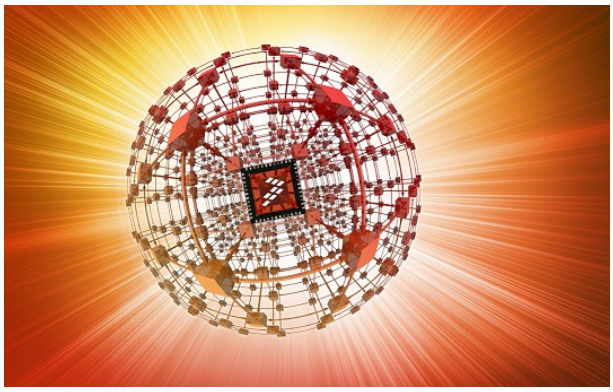
Laboratorium Robotyki

Programowanie mikrokontrolera z rodziny Kinetis K40 w środowisku CodeWarrior Development Studio[1]

Jan Kędzierski
Marek Wnuk



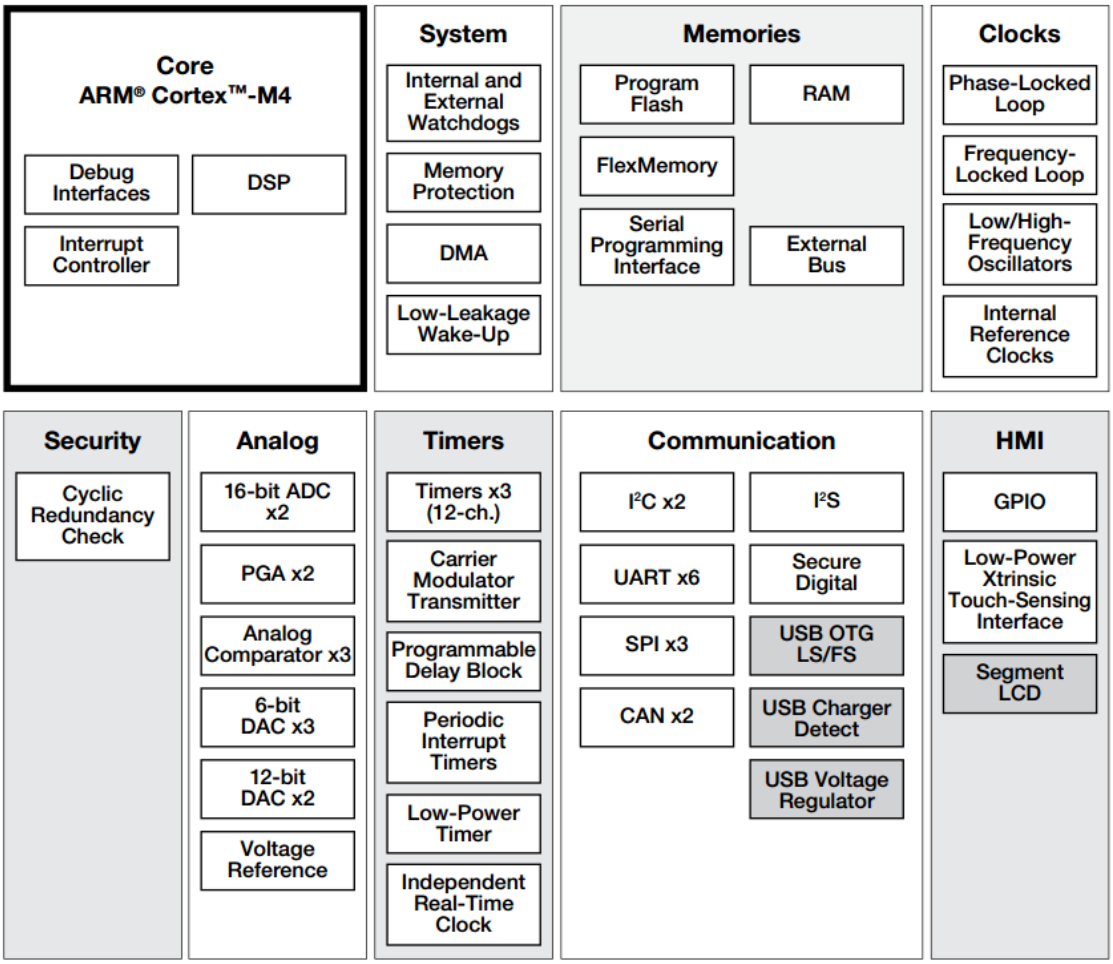
wer. 1.4
Wrocław 2015

[1]Dokument stanowi instrukcję do ćwiczenia w ramach kursu *Sterowniki robotów*.

