



Exceptions and Interrupts

Lecture 3



Exceptions and Interrupts

used by RP2040

- Exceptions
- Interrupts
- Boot



Exceptions

for the ARM Cortex-M0+ processor



Bibliography

for this section

Joseph Yiu, *The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors, 2nd Edition*

- Chapter 4 - *Architecture*
 - Section 4.4 - *Stack Memory Operations*
 - Section 4.5 - *Exceptions and Interrupts*
- Chapter 8 - *Exceptions and Interrupts*
 - Section 8.1 - *What are Exceptions and Interrupts*
 - Section 8.2 - *Exception types on Cortex-M0 and Cortex-M0+*



Processor Exceptions

what happens if something does not work as required





ARM Cortex-M0+ Exceptions

what happens if something does not work as required





Exception (HardFault) Handling

ARM Cortex-M0+ has one **actual exception**, *HardFault*



- the exception table of RP2040 at address 0x1000_0100 (start of the boot area + 4 bytes)
- the processor generates a *Reset* exception when it starts



Interrupts

for ARM Cortex-M0+



Bibliography

for this section

Joseph Yiu, *The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors, 2nd Edition*

- Chapter 8 - *Exceptions and Interrupts*
 - Section 8.1 - *What are Exceptions and Interrupts*
 - Section 8.3 - *Brief Overview of the NVIC*
 - Section 8.4 - *Definition of Exception Priority Levels*
 - Section 8.5 - *Vector Table*
 - Section 8.6 - *Exception Sequence Overview*
- Chapter 11 - *Fault Handling*
 - Section 11.1 - *Fault Exception Overview*
 - Section 11.2 - *What Can Cause a Fault*
 - Section 11.7 - *Lockup*



ARM Cortex-M0+ Interrupts

some hardware device notifies the MCU





Interrupt Handling

ARM Cortex-M0+



IRQ Interrupt Request

ISR Interrupt Service Routine

- the interrupt vector (table) of RP2040 starts at address 0x1000_0040 (after the exceptions table with 15 interrupts)
- ARM Cortex-M0+ has a maximum of 32 interrupt requests (IRQs)



Exceptions are Software Interrupt Requests

with a negative IRQ number and a higher priority



- Reset (-14)
- HardFault (-13)
- SVC (-5)
- PendSV (-2)
- SysTick (-1)

| IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source |
|-----|------------------|-----|------------------|-----|------------------|-----|------------------|-----|------------------|
| 0 | TIMER_IRQ_0 | 6 | XIP_IRQ | 12 | DMA_IRQ_1 | 18 | SPI0_IRQ | 24 | I2C1_IRQ |
| 1 | TIMER_IRQ_1 | 7 | PIO0_IRQ_0 | 13 | IO_IRQ_BANK0 | 19 | SPI1_IRQ | 25 | RTC_IRQ |
| 2 | TIMER_IRQ_2 | 8 | PIO0_IRQ_1 | 14 | IO_IRQ_QSPI | 20 | UART0_IRQ | | |
| 3 | TIMER_IRQ_3 | 9 | PIO1_IRQ_0 | 15 | SIO_IRQ_PROC0 | 21 | UART1_IRQ | | |
| 4 | PWM_IRQ_WRAP | 10 | PIO1_IRQ_1 | 16 | SIO_IRQ_PROC1 | 22 | ADC_IRQ_FIFO | | |
| 5 | USBCTRL_IRQ | 11 | DMA_IRQ_0 | 17 | CLOCKS_IRQ | 23 | I2C0_IRQ | | |



Boot

of the RP2040



Bibliography

for this section

Raspberry Pi Ltd, *RP2040 Datasheet*

- Chapter 2 - *System Description*
 - Section 2.7 - *Boot sequence*
 - Section 2.8 - *Bootrom*
 - Subsection 2.8.1 - *Processor Controlled Boot Sequence*



Boot

how the ARM Cortex-M0+ starts



- the *start_address* for RP2040 is 0x1000_0100
- RP2040 has another boot loader that it loads from 0x1000_0000



Boot

The RP2040 boot process



* drawing is not at scale, code and data are significantly greater than the interrupt vector

The internal boot loader cannot be overwritten and assures that bricking the device is difficult.



