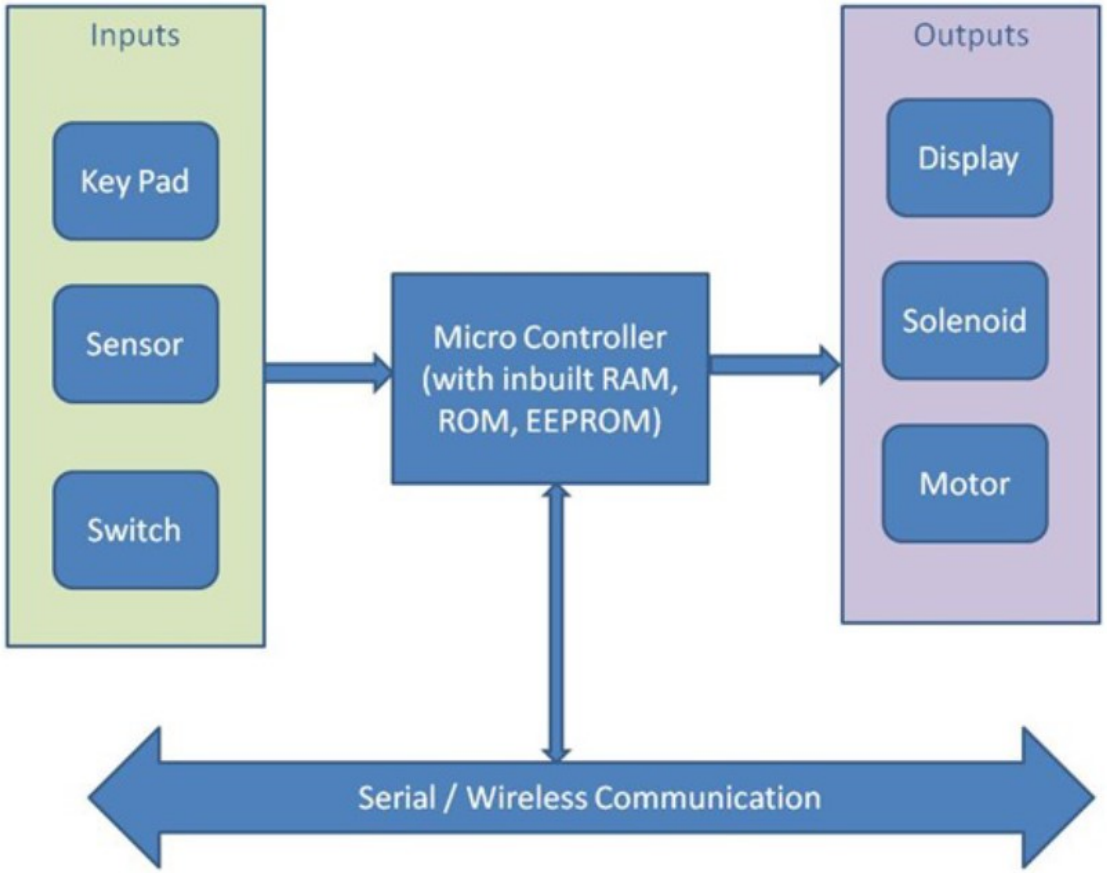
Návrh systémů IoT

5. IoT zařízení, Raspberry Pi Pico.

Stanislav Vítek
Katedra radioelektroniky
České vysoké učení technické v Praze

IoT zařízení



Počítače: k všeobecnému použití (general purpose) nebo dedikované

Obecné procesory

- Programovatelné zařízení
  - mikroprocesor
  - mikrokontrolér
- Hlavní části
  - Programová a datová paměť
  - Obecná datová cesta
  - Výbava registry
  - Obecná ALU
- Aplikačně-specifické procesory (ASIC)
  - Optimalizovaná datová cesta
  - Speciální funkční bloky

Dedikované procesory

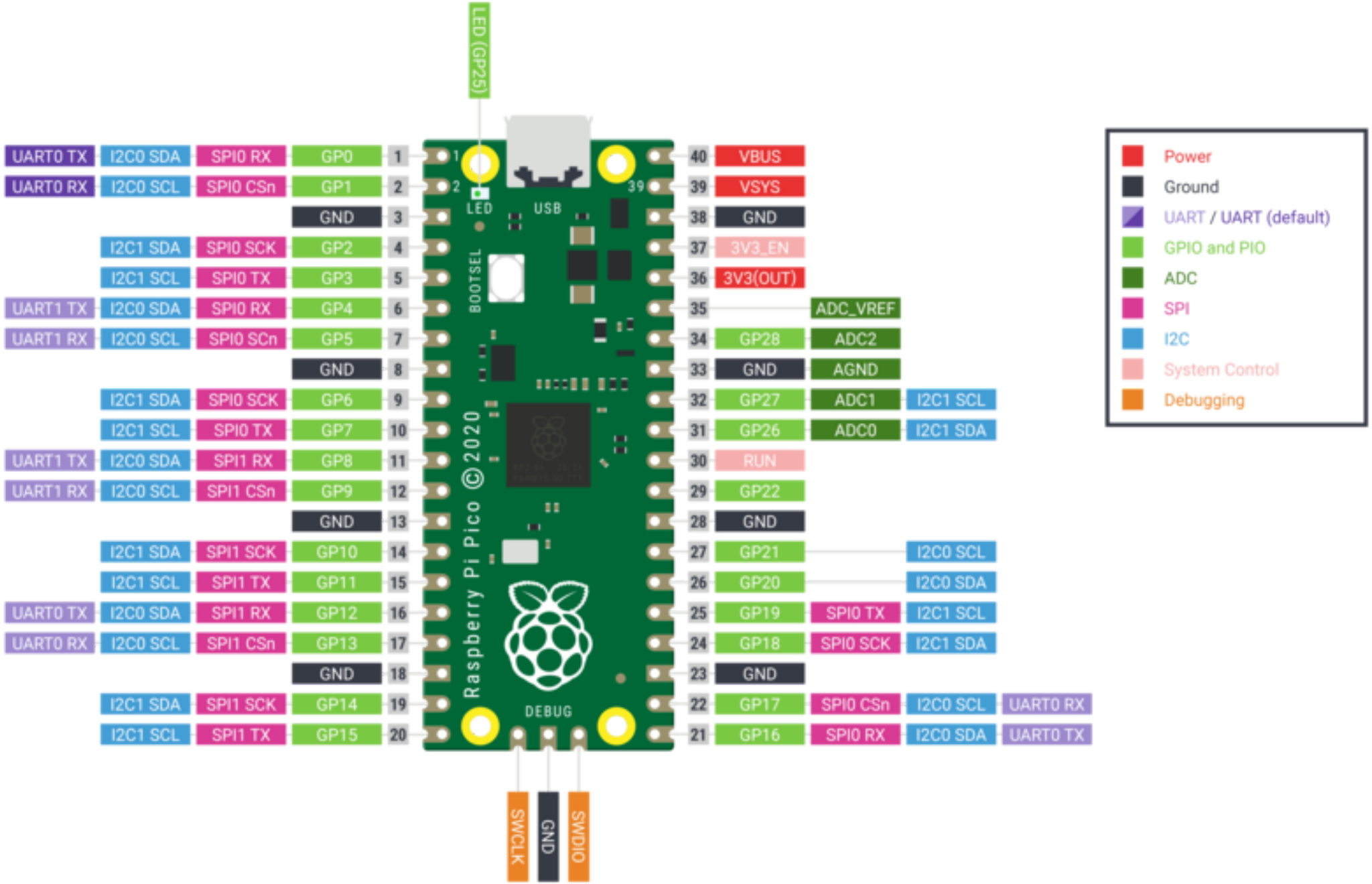
- Jednoúčelový digitální obvod
- Hlavní části
  - Komponenty nutné k provádějí jednoho programu
  - Nemá programovou paměť
- Výhody
  - Malý
  - Rychlý
  - Nízká spotřeba

