Technologia GSM w elektronice (10)

Open AT – System przerwań

W poprzednim odcinku dotyczącym programowania modułów GSM Sierra Wireless zajmowaliśmy się zagadnieniem wielowątkowości oraz mechanizmów z tym związanych. Teraz przyszła kolej na omówienie systemu przerwań dostępnych w Open AT.

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 17855, pass: 4s406qj2
- poprzednie części kursu

System operacyjny Open AT, pod kontrolą którego pracują programowalne moduły *Sierra Wireless AirPrime,* jest, jak wspomnieliśmy na początku naszego cyklu, systemem operacyjnym czasu rzeczywistego (RTOS). Podstawowym wymaganiem dla takich systemów jest określenie najgorszego (najdłuższego) czasu, po którym urządzenie, niezależnie od pozostałych warunków, podejmie działanie w reakcji na wystąpienie określonego zdarzenia.

W Open AT procesy krytyczne pod względem czasu powinny być obsługiwane przez system przerwań IRQ. Źródłami przerwań w Open AT mogą być:

- linie przerwań sprzętowych INT0 i INT1,
- układ Audio,
- Timer TCU (Timer & Capture Unit),
- układ kontroli interfejsów SPI oraz I²C.

W systemie dostępne mamy dwa poziomy przerwań: *High Level Interuption* oraz *Low Level interuption* (**rysunek 1**). Ponadto, dla przerwań niskopoziomowych, wyróżniono przerwania typu Fast Interruption REQUEST (FIQ) mające zastosowanie jedynie w odniesieniu do sprzętowych przerwań zewnętrznych (INT0 i INT1). Umiejscowienie przerwań w systemie pod względem priorytetu wykonywania przedstawia diagram na rysunku 1

Przerwania wysokiego poziomu mają priorytet wyższy niż wszystkie wątki aplikacji, ale niższy niż priorytet stosu GSM. Dzięki temu gwarantują czas do obsługi zdarzenia wynoszący około 10 ms. Przerwania niskiego poziomu mają priorytet wyższy niż stos GSM i pozwalają na obsłużenie zdarzenia w czasie poniżej 1 ms. Przerwania FIQ mają najwyższy priorytet w systemie. Tu czas obsługi to jedynie 50 µs.

Używając przerwań niskiego poziomu, należy zwrócić szczególną uwagę na to, aby kod zawarty w obsłudze przerwania był krótki i nie zakłócił pracy stosu GSM. Z tego również powodu w obsłudze przerwań niskiego poziomu nie można stosować niektórych funkcji ADL. Próba ich użycia zakończy się następującym kodem błędu: *ADL_RET_ERR_SERVICE_LOCKED.*

Aby skorzystać z systemu przerwań, należy najpierw zadeklarować funkcję obsługi, która będzie wywoływana przez system w momencie pojawienia się określonego zdarzenia.

W tym celu należy wykorzystać funkcję *adl_irqSubscribe().* Prototyp funkcji ma następującą postać:

s32 adl_irqSubscribe (adl_irqHandler_f IrqHandler,
adl_irqNotificationLevel_e NotificationLevel,
adl_irqPriorityLevel_e PriorityLevel,
adl_irqOptions_e Options);

gdzie

adl_irqHandler – funkcja obsługi przerwania, która będzie wykonywana po wystąpieniu przerwania

NotificationLevel – poziom przerwania, do wyboru: ADL_IRQ_NOTIFY_HIGH_LEVEL dla przerwania wysokiego poziomu lub ADL_IRQ_NOTIFY_LOW_LEVEL dla przerwania niskiego poziomu

PriorityLevel – prorytet przerwania. Wartość od 0 (najniższy priorytet) do wartości uzyskanej za pomocą funkcji *adl_irqGetCapabilities()* pomniejszonej o 1 (najwyższy priorytet)

