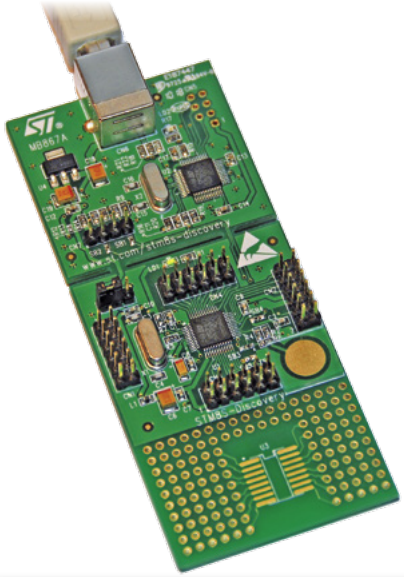
Mikrokontrolery STM8S – pierwsze kroki (2)

Mikrokontrolery STM8 są stosunkowo młodą rodziną 8-bitowych układów oferowanych przez STMicroelectronics. Charakteryzują się one wysoką wydajnością i dość bogatym wyposażeniem w układy peryferyjne. W pierwszej części artykułu, która ukazała się w poprzednim numerze EP, przedstawione zostały podstawowe cechy mikrokontrolerów STM8S, procedura tworzenia projektu w środowisku STVD oraz dwa przykładowe programy demonstrujące sposób obsługi portu GPIO, liczników uniwersalnych oraz konfigurację sygnałów taktujących. W niniejszej, drugiej części, pokazane zostanie generowanie sygnału PWM, obsługa przerwań oraz pola dotykowego.



Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 15505, pass: 27mdt418
• pierwsza część artykułu

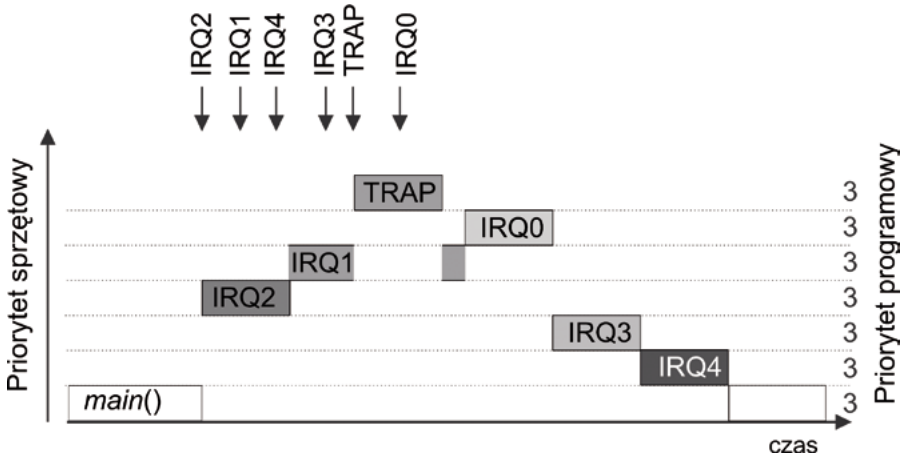
W poprzedniej części artykułu zawarte zostały dwa programy. Program 1 prezentował procedurę tworzenia projektu w środowisku *ST Visual Develop* (STVD) oraz obsługą portu GPIO. Widocznym efektem działania programu było miganie diodą znajdującą się w zestawie uruchomieniowym STM8S Discovery. Program 2 demonstrował wykorzystanie licznika TIM3 oraz sposób zmiany źródła sygnału taktującego mikrokontroler, a efektem jego działania było również miganie diodą, ale wykonywane w ściśle odmierzanych odstępach czasu. Ponadto, w pierwszej części artykułu opisano najważniejsze cechy pozostałych liczników uniwersalnych, tj. TIM1, 2 i 4.