Set Fault Handler

bare metal, pac or embassy-rs

```
// defined by the cortex-m-rt crate
pub struct ExceptionFrame {
    r0: u32,
    r1: u32,
    r2: u32,
    r3: u32,
    r12: u32,
    lr: u32,
    pc: u32,
    xpsr: u32,
}
```

HardFault never returns

```
1  #[exception]
2  unsafe fn HardFault(_frame: &ExceptionFrame) -> ! {
3      panic!("HardFault {:?}", frame);
4  }
```



Set SysTick Handler

bare metal, PAC or embassy-rs

```
1  #[exception]
2  unsafe fn SysTick() {
3      // execute at a fixed interval
4  }
```



Set Interrupt Handlers

bare metal, PAC

embassy-rs already defined the interrupts as it needs them

| IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source |
|-----|------------------|-----|------------------|-----|------------------|-----|------------------|-----|------------------|
| 0 | TIMER_IRQ_0 | 6 | XIP_IRQ | 12 | DMA_IRQ_1 | 18 | SPI0_IRQ | 24 | I2C1_IRQ |
| 1 | TIMER_IRQ_1 | 7 | PIO0_IRQ_0 | 13 | IO_IRQ_BANK0 | 19 | SPI1_IRQ | 25 | RTC_IRQ |
| 2 | TIMER_IRQ_2 | 8 | PIO0_IRQ_1 | 14 | IO_IRQ_QSPI | 20 | UART0_IRQ | | |
| 3 | TIMER_IRQ_3 | 9 | PIO1_IRQ_0 | 15 | SIO_IRQ_PROC0 | 21 | UART1_IRQ | | |
| 4 | PWM_IRQ_WRAP | 10 | PIO1_IRQ_1 | 16 | SIO_IRQ_PROC1 | 22 | ADC_IRQ_FIFO | | |
| 5 | USBCTRL_IRQ | 11 | DMA_IRQ_0 | 17 | CLOCKS_IRQ | 23 | I2C0_IRQ | | |

```
1  #[interrupt]
2  unsafe fn IO_IRQ_BANK0 {
3      // so some work when a pin interrupt triggers
4  }
```



Use interrupts in embassy-rs

embassy-rs registers interrupt handlers and exposes a high level API

| IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source | IRQ | Interrupt Source |
|-----|------------------|-----|------------------|-----|------------------|-----|------------------|-----|------------------|
| 0 | TIMER_IRQ_0 | 6 | XIP_IRQ | 12 | DMA_IRQ_1 | 18 | SPI0_IRQ | 24 | I2C1_IRQ |
| 1 | TIMER_IRQ_1 | 7 | PI0B_IRQ_0 | 13 | IO_IRQ_BANK0 | 19 | SPI1_IRQ | 25 | RTC_IRQ |
| 2 | TIMER_IRQ_2 | 8 | PI0B_IRQ_1 | 14 | IO_IRQ_QSPI | 20 | UART0_IRQ | | |
| 3 | TIMER_IRQ_3 | 9 | PI01_IRQ_0 | 15 | SIO_IRQ_PROC0 | 21 | UART1_IRQ | | |
| 4 | PWM_IRQ_WRAP | 10 | PI01_IRQ_1 | 16 | SIO_IRQ_PROC1 | 22 | ADC_IRQ_FIFO | | |
| 5 | USBCTRL_IRQ | 11 | DMA_IRQ_0 | 17 | CLOCKS_IRQ | 23 | I2C0_IRQ | | |

```
1  #[embassy_executor::main]
2  async fn main(_spawner: Spawner) {
3      let p = embassy_rp::init(Default::default());
4      let mut button = Input::new(p.PIN_20, Pull::None);
5
6      loop {
7          info!("Waiting for the button press");
8
9          // waits for interrupt (sent by button)
10         // IO_IRQ_BANK0
11         button.wait_for_high().await;
12
13         info!("Button was pressed");
14     }
15 }
```



Conclusion

we talked about

- Exceptions
- Interrupts
- How the RP2040 boots and loads the software