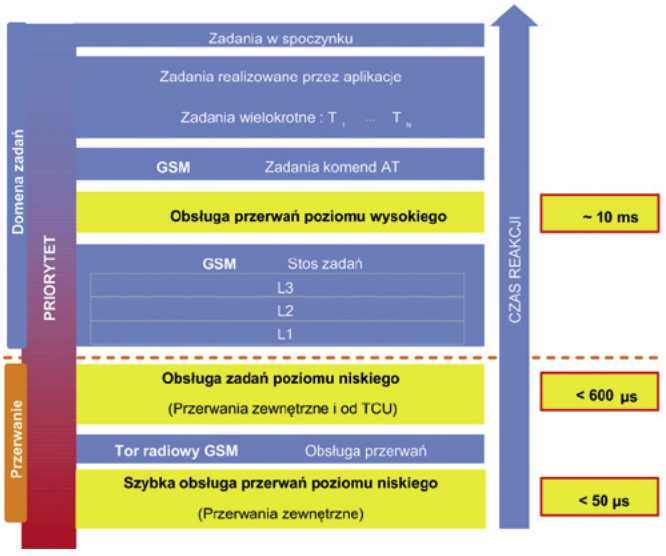
Options – opcje dotyczące automatycznego odczytu struktury z informacją o źródle przerwania, istotne tylko w przypadku przerwań wysokiego poziomu

Wartość zwracana przez funkcję *irqSubscribe()* to handler (uchwyt), który będzie wykorzystany jako argument w funkcjach serwisów stosujących przerwania, na przykład serwisu przerwań zewnętrznych INT0/INT1 (Serwis ExtInt), serwisu Timera TCU czy serwisu Audio.

Najlepiej będzie, jeśli w tym miejscu posłużymy się przykładem z **listingu 1**, w którym przerwanie niskiego poziomu wykorzystano do obsługi pinu zewnętrznego przerwania INT0.

Opis funkcji służących do deklarowania przerwań oraz funkcji callback do ich obsługi znajdziemy w dokumentacji w rozdziale IRQ Sernice, natomiast opis funkcji odpowiedzialnych za obsługę pinów przerwań zewnętrznych INT0 oraz INT1 znajdziemy w dokumentacji w rozdziale ExtInt ADL Sernice.

W przedstawionym przykładzie system przerwań zostaje wykorzystany do obsługi przerwania zewnętrznego na pinie INT0. Najpierw, za pomocą funkcji *adl_irqGetCapabilities()* pobieramy wartość najwyższego



Rysunek 1. Diagram procesów w Open AT

możliwego priorytetu, jaki jest dostępny w systemie dla przerwań. Jak wspomnieliśmy wcześniej, to wartość pomiędzy 0 a wartością, jaką zwraca wspomniana funkcja. Po zadeklarowaniu przerwania niskiego poziomu i wskazaniu funkcji *callback* do jego obsługi (w naszym wypadku *MyExtIntIrqHandler*) wykonujemy powiązania z zewnętrznym pinem przerwania INT0 za pomocą *extintSubscribe()*. Jako argumenty podajemy numer pinu (do wyboru mamy INT0 lub INT1) oraz wcześniej otrzymany handler.

Traces

Id	Level	Message
ADL	1	Binary header at 00260000
ADL	16	[ADL PORT] subs (00262DBD) : 0
ADL	16	[ADL PORT] subs (002649A3) : 1
ADL	22	[ADL] flash subs 2 : -4
ADL	22	Flh Obj 0000 Len : 4
ADL	22	Read Flh Obj 0000 (4) : 0
ADL	16	[ADL PORT] event : 0 (port 80 ; state 0)
ADL	16	[ADL PORT] event : 0 (port 70 ; state 0)
ADL	16	[ADL PORT] event : 0 (port 01 ; state 0)
ADL	16	[ADL PORT] event : 0 (port 02 ; state 0)
ADL	16	[ADL PORT] event : 0 (port 03 ; state 0)
ADL	22	Flh Obj 0000 Len : 4
ADL	16	[ADL PORT] unsubs (1) : 0
ADL	1	Highest priority = 3
LLH	1	zostalo wywolane przerwanie
LLH	1	Context Id = fd
LLH	1	zostalo wywolane przerwanie
LLH	1	Context Id = fd

