Programowanie mikrokontrolerów 2.0

Przerwania i wyjątki

Marcin Engel Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

26 października 2021

Wyjątki

- Wyjątek – zdarzenie, którego obsługa wymaga przerwania normalnego wykonywania programu i wykonania pewnych czynności związanych z tym zdarzeniem
- Przykłady wyjątków:
  - zakończenie operacji wejścia-wyjścia
  - przepełnienie licznika
  - zakończenie konwersji analogowo-cyfrowej
  - zmiana stanu wyprowadzenia
  - niepoprawny opkod
  - dzielenie przez zero
  - błąd dostępu do pamięci
  - ...

Wyjątki (ang. *exceptions*) w STM32F411

- Zerowanie mikrokontrolera – *Reset*
- Przerwanie niemaskowalne – *NMI*
- Błędy (ang. *faults*):
  - błąd podczas obsługi wyjątku – *Hard fault*
  - błąd ochrony pamięci – *Memory management fault*
  - błąd szyny – *Bus fault*
  - błąd wykonania rozkazu (np. nieprawidłowy opkod, dzielenie przez zero) – *Usage fault*
- Przerwania wspomagające implementację systemu operacyjnego
  - synchroniczne spowodowane wykonaniem rozkazu `SVC`
  - asynchroniczne przerwanie programowe – *PendSV*
  - przerwanie od zegara systemowego `SysTick`
- Przerwania od układów peryferyjnych – *IRQ*
- W dalszej części wykładu będziemy używali terminu *przerwanie* dla wszystkich wyjątków

Numery przewań

- Są zdefiniowane w CMSIS jako typ wyliczeniowy

  `typedef enum {...} IRQn_Type;`
- Przerwania od peryferii mają numery nieujemne

```
WWDG_IRQn                   = 0,
...
TIM2_IRQn                   = 28,
...
```

- Pozostałe przerwania (tzw. systemowe) mają numery ujemne

```
NonMaskableInt_IRQn   = -14,
...
MemoryManagement_IRQn = -12,
BusFault_IRQn         = -11,
UsageFault_IRQn       = -10,
...
SVCall_IRQn           = -5,
...
SysTick_IRQn          = -1,
```

Priorytety przerwań

- Przerwanie może mieć ujemną lub nieujemną wartość priorytetu
- Im mniejsza wartość, tym wyższy priorytet
- Trzy przerwania mają ustalony, ujemny i niekonfigurowalny priorytet:
  - *Reset* – priorytet -3
  - *NMI* – priorytet -2
  - *Hard fault* – priorytet -1
- Pozostałym przerwaniom można nadać priorytet, który jest liczbą nieujemną z zakresu od 0 do 15
- Przewania są obsługiwane w kolejności malejących priorytetów
- Jeśli oczekujące przerwania mają ten sam priorytet, to najpierw obsługiwane jest to o mniejszym numerze

Priorytety przerwań

- Priorytety przerwań są przechowywane w różnych rejestrach
- Do ustawiania priorytetu przerwania służy funkcja biblioteczna

```
void NVIC_SetPriority(IRQn_Type IRQn,
                      uint32_t priority)
```

- Do odczytywania wartości priorytetu służy funkcja biblioteczna

```
uint32_t NVIC_GetPriority(IRQn_Type IRQn)
```

- W obu wartość `IRQn` może być ujemna

Grupowanie priorytetów i przerwania zagnieżdżone

- Czterobitowa wartość priorytetu składa się z dwóch pól:
  - priorytetu wywłaszczania – bardziej znaczące bity
  - podpriorytetu – mniej znaczące bity
- Domyślnie wszystkie bity składają się na priorytet wywłaszczania
- Przerwania zagnieżdżone:
  - przerwania o tym samym lub niższym priorytecie wywłaszczania oczekują na obsłużenie, nie przerywając wykonania bieżącej funkcji obsługi
  - przerwania o wyższym priorytecie wywłaszczania przerywają wykonanie bieżącej funkcji obsługi i są obsługiwane w pierwszej kolejności

