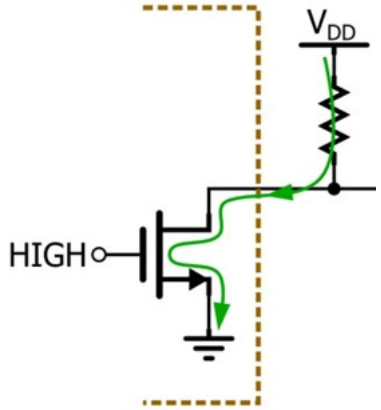
Systemy wbudowane dla automatyki W05

dr inż. Krzysztof Urbański

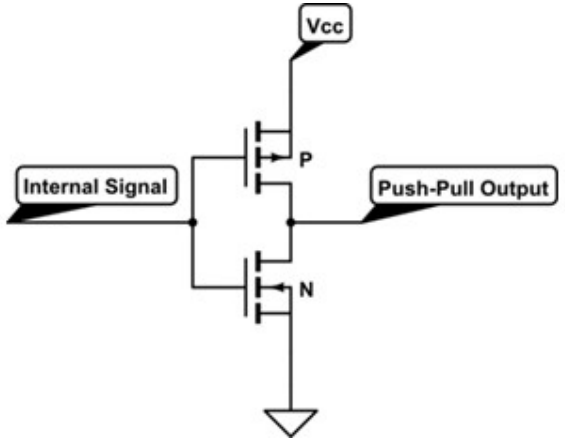
Dodatkowe materiały do kursu

Źródła: materiały własne, publicznie dostępne dokumenty w dostępie otwartym oraz noty katalogowe i oficjalna dokumentacja prezentowanych rozwiązań.

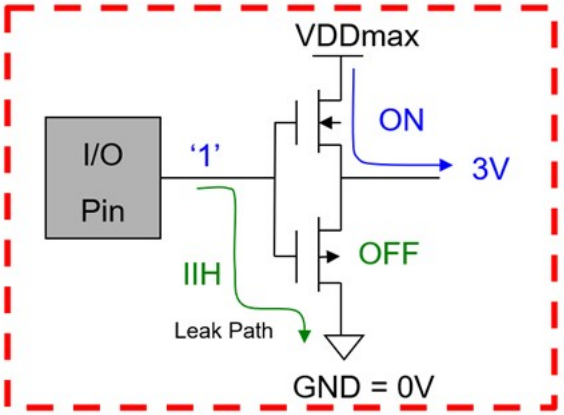
Komentarze do quizu



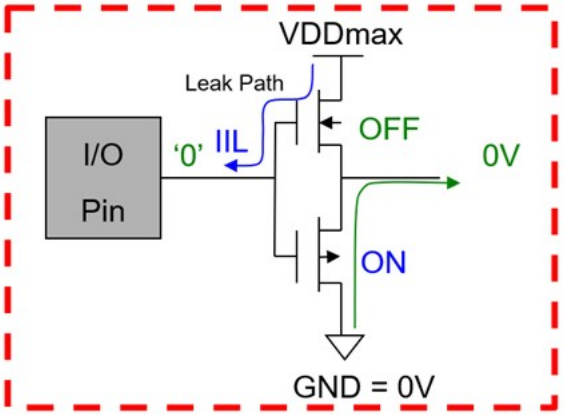
Wyjście typu open drain.



Wyjście push-pull



wejście, stan ‘1’



wejście, stan ‘0’