Spis treści

1 Wprowadzenie

Firma Freescale posiada obecnie w ofercie wiele mikrokontrolerów opartych na rdzeniu ARM Cortex-M4, które składają się na rodzinę Kinetis. Układy mają wspólną podstawę konstrukcyjną, która oprócz jednakowego rdzenia obejmuje, typowe dla mikrokontrolerów, bloki funkcjonalne w postaci pamięci (Flash i SRAM), peryferii analogowych (16-bitowe przetworniki A/C, 12-bitowe przetworniki C/A, komparatory analogowe, wzmacniacze operacyjne z programowalnym wzmocnieniem, źródło napięcia odniesienia) oraz peryferii cyfrowych, do których należą liczne interfejsy komunikacyjne (SPI, I2C, USART, I2S, CAN), moduł do obliczania sumy kontrolnej CRC, układy licznikowe ze wsparciem sterowania silników prądu stałego, układ zegara czasu rzeczywistego RTC, moduł obsługi kart SDHC, kontroler klawiatur pojemnościowych oraz kontroler DMA. Dodatkowo wszystkie układy posiadają rozbudowaną warstwę zarządzania energią, o czym świadczy duża liczba zaimplementowanych trybów pracy (pobór prądu nie przekracza 200 μA/MHz) oraz szeroki zakres napięcia zasilania (dolny próg to 1,7 V).

