

Memory Mapped IO used for GPIO

Lecture 2

GPIO for RP2040

- Memory Mapped I/O
 - GPIO Peripheral
- Embedded Rust Stack
- embassy-rs

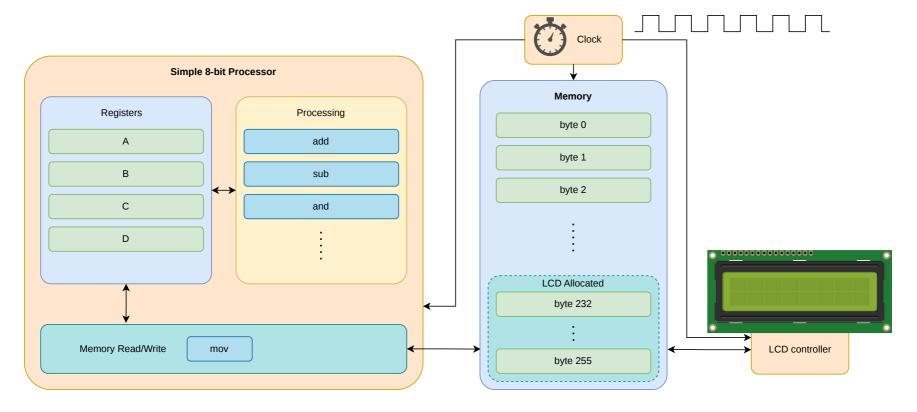


MMIO

Memory Mapped Input Output



a simple 8 bit processor with a text display

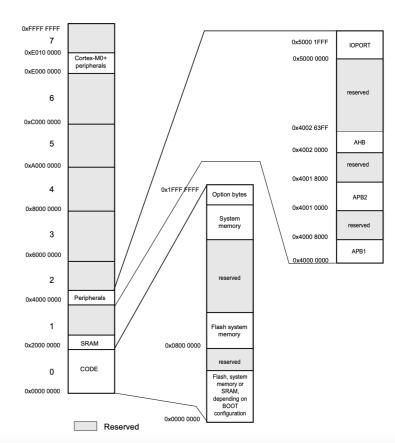


STM32L0x2

A real MCU

| Cortex-M0+ Peripherals | MCU's <i>settings</i> and internal peripherals, available at the same address on all M0+ |
|---------------------------|--|
| Peripherals | GPIO, USART, SPI, I2C, USB, etc |
| Flash | The storage space |
| SRAM | RAM memory |
| @0x0000_0000 | Alias for SRAM or Flash |





System Control Registers

@0xe000 0000

Compute the actual address

• 0xe000_0000 + Offset

Examples:

- SYST_CSR: 0xe000_e010 (0xe000_0000 + 0xe010)
- CPUID: 0xe000_ed00 (0xe000_0000 + 0xed00)

```
const SYS_CTRL: usize = 0xe000_0000;
const CPUID: usize = 0xed00;

let cpuid_reg = (SYS_CTRL + CPUID) as *const u32;
let cpuid_value = unsafe { *cpuid_reg };
```

△ Compilers optimize code and processors use cache!

| Offset | Name | Info |
|--------|------------|--|
| 0xe010 | SYST_CSR | SysTick Control and Status Register |
| 0xe014 | SYST_RVR | SysTick Reload Value Register |
| 0xe018 | SYST_CVR | SysTick Current Value Register |
| 0xe01c | SYST_CALIB | SysTick Calibration Value Register |
| 0xe100 | NVIC_ISER | Interrupt Set-Enable Register |
| 0xe180 | NVIC_ICER | Interrupt Clear-Enable Register |
| 0xe200 | NVIC_ISPR | Interrupt Set-Pending Register |
| 0xe280 | NVIC_ICPR | Interrupt Clear-Pending Register |
| 0xe400 | NVIC_IPR0 | Interrupt Priority Register 0 |
| 0xe404 | NVIC_IPR1 | Interrupt Priority Register 1 |
| 0xe408 | NVIC_IPR2 | Interrupt Priority Register 2 |
| 0xe40c | NVIC_IPR3 | Interrupt Priority Register 3 |
| 0xe410 | NVIC_IPR4 | Interrupt Priority Register 4 |
| 0xe414 | NVIC_IPR5 | Interrupt Priority Register 5 |
| 0xe418 | NVIC_IPR6 | Interrupt Priority Register 6 |
| 0xe41c | NVIC_IPR7 | Interrupt Priority Register 7 |
| 0xed00 | CPUID | CPUID Base Register |
| 0xed04 | ICSR | Interrupt Control and State Register |
| 0xed08 | VTOR | Vector Table Offset Register |
| 0xed0c | AIRCR | Application Interrupt and Reset Control Register |
| 0xed10 | SCR | System Control Register |
| 0xed14 | CCR | Configuration and Control Register |

Compiler Optimization



compilers optimize code

Write bytes to the UART (serial port) data register

```
const UART_TX: *const u8 = 0x400_3400;
for b in b"Hello, World".iter() {
    unsafe { UART_TX.write(*b); }
}
```

- 1. The compiler does not know that UART_TX is a register and uses it as a memory address.
- 2. Writing several values to the same memory address will result in having the last value stored at that address.
- 3. The compiler optimizes the code write the value

```
const UART_TX: *const u8 = 0x400_3400;
unsafe { UART_TX.write(b'd'); }
```

.