Walka o energię: fizyka, podzespoły, oprogramowanie

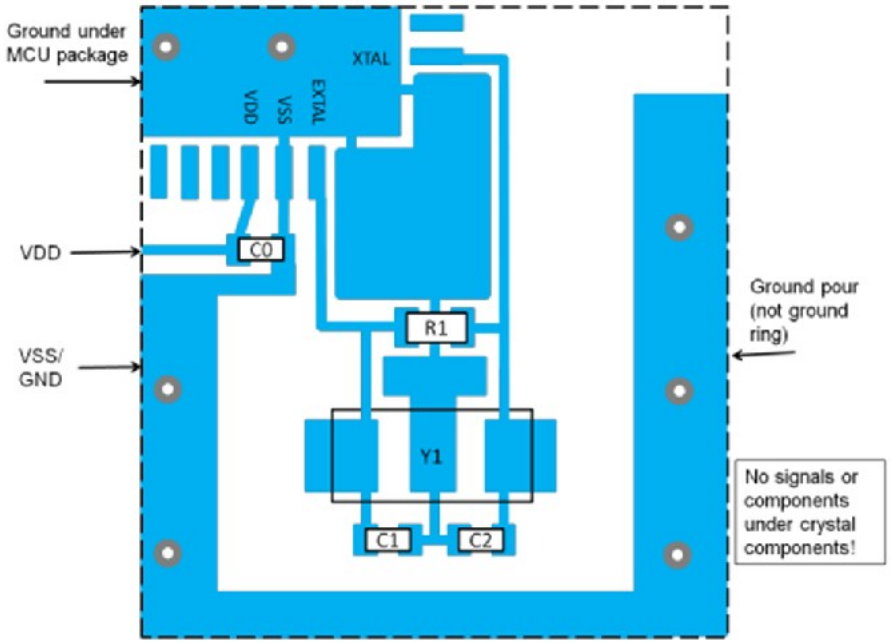


Figure 2-3. Crystal layout for low power oscillator

Rodzina	Tryb	Pobór prądu	Czas
Kinetis L	WAIT VLPW STOP VLPS VLLS3 VLLS1 VLLS0	3.18 mA @ 48 MHz (66 µA/MHz) 93.4 µA @ 4 MHz (23.4 µA/MHz) 255 µA 1.86 µA 1.24 µA 0.6 µA 160 nA	0 0 4.5 µs 4.5 µs 47 µs 105 µs 106 µs
LPC800	Sleep mode Deep-sleep mode Power-down mode Deep power-down mode with wake-up timer without wake-up timer	1.8 mA @ 30 MHz (60 µA/MHz) 150 µA 0.9 µA 1 µA 170 nA	2.6 µs 4 µs 50 µs brak danych 215 µs
SAM D20	IDLE0 IDLE1 IDLE2 STANDBY	2.35 mA @ 48 MHz (49 µA/MHz) 1.9 mA 1.7 mA 2 µA	3.5 µs 12 µs 12.5 µs 20 µs
EFM32 Zero Gecko	EM1 Sleep Mode EM2 Deep Sleep EM3 Stop Mode EM4 Shutoff Mode	1.15 mA @ 24 MHz (48 µA/MHz) 0.9 µA 0.5 µA 20 nA	0 2 µs 2 µs 160 µs

Projekt PCB: XTAL i konfiguracja PLL krytycznie ważne
W RP2350 dojdą dodatkowo komponenty przetwornicy DC-DC.

Rys. po prawej: Kilka przykładów trybów pracy energooszczędnych MCU, typowe wartości prądów i czasy wybudzenia.

RP2350: *low-power states* (dla porównania)

Low Power State	VREG_VIN (µA)	VREG_AVDD (µA)	IOVDD (µA)	QSPI_IOVDD (µA)	ADC_IOVDD (µA)	USB_OTP_VDD (µA)	Total Power (µW)
P1.0	128	0.5	11	22	1	3.5	548
P1.1	77	0.5	11	22	1	3.5	380
P1.2	79	0.5	11	22	1	3.5	380
P1.3	26	0.5	11	22	1	3.5	204
P1.4	120	0.5	11	22	1	3.5	520
P1.5	67	0.5	11	22	1	3.5	345
P1.6	68	0.5	11	22	1	3.5	350
P1.7	19	0.5	11	22	1	3.5	188
RUN=low	40	187	69	22	1	35	1170

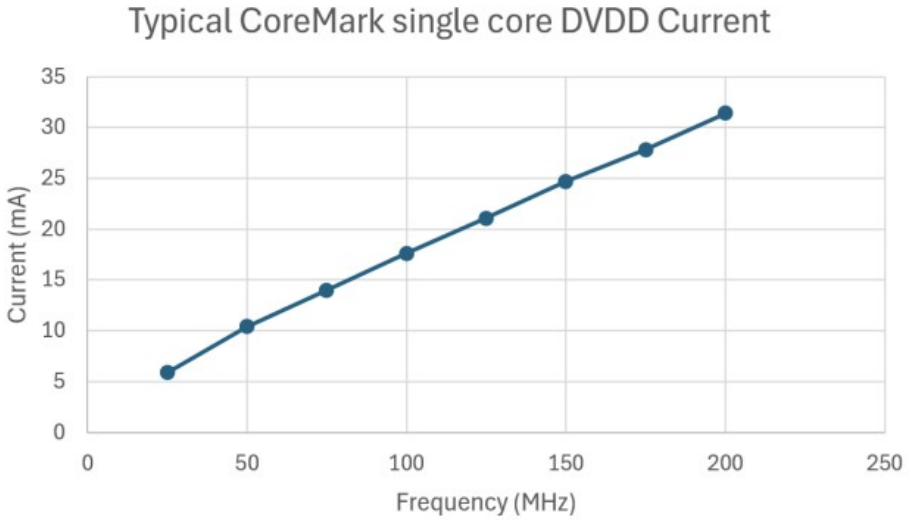
→Patrz dokumentacja RP2350.

Wniosek: to nie jest MCU który może pracować 2 lata zasilany z niewielkiej baterii.

Zadanie: RP2350 jest zasilany z baterii 3.0V, 50 mAh. MCU działa w trybie P1.7, ale co minutę wybudza się na 5 sekund do trybu RUN (PLL=50MHz). Po jakim czasie trzeba będzie wymienić baterię?

