Programowanie mikrokontrolerów Xmega (5)

System zdarzeń

W mikrokontrolerach XMEGA układów peryferyjnych jest bardzo dużo, ale rdzeń procesora jest tylko jeden. Mogłoby się okazać, że rdzeń nie jest w stanie przetwarzać danych wysyłanych przez peryferia. Projektanci układów XMEGA udostępnili narzędzia, pozwalające peryferiom komunikować się ze sobą zupełnie bez pośrednictwa rdzenia – są to DMA oraz system zdarzeń. W tym artykule zajmiemy się zdarzeniami, a DMA zostanie omówione w kolejnych częściach.

Musimy dokładnie zrozumieć różnicę między przerwaniem, a zdarzeniem. Jest to niezbędne każdemu, kto kiedykolwiek będzie zajmował się programowaniem mikrokontrolerów.

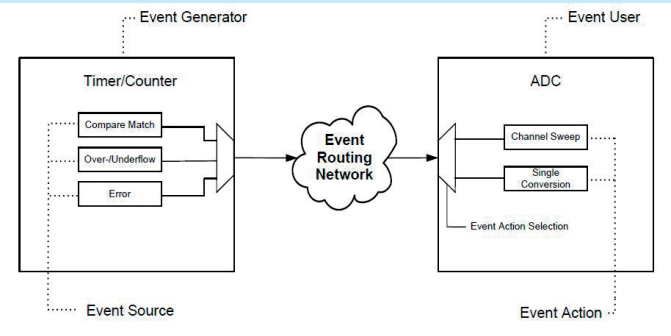
Zdarzenie a przerwanie

Po zgłoszeniu przerwania, procesor przerywa dotychczas wykonywane operacje, przechodzi do procedury obsługi przerwania, po czym wraca do dotychczas wykonywanego programu. Zdarzenie może zostać zrealizowane **bez jakiejkolwiek reakcji ze strony rdzenia procesora**. Służy do tego, aby jeden układ peryferyjny mógł sterować innym układem. Przykładem zdarzenia może być przepełnienie timera, które inicjuje pomiar przetwornikiem analogowo-cyfrowym. Innym zdarzeniem może być zmiana stanu komparatora analogowego, która jest rejestrowana przez timer.

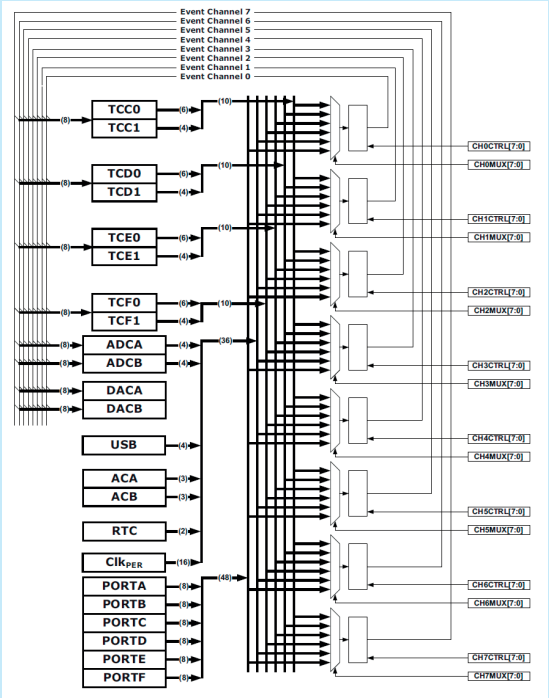
Zdarzenie może wygenerować układ zwany nadajnikiem. Można zdarzenie generować również programowo. Do każdego nadajnika możemy podłączyć jeden lub kilka odbiorników. Większość peryferiów dostępnych w XMEGA może być zarówno nadajnikami i odbiornikami. Nic nie stoi na przeszkodzie, by odbiornik jednego zdarzenia był nadajnikiem kolejnego. W ten sposób można łączyć timery w łańcuszek i można stworzyć timer 64-bitowy lub jeszcze dłuższy. Przykład nadajnika i odbiornika przedstawiono na **rysunku 1**.