No Compiler Optimization

CPUID: **0xe000_ed00** (*0xe000_0000 + 0xed00*)

```
use core::ptr::read_volatile;

const SYS_CTRL: usize = 0xe000_0000;

const CPUID: usize = 0xed00;

let cpuid_reg = (SYS_CTRL + CPUID) as *const u32;

unsafe {
    read_volatile(cpuid_reg) // avoid cache
}
```

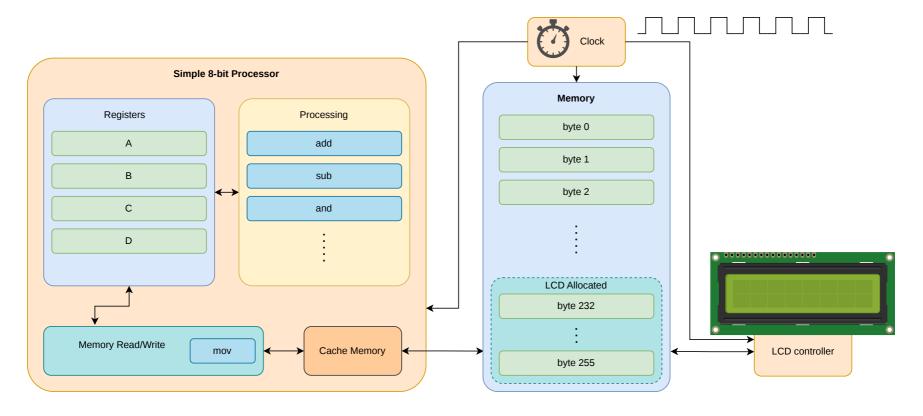
| <pre>read_volatile ,</pre> | no compiler |
|----------------------------|---|
| write_volatile | optimization |
| read, write, *p | use compiler optimization |

| Offset | Name | Info |
|--------|------------|--|
| 0xe010 | SYST_CSR | SysTick Control and Status Register |
| 0xe014 | SYST_RVR | SysTick Reload Value Register |
| 0xe018 | SYST_CVR | SysTick Current Value Register |
| 0xe01c | SYST_CALIB | SysTick Calibration Value Register |
| 0xe100 | NVIC_ISER | Interrupt Set-Enable Register |
| 0xe180 | NVIC_ICER | Interrupt Clear-Enable Register |
| 0xe200 | NVIC_ISPR | Interrupt Set-Pending Register |
| 0xe280 | NVIC_ICPR | Interrupt Clear-Pending Register |
| 0xe400 | NVIC_IPR0 | Interrupt Priority Register 0 |
| 0xe404 | NVIC_IPR1 | Interrupt Priority Register 1 |
| 0xe408 | NVIC_IPR2 | Interrupt Priority Register 2 |
| 0xe40c | NVIC_IPR3 | Interrupt Priority Register 3 |
| 0xe410 | NVIC_IPR4 | Interrupt Priority Register 4 |
| 0xe414 | NVIC_IPR5 | Interrupt Priority Register 5 |
| 0xe418 | NVIC_IPR6 | Interrupt Priority Register 6 |
| 0xe41c | NVIC_IPR7 | Interrupt Priority Register 7 |
| 0xed00 | CPUID | CPUID Base Register |
| 0xed04 | ICSR | Interrupt Control and State Register |
| 0xed08 | VTOR | Vector Table Offset Register |
| 0xed0c | AIRCR | Application Interrupt and Reset Control Register |
| 0xed10 | SCR | System Control Register |
| 0xed14 | CCR | Configuration and Control Register |





with cache



No Cache or Flush Cache



- Cache types:
 - write-through data is written to the cache and to the main memory (bus)
 - write-back data is written to the cache and later to the main memory (bus)
- few Cortex-M MCUs have cache
- the Memory Mapped I/O region is set as nocache
- for chips that use cache
 - nocache regions have to be set manually (if MCU knows)
 - or, the cache has to be flushed before a volatile_read and after a volatile_write
 - beware DMA controllers that can't see the cache contents

Read the CPUID

About the MCU

```
use core::ptr::read volatile;
     const SYS CTRL: usize = 0xe000 0000;
     const CPUID: usize = 0xed00;
     let cpuid req = (SYS CTRL + CPUID) as *const u32;
     let cpuid value = unsafe {
         read volatile(cpuid req)
 8
 9
     };
10
     // shift right 24 bits and keep only the last 8 bits
11
     let variant = (cpuid value >> 24) & 0b1111 1111;
12
13
14
     // shift right 16 bits and keep only the last 4 bits
15
     let architecture = (cpuid value >> 16) & 0b1111;
16
     // shift right 4 bits and keep only the last 12 bits
17
     let part no = (cpuid value >> 4) & 0b11 1111 1111;
18
19
     // shift right 0 bits and keep only the last 4 bits
20
     let revision = (cpuid value >> 0) & 0b1111;
```

CPUID Register

Offset: 0xed04

| Bits | Name | Description | Туре | Reset |
|-------|--------------|---|------|-------|
| 31:24 | IMPLEMENTER | Implementor code: 0x41 = ARM | RO | 0x41 |
| 23:20 | VARIANT | Major revision number n in the rnpm revision status: 0x0 = Revision 0. | RO | 0x0 |
| 19:16 | ARCHITECTURE | Constant that defines the architecture of the processor: 0xC = ARMv6-M architecture. | RO | Охс |
| 15:4 | PARTNO | Number of processor within family: 0xC60 = Cortex-M0+ | RO | 0xc60 |
| 3:0 | REVISION | Minor revision number m in the rnpm revision status: 0x1 = Patch 1. | RO | 0x1 |