Program 3 – generowanie sygnału PWM, przerwania

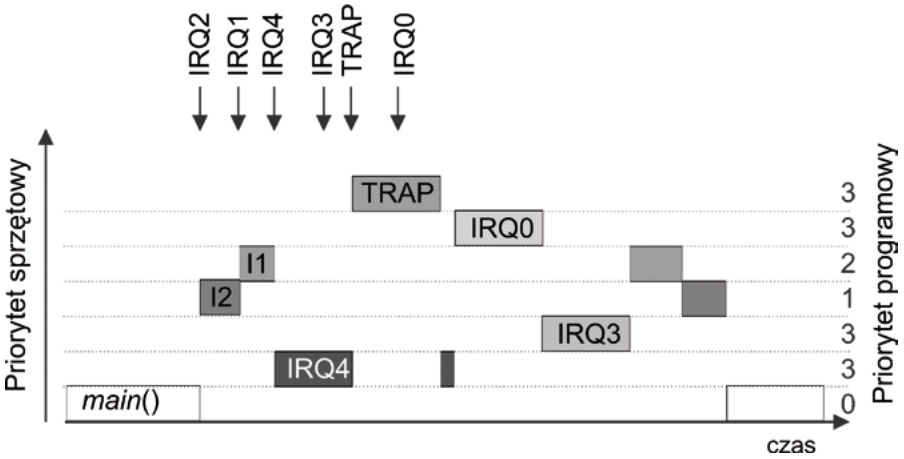
Naszym kolejnym zadaniem będzie napisanie programu, dzięki któremu dioda LED będzie pulsować, tzn. na zmianę, stopniowo zwiększać i zmniejszać jasność. W tym celu wykorzystany zostanie sygnał PWM, a do jego generowania wykorzystany zostanie licznik TIM3. Aby zmieniać jasność diody konieczne będzie modyfikowanie wypełnienia sygnału PWM, do czego wykorzystana zostanie procedura obsługi przerwania zgłaszanego przez licznik TIM1, który będzie służy do odliczania czasu.

Jak wspomniano w poprzedniej części, jedną z funkcji liczników TIM1...3 jest możliwość generowania sygnału PWM. Wykorzystuje się w tym celu jeden z kanałów licznika. Ponieważ sygnał PWM ma sterować jasnością diody podłączonej do GPIOD0 należy wybrać licznik, którego kanał ma wyjście podłączone do tej właśnie linii. W przypadku mikrokontrolera STM8S105 jest to kanał OC2 licznika TIM3. Konfigurując licznik należy najpierw ustalić częstotliwość jego taktowania (poprzez wybór dzielnika częstotliwości) oraz okres. Będzie to jednocześnie okres sygnału PWM. W kolejnym kroku należy skonfigurować wybrany kanał licznika wykorzystując w tym celu funkcję *TIMx_OCyInit*( ), gdzie za *x* należy wstawić numer licznika, a za *y* – numer kanału. W funkcji tej należy wybrać tryb PWM (w trybie 1, wyjście kanału jest w stanie wysokim do czasu osiągnięcia wartości odniesienia zadanej dla danego kanału, w trybie 2 – jest w stanie niskim), aktywować generowanie sygnału na odpowiedniej linii portu GPIO, ustawić wartość odniesienia dla kanału oraz stan aktyny linii (wysoki lub niski). Stosunek zadanej dla danego kanału wartości odniesienia do okresu licznika określa wypełnienie sygnału PWM. Po skonfigurowaniu i uruchomieniu licznika sygnał PWM może być generowany bezpośrednio na podaną linię GPIO, dzięki czemu nie ma potrzeby jakiejkolwiek dalszej jego obsługi programowej. Ponieważ jednak w naszym przypadku chcielibyśmy, aby dioda miała zmienną jasność, konieczne będzie sterowanie wypełnieniem sygnału. Uzyskane to zostanie poprzez zmiany wartości odniesienia zadawanej dla kanału OC2 licznika TIM3. Aby zapewnić równomierne pulsowanie diody, zmiany wypełnienia powinny być wykonywane w równych odstępach czasu. Do ich odmierzania wykorzystany zostanie licznik TIM1, który co 1 ms zgłosi przerwanie. Kod odpowiedzialny za zmianę wypełnienia sygnału PWM umieszczony zostanie w procedurze obsługi przerwania *TIM1 update*.

Przerwania w mikrokontrolerach STM8 mogą być zgłaszane zarówno przez układy peryferyjne mikrokontrolera (np. liczniki, przetwornik AC, układy SPI, I²C i UART) jak i z zewnątrz poprzez linie GPIO. System



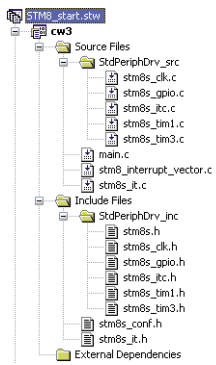
Rysunek 7. Przykład kolejności obsługi przerwań – model jednopoziomowy



