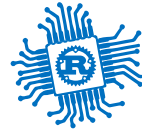


# Memory Mapped IO used for GPIO

Lecture 2



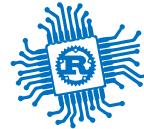
# Accessing Peripherals

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



# MMIO

Memory Mapped Input Output



# Bibliography

for this section

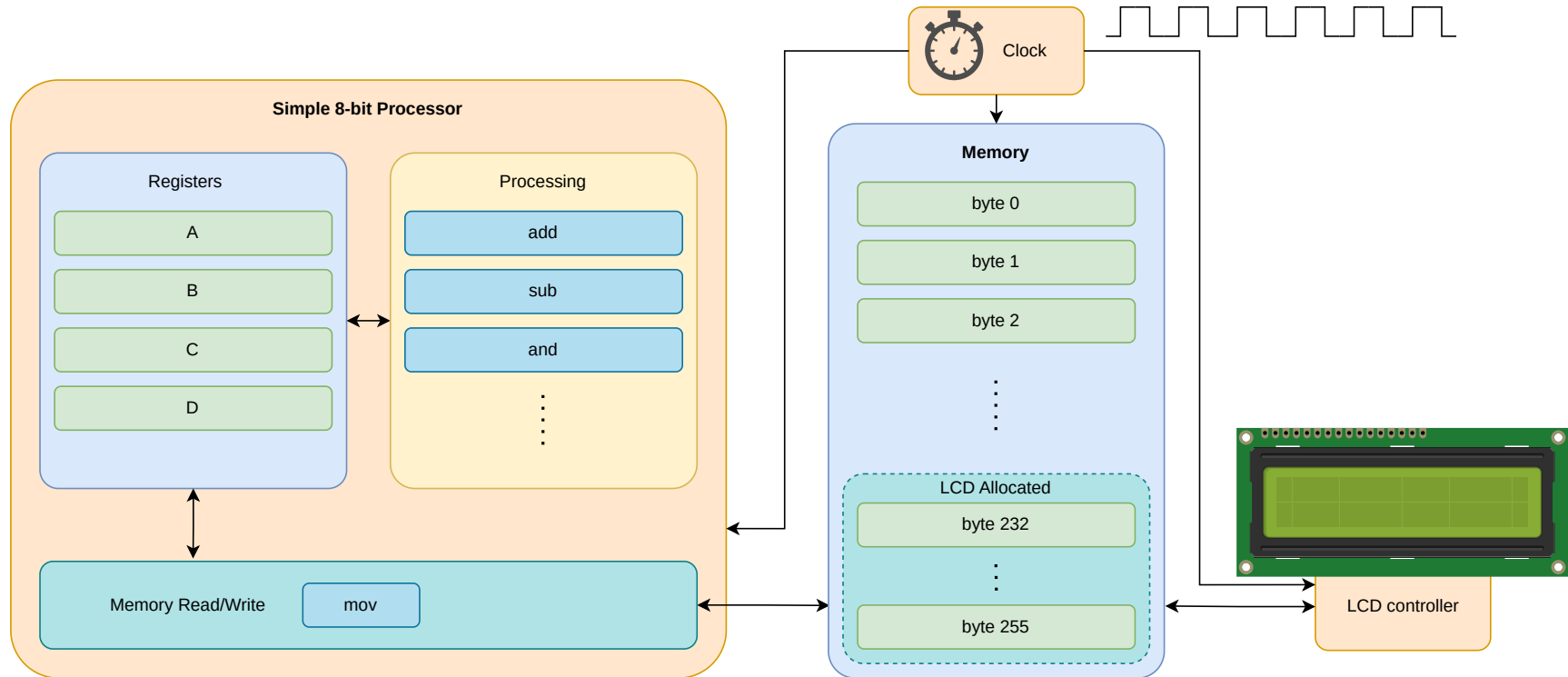
**Joseph Yiu**, *The Definitive Guide to ARM® Cortex®-M23 and Cortex-M33 Processors*

- Chapter 6 - *Memory system*
  - Section 6.1 - *Overview of the memory system*
  - Section 6.2 - *Memory map*
  - Section 6.11 - *Memory systems in microcontrollers*



# 8 bit processor

a simple 8 bit processor with a text display

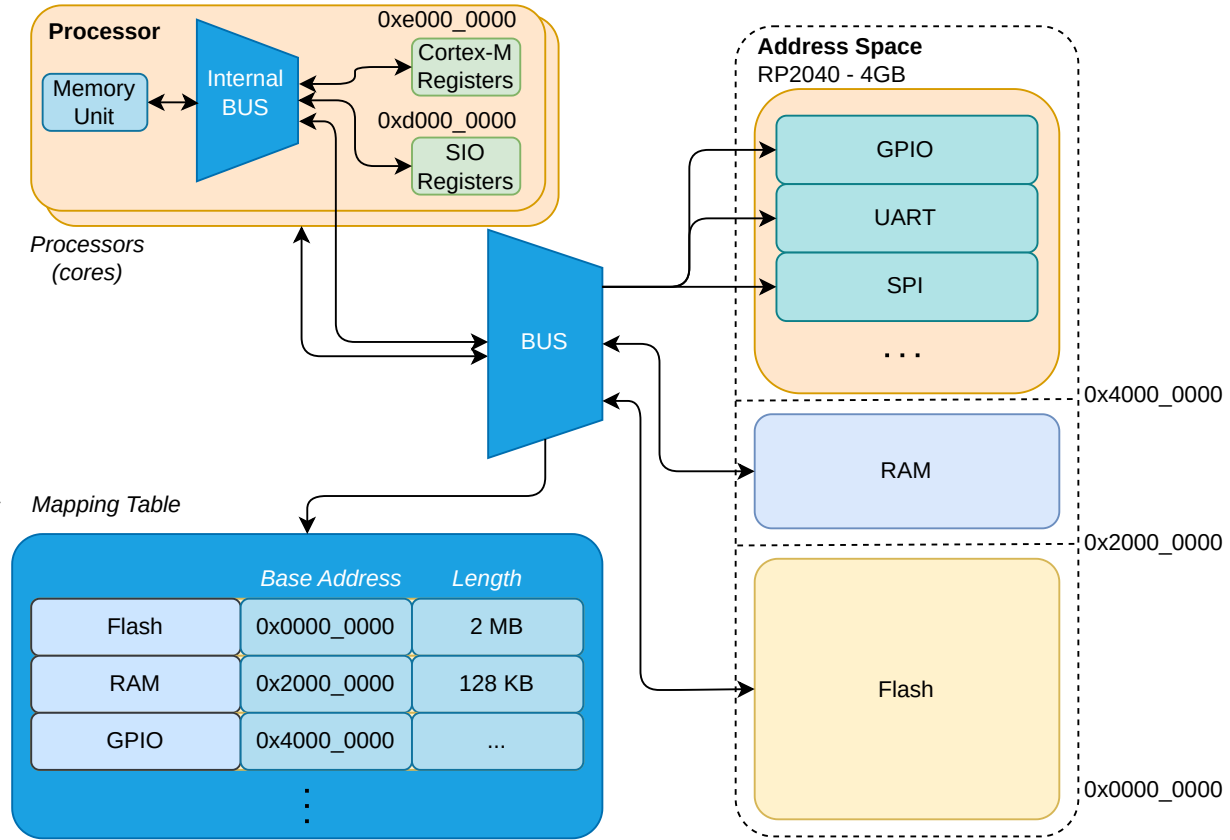




# The Bus

example for RP2

1. **Memory Controller** asks for data transfer
2. **Internal Bus Routes** the request
  - to the *External Bus* or
  - to the *Internal Peripherals*
3. **External Bus Routes** the request based on the *Address Mapping Table*
  1. to **RAM**
  2. to **Flash**
  3. to an **External Peripheral**