AIRCR

Application Interrupt and Reset Control Register

```
use core::ptr::read volatile;
     use core::ptr::write volatile;
     const SYS CTRL: usize = 0xe000 0000;
     const AIRCR: usize = 0xed0c;
     const VECTKEY: u32 = 16;
     const SYSRESETREQ: u32 = 2;
     let aircr register = (SYS CTRL + AIRCR) as *mut u32;
10
     let mut aircr value = unsafe {
12
         read_volatile(aircr_register)
13
     };
14
15
     aircr value = aircr value & ~(0x1111 << VECTKEY);
     aircr value = aircr value | (0x05fa << VECTKEY);</pre>
16
     aircr value = aircr value | (1 << SYSRESETREQ);</pre>
17
18
19
     unsafe {
         write volatile(aircr_register, aircr_value);
20
21
```

AIRCR Register

Offset: 0xed0c

| Bits | Name | Description | Туре | Reset |
|-------|-----------|--|------|--------|
| 31:16 | VECTKEY | Register key: Reads as Unknown On writes, write 0x05FA to VECTKEY, otherwise the write is ignored. | RW | 0x0000 |
| 15 | ENDIANESS | Data endianness implemented: 0 = Little-endian. | RO | 0x0 |
| 14:3 | Reserved. | - | - | - |

| Bits | Name | Description | Туре | Reset |
|------|---------------|---|------|-------|
| 2 | SYSRESETREQ | Writing 1 to this bit causes the SYSRESETREQ signal to the outer system to be asserted to request a reset. The intention is to force a large system reset of all major components except for debug. The C_HALT bit in the DHCSR is cleared as a result of the system reset requested. The debugger does not lose contact with the device. | RW | 0x0 |
| 1 | VECTCLRACTIVE | Clears all active state information for fixed and configurable exceptions. This bit: is self-clearing, can only be set by the DAP when the core is halted. When set: clears all active exception status of the processor, forces a return to Thread mode, forces an IPSR of 0. A debugger must re-initialize the stack. | RW | 0x0 |
| 0 | Reserved. | - | - | - |



Read and Write

they do stuff

- Read
 - reads the value of a register
 - might ask the peripheral to do something
- Write
 - writes the value to a register
 - might ask the peripheral to do something
 - SYSRESETREQ



AIRCR Register

Offset: 0xed0c

| Bits | Name | Description | Туре | Reset |
|-------|-----------|--|------|--------|
| 31:16 | VECTKEY | Register key: Reads as Unknown On writes, write 0x05FA to VECTKEY, otherwise the write is ignored. | RW | 0x0000 |
| 15 | ENDIANESS | Data endianness implemented: 0 = Little-endian. | RO | 0x0 |
| 14:3 | Reserved. | - | - | - |

| Bits | Name | Description | Туре | Reset |
|------|---------------|---|------|-------|
| 2 | SYSRESETREQ | Writing 1 to this bit causes the SYSRESETREQ signal to the outer system to be asserted to request a reset. The intention is to force a large system reset of all major components except for debug. The C_HALT bit in the DHCSR is cleared as a result of the system reset requested. The debugger does not lose contact with the device. | RW | 0x0 |
| 1 | VECTCLRACTIVE | Clears all active state information for fixed and configurable exceptions. This bit: is self-clearing, can only be set by the DAP when the core is halted. When set: clears all active exception status of the processor, forces a return to Thread mode, forces an IPSR of 0. A debugger must re-initialize the stack. | RW | 0x0 |
| 0 | Reserved. | - | - | - |



System View Description

```
<device schemaVersion="1.1"</pre>
       xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xs:noNamespaceSchemaLocation="CMSIS-SVD.xsd">
       <name>RP2040</name>
       <peripherals>
         <name>PPB</name>
         <baseAddress>0xe0000000/baseAddress>
          <register>
           <name>CPUID</name>
           <addressOffset>0xed00</addressOffset>
10
           <resetValue>0x410cc601</resetValue>
11
           <fields>
12
              <field>
13
                <name>IMPLEMENTER</name>
14
                <description>Implementor code: 0x41 = ARM</description>
15
                <bitRange>[31:24]</bitRange>
16
                <access>read-only</access>
              </field>
17
18
              <!-- rest of the fields of the register -->
19
           </fields>
20
         </register>
21
       </peripherals>
     </device>
```



Bitwise Ops

How to set and clear bits





set the 1 on position bit of register

```
fn set_bit(register: usize, bit: u8) -> usize {
    // assume register is 0b1000, bit is 2
    // 1 << 2 is 0b0100
4    // 0b1000 | 0b0100 is 0b1100
5    register | 1 << bit
6 }</pre>
```

Set multiple bits

```
fn set_bits(register: usize, bits: usize) -> usize {
      // assume register is 0b1000, bits is 0b0111
      // 0b1000 | 0b0111 is 0b1111
      register | bits
}
```



Clear bit

set the 0 on position bit of register

```
fn clear_bit(register: usize, bit: u8) -> usize {
    // assume register is 0b1100, bit is 2

    // 1 << 2 is 0b0100

4    // !(1 << 3) is 0b1011

5    // 0b1100 & 0b1011 is 0b1000

register & !(1 << bit)

7  }</pre>
```