Rysunek 8. Przykład kolejności obsługi przerwań – model 4-poziomowy. Przerwanie IRQ1 – priorytet programowy = 2, IRQ2 – priorytet programowy = 1, pozostałe przerwania – priorytet programowy = 3

priorytetów przerwań jest stosunkowo prosty i opiera się na 32 wektorach przerwań na stałe, sprzętowo przypisanych do poszczególnych podukładów i wejść. Lista przypisań dla STM8S105 podana jest w **tabeli 2**. Im niższy jest numer przerwania tym wyższy jest jego priorytet. Standardowo przyjęty jest jednopoziomowy model obsługi przerwań, w którym rozpoczęta procedura obsługi przerwania nie może być przerwana, nawet jeśli nadejdzie zgłoszenie przerwania o wyższym priorytecie. Dopiero po zakończeniu obsługi kontroler przerwań decyduje, które przerwanie obsłużyć jako kolejne (decyduje numer przerwania). Jedynym wyjątkiem jest obsługa przerwań i zdarzeń TLI, TRAP i RESET, których obsługa wykonywana jest bezzwłocznie. Przykład kolejności obsługi przerwań pokazany jest na **rysunku 7**.

Oprócz modelu jednopoziomowego kontroler udostępnia także model 4-poziomowy, w którym każdemu przerwaniu można przypisać priorytety od 1 do 3 (najwyższy), a priorytet 0 (najniższy) zarezerwowany jest dla programu głównego. W tym modelu, gdy podczas obsługi przerwania zostanie zgłoszone przerwanie o wyższym priorytecie programowym, aktualnie wykonywana procedura obsługi zostanie wstrzymana i rozpocznie się obsługa przerwania zgłoszonego. W przypadku, gdy na obsługę oczekują dwa lub więcej przerwań o takim samym priorytecie programowym o kolejności ich obsługi decyduje priorytet sprzętowy. Przykład kolejności obsługi przerwań pokazany jest na **rysunku 8**.



Rysunek 9. Struktura plików w programie 3

W rozpatrywanym obecnie przykładzie, procedura tworzenia projektu będzie taka sama jak opisana w części 1 artykułu z tym, że trzeba dodatkowo dołączyć do projektu biblioteki *stm8s_clk, stm8s_gpio, stm8s_tim1, stm8s_tim3 oraz stm8s_it*. Zawierają one, kolejno, funkcje obsługi źródeł sygnałów taktujących, portów GPIO, licznika TIM1, licznika TIM3 oraz obsługi przerwań. Ponadto należy utworzyć i dołączyć do projektu dwa puste pliki o nazwach *stm8s_it.c* oraz *stm8s_it.h*. Na **rysunku 9** pokazano strukturę plików dla programu 3.

Na początku pliku *main.c* należy dołączyć poleceniem *#include* plik nagłówkowy *stm8s.h*. Polecenia związane z konfiguracją mikrokontrolera zawarte zostaną w funkcjach *CLK_Config*( ), *GPIO_Config*( ) i *TIM_Config*( ), a dwie pierwsze z nich będą identyczne jak w programie 2. W funkcji *TIM_Config*( ) skonfigurować należy dwa liczniki. Licznik TIM3 posłuży do generowania sygnału PWM, a TIM1 – do odmierzania czasu, po którym wypełnienie sygnału PWM będzie należało zmienić. KonfiguracjęTIM3 rozpoczynamy od wyboru dzielnika częstotliwości i okresu (funkcja *TIM3_TimeBaseInit*( )). Następnie, wykorzystując funkcję *TIM3_OC2Init*( ) określamy tryb pracy generatora PWM, włączamy generowanie sygnału na linii GPIOD0, ustawiamy zerową wartość odniesienia (dioda będzie wygaszona) oraz ustalamy wysoki stan aktyny linii wyjściowej. Po włączeniu licznika funkcją *TIM3_Cmd*( ) rozpoczyna on generowanie sygnału PWM. Częstotliwość i wypełnienie PWM opisywane są następującymi zależnościami:

$$f_{PWM}=f_{TIM2}/(TIM\_ARR+1)$$

gdzie:

f_{PWM} – częstotliwość sygnału PWM,

f_{TIM2} – częstotliwość taktowania licznika (po uwzględnieniu dzielnika częstotliwości),

TIM_ARR – okres licznika podawany w funkcji *TIM3_TimeBaseInit*( ).

$$W_{PWM}=[TIM\_CCR/(TIM\_ARR + 1)]*100$$

gdzie:

Tabela 2. Lista przerwań w STM8S105

IRQ	Źródło	Opis
	RESET	Reset
	TRAP	Przerwanie programowe
0	TLI	Przerwanie zew. najwyższego poziomu
1	AWU	Automatyczne wybudzenie z trybu zatrzymania
2	CLK	Przerwanie zegarowe
3	EXTI0	Przerwanie zew. – port a
4	EXTI1	Przerwanie zew. – port b
5	EXTI2	Przerwanie zew. – port c
6	EXTI3	Przerwanie zew. – port d
7	EXTI4	Przerwanie zew. – port e
10	SPI	Zakończono transmisję
11	TIM1	Update/underflow/overflow/break/trigger
12	TIM1	Capture/compare
13	TIM2	Update/underflow
14	TIM2	Capture/compare
15	TIM3	Update/underflow
16	TIM3	Capture/compare
19	I²C	Przerwanie ogólne
20	UART2	Zakończono transmisję
21	UART2	Pełny rejestr odbiorczy
22	ADC1	Zakończono konwersję/alarm watchdoga analogowego
23	TIM4	Update/underflow
24	FLASH	Zakończenie zapisu/próba zapisu do obszaru chronionego

W_{PWM} – wypełnienie sygnału PWM określone w %,

TIM_CCR – wartość odniesienia w kanale OC.

Konfigurowanie licznika TIM1 również rozpoczynamy od wyboru dzielnika częstotliwości i okresu. Wybór tych parametrów zależy od tego, jak często ma zmieniać się wypełnienie sygnału PWM, a więc, jak szybko dioda ma pulsować. Przyjmując dzielnik równy 8000 i okres równy 1, czyli zliczanie dwóch impulsów (od 0 do 1), uzyska się pulsowanie diody z częstotliwością 0,5 Hz. Ponieważ licznik TIM1 daje więcej możliwości niż poprzednio konfigurowany TIM3, wywołując funkcję *TIM1_TimeBaseInit*( ) trzeba wśród parametrów wybrać także tryb zliczania (w naszym przypadku – w górę) oraz ustawić licznik powtórzeń (wartość 0 – ponieważ nie będzie on wykorzystywany). Po skonfigurowaniu licznika należy zdefiniować, które ze zdarzeń związanych z jego działaniem będzie generowało przerwanie. W naszym przypadku będzie to zdarzenie przeładowania licznika, czyli *update*. Odpowiedni wybór i aktywację przerwania wykonuje się wykorzystując funkcję *TIM1_ITConfig*( ). Ostatnią operacją w funkcji *TIM_Config*( ) będzie aktywacja licznika TIM1 przy użyciu funkcji *TIM1_Cmd*( ).

Kod głównej części programu 3 pokazany został na **listingu 3**. Warto zauważyć, że w funkcji *main*( ) znajdzie się tylko wywołanie funkcji konfigurujących poszczególne

podukłady mikrokontrolera oraz funkcji *enableInterrupts*( ), która globalnie włącza obsługę przerwań. Pętla *while* pozostanie pusta, ponieważ operacje związane z obsługą zmian jasności diody zawarte będą w funkcji obsługi przerwania od licznika TIM1. Jej napisanie będzie kolejnym etapem tworzenia programu. W pierwszej kolejności należy otworzyć plik *stm8s_it.h* i uzupełnić go o prototyp funkcji *TIM1_UPD_OVF_TRG_BRK_IRQHandler*( ). Kod źródłowy pliku *stm8s_it.h* zawarty został na **listingu 4**. Następnie należy przejść do pliku *stm8s_it.c* i na jego początku dołączyć poleceniami *#include* pliki nagłówkowe *stm8s.h* i *stm8s_it.h*. W dalszej części zawarty zostanie kod funkcji obsługi przerwania. Ponieważ dioda ma pulsować, przy każdym wywołaniu funkcji należy zmienić wypełnienie sygnału PWM sterującego jasnością diody, a więc zmodyfikować wartość odniesienia w kanale OC2 licznika TIM3. Służy do tego funkcja *TIM3_SetCompare2*( ). Wartość, którą należy zapisać do rejestrów licznika można np. przechowywać na zmiennej *wypelnieniePWM* i w zależności od stanu zmiennej *kierunekZmian* zwiększać lub zmniejszać przy każdym zgłoszeniu przerwania. Należy przy tym pamiętać, aby po osiągnięciu skrajnych wypełnień (zerowego lub całkowitego) zmienić kierunek zmian wartości na przeciwny. Ponadto, obie zmienne muszą być zadeklarowane jako statyczne, aby ich wartości były przechowywane pomiędzy kolejnymi wywołaniami funkcji. Ostatnia operacją, którą należy wykonać jest wyzerowanie znacznika obsługi przerwania, aby mogło być ono zgłoszone ponownie. Służy do tego celu funkcja *TIM1_ClearITPendingBit*( ). Kod źródłowy pliku *stm8s_it.c* pokazano na **listingu 5**.

