MSP430 w przykładach (4)

Obsługa portów wejścia-wyjścia

W tym odcinku kursu omówimy sposób sterowania liniami wejścia/wyjścia mikrokontrolera MSP430. Zaprezentujemy przykłady obsługi urządzeń peryferyjnych dołączonych do wyprowadzeń mikrokontrolera (dioda RGB, przycisk SW1). Omówimy sposób użycia przygotowanej na potrzeby kursu biblioteki obsługi alfanumerycznego wyświetlacza LCD ze sterownikiem Hitachi HD44780.

Zainstalowany w module „Komputerek" mikrokontroler MSP430f1232 ma trzy porty I/O. Porty P1 oraz P3 mają po 8 linii, natomiast port P2 jest ma 6 linii. W sumie, w mikrokontrolerze MSP430f1232 mamy do dyspozycji 22 linie wejścia/wyjścia, a 14 z nich (porty P1, P2) może służyć jako wejścia sygnałów przerwań. Wyprowadzenia mikrokontrolera pokazano na **rysunku 1**.

Dodatkowe informacje:
W materiałach dodatkowych na płycie CD i serwerze FTP publikujemy filmy ilustrujące działanie przykładów prezentowanych w artykule.

Konfigurowanie portów I/O

Porty wejścia/wyjścia w MSP430 konfigurujemy za pomocą 8-bitowych rejestrów (szczegółowy opis w materiałach dołączonych na CD/FTP). Ustawiając i zerując bity w rejestrach możemy sterować pracą każdej z linii portów. Linie portów MSP430 mogą pracować w trybie pracy funkcyjnej lub w trybie pracy wejścia/wyjścia. Tryb pracy linii wybiera się za pomocą rejestru PxSEL (0 – wejście-wyście, 1 – funkcyjny). W wypadku, gdy linia pracuje w trybie wejścia/wyjścia, to manipulując bitami rejestru PxDIR możemy wybrać kierunek linii (0 -wejście, 1- wyjście). Jeśli linia pracuje w trybie wejścia/wyjścia (PxSEL=0) i jest ustawiona w kierunku wyjścia (PxDIR=1), to korzystając z rejestru PxOUT możemy ustawić zmieniać poziom na wyjściu (0 – niski, 1 – wysoki). Sterując wyjściami trzeba pamiętać, że restart mikrokontrolera nie zmienia wartości rejestru PxOUT (zerowany jest rejestr kierunku PxDIR). Po starcie MSP430 możemy zmienić kierunek linii z wejścia na wyjście, a na linii automatycznie pojawi się uprzednio ustawiony stan. Gdy linia pracuje w trybie wejścia/wyjścia (PxSEL=0) i jest ustawiona w kierunku wejścia (PxDIR=0), to poziom na wejściu możemy odczytać korzystając z rejestru PxIN (0 – niski, 1 – wysoki).

Linie portów pierwszego P1, oraz drugiego P2 mogą pracować w trybie źródeł przerwań. Źródło przerwań konfigurują bity rejestru PxIE (0 – wyłączone, 1 – włączone). Zbocze sygnału wyzwalającego przerwanie konfigurujemy za pomocą rejestru PxIES (0 – narastające , 1 – opadające). W rejestrze PxIFG przechowywane są flagi przerwań. Flagi ustawiane są automatycznie w momencie wystąpienia przerwania (gdy źródło jest aktywne i wystąpi zbocze wyzwalające przerwanie), ale