Clear multiple bits

```
fn clear_bits(register: usize, bits: usize) -> usize {
    // assume register is 0b1111, bits is 0b0111

    // !bits = 0b1000

    // 0b1111 & 0b1000 is 0b1000

register & !bits

}
```



Flip bit

flip the bit on position bit of register

```
fn flip_bit(register: usize, bit: u8) -> usize {
    // assume register is 0b1000, bit is 2
    // 1 << 2 is 0b0100
    // 0b1100 ^ 0b0100 is 0b1000
    register ^ 1 << bit
}</pre>
```

Flip multiple bits

```
fn flip_bits(register: usize, bits: usize) -> usize {
    // assume register is 0b1000, bits is 0b0111
    // 0b1000 ^ 0b0111 is 0b1111
    register ^ bits
}
```



GPIO

General Purpose Input Output for RP2040

Bibliography

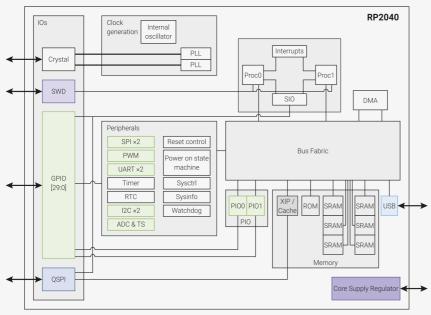
for this section

Raspberry Pi Ltd, RP2040 Datasheet

- Chapter 2 *System Description*
 - Section 2.3 *Processor subsystem*
 - Subsection 2.3.1 *SIO*
 - Subsection 2.3.1.2 GPIO Control
 - Section 2.4 Cortex-M0+ (except NVIC and MPU)
 - Section 2.19 *GPIO* (except Interrupts)



GPIO



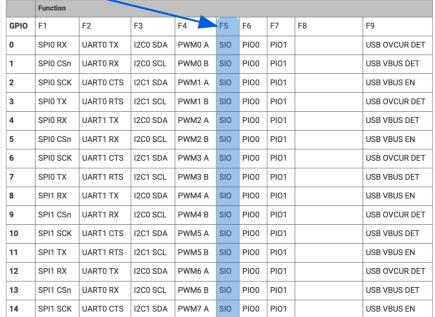
IO Bank (GPIO): Use the correct MUX function (F5)

SIO: Set the pin as Input or Output

Peripherals

Single Cycle Input/Output, is able to control SIO the GPIO pins

Multiplexes the functions of the GPIO pins



SIO Registers

The SIO registers start at a base address of 0xd0000000 (defined as SIO_BASE in SDK).

| Name | Info |
|--------------|--|
| CPUID | Processor core identifier |
| GPIO_IN | Input value for GPIO pins |
| GPIO_HI_IN | Input value for QSPI pins |
| GPIO_OUT | GPIO output value |
| GPIO_OUT_SET | GPIO output value set |
| GPIO_OUT_CLR | GPIO output value clear |
| GPIO_OUT_XOR | GPIO output value XOR |
| GPIO_OE | GPIO output enable |
| GPIO_OE_SET | GPIO output enable set |
| GPIO_OE_CLR | GPIO output enable clear |
| | CPUID GPIO_IN GPIO_HI_IN GPIO_OUT GPIO_OUT_SET GPIO_OUT_CLR GPIO_OUT_XOR GPIO_OE GPIO_OE GPIO_OE_SET |

Input

- set GPIO_OE bit x to 0
- read GPIO_IN bit x
- Ouput
 - set GPIO_OE bit x to 1
 - write GPIO_OUT bit x

GPIO_OE

| Bits | Description | Туре | Reset |
|-------|---|------|-----------|
| 31:30 | Reserved. | - | - |
| 29:0 | Set output enable (1/0 → output/input) for GPI0029. Reading back gives the last value written. If core 0 and core 1 both write to GPI0_0E simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result. | RW | 0x0000000 |

GPIO_IN

| Bits | Description | Туре | Reset |
|-------|-------------------------|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Input value for GPI0029 | RO | 0x00000000 |

GPIO_OUT

| Bits | Description | Туре | Reset |
|-------|--|------|-----------|
| 31:30 | Reserved. | - | - |
| 29:0 | Set output level (1/0 → high/low) for GPI0029. Reading back gives the last value written, NOT the input value from the pins. If core 0 and core 1 both write to GPI0_OUT simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result. | RW | 0x0000000 |

SIO Input

The SIO registers start at a base address of 0xd0000000 (defined as SIO_BASE in SDK).

| Offset | Name | Info |
|--------|--------------|---------------------------|
| 0x000 | CPUID | Processor core identifier |
| 0x004 | GPIO_IN | Input value for GPIO pins |
| 0x008 | GPIO_HI_IN | Input value for QSPI pins |
| 0x010 | GPIO_OUT | GPIO output value |
| 0x014 | GPIO_OUT_SET | GPIO output value set |
| 0x018 | GPIO_OUT_CLR | GPIO output value clear |
| 0x01c | GPIO_OUT_XOR | GPIO output value XOR |
| 0x020 | GPIO_OE | GPIO output enable |
| 0x024 | GPIO_OE_SET | GPIO output enable set |
| 0x028 | GPIO_OE_CLR | GPIO output enable clear |
| - | | |