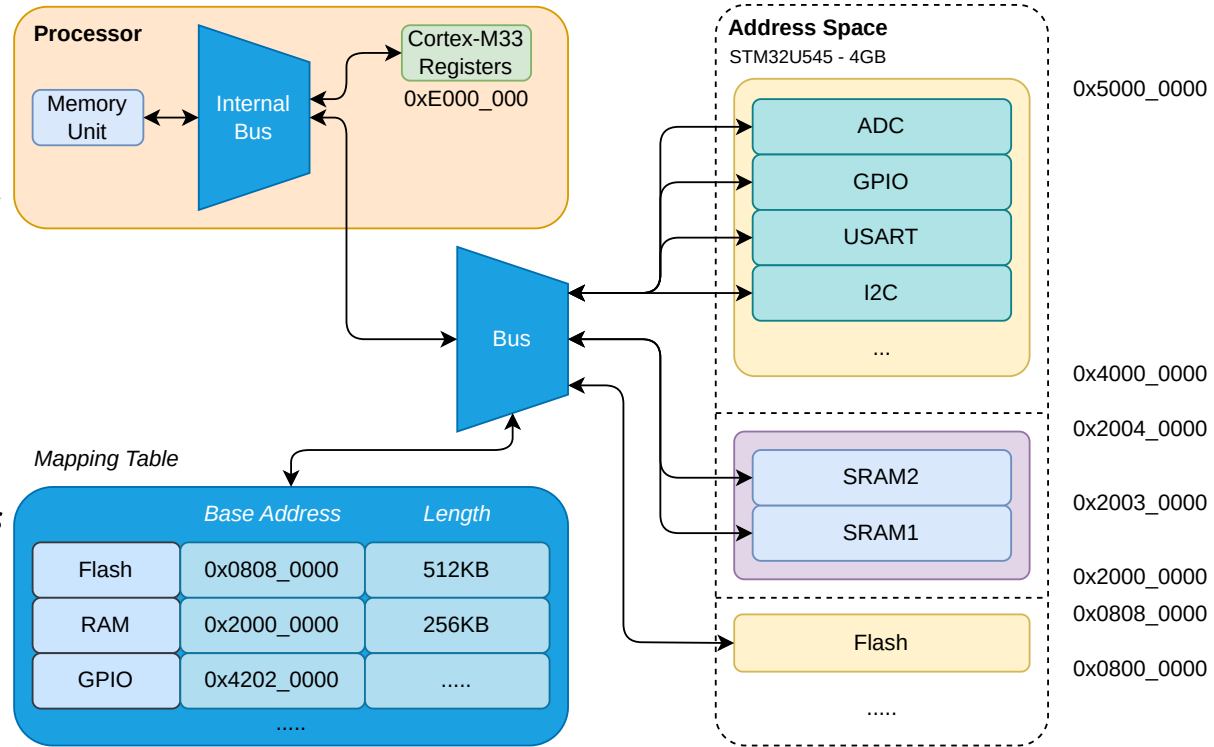




# The Bus

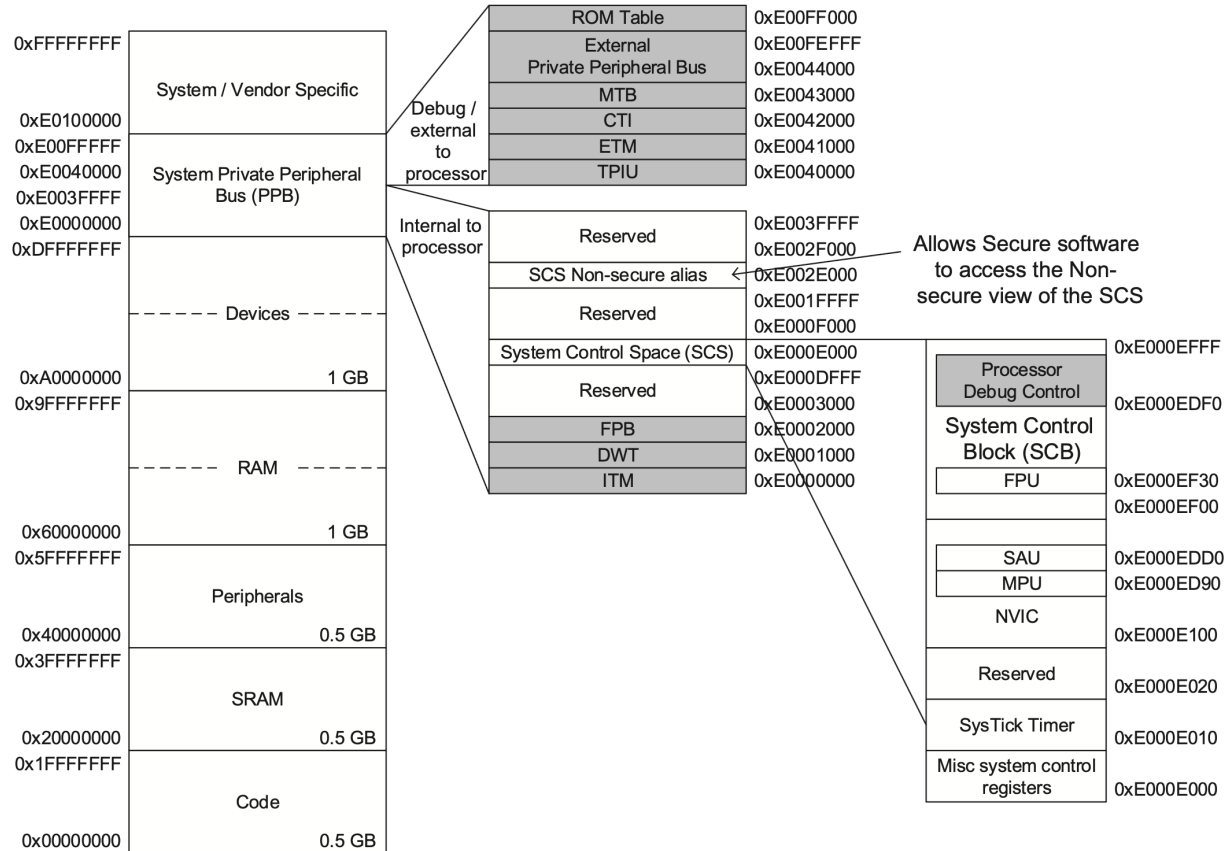
example for STM32U545RE

1. **Memory Controller** asks for data transfer
2. **Internal Bus Routes** the request
  - to the *External Bus* **or**
  - to the *Internal Peripherals*
3. **External Bus Routes** the request based on the *Address Mapping Table*
  1. to **RAM**
  2. to **Flash**
  3. to an **External Peripheral**





# Cortex-M33 Memory Layout





# System Control Registers

Cortex-M0+<sup>[1]</sup> SCR Peripheral @0xe000\_0000

Compute the actual address

$$e000\_0000_{(16)} + register\_offset$$

Register Examples:

- SYST\_CSR: **0xe000\_e010** ( $0xe000\_0000 + 0xe010$ )
- CPUID: **0xe000\_ed00** ( $0xe000\_0000 + 0xed00$ )

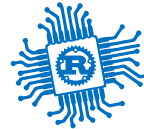
```
const SYS_CTRL_ADDR: usize = 0xe000_0000;
const CPUID_OFFSET: usize = 0xed00;

let cpuid_reg = (SYS_CTRL_ADDR+CPUID_OFFSET) as *const u32;
let cpuid_value = unsafe { *cpuid_reg };
// or
let cpuid_value = unsafe { cpuid_reg.read() };
```

⚠ Compilers optimize code and processors use cache!

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

1. Cortex-M33 has some additional registers ↔



# Compiler Optimization

compilers optimize code

Write bytes to the `UART` (serial port) data register

```
1 // we use mut as we need to write to it
2 const UART_TX: *mut u8 = 0x4003_4000 as *mut u8;
3 // b".." means ASCII string (Rust uses UTF-8 strings by default)
4 for character in b"Hello, World".iter() {
5     // character is &char, so we use *character to get the value
6     unsafe { UART_TX.write(*character); }
7 }
```

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

```
1 const UART_TX: *mut u8 = 0x4003_4000;
2 unsafe { UART_TX.write(b'd'); }
```



# No Compiler Optimization

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

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

const SYS_CTRL_ADDR: usize = 0xe000_0000;
const CPUID_OFST: usize = 0xed00;

let cpuid_reg = (SYS_CTRL_ADDR + CPUID_OFST) as *const u32;
unsafe {
    // avoid compiler optimization
    read_volatile(cpuid_reg)
}
```

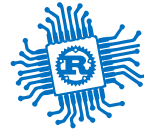
`read_volatile`,  
`write_volatile`

**no compiler  
optimization**

`read`, `write`, `*p`

**use compiler  
optimization**

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



# No Compiler Optimization

Write bytes to the `UART` (serial port) data register

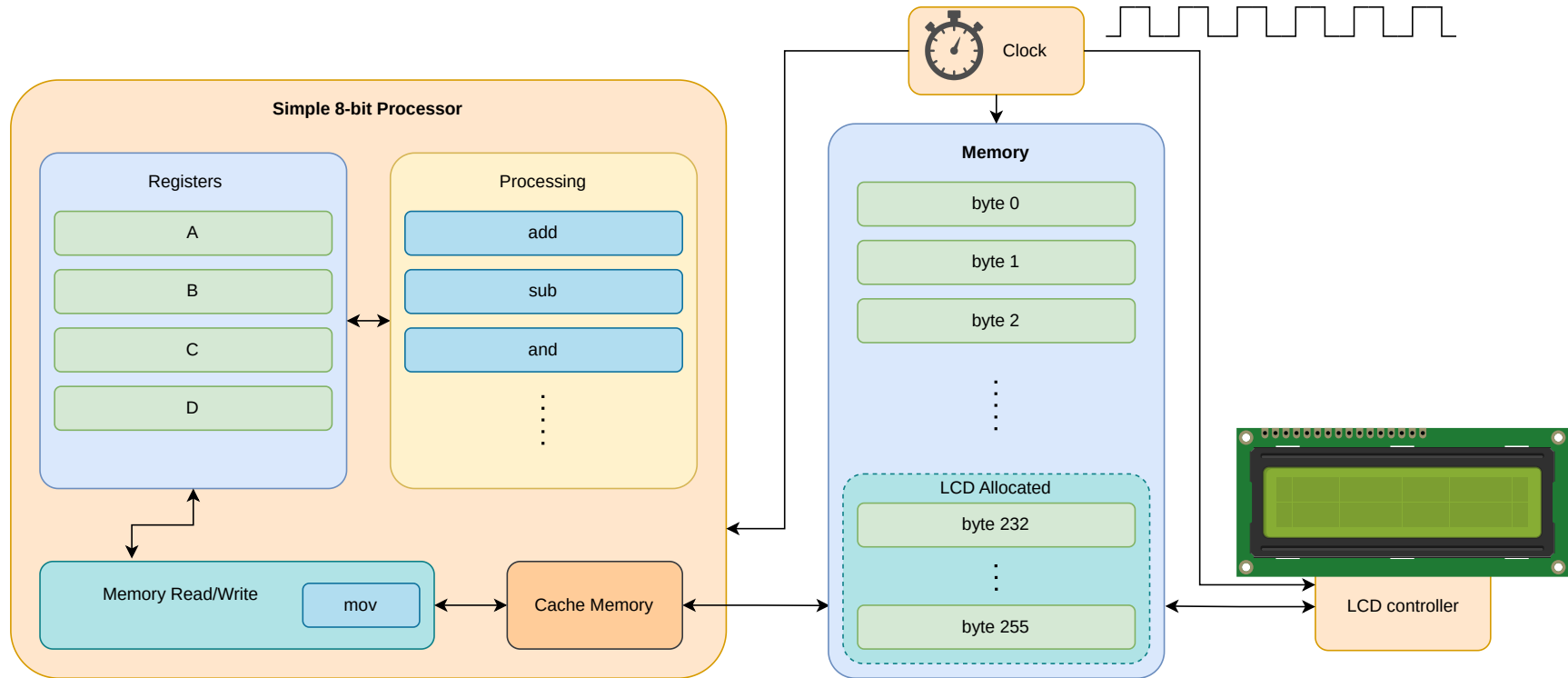
```
1  use core::ptr::write_volatile;
2
3  // we use mut as we need to write to it
4  const UART_TX: *mut u8 = 0x4003_4000 as *mut u8;
5  // b".." means ASCII string (Rust uses UTF-8 strings by default)
6  for character in b"Hello, World".iter() {
7      // character is &char, so we use *character to get the value
8      unsafe { write_volatile(UART_TX, *character); }
9  }
```

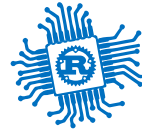
The compiler **knows** that `UART_TX` **must be written** every time.



# 8 bit processor

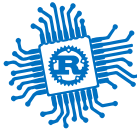
with cache





# 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_ADDR: usize = 0xe000_0000;
const CPUID_OFST: usize = 0xed00;

let cpuid_reg = (SYS_CTRL_ADDR+CPUID_OFST) as *const u32;
let cpuid_value = unsafe {
    read_volatile(cpuid_reg)
};

// shift right 20 bits and keep only the last 4 bits
let variant = (cpuid_value >> 20) & 0b1111;

// shift right 16 bits and keep only the last 4 bits
let architecture = (cpuid_value >> 16) & 0b1111;

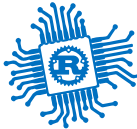
// shift right 4 bits and keep only the last 12 bits
let part_no = (cpuid_value >> 4) & 0b1111_1111_1111;

// shift right 0 bits and keep only the last 4 bits
let revision = (cpuid_value >> 0) & 0b1111;
```