Ostatnim krokiem tworzenia programu będzie modyfikacja pliku *stm8_interrupt_vector.c*, który zawiera tablicę wektorów przerwań. Należy go uzupełnić o dołączenie pliku nagłówkowego *stm8s_it.h* (polecenie *#include*) zawierającego deklarację napisanej przez nas procedury obsługi przerwania oraz zmodyfikować wpis w tablicy wektorów przerwań dotyczący przerwania IRQ11, które przypisane jest do licznika TIM1. Odpowiednią linię (znajduje się około linii 37 w edytowanym pliku) należy zamienić na:

```
{0x82, (interrupt_handler_t)
TIM1_UPD_OVF_TRG_BRK_IRQHandler},
/* irq11 */
```

Program 4 – obsługa pola dotykowego

Naszym ostatnim zadaniem będzie dodanie do programu 3 obsługi pola dotykowego. Po jego dotknięciu zmieniać się będzie częstotliwość pulsowania diody.

Samodzielna, bezpośrednia obsługa pola dotykowego jest dość skomplikowana. Na szczęście z pomocą przychodzi nam biblioteka oferowana przez STM o nazwie *STM8 Touch Sensing Library* (TSL), która automatyzuje cały proces. Jej użycie sprowadza się do dodania do programu dwóch funkcji związanych z obsługą pól dotykowych oraz do odwołania się do odpowiednich zmiennych w celu określenia czy, i które pole zostało dotknięte. Ponadto konieczne jest zmodyfikowanie ustawień biblioteki do pracy w określonej przez nas konfiguracji sprzętowej.

Tworząc program realizujący opisane zadanie można wyjść od utworzenia projektu i kodu jak dla programu 3. Następnie, ponieważ wykorzystana zostanie biblioteka obsługi pól dotykowych, należy ją dołączyć do projektu. Wygodnie będzie dla jej plików utworzyć w strukturze projektu nowe podgrupy o nazwach *TouchSensingLib_src* i *TouchSensingLib_inc* w grupach *Soure Files* i *Include Files*. Do grupy *TouchSensingLib_src* należy dodać pliki *stm8_tsl_api.c*, *stm8_tsl_rc_acquisition.c*, *stm8_tsl_services.c*, *stm8_tsl_singlechannelkey.c*, *stm8_tsl_timebase.c* z katalogu *C:\ST8\STM8_Touch_sensing_lib\libraries\STM8_TouchSensing_Driver\inc*. Natomiast do grupy *TouchSensingLib_inc* – pliki *stm8_tsl_api.h*, *stm8_tsl_checkconfig.h*, *stm8_tsl_internal.h*, *stm8_tsl_rc_acquisition.h*, *stm8_tsl_services.h*, *stm8_tsl_singlechannelkey.h*, *stm8_tsl_timebase.h* z katalogu *C:\ST8\STM8_Touch_sensing_lib\libraries\STM8_TouchSensing_Driver\src*. Z tego samego katalogu należy skopiować do katalogu głównego własnego projektu plik *stm8_tsl_conf_RC_TOADAPT.h* a następnie zmienić