GPIO OE

| Bits | Description | Туре | Reset |
|-------|--|------|-----------|
| 31:30 | Reserved. | - | - |
| 29:0 | Set output enable (1/0 → output/input) for GPI0029. Reading back gives the last value written. If core 0 and core 1 both write to GPI0_OE simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result. | RW | 0x0000000 |

GPIO IN

| Bits | Description | Туре | Reset |
|-------|-------------------------|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Input value for GPI0029 | RO | 0x00000000 |

```
use core::ptr::read_volatile;
     use core::ptr::write_volatile;
     const GPIO OE: *mut u32 = 0xd000 0020 as *mut u32;
     const GPIO_IN: *const u32= 0xd000_0004 as *const u32;
 6
     let value = unsafe {
         // write volatile(GPIO_OE, !(1 << pin));</pre>
         let gpio oe = read volatile(GPIO OE);
10
         // set bin `pin` of `gpio_oe` to 0 (input)
11
         gpio_oe = gpio_oe & !(1 << pin);</pre>
12
         write_volatile(GPIO_OE, gpio_oe);
13
         read_volatile(GPIO_IN) >> pin & 0b1
14 };
```

SIO Input

The SIO registers start at a base address of 0xd0000000 (defined as SIO_BASE in SDK).

| Offset | Name | Info |
|--------|--------------|---------------------------|
| 0x000 | CPUID | Processor core identifier |
| 0x004 | GPIO_IN | Input value for GPIO pins |
| 0x008 | GPIO_HI_IN | Input value for QSPI pins |
| 0x010 | GPIO_OUT | GPIO output value |
| 0x014 | GPIO_OUT_SET | GPIO output value set |
| 0x018 | GPIO_OUT_CLR | GPIO output value clear |
| 0x01c | GPIO_OUT_XOR | GPIO output value XOR |
| 0x020 | GPIO_OE | GPIO output enable |
| 0x024 | GPIO_OE_SET | GPIO output enable set |
| 0x028 | GPIO_OE_CLR | GPIO output enable clear |

GPIO_OE_SET

| Bits | Description | Туре | Reset |
|-------|--|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Perform an atomic bit-clear on GPIO_OE, i.e. 6PIO_0E &= ~wdata | wo | 0x00000000 |

GPIO IN

| Bits | Description | Туре | Reset |
|-------|-------------------------|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Input value for GPI0029 | RO | 0x00000000 |

```
use core::ptr::read_volatile;
use core::ptr::write_volatile;

const GPIO_OE_CLR: *mut u32= 0xd000_0028 as *mut u32;

const GPIO_IN: *const u32= 0xd000_0004 as *const u32;

let value = unsafe {
    // set bit `pin` of `GPIO_OE` to 0 (input)
    write_volatile(GPIO_OE_CLR, 1 << pin);
    read_volatile(GPIO_IN) >> pin & 0b1
};
```

SIO Output

The SIO registers start at a base address of 0xd0000000 (defined as SIO_BASE in SDK).

| CPUID | Processor core identifier |
|--------------|--|
| | |
| GPIO_IN | Input value for GPIO pins |
| GPIO_HI_IN | Input value for QSPI pins |
| GPIO_OUT | GPIO output value |
| GPIO_OUT_SET | GPIO output value set |
| GPIO_OUT_CLR | GPIO output value clear |
| GPIO_OUT_XOR | GPIO output value XOR |
| GPIO_OE | GPIO output enable |
| GPIO_OE_SET | GPIO output enable set |
| GPIO_OE_CLR | GPIO output enable clear |
| | GPIO_HI_IN GPIO_OUT GPIO_OUT_SET GPIO_OUT_CLR GPIO_OUT_XOR GPIO_OE GPIO_OE_SET |

GPIO_OE_CLR

| Bits | Description | Туре | Reset |
|-------|--|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Perform an atomic bit-clear on GPIO_OE, i.e. GPIO_OE &= ~wdata | WO | 0x00000000 |

GPIO OUT

| Bits | Description | Туре | Reset |
|-------|--|------|-----------|
| 31:30 | Reserved. | - | - |
| 29:0 | Set output level (1/0 → high/low) for GPI0029. Reading back gives the last value written, NOT the input value from the pins. If core 0 and core 1 both write to GPI0_OUT simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result. | RW | 0x0000000 |

```
use core::ptr::read volatile;
     use core::ptr::write volatile;
     const GPIO OE SET: *mut u32= 0xd000 0024 as *mut u32;
     const GPIO OUT: *mut u32 = 0xd000 0010 as *mut u32;
 6
     unsafe {
       // set bit `pin` of GPIO OE to 1 (output)
       write volatile(GPIO OE SET, 1 << pin);</pre>
       // write volatile(GPIO OUT, (value & 0b1) << pin);</pre>
10
11
       let gpio_out = read_volatile(GPIO_OUT);
12
       gpio_out = gpio_out | (value & 0b1) << pin;</pre>
13
       write_volatile(GPIO_OUT, gpio_out);
14
     };
```

SIO Output

efficient

The SIO registers start at a base address of 0xd0000000 (defined as SIO_BASE in SDK).