Ważną zaletą zdarzenia nad przerwaniem jest duża szybkość reakcji. Wystarczą zaledwie dwa cykle zegarowe od wygenerowania zdarzenia do jego realizacji przez odbiornik. W przypadku przerwania – samo wejście do procedury zajmuje dużo więcej czasu. Zdarzenia mają w **100% przewidywalny czas realizacji** i jak już wspomniano, nie angażują procesora, więc może zajmować się innymi zadaniami. Mało tego, **rdzeń procesora można nawet wyłączyć**, wprowadzając go w stan uśpienia! W związku z tym zastosowanie systemu zdarzeń może radykalnie zmniejszyć zużycie energii.

Nadajniki i odbiorniki połączone są ze sobą przy pomocy sieci połączeń (*rounting network*) przedstawionej na **rysunku 2**. W przypadku mikrokontrolera ATxmega128A3U składa się ona z ośmiu kanałów, które doprowadzone są do wszystkich peryferiów, mogących funkcjonować jako odbiorniki zdarzeń. Na „początku" każdego kanału jest multiplekser, czyli elektroniczny przełącznik, którym można ustawić, jaki układ peryferyjny ma być nadajnikiem zdarzenia. W ten sposób możemy budować sieć połączeń według własnych potrzeb (jeszcze trochę i będzie jak w FPGA…). Choć może wydawać



Rysunek 1. Timer i przetwornik ADC połączone systemem zdarzeń



Rysunek 2. Sieć połączeń (*rounting network*)

się to bardzo skomplikowane, w praktyce wykorzystanie systemu zdarzeń jest trywialnie proste.

System zdarzeń nie jest wyłącznie biernym pośrednikiem i siecią połączeń. Umożliwia także kilka ciekawych funkcji, takich jak **filtr cyfrowy** oraz **dekoder kwadraturowy**. Filtr umożliwia likwidację szumów, co przydatne jest w sytuacji, gdy nadajnikiem zdarzenia jest pin procesora połączony np. z przyciskiem. Dekoder kwadraturowy umożliwia bardzo proste podłączanie enkoderów obrotowych, takich jak w pokrętłach oscyloskopów. Dzięki całkowicie sprzętowej obsłudze, zastosowanie enkodera w programie staje się równie proste jak podłączenie klawiatury.

Proste zdarzenia

Na początek sprawa bardzo prosta – zwyczajne zliczanie impulsów. Zdarzenie będzie generował pin E5, czyli ten, który jest połączony z przyciskiem FLIP na płytce X3--DIL64. Odbiornikiem zdarzenia będzie timer C0, który będzie zwiększał swoją wartość o 1 przy każdym wciśnięciu przycisku. Schemat układu do zbudowania na płytce stykowej przedstawiono na **rysunku 3**. Przejdźmy od razu do pisania programu!

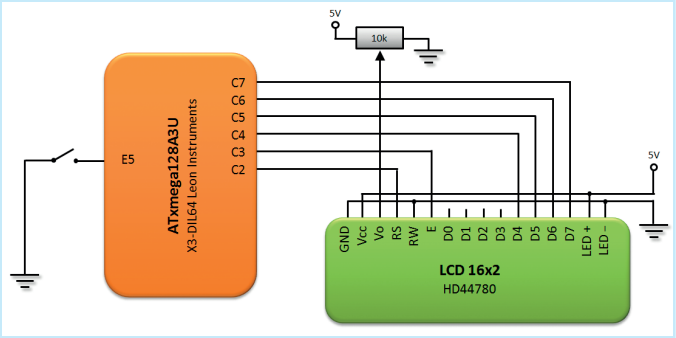
Zaczynamy, jak to zwykle, od skonfigurowania pinów. Zdarzenie ma generować pin E5, dlatego musimy go skonfigurować jako wejście oraz uaktywnić rezystor podciągający do zasilania. Czytelnicy pamiętający artykuł o przerwaniach w Xmega (EP 12/2013) zapewne zwrócą uwagę, że poniższy kod niczym nie różni się, od tego, który uaktywnia przerwania. W rzeczy samej! Parametr PORT_ISC_FALLING_gc wpisany do rejestru PINxCTRL konfiguruje, co ma wywoływać przerwania, jak i również zdarzenia.