Vestavné systémy

- Dedikovaná funkcionalita
- Provoz v reálném čase
- Malé rozměry a nízká hmotnost
- Nízká spotřeba energie
- Drsné prostředí
- Provoz kritický z hlediska bezpečnosti
- Cenově výhodné

Raspberry Pi Pico

- Všechny předchozí desky Raspberry Pi, jako například Raspberry Pi 3 Model B+, Raspberry Pi 4 Model B nebo menší Raspberry Pi Zero, byly vybaveny procesory Broadcom (BCM2835, BCM2836, BCM2711 atd.).
- Raspberry Pi Pico je vybaven RP2040, což je mikrokontrolér navržený Raspberry Pi, první vlastní procesor od Raspberry Pi Foundation.
- RP2040 je založen na dvou jádrech ARM Cortex-M0+ s taktovací frekvencí až 133 MHz a je vyráběn 40 nm technologií.
- MCU RP2040 má také port MicroPython a zavaděč UF2 v paměti ROM pro snadné nahrání programu.

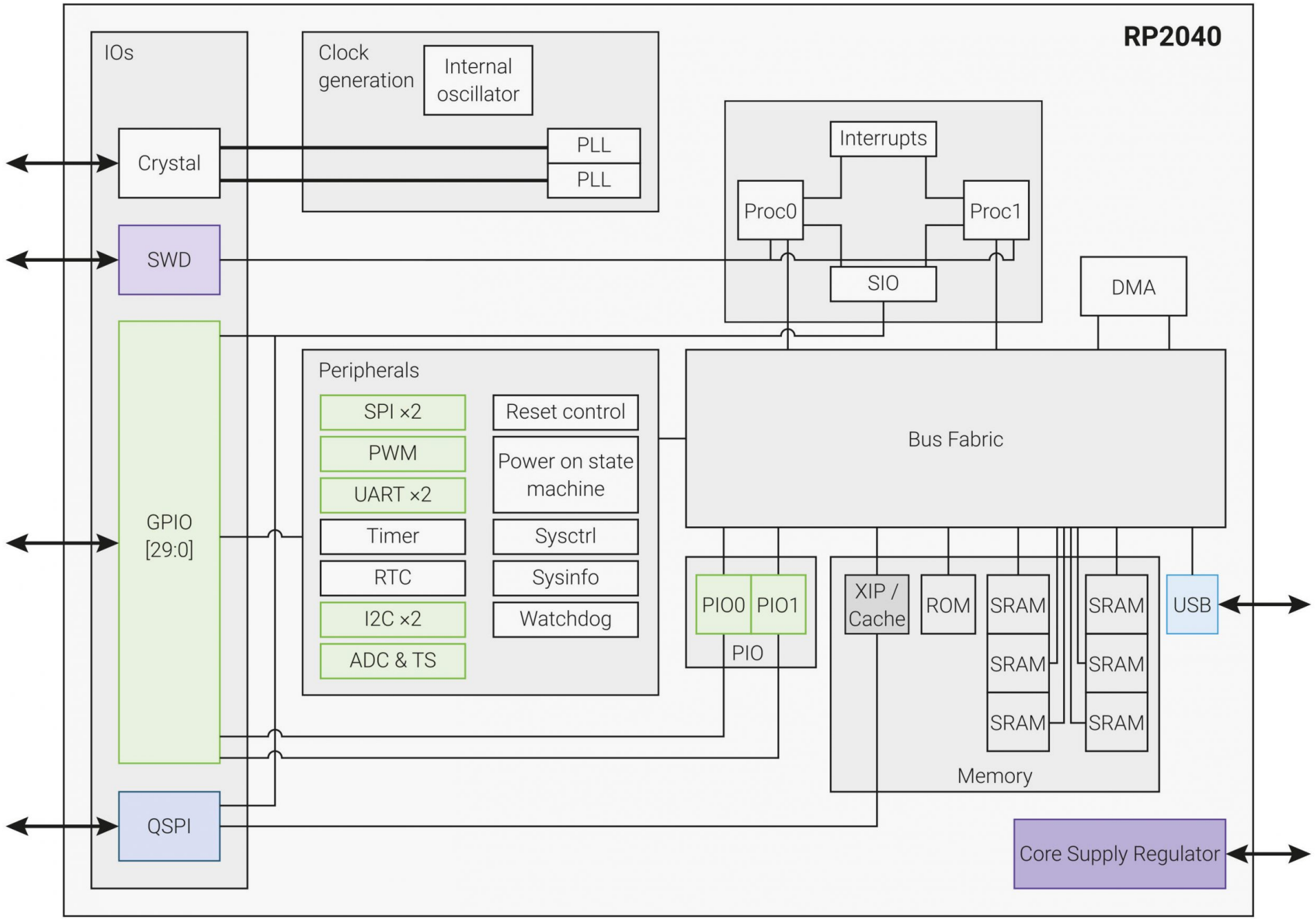

LED (GP25)
UART0 TX I2C0 SDA SPI0 RX GP0 1
UART0 RX I2C0 SCL SPI0 CSn GP1 2
GND 3
I2C1 SDA SPI0 SCK GP2 4
I2C1 SCL SPI0 TX GP3 5
UART1 TX I2C0 SDA SPI0 RX GP4 6
UART1 RX I2C0 SCL SPI0 SCn GP5 7
GND 8
I2C1 SDA SPI0 SCK GP6 9
I2C1 SCL SPI0 TX GP7 10
UART1 TX I2C0 SDA SPI1 RX GP8 11
UART1 RX I2C0 SCL SPI1 CSn GP9 12
GND 13
I2C1 SDA SPI1 SCK GP10 14
I2C1 SCL SPI1 TX GP11 15
UART0 TX I2C0 SDA SPI1 RX GP12 16
UART0 RX I2C0 SCL SPI1 CSn GP13 17
GND 18
I2C1 SDA SPI1 SCK GP14 19
I2C1 SCL SPI1 TX GP15 20
40 VBUS
39 VSYS
38 GND
37 3V3_EN
36 3V3(OUT)
35 ADC_VREF
34 GP28 ADC2
33 GND AGND
32 GP27 ADC1 I2C1 SCL
31 GP26 ADC0 I2C1 SDA
30 RUN