Rysunek 2. Przykład działania aplikacji wykorzystującej przerwania

Do obsługi pinów przerwań zewnętrznych możemy używać nawet dwóch przerwań naraz – jedno niskiego i jedno wysokiego poziomu. Taka możliwość jest przydatna, gdy potrzebujemy bardzo szybkiej reakcji, którą gwarantuje przerwanie niskopoziomowe oraz bardziej rozbudowanej obsługi, którą możemy uzyskać dzięki przerwaniu wysokiego poziomu.

Jak wspomniałem wcześniej, ze względu na wysoki priorytet przerwań niskiego poziomu, w funkcji ich obsługi należy zawrzeć możliwie krótki kod, ponieważ zbyt długie wykonywanie funkcji obsługi mogłoby np. zakłócić pracę stosu GSM. Jeśli do obsługi jakiegoś zdarzenia mamy zadeklarowane dwa przerwania, to funkcja *callback* przerwania niskiego poziomu, zwracając wartość TRUE, powoduje automatyczne wywołanie funkcji *callback* przerwania wysokiego poziomu. W funkcji obsługi przerwania wysokiego poziomu możemy zawrzeć procedury wykonujące pozostałe czynności związane z obsługą danego zdarzenia. W naszym przykładzie wykorzystane jest jedynie przerwanie niskiego poziomu.

Ostatnim argumentem funkcji *extintSubscribe()* jest tablica konfigurująca *extintConfig*. Pozwala ona na określenie, czy przerwanie ma być wyzwalane zboczem, czy poziomem oraz daje możliwość zastosowania jednego z dwóch dostępnych filtrów, co pozwala na przykład na wyeliminowanie efektu drgań.

Przerwania, tak jak omawiane w poprzednim odcinku wątki, to oddzielne procesy w systemie. Podobnie jak wątki, mają nadane przez system ID (0xfd dla przerwania niskopoziomowego oraz 0xfe dla prze-

Listing 1. Obsługa zewnętrznego przerwania INT0

```
#include "adl_global.h"
void main_task ( void );
const adl_InitTasks_t adl_InitTasks [] =
{
                { main_task,  3000, „main", 1 },
                { 0, 0, 0, 0 }
};
const u32 adl_InitIRQLowLevelStackSize = 1024;

s32 IrqHandle; // IRQ service handle
s32 ExtIntHandle; //ExInt handle
adl_irqCapabilities_t Caps;
adl_extintConfig_t extintConfig =
        { ADL_EXTINT_SENSITIVITY_RISING_EDGE , ADL_EXTINT_FILTER_BYPASS_MODE ,0,0, NULL };

bool MyExtIntIrqHandler ( adl_irqID_e Source, adl_irqNotificationLevel_e NotificationLevel,adl_irqEventData_t * Data )
{
        //kod obslugi przerwania
        TRACE((1,"zostalo wywolane przerwanie"));
TRACE((1,"Context Id = %x", adl_ctxGetID ()));//odczytaj context ID
        return FALSE; //nie wywoluj przerwania wys. poziomu dla tego samego zgloszenia
}
void main_task ( void )
{
adl_irqGetCapabilities ( &Caps );
TRACE((1,"Highest priority = %d", Caps.PriorityLevelsCount));
IrqHandle = adl_irqSubscribe ( MyExtIntIrqHandler, ADL_IRQ_NOTIFY_LOW_LEVEL, Caps.PriorityLevelsCount - 1, ADL_IRQ_
OPTION_AUTO_READ );
ExtIntHandle = adl_extintSubscribe ( ADL_EXTINT_PIN0 , IrqHandle, NULL, &extintConfig );
}
```