Listing 4.1. Szablon procedury obsługi przerwania od linii I/O portu P1, P2

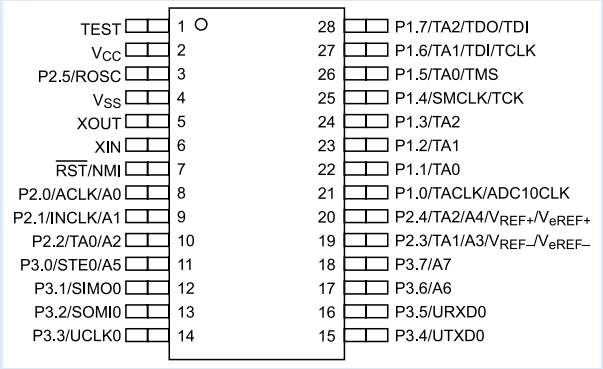
```
#pragma vector=PORT1_VECTOR     //procedura obsługi
__interrupt void Port_1(void)   //przerwania dla portu P1
{                               //
  if( P1IFG & BIT0)             //sprawdź wejście P1.0
  {                             //jeśli przerwanie od P1.0 to
    // instrukcje               //wykonaj kod obsługi zdarzenia
    P1IFG &=~ BIT0;             //wyzeruj flagę przerwania
  }                             //
 // ...                         //analogicznie dla pozostałych 7
}                               //wejść portu P1: 1,2,3,4,5,6,7

#pragma vector=PORT2_VECTOR     //procedura obsługi
__interrupt void Port_2(void)   //przerwania dla portu P2
{                               //
                                //
    // ...                      //obsługa jak dla portu P1
}                               //
```

Tabela 1. Rejestry konfiguracyjne portów wejścia-wyjścia MSP430f1232

Nazwa Rejestru. (*)	Opis	Prawa Dostępu.	bit	
			"0"	"1"
PxSEL	tryb pracy	odczyt/zapis	wejścia-wyjścia	funkcyjny
PxDIR	kierunek linii	odczyt/zapis	wyjście	wejście
PxIN	stan wejścia	tylko odczyt	stan niski	stan wysoki
PxOUT	stan wyjścia	odczyt/zapis	stan niski	stan wysoki
PxIE	obsługa przerwań	odczyt/zapis	włączone	wyłączone
PxIES	wybór zbocza sygnału wyzwalającego przerwanie	odczyt/zapis	zbocze narastające (zmiana stanu z niskiego na wysoki)	zbocze opadające (zmiana stanu z wysokiego na niski)
PxIFG	flaga przerwania	odczyt/zapis	brak reakcji	wywołanie procedury obsługi przerwania

* znak „x" w nazwie rejestru oznacza, że opis dotyczy dowolnego z rejestrów portu P1/P2/P3
* po włączeniu zasilania rejestry PxSEL, PxDIR, PxIE, PxIFG są zerowane (linie mikrokontrolera pracują w trybie we-wy i są ustawione w kierunku wejścia)



Rysunek 1. Rozmieszczenie wyprowadzeń mikrokontrolera MSP430f1232

mogą być także ustawiane przez programistę. Przerwania obsługiwane są przez dwa wektory przerwań (jeden dla portu P1, drugi dla portu P2). W procedurze obsługi przerwania sprawdzając, który bit w rejestrze PxIFG został ustawiony możemy określić, które wejście wywołało przerwanie. Ponieważ flaga przerwania nie jest czyszczona automatycznie, to musimy wyzerować ją samodzielnie. Szablon procedury obsługi przerwania dla portu P1, oraz P2 umieszczono na **listingu 1**. Informacje o rejestrach MSP430f1232 podaje **tabela 2**.

Sterowanie wyjściem

W pierwszym prezentowanym przykładzie *„Sterowanie diodą RGB”, będziemy sterować wyjściami mikrokontrolera.* Żeby zaobserwować zmiany stanów logicznych na liniach, będziemy sterować wyjściami, do których dołączono trójkolorową diodę świecącą RGB. W module „Komputerek” dioda RGB jest konfigurowana za pomocą zworek JP2, JP3, JP4. Żeby dołączyć diodę do linii I/O mikrokontrolera należy zworki ustawić w pozycji LED (linia P2.3 – dioda koloru czerwonego, linia P2.4 – dioda koloru zielonego, linia P2.5 – dioda koloru niebieskiego). Pozostałe zworki należy ustawić w pozycji IO/Off, a zworki JP7, JP8 dołączające rezonator kwarcowy do źródła zegarowego LFXT1, należy zdjąć.