Grupowanie priorytetów

- Do definiowania miejsca podziału wartości priorytetu na pola służy funkcja

  ```
  void
  NVIC_SetPriorityGrouping(uint32_t PriorityGroup)
  ```

- Parametr ustala postać priorytetu

`PriorityGroup`	Postać priorytetu
3	`0bxxxx`
4	`0bxxx.y`
5	`0bxx.yy`
6	`0bx.yyy`
7	`0byyyy`

- Mamy zależność

  ```
  PriorityGroup = 7 - prio_bits
  ```

Grupowanie priorytetów

- Mogą okazać się także przydatne funkcje

```
uint32_t
NVIC_EncodePriority(uint32_t PriorityGroup,
                    uint32_t PreemptPriority,
                    uint32_t SubPriority)

void
NVIC_DecodePriority(uint32_t Priority,
                    uint32_t PriorityGroup,
                    uint32_t *pPreemptPriority,
                    uint32_t *pSubPriority)

uint32_t
NVIC_GetPriorityGrouping(void)
```

- Wrócimy do tego na następnym wykładzie

Tablica przerwań

- Tablica przerwań zawiera
  - początkową wartość wskaźnika stosu
  - adresy funkcji obsługi wszystkich przerwań
- Patrz plik `interrupt_vector_stm32f411xe.c` w katalogu `/opt/arm/stm32/inc`

```
/* Interrupt table */
__attribute__ ((section(".isr_vector")))
void (* const g_pfnVectors[])(void) = {
  (void*)&_estack,
  Reset_Handler,
  NMI_Handler,
  ...
  EXT15_10_IRQHandler,
  ...
}
```

Włączanie i wyłącznie przerwań

- Gdy pojawia się przerwanie, sprzęt ustawia bit oznaczający, że przerwanie oczekuje na obsługę
- Funkcja obsługi zostanie wywołana, gdy przerwanie jest indywidualnie włączone
- Domyślnie przerwania są indywidualnie wyłączone
- Do włączania i wyłączania poszczególnych przerwań o nieujemnych numerach służą funkcje

  `void NVIC_EnableIRQ(IRQn_Type IRQn)`
  `void NVIC_DisableIRQ(IRQn_Type IRQn)`
- Systemowe
  - zegar systemowy – rejestr `SysTick->CTRL`
  - *Memory management fault*, *Bus fault* i *Usage fault* – rejestr `SCB->SHCSR`
  - *PendSV* – włączone (aktywowanie w rejestrze `SCB->ICSR`)
  - *SVCall* – włączone (aktywowanie instrukcją `SVC`)
  - Patrz Programming manual PM0214 w katalogu `/opt/arm/stm32/doc`

Włączanie i wyłącznie przerwań

- Oprócz tego można globalnie wyłączać i włączać wszystkie przerwania o konfigurowalnym priorytecie

  ```
  void __disable_irq(void)
  void __enable_irq(void)
  ```

- Powyższe funkcje wywołują instrukcje `cpsid i`, `cpsie i`, które operują na rejestrze `PRIMASK`
- Można też wyłączać i włączać wszystkie przerwania maskowalne (o konfigurowalnym priorytecie i *Hard fault*)

  ```
  void __disable_fault_irq(void)
  void __enable_fault_irq(void)
  ```

- Powyższe funkcje wywołują instrukcje `cpsid f`, `cpsie f`, które operują na rejestrze `FAULTMASK`
- Domyślnie wszystkie przerwania są globalnie włączone

Blokowanie przerwań o niskich priorytetach

- Jeśli rejestr `BASEPRIO` zawiera wartość niezerową, to obsługiwane są tylko przerwania o priorytetach wyższych niż reprezentowany przez tę wartość
- Dostęp do rejestru `BASEPRIO`

```
void __set_BASEPRIO(uint32_t value)
uint32_t __get_BASEPRIO(void)
```

- Wartość parametru `value` to numer priorytetu odpowiednio przesunięty w lewo

```
prio_bits = 7 - NVIC_GetPriorityGrouping();
value = priority << 8 - prio_bits;
```