Pamiętajmy, że żeby uaktywnić przerwanie, trzeba było jeszcze skonfigurować kontroler przerwań PMIC oraz odblokować przerwania, które chcemy używać, a na końcu trzeba jeszcze wykonać instrukcję sei(). Jednak my tego nie zrobimy. Mamy przecież zająć się systemem zdarzeń.

Wykorzystajmy kanał 0 systemu zdarzeń. W procesorze ATxmega128A3U, dostępne jest osiem kanałów zdarzeń i moglibyśmy użyć dowolnego z nich. Musimy wskazać, jaki układ ma być nadajnikiem zdarzenia i wpisać go do rejestru EVSYS.CH0MUX. Rzućmy okiem na dokumentację, której fragment przedstawiono na **rysunku 4**.

Jak widać, wybór jest bardzo szeroki i bardzo pomocne są podpowiedzi, generowane automatycznie przez Atmel Studio podczas wpisywania tekstu. Aby zdarzenie generował pin E5, musimy do rejestru CH0MUX wpisać wartość EVSYS_CHMUX_PORTE_PIN5_gc.

Ponieważ przycisk jest elementem mechanicznym, generującym chaotyczne wielokrotne impulsy podczas przyciskania, dobrym pomysłem będzie zastosowanie filtru cyfrowego. W przeciwnym razie, procesor mógłby pojedyncze wciśnięcie zinterpretować jako kilka, kilkanaście, lub nawet kilkadziesiąt wciśnięć. Filtr cyfrowy powoduje ignorowanie impulsów trwających krócej niż wyznaczoną liczbę cykli zegara systemowego. Maksymalna wartość, jaką możemy wybrać to 8 cykli zegarowych – prawdę mówiąc, nie jest to ilość wystarczająca do całkowitego wyeliminowania problemu, ale choć trochę go zmniejszymy. Aby uruchomić filtr cyfrowy, do rejestru CH0CTRL musimy wpisać wartość EVSYS_DIGFILT_8SAMPLES_gc.



Rysunek 3. Schemat układu demonstrującego podstawowe funkcje systemu zdarzeń

Table 6-3. CHnMUX[7:0] bit settings.