29 GP22
28 GND
27 GP21 I2C0 SCL
26 GP20 I2C0 SDA
25 GP19 SPI0 TX I2C1 SCL
24 GP18 SPI0 SCK I2C1 SDA
23 GND
22 GP17 SPI0 CSn I2C0 SCL UART0 RX
21 GP16 SPI0 RX I2C0 SDA UART0 TX
LED
USB
BOOTSEL
Raspberry Pi Pico ©2020
DEBUG
SWCLK
GND
SWDIO
Power
Ground
UART / UART (default)
GPIO and PIO
ADC
SPI
I2C
System Control
Debugging

Raspberry Pi Pico

- Mikrokontrolér RP2040
- 2 MB paměti SPI Flash
- Port Micro-USB typu B pro napájení a programování
- 40 vstupních a výstupních pinů typu DIP s okrajovým zalisováním
- 3pinové rozhraní ARM pro sériové ladění (SWD)
- 12 MHz krystalový oscilátor
- Tlačítko pro výběr bootování
- Jedna uživatelská LED dioda (připojená ke GPIO 25, na modelu **W** k wi-fi řadiči)
- 3,3V Buck-Boost SMPS převodník

Mikrokontrolér RP2040

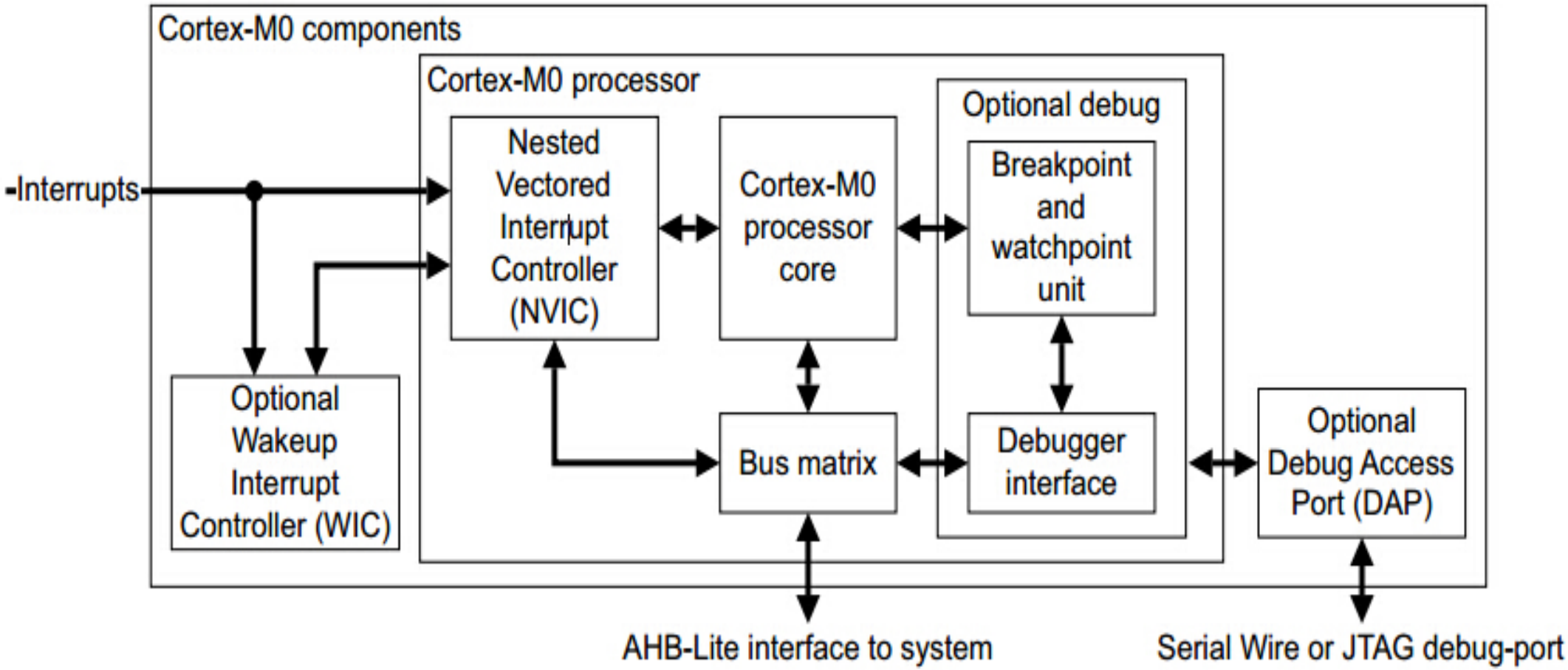
- Dvě jádra ARM Cortex-M0+
- Taktovací frekvence až 133 MHz
- 264 KB vestavěné paměti SRAM
- 30 pinů GPIO
- Až 16 MB paměti Flash mimo čip
- Čtyřkanálový ADC s 12bitovým rozlišením
- Programovatelné IO (PIO)
- Další periferie - 2x UART, 2x SPI řadič, 2x I2C řadič, 16 PWM kanálů, USB 1.1 řadič


RP2040
IOs
Crystal
SWD
GPIO
[29:0]
QSPI
Clock
generation
Internal
oscillator
PLL
PLL
Interrupts
Proc0
Proc1
SIO
DMA
Peripherals
SPI ×2
PWM
UART ×2
Timer
RTC
I2C ×2
ADC & TS
Reset control
Power on state
machine
Sysctrl
Sysinfo
Watchdog
Bus Fabric
PIO0
PIO1
PIO
XIP /
Cache
ROM
SRAM
SRAM
SRAM
SRAM
SRAM
SRAM
Memory
USB
Core Supply Regulator