## CPUID Register

Offset: 0xed00

| Bits  | Name         | Description                                                                          | Type | Reset |
|-------|--------------|--------------------------------------------------------------------------------------|------|-------|
| 31:24 | IMPLEMENTER  | Implementor code: 0x41 = ARM                                                         | RO   | 0x41  |
| 23:20 | VARIANT      | Major revision number n in the rnpn revision status: 0x0 = Revision 0.               | RO   | 0x0   |
| 19:16 | ARCHITECTURE | Constant that defines the architecture of the processor: 0xC = ARMv6-M architecture. | RO   | 0xc   |
| 15:4  | PARTNO       | Number of processor within family: 0xC60 = Cortex-M0+                                | RO   | 0xc60 |
| 3:0   | REVISION     | Minor revision number m in the rnpn 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_ADDR: usize = 0xe000_0000;
const AIRCR_OFST: usize = 0xed0c;

const VECTKEY_POS: u32 = 16;
const SYSRESETREQ_POS: u32 = 2;

let aircr_register = (SYS_CTRL + AIRCR) as *mut u32;
let mut aircr_value = unsafe {
    read_volatile(aircr_register)
};

aircr_value = aircr_value & !(0xffff << VECTKEY_POS);
aircr_value = aircr_value | (0x05fa << VECTKEY_POS);
aircr_value = aircr_value | (1 << SYSRESETREQ_POS);

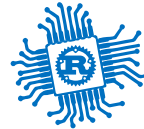
unsafe {
    write_volatile(aircr_register, aircr_value);
}
```

# AIRCR Register

Offset: 0xed0c

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

| Bits | Name          | Description                                                                                                                                                                                                                                                                                                                               | Type | 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

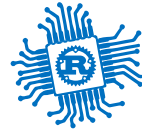
- Read
  - reads the value of a register
  - might ask the peripheral to do something
- Write
  - writes the value to a register
  - might ask the peripheral to do something
    - SYSRESETREQ

## AIRCR Register

Offset: 0xed0c

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

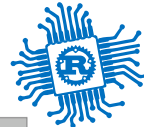
| Bits | Name          | Description                                                                                                                                                                                                                                                                                                                               | Type | 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.     | -                                                                                                                                                                                                                                                                                                                                         | -    | -     |



# SVD XML File

## System View Description

```
1  <device schemaVersion="1.1"
2    xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xs:noNamespaceSchemaLocation="CMSIS-SVD.xsd">
3    <name>RP2040</name>
4    <peripherals>
5      <name>PPB</name>
6      <baseAddress>0xe0000000</baseAddress>
7      <register>
8        <name>CPUID</name>
9        <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>
17         </field>
18         <!-- rest of the fields of the register -->
19       </fields>
20     </register>
21   </peripherals>
22 </device>
```



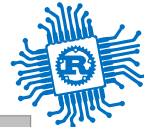
# tock-registers

## define registers format

```
1  use tock_registers::register_bitfields;
2  register_bitfields! {u32,
3      CPUID [
4          IMPLEMENTER OFFSET(24) NUMBITS(8) [],
5          VARIANT OFFSET(20) NUMBITS(4) [],
6          ARCHITECTURE OFFSET(16) NUMBITS(4) [
7              ARM_V6_M = 0xc,
8              ARM_V8_M = 0xa
9          ],
10         PARTNO OFFSET(4) NUMBITS(12) [
11             CORTEX_M0P = 0xc60,
12             CORTEX_M33 = 0xd21
13         ],
14         REVISION OFFSET(0) NUMBITS(2) []
15     ],
16     AIRCR [
17         VECTKEY OFFSET(16) NUMBITS(8) [KEY = 0x05fa],
18         ENDIANESS OFFSET(15) NUMBITS(1) [],
19         SYSRESETREQ OFFSET(2) NUMBITS(1) [],
20         VECTCLRACTIVE OFFSET(1) NUMBITS(1) []
21     ]
22 }
```

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

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



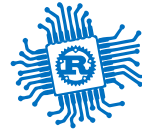
# tock-registers

define a structure for the peripheral

```
use tock_registers::register_structs;
use tock_registers::registers::{ReadOnly, ReadWrite};

// generates a C-style SysCtrl struct
register_structs! {
SysCtrl {
    // we registers up to 0xed00
    (0x0000 => _reserved1),
    // we define the CPUID register
    (0xed00 => cpuid: ReadOnly<u32, CPUID::Register>),
    // we registers up to 0xed
    (0xed04 => _reserved2),
    // we define the AIRCR register
    (0xed0c => aircr: ReadWrite<u32, AIRCR::Register>),
    // we ignore the rest of the registers
    (0xed10 => @END),
}
}
```

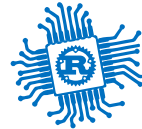
| Offset | Name                       | Info                                             |
|--------|----------------------------|--------------------------------------------------|
| 0xe010 | <a href="#">SYST_CSR</a>   | SysTick Control and Status Register              |
| 0xe014 | <a href="#">SYST_RVR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



# Reset the processor

using tock-registers

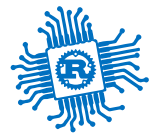
```
1  const SYS_CTRL_ADDR: usize = 0xe000_0000;
2
3  register_bitfields! {u32,
4      // ...
5      AIRCR [
6          VECTKEY OFFSET(16) NUMBITS(8) [KEY = 0x05fa],
7          ENDIANESS OFFSET(15) NUMBITS(1) [],
8          SYSRESETREQ OFFSET(2) NUMBITS(1) [],
9          VECTCLRACTIVE OFFSET(1) NUMBITS(1) []
10     ]
11 }
12
13 register_structs! {
14     SysCtrl {
15         (0xed0c => aircr: ReadWrite<u32, AIRCR::Register>),
16     }
17 }
18
19 let sys_ctrl = unsafe { &*(SYS_CTRL_ADDR as *const SysCtrl) }; // C: struct SysCtrl *sys_ctrl = SYS_CTRL_ADDR;
20
21 sys_ctrl.aircr
22     .modify(AIRCR::VECTKEY::KEY + AIRCR::SYSRESETREQ::SET);
```



# Read the CPUID

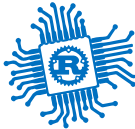
using tock-registers

```
1  const SYS_CTRL_ADDR: usize = 0xe000_0000;
2  register_bitfields! {u32,
3      CPUID [
4          IMPLEMENTER OFFSET(24) NUMBITS(8) [],
5          VARIANT OFFSET(20) NUMBITS(4) [],
6          ARCHITECTURE OFFSET(16) NUMBITS(4) [ARMv6M = 0xc, ARMv8M0 = 0xa],
7          PARTNO OFFSET(4) NUMBITS(12) [CORTEX_M0P = 0xc60, CORTEX_M33 = 0xd21],
8          REVISION OFFSET(0) NUMBITS(2) []
9      ],
10     // ...
11 }
12 let sys_ctrl = unsafe { &*(SYS_CTRL_ADDR as *const SysCtrl) };
13
14 let variant = sys_ctrl.cpuid.read(CPUID::VARIANT);
15 let revision = sys_ctrl.cpuid.read(CPUID::REVISION);
16 let architecture = sys_ctrl.cpuid.read(CPUID::ARCHITECTURE);
17 let part_no = sys_ctrl.cpuid.read(CPUID::PARTNO);
18
19 if part_no == CPUID::PARTNO::Value::CORTEX_M0P as u32 {
20     // this is a Cortex-M0+
21 } else if part_no == CPUID::PARTNO::Value::CORTEX_M33 as u32 {
```



# GPIO

General Purpose Input Output for RP2040



# Bibliography

for this section

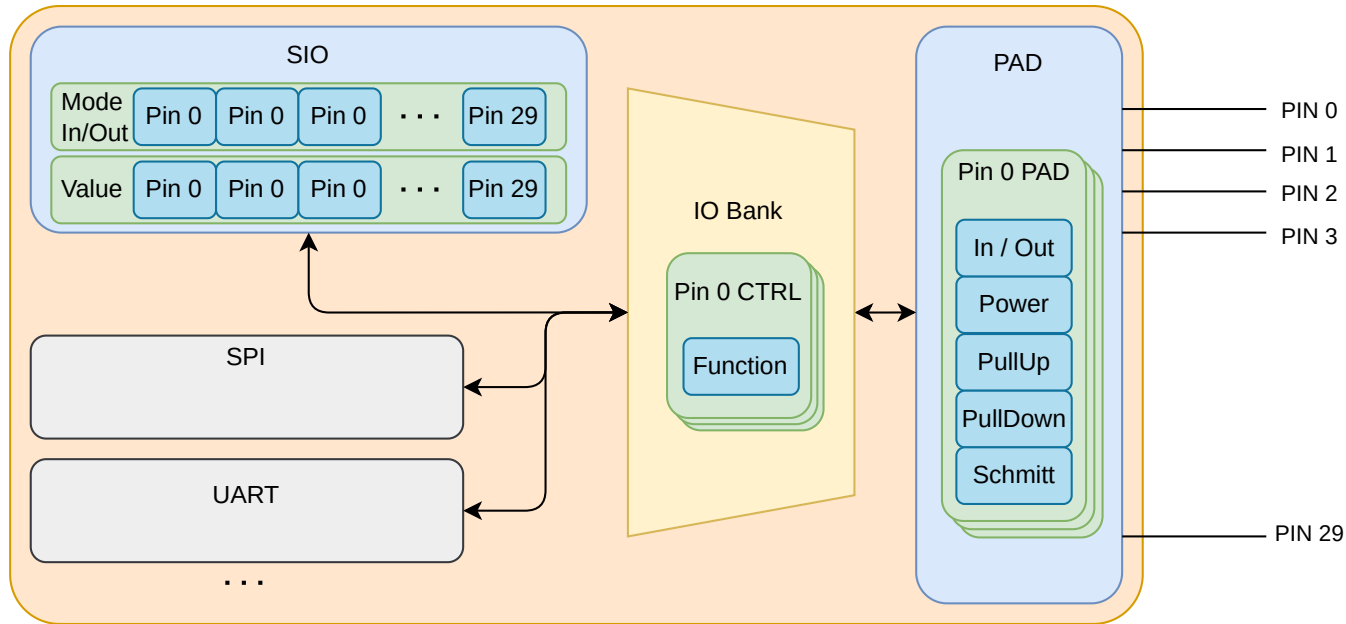
## **Raspberry Pi Ltd, *RP2040 Datasheet***

- 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)



# RP2040 GPIO Pins

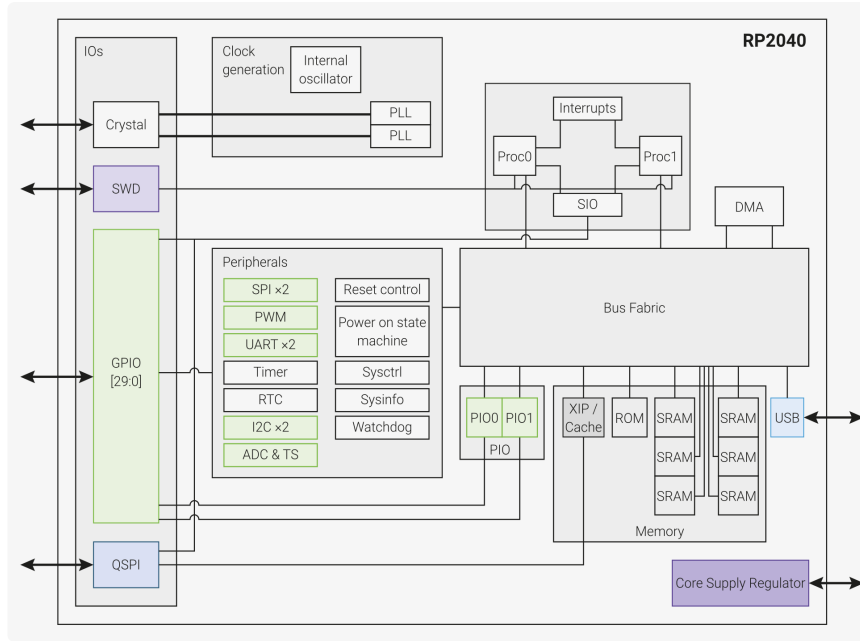
GPIO pins are connected to the processor pins through three peripherals





# GPIO

## Peripherals



SIO

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

GPIO

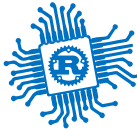
Multiplexes the functions of the GPIO pins



*SIO*: Set the pin as Input or Output

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

*PAD*: Set the pin input and output parameters



# SIO Registers

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

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

- Input
  - set GPIO\_OE bit x to 0
  - read GPIO\_IN bit x
- Output
  - set GPIO\_OE bit x to 1
  - write GPIO\_OUT bit x

## GPIO\_OE

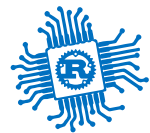
| Bits  | Description                                                                                                                                                                                                                                                                                                                            | Type | Reset      |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                                                                                                                                                                                                                                                                              | -    | -          |
| 29:0  | Set output enable (1/0 → output/input) for GPIO0...29.<br>Reading back gives the last value written.<br>If core 0 and core 1 both write to GPIO_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   | 0x00000000 |

## GPIO\_IN

| Bits  | Description                | Type | Reset      |
|-------|----------------------------|------|------------|
| 31:30 | Reserved.                  | -    | -          |
| 29:0  | Input value for GPIO0...29 | RO   | 0x00000000 |

## GPIO\_OUT

| Bits  | Description                                                                                                                                                                                                                                                                                                                                                           | Type | Reset      |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                                                                                                                                                                                                                                                                                                             | -    | -          |
| 29:0  | Set output level (1/0 → high/low) for GPIO0...29.<br>Reading back gives the last value written, NOT the input value from the pins.<br>If core 0 and core 1 both write to GPIO_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   | 0x00000000 |



# SIO Input

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

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

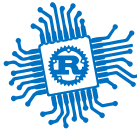
## GPIO\_OE

| Bits  | Description                                                                                                                                                                                                                                                                                                                                  | Type | Reset      |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                                                                                                                                                                                                                                                                                    | -    | -          |
| 29:0  | Set output enable (1/0 → output/input) for GPIO0...29.<br>Reading back gives the last value written.<br>If core 0 and core 1 both write to GPIO_OE simultaneously (or to a SET/CLR/XOR alias),<br>the result is as though the write from core 0 took place first,<br>and the write from core 1 was then applied to that intermediate result. | RW   | 0x00000000 |

## GPIO\_IN

| Bits  | Description                | Type | Reset      |
|-------|----------------------------|------|------------|
| 31:30 | Reserved.                  | -    | -          |
| 29:0  | Input value for GPIO0...29 | RO   | 0x00000000 |