Kod programu realizującego sterowanie diodą RGB prezentujemy na **listingu 4.2.** W pierwszych liniach programu dołączane są pliki nagłówkowe. Następnie definiowana jest stała *TAKTY_ZEGARA_1_SEK*. Wartość przypisana do stałej obliczana jest ze **wzoru 4.2** i określa ilość taktów zegara CPU, których czas wykonania trwa około sekundy.

$$xTaktyCPU = tCzekaj \times fMCLK \quad (4.1)$$

gdzie:
tCzekaj – obliczany czas trwania opóźnienia (s)
fMCLK – częstotliwości sygnału zegarowego MCLK, taktującego CPU (Hz)

W dalszej części programu obliczona wartość jest używana jako argument funkcji *__delay_cycles()* i służy do generowania programowego opóźnienia o czasie trwania 3 i 6 sekund. W programie głównym zatrzymujemy pracę układu Watchdog, oraz ustawiamy kierunek linii I/O. W MSP430 żeby zmniejszyć pobór prądu mikrokontrolera należy linie I/O, do których nie są dołączone urządzenia peryferyjne przełączyć w tryb pracy wejścia-wyjścia, a kierunek linii ustawić jako wyjście.

Konfiguracja używanych linii uzależniona jest od działania urządzenia peryferyjnego. W module „Komputerek” linie P1.0, P1.1 dołączone są na stałe do przycisków SW1, SW2, a linia P3.4 do układu nadawczego TX transmisji szeregowej UART. Linie te konfigurujemy jako wejścia. Linie do sterowania diodą RGB również konfigurujemy jako wejścia (tymczasowo). Pozostałe linie portów P1, P2, P3 konfigurujemy jako wyjścia. W kolejnych instrukcjach programu konfigurujemy linie sterujące diodą RGB. Linie zerujemy, a następnie zmieniany kierunek linii z wejścia na wyjście (po zmianie kierunku na linie zostaną wyzerowane – diody LED będą zgaszone).

W pętli głównej programu rozpoczynamy sterowanie diodą RGB. Najpierw włączamy i wyłączamy diodę koloru czerwonego. Następnie w analogiczny sposób sterujemy diodami koloru zielonego i niebieskiego. Na zakończenie pętli głównej programu jednocześnie włączamy, a następnie wyłączamy diody koloru czerwonego i zielonego. W efekcie zmieszania barw otrzymujemy światło koloru żółtego.

Listing 4.2. Sterowanie diodą RGB

```
//”MSP430 w przykładach”
//Przykład 4.1. Sterowanie diodą RGB
                          //pliki nagłówkowe
#include „io430.h”  //rejestry procesora
#include “intrinsics.h”  //instrukcje procesora
                          //definicje
#define  TAKTY_ZEGARA_1_SEK  740000  //opóźnienie około 1 sekunda

void main( void )  // program główny
{
  WDTCTL = WDTPW + WDTHOLD;   //zatrzymaj układ Watchdog
  //ustaw kierunek linii we-wy (SW1, SW2, TX, LEDx) - wejścia
  P1DIR  = 0xfc;  P2DIR = 0xc7;  P3DIR = 0xef;
  //konfigurowanie linii sterujących RGB
  P2OUT &=~ (BIT5 + BIT4 + BIT3);  //poziom niski
  P2DIR |=  (BIT5 + BIT4 + BIT3);  //kierunek wyjście (diody wyłączone)
  while(1)  // pętla główna programu
  {
    P2OUT |=  BIT3;  // włącz diodę LED1, (kolor czerwony)
    __delay_cycles(TAKTY_ZEGARA_1_SEK * 3);  //czekaj ok. 3 s
    P2OUT &=~ BIT3;  //wyłącz diodę LED1
    P2OUT |=  BIT4;  //włącz diodę LED2, (kolor zielony)
    __delay_cycles(TAKTY_ZEGARA_1_SEK * 6);  //czekaj ok. 6 s
    P2OUT &=~ BIT4;  //wyłącz diodę LED2
    P2OUT |=  BIT5;  //włącz diodę LED3, (kolor niebieski)
    __delay_cycles(TAKTY_ZEGARA_1_SEK * 3);  //czekaj ok. 3 s
    P2OUT &=~ BIT5;  //wyłącz diodę LED3
    //włącz diody LED1, LED2
    P2OUT |=  (BIT4 + BIT3 );  //(kolory czerwony + zielony = żółty)
    __delay_cycles(TAKTY_ZEGARA_1_SEK * 6);  //czekaj ok. 6 s
    P2OUT &=~ (BIT4 + BIT3);  //wyłącz diody LED1, LED2
  };
}
```