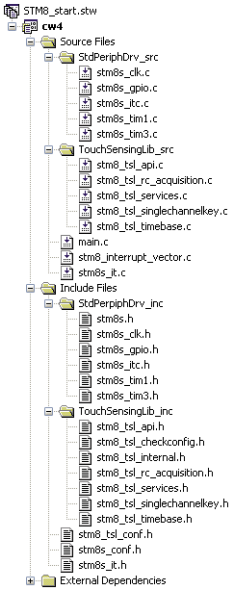
Listing 3. Kod źródłowy programu 3 – plik *main.c.*

```
#include "stm8s.h"
void CLK_Config(void);
void GPIO_Config(void);
void TIM_Config(void);

int main(void) {
  CLK_Config();
  GPIO_Config();
  TIM_Config();

  //aktywacja przerwan
  enableInterrupts();

  //Glowna pelta programu
  while(1) {
    //pusta petla - obsluga zmian PWM poprzez przerwania
  }
}

void CLK_Config(){
  //*** Kod jak w programie 2
}

void GPIO_Config(){
  //*** Kod jak w programie 2
}

void TIM_Config(){
  //Konfiguracja licznikow TIM
  //TIM3 posluzy do generowania PWM
  TIM3_DeInit(); //de-konfiguracja licznika TIM3

  //Inicjalizacja TIM3
  // - prescaler = 1 (dopuszalne tylko potegi 2)
  // - okres = 999 => 1/(16MHz/1000) 62.5us
  //uwaga - licznik zlicza tylko w gore
  TIM3_TimeBaseInit(TIM3_PRESCALER_1, 999);

  //Inicjalizacja kanalu 2 TIM3 do generowania sygnalu PWM:
  // - tryb PWM1 (przy starcie stan wysoki,
  //   zmiana na niski po osiagnieciu wartosci zadanej w kanale OC2),
  // - wyjscie wlaczone,
  // - poczatkowa wartosc OC2 = 0 (zerowe wypelnienie PWM)
  // - aktywny stan niski sygnalu wyjsciowego.
  TIM3_OC2Init(TIM3_OCMODE_PWM1, TIM3_OUTPUTSTATE_ENABLE, 0, TIM3_OCPOLARITY_LOW);

  //wlacz TIM3
  TIM3_Cmd(ENABLE);

  //TIM1 posluzy do odliczania czasu
  TIM1_DeInit(); //de-konfiguracja licznika TIM1

  //Inicjalizacja TIM1
  // - prescaler=8000->16MHz/8000
  //   =zliczanie od 0 do 1 wiec reload co 1ms
  //   (dopuszalne dowolne wartosci prescalera),
  // - zliczanie w gore (dostepne tez inne tryby)
  // - okres = 1
  // - licznik powtorzen =0
  TIM1_TimeBaseInit(8000, TIM1_COUNTERMODE_UP, 1, 0);

  //Ustawienie przerwania od przepelnienia licznika
  TIM1_ITConfig(TIM1_IT_UPDATE, ENABLE);

  //wlacz TIM1
  TIM1_Cmd(ENABLE);
}
```

Listing 4. Kod źródłowy pliku *stm8s_it.h.*

```
#ifndef __STM8S_IT_H
#define __STM8S_IT_H
@far @interrupt void TIM1_UPD_OVF_TRG_BRK_IRQHandler(void);

#endif
```

jego nazwę na *stm8_tsl_conf.h* i również dołączyć do struktury plików projektu, do grupy *Include Files*. Na **rysunku 10** pokazano strukturę plików dla programu 4.

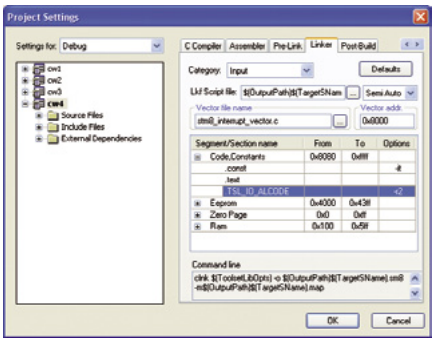


Rysunek 10. Struktura plików w programie 4

Ponieważ biblioteka wymaga, by niektóre z jej funkcji znajdowały się pod parzystymi adresami pamięci, należy zmodyfikować ustawienia linkera. W opcjach projektu, w zakładce *Linker* należy wybrać z listy rozwijalnej kategorię *Input*, a następnie, w tabelce rozwinąć grupę *Code, Constants* i dodać wpis *.TSL_IO_ALCODE* z wartością opcji ustawioną na *-r2*. Na **rysunku 11** przedstawiono widok okna po wprowadzeniu opisanej zmiany. Należy opamiętać, by wprowadzić ją zarówno dla kompilacji typu *debug* jak i *release*.

Jak już wspomniano, do poprawnej pracy biblioteka wymaga skonfigurowania dzięki czemu dostosowana zostaje do konkretnego środowiska sprzętowego wykorzystywanego w danym projekcie. Ustawień dokonuje się poprzez modyfikację pliku *stm8_tsl_conf.h*. Po jego otwarciu należy wprowadzić zmiany zestawione poniżej. Lista przedstawiona jest w konwencji: (przybliżony numer linii w edytowanym pliku) *NAZWA* stałej – nowa wartość – znaczenie.