- Patrz plik `irq.h` w katalogu `/opt/arm/stm32/inc`, wrócimy do tego na następnym wykładzie

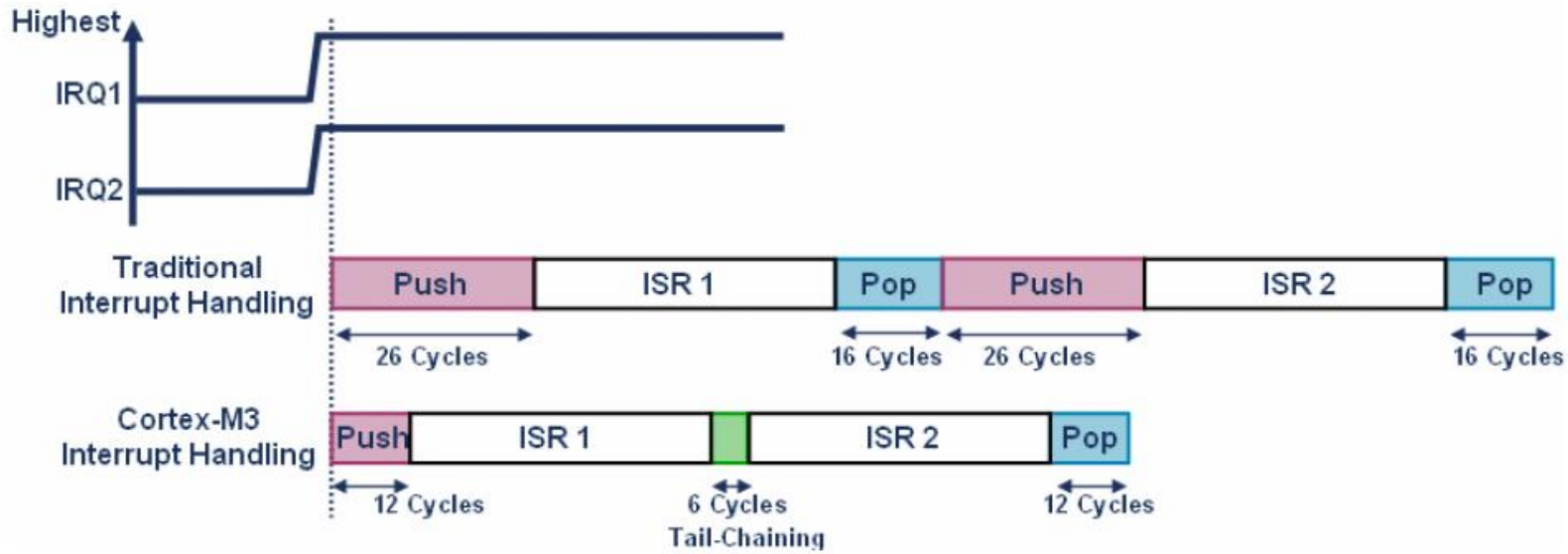
Jak rozpoczyna się obsługa przerwania?

- Jeśli na obsługę oczekuje przerwanie o dostatecznie dużym priorytecie (por. poprzedni slajd) oraz
  - nie jest obsługiwane inne przerwanie lub
  - nowe przerwanie ma wyższy priorytet wywłaszczenia niż przerwanie obsługiwane
- to
  - wartości rejestrów `R0`, `R1`, `R2`, `R3`, `R12`, `LR`, `PC`, `PSR` są odkładane na stos
  - z tablicy przerwań jest odczytywany adres funkcji obsługi
  - do rejestru `LR` jest zapisywana specjalna wartość `EXC_RETURN`
  - rozpoczyna się wykonanie funkcji obsługi przerwania
- To nie jest do końca prawda: obsługa spóźnionych przerwań!

Jak kończy się obsługa przerwania?