Obsługa wejścia

W drugim prezentowanym przykładzie *„Dzwonek do drzwi”*, będziemy odczytywać stan wejścia mikrokontrolera. Przykład zaprezentujemy w dwóch wersjach. W pierwszej stan wejścia odczytywany będzie cyklicznie, w drugiej przy wykorzystaniu przerwań. W obu przypadkach obsługiwana będzie linia numer 1 portu pierwszego P1, do której w module „Komputerek” dołączony jest przycisk SW1. Wciśniecie przycisku (dzwonka) sygnalizowane będzie dźwiękiem generowanym przy użyciu zamontowanego na płycie modułu brzęczyka. Żeby podłączyć brzęczyk do linii we-wy mi-

krokontrolera zworkę JP6 należy ustawić w pozycji Spk. Pozostałe zworki modułu należy ustawić w pozycji IO/Off, a zworki JP7, JP8 dołączające rezonator kwarcowy do źródła zegarowego LFXT1 należy zdjąć.

Kod programu, w wersji pierwszej prezentujemy na **listingu 4.3.** W pierwszych liniach programu zatrzymywany jest układ Watchdog. Następnie linia P1.1 do której dołączony jest przycisk SW1 jest przełączana w tryb wyjścia. Aby zmniejszyć poboru prądu, linia P1.0, do której jest dołączony przycisk SW2 oraz linia P3.4 transmisji szeregowej UART, również są skonfigurowane jako wejścia. Pozostałe linie ustawiane są w kierunku wyjścia.

W pętli głównej programu cyklicznie sprawdzamy wartość bitu numer 1 w rejestrze P1IN. Jeśli bit ma wartość „0", co oznacza, że na linii jest stan niski (przycisk SW1 jest wciśnięty) to rozpoczynamy sterowanie brzęczykiem. Stan linii sterującej brzęczykiem zmieniamy na przeciwny, oraz generujemy opóźnienie trwające około 1 milisekundy. Przytrzymanie przycisku przez czas dłuższy niż wprowadzone opóźnienie spowoduje, że w kolejnej pętli programu ponownie zostanie wykryte wciśnięcie przycisku, a stan na linii sterującej brzęczykiem zmieni się na przeciwny. Wówczas na linii sterującej brzęczykiem pojawi się sygnał prostokątny, a brzęczyk (dzwonek) zacznie generować dźwięk.

Kod programu, w wersji drugiej zamieszczono na **listingu 4.4.** Analogicznie jak w wersji pierwszej, zatrzymujemy pracę układu Watchdog i konfigurujemy linie I/O. Następnie konfigurujemy wejście P1.1 (przycisk SW1) do pracy w trybie przerwań. Najpierw ustawiamy zbocze sygnału wyzwalającego przerwanie. Ponieważ program ma reagować na wciśnięcie przycisku SW1 (zmiana poziomu na wejściu z wysokiego na niski), to ustawiamy zbocze opadające. Następnie aktywujemy przerwania od linii P1.1. W kolejnych instrukcjach włączamy obsługę przerwań, oraz usypiamy mikrokontroler. MSP430f1232 wprowadzany jest w tryb uśpienia LPM4, w którym wyłączone są źródła DCO, LFXT1 oraz jest zatrzymane CPU. Sygnały zegarowe MCLK, SMCLK, ACLK są wyłączone, a w procesorze odświeżana jest tylko zawartość pamięci RAM. Mikrokontroler przebywa w trybie uśpienia LPM4, do momentu wciśnięcia przycisku SW1. Wówczas poziom sygnału na wejściu P1.1 zmienia się z wysokiego na niski, spełnione są warunki dotyczące zbocza sygnału wyzwalającego przerwanie, ustawiana jest flaga przerwania, a program przechodzi do procedury obsługi przerwania.

W procedurze obsługi przerwania zerujemy flagę przerwania, a następnie wykonujemy instrukcję __bic_SR_register_on_exit (po wyjściu z procedury obsługi przerwania nie wracaj do trybu uśpienia LPM4, rozpocznij wykonanie pętli głównej programu). W kolejnych instrukcjach pętli głównej programu odczytywany jest stan przycisku i dopóki przycisk jest wciśnięty, na wyjście P1.3 podawany jest sygnał prostokątny, sterujący brzęczykiem. Puszczenie przycisku przerywa sterowanie wyjściem P1.3 i tym samym generowanie dźwięku, a mikrokontroler przechodzi w tryb uśpienia LPM4. W trybie uśpienie MSP430f1232 pozostaje do momentu, ponownego wciśnięcia przycisku SW1.