- (40) *MCU_SELECTION* – 4 – ustawienie typu mikrokontrolera na STM8S
- (65) *TIMACQ – TIM2* – wybór licznika, który biblioteka wykorzystywać będzie do odmierzania czasu; standardowo przyjęty TIM3 jest już w naszym programie wykorzystywany do generowania sygnału PWM



Rysunek 11. Okno ustawień linkera

Listing 5. Kod źródłowy procedury obsługi przerwania w programie 3 – plik *stm8s_it.c.*

```
#include "stm8s.h"
#include "stm8s_it.h"
void TIM1_UPD_OVF_TRG_BRK_IRQHandler(void) {
  static unsigned int wypelnieniePWM = 0;
  static signed char kierunekZmian = +1;

  if (kierunekZmian > 0) {
    wypelnieniePWM++;
    // zmiana kierunku po osiagnieciu max jasnosci
    if (wypelnieniePWM  >= 1000)
      kierunekZmian = -1;
  } else {
    wypelnieniePWM--;
    // zmiana kierunku po osiagnieciu min jasnosci
    if (wypelnieniePWM <= 0)
      kierunekZmian = +1;
  }
  // Zmien wypelnienie PWM (czyli jasnosc diody)
  TIM3_SetCompare2(wypelnieniePWM);
  // Wyczysc flage przerwania
  TIM1_ClearITPendingBit(TIM1_IT_UPDATE);
}
```

- (66) *TIMACQ_CNTR_ADD* – 0x530A – adres rejestru CNTRH licznika TIM2
- (101) *LOADREF_PORT_ADDR – GPIOC_BaseAddress* – adres portu, do którego podłączone jest pole dotykowe; w zestawie STM8S Discovery jest to port C
- (103) *LOADREF_BIT* – 0x04 – linia portu, do której podłączona jest linia referencyjna pola dotykowego; w zestawie STM8S Discovery jest to linia PC2.
- (118) *SCKEY_P1_KEY_COUNT* – 1 – liczba wykorzystywanych pól dotykowych dla pierwszego portu
- (121–123) *SCKEY_P1_B*, *SCKEY_P1_C*, *SCKEY_P1_D* – wszystkie 0 – maski kolejnych linii pól dotykowych; ponieważ ich nie wykorzystujemy, należy je wyzerować
- (146) *SCKEY_P2_KEY_COUNT* – 0 – liczba wykorzystywanych pól dotykowych dla drugiego portu; ponieważ nie mamy pól podłączonych do kolejnych portów, należy stałą wyzerować
- (150–157) *SCKEY_P2_A…H* – 0 – maski linii pól dotykowych dla drugiego portu, ponieważ ich nie wykorzystujemy, należy je wszystkie wyzerować
- (174) *SCKEY_P3_KEY_COUNT* – 0 – liczba wykorzystywanych pól dotykowych dla trzeciego portu; ponieważ nie mamy pól podłączonych do kolejnych portów, należy stałą wyzerować
- (178–185) *SCKEY_P3_A…H* – 0 – maski linii pól dotykowych dla trzeciego portu, ponieważ ich nie wykorzystujemy, należy je wszystkie wyzerować
- (199) *NUMBER_OF_MULTI_CHANNEL_KEYS* – 0 – liczba pól dotykowych obsługiwanych w trybie wielokanałowym; brak takich pól
- (289) *GPIOC_ELECTRODES_MASK* – 0x0A – maska linii portu GPIO, do których podłączone są elektrody sterujące polem dotykowym; w zestawie podłączone są one do linii PC1 i PC2
- (287–295) *GPIOx_ELECTRODES_MASK* – 0x00 – wszystkie stałe oprócz *GPIOC_ELECTRODES_MASK* należy wyzerować

Gdy wszystkie ustawienia konfiguracyjne zostaną wprowadzone, można rozpocząć uzupełnianie kodu programu. Na początku pliku *main.c* należy dodać polecenie *#include* dołączające plik nagłówkowy *STM8_TSL_API.h*. Następnie, wewnątrz funkcji *main*( ), zaraz za wywołaniem funkcji konfigurujących podukłady mikrokontrolera należy umieścić wywołanie funkcji inicjalizującej bibliotekę o nazwie *TSL_Init*( ). Oprócz inicjalizacji należy jeszcze określić, które z pól dotykowych są w systemie obecne oraz je aktywować. Wykonuje się to ustawiając wartość 1 w polach. *Setting.b.IMPLEMENTED* i *.Setting.b.ENABLED* struktur zawartych w tablicy *sSCKeyInfo[ ]*. Całą obsługę pól dotykowych zawiera i automatyzuje wykorzystywana przez nas biblioteka *TSL*, jednak aby mogła ona te zadania wykonać należy w głównej pętli programu umieścić wywołanie funkcji *TSL_Action*( ). Realizuje ona tzw. maszynę stanów, która m.in. monitoruje stan pól dotykowych i odnotowuje go na odpowiednich zmiennych. W szczególności, zmienna *TSLState* określa, czy maszyna stanów zakończyła aktualny cykl pracy, zmienna *TSL_GlobalSetting.b.CHANGED* określa, czy zmienił się stan któregoś z pól, a zmienna *sSCKeyInfo[0].Setting.b.DETECTED* określa, czy pole o numerze 0 zostało przyciśnięte. Ponieważ w treści zadania przyjęte zostało, że po naciśnięciu pola dotykowego zmienić ma się częstotliwość pulsowania diody LED, w reakcji na wykrycie dotknięcia powinniśmy zmieniać wartość okresu licznika TIM1. Przyjąć można, że będą to wartości 1, 4 i 7. By zapamiętać aktualny wybór należy w funkcji *main*( ) zadeklarować dodatkową zmienną, np.: *szybkoscLED*, której wartość będzie zmieniania po wykryciu dotknięcia przycisku. Kod źródłowy funkcji *main*( ) podany jest na **listingu 6**.