ARM Cortex-M0+

- Komunikační rozhraní jádra
  - Externní AHB-Lite interface -> busfabric
  - Debug Access Port (DAP)
  - Single-cycle I/O Port -> SIO periferie
- Konfigurace jádra
  - 32bitový, Little Endian, 8 MPU (Memory Protection Unit)
  - Podpora ladění (2 drátové rozhraní SWD)
  - 26 ext. přerušení, 34 WIC (Wake-up Interrupt Controller)
  - Všechny registry resetovány po restartu

ARM Cortex-M0+ architektura



Zdroje hodinového kmitočtu

clk_ref

- Interní Ring Oscilator (ROSC), může být přepnutý na externí krystalový (XOSC)
- 6-12MHz

clk_sys

- Během reset napájen z clk_ref, pak většinou přepnutý na PLL
- 125MHz

clk_peri

- Typicky napájen z clk_sys, ale umožňuje periferiím být nezávislé (clk_sys je možné softwarově změnit)
- 12-125MHz

Zdroje hodinového kmitočtu

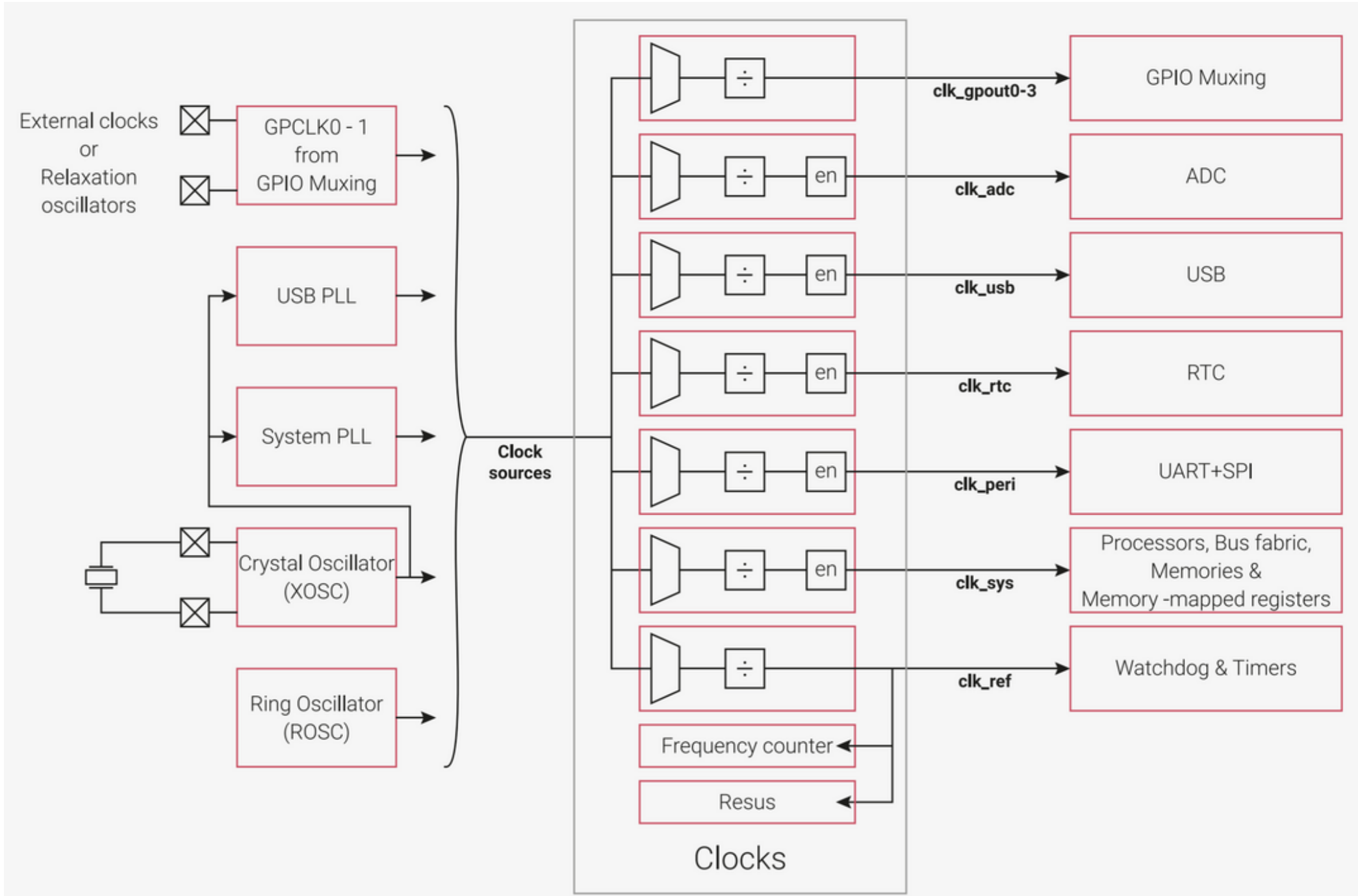