| Offset | Name | Info |
|--------|--------------|---------------------------|
| 0x000 | CPUID | Processor core identifier |
| 0x004 | GPIO_IN | Input value for GPIO pins |
| 0x008 | GPIO_HI_IN | Input value for QSPI pins |
| 0x010 | GPIO_OUT | GPIO output value |
| 0x014 | GPIO_OUT_SET | GPIO output value set |
| 0x018 | GPIO_OUT_CLR | GPIO output value clear |
| 0x01c | GPIO_OUT_XOR | GPIO output value XOR |
| 0x020 | GPIO_OE | GPIO output enable |
| 0x024 | GPIO_OE_SET | GPIO output enable set |
| 0x028 | GPIO_OE_CLR | GPIO output enable clear |

GPIO OUT SET

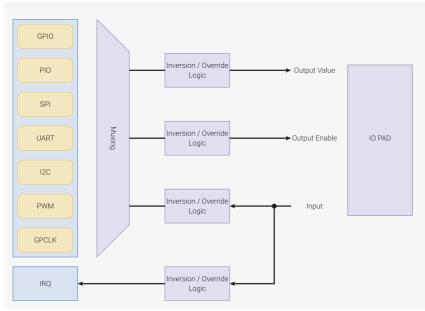
| Bits | Description | Туре | Reset |
|-------|---|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Perform an atomic bit-set on GPIO_OUT, i.e. GPIO_OUT = wdata | wo | 0x00000000 |

GPIO OUT CLR

| Bits | Description | Туре | Reset |
|-------|--|------|------------|
| 31:30 | Reserved. | - | - |
| 29:0 | Perform an atomic bit-clear on GPIO_OUT, i.e. GPIO_OUT &= ~wdata | wo | 0x00000000 |

```
use core::ptr::read volatile;
     use core::ptr::write volatile;
     const GPIO OE SET: *mut u32= 0xd000 0024 as *mut u32;
     const GPIO OUT SET:*mut u32= 0xd000 0014 as *mut u32;
     const GPIO OUT CLR:*mut u32= 0xd000 0018 as *mut u32;
     unsafe {
         write_volatile(GPIO_OE_SET, 1 << pin);</pre>
         let reg = match value {
10
11
           0 => GPIO OUT CLR,
            _ => GPIO_OUT SET
12
13
14
         write_volatile(reg, 1 << pin);</pre>
15
    };
```

IO Bank



The User Bank IO registers start at a base address of 0x40014000 (defined as IO_BANKO_BASE in SDK).

| Offset | Name | Info |
|--------|--------------|---|
| 0x000 | GPI00_STATUS | GPIO status |
| 0x004 | GPI00_CTRL | GPIO control including function select and overrides. |

■ set FUNCSEL to 5 (SIO)

GPIOx_CTRL



Offset: 0x004, 0x00c, ... 0x0ec (0x4 + 8*x)

| Bits | Name | Description | Туре | Reset |
|-------|-----------|---|------|-------|
| 31:30 | Reserved. | - | - | - |
| 29:28 | IRQOVER | $0x0 \rightarrow don't$ invert the interrupt $0x1 \rightarrow don't$ invert the interrupt $0x2 \rightarrow don't$ interrupt low $0x3 \rightarrow don't$ interrupt high | RW | 0x0 |
| 27:18 | Reserved. | - | - | - |
| 17:16 | INOVER | $0x0 \rightarrow don't$ invert the peri input $0x1 \rightarrow invert$ the peri input $0x2 \rightarrow drive$ peri input $low 0x3 \rightarrow drive$ peri input high | RW | 0x0 |
| 15:14 | Reserved. | - | - | - |
| 13:12 | OEOVER | 0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output | RW | 0x0 |
| 11:10 | Reserved. | - | - | - |
| 9:8 | OUTOVER | 0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4:0 | FUNCSEL | Function select. 31 == NULL. See GPIO function table for available functions. | RW | 0x1f |

IO Bank Input

The User Bank IO registers start at a base address of 0x40014000 (defined as IO_BANKO_BASE in SDK).

| Offset | Name | Info |
|--------|--------------|---|
| 0x000 | GPI00_STATUS | GPIO status |
| 0x004 | GPI00_CTRL | GPIO control including function select and overrides. |

```
use core::ptr::read_volatile;
     use core::ptr::write volatile;
     const GPIOX_CTRL: u32 = 0x4001_4004;
     const GPIO OE CLR: *mut u32= 0xd000 0028 as *mut u32;
     const GPIO IN: *const u32= 0xd000 0004 as *const u32;
     let gpio_ctrl = (GPIOX_CTRL + 8 * pin) as *mut u32;
 9
     let value = unsafe {
11
         write_volatile(gpio_ctrl, 5);
12
         write volatile(GPIO OE CLR, 1 << pin);</pre>
13
         read volatile(GPIO IN) >> pin & 0b1
14
    };
```

GPIOx_CTRL



Offset: 0x004, 0x00c, ... 0x0ec (0x4 + 8*x)

| Bits | Name | Description | Туре | Reset |
|-------|-----------|---|------|-------|
| 31:30 | Reserved. | | - | - |
| 29:28 | IRQOVER | $0x0 \rightarrow don't$ invert the interrupt $0x1 \rightarrow don't$ invert the interrupt $0x2 \rightarrow don't$ interrupt low $0x3 \rightarrow don't$ interrupt high | RW | 0x0 |
| 27:18 | Reserved. | - | - | - |
| 17:16 | INOVER | 0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high | RW | 0x0 |
| 15:14 | Reserved. | - | - | - |
| 13:12 | OEOVER | 0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output | RW | 0x0 |
| 11:10 | Reserved. | - | - | - |
| 9:8 | OUTOVER | 0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4:0 | FUNCSEL | Function select. 31 == NULL. See GPIO function table for available functions. | RW | 0x1f |