- Jeśli wykonanie funkcji obsługi przerwania zostało zakończone i nie ma innego przerwania do obsługi, to wartości rejestrów są zdejmowane ze stosu i następuje powrót do miejsca, w którym przerwane zostało wykonanie kodu
- Jeśli wykonanie funkcji obsługi przerwania zostało zakończone i może zostać obsłużone kolejne oczekujące przerwanie, to sterowanie jest przekazywane do funkcji jego obsługi bez odtwarzania stanu ze stosu (łańcuchowa obsługa przerwań)

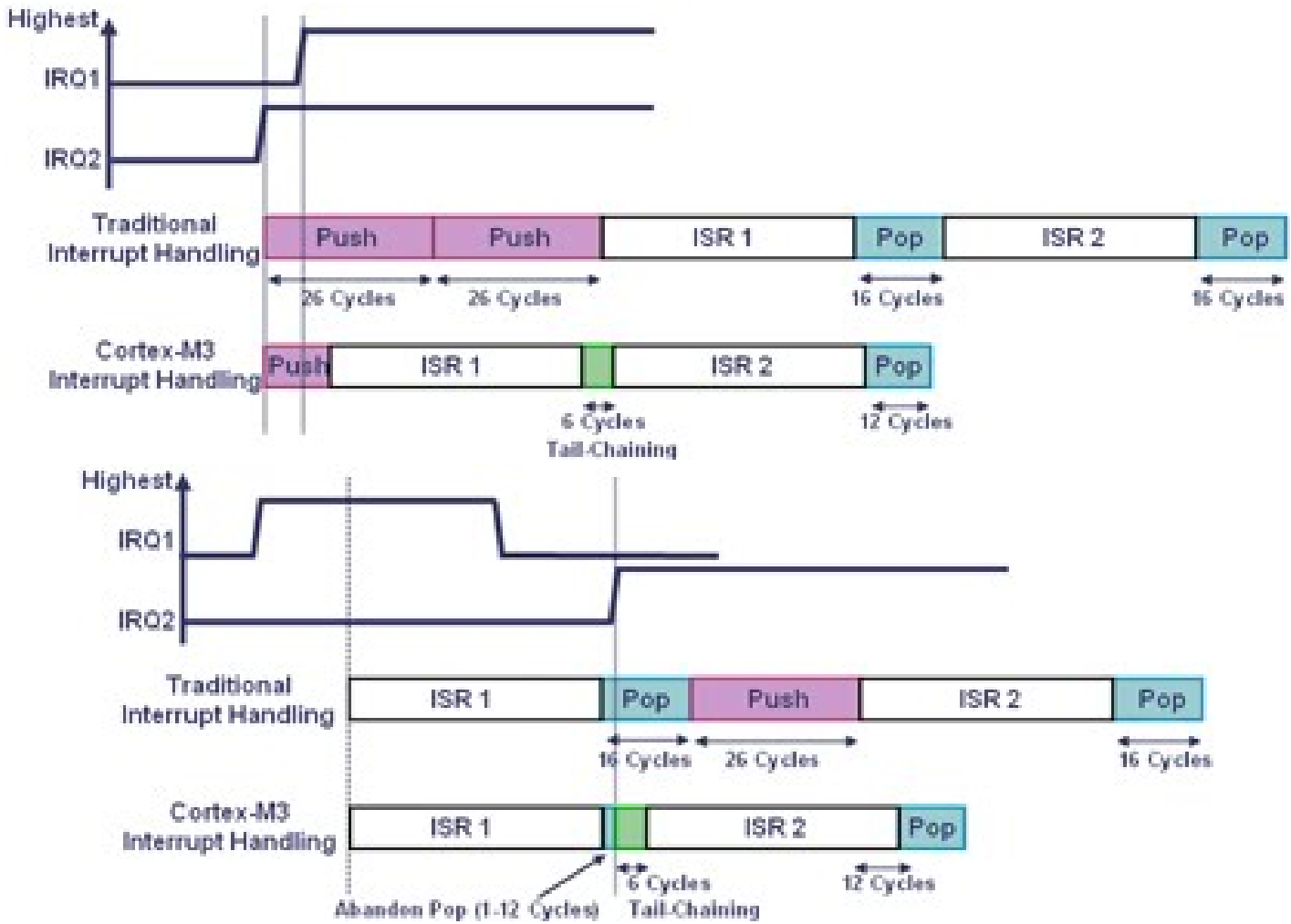
Łańcuchowa obsługa przerwań



Źródło: `http://www.arm.com`

- Może dochodzić do zagłodzenia!