clk_usb, clk_adc

- Referenční hodiny pro USB a ADC
- 48MHz

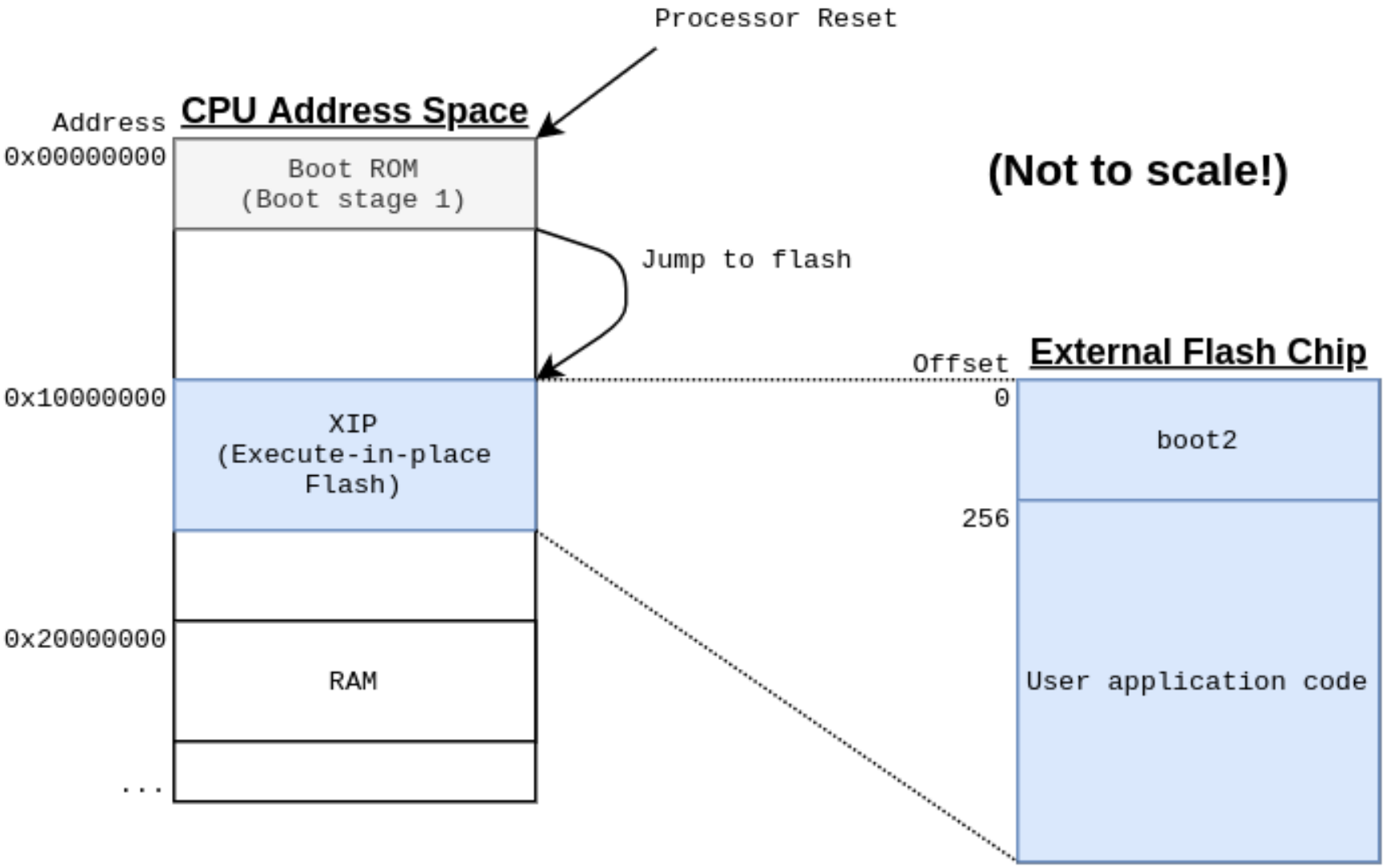
clk_rtc

- RTC dělí knitočet pro získání 1s reference
- 46875Hz

Další podrobnosti v dokumentaci.

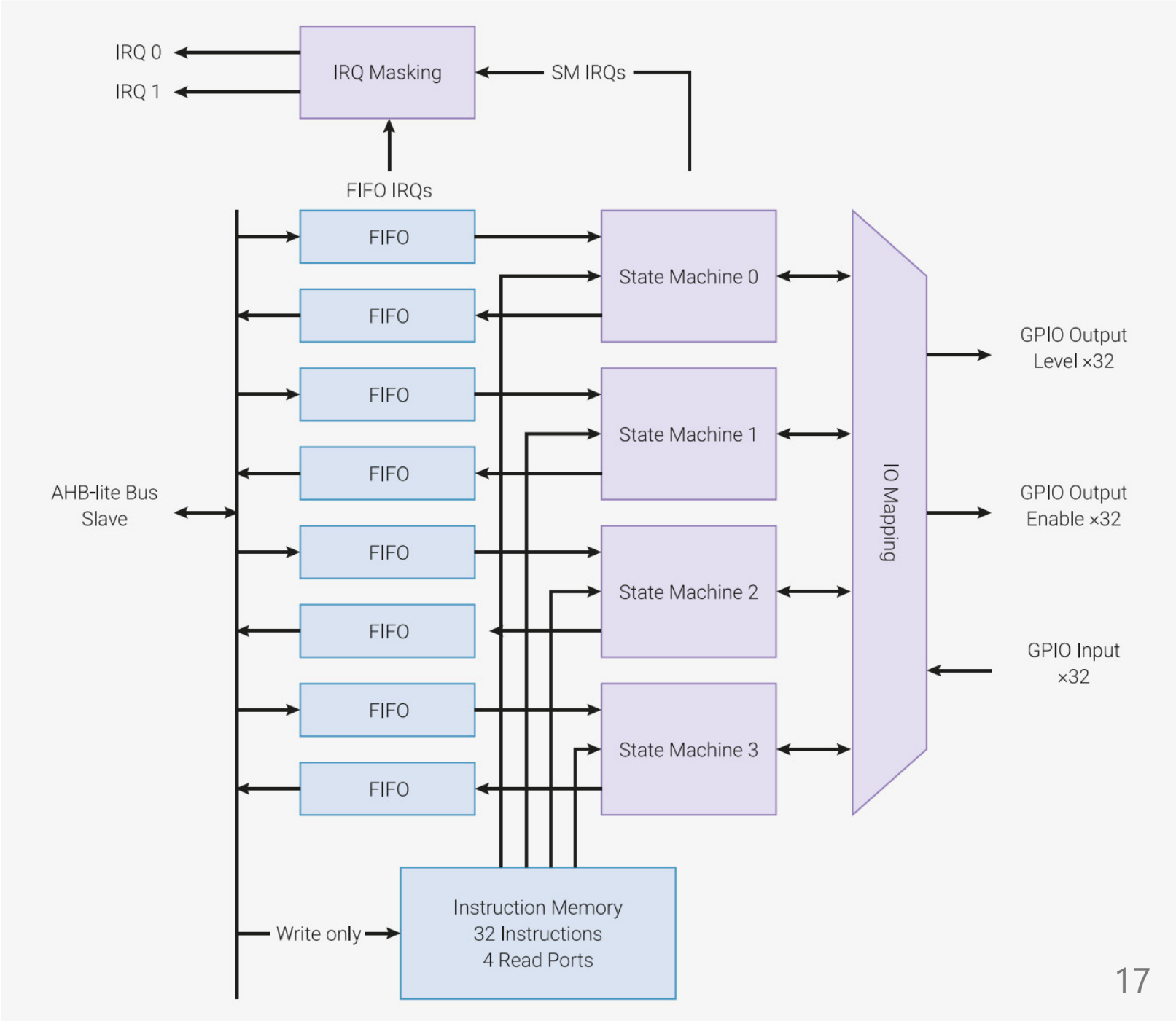

External clocks
or
Relaxation
oscillators
GPCLK0 - 1
from
GPIO Muxing
USB PLL
System PLL
Crystal Oscillator
(XOSC)
Ring Oscillator
(ROSC)
Clock
sources
÷
en
clk_gpout0-3
clk_adc
clk_usb
clk_rtc
clk_peri
clk_sys
clk_ref
Frequency counter
Resus
Clocks
GPIO Muxing
ADC
USB
RTC
UART+SPI
Processors, Bus fabric,
Memories &
Memory -mapped registers
Watchdog & Timers

Paměť


Processor Reset
Address
CPU Address Space
0x00000000
Boot ROM
(Boot stage 1)
(Not to scale!)
Jump to flash
Offset
External Flash Chip
0x10000000
XIP
(Execute-in-place
Flash)
0
boot2
256
0x20000000
RAM
User application code
...