Niezależnie od tego, czy moduł „Komputerek" zaprogramujemy pierwszą czy drugą wersją oprogramowania, po naciśnięciu przycisku SW1 będzie generо-

Konfigurowanie rejestrów MSP430. Operacje bitowe.
Mikrokontroler MSP430 konfigurujemy modyfikując wartości bitów w rejestrach procesora. W jaki sposób ustawić, wyzerować, odczytać, oraz zmienić stanu bitu na przeciwny przećwiczymy na przykładzie rejestru kierunku portu P1, czyli P1DIR.
Instrukcja P1DIR = 0x11 (00010001b) ustawia bity 5 i 0 (wyjścia) oraz zeruje pozostałe (wejścia). Jeśli chcemy ustawić bity 5 i 0 oraz nie zmieniać wartości pozostałych bitów, to korzystamy z operatora bitowego „|" (or – suma). Instrukcja ustawienia bitów wygląda następująco:
P1DIR | = 0x11;
Jeśli chcemy wyzerować bity 5 i 0, a pozostałe bity pozostawić bez zmian, korzystamy z operatorów bitowych „&" (and - mnożenie), oraz „~" (not – negacja). W omawianym przypadku instrukcja zerowania bitów wygląda następująco:
P1DIR & = ~0x11;
Jeśli chcemy wartość bitów 5 i 0 zmienić na przeciwną, a pozostałe bity pozostawić bez zmian, to korzystamy z operatora bitowego „^" (xor - suma do dwóch). Instrukcja zmiany bitów wygląda następująco:
P1DIR ^= 0x11;
Jeśli chcemy sprawdzić wartość bitu, to korzystamy z funkcji „if" oraz operatora bitowego „&". Instrukcja sprawdzenia, czy bit 0 jest ustawiony:
if(P1DIR & 0x01) //(true – bit ustawiony)
Instrukcja sprawdzenia, czy bit 0 jest wyzerowany:
if(! (P1DIR & 0x01)) //(true – bit wyzerowany)
W środowisku IAR, umieszczono definicje bitów mikrokontrolera (BIT0, BIT1, ..., BIT7). Jeśli skorzystamy z definicji bitów to instrukcje ustawiania, zerowania, zmiany stanu, odczytu wartości bitów będą bardziej czytelne. W naszym wypadku instrukcje ustawienia, oraz zerowania bitów 5 i 0 wyglądają następująco:
P1DIR |= (BIT5 + BIT0);
P1DIR &=~ (BIT5 + BIT0);

wany sygnał dźwiękowy. Różnice w działaniu modułu zaobserwujemy mierząc pobór prądu. Otóż w wersji pierwszej mikrokontroler cyklicznie sprawdza stan wejścia permanentnie przebywając w trybie aktywnym AM i pobiera około 204 μA prądu. W drugiej wersji programu mikrokontroler jest uśpiony w trybie LPM4, w którym pobór prądu wynosi około 0,5 μA. W czasie, gdy generowany jest sygnał dźwiękowy, to w pierwszym wypadku pobór prądu wynosi około 318 μA, w drugim około 338 μA. Zakładając, że w ciągu doby przycisk (dzwonek) będzie wciskany 20 razy i za każdym razem na czas jednej minuty, to w wypadku programu z obsługą przerwań pobór prądu będzie 40 razy mniejszy!!! Zasilając moduł „Komputerek" z baterii CR2032 o pojemności 240 mAh, w pierwszym wypadku na jednym komplecie baterii moduł będzie pracował niecałe 49 dni, natomiast w drugim ponad 5 lat (wzór 2, EP11/12). W energooszczędnych aplikacjach powinnyśmy zatem wejścia mikrokontrolera obsługiwać w trybie przerwań.