```
1 use core::ptr::read_volatile;
2 use core::ptr::write_volatile;
3
4 const GPIO_OE: *mut u32 = 0xd000_0020 as *mut u32;
5 const GPIO_IN: *const u32 = 0xd000_0004 as *const u32;
6
7 let value = unsafe {
8     // write_volatile(GPIO_OE, !(1 << pin));
9     let mut gpio_oe = read_volatile(GPIO_OE);
10    // set bin `pin` of `gpio_oe` to 0 (input)
11    gpio_oe = gpio_oe & !(1 << pin);
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  | <code>CPUID</code>        | Processor core identifier |
| 0x004  | <code>GPIO_IN</code>      | Input value for GPIO pins |
| 0x008  | <code>GPIO_HI_IN</code>   | Input value for QSPI pins |
| 0x010  | <code>GPIO_OUT</code>     | GPIO output value         |
| 0x014  | <code>GPIO_OUT_SET</code> | GPIO output value set     |
| 0x018  | <code>GPIO_OUT_CLR</code> | GPIO output value clear   |
| 0x01c  | <code>GPIO_OUT_XOR</code> | GPIO output value XOR     |
| 0x020  | <code>GPIO_OE</code>      | GPIO output enable        |
| 0x024  | <code>GPIO_OE_SET</code>  | GPIO output enable set    |
| 0x028  | <code>GPIO_OE_CLR</code>  | GPIO output enable clear  |

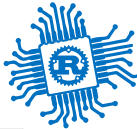
## GPIO\_OE\_SET

| Bits  | Description                                                                     | Type | Reset      |
|-------|---------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                       | -    | -          |
| 29:0  | Perform an atomic bit-clear on GPIO_OE, i.e. <code>GPIO_OE &amp;= ~wdata</code> | WO   | 0x00000000 |

## GPIO\_IN

| Bits  | Description                | Type | Reset      |
|-------|----------------------------|------|------------|
| 31:30 | Reserved.                  | -    | -          |
| 29:0  | Input value for GPIO0...29 | RO   | 0x00000000 |

```
1 use core::ptr::read_volatile;
2 use core::ptr::write_volatile;
3
4 const GPIO_OE_CLR: *mut u32 = 0xd000_0028 as *mut u32;
5 const GPIO_IN: *const u32 = 0xd000_0004 as *const u32;
6
7 let value = unsafe {
8     // set bit `pin` of `GPIO_OE` to 0 (input)
9     write_volatile(GPIO_OE_CLR, 1 << pin);
10    read_volatile(GPIO_IN) >> pin & 0b1
11 };
```



# SIO Output

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

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

## GPIO\_OE\_CLR

| Bits  | Description                                                                     | Type | Reset      |
|-------|---------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                       | -    | -          |
| 29:0  | Perform an atomic bit-clear on GPIO_OE, i.e. <code>GPIO_OE &amp;= ~wdata</code> | WO   | 0x00000000 |

## GPIO\_OUT

| Bits  | Description                                                                                                                                                                                                                                                                                                                                                                 | Type | Reset      |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                                                                                                                                                                                                                                                                                                                   | -    | -          |
| 29:0  | Set output level (1/0 → high/low) for GPIO0...29.<br>Reading back gives the last value written, NOT the input value from the pins.<br>If core 0 and core 1 both write to GPIO_OUT simultaneously (or to a SET/CLR/XOR alias),<br>the result is as though the write from core 0 took place first,<br>and the write from core 1 was then applied to that intermediate result. | RW   | 0x00000000 |

```
1 use core::ptr::read_volatile;
2 use core::ptr::write_volatile;
3
4 const GPIO_OE_SET: *mut u32 = 0xd000_0024 as *mut u32;
5 const GPIO_OUT: *mut u32 = 0xd000_0010 as *mut u32;
6
7 unsafe {
8     // set bit `pin` of GPIO_OE to 1 (output)
9     write_volatile(GPIO_OE_SET, 1 << pin);
10    // write_volatile(GPIO_OUT, (value & 0b1) << pin);
11    let mut gpio_out = read_volatile(GPIO_OUT);
12    gpio_out = gpio_out | (value & 0b1) << pin;
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  | <a href="#">CPUID</a>        | Processor core identifier |
| 0x004  | <a href="#">GPIO_IN</a>      | Input value for GPIO pins |
| 0x008  | <a href="#">GPIO_HI_IN</a>   | Input value for QSPI pins |
| 0x010  | <a href="#">GPIO_OUT</a>     | GPIO output value         |
| 0x014  | <a href="#">GPIO_OUT_SET</a> | GPIO output value set     |
| 0x018  | <a href="#">GPIO_OUT_CLR</a> | GPIO output value clear   |
| 0x01c  | <a href="#">GPIO_OUT_XOR</a> | GPIO output value XOR     |
| 0x020  | <a href="#">GPIO_OE</a>      | GPIO output enable        |
| 0x024  | <a href="#">GPIO_OE_SET</a>  | GPIO output enable set    |
| 0x028  | <a href="#">GPIO_OE_CLR</a>  | GPIO output enable clear  |

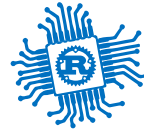
## GPIO\_OUT\_SET

| Bits  | Description                                                                | Type | Reset      |
|-------|----------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                  | -    | -          |
| 29:0  | Perform an atomic bit-set on GPIO_OUT, i.e. <code>GPIO_OUT  = wdata</code> | WO   | 0x00000000 |

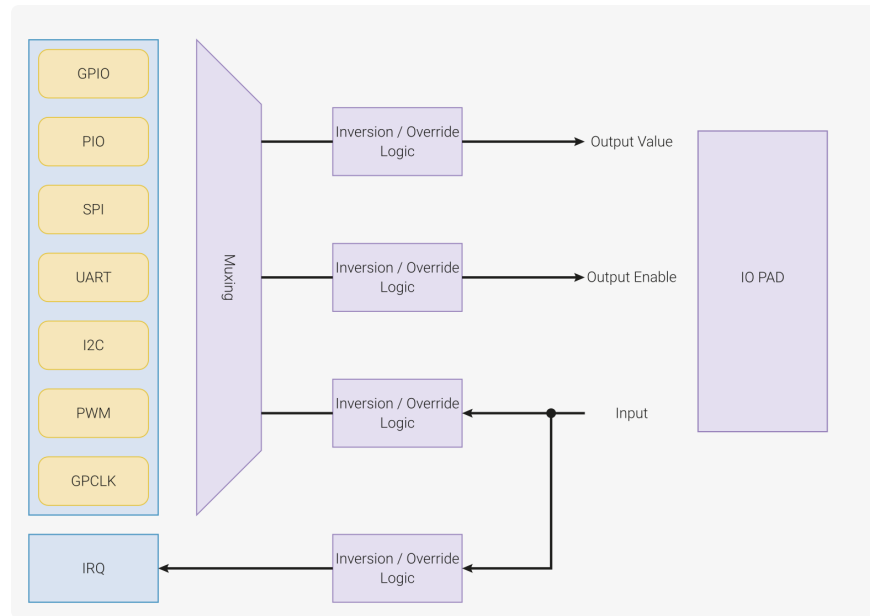
## GPIO\_OUT\_CLR

| Bits  | Description                                                                       | Type | Reset      |
|-------|-----------------------------------------------------------------------------------|------|------------|
| 31:30 | Reserved.                                                                         | -    | -          |
| 29:0  | Perform an atomic bit-clear on GPIO_OUT, i.e. <code>GPIO_OUT &amp;= ~wdata</code> | WO   | 0x00000000 |