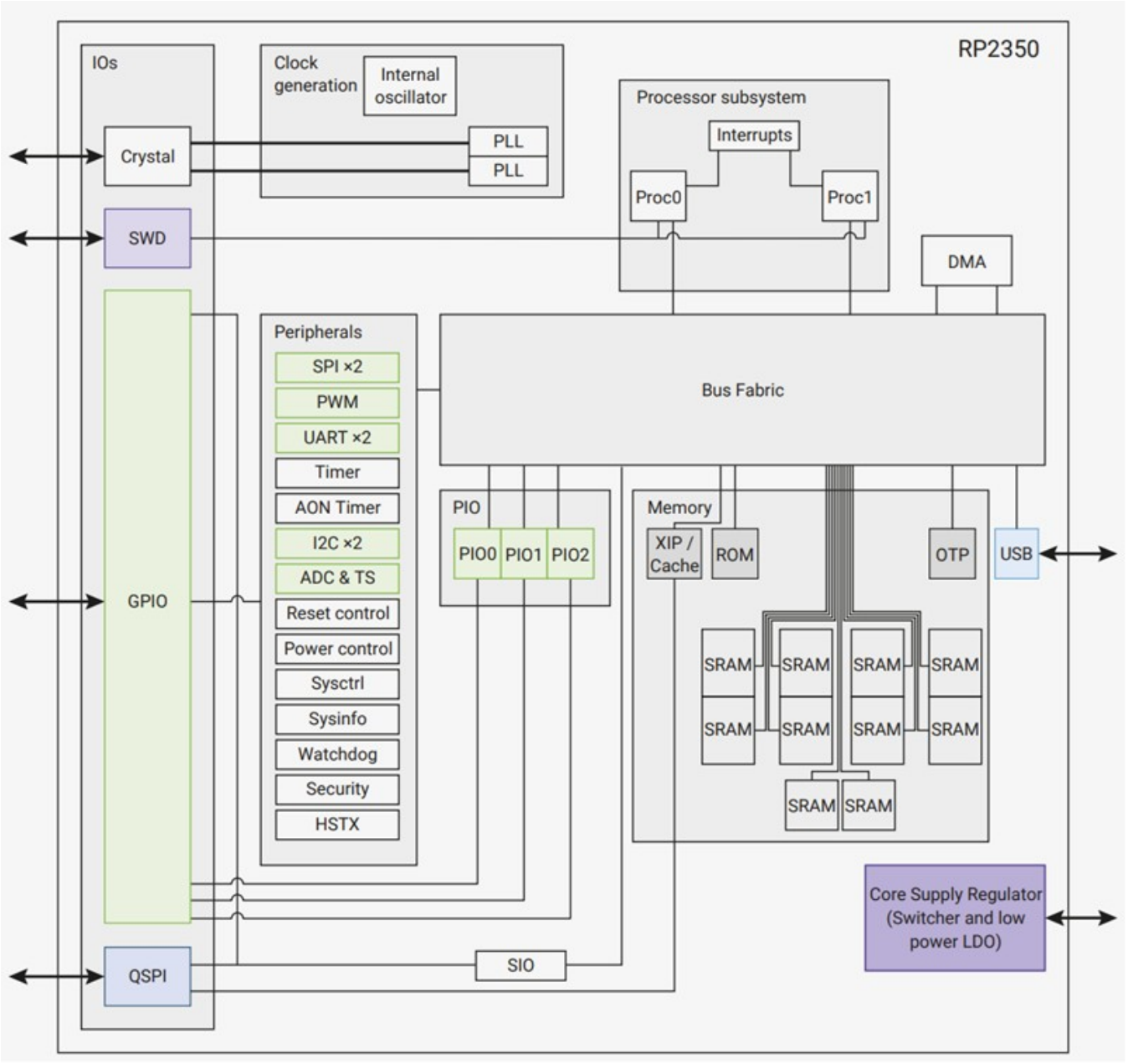
RP2350: energia - typowe scenariusze użycia

Use Case	Condition	VREG_VIN (µA)	VREG_AVDD (µA)	IOVDD (µA)	QSPI_IOVDD (µA	ADC_IOVDD (µA)	USB_OTP_VDD (µA)	Total Power (mW)
USB Boot mode	Bus Idle (average)	6530	220	437	22	1	375	25
	During Boot (peak)						6050	
	During UF2 write (average)						1280	
hello_serial		14690	216	506	22	1	62	51.1
hello_usb		14700	216	453	22	1	570	52.7
hello_adc		14680	216	506	22	142	62	51.6
CoreMark benchmark	Single core @150MHz	11000	212	455	22	1	90	38.7



Częściej będzie to praca na akumulatorze, wtedy jak najbardziej warto zacząć od kontroli częstotliwości pracy rdzenia →todo, laborka zegarowa