Listing 4.3. Dzwonek do drzwi. Wersja 1
```
//"MSP430 w przykładach"
//Przykład 4.2. Dzwonek do drzwi. Wersja 1.
                        // pliki nagłówkowe
#include „io430.h"       // rejestry procesora
#include "intrinsics.h"  // instrukcje procesora

void main( void )  // program główny
{
  WDTCTL = WDTPW + WDTHOLD;  //zatrzymaj układ Watchdog
  // ustaw kierunek linii we-wy, (P1.1, P1.0, P3.4) - wejścia
  P1DIR = 0xfc;  P2DIR = 0xff;  P3DIR = 0xef;
  while(1)  //pętla główna programu
  {
    if(!(P1IN & BIT1))  //sprawdź stanu przycisku SW1,
    {  //gdy wciśnięto przycisk
       P1OUT ^= BIT3;  //zmień na przeciwny poziom napięcia
       __delay_cycles(740);  //na linii sterującej brzęczykiem
    }; //(generuj sygnał dźwiękowy)
  };
}
```

Listing 4.4. Dzwonek do drzwi. Wersja 2
```
//"MSP430 w przykładach"
//Przykład numer 4.2. Dzwonek do drzwi. Wersja 2.
                       // pliki nagłówkowe
#include „io430.h"      // rejestry procesora
#include "intrinsics.h" // instrukcje procesora

void main( void )  //program główny
{
  WDTCTL = WDTPW + WDTHOLD;  //zatrzymaj układ Watchdog
  //ustaw kierunek linii we-wy (P1.1, P1.0, P3.4) - wejścia
  P1DIR = 0xfc;  P2DIR = 0xff;  P3DIR = 0xef;
  //konfigurowanie linii P1.1/SW1
  P1IES |= BIT1;  //ustaw zbocze wyzwalania 1->0
  P1IE |= BIT1;  //włącz źródło przerwań
  __bis_SR_register(GIE);  //odblokuj obsługę przerwań
  while(1)  //pętla główna programu
  {
    //tryb uśpienia LPM4, źródła: DCO, LFXT1 są wyłączone
    //sygnały zegarowe: MCLK,ACLK,SMCLK są wyłączone,
    //tylko odświeżanie RAM
    __bis_SR_register(CPUOFF + SCG0 + SCG1 + OSCOFF);
    while(!(P1IN & BIT1))
    {  // dopóki przycisk SW1 jest wciśnięty
      P1OUT ^= BIT3;  // generuj sygnał dźwiękowy
      __delay_cycles(740);
    };
  };
}

#pragma vector=PORT1_VECTOR  //procedura obsługi przerwania
__interrupt void Port_1(void)  //dla linii we-wy portu P1
{
    if(P1IFG & BIT1)  //jeśli przerwanie od P1.1
    {
        P1IFG &=~ BIT1;  //zeruj flagę przerwania
        __bic_SR_register_on_exit(CPUOFF + SCG0 + SCG1 + OSCOFF
);
    }  //po wyjściu z procedury obsługi
}  //przerwania, wyjdź z trybu LPM4
```

Biblioteka obsługi LCD

Specjalnie na potrzeby kursu autor przygotował bibliotekę obsługi alfanumerycznego wyświetlacza LCD ze sterownikiem Hitachi HD44780. Biblioteka, wraz z opisem działania została umieszczona w materiałach dodatkowych na CD. W przykładzie *„Obsługa wyświetlacza LCD"* zademonstrowano praktyczne użycie procedur zdefiniowanych w bibliotece. Przy wykorzystaniu procedur jest konfigurowany interfejs komunikacyjny wyświetlacza, wywołana sekwencja startowa oraz inicjowany sterownik. Następnie, w pamięci CG_RAM definiowany jest znak „ą" i na ekranie LCD jest wypisywana sentencję autorstwa Erazma z Rotterdamu, o następującej treści: „Dla chcącego nie ma nic trudnego". Sentencja wypisywana jest przy użyciu procedury LcdPiszTekst.

Znak o kodzie „ą" nie jest zdefiniowany w pamięci CG_ROM wyświetlacza, więc do wyświetlenia znaku używana jest definicja znaku z pamięci CG_RAM. Kursor wyświetlacza jest ustawiany w miejscu wyświetlenia znaku „ą" i przy użyciu procedury LcdPiszZnak znak jest wyświetlany na ekranie. Po wyświetleniu tekstu sentencji następuje 3-sekundowa sekwencja sterowania podświetlaniem ekranu. Na wyjście P1.2 mikrokontrolera jest podawany sygnał prostokątny o częstotliwości 100 Hz i wypełnieniu regulowanym parametrem JASNOSC (domyślnie 20%). Po zakończeniu sterowania podświetlaniem ekranu mikrokontroler wprowadzany jest w tryb uśpienia LPM4.

Łukasz Krysiewicz, EP