```
1 use core::ptr::read_volatile;
2 use core::ptr::write_volatile;
3
4 const GPIO_OE_SET: *mut u32 = 0xd000_0024 as *mut u32;
5 const GPIO_OUT_SET: *mut u32 = 0xd000_0014 as *mut u32;
6 const GPIO_OUT_CLR: *mut u32 = 0xd000_0018 as *mut u32;
7
8 unsafe {
9     write_volatile(GPIO_OE_SET, 1 << pin);
10    let reg = match value {
11        0 => GPIO_OUT_CLR,
12        _ => GPIO_OUT_SET
13    };
14    write_volatile(reg, 1 << pin);
15 }
```



# IO Bank



The User Bank IO registers start at a base address of **0x40014000** (defined as **IO\_BANK0\_BASE** in SDK).

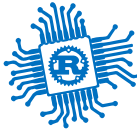
| Offset | Name                         | Info                                                  |
|--------|------------------------------|-------------------------------------------------------|
| 0x000  | <a href="#">GPIO0_STATUS</a> | GPIO status                                           |
| 0x004  | <a href="#">GPIO0_CTRL</a>   | GPIO control including function select and overrides. |

- set **FUNCSEL** to **5** (**SIO**)

## GPIOx\_CTRL

Offset: 0x004, 0x00c, ... 0x0ec ( $0x4 + 8 \times x$ )

| Bits  | Name      | Description                                                                                                                                                                                            | Type | Reset |
|-------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-------|
| 31:30 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 29:28 | IRQOVER   | 0x0 → don't invert the interrupt<br>0x1 → invert the interrupt<br>0x2 → drive interrupt low<br>0x3 → drive interrupt high                                                                              | RW   | 0x0   |
| 27:18 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 17:16 | INOVER    | 0x0 → don't invert the peri input<br>0x1 → invert the peri input<br>0x2 → drive peri input low<br>0x3 → drive peri input high                                                                          | RW   | 0x0   |
| 15:14 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 13:12 | OEOVER    | 0x0 → drive output enable from peripheral signal selected by funcsel<br>0x1 → drive output enable from inverse of peripheral signal selected by funcsel<br>0x2 → disable output<br>0x3 → enable output | RW   | 0x0   |
| 11:10 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 9:8   | OUTOVER   | 0x0 → drive output from peripheral signal selected by funcsel<br>0x1 → drive output from inverse of peripheral signal selected by funcsel<br>0x2 → drive output low<br>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_BANK0_BASE` in SDK).

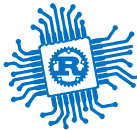
| Offset | Name                         | Info                                                  |
|--------|------------------------------|-------------------------------------------------------|
| 0x000  | <a href="#">GPIO0_STATUS</a> | GPIO status                                           |
| 0x004  | <a href="#">GPIO0_CTRL</a>   | GPIO control including function select and overrides. |

```
1 use core::ptr::read_volatile;
2 use core::ptr::write_volatile;
3
4 const GPIOX_CTRL: usize = 0x4001_4004;
5 const GPIO_OE_CLR: *mut u32 = 0xd000_0028 as *mut u32;
6 const GPIO_IN: *const u32 = 0xd000_0004 as *const u32;
7
8 let gpio_ctrl = (GPIOX_CTRL + 8 * pin) as *mut u32;
9
10 let value = unsafe {
11     write_volatile(gpio_ctrl, 5);
12     write_volatile(GPIO_OE_CLR, 1 << pin);
13     read_volatile(GPIO_IN) >> pin & 0b1
14 };
```

## GPIOx\_CTRL

Offset: 0x004, 0x00c, ... 0x0ec ( $0x4 + 8 \times x$ )

| Bits  | Name      | Description                                                                                                                                                                                            | Type | Reset |
|-------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-------|
| 31:30 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 29:28 | IRQOVER   | 0x0 → don't invert the interrupt<br>0x1 → invert the interrupt<br>0x2 → drive interrupt low<br>0x3 → drive interrupt high                                                                              | RW   | 0x0   |
| 27:18 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 17:16 | INOVER    | 0x0 → don't invert the peri input<br>0x1 → invert the peri input<br>0x2 → drive peri input low<br>0x3 → drive peri input high                                                                          | RW   | 0x0   |
| 15:14 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 13:12 | OEOVER    | 0x0 → drive output enable from peripheral signal selected by funcsel<br>0x1 → drive output enable from inverse of peripheral signal selected by funcsel<br>0x2 → disable output<br>0x3 → enable output | RW   | 0x0   |
| 11:10 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 9:8   | OUTOVER   | 0x0 → drive output from peripheral signal selected by funcsel<br>0x1 → drive output from inverse of peripheral signal selected by funcsel<br>0x2 → drive output low<br>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_BANK0_BASE` in SDK).

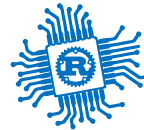
| Offset | Name                         | Info                                                  |
|--------|------------------------------|-------------------------------------------------------|
| 0x000  | <a href="#">GPIO0_STATUS</a> | GPIO status                                           |
| 0x004  | <a href="#">GPIO0_CTRL</a>   | GPIO control including function select and overrides. |