RP2350: schemat blokowy

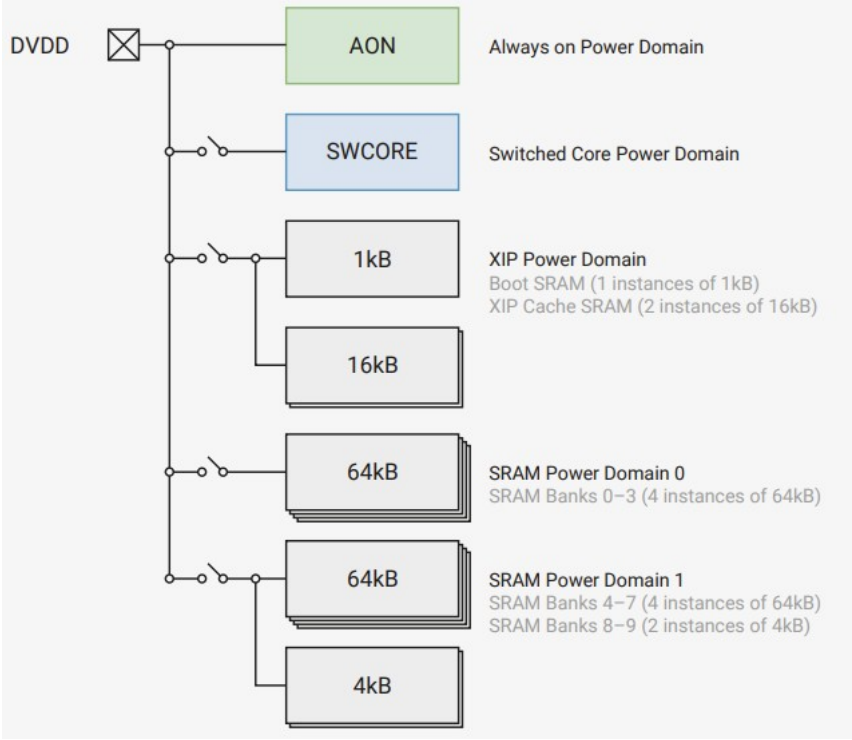


RP2350: źródła RESET

Reset Source	SW-DP	AON Scratch	POWMAN	Power State	Double Tap	Rescue
POR	reset	reset	hard reset	→ P0.0	reset	reset
BOR	reset	reset	hard reset	→ P0.0	reset	reset
EXTERNAL RESET (RUN)	reset	reset	hard reset	→ P0.0	—	reset
DEBUGGER RESET REQ	—	—	hard reset	→ P0.0	—	reset
DEBUGGER RESCUE	—	—	hard reset	→ P0.0	—	set
WATCHDOG POWMAN ASYNC RESET	—	—	hard reset	→ P0.0	—	—
WATCHDOG POWMAN RESET	—	—	soft reset	→ P0.0	—	—
WATCHDOG SWCORE RESET	—	—	—	→ P0.0	—	—
SWCORE POWERDOWN	—	—	—	→ P0.x	—	—
GLITCH_DETECTOR	—	—	—	—	—	—
WATCHDOG RESET PSM	—	—	—	—	—	—

Laborka 3.1 poszła nieźle, 3.2 przetarte szlaki (i wyżej podniesioną poprzeczkę).
Wskazówka: Watchdog jako programowy reset.
Jest kilka innych sposobów, ale niektóre internetowe propozycje zawierają poważne błędy (stany początkowe rejestrów, jest więcej niż 1 rdzeń – a wszystkie trzeba poprawnie zresetować!). Jest też inny powód dla którego warto wybrać WD zamiast np. kombinować z resetowaniem poprzez fizyczny pin RUN, ma to związek z oszczędzaniem energii.

RP2350: domeny zasilania



Power State	Description	AON	SWCORE	XIP	SRAM0	SRAM1
P0.0	Normal Operation	**on**	**on**	**on**	**on**	**on**
P0.1	Normal Operation (SRAM1 off)	**on**	**on**	**on**	**on**	*off*
P0.2	Normal Operation (SRAM0 off)	**on**	**on**	**on**	*off*	**on**
P0.3	Normal Operation (SRAM0 & SRAM1 off)	**on**	**on**	**on**	*off*	*off*
P1.0	Low Power	**on**	*off*	**on**	**on**	**on**
P1.1	Low Power (SRAM1 off)	**on**	*off*	**on**	**on**	*off*
P1.2	Low Power (SRAM0 off)	**on**	*off*	**on**	*off*	**on**
P1.3	Low Power (SRAM0 & SRAM1 off)	**on**	*off*	**on**	*off*	*off*
P1.4	Low Power (XIP off)	**on**	*off*	*off*	**on**	**on**
P1.5	Low Power (XIP & SRAM1 off)	**on**	*off*	*off*	**on**	*off*
P1.6	Low Power (XIP & SRAM0 off)	**on**	*off*	*off*	*off*	**on**
P1.7	Low Power (XIP & SRAM0 & SRAM1 off)	**on**	*off*	*off*	*off*	*off*
OFF	Not Powered	*off*	*off*	*off*	*off*	*off*

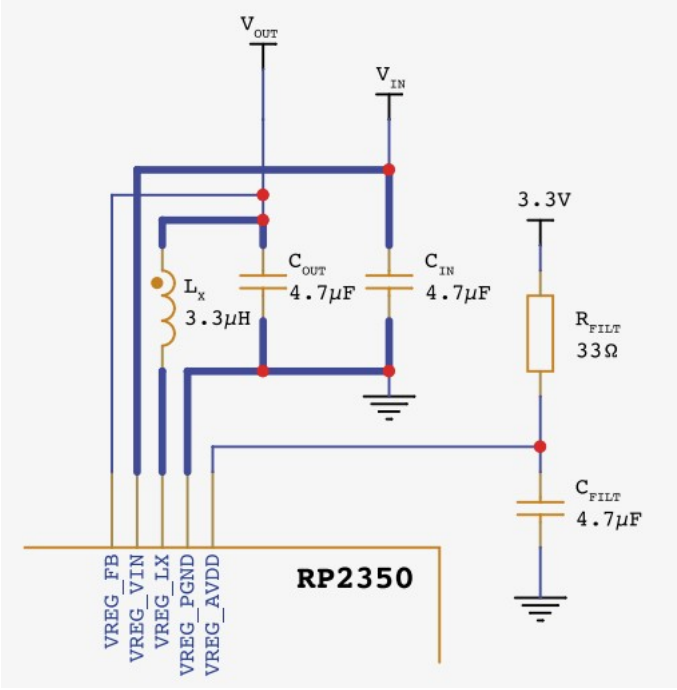
Wskazówka: zauważ AON z uzupełnionej instrukcji do Lab03. Przypadek? Nie sądzę.

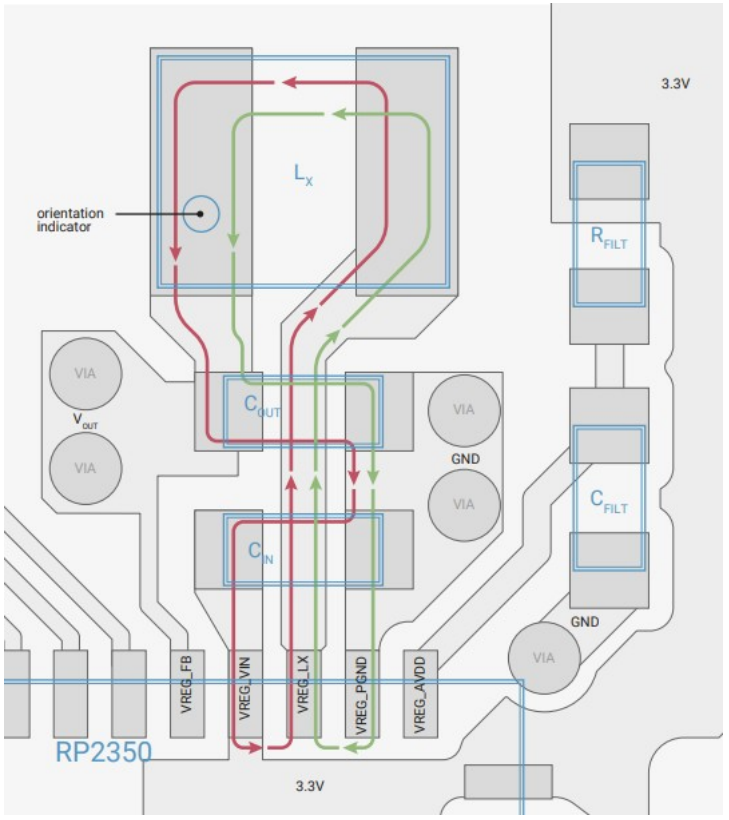
RP2350: Zasilanie rdzenia

Wewnętrzna przetwornica (**nie jest to LDO!**), z zaprojektowanym dedykowanym induktorem:

„...Raspberry Pi have worked with Abracon to create a custom 2.0×1.6mm 3.3µH polarity marked inductor, part number AOTA-B201610S3R3-101-T”

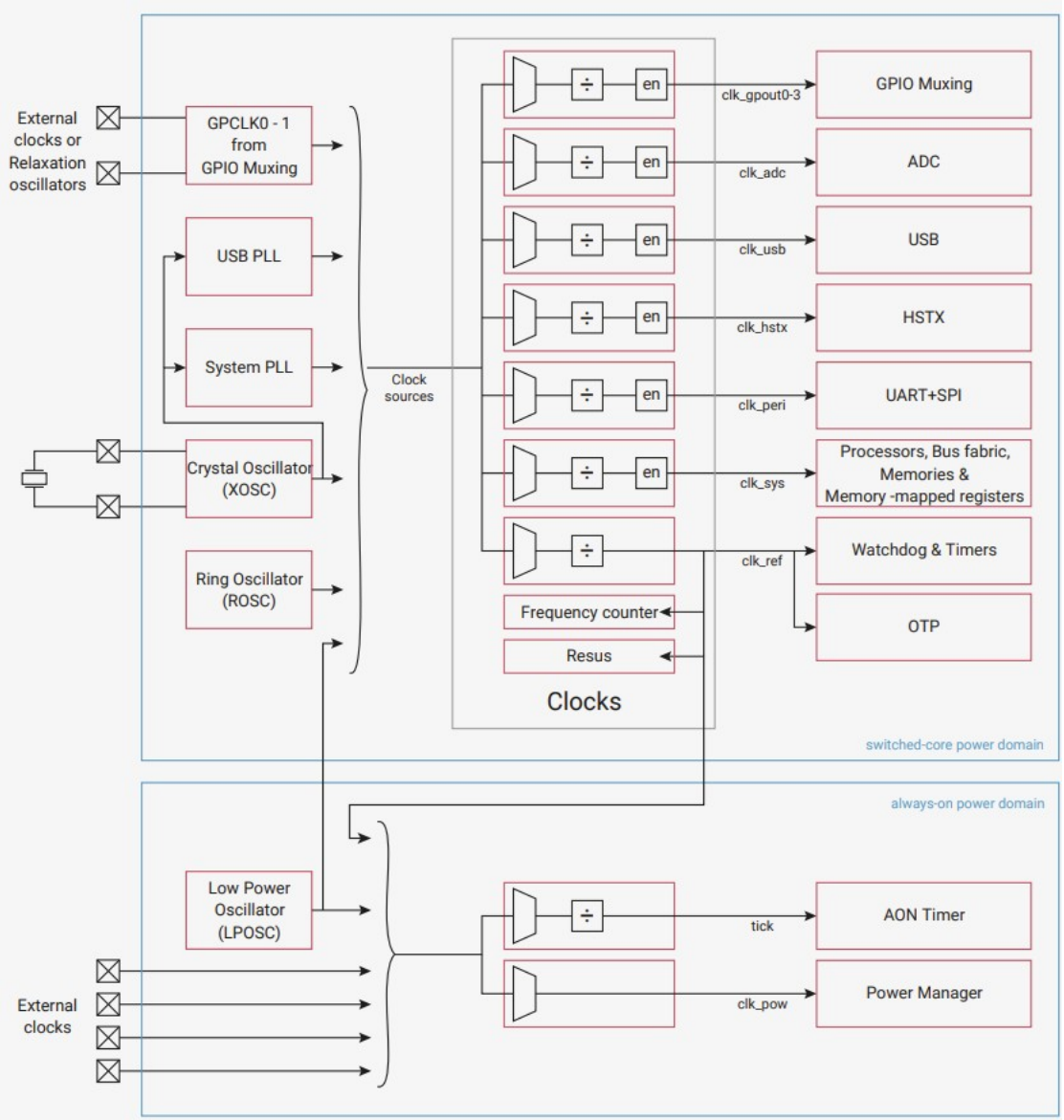
Napięcie rdzenia: 1.1 V → mniejszy dynamiczny pobór prądu.





