coarse address decode
  - Route requests (addresses, write data) to the downstream port indicated by the initial address decode
  - Route responses (read data, bus errors) from the correct arbiter back to the upstream port
- Arbiters
  - Manage concurrent requests to a downstream port
  - Route responses (read data, bus errors) to the correct splitter
  - Implement bus priority rules

The main crossbar on RP2040 consists of 4 1:10 splitters and 10 4:1 arbiters, with a mesh of 40 AHB-Lite bus channels between them. Note that, as AHB-Lite is a pipelined bus, the splitter may be routing back a response to an earlier request from downstream port A, whilst a new request to downstream port B is already in progress. This does not incur any cycle penalty.

2.1.1.1. Bus Priority

The arbiters in the main AHB-Lite crossbar implement a two-level bus priority scheme. Priority levels are configured per-master, using the BUS_PRIORITY register in the BUSCTRL register block.

When there are multiple simultaneous accesses to same arbiter, any requests from high-priority masters (priority level 1) will be considered before any requests from low-priority masters (priority 0). If multiple masters of the same priority level attempt to access the same slave simultaneously, a round-robin tie break is applied, i.e. the arbiter grants access to each master in turn.

NOTE

Priority arbitration only applies to multiple masters attempting to access the **same** slave on the same cycle. Accesses to different slaves, e.g. different SRAM banks, can proceed simultaneously.

When accessing a slave with zero wait states, such as SRAM (i.e. can be accessed once per system clock cycle), high-priority masters will never observe any slowdown or other timing effects caused by accesses from low-priority masters. This allows *guaranteed* latency and throughput for hard real time use cases; it does however mean a low-priority master may get stalled until there is a free cycle.

2.1.1.2. Bus Performance Counters

The performance counters automatically count accesses to the main AHB-Lite crossbar arbiters. This can assist in diagnosing performance issues, in high-traffic use cases.

There are four performance counters. Each is a 24-bit saturating counter. Counter values can be read from `BUSCTRL_PERFCTRx`, and cleared by writing any value to `BUSCTRL_PERFCTRx`. Each counter can count one of the 20 available events at a time, as selected by `BUSCTRL_PERFSELx`. The available bus events are:

PERFSELx	Event	Description
0	APB access, contested	Completion of an access to the APB arbiter (which is upstream of all APB peripherals), which was previously delayed due to an access by another master.
1	APB access	Completion of an access to the APB arbiter
2	FASTPERI access, contested	Completion of an access to the FASTPERI arbiter (which is upstream of PIOs, DMA config port, USB, XIP aux FIFO port), which was previously delayed due to an access by another master.
3	FASTPERI access	Completion of an access to the FASTPERI arbiter
4	SRAM5 access, contested	Completion of an access to the SRAM5 arbiter, which was previously delayed due to an access by another master.
5	SRAM5 access	Completion of an access to the SRAM5 arbiter
6	SRAM4 access, contested	Completion of an access to the SRAM4 arbiter, which was previously delayed due to an access by another master.
7	SRAM4 access	Completion of an access to the SRAM4 arbiter
8	SRAM3 access, contested	Completion of an access to the SRAM3 arbiter, which was previously delayed due to an access by another master.
9	SRAM3 access	Completion of an access to the SRAM3 arbiter
10	SRAM2 access, contested	Completion of an access to the SRAM2 arbiter, which was previously delayed due to an access by another master.
11	SRAM2 access	Completion of an access to the SRAM2 arbiter
12	SRAM1 access, contested	Completion of an access to the SRAM1 arbiter, which was previously delayed due to an access by another master.
13	SRAM1 access	Completion of an access to the SRAM1 arbiter
14	SRAM0 access, contested	Completion of an access to the SRAM0 arbiter, which was previously delayed due to an access by another master.
15	SRAM0 access	Completion of an access to the SRAM0 arbiter

PERFSELx	Event	Description
16	XIP_MAIN access, contested	Completion of an access to the XIP_MAIN arbiter, which was previously delayed due to an access by another master.
17	XIP_MAIN access	Completion of an access to the XIP_MAIN arbiter
18	ROM access, contested	Completion of an access to the ROM arbiter, which was previously delayed due to an access by another master.
19	ROM access	Completion of an access to the ROM arbiter

2.1.2. Atomic Register Access

Each peripheral register block is allocated 4kB of address space, with registers accessed using one of 4 methods, selected by address decode.