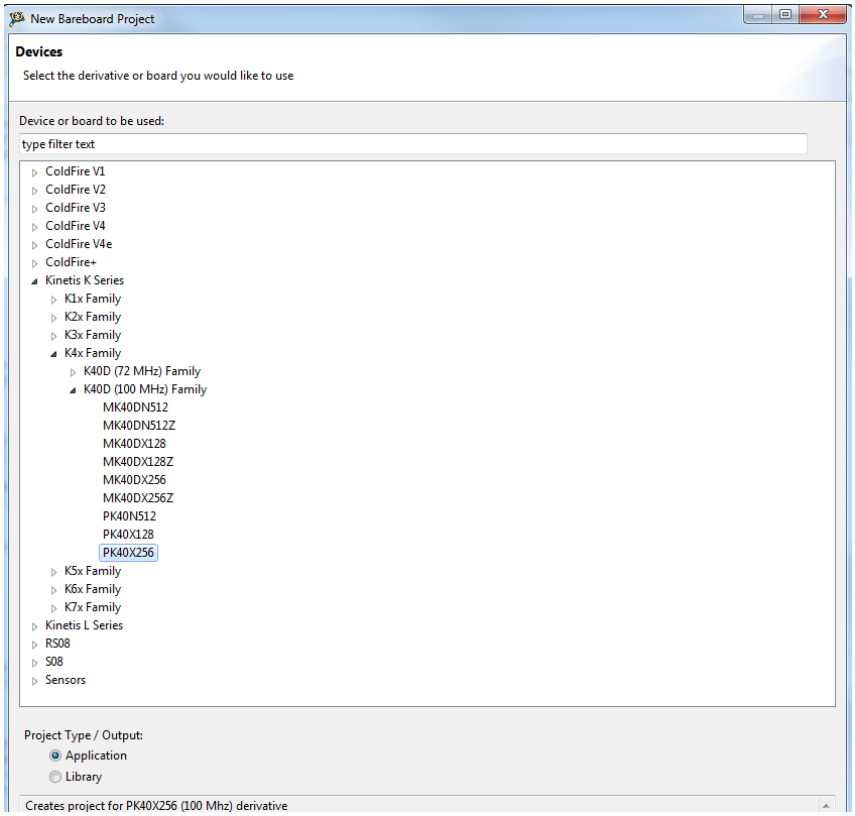


Celem ćwiczenia jest zapoznanie się z modułami, zawierającymi jednostkę główną w postaci 32-bitowego mikrokontrolera K40X256VLQ100 [1] z rdzeniem ARM Cortex-M4[6]. W trakcie ćwiczenia zostanie wykorzystane środowisko programistyczne CodeWarrior Development Studio (Eclipse) [2]. W ramach ćwiczenia należy wygenerować przy pomocy narzędzia Processor Expert [3] kod inicjujący układ, następnie uzupełnić go tak, aby realizował odpowiednie zadania. Przed przystąpieniem do ćwiczenia, należy zapoznać się z zestawem ćwiczeniowym, w szczególności z jego schematem ideowym.

2 Tworzenie nowego projektu

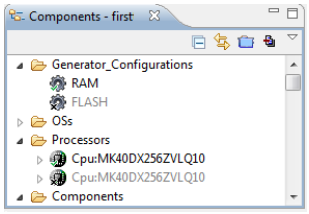
W celu utworzenia projektu należy posłużyć się kreatorem. W tym celu wybieramy menu **File**, następnie **New** i klikamy na **Bareboard Project**. Pojawi się okno kreatora, w którym wpisujemy nazwę projektu oraz tworzymy nową ścieżkę przestrzeni roboczej. Należy zwrócić uwagę, że CW ma już wybraną domyślną ścieżkę, którą należy zmienić wg wskazania prowadzącego. **UWAGA! Po wybraniu ścieżki, na jej końcu, dopisujemy nazwę folderu, w którym CW umieści wszystkie pliki - może być taka sama jak nazwa projektu.** Następnie postępujemy wg poniższych instrukcji:

- **Devices:** wybieramy układ *PK40X256 (100 Mhz)*,
- **Connections:** wybieramy dla zestawu KwikStik [7] *Segger J-Link / J-Trace / SWO (SWD based)*, dla zestawu TWR-K40X256 [8] *Open source JTAG*,
- **Language and Build Tools Options**
  - **Language:** pozostawiamy *C*,
  - **Floating Point:** pozostawiamy *Software*,
  - **ARM Build Tools:** wybieramy *Freescale*,
- **Rapid Aplication Development**
  - **Rapid Aplication Development:** zaznaczamy *Processor Expert*,
  - **Start with perspective designed for:** wybieramy *Hardware configuration*,

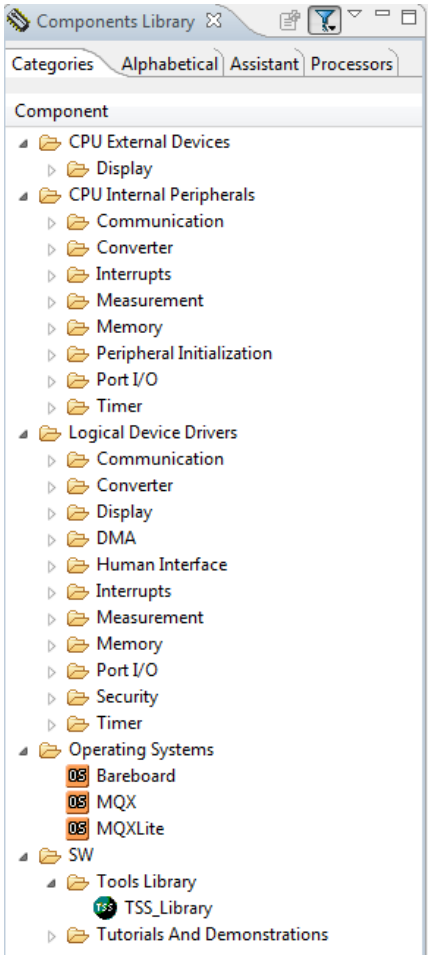


Rysunek 1: Wybór mikrokontrolera

3 Konfiguracja peryferii układu



Wszystkie opisane poniżej przykłady konfiguracji wybranych peryferii opierają się na środowisku Processor Expert (PE). Jest to graficzne, zautomatyzowane środowisko przyspieszające tworzenie złożonych aplikacji wbudowