

Typed EventBits i koordynacja stanu

wait any/all, keep/clear-on-exit i timeout snapshot

Karta	K13 / 16	Czas	30 minut
Platforma	Hazard3 / RV32I	Bity	UART/GPIO/APP
Maska all	0x7	Storage	static, heap delta 0
Wersja	v00.01	UUID	5fbcd7c6-...
		karty	

Typowana maska

```
enum class ReadyBit : EventBits_t {
    uart = 1u << 0, gpio = 1u << 1, app = 1u << 2
};
auto all = Mask{ReadyBit::uart} | Mask{ReadyBit::gpio} |
    Mask{ReadyBit::app};
```

Kontrakt K13. Bit opisuje stan true/false. Ponowne set nie zwiększa licznika. Wynik wait jest snapshotem; trzeba oddzielnie sprawdzić, czy spełnia politykę any/all. Clear-on-exit jest jawną decyzją współdzieloną z waiterami.

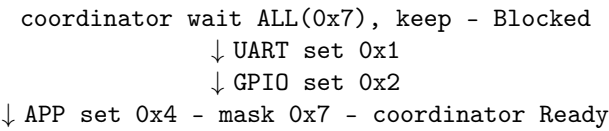
Plan 30 minut

Czas	Tryb	Dowód
0–5	masks	enum 1,2,4; all=7
5–10	publish	coordinator Blocked; trzy taski set
10–15	all/keep	snapshot i current = 7
15–20	clear	tylko GPIO: 7–5
20–25	timeout	all false po 3 tickach, snapshot 5
25–30	any/clear	UART APP true; snapshot 5, final 0

Predykcja

UART|GPIO|APP = _____. Czy dwukrotne set UART da maskę 2? _____. Dlaczego?

1 Trzech publisherów, jeden wait all



Pomiar	Predykcja	Odczyt
coordinator przy first set	Blocked	_____
UART first set result	0x1	_____
UART repeated set	0x1	_____
wait all snapshot	0x7	_____
all satisfied	true	_____
current po keep	0x7	_____

Dlaczego repeated set nie liczy zdarzeń?

Set wykonuje logiczne OR. Jeśli UART musi policzyć pięć ramek, EventGroup nie przechowa tej liczby. Potrzebny jest counter, queue albo notification count.

Podaj przykład poprawnego stanu bitowego i błędnego licznika bitowego:

2 Jawne clear jednego bitu

```
auto before = group.clear(Mask{ReadyBit::gpio}); // returns 0x7
auto after  = group.bits();                      // now 0x5
```

Clear return: _____. Maska po clear: _____. Które subsystemy pozostają ready?
_____.

3 Timeout jest snapshotem, nie osobnym błędem

Po usunięciu GPIO maska 0x5 nie spełnia ALL(0x7).

Pomiar	Predykcja	Odczyt
timeout	3 ticki	_____
returned snapshot	0x5	_____
ALL satisfied	false	_____
current po keep	0x5	_____

Nie testuj timeoutu przez `result==0`: częściowa maska może być niezerowa. Poprawny test porównuje `result & wanted` według any/all.

4 Any i clear-on-exit

```
auto wanted = Mask{ReadyBit::uart} | Mask{ReadyBit::app};
auto result = group.wait(wanted, WaitMode::any,
                        ClearMode::clear_on_exit, 0);
```

Pomiar	Predykcja	Odczyt
snapshot before clear	0x5	_____
ANY satisfied	true	_____
bits cleared	UART i APP	_____
final current mask	0x0	_____

Race do omówienia

Dwóch waiterów czeka na ten sam bit z clear-on-exit. Który zobaczy stan i czy drugi ma gwarancję sukcesu?

ISR boundary

`xEventGroupSetBitsFromISR` nie edytuje grupy bezpośrednio: odkłada operację do timer service task. To wymaga działającego daemon taska i jawnej analizy opóźnienia; K13 mierzy wyłącznie task-context API.

5 Hazard3/GDB i zaliczenie

```
make check
riscv64-unknown-elf-gdb build/task01_event_group/prog.elf
b event_group_debug_checkpoint
```

```
p g_coordinator_state_seen
p g_worker_set_result
p/x g_uart_repeat_mask
p/x g_all_snapshot
p/x g_mask_after_gpio_clear
p/x g_timeout_snapshot
p g_timeout_satisfied
p g_timeout_ticks
p/x g_any_snapshot
p/x g_final_mask
p g_event_group_pass
```

Zaliczenie

- ☐ buduję typed maskę UART/GPIO/APP = 0x7;
- ☐ pokazuję Blocked, wait all/keep i snapshot 0x7;
- ☐ repeated UART set zostaje 0x1, więc nie jest counterem;
- ☐ clear GPIO zmienia 0x7 na 0x5;
- ☐ timeout 3 zwraca snapshot 0x5 i unsatisfied;
- ☐ any/clear zwraca 0x5 i pozostawia final 0x0.

Wyjście

Dlaczego `result!=0` nie dowodzi spełnienia wait all?

Kiedy clear-on-exit może zgubić informację drugiemu waiterowi?

Następna karta K14: software timer, daemon task, one-shot, auto-reload, callback context oraz bezpieczna granica lifetime.