- `Addr + 0x0000` : normal read write access
- `Addr + 0x1000` : atomic XOR on write
- `Addr + 0x2000` : atomic bitmask set on write
- `Addr + 0x3000` : atomic bitmask clear on write

This allows individual fields of a control register to be modified without performing a read-modify-write sequence in software: instead the changes are posted to the peripheral, and performed in-situ. Without this capability, it is difficult to safely access IO registers when an interrupt service routine is concurrent with code running in the foreground, or when the two processors are running code in parallel.

The four atomic access aliases occupy a total of 16kB. Most peripherals on RP2040 provide this functionality natively, and atomic writes have the same timing as normal read/write access. Some peripherals (I2C, UART, SPI and SSI) instead have this functionality added using a bus interposer, which translates upstream atomic writes into downstream read-modify-write sequences, at the boundary of the peripheral. This extends the access time by two system clock cycles.

The SIO (Section 2.3.1), a single-cycle IO block attached directly to the cores' IO ports, does **not** support atomic accesses at the bus level, although some individual registers (e.g. GPIO) have set/clear/xor aliases.

2.1.3. APB Bridge

The APB bridge interfaces the high-speed main AHB-Lite interconnect to the lower-bandwidth peripherals. Whilst the AHB-Lite fabric offers zero-wait-state access everywhere, APB accesses have a cycle penalty:

- APB bus accesses take two cycles minimum (setup phase and access phase)
- The bridge adds an additional cycle to read accesses, as the bus request and response are registered
- The bridge adds **two** additional cycles to write accesses, as the APB setup phase can not begin until the AHB-Lite write data is valid

As a result, the throughput of the APB portion of the bus fabric is somewhat lower than the AHB-Lite portion. However, there is more than sufficient bandwidth to saturate the APB serial peripherals.

2.1.4. Narrow IO Register Writes

Memory-mapped IO registers on RP2040 ignore the width of bus read/write accesses. They treat all writes as though they were 32 bits in size. This means software can not use byte or halfword writes to modify part of an IO register: any write to an address where the 30 address MSBs match the register address will affect the contents of the entire

register.

To update part of an IO register, without a read-modify-write sequence, the best solution on RP2040 is atomic set/clear/XOR (see Section 2.1.2). Note that this is more flexible than byte or halfword writes, as any combination of fields can be updated in one operation.

Upon a 8-bit or 16-bit write (such as a `strb` instruction on the Cortex-M0+), an IO register will sample the entire 32-bit write databus. The Cortex-M0+ and DMA on RP2040 will always replicate narrow data across the bus:

Pico Examples: https://github.com/raspberrypi/pico-examples/blob/master/system/narrow_io_write/narrow_io_write.c Lines 19 - 62

```
19 int main() {
20     stdio_init_all();
21
22     // We'll use WATCHDOG_SCRATCH0 as a convenient 32 bit read/write register
23     // that we can assign arbitrary values to
24     io_rw_32 *scratch32 = &watchdog_hw->scratch[0];
25     // Alias the scratch register as two halfwords at offsets +0x0 and +0x2
26     volatile uint16_t *scratch16 = (volatile uint16_t *) scratch32;
27     // Alias the scratch register as four bytes at offsets +0x0, +0x1, +0x2, +0x3:
28     volatile uint8_t *scratch8 = (volatile uint8_t *) scratch32;
29
30     // Show that we can read/write the scratch register as normal:
31     printf("Writing 32 bit value\n");
32     *scratch32 = 0xdeadbeef;
33     printf("Should be 0xdeadbeef: 0x%08x\n", *scratch32);
34
35     // We can do narrow reads just fine -- IO registers treat this as a 32 bit
36     // read, and the processor/DMA will pick out the correct byte lanes based
37     // on transfer size and address LSBs
38     printf("\nReading back 1 byte at a time\n");
39     // Little-endian!
40     printf("Should be ef be ad de: %02x ", scratch8[0]);
41     printf("%02x ", scratch8[1]);
42     printf("%02x ", scratch8[2]);
43     printf("%02x\n", scratch8[3]);
44
45     // Byte writes are replicated four times across the 32-bit bus, and IO
46     // registers usually sample the entire write bus.
47     printf("\nWriting 8 bit value 0xa5 at offset 0\n");
48     scratch8[0] = 0xa5;
49     // Read back the whole scratch register in one go
50     printf("Should be 0xa5a5a5a5: 0x%08x\n", *scratch32);
51
52     // The IO register ignores the address LSBs [1:0] as well as the transfer
53     // size, so it doesn't matter what byte offset we use
54     printf("\nWriting 8 bit value at offset 1\n");
55     scratch8[1] = 0x3c;
56     printf("Should be 0x3c3c3c3c: 0x%08x\n", *scratch32);
57
58     // Halfword writes are also replicated across the write data bus
59     printf("\nWriting 16 bit value at offset 0\n");
60     scratch16[0] = 0xf00d;
61     printf("Should be 0xf00df00d: 0x%08x\n", *scratch32);
62 }
```

2.1.5. List of Registers

The Bus Fabric registers start at a base address of `0x40030000` (defined as BUSCTRL_BASE in SDK).

Table 4. List of BUSCTRL registers

Offset	Name	Info
0x00	BUS_PRIORITY	Set the priority of each master for bus arbitration.
0x04	BUS_PRIORITY_ACK	Bus priority acknowledge
0x08	PERFCTR0	Bus fabric performance counter 0
0x0c	PERFSEL0	Bus fabric performance event select for PERFCTR0
0x10	PERFCTR1	Bus fabric performance counter 1
0x14	PERFSEL1	Bus fabric performance event select for PERFCTR1
0x18	PERFCTR2	Bus fabric performance counter 2
0x1c	PERFSEL2	Bus fabric performance event select for PERFCTR2
0x20	PERFCTR3	Bus fabric performance counter 3
0x24	PERFSEL3	Bus fabric performance event select for PERFCTR3

BUSCTRL: BUS_PRIORITY Register

Offset: 0x00

Description

Set the priority of each master for bus arbitration.

Table 5. BUS_PRIORITY Register

Bits	Description	Type	Reset
31:13	Reserved.	-	-
12	**DMA_W**: 0 - low priority, 1 - high priority	RW	0x0
11:9	Reserved.	-	-
8	**DMA_R**: 0 - low priority, 1 - high priority	RW	0x0
7:5	Reserved.	-	-
4	**PROC1**: 0 - low priority, 1 - high priority	RW	0x0
3:1	Reserved.	-	-
0	**PROC0**: 0 - low priority, 1 - high priority	RW	0x0

BUSCTRL: BUS_PRIORITY_ACK Register

Offset: 0x04

Description

Bus priority acknowledge

Table 6. BUS_PRIORITY_ACK Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Goes to 1 once all arbiters have registered the new global priority levels. Arbiters update their local priority when servicing a new nonsequential access. In normal circumstances this will happen almost immediately.	RO	0x0

BUSCTRL: PERFCTR0 Register

Offset: 0x08

Description

Bus fabric performance counter 0

Table 7. PERFCTR0 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 0 Count some event signal from the busfabric arbiters. Write any value to clear. Select an event to count using PERFSEL0	WC	0x000000

BUSCTRL: PERFSEL0 Register

Offset: 0x0c

Description

Bus fabric performance event select for PERFCTR0

Table 8. PERFSEL0 Register

Bits	Description	Type	Reset
31:5	Reserved.	-	-
4:0	Select an event for PERFCTR0. Count either contested accesses, or all accesses, on a downstream port of the main crossbar.	RW	0x1f
	Enumerated values:		
	0x00 → APB_CONTESTED		
	0x01 → APB		
	0x02 → FASTPERI_CONTESTED		
	0x03 → FASTPERI		
	0x04 → SRAM5_CONTESTED		
	0x05 → SRAM5		
	0x06 → SRAM4_CONTESTED		
	0x07 → SRAM4		
	0x08 → SRAM3_CONTESTED		
	0x09 → SRAM3		
	0x0a → SRAM2_CONTESTED		
	0x0b → SRAM2		
	0x0c → SRAM1_CONTESTED		
	0x0d → SRAM1		
	0x0e → SRAM0_CONTESTED		
	0x0f → SRAM0		
	0x10 → XIP_MAIN_CONTESTED		
	0x11 → XIP_MAIN		
	0x12 → ROM_CONTESTED		
	0x13 → ROM		

BUSCTRL: PERFCTR1 Register

Offset: 0x10

Description

Bus fabric performance counter 1

Table 9. PERFCTR1 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 1 Count some event signal from the busfabric arbiters. Write any value to clear. Select an event to count using PERFSEL1	WC	0x000000

BUSCTRL: PERFSEL1 Register

Offset: 0x14

Description

Bus fabric performance event select for PERFCTR1

Table 10. PERFSEL1 Register

Bits	Description	Type	Reset
31:5	Reserved.	-	-
4:0	Select an event for PERFCTR1. Count either contested accesses, or all accesses, on a downstream port of the main crossbar.	RW	0x1f
	Enumerated values:		
	0x00 → APB_CONTESTED		
	0x01 → APB		
	0x02 → FASTPERI_CONTESTED		
	0x03 → FASTPERI		
	0x04 → SRAM5_CONTESTED		
	0x05 → SRAM5		
	0x06 → SRAM4_CONTESTED		
	0x07 → SRAM4		
	0x08 → SRAM3_CONTESTED		
	0x09 → SRAM3		
	0x0a → SRAM2_CONTESTED		
	0x0b → SRAM2		
	0x0c → SRAM1_CONTESTED		
	0x0d → SRAM1		
	0x0e → SRAM0_CONTESTED		
	0x0f → SRAM0		
	0x10 → XIP_MAIN_CONTESTED		
	0x11 → XIP_MAIN		
	0x12 → ROM_CONTESTED		
	0x13 → ROM		

BUSCTRL: PERFCTR2 Register

Offset: 0x18

Description

Bus fabric performance counter 2

Table 11. PERFCTR2 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 2 Count some event signal from the busfabric arbiters. Write any value to clear. Select an event to count using PERFSEL2	WC	0x000000

BUSCTRL: PERFSEL2 Register

Offset: 0x1c

Description

Bus fabric performance event select for PERFCTR2

Table 12. PERFSEL2 Register

Bits	Description	Type	Reset
31:5	Reserved.	-	-
4:0	Select an event for PERFCTR2. Count either contested accesses, or all accesses, on a downstream port of the main crossbar.	RW	0x1f
	Enumerated values:		
	0x00 → APB_CONTESTED		
	0x01 → APB		
	0x02 → FASTPERI_CONTESTED		
	0x03 → FASTPERI		
	0x04 → SRAM5_CONTESTED		
	0x05 → SRAM5		
	0x06 → SRAM4_CONTESTED		
	0x07 → SRAM4		
	0x08 → SRAM3_CONTESTED		
	0x09 → SRAM3		
	0x0a → SRAM2_CONTESTED		
	0x0b → SRAM2		
	0x0c → SRAM1_CONTESTED		
	0x0d → SRAM1		
	0x0e → SRAM0_CONTESTED		
	0x0f → SRAM0		
	0x10 → XIP_MAIN_CONTESTED		
	0x11 → XIP_MAIN		
	0x12 → ROM_CONTESTED		