```
1 use core::ptr::read_volatile;
2 use core::ptr::write_volatile;
3
4 const GPIOX_CTRL: u32 = 0x4001_4004;
5 const GPIO_OE_SET: *mut u32 = 0xd000_0024 as *mut u32;
6 const GPIO_OUT_SET: *mut u32 = 0xd000_0014 as *mut u32;
7 const GPIO_OUT_CLR: *mut u32 = 0xd000_0018 as *mut u32;
8
9 let gpio_ctrl = (GPIOX_CTRL + 8 * pin) as *mut u32;
10 unsafe {
11     write_volatile(gpio_ctrl, 5);
12     write_volatile(GPIO_OE_SET, 1 << pin);
13     let reg = match value {
14         0 => GPIO_OUT_CLR,
15         _ => GPIO_OUT_SET
16     };
17     write_volatile(reg, 1 << pin);
18 };
```

## GPIOx\_CTRL

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

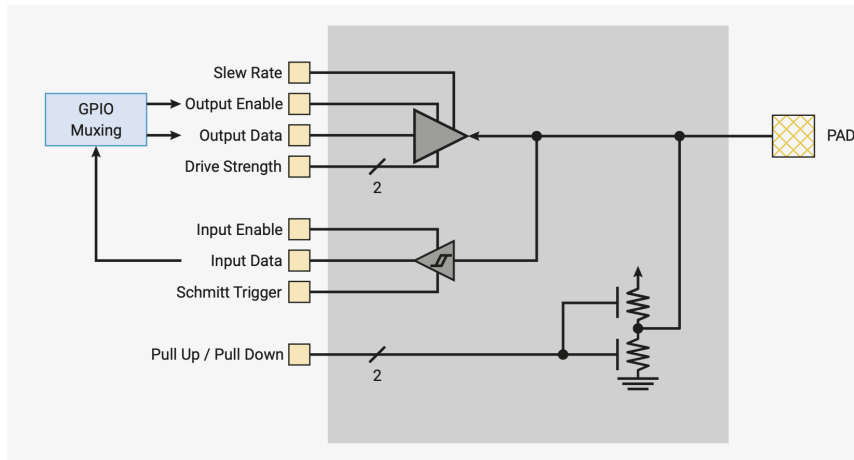
| Bits  | Name      | Description                                                                                                                                                                                            | Type | Reset |
|-------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-------|
| 31:30 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 29:28 | IRQOVER   | 0x0 → don't invert the interrupt<br>0x1 → invert the interrupt<br>0x2 → drive interrupt low<br>0x3 → drive interrupt high                                                                              | RW   | 0x0   |
| 27:18 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 17:16 | INOVER    | 0x0 → don't invert the peri input<br>0x1 → invert the peri input<br>0x2 → drive peri input low<br>0x3 → drive peri input high                                                                          | RW   | 0x0   |
| 15:14 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 13:12 | OEOVER    | 0x0 → drive output enable from peripheral signal selected by funcsel<br>0x1 → drive output enable from inverse of peripheral signal selected by funcsel<br>0x2 → disable output<br>0x3 → enable output | RW   | 0x0   |
| 11:10 | Reserved. | -                                                                                                                                                                                                      | -    | -     |
| 9:8   | OUTOVER   | 0x0 → drive output from peripheral signal selected by funcsel<br>0x1 → drive output from inverse of peripheral signal selected by funcsel<br>0x2 → drive output low<br>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  |



# Pad Control

## GPIOx Register

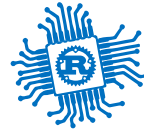
Offset: 0x004, 0x008, ... 0x078 ( $0x4 + 4 \cdot x$ )



The User Bank Pad Control registers start at a base address of `0x4001c000` (defined as `PADS_BANK0_BASE` in SDK).

| Offset | Name           | Info                             |
|--------|----------------|----------------------------------|
| 0x00   | VOLTAGE_SELECT | Voltage select. Per bank control |
| 0x04   | GPIO0          | Pad control register             |
| 0x08   | GPIO1          | Pad control register             |
| 0x0c   | GPIO2          | Pad control register             |
| 0x10   | GPIO3          | Pad control register             |
| 0x14   | GPIO4          | Pad control register             |
| 0x18   | GPIO5          | Pad control register             |

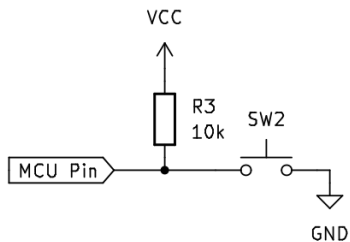
| Bits | Name      | Description                                                          | Type | Reset |
|------|-----------|----------------------------------------------------------------------|------|-------|
| 31:8 | Reserved. | -                                                                    | -    | -     |
| 7    | OD        | Output disable. Has priority over output enable from peripherals     | RW   | 0x0   |
| 6    | IE        | Input enable                                                         | RW   | 0x1   |
| 5:4  | DRIVE     | Drive strength.<br>0x0 → 2mA<br>0x1 → 4mA<br>0x2 → 8mA<br>0x3 → 12mA | RW   | 0x1   |
| 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 GPIO $x$ \_CTRL to 5
- set the GPIO\_OE\_CLR bit  $x$  to 1
- read the GPIO\_IN bit  $x$
- *adjust the GPIO $x$  fields to set the pull up/down resistor*



# Output

write a value to pin  $x$

- set the FUNCSEL field of GPIO $x$ \_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 GPIO $x$  fields to set the output current*



# GPIO

General Purpose Input Output for STM32U545



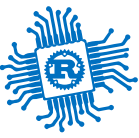
# Bibliography

## Reference manuals

### **STMicroelectronics**, *STM32U5 Series based on Arm®-based 32-bit MCUs*

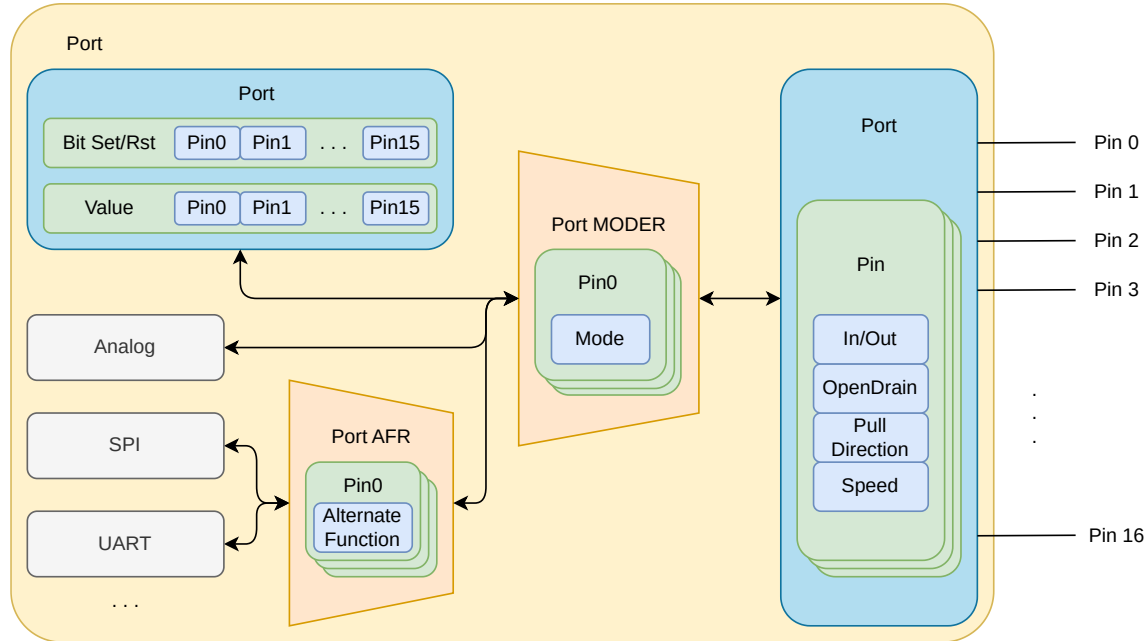
---

- Chapter 2 - *Memory and Bus architecture*
  - Section 2.3 - *Memory organization*
- Chapter 11 - *Reset and Clock Control*
  - Subsection 11.4.24 - *Peripherals clock gating and autonomous mode*
- Chapter 13 - *General-purpose I/Os (GPIO)*



# STM32U545 GPIO Pins

GPIO pins are split into smaller groups of up to 16 pins called Ports.





# GPIO Registers

## RCC AHB2 peripheral clock enable register 1

| 31       | 30       | 29   | 28        | 27        | 26   | 25   | 24         | 23         | 22   | 21          | 20      | 19    | 18     | 17      | 16    |
|----------|----------|------|-----------|-----------|------|------|------------|------------|------|-------------|---------|-------|--------|---------|-------|
| SRAM3 EN | SRAM2 EN | Res. | SDMM C2EN | SDMM C1EN | Res. | Res. | OTFDE C2EN | OTFDE C1EN | Res. | OCTOS PIMEN | SAESE N | PKAEN | RNGE N | HASHE N | AESEN |
| r/w      | r/w      |      | r/w       | r/w       |      |      | r/w        | r/w        |      | r/w         | r/w     | r/w   | r/w    | r/w     | r/w   |

| 15           | 14     | 13   | 12           | 11   | 10       | 9        | 8        | 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |
|--------------|--------|------|--------------|------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| OTGH SPHYE N | OTGE N | Res. | DCMI PSSIE N | Res. | ADC12 EN | GPIOJ EN | GPIOIE N | GPIOH EN | GPIOG EN | GPIOF EN | GPIOE EN | GPIOD EN | GPIOC EN | GPIOB EN | GPIOA EN |
| r/w          | r/w    |      | r/w          |      | r/w      | r/w      | r/w      | r/w      | r/w      | r/w      | r/w      | r/w      | r/w      | r/w      | r/w      |

- Enable I/O Port  $x$  clock signal (  $x$  from A to J )
  - Set the corresponding GPIOxEN bit
- Input
  - Set GPIOx\_MODER bits for pin  $y$  to 0b00
  - Read GPIOx\_IDR bit  $y$
- Output
  - Set GPIOx\_MODER bits for pin  $y$  to 0b01
  - Set GPIOx\_ODR bit  $y$

## GPIOx\_MODER

| 31          | 30          | 29          | 28          | 27          | 26          | 25         | 24         | 23         | 22         | 21         | 20         | 19         | 18         | 17         | 16         |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| MODE15[1:0] | MODE14[1:0] | MODE13[1:0] | MODE12[1:0] | MODE11[1:0] | MODE10[1:0] | MODE9[1:0] | MODE8[1:0] | MODE7[1:0] | MODE6[1:0] | MODE5[1:0] | MODE4[1:0] | MODE3[1:0] | MODE2[1:0] | MODE1[1:0] | MODE0[1:0] |
| r/w         | r/w         | r/w         | r/w         | r/w         | r/w         | r/w        | r/w        | r/w        | r/w        | r/w        | r/w        | r/w        | r/w        | r/w        | r/w        |

Bits 31:0 **MODEy[1:0]**: Port x configuration I/O pin  $y$  ( $y = 15$  to 0)

These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)

## GPIOx\_ODR

| 31   | 30   | 29   | 28   | 27   | 26   | 25   | 24   | 23   | 22   | 21   | 20   | 19   | 18   | 17   | 16   |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. |
|      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |

| 15   | 14   | 13   | 12   | 11   | 10   | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| OD15 | OD14 | OD13 | OD12 | OD11 | OD10 | OD9 | OD8 | OD7 | OD6 | OD5 | OD4 | OD3 | OD2 | OD1 | OD0 |
| r/w  | r/w  | r/w  | r/w  | r/w  | r/w  | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODy**: Port output data I/O pin  $y$  ( $y = 15$  to 0)

These bits can be read and written by software.

## GPIOx\_IDR

| 31   | 30   | 29   | 28   | 27   | 26   | 25   | 24   | 23   | 22   | 21   | 20   | 19   | 18   | 17   | 16   |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. |
|      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |

| 15   | 14   | 13   | 12   | 11   | 10   | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |
| r    | r    | r    | r    | r    | r    | r   | r   | r   | r   | r   | r   | r   | r   | r   | r   |

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDy**: Port x input data I/O pin  $y$  ( $y = 15$  to 0)

These bits are read-only. They contain the input value of the corresponding I/O port.

# GPIO Port enable

## RCC AHB2 peripheral clock enable register 1

### 11.8.30 RCC AHB2 peripheral clock enable register 1 (RCC\_AHB2ENR1)

Address offset: 0x08C

Reset value: 0x4000 0000 (for STM32U535/545)

Reset value: 0xC000 0000 (for STM32U575/585/59x/5Ax/5Fx/5Gx)

Access: no wait state, word, half-word, and byte access

*Note:* When the peripheral clock is not active, read or write access to peripheral registers is not supported.

|                    |             |      |                    |              |             |            |               |               |            |                |            |            |            |            |            |
|--------------------|-------------|------|--------------------|--------------|-------------|------------|---------------|---------------|------------|----------------|------------|------------|------------|------------|------------|
| 31                 | 30          | 29   | 28                 | 27           | 26          | 25         | 24            | 23            | 22         | 21             | 20         | 19         | 18         | 17         | 16         |
| SRAM3<br>EN        | SRAM2<br>EN | Res. | SDMM<br>C2EN       | SDMM<br>C1EN | Res.        | Res.       | OTFDE<br>C2EN | OTFDE<br>C1EN | Res.       | OCTOS<br>PIMEN | SAESE<br>N | PKAEN      | RNGE<br>N  | HASHE<br>N | AESEN      |
| rw                 | rw          |      | rw                 | rw           |             |            | rw            | rw            |            | rw             | rw         | rw         | rw         | rw         | rw         |
| 15                 | 14          | 13   | 12                 | 11           | 10          | 9          | 8             | 7             | 6          | 5              | 4          | 3          | 2          | 1          | 0          |
| OTGH<br>SPHYE<br>N | OTGE<br>N   | Res. | DCML<br>PSSIE<br>N | Res.         | ADC12<br>EN | GPIOE<br>N | GPIOE<br>N    | GPIOE<br>N    | GPIOE<br>N | GPIOE<br>N     | GPIOE<br>N | GPIOE<br>N | GPIOE<br>N | GPIOE<br>N | GPIOE<br>N |
| rw                 | rw          |      | rw                 |              | rw          | rw         | rw            | rw            | rw         | rw             | rw         | rw         | rw         | rw         | rw         |

...

Bit 1 **GPIOBEN**: I/O port B clock enable

This bit is set and cleared by software.

0: I/O port B clock disabled

1: I/O port B clock enabled

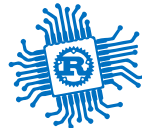
Bit 0 **GPIOAEN**: I/O port A clock enable

This bit is set and cleared by software.

0: I/O port A clock disabled

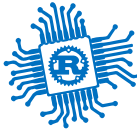
1: I/O port A clock enabled

## Memory map and peripheral register boundary addresses



| Bus  | Secure boundary address   | Nonsecure boundary address | Size (bytes) | Peripheral   | Peripheral register map                   | STM32U535/545 | STM32U575/585 | STM32U59x/5Ax | STM32U5Fx/5Gx |
|------|---------------------------|----------------------------|--------------|--------------|-------------------------------------------|---------------|---------------|---------------|---------------|
| AHB3 | 0x5602 6000 - 0x5FFF FFFF | 0x4602 6000 - 0x4FFF FFFF  | 164 M        | Reserved     | -                                         | -             | -             | -             | -             |
|      | 0x5602 5000 - 0x5602 5FFF | 0x4602 5000 - 0x4602 5FFF  | 4 K          | LPDMA1       | <a href="#">LPDMA register map</a>        | X             | X             | X             | X             |
|      | 0x5602 4000 - 0x5602 4FFF | 0x4602 4000 - 0x4602 4FFF  | 4 K          | ADF1         | <a href="#">ADF register map</a>          | X             | X             | X             | X             |
|      | 0x5602 3C00 - 0x5602 3FFF | 0x4602 3C00 - 0x4602 3FFF  | 1 K          | Reserved     | -                                         | -             | -             | -             | -             |
|      | 0x5602 3800 - 0x5602 3BFF | 0x4602 3800 - 0x4602 3BFF  | 1 K          | GTZC2_MPCBB4 | <a href="#">GTZC2 MPCBB4 register map</a> | X             | X             | X             | X             |
|      | 0x5602 3400 - 0x5602 37FF | 0x4602 3400 - 0x4602 37FF  | 1 K          | GTZC2_TZIC   | <a href="#">GTZC2 TZIC register map</a>   | X             | X             | X             | X             |
|      | 0x5602 3000 - 0x5602 33FF | 0x4602 3000 - 0x4602 33FF  | 1 K          | GTZC2_TZSC   | <a href="#">GTZC2 TZSC register map</a>   | X             | X             | X             | X             |
|      | 0x5602 2400 - 0x5602 2FFF | 0x4602 2400 - 0x4602 2FFF  | 3 K          | Reserved     | -                                         | -             | -             | -             | -             |
|      | 0x5602 2000 - 0x5602 23FF | 0x4602 2000 - 0x4602 23FF  | 1 K          | EXTI         | <a href="#">EXTI register map</a>         | X             | X             | X             | X             |
|      | 0x5602 1C00 - 0x5602 1FFF | 0x4602 1C00 - 0x4602 1FFF  | 1 K          | Reserved     | -                                         | -             | -             | -             | -             |
|      | 0x5602 1800 - 0x5602 1BFF | 0x4602 1800 - 0x4602 1BFF  | 1 K          | DAC1         | <a href="#">DAC register map</a>          | X             | X             | X             | X             |
|      | 0x5602 1400 - 0x5602 17FF | 0x4602 1400 - 0x4602 17FF  | 1 K          | Reserved     | -                                         | -             | -             | -             | -             |
|      | 0x5602 1000 - 0x5602 13FF | 0x4602 1000 - 0x4602 13FF  | 1 K          | ADC4         | <a href="#">ADC register map</a>          | X             | X             | X             | X             |
|      | 0x4602 0C00 - 0x4602 0FFF | 0x4602 0C00 - 0x4602 0FFF  | 1 K          | RCC          | <a href="#">RCC register map</a>          | X             | X             | X             | X             |