Łańcuchowa obsługa przerwań, dalsze przykłady



Źródło: http://www.arm.com

ABI funkcji i przerwań w Cortex-M

- Pierwsze 4 argumenty funkcji są przekazywane w rejestrach `R0` do `R3`, pozostałe przez stos
- Funkcja może dowolnie modyfikować rejestry `R0` do `R3` i `R12`
- Funkcja musi przywrócić wartości rejestrów `R4` do `R11`, `LR` i `SP` sprzed wywołania
- Przed wywołaniem funkcji obsługi przerwania procesor odkłada na stos wartości rejestrów `R0` do `R3`, `R12`, adres powrotu, `PSR`, `LR`
- Funkcja, która ma obsługiwać przerwanie, to najzwyklejsza funkcja języka C o sygnaturze

  ```
  void Handler(void);
  ```

- Nie są potrzebne żadne dodatkowe atrybuty!

ABI przerwań, problem

- Skąd procesor wie, czy instrukcja `bx lr` jest powrotem ze zwykłej funkcji, czy z obsługi przerwania (trzeba coś zdjąć ze stosu)?

ABI przerwań, rozwiązanie problemu

- Przed wywołaniem funkcji obsługującej przerwanie procesor zapisuje do rejestru `LR` wartość:
  - `0xfffffff1` – powrót do obsługi innego przerwania (ang. *privileged handler*), odtwórz stan z `MSP`, po powrocie używaj `MSP`
  - `0xfffffff9` – powrót do wątku uprzywilejowanego (ang. *privileged thread*), odtwórz stan z `MSP`, po powrocie używaj `MSP`
  - `0xfffffffd` – powrót do wątku użytkownika (ang. *user thread*), odtwórz stan z `PSP`, po powrocie używaj `PSP`
  - `0xffffffe1`, `0xffffffe9`, `0xffffffed` – jak wyżej, ale z odtwarzaniem stanu rejestrów zmiennoprzecinkowych

Przerwania zewnętrzne

- Każde wyprowadzenie mikrokontrolera może być źródłem przerwania zewnętrznego
- Przerwanie zewnętrzne może być wyzwalane zboczem narastającym, zboczem opadającym lub każdą zmianą stanu
- Za wykrycie tych zdarzeń i przekazanie ich do układu `NVIC` odpowiada podukład `EXTI`
- Możemy chcieć podłączyć do mikrokontrolera zewnętrzny układ zgłaszający zdarzenie poziomem
- Nie są obsługiwane przerwania wyzwalane poziomem
- Jak sobie z tym poradzić, opowiemy w przyszłości

Cechy układu `EXTI`

- Jest wyposażony w 21 linii:
  - do i-tej linii ($i = 0, \ldots, 15$) może zostać podłączone i-te wyprowadzenie dowolnego portu wejścia-wyjścia
  - linie 16, 17, 18, 21, 22 są wykorzystywane przez zegar czasu rzeczywistego, kontroler napięcia, kontroler USB i na razie nie będziemy o nich mówić
- Przerwanie na każdej linii może być konfigurowane i wyzwalane niezależnie od pozostałych
- Rejestry `EXTI->IMR`, `EXTI->RTSR`, `EXTI->FTSR`, `EXTI->SWIER`, `EXTI->PR`
- W każdym z nich i-ty bit ($i = 0, \ldots, 15$) odnosi się do i-tej linii

Konfigurowanie kontrolera przerwań zewnętrznych

- Ustawienie bitu w rejestrze `EXTI->IMR` włącza przerwania na linii
- Ustawienie bitu w rejestrze `EXTI->FTSR` konfiguruje wyzwalanie zboczem opadającym
- Ustawienie bitu w rejestrze `EXTI->RTSR` konfiguruje wyzwalanie zboczem narastającym
- Dopuszczalne jest wyzwalanie zarówno zboczem opadającym, jak i narastającym