PIO

- Implementuje řadu protokolů na HW úrovni
- Může pracovat nezávisle na CPU
- Každý RP2040 má dvě PIO instance

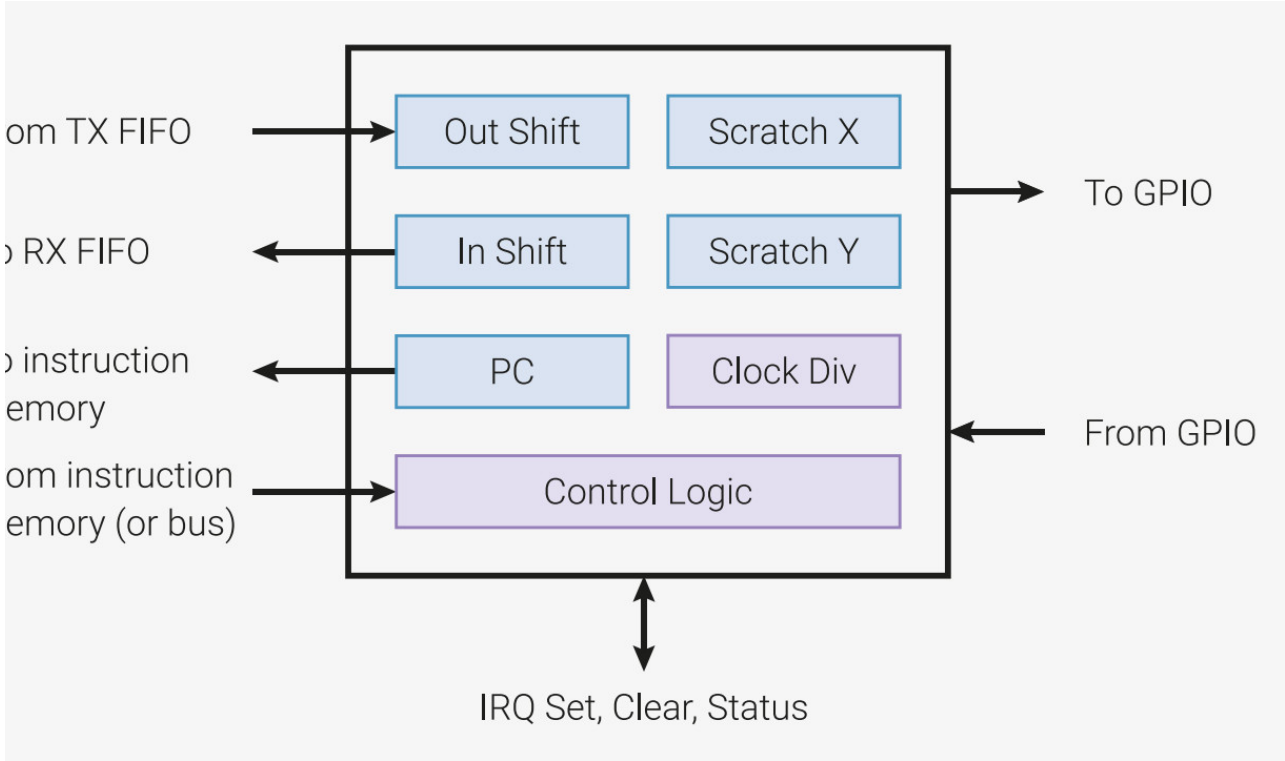


PIO State machine

- Každý PIO má 4 **stavové stroje**. Tyto stavové stroje fungují jako malé, velmi omezené procesory schopné spouštět instrukce, které se nacházejí ve sdílené paměti instrukcí.
- Paměť instrukcí může pojmout celkem až **32 instrukcí**. Každý stavový stroj však může čerpat instrukce odkudkoli z této paměti.
- Můžete například vytvořit 4 samostatné programy, každý s 8 instrukcemi, které se spustí samostatně na stavových strojích. Případně můžete mít jeden program s 16 instrukcemi, který poběží na všech 4 stavových strojích (a zbylých 16 míst pro instrukce nebude dělat nic).

PIO State machine

- Každý stavový stroj má přístup ke 2 FIFO, které můžete použít k odesílání a přijímání dat do/z hlavního procesoru
- Ve výchozím nastavení je jedna FIFO určena pro odchozí data a druhá pro data příchozí.
- Lze je však nastavit tak, aby se obě používaly pro odchozí nebo obě pro příchozí data, pokud by bylo třeba zdvojnásobit velikost vyrovnávací paměti.



PIO State machine

- Kromě toho má každý stavový stroj přístup ke sdílené bance 8 příznaků přerušení. Ty mají různé využití, například pro synchronizaci stavových strojů nebo upozornění procesoru, že jsou některá data připravena ke spotřebování.
- A konečně, každý stavový stroj může ovládat kterýkoli z 32 pinů GPIO RP2040. Aby však stavový stroj mohl ovládat skupinu pinů, musí být tyto piny seskupeny do souvislé sady.
- Každý stavový stroj (nezapomeňte: v RP2040 je celkem 8 stavových strojů - 4 v každém PIO) se skládá ze sady registrů a určité řídicí logiky.

Micropython

- https://docs.micropython.org/en/latest/rp2/general.html
- https://docs.micropython.org/en/latest/rp2/tutorial/intro.html

Obecná kontrola RPi

Modul machine

- obecné vrstva HW abstrakce

```
import machine

machine.freq()          # get the current frequency of the CPU
machine.freq(240000000) # set the CPU frequency to 240 MHz
```

Modul rp2

- funkcionalita specifická pro RP2040

```
import rp2
```

Modul machine

- Modul obsahuje specifické funkce související s hardwarem na konkrétní desce.
- Většina funkcí modulu umožňuje přímý a neomezený přístup k hardwarovým blokům systému (jako je procesor, časovače, sběrnice atd.) a jejich ovládání.
- Při nesprávném použití může dojít k poruše, zablokování, v krajním případě i k poškození hardwaru.
- Na vhodném hardwaru nabízí MicroPython možnost psát obsluhy přerušení v jazyce Python. Obsluhy přerušení - známé také jako rutiny obsluhy přerušení (ISR) - jsou definovány jako callback funkce. Ty se provádějí v reakci na událost, jako je spuštění časovače nebo změna napětí na pinu.