Listing 2. Przykład maskowania przerwań

```
adl_irqCapabilities_t Caps;
adl_irqConfig_t Config;

adl_irqGetCapabilities ( &Caps );

Config.PriorityLevel = Caps.PriorityLevelsCount - 1; // Najwyższy priorytet
Config.Enable = TRUE; // Włącz obsługe przerwan
Config.Options = ADL_IRQ_OPTION_AUTO_READ; // Auto-read

MyIRQHandle = adl_irqSubscribeExt ( MyIRQHandler,
ADL_IRQ_NOTIFY_LOW_LEVEL, &Config );

(…)

// gdzies w programie
// Maskuj przerwanie
adl_irqGetConfig ( MyIRQHandle, &Config );
Config.Enable = FALSE;
adl_irqSetConfig ( MyIRQHandle, &Config );

(…)

// dalej w programie
// Wyłącz maskowanie
adl_irqGetConfig ( MyIRQHandle, &Config );
Config.Enable = TRUE;
adl_irqSetConfig ( MyIRQHandle, &Config );


#include „adl_global.h"
#include „generated.h"

s32 TCUHandle; // TCU service handle
s32 IrqHandle; // IRQ service handle

// TCU - tablica konfiguracyjna: 5ms, cykliczny
adl_tcuTimerSettings_t Config = { { 5, ADL_TCU_TIMER_UNIT_MS }, TRUE };
```

Listing 3. Przykład użycia jednostki TCU do realizacji timera odmierzającego czas z dokładnością do 1 ms

```
// funkcja callback dla przerwania od timera
bool MyTCUHandler (adl_irqID_e Source, adl_irqNotificationLevel_e NotificationLevel, adl_irqEventData_t * Data )
{
if ( Source == ADL_IRQ_ID_TIMER ) // Check for Timer event
        {
        TRACE (( 1, „Timer event” ));
        //wykonaj zadanie
        }
return TRUE;
}

void main_task ( void )
{
IrqHandle = adl_irqSubscribe ( MyTCUHandler, ADL_IRQ_NOTIFY_LOW_LEVEL, 0, 0);
TCUHandle = adl_tcuSubscribe ( ADL_TCU_ACCURATE_TIMER, IrqHandle, 0,&Config, NULL );
adl_tcuStart ( TCUHandle );
}
```

rwania wysokopoziomowego) oraz wymagają zadeklarowania pamięci stosu.

W przykładzie z listingu 1 zarówno stos przerwania, jak i tablica inicjalizująca wątki zostały zawarte w kodzie programu. Pamiętając jednak, że dużo wygodniejszym sposobem jest wykorzystanie formularza dostępnego po kliknięciu na plik generated.c, w kolejnych przykładach będę się posługiwał tym właśnie sposobem i wartości te nie będą prezentowane w kodzie.

Efekt działania naszej aplikacji możemy zaobserwować w okienku Trace (**rysunek 2**).

Patrząc na rysunek 2, zauważymy kolejną różnicę w porównaniu z tradycyjną aplikacją niewykorzystującą przerwań. Standardowe funkcje programu generują logi za pomocą poziomów ADL, natomiast logi TRACE z funkcji przerwań niskopoziomowych są generowane pod Id LLH (dla przerwań wysokiego poziomu będzie to HHL).

Jeśli zależy nam na wyjątkowo szybkiej obsłudze przerwania, to możemy jeszcze podwyższyć priorytet przerwania niskiego poziomu i uzyskać czas reakcji około 50 μs.

W tym celu należy wykorzystać funkcję *adl_extintSetFIQStatus()*, a więc w naszym programie należałoby dodać linijkę:
adl_extintSetFIQStatus (ExtIntHandle, TRUE);

Jeśli chcielibyśmy w naszej aplikacji skorzystać z możliwości maskowania przerwań, to zamiast funkcji *adl_irqSubscribe()*, powinniśmy użyć funkcji *adl_irqSubscribeExt()*. Fragmenty takiego programu przedstawiono na **listingu 2**.