RP2350: zegary

Do wyboru: XCLK, PLL, XOSC, ROSC (4..20 MHz), LPOSC 32 kHz



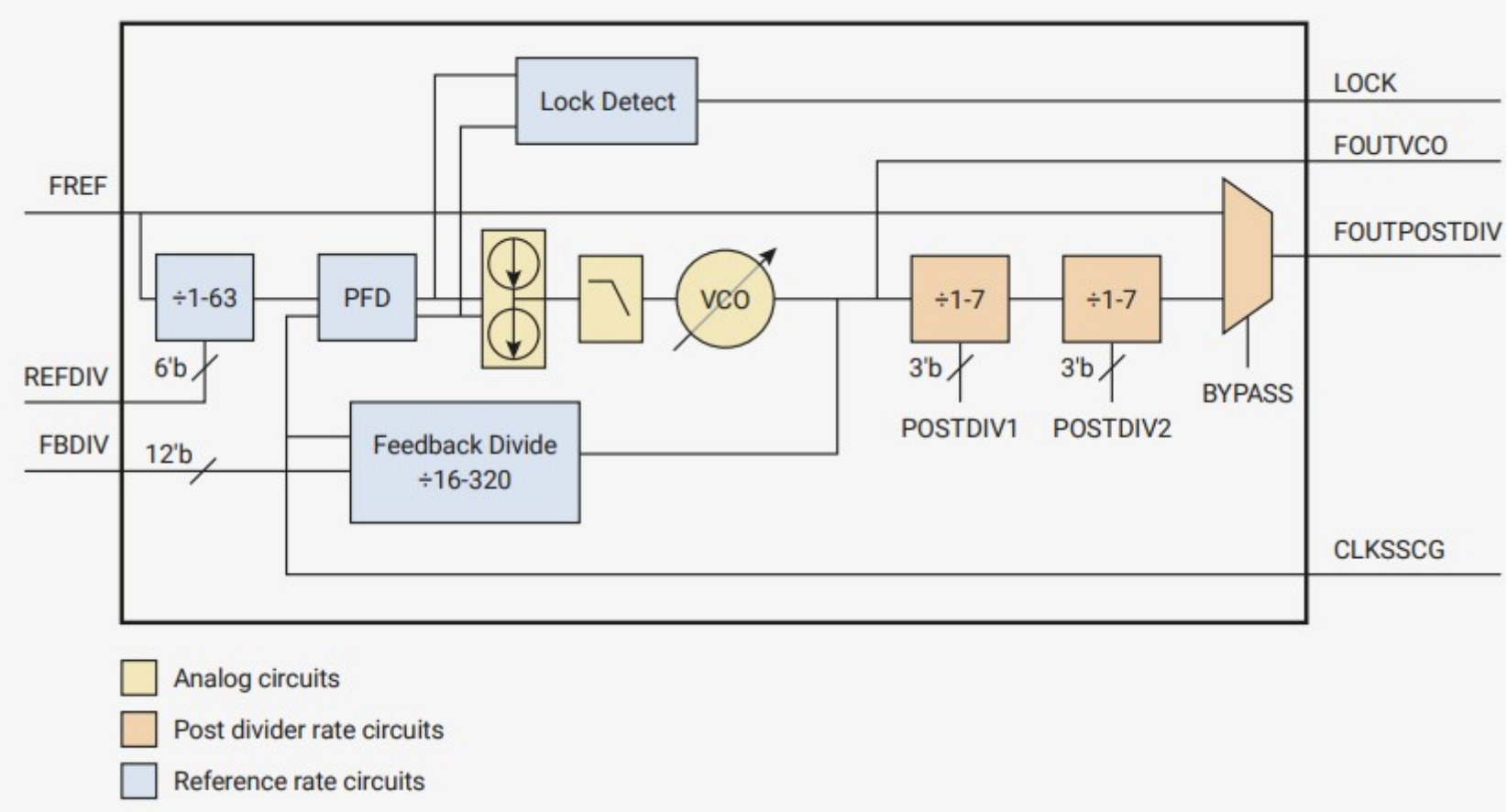
Jak tego użyć w realnym projekcie?

```
#include "hardware/clocks.h"
#include "hardware/pll.h"

//sprawdzanie na czym stoimy:
uint f_pll_sys = frequency_count_khz(CLOCKS_FC0_SRC_VALUE_PLL_SYS_CLKSRC_PRIMARY);
uint f_rosc = frequency_count_khz(CLOCKS_FC0_SRC_VALUE_ROSC_CLKSRC);
uint f_clk_peri = frequency_count_khz(CLOCKS_FC0_SRC_VALUE_CLK_PERI);
//...oraz kilka innych → patrz przykłady lub nagłówki

pll_deinit(pll_sys);
```

Przykład: **50 MHz**

```
// switch system clock to reference (XOSC)
clock_configure(clk_sys, CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLK_REF, 0, 12E6, 12E6);
pll_init(pll_sys, 1, 1500E6, 6, 5);
// configure sys pll to 50MHz
clock_configure(clk_sys,
    CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX,
    CLOCKS_CLK_SYS_CTRL_AUXSRC_VALUE_CLKSRC_PLL_SYS, 50E6, 50E6);

stdio_init_all(); //inaczej UART się zdziwi
```

RP2350: PLL



Do zadania na laborkę z zegarami:

- 1500 MHz VCO / 6 / 2 = 125 MHz
- 750 MHz VCO / 6 / 1 = 125 MHz
- Ta sama „legalna” częstotliwość może być uzyskana na kilka sposobów → jitter, zużycie energii
- Nie wszystkie kombinacje ustawień są dozwolone → w SDK jest skrypt vcocalc.py który to policzy.

https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/scripts/vcocalc.py

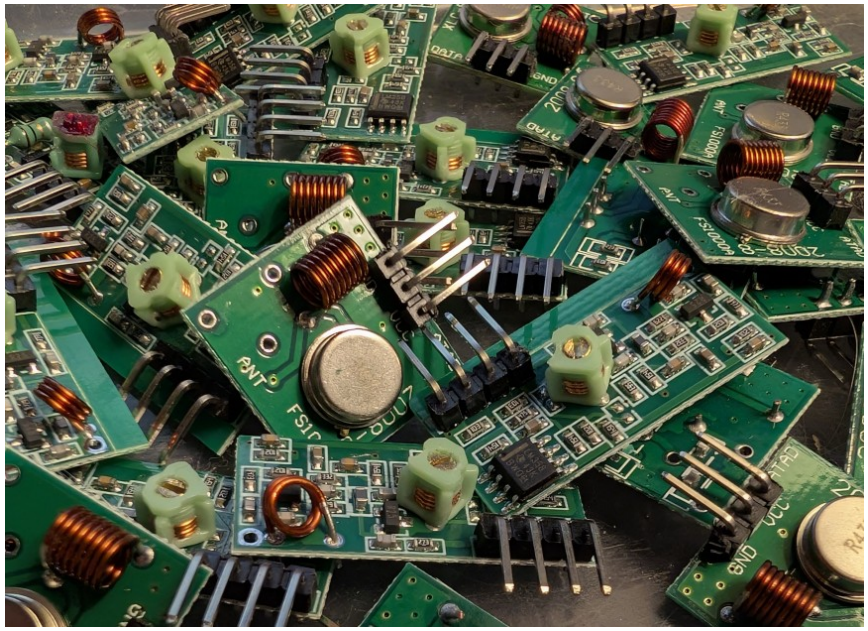
PLL (*phase-locked loop*)

- wymaga rezonatora kwarcowego (stabilność!),
- częstotliwość minimalna rzędu 1..50 MHz,
- większe zużycie prądu w porównaniu do FLL,
- w niektórych układach wymaga starannego zaprojektowania filtra dolnoprzepustowego,
- mniejsza stabilność, ryzyko utraty synchronizacji, dłuższy czas „rozruchu”.

FLL (*frequency-locked loop*)

- sygnał wejściowy: oscylator RC lub kwarc,
- działa dla częstotliwości f0 rzędu 32 kHz, ale także rzędu MHz,
- stabilniejsze działanie
- sygnał wyjściowy niezgodny w fazie z wejściowym,
- szybsze osiągnięcie synchronizacji,
- zużywa mniej energii.

OOK!



OOK: On–off keying

Sprzęt: przykładowa oferta z popularnego portalu (wysyłka gratis).

Dlaczego tak tanio?!

1: ANT 2:GND 3:DATA 4:VCC → 1-bitowa linia danych?

OOK, podejście 1.