CHnMUX[7:4]	CHnMUX[3:0]	Group configuration	Event source
0000	0 0 0 0		None (manually generated events only)
0000	0 0 0 1		(Reserved)
0000	0 0 1 X		(Reserved)
0000	0 1 X X		(Reserved)
0000	1 0 0 0	RTC_OVF/RTC32_OVF	RTC overflow / RTC32 overflow
0000	1 0 0 1	RTC_CMP	RTC compare match
0000	1 0 1 0		USB start of frame on CH0 (see Table 6-4 on page 78) USB error on CH1 (see Table 6-4 on page 78) USB overflow on CH2 (see Table 6-4 on page 78) USB setup on CH3 (see Table 6-4 on page 78)
0000	1 0 1 X		(Reserved)
0000	1 1 X X		(Reserved)
0001	0 0 0 0	ACA_CH0	ACA channel 0
0001	0 0 0 1	ACA_CH1	ACA channel 1
0001	0 0 1 0	ACA_WIN	ACA window
0001	0 0 1 1	ACB_CH0	ACB channel 0
0001	0 1 0 0	ACB_CH1	ACB channel 1
0001	0 1 0 1	ACB_WIN	ACB window
0001	0 1 1 X		(Reserved)
0001	1 X X X		(Reserved)
0010	0 0 n	ADCA_CHn	ADCA channel n (n =0, 1, 2 or 3)
0010	0 1 n	ADCB_CHn	ADCB channel n (n=0, 1, 2 or 3)
0010	1 X X X		(Reserved)
0011	X X X X		(Reserved)
0100	X X X X		(Reserved)
0101	0 n	PORTA_PINn(1)	PORTA pin n (n= 0, 1, 2 ... or 7)
0101	1 n	PORTB_PINn(1)	PORTB pin n (n= 0, 1, 2 ... or 7)
0110	0 n	PORTC_PINn(1)	PORTC pin n (n= 0, 1, 2 ... or 7)
0110	1 n	PORTD_PINn(1)	PORTD pin n (n= 0, 1, 2 ... or 7)
0111	0 n	PORTE_PINn(1)	PORTE pin n (n= 0, 1, 2 ... or 7)
0111	1 n	PORTF_PINn(1)	PORTF pin n (n= 0, 1, 2 ... or 7)
1000	M	PRESCALER_M	Clk_{PER} divide by 2^M (M=0 to 15)
1001	X X X X		(Reserved)
1010	X X X X		(Reserved)
1011	X X X X		(Reserved)
1100	0 E	See Table 6-4	Timer/counter C0 event type E
1100	1 E	See Table 6-4	Timer/counter C1 event type E
1101	0 E	See Table 6-4	Timer/counter D0 event type E
1101	1 E	See Table 6-4	Timer/counter D1 event type E
1110	0 E	See Table 6-4	Timer/counter E0 event type E
1110	1 E	See Table 6-4	Timer/counter E1 event type E
1111	0 E	See Table 6-4	Timer/counter F0 event type E
1111	1 E	See Table 6-4	Timer/counter F1 event type E

Notes: 1. The description of how the ports generate events is described in "Port Event" on page 144.
2. The different USB events can be selected for only event channel, 0 to 3.

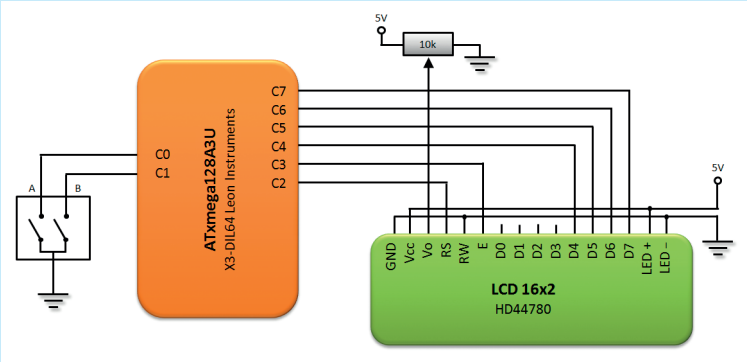
Rysunek 4. Wybór nadajnika zdarzenia

To wszystko, jeśli chodzi o nadajnik. Przejdźmy teraz do konfiguracji odbiornika zdarzenia, czyli timera C0. Timery w mikrokontrolerach XMEGA zostały dokładniej opisane w poprzedniej części kursu w EP 02/1024.

Dotychczas w naszych ćwiczeniach, timer był zawsze taktowany sygnałem zegarowych podzielonym wstępnie przez preskaler, co konfigurowaliśmy w rejestrze CTRLA timera. Z punktu widzenia timera, system zdarzeń jest takim samym sygnałem taktującym jak sygnał zegarowy – impulsy przychodzące przez system zdarzeń mogą być traktowane przez niego jak sygnał zegarowy. Dlatego wybór kanału systemu zdarzeń dokonujemy również w rejestrze CTRLA.