Oprócz linii zewnętrznych źródłem przerwań dla systemu może być również jednostka TCU (*Timer&Capture Unit*). Można jej użyć jako precyzyjnego timera lub licznika zdarzeń.

Pamiętamy z poprzednich odcinków, że standardowe timery software'owe potrafiły pracować z maksymalną dokładnością równą 18,5 ms. W przypadku timera TCU dokładność wynosi 1 ms. Przykładową aplikację dokładnego timera opartego na jednostce TCU przedstawiono na listingu 3. Parametry takie jak stos oraz struktura wątków zostały skonfigurowane za pomocą formularza generated.c.

W przedstawionym przykładzie tworzymy timer, który co 5 ms wywołuje funkcję *callback* (w naszym wypadku *MyTCUHandler*). Powiązanie przerwania z serwisem TCU zostało utworzone za pomocą funkcji *adl_tcuSubscribe()*. Argumentem, oprócz handlera przerwania, jest tu tablica konfiguracyjna, w której oprócz czasu timera możemy ustawić jego cykliczność. Jednostkę TCU należy jeszcze wystartować komendą *adl_tcuStart()*. Od tej pory kod zawarty w funkcji *MyTCUHandler ()* będzie wykonywany co 5 ms.

Innym zastosowaniem jednostki TCU jest licznik zdarzeń zewnętrznych. Przykładową aplikację zamieszczono na **listingu 4**.

W przedstawionym przykładzie używamy tej samej funkcji subskrybującej, co poprzednio, natomiast argument pierwszy (*ADL_TCU_EVENT_CAPTURE*) mówi nam, że chcemy korzystać z licznika zdarzeń. Tablica konfiguracyjna *Config* nakazuje zliczanie zdarzeń występujących na pinie przerwań INT0, reakcję na narastające zbocze oraz na wywoływanie funkcji *callback* (*MyTCUHandler)* dla przerwania co 15 ms. Natomiast w funkcji *callback* następuje odczyt, ile zdarzeń wystąpiło od poprzedniego wywołania funkcji.

Więcej informacji na temat produktów Sierra Wireless można znaleźć na stronach producenta: *www.sierrawireless.com* lub kontaktując się z firmą ACTE Sp. z o.o., która jest oficjalnym dystrybutorem opisywanych produktów oraz zapewnia pełne wsparcie techniczne.

Adrian Chrzanowski
Acte Sp. z o.o.

Listing 4. Przykład aplikacji licznika zdarzeń zewnętrznych zrealizowanego z użyciem TCU

```
#include „adl_global.h”
#include „generated.h”

s32 TCUHandle;
s32 IrqHandle;

adl_tcuEventCaptureSettings_t Config = { ADL_EXTINT_PIN0, ADL_TCU_EVENT_TYPE_RISING_EDGE, 15, NULL};

bool MyTCUHandler (adl_irqID_e Source, adl_irqNotificationLevel_e
NotificationLevel, adl_irqEventData_t * Data )
{
if ( Source == ADL_IRQ_ID_EVENT_CAPTURE )
        {
        if ( Data->Instance == 0 ) // Sprawdz Pin
                {
                // Get Source Data
                u32 SourceData = ( u32 ) Data->SourceData;

                TRACE (( 1, "%d zdarzen wystapilo od czasu ostatniego wywolania”, SourceData));
                }
        }
return TRUE;
}

void main_task ( void )
{
IrqHandle = adl_irqSubscribe ( MyTCUHandler, ADL_IRQ_NOTIFY_LOW_LEVEL, 0, ADL_IRQ_OPTION_AUTO_READ );
TCUHandle = adl_tcuSubscribe ( ADL_TCU_EVENT_CAPTURE, IrqHandle, 0,&Config, NULL );
adl_tcuStart ( TCUHandle );
}
```