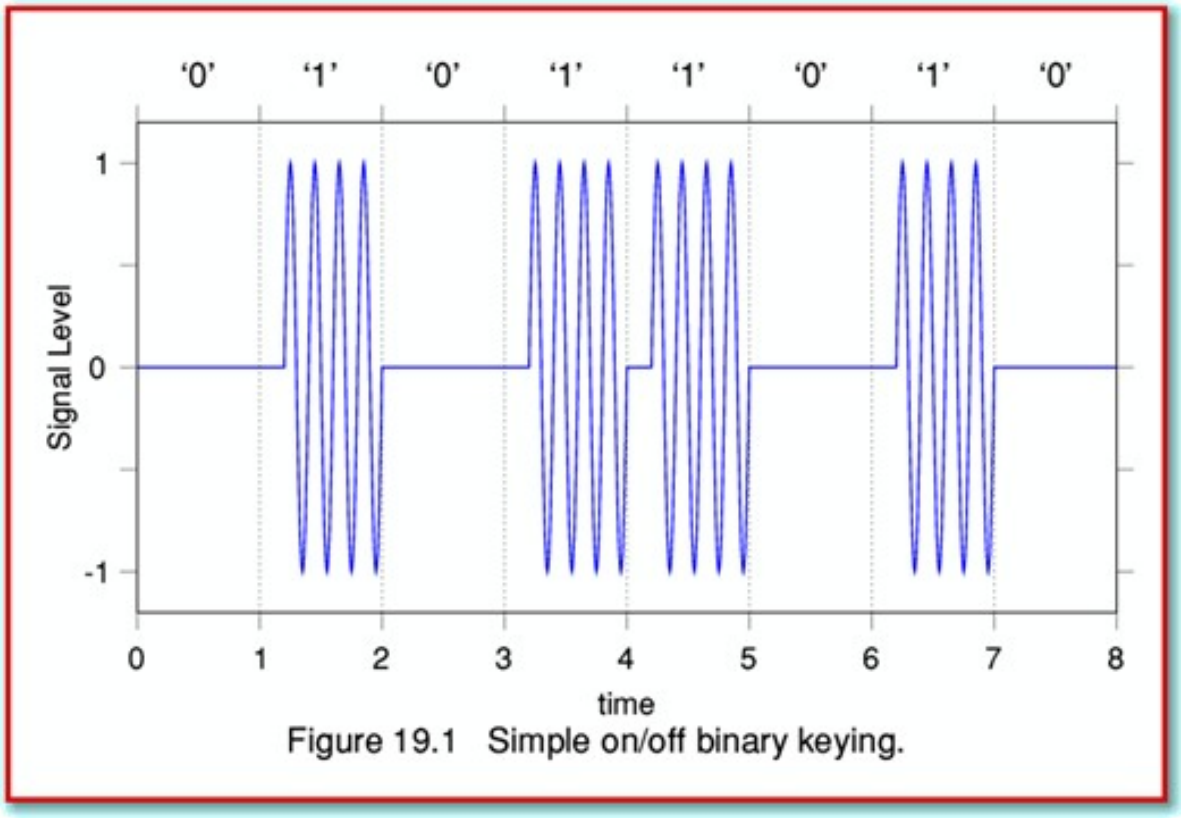


Figure 19.1 Simple on/off binary keying.

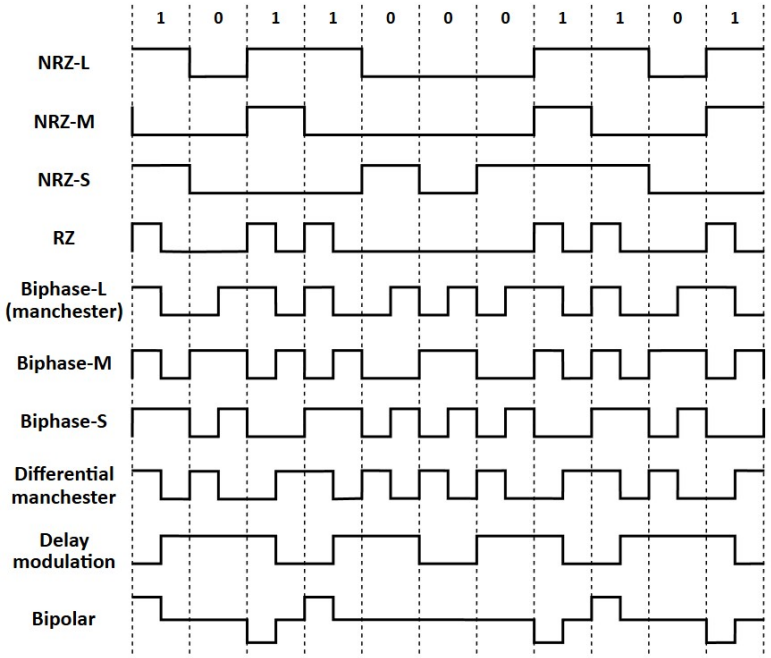
Podejście naiwne: bit ‘0’ wyłącza nośną, bit ‘1’ włącza.

Pytanie: dlaczego tak się nie robi?

OOK, podejście 1b.

- Można modulować nośną sygnałem i niższej częstotliwości gdy ‘1’ (nadawany jest ton) lub cisza gdy ‘0’, ale to nie jest już OOK, oraz nie rozwiązuje problemu ciągów 00000 czy 11111.
- Energetycznie niekorzystne (długi czas emisji nośnej).
- Wolniejsza transmisja gdy niska częstotliwość tonu, albo wracamy do punktu wyjścia.

OOK, podejście 2. Wikipedia



- Odporność na długie strumienie 00..000/111...111
- bieżąca synchronizacja zegarów Tx/Rx
- Utrzymanie ARW (AGC) w odbiorniku RF
- Bipolar: zmniejszenie pasma sygnału
- Skąd wiemy że zaczyna się ramka?
- Jak wstępnie ustawić ‘tick’ (jednostkowy takt zegara)?

OOK: typowy scenariusz przekazu danych cyfrowych

- Odbiornik czeka na preambułę (*preamble*).
- Gdy odbierze pierwsze bity (preambuły? Jeszcze nie wiemy), to może być zmierzona prędkość modulacji strumienia bitów (*clock recovery*) oraz podjęta decyzja o ew. kontynuacji odbierania i dekodowania dalszych bitów.
- Tor analogowy odbiornika dostosowuje wzmocnienie (AGC) i częstotliwość (AFC) do parametrów odbieranego sygnału.
- Kiedy ciąg bitów zostanie poprawnie zdekodowany jako preambuła, to odbiornik jest gotowy do przyjęcia danych (czas tego dostosowania to *settling time*).

Przykład:
Cisza radiowa (idle): 000000.... – nadajnik wyłączony.
Preambuła: 101010101010 [N par ‘10’], bit ‘0’: 001, bit ‘1’: 011.
Średnio nadajnik emituje fale radiowe przez 50% czasu trwania ramki, łatwo można to zmienić inaczej kodując preambułę i bity 0/1, np. ‘0’:10000, ‘1’:11000

c.d.n.