Obsługa przerwań zewnętrznych

- Ustawiony bit w rejestrze `EXTI->PR` oznacza, że pojawiło się zdarzenie, które może wyzwolić przerwanie
- Bit w rejestrze `EXTI->PR` zeruje się, zapisując do niego 1!
- Programowe wyzwolenie przerwania umożliwia rejestr `EXTI->SWIER`
- Jeśli bit w `EXTI->SWIER` ma wartość 0 to ustawienie go na 1 powoduje ustawienie odpowiedniego bitu w `EXTI->PR`
- Bit w `EXTI->SWIER` jest zerowany poprzez zerowanie bitu w `EXTI->PR`

Wybieranie źródła przerwania dla linii

- W rejestrach `SYSCFG->EXTICR[`i`]` dla $i = 0..3$ znajduje się informacja, które porty są podłączone do poszczególnych linii przerwań zewnętrznych
- Informacja o liniach 0, 1, 2, 3 znajduje się w rejestrze `SYSCFG->EXTICR[0]`
- Informacja o liniach 4, 5, 6, 7 znajduje się w rejestrze `SYSCFG->EXTICR[1]` itd.
- Przykładowe podpięcie wyprowadzenia `PC13` do linii 13:

```
SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PC;
```

- Uwaga na indeksy!
- Trzeba pamiętać o uprzednim włączeniu taktowania układu `SYSCFG`

```
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
```

Obsługa po stronie NVIC

- Numery przerwań generowanych przez linie zewnętrzne:
  - `EXTI0_IRQn`, `EXTI1_IRQn`, `EXTI2_IRQn`, `EXTI3_IRQn`, `EXTI4_IRQn`
  - `EXTI9_5_IRQn`
  - `EXTI15_10_IRQn`
- Obsługa, jeśli włączone jest przerwanie tylko na jednej linii spośród 10 do 15:

```
void EXTI15_10_IRQHandler(void) {
  EXTI->PR = EXTI_PR_PR13;

  /* Tu wstaw kod obsługujący przerwanie. */
  ...
}
```

- Należy pamiętać o wyzerowaniu bitu w rejestrze `EXTI->PR`
- Co trzeba zrobić, jeśli aktywne są przerwania na kilku liniach?

Konfigurowanie przerwań zewnętrznych – podsumowanie

- Włącz taktowanie portów wejścia-wyjścia, czyli odpowiednich układów `GPIOx`
- Włącz taktowanie układu `SYSCFG`
- Skonfiguruj wejścia za pomocą rejestrów układów `GPIOx` (rezystor podciągający, ściągający do masy lub jego brak)
- Skonfiguruj kontroler przerwań zewnętrznych:
  - wybierz źródła przerwania dla poszczególnych linii, rejestr `SYSCFG->EXTICR[i]`
  - ustal sposób wyzwalania przerwań (zbocze narastające, opadające, oba), rejestry `EXTI->FTSR`, `EXTI->RTSR`
  - włącz przerwania na odpowiednich liniach, rejestr `EXTI->IMR`

Konfigurowanie przerwań zewnętrznych – podsumowanie

- Włącz taktowanie, przykładowo

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
```

- Zamiast operować na bitach rejestrów, wykorzystaj funkcję `GPIOinConfigure`, przykładowo

```
GPIOinConfigure(GPIOC, 13, GPIO_PuPd_UP,
                EXTI_Mode_Interrupt,
                EXTI_Trigger_Falling);
```

- Skasuj znaczniki oczekujących przerwań, zapisując 1 w odpowiednich bitach rejestru `EXTI->PR`
- Skonfiguruj kontroler `NVIC`:
  - ustaw żądane priorytety
  - włącz odpowiednie przerwania, przykładowo

```
NVIC_EnableIRQ(EXTI15_10_IRQn);
```

Ciekawostka

- Te instrukcje mogą zmienić zawartość rejestru `EXTI->PR`

```
EXTI->PR |= 0;
EXTI->PR = EXTI->PR;
```