IO Bank Output

The User Bank IO registers start at a base address of 0x40014000 (defined as IO_BANKO_BASE in SDK).

| Offset | Name | Info |
|--------|--------------|---|
| 0x000 | GPI00_STATUS | GPIO status |
| 0x004 | GPI00_CTRL | GPIO control including function select and overrides. |

```
use core::ptr::read volatile;
     use core::ptr::write volatile;
     const GPIOX CTRL: u32 = 0x4001 4004;
     const GPIO OE SET: *mut u32= 0xd000 0024 as *mut u32;
     const GPIO OUT SET: *mut u32= 0xd000 0014 as *mut u32;
     const GPIO OUT CLR:*mut u32= 0xd000 0018 as *mut u32;
     let qpio ctrl = (GPIOX CTRL + 8 * pin) as *mut u32;
     unsafe {
11
         write volatile(qpio ctrl, 5);
12
         write volatile(GPIO OE SET, 1 << pin);</pre>
13
        let reg = match value {
14
           0 => GPIO OUT CLR,
           _ => GPIO_OUT SET
15
16
17
         write volatile(reg, 1 << pin);</pre>
18
    };
```

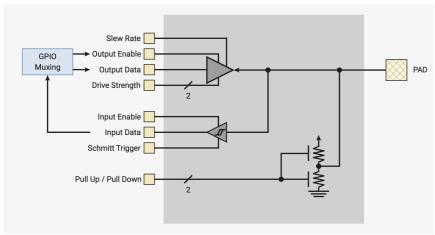
$GPIOx_CTRL$



Offset: 0x004, 0x00c, ... 0x0ec (0x4 + 8*x)

| Bits | Name | Description | Туре | Reset |
|-------|-----------|---|------|-------|
| 31:30 | Reserved. | - | - | - |
| 29:28 | IRQOVER | $0x0 \rightarrow don't$ invert the interrupt $0x1 \rightarrow invert$ the interrupt $0x2 \rightarrow drive$ interrupt low $0x3 \rightarrow drive$ interrupt high | RW | 0x0 |
| 27:18 | Reserved. | - | - | - |
| 17:16 | INOVER | $0x0 \rightarrow don't$ invert the peri input $0x1 \rightarrow invert$ the peri input $0x2 \rightarrow drive$ peri input low $0x3 \rightarrow drive$ peri input high | RW | 0x0 |
| 15:14 | Reserved. | - | - | - |
| 13:12 | OEOVER | $\begin{array}{l} 0x0 \rightarrow \text{drive output enable from peripheral signal selected} \\ by \text{funcsel} \\ 0x1 \rightarrow \text{drive output enable from inverse of peripheral} \\ \text{signal selected by funcsel} \\ 0x2 \rightarrow \text{disable output} \\ 0x3 \rightarrow \text{enable output} \end{array}$ | RW | 0x0 |
| 11:10 | Reserved. | | - | - |
| 9:8 | OUTOVER | $\begin{array}{c} 0x0 \rightarrow \text{drive output from peripheral signal selected by} \\ \text{funcsel} \\ 0x1 \rightarrow \text{drive output from inverse of peripheral signal} \\ \text{selected by funcsel} \\ 0x2 \rightarrow \text{drive output low} \\ 0x3 \rightarrow \text{drive output high} \end{array}$ | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4:0 | FUNCSEL | Function select. 31 == NULL. See GPIO function table for available functions. | RW | 0x1f |

Pad Control



The User Bank Pad Control registers start at a base address of 0x4001c000 (defined as PADS_BANKO_BASE in SDK).

| Name | Info |
|----------------|---|
| VOLTAGE_SELECT | Voltage select. Per bank control |
| GPI00 | Pad control register |
| GPI01 | Pad control register |
| GPIO2 | Pad control register |
| GPI03 | Pad control register |
| GPIO4 | Pad control register |
| GPI05 | Pad control register |
| 3 | PIO0 PIO0 PIO0 PIO0 PIO0 PIO0 PIO0 PIO0 |

GPIOx Register



Offset: 0x004, 0x008, ... 0x078 (0x4 + 4*x)

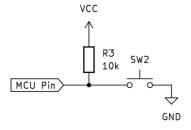
| Bits | Name | Description | Туре | Reset |
|------|-----------|--|------|-------|
| 31:8 | Reserved. | - | - | - |
| 7 | OD | Output disable. Has priority over output enable from peripherals | RW | 0x0 |
| 6 | IE | Input enable | RW | 0x1 |

| Bits | Name | Description | Туре | Reset |
|------|----------|---------------------------------------|------|-------|
| 5:4 | DRIVE | Drive strength. | RW | 0x1 |
| | | 0x0 → 2mA | | |
| | | 0x1 → 4mA | | |
| | | 0x2 → 8mA | | |
| | | 0x3 → 12mA | | |
| 3 | PUE | Pull up enable | RW | 0x0 |
| 2 | PDE | Pull down enable | RW | 0x1 |
| 1 | SCHMITT | Enable schmitt trigger | RW | 0x1 |
| 0 | SLEWFAST | Slew rate control. 1 = Fast, 0 = Slow | RW | 0x0 |