```
1 use core::ptr::write_volatile;
2 use core::ptr::read_volatile;
3
4 const RCC_BASE_ADDR: usize = 0x4602_0C00;
5 const RCC_AHB2ENR1: *mut u32 =
6     (RCC_BASE_ADDR + 0x8C) as *mut u32;
7
8 unsafe {
9     let value = read_volatile(RCC_AHB2ENR1);
10    // enable `port`, where A is 0, B is 1, ...
11    write_volatile(RCC_AHB2ENR1, value | (1 << port));
12 }
```



# GPIO Output

## GPIOx\_MODER - address offset: 0x00

|             |    |             |    |             |    |             |    |             |    |             |    |            |    |            |    |
|-------------|----|-------------|----|-------------|----|-------------|----|-------------|----|-------------|----|------------|----|------------|----|
| 31          | 30 | 29          | 28 | 27          | 26 | 25          | 24 | 23          | 22 | 21          | 20 | 19         | 18 | 17         | 16 |
| MODE15[1:0] |    | MODE14[1:0] |    | MODE13[1:0] |    | MODE12[1:0] |    | MODE11[1:0] |    | MODE10[1:0] |    | MODE9[1:0] |    | MODE8[1:0] |    |
| rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw         | rw | rw         | rw |
| 15          | 14 | 13          | 12 | 11          | 10 | 9           | 8  | 7           | 6  | 5           | 4  | 3          | 2  | 1          | 0  |
| MODE7[1:0]  |    | MODE6[1:0]  |    | MODE5[1:0]  |    | MODE4[1:0]  |    | MODE3[1:0]  |    | MODE2[1:0]  |    | MODE1[1:0] |    | MODE0[1:0] |    |
| rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw         | rw | rw         | rw |

Bits 31:0 **MODEy[1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)

## GPIOx\_ODR - address offset: 0x14

|      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 31   | 30   | 29   | 28   | 27   | 26   | 25   | 24   | 23   | 22   | 21   | 20   | 19   | 18   | 17   | 16   |
| Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. | Res. |
| 15   | 14   | 13   | 12   | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| OD15 | OD14 | OD13 | OD12 | OD11 | OD10 | OD9  | OD8  | OD7  | OD6  | OD5  | OD4  | OD3  | OD2  | OD1  | OD0  |
| rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   | rw   |

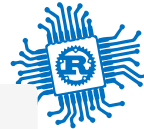
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODy**: Port output data I/O pin y (y = 15 to 0)

These bits can be read and written by software.

| Bus           | Secure boundary address   | Nonsecure boundary address | Size (bytes) | Peripheral | Peripheral register map | STM32U535/545 | STM32U575/585 | STM32U59x/5Ax | STM32U5Fx/5Gx |
|---------------|---------------------------|----------------------------|--------------|------------|-------------------------|---------------|---------------|---------------|---------------|
| AHB2 (cont'd) | 0x5202 2400 - 0x5202 27FF | 0x4202 2400 - 0x4202 27FF  | 1 K          | GPIOJ      | GPIO register map       | -             | -             | X             | X             |
|               | 0x5202 2000 - 0x5202 23FF | 0x4202 2000 - 0x4202 23FF  | 1 K          | GPIOI      |                         | -             | X             | X             | X             |
|               | 0x5202 1C00 - 0x5202 1FFF | 0x4202 1C00 - 0x4202 1FFF  | 1 K          | GPIOH      |                         | X             | X             | X             | X             |
|               | 0x5202 1800 - 0x5202 1BFF | 0x4202 1800 - 0x4202 1BFF  | 1 K          | GPIOG      |                         | X             | X             | X             | X             |
|               | 0x5202 1400 - 0x5202 17FF | 0x4202 1400 - 0x4202 17FF  | 1 K          | GPIOF      |                         | -             | X             | X             | X             |
|               | 0x5202 1000 - 0x5202 13FF | 0x4202 1000 - 0x4202 13FF  | 1 K          | GPIOE      |                         | X             | X             | X             | X             |
|               | 0x5202 0C00 - 0x5202 0FFF | 0x4202 0C00 - 0x4202 0FFF  | 1 K          | GPIOD      |                         | X             | X             | X             | X             |
|               | 0x5202 0800 - 0x5202 0BFF | 0x4202 0800 - 0x4202 0BFF  | 1 K          | GPIOC      |                         | X             | X             | X             | X             |
|               | 0x5202 0400 - 0x5202 07FF | 0x4202 0400 - 0x4202 07FF  | 1 K          | GPIOB      |                         | X             | X             | X             | X             |
|               | 0x5202 0000 - 0x5202 03FF | 0x4202 0000 - 0x4202 03FF  | 1 K          | GPIOA      |                         | X             | X             | X             | X             |

```
1 use core::ptr::{write_volatile, read_volatile};
2
3 const GPIOA_BASE_ADDR: usize = 0x4202_0000;
4 const GPIOA_MODER: *mut u32 =
5     GPIOA_BASE_ADDR as *mut u32;
6 const GPIOA_ODR: *mut u32 =
7     (GPIOA_BASE_ADDR + 0x14) as *mut u32;
8
9 unsafe {
10     let val = read_volatile(GPIOA_MODER);
11     let mask = !(0b11 << (2 * pin));
12     // '0b01' is General Purpose Output mode
13     let val = (val & mask) | (0b01 << (2 * pin));
14     write_volatile(GPIOA_MODER, val);
15     // set pin high
16     write_volatile(GPIOA_ODR, 1 << pin);
17 }
```



# GPIO Input

## GPIOx\_MODER - address offset: 0x00

|             |    |             |    |             |    |             |    |             |    |             |    |            |    |            |    |
|-------------|----|-------------|----|-------------|----|-------------|----|-------------|----|-------------|----|------------|----|------------|----|
| 31          | 30 | 29          | 28 | 27          | 26 | 25          | 24 | 23          | 22 | 21          | 20 | 19         | 18 | 17         | 16 |
| MODE15[1:0] |    | MODE14[1:0] |    | MODE13[1:0] |    | MODE12[1:0] |    | MODE11[1:0] |    | MODE10[1:0] |    | MODE9[1:0] |    | MODE8[1:0] |    |
| rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw         | rw | rw         | rw |
| 15          | 14 | 13          | 12 | 11          | 10 | 9           | 8  | 7           | 6  | 5           | 4  | 3          | 2  | 1          | 0  |
| MODE7[1:0]  |    | MODE6[1:0]  |    | MODE5[1:0]  |    | MODE4[1:0]  |    | MODE3[1:0]  |    | MODE2[1:0]  |    | MODE1[1:0] |    | MODE0[1:0] |    |
| rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw         | rw | rw         | rw |

Bits 31:0 **MODEy[1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)

## GPIOx\_PUPDR - address offset: 0x0C

|             |    |             |    |             |    |             |    |             |    |             |    |            |    |            |    |
|-------------|----|-------------|----|-------------|----|-------------|----|-------------|----|-------------|----|------------|----|------------|----|
| 31          | 30 | 29          | 28 | 27          | 26 | 25          | 24 | 23          | 22 | 21          | 20 | 19         | 18 | 17         | 16 |
| PUPD15[1:0] |    | PUPD14[1:0] |    | PUPD13[1:0] |    | PUPD12[1:0] |    | PUPD11[1:0] |    | PUPD10[1:0] |    | PUPD9[1:0] |    | PUPD8[1:0] |    |
| rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw         | rw | rw         | rw |
| 15          | 14 | 13          | 12 | 11          | 10 | 9           | 8  | 7           | 6  | 5           | 4  | 3          | 2  | 1          | 0  |
| PUPD7[1:0]  |    | PUPD6[1:0]  |    | PUPD5[1:0]  |    | PUPD4[1:0]  |    | PUPD3[1:0]  |    | PUPD2[1:0]  |    | PUPD1[1:0] |    | PUPD0[1:0] |    |
| rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw          | rw | rw         | rw | rw         | rw |

Bits 31:0 **PUPDy[1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

## GPIOx\_IDR - address offset: 0x10

|      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 31   | 30   | 29   | 28   | 27   | 26   | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  |
| Res  | Res  | Res  | Res  | Res  | Res  | Res | Res | Res | Res | Res | Res | Res | Res | Res | Res |
| 15   | 14   | 13   | 12   | 11   | 10   | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
| ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |
| r    | r    | r    | r    | r    | r    | r   | r   | r   | r   | r   | r   | r   | r   | r   | r   |

Bits 31:16 **Reserved**, must be kept at reset value.

Bits 15:0 **IDy**: Port x input data I/O pin y (y = 15 to 0)

These bits are read-only. They contain the input value of the corresponding I/O port.

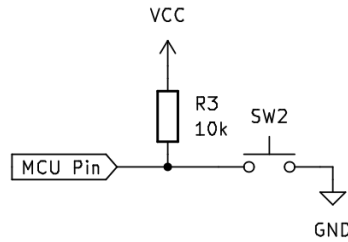
```
1 use core::ptr::{write_volatile, read_volatile};
2
3 const GPIOC_BASE_ADDR: usize = 0x4202_0800;
4 const GPIOC_MODER: *mut u32 =
5     GPIOC_BASE_ADDR as *mut u32;
6 const GPIOC_PUPDR: *mut u32 =
7     (GPIOC_BASE_ADDR + 0x0C) as *mut u32;
8 const GPIOC_IDR: *mut u32 =
9     (GPIOC_BASE_ADDR + 0x10) as *mut u32;
10
11 unsafe {
12     let val = read_volatile(GPIOC_MODER);
13     let val = val & !(0b11 << (2 * pin));
14     // configure pin as input
15     write_volatile(GPIOC_MODER, val);
16
17     let val = read_volatile(GPIOC_PUPDR);
18     let mask = !(0b11 << (2 * pin));
19     let val = (val & mask) | (0b10 << (2 * pin));
20     // configure pull down (0b10)
21     write_volatile(GPIOC_PUPDR, val);
22
23     // read pin value
24     let value = read_volatile(GPIOC_IDR);
25     let pin_val = (value & (1 << pin)) != 0;
26 }
```



# Input

read the value from pin `x` of port `y`

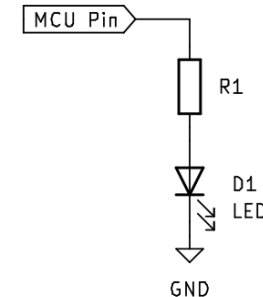
- enable the Port clock signal
- configure the pin as input by setting the pin's bits of the respective port's Mode Register, `GPIOy_MODER`, to `0b10`.
- read the `x` bit of the `GPIOy_IDR` register.
- *adjust the `GPIOy_PUPDR` fields to set the pull up/down resistor*

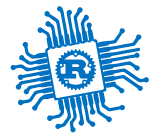


# Output

write a value to pin `x` of port `y`

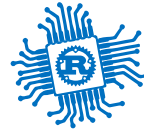
- enable the Port clock signal
- configure the pin as output by setting the pin's bits of the respective port's Mode Register, `GPIOy_MODER`, to `0b10`.
- set the `x` bit of the `GPIOy_ODR` register.
- *adjust the Port `y` registers to set the speed and configuration*





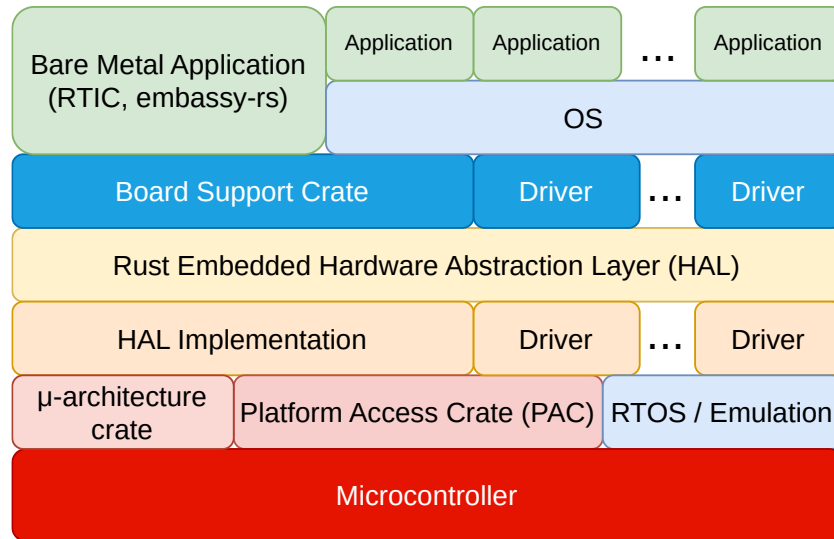
# Rust Embedded HAL

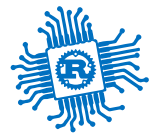
The Rust API for embedded systems



# The Rust Embedded Stack

|                           |                                                                                                 |
|---------------------------|-------------------------------------------------------------------------------------------------|
| Framework                 | Tasks, Memory Management, Network etc. <code>embassy-rs</code> , <code>rtic</code>              |
| BSC                       | Board Support Crate <code>embassy-rp</code> , <code>rp-pico</code> , <code>embassy-stm32</code> |
| <i>HAL Implementation</i> | Uses the PAC and exports a standard HAL towards the upper levels <code>embassy-stm32</code>     |
| PAC                       | Accesses registers, usually created automatically from SVD                                      |





# GPIO HAL

A set of standard traits

All devices should implement these traits for GPIO.

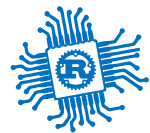
```
1 pub enum PinState {  
2     Low,  
3     High,  
4 }
```

## 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<(), Self::Error> { ... }  
}
```