Přístup do paměti

- Modul definuje tři objekty pro přímý přístup do paměti

machine.mem8

- Zápis/čtení 8 bitů paměti

machine.mem16

- Zápis/čtení 16 bitů paměti

machine.mem32

- Zápis/čtení 32 bitů paměti

Příklad přístup do paměti

- Příklad specifický pro platformu STM32

```
import machine
from micropython import const

GPIOA = const(0x48000000)
GPIO_BSRR = const(0x18)
GPIO_IDR = const(0x10)

# set PA2 high
machine.mem32[GPIOA + GPIO_BSRR] = 1 << 2

# read PA3
value = (machine.mem32[GPIOA + GPIO_IDR] >> 3) & 1
```

Reset zařízení 1/2

machine.reset()

- Reset zařízení se stejným efektem jako externím Reset signálem

machine.soft_reset()

- Soft reset interpretu, odstraní všechny objekty Pythonu a resetuje haldu Pythonu.
- Pokusí se zachovat způsob, kterým je uživatel připojen k MicroPython REPL (např. sériový, USB, Wifi).

machine.reset_cause()

- Vrátí příčinu resetu
- Příčinu popisují konstanty

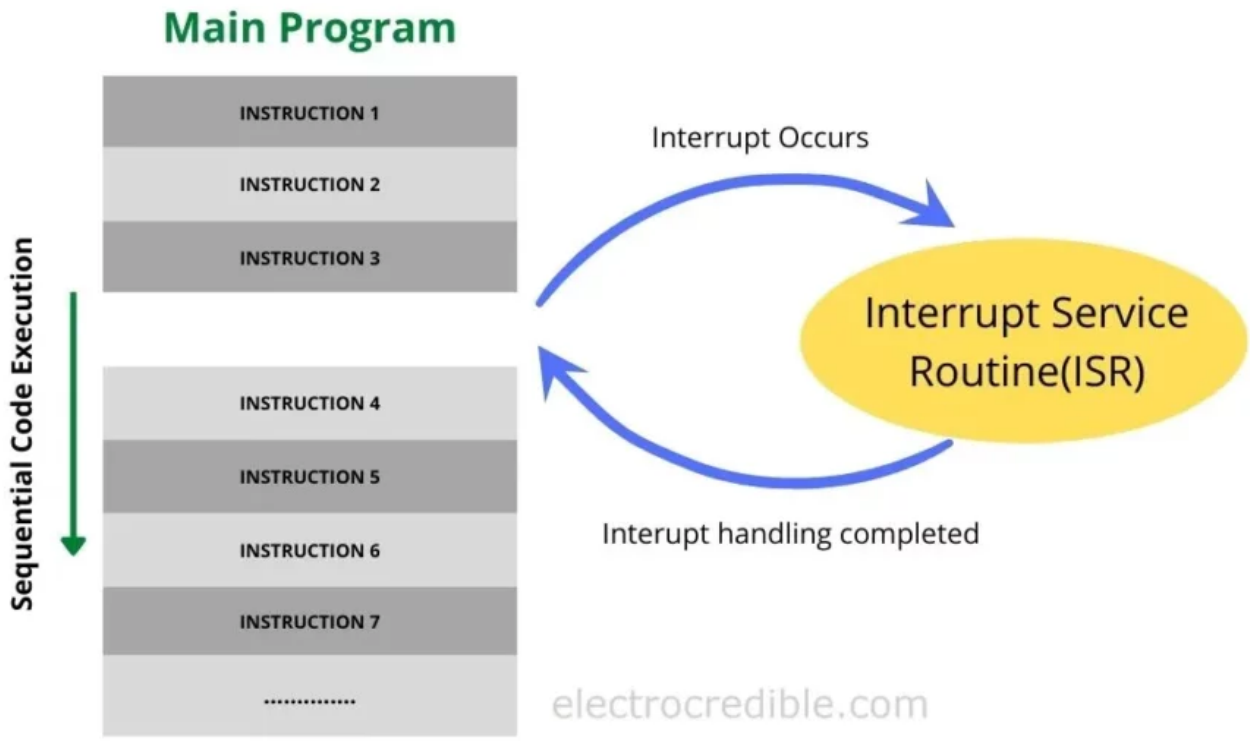
Reset zařízení 2/2

machine.bootloader([value])

- Resetuje zařízení a vstoupí do jeho zavaděče.
- Obvykle se používá k uvedení zařízení do stavu, kdy je možné naprogramovat nový firmware.
- Některé porty podporují předání volitelného argumentu value, který může řídit, do kterého zavaděče se má vstoupit nebo co se mu má předat.

Přerušení

- Přerušení zpracovávají části softwaru nazývané Interrupt Service Routine(ISR).
- Když dojde k přerušení, procesor začne provádět kód v rámci této rutiny.
- Po dokončení úkolu v rutině pokračuje procesor ve vykonávání kódu od místa, kde skončil.



Přerušení

- přerušení lze zakázat (vypnout) a opětovně povolit.
- Některé subsystémy vyžadují přerušení ke správnému fungování, takže jejich dlouhodobé vypnutí může ohrozit funkčnost jádra (např. watchdog).
- Přerušení by měla být zakázána pouze na minimální dobu a poté opět povolena do předchozího stavu.

```
import machine

# Disable interrupts
state = machine.disable_irq()

# Do a small amount of time-critical work here

# Enable interrupts
machine.enable_irq(state)
```

Přerušení

machine.disable_irq()

- Zakáže požadavky na přerušení.
- Vrací předchozí stav IRQ, který by měl být považován za neprůhlednou hodnotu.
- Tato návratová hodnota by měla být předána funkci enable_irq() pro obnovení přerušení do původního stavu před voláním disable_irq().

machine.enable_irq(state)

- Znovu povolí požadavky na přerušení.
- Parametrem state by měla být hodnota, která byla vrácena z posledního volání funkce disable_irq().

Napájení

machine.freq([Hz])

- Vrací frekvenci procesoru v Hz. Na některých portech lze tuto funkci použít také k nastavení frekvence procesoru zadáním Hz.

machine.idle()

- Zastaví takt procesoru, což je užitečné pro snížení spotřeby energie kdykoli během krátkých nebo dlouhých období.
- Periferie pokračují v práci a provádění se obnoví, jakmile je vyvoláno jakékoli přerušení (na mnoha portech to zahrnuje přerušení systémového časovače, ke kterému dochází v pravidelných intervalech v řádu milisekund).

Napájení

machine.lightsleep([time_ms])

machine.deepsleep([time_ms])

- Zastaví provádění programu a pokusí se o přechod do stavu nízké spotřeby.
- Pokud je zadáno **time_ms**, bude to maximální doba v milisekundách, po kterou bude uspání trvat. Jinak může uspání trvat neomezeně dlouho.
- S časovým limitem nebo bez něj se může provádění programu kdykoli obnovit, pokud dojde k událostem, které vyžadují zpracování. Takové události nebo zdroje probuzení by měly být nakonfigurovány před uspáním, jako je změna pinu nebo časový limit RTC.

Napájení 3/4

Přesné chování a možnosti úspory energie u režimů **lightsleep** a **deepsleep** jsou velmi závislé na základním hardwaru, ale obecné vlastnosti jsou následující:

- **lightsleep** zachovává RAM a stav.
    - Po probuzení se pokračuje v provádění od bodu, kde byl požadován spánek, všechny subsystémy jsou funkční.
- **deepsleep** nesmí zachovat paměť RAM ani žádný jiný stav systému (např. periferie nebo síťová rozhraní).
    - Po probuzení je provádění obnoveno z hlavního skriptu, podobně jako při tvrdém nebo zapnutém resetu.
    - Funkce reset_cause() vrátí hodnotu machine.DEEPSLEEP, kterou lze použít k rozlišení probuzení v hlubokém spánku od jiných resetů.

Napájení 4/4

machine.wake_reason()

- Vrací důvod probuzení ze spánku
- Zdroj popisují konstanty

Další funkce

Další užitečné funkce jsou shrnuty na této stránce.

Časovač

- Periferie systémového časovače RP2040 poskytuje globální mikrosekundovou časovou základnu a generuje pro ni přerušení.
- Současně je k dispozici softwarový časovač, kterých je k dispozici neomezený počet (pokud to paměť dovolí).
- Časovač je popsán třídou machine.Timer

```
from machine import Timer

tim = Timer(period=5000, mode=Timer.ONE_SHOT, callback=lambda t:print(1))
tim.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(2))
```

GPIO

- GPIO popisuje třída machine.Pin

```
from machine import Pin

p0 = Pin(0, Pin.OUT)    # create output pin on GPIO0
p0.on()                 # set pin to "on" (high) level
p0.off()                # set pin to "off" (low) level
p0.value(1)             # set pin to on/high

p2 = Pin(2, Pin.IN)     # create input pin on GPIO2
print(p2.value())       # get value, 0 or 1

p4 = Pin(4, Pin.IN, Pin.PULL_UP) # enable internal pull-up resistor
p5 = Pin(5, Pin.OUT, value=1) # set pin high on creation
```

GPIO s přerušením

```
import time
from machine import Pin

pin_button = Pin(14, mode=Pin.IN, pull=Pin.PULL_UP)
pin_led    = Pin(16, mode=Pin.OUT)


def button_isr(pin):
  pin_led.value(not pin_led.value())

pin_button.irq(trigger=Pin.IRQ_FALLING, handler=button_isr)

while True:
  ...
```

ADC

- ADC je popsán třídou machine.ADC

```
from machine import ADC
import utime

sensor_temp = ADC(4)
conversion_factor = 3.3 / (65535)

while True:
    reading = sensor_temp.read_u16() * conversion_factor
    temperature = 27 - (reading - 0.706)/0.001721
    print(temperature)
    utime.sleep(2)
```

UART

- UART je popsán třídou machine.UART
- RP2040 má dvě UART periférie (UART0 a UART1)
  - Programovatelná délka dat (5-8 bitů) a počtu stop bitů (1 nebo 2)
  - FIFO v obou směrech až 32 bytů
- Přerušením lze sledovat příchod nebo odchod dat, status zařízení, chybu komunikace nebo timeout příjmu dat
- Obě zařízení lze konfigurovat na různých dvojicích TX a RX pinů
  - UART0: GP0-GP1, GP12-GP13, GP16-GP17
  - UART1: GP4-GP5, GP8-GP9

UART

```
from machine import Pin, UART
import time

uart = UART(1, baudrate=9600, tx=Pin(4), rx=Pin(5))
uart.init(bits=8, parity=None, stop=2)

led = Pin("LED", Pin.OUT)

while True:
    uart.write('t')
    if uart.any():
        data = uart.read()
        if data== b'm':
            led.toggle()
    time.sleep(1)
```

PIO

- PIO MCU RP2040 je požné i v rámci Micropythonu programovat na nižší úrovni
- PIO assembler: JMP, WAIT, IN, OUT, PUSH, PULL, MOV, IRQ, a SET
  - instrukce jsou zaměřené na manipulaci s bity
  - trvají právě jeden strojový cyklus
  - neimplementují žádné aritmetické operace
  - jediná logická operace je přesun s bitovým doplňkem
- Modul rp2 implementuje wrapper pro instrukce v assembleru
  - Např. funkce **set()** vytváří wrapper pro instrukcí **SET**, která přepíná stav GPIO pinu nezávisle na hlavním procesoru.
- Příklady na gihub.com

PIO

- Instrukce jsou vysoce optimalizovány na následující operace
  - pro serializaci a deserializaci proudů bitů do/ze slov,
  - posun slov do FIFO a vybírání slov z FIFO.
- Kromě lze vložit instrukce z externích zdrojů k provedení v libovolném hodinovém cyklu.
- Každá instrukce má navíc vyhrazeno několik bitů pro postranní nastavení (side-set) a/nebo zpoždění instrukce.
  - Side-set umožňuje další přímou manipulaci s bity na pinech GPIO a provádí se paralelně a nezávisle na vlastní instrukci.

Příklad - generování obdélníkového signálu

- Cílem je vygenerovat signál na pinu 28
- Stavový stroj pracuje na frekvenci **clk_sys** - 125 MHz
  - Každá instrukce trvá právě jeden hodinový takt, lze prodloužit na 32 (1+31)
  - 32 cyklů trvá 256 ns -> frekvence signálu bude 1.953 MHz

```
import time
import rp2
from machine import Pin
@rp2.asm_pio(set_init=rp2.PIO.OUT_LOW)
def blink():
    wrap_target()
    set(pins, 1)   [31]
    set(pins, 0)   [31]
    wrap()
sm = rp2.StateMachine(0, blink, set_base=Pin(28))
sm.active(1)
```

Příklad - generování obdélníkového signálu

- Program pro PIO je implementován ve funkci **blink()**
- Pro použití s PIO je třeba použít dekorátor **@rp2.asm_pio**
  - Parametry dekorátoru jsou např. nastavení pinu a FIFO
- Funkce **wrap()** a **wrap_target()** označující začátek kódu pro state machine, umožňují realizovat automaticky cyklus bez použití instrukce **JMP**
- Frekvenci signálu lze měnit
  - změnou délky trvání instrukce
  - změnou frekcence stavového stroje

```
sm = rp2.StateMachine(0, blink, freq=2500,set_base=Pin(28))
```