Bits	Description	Type	Reset
	0x13 → ROM		

BUSCTRL: PERFCTR3 Register

Offset: 0x20

Description

Bus fabric performance counter 3

Table 13. PERFCTR3 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 3 Count some event signal from the busfabric arbiters. Write any value to clear. Select an event to count using PERFSEL3	WC	0x000000

BUSCTRL: PERFSEL3 Register

Offset: 0x24

Description

Bus fabric performance event select for PERFCTR3

Table 14. PERFSEL3 Register

Bits	Description	Type	Reset
31:5	Reserved.	-	-
4:0	Select an event for PERFCTR3. Count either contested accesses, or all accesses, on a downstream port of the main crossbar.	RW	0x1f
	Enumerated values:		
	0x00 → APB_CONTESTED		
	0x01 → APB		
	0x02 → FASTPERI_CONTESTED		
	0x03 → FASTPERI		
	0x04 → SRAM5_CONTESTED		
	0x05 → SRAM5		
	0x06 → SRAM4_CONTESTED		
	0x07 → SRAM4		
	0x08 → SRAM3_CONTESTED		
	0x09 → SRAM3		
	0x0a → SRAM2_CONTESTED		
	0x0b → SRAM2		
	0x0c → SRAM1_CONTESTED		
	0x0d → SRAM1		
	0x0e → SRAM0_CONTESTED		
	0x0f → SRAM0		
	0x10 → XIP_MAIN_CONTESTED		

Bits	Description	Type	Reset
	0x11 → XIP_MAIN		
	0x12 → ROM_CONTESTED		
	0x13 → ROM		

2.2. Address Map

The address map for the device is split in to sections as shown in Table 15. Details are shown in the following sections. Unmapped address ranges raise a bus error when accessed.

2.2.1. Summary

Table 15. Address Map Summary

ROM	0x00000000
XIP	0x10000000
SRAM	0x20000000
APB Peripherals	0x40000000
AHB-Lite Peripherals	0x50000000
IOPORT Registers	0xd0000000
Cortex-M0+ internal registers	0xe0000000

2.2.2. Detail

ROM:

ROM_BASE	0x00000000

XIP:

XIP_BASE	0x10000000
XIP_NOALLOC_BASE	0x11000000
XIP_NOCACHE_BASE	0x12000000
XIP_NOCACHE_NOALLOC_BASE	0x13000000
XIP_CTRL_BASE	0x14000000
XIP_SRAM_BASE	0x15000000
XIP_SRAM_END	0x15004000
XIP_SSI_BASE	0x18000000

SRAM. SRAM0-3 striped:

SRAM_BASE	0x20000000
SRAM_STRIPED_BASE	0x20000000

SRAM_STRIPED_END	0x20040000

SRAM 4-5 are always non-striped:

SRAM4_BASE	0x20040000
SRAM5_BASE	0x20041000
SRAM_END	0x20042000

Non-striped aliases of SRAM0-3:

SRAM0_BASE	0x21000000
SRAM1_BASE	0x21010000
SRAM2_BASE	0x21020000
SRAM3_BASE	0x21030000

APB Peripherals:

SYSINFO_BASE	0x40000000
SYSCFG_BASE	0x40004000
CLOCKS_BASE	0x40008000
RESETS_BASE	0x4000c000
PSM_BASE	0x40010000
IO_BANK0_BASE	0x40014000
IO_QSPI_BASE	0x40018000
PADS_BANK0_BASE	0x4001c000
PADS_QSPI_BASE	0x40020000
XOSC_BASE	0x40024000
PLL_SYS_BASE	0x40028000
PLL_USB_BASE	0x4002c000
BUSCTRL_BASE	0x40030000
UART0_BASE	0x40034000
UART1_BASE	0x40038000
SPI0_BASE	0x4003c000
SPI1_BASE	0x40040000
I2C0_BASE	0x40044000
I2C1_BASE	0x40048000
ADC_BASE	0x4004c000
PWM_BASE	0x40050000
TIMER_BASE	0x40054000
WATCHDOG_BASE	0x40058000
RTC_BASE	0x4005c000