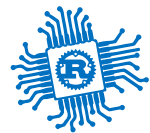
# Bare metal

This is how a Rust application would look like

```
1  #![no_std]
2  #![no_main]
3
4  use cortex_m_rt::entry;
5  use core::panic::PanicInfo;
6
7  #[entry]
8  fn main() -> ! {
9      // your code goes here
10
11     loop { }
12 }
13
14 #[panic_handler]
15 pub fn panic(_info: &PanicInfo) -> ! {
16     loop { }
17 }
```

## Rules

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

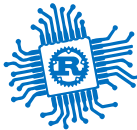


# Bare metal without PAC & HAL

This is how a Rust application would look like on the RP2

```
1  #![no_std]
2  #![no_main]
3
4  use core::ptr::{read_volatile, write_volatile};
5  use cortex_m_rt::entry;
6  use core::panic::PanicInfo;
7
8  const PIN: u32 = ...; // 0 to 20
9
10 const GPIOX_CTRL: u32 = 0x4001_4004;
11 const GPIO_OE_SET: *mut u32 = 0xd000_0024 as *mut u32;
12 const GPIO_OUT_SET: *mut u32 = 0xd000_0014 as *mut u32;
13 const GPIO_OUT_CLR: *mut u32 = 0xd000_0018 as *mut u32;
14
15 #[panic_handler]
16 pub fn panic(_info: &PanicInfo) -> ! {
17     loop { }
18 }
```

```
18  #[entry]
19  fn main() -> ! {
20      let gpio_ctrl = (GPIOX_CTRL + 8 * PIN) as *mut u32;
21      unsafe {
22          write_volatile(gpio_ctrl, 5);
23          write_volatile(GPIO_OE_SET, 1 << PIN);
24          let reg = match value {
25              0 => GPIO_OUT_CLR,
26              _ => GPIO_OUT_SET
27          };
28          write_volatile(reg, 1 << PIN);
29      };
30
31      loop { }
32  }
```

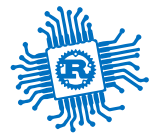


# Bare metal without PAC & HAL

This is how a Rust application would look like on the STM32U545RE

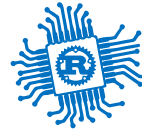
```
1  #![no_std]
2  #![no_main]
3
4  use core::ptr::{read_volatile, write_volatile};
5  use cortex_m_rt::entry;
6  use core::panic::PanicInfo;
7
8  const PORT_x: u32 = ...; // A is 0, B is 1, ...
9  const PIN: u32 = ...;    // Ranging from 0 to 15
10
11  const GPIOx_BASE_ADDR: usize =
12      0x4202_0000 + 0x400 * PORT_x;
13  const GPIOx_MODER: *mut u32 =
14      GPIOx_BASE_ADDR as *mut u32;
15  const GPIOx_ODR: *mut u32 =
16      (GPIOx_BASE_ADDR + 0x14) as *mut u32;
17  const RCC_BASE_ADDR: usize = 0x4602_0C00;
18  const RCC_AHB2ENR1: *mut u32 =
19      (RCC_BASE_ADDR + 0x8C) as *mut u32;
```

```
20  #[panic_handler]
21  pub fn panic(_info: &PanicInfo) -> ! {
22      loop { }
23  }
24
25  #[cortex_m_rt::entry]
26  fn main() -> ! {
27      unsafe {
28          let mut val = read_volatile(RCC_AHB2ENR1);
29          let val = val | (1 << PORT_x);
30          write_volatile(RCC_AHB2ENR1, val);
31      }
32
33      unsafe {
34          let val = read_volatile(GPIOx_MODER);
35          let mask = !(0b11 << (2 * PIN));
36          let val = (val & mask) | (0b01 << (2 * PIN));
37          write_volatile(GPIOx_MODER, val);
38          write_volatile(GPIOx_ODR, 1 << PIN);
39      }
40  }
```



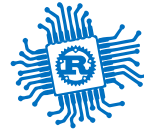
# embassy-rs

Embedded Asynchronous



# embassy-rs

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



# GPIO Input

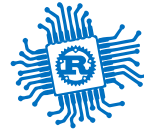
## RP2

```
1  #![no_std]
2  #![no_main]
3
4  use embassy_executor::Spawner;
5  use embassy_rp::gpio;
6  use gpio::{Input, Pull};
7
8  #[embassy_executor::main]
9  async fn main(_spawner: Spawner) {
10     let p = embassy_rp::init(Default::default());
11     let pin = Input::new(p.PIN_3, Pull::Up);
12
13     if pin.is_high() {
14
15     } else {
16
17     }
18 }
```

## STM32U545RE

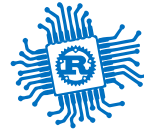
```
1  #![no_std]
2  #![no_main]
3
4  use embassy_executor::Spawner;
5  use embassy_stm32::gpio;
6  use gpio::{Input, Pull};
7
8  #[embassy_executor::main]
9  async fn main(_spawner: Spawner) {
10     let p = embassy_stm32::init(Default::default());
11     let pin = Input::new(p.PC13, Pull::Down);
12
13     if pin.is_high() {
14
15     } else {
16
17     }
18 }
```

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



# GPIO Input for RP2 vs STM32U545RE

```
1 // STM32U545RE
2
3 #![no_std]
4 #![no_main]
5
6 use embassy_executor::Spawner;
7 use embassy_stm32::gpio;
8 use gpio::{Input, Pull};
9
10 #[embassy_executor::main]
11 async fn main(_spawner: Spawner) {
12     let p = embassy_stm32::init(Default::default());
13     let pin = Input::new(p.PC13, Pull::Down);
14
15     if pin.is_high() {
16
17     } else {
18
19     }
20 }
```



# GPIO Output

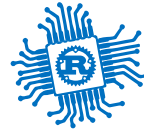
## RP2

```
1  #![no_std]
2  #![no_main]
3
4  use embassy_executor::Spawner;
5  use embassy_rp::gpio;
6  use gpio::{Level, Output};
7
8  #[embassy_executor::main]
9  async fn main(_spawner: Spawner) {
10     let p = embassy_rp::init(Default::default());
11     let mut pin = Output::new(p.PIN_2, Level::Low);
12
13     pin.set_high();
14 }
```

## STM32U545RE

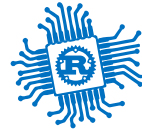
```
1  #![no_std]
2  #![no_main]
3
4  use embassy_executor::Spawner;
5  use embassy_stm32::gpio;
6  use gpio::{Level, Output, Speed};
7
8  #[embassy_executor::main]
9  async fn main(_spawner: Spawner) {
10     let p = embassy_stm32::init(Default::default());
11     let mut pin = Output::new(
12         p.PA5,
13         Level::Low,
14         Speed::Medium
15     );
16
17     pin.set_high();
18 }
```

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



# GPIO Output for RP2 vs STM32U545RE

```
1 // STM32U545RE
2
3 #[no_std]
4 #[no_main]
5
6 use embassy_executor::Spawner;
7 use embassy_stm32::gpio;
8 use gpio::{Level, Output, Speed};
9
10 #[embassy_executor::main]
11 async fn main(_spawner: Spawner) {
12     let p = embassy_stm32::init(Default::default());
13     let mut pin = Output::new(p.PA5, Level::Low, Speed::Medium);
14
15     pin.set_high();
16 }
```



# Conclusion

we talked about

- Memory Mapped IO
- RP2040 GPIO
  - Single Cycle IO
  - IO Bank
  - Pad
- STM32U545RE
  - GPIO
  - Port MODER
  - Port Setup
- The Rust embedded standard stack
- Bare metal Rust
- The embassy-rs framework