Input

read the value from pin x

- set the FUNCSEL field of GPIOx CTRL to 5
- set the GPIO_OE_CLR bit x to 1
- read the GPIO IN bit x
- adjust the GPIOx fields to set the pull up/down resistor

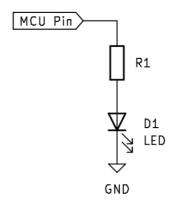


Output



write a value to pin x

- set the FUNCSEL field of GPIOx CTRL to 5
- set the GPIO_OE_SET bit x to 1
- if the value
 - is 0, set the GPIO OUT CLR bit x to 1
 - is 1, set the GPIO OUT SET bit x to 1
- adjust the GPIOx fields to set the output current







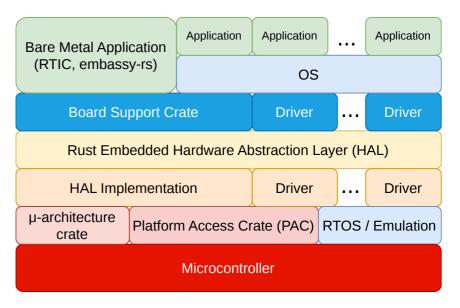
Rust Embedded HAL

The Rust API for embedded systems





| Framework | Tasks, Memory Management, Network etc. embassy-rs, rtic |
|-----------------------|--|
| BSC | Board Support Crate embassy-rp, rp-pico |
| HAL Implementation | Uses the PAC and exports a standard HAL towards the upper levels embassy-rp |
| PAC | Accesses registers, usually created automatically from SVD files - rp2040_pac , rp-pac |







A set of standard traits

All devices should implement these traits for GPIO.

```
pub enum PinState {
Low,
High,
}
```

Input

```
pub trait InputPin: ErrorType {
    // Required methods
    fn is_high(&mut self) -> Result<bool, Self::Error>;
    fn is_low(&mut self) -> Result<bool, Self::Error>;
}
```

Output

```
pub trait OutputPin: ErrorType {
    // Required methods
    fn set_low(&mut self) -> Result<(), Self::Error>;
    fn set_high(&mut self) -> Result<(), Self::Error>;

    // Provided method
    fn set_state(&mut self, state: PinState) -> Result<(), Sel:
}</pre>
```

Bare metal

道图》

This is how a Rust application would look like

```
#![no_std]
     #![no_main]
     use cortex_m_rt::entry;
     #[entry]
     fn main() -> ! {
      // your code goes here
 9
10
         loop { }
11
12
13
     #[panic_handler]
     pub fn panic(_info: &PanicInfo) -> ! {
15
         loop { }
16
```

Rules

- 1. never exit the main function
- 2. add a panic handler that does not exit

Bare metal without PAC & HAL

This is how a Rust application would look like

```
#![no std]
     #![no_main]
     use core::ptr::{read volatile, write volatile};
     use cortex m rt::entry;
 6
     const GPIOX CTRL: u32 = 0 \times 4001 + 4004;
     const GPIO OE SET: *mut u32= 0xd000 0024 as *mut u32;
     const GPIO OUT SET:*mut u32= 0xd000 0014 as *mut u32;
     const GPIO OUT CLR:*mut u32= 0xd000 0018 as *mut u32;
11
     #[panic handler]
12
     pub fn panic( info: &PanicInfo) -> ! {
13
14
         loop { }
15
```

```
#[entry]
     fn main() -> ! {
         let gpio ctrl = GPIOX CTRL + 8 * pin as *mut u32;
20
         unsafe {
22
             write volatile(qpio ctrl, 5);
23
             write volatile(GPIO OE SET, 1 << pin);</pre>
             let req = match value {
24
             0 => GPIO OUT CLR,
              => GPIO OUT SET
27
28
             write volatile(req, 1 << pin);</pre>
29
30
31
         loop { }
32
```



embassy-rs

Embedded Asynchronous

embassy-rs

- framework
- uses the rust-embedded-hal
- Features
 - Real-time
 - Low power
 - Networking
 - Bluetooth
 - USB
 - Bootloader and DFU





```
#![no_std]
     #![no_main]
     use embassy executor::Spawner;
     use embassy_rp::gpio;
     use gpio::{Input, Pull};
     #[embassy executor::main]
     async fn main(_spawner: Spawner) {
10
         let p = embassy_rp::init(Default::default());
         let pin = Input::new(p.PIN_3, Pull:Up);
11
12
         if pin.is_high() {
13
14
15
         } else {
16
17
18
```

The main function is called by the embassy-rs framework, so it can exit.





```
#![no_std]
#![no_main]

use embassy_executor::Spawner;
use embassy_rp::gpio;
use gpio::{Level, Output};

#[embassy_executor::main]
async fn main(_spawner: Spawner) {
    let p = embassy_rp::init(Default::default());
    let mut pin = Output::new(p.PIN_2, Level::Low);

pin.set_high();
}
```

The main function is called by the embassy-rs framework, so it can exit.

Conclusion

B

we talked about

- Memory Mapped IO
- RP2040 GPIO
 - Single Cycle IO
 - IO Bank
 - Pad
- The Rust embedded standard stack
- Bare metal Rust
- The embassy-rs framework