Aby uzyskać łatwiejsze do zauważenia efekty wizualne warto zwiększyć nominalną częstotliwość pulsowania diody. W tym celu w funkcji *TIM_Conf*( ) należy zmienić wartość dzielnika częstotliwości dla liczni-

Listing 6. Kod źródłowy funkcji *main( )* programu 4

```
int main(void) {
  unsigned char szybkoscLED=1;
  CLK_Config();
  GPIO_Config();
  TIM_Config();
  // Inicjalizacja obslugi klawiszy dotykowych
  TSL_Init();
  // Inicjalizacja klawiszy (jest tylko jeden) i ich wlaczenie
  sSCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sSCKeyInfo[0].Setting.b.ENABLED = 1;
  //aktywacja przerwan
  enableInterrupts();
  //Glowna pelta programu
  while(1) {
    //w petli tylko obsluga maszyny stanow biblioteki TSL
    //oraz zmiana okresu licznika TIM1 w celu zmiany
    //szybkosci pulsowania diody
    TSL_Action(); //obsluga maszyny stanow TSL
    //Sprawdzenie, czy byla zmiana stanu klawisza
    //i czy nic innego (zwiazanego z dotykiem) sie nie dzieje.
    if ((TSL_GlobalSetting.b.CHANGED)&&(TSLState == TSL_IDLE_STATE)) {
      // Kasowanie flagi zmiany
      TSL_GlobalSetting.b.CHANGED = 0;
      // Sprawdzenie, czy klawisz 0 zostal nacisnity
      if (sSCKeyInfo[0].Setting.b.DETECTED) {
        if (szybkoscLED>=3) {szybkoscLED=0;}
        szybkoscLED++;
        TIM1_SetAutoreload(szybkoscLED*3-2);
        TIM1_SetCounter(0);
      }
    }
  }
}
```

ka TIM1 z 8000 na 2000. Warto przy tym zauważyć, że w funkcji tej nie ma potrzeby konfigurowania liczników, które wykorzystywane są przez bibliotekę obsługi pól dotykowych. W naszym przypadku są to liczniki TIM2 i TIM4. Ich konfiguracja wykonywana jest przez tę bibliotekę automatycznie. Wymagane jest jednak, aby licznik określony jako *TIMACQ* był licznikiem 16-bitowym z 8-bitowym dzielnikiem częstotliwości, a *TIMTICK* był podstawowym licznikiem 8-bitowym. Ponadto, częstotliwość sygnału taktującego liczniki musi być równa 16 MHz.

Ostatnim krokiem, który trzeba wykonać przed uruchomieniem programu jest modyfikacja pliku *stm8_interrupt_vector.c*. Należy go uzupełnić o dołączenie pliku nagłówkowego *STM8_TSL_API.h* (polecenie *#include*) zawierającego deklarację procedury obsługi przerwania od licznika TIM4 oraz zmodyfikować wpis w tablicy wektorów przerwań dotyczący przerwania IRQ23. Odpowiednią linię (znajduje się około linii 50 w edytowanym pliku) należy zamienić na:

```
{0x82, (interrupt_handler_t)TSL_
Timer_ISR}, /* irq23 */
```

Przedstawione w obu częściach artykułu programy pokazują kilka podstawowych możliwości układów STM8S oraz procedurę tworzenia projektów je wykorzystujących. Układy te wydają się być ciekawą alternatywą dla innych kontrolerów 8-bitowych. Szczególnie interesujące mogą być dla inżynierów znających układy STM32, ponieważ oferowane przez większość bloków peryferyjnych możliwości oraz sposób ich obsługi wykazują w obu rodzinach wiele podobieństw. Dzięki temu możliwe jest stosunkowo łatwe i szybkie poznanie układów drugiej rodziny, gdy zna się już jedną z nich.

Marek Galewski
marg@mech.pg.gda.pl