Listing 1. Program demonstrujący podstawowe funkcje systemu zdarzeń

```
#define  F_CPU    2000000UL
#include <avr/io.h>
#include <util/delay.h>
#include „hd44780.h”

int main(void) {

    // przyciski
    PORTE.DIRCLR    =    PIN5_bm;                    // pin E5 jako wejście (przycisk FLIP)
    PORTE.PIN5CTRL  =    PORT_OPC_PULLUP_gc|         // podciągnięcie do zasilania
                         PORT_ISC_FALLING_gc;        // zdarzenie mam wywoływać zbocze malejące

    // konfiguracja systemu zdarzeń
    EVSYS.CH0MUX    =    EVSYS_CHMUX_PORTE_PIN5_gc;  // pin E5 wywołuje zdarzenie
    EVSYS.CH0CTRL   =    EVSYS_DIGFILT_8SAMPLES_gc;  // filtr cyfrowy

    // konfiguracja timera
    TCC0.CTRLB      =    TC_WGMODE_NORMAL_gc;        // tryb normalny
    TCC0.CTRLA      =    TC_CLKSEL_EVCH0_gc;         // ustawienie źródła sygnały na kanał 0 systemu zdarzeń

    // wyświetlacz
    LcdInit();

    while(1) {
        // wyświetlenie aktualnej wartości licznika CNT
        // CNT = ...
        LcdClear();
        Lcd(„CNT = „);
        LcdDec(TCC0.CNT);
        _delay_ms(100);                              // czekanie 100ms
    }
}
```



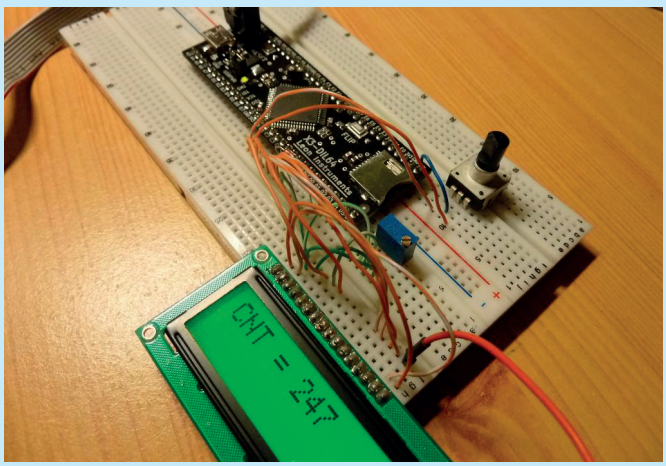
Rysunek 5. Schemat układu z enkoderem obrotowym

…i to wszystko! Pozostaje tylko inicjalizacja wyświetlacza LCD (EP 12/2014) oraz procedura pętli głównej, cyklicznie wyświetlająca aktualną wartość rejestru CNT timera C0. Zwróć uwagę, że w pętli głównej nie ma jakiejkolwiek instrukcji związanej z pinem E5.

Kod pierwszego programu demonstrującego możliwości systemu zdarzeń przedstawiono na **listingu 1**.

Enkoder obrotowy

Enkodery obrotowe, zwane również impulsatorami, służą do budowy przyjaznych interfejsów człowiek-komputer. Istnieje wiele parametrów, które łatwiej jest regulować kręcąc pokrętłem niż wpisując wartości liczbowe z klawiatury. W odbiornikach radiowych do regulacji głośności dawniej stosowano potencjometry, a współcześnie, w dobie techniki cyfrowej, stosuje się enkodery. Urządzenie, w którym znajdziemy kilka lub nawet kilkanaście enkoderów to oscyloskop. Mikrokontrolery XMEGA umożliwiają bardzo łatwe wykorzystanie enkoderów, dzięki systemowi zdarzeń i wbudowanemu dekoderowi kwadraturowemu. Schemat układu demonstracyjnego z enkoderem przedstawia **rysunek 5**, a jego przykład wykonania na płytce stykowej przedstawia **fotografia 6**.



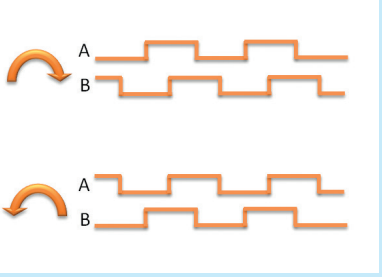
Fotografia 6. Prototyp na płytce stykowej

Enkoder ma piny oznaczone literami A i B oraz masę. Jeśli enkoder ma możliwość wciskania, wówczas są jeszcze dwie dodatkowe nóżki. Piny A i B muszą być podciągnięte do zasilania rezystorami pull-up – na szczęście XMEGA jest wyposażona w takie i nie musimy dodawać osobnych. Wewnątrz niego znajdują się dwa styki, które są zwierane do masy w zależności od pozycji pokrętła. Obracając pokrętło uzyskamy przebiegi takie, jak na **rysunku 7** – przesunięte o 90° w zależności od kierunku obrotu. Dzięki sprzętowemu dekoderowi, jaki znajduje się w mikrokontrolerach XMEGA, nie musimy wgłębiać się w szczegóły i wnikliwie badać tych przebiegów. Dekoder zrobi wszystko za nas, a pozycję pokrętła będziemy mogli łatwo i szybko odczytać, korzystając z rejestru CNT timera.

Program przedstawiono na **listingu 2**. Na początku funkcji *main()*, musimy skonfigurować piny, które będzie wykorzystywał enkoder. Ustawiamy je jako wejścia i włączamy wewnętrzne rezystory pull-up. Ponieważ piny wykorzystywane przez enkoder znajdują się w obrębie jednego portu, do ich konfiguracji możemy wykorzystać rejestr PORTCFG.MPCMASK i zdefiniować w nim, jakie piny chcemy skonfigurować. Następnie, po wpisaniu odpowiednich wartości do rejestru PORTx.PINxCTRL (wskazującego, które konkretnie piny chcemy konfigurować w następnej instrukcji), ustawienia te zostaną skopiowane do rejestru kontrolnego każdego pinu wskazanego w MPCMASK. W ten sposób możemy szybko skonfigurować piny C1 i C0, które podłączymy do enkodera.

Listing 2. Program demonstrujący działanie dekodera kwadraturowego

```
#define  F_CPU    2000000UL
#include <avr/io.h>
#include <util/delay.h>
#include „hd44780.h”

int main(void) {

    // wejścia enkodera
    PORTCFG.MPCMASK    =    0b00000011;                  // wybór pinów 0 i 1 do konfiguracji
    PORTC.PIN0CTRL     =    PORT_ISC_LEVEL_gc |          // reagowanie na poziom niski
                            PORT_OPC_PULLUP_gc;          // podciągnięcie do zasilania

    // konfiguracja systemu zdarzeń
    EVSYS.CH0MUX       =    EVSYS_CHMUX_PORTC_PIN0_gc;   // pin C0 wywołuje zdarzenie
    EVSYS.CH0CTRL      =    EVSYS_QDEN_bm|               // włączenie dekodera w systemie zdarzeń
                            EVSYS_DIGFILT_8SAMPLES_gc;   // filtr cyfrowy

    // konfiguracja timera
    TCC0.CTRLA         =    TC_CLKSEL_EVCH0_gc;  // taktowanie systemem zdarzeń
    TCC0.CTRLD         =    TC_EVACT_QDEC_gc |   // włączenie dekodera kwadraturowego
                            TC_EVSEL_CH0_gc;     // dekoder zlicza impulsy z kanału 0 sys zdarzeń

    // wyświetlacz
    LcdInit();

    while(1) {
        // wyświetlenie aktualnej wartości licznika CNT
        // CNT = ...
        LcdClear();
        Lcd(„CNT = „);
        LcdDec(TCC0.CNT / 4);  // jeden przeskok to 4 impulsy
        _delay_ms(100);  // czekanie 100ms
    }
}
```



Rysunek 7. Przebiegi na wyjściach A i B enkodera

Następnie, przechodzimy do konfiguracji systemu zdarzeń. W rejestrze EVSYS.CH0MUX wskazujemy pierwszy z dwóch pinów, wykorzystywanych przez enkoder. Pamiętaj, że enkoder musi być podłączony do pinów sąsiadujących ze sobą!

Każdy kanał systemu zdarzeń ma swój rejestr kontrolny. Zobaczmy fragment dokumentacji tego rejestru na **rysunku 8**. Interesuje nas bit QDEN, uruchamiający dekoder kwadraturowy. Pozostałe opcje związane z dekoderem są związane z enkoderami z indeksowaniem. Przydatną rzeczą, jaką jeszcze możemy w tym rejestrze ustawić, to filtr cyfrowy. Filtr cyfrowy sprawia, że sygnał zostanie rozpoznan,y jeżeli będzie utrzymywał się na wejściu dłużej niż wybrana liczba cykli zegarowych. Ustawmy więc filtr na osiem cykli.

Dekoder kwadraturowy jest nadajnikiem zdarzenia, a odbiornikiem musi być timer, który będzie zliczał odebrane impulsy. Wykorzystajmy do tego celu timer C0. W rejestrze CTRLA, jak pamiętamy z odcinka o timerach w XMEGA, musimy w nim zdefiniować źródło sygnału taktującego. Jest nim oczywiście kanał 0 systemu zdarzeń. W rejestrze CTRLD musimy tak skonfigurować timer, by był w stanie poprawnie odbierać impulsy z dekodera. W szczególności jest to ważne, by timer wiedział kiedy ma zwiększyć swoją wartość, a kiedy zmniejszyć, w zależności od kierunku obrotu enkodera.

I to wszystko! Wystarczy teraz dopisać prostą procedurę pokazującą stan licznika na wyświetlaczu LCD. Pamiętać trzeba, że pojedynczy impuls enkodera powoduje zwiększenie lub zmniejszenie wartości timera o 4 – dlatego przed wyświetleniem wartości rejestru TCC0.CNT, musimy ją podzielić najpierw podzielić przez 4.

Dominik Leon Bieczyński
www.leon-instruments.pl

6.8.2 CHnCTRL – Event Channel n Control register

Bit	7	6	5	4	3	2	1	0
	–	QDIRM[1:0]		QDIEN	QDEN	DIGFILT[2:0]		
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – Reserved**
  This bit is reserved and will always be read as zero. For compatibility with future devices, always write this bit to zero when this register is written.
- **Bit 6:5 – QDIRM[1:0]: Quadrature Decode Index Recognition Mode**
  These bits determine the quadrature state for the QDPH0 and QDPH90 signals, where a valid index signal is recognized and the counter index data event is given according to Table 6-5 on page 79. These bits should only be set when a quadrature encoder with a connected index signal is used.These bits are available only for CH0CTRL, CH2CTRL, and CH4CTRL.

Table 6-5. QDIRM bit settings..

QDIRM[1:0]		Index recognition state
0	0	{QDPH0, QDPH90} = 0b00
0	1	{QDPH0, QDPH90} = 0b01
1	0	{QDPH0, QDPH90} = 0b10
1	1	{QDPH0, QDPH90} = 0b11

- **Bit 4 – QDIEN: Quadrature Decode Index Enable**
  When this bit is set, the event channel will be used as a QDEC index source, and the index data event will be enabled.
  This bit is available only for CH0CTRL, CH2CTRL, and CH4CTRL.
- **Bit 3 – QDEN: Quadrature Decode Enable**
  Setting this bit enables QDEC operation.
  This bit is available only for CH0CTRL, CH2CTRL, and CH4CTRL.
- **Bit 2:0 – DIGFILT[2:0]: Digital Filter Coefficient**
  These bits define the length of digital filtering used. Events will be passed through to the event channel only when the event source has been active and sampled with the same level for the number of peripheral clock cycles defined by DIGFILT.

Table 6-6. Digital filter coefficient values .

DIGFILT[2:0]	Group configuration	Description
000	1SAMPLE	One sample
001	2SAMPLES	Two samples
010	3SAMPLES	Three samples
011	4SAMPLES	Four samples
100	5SAMPLES	Five samples
101	6SAMPLES	Six samples
110	7SAMPLES	Seven samples
111	8SAMPLES	Eight samples

Rysunek 8. Dokumentacja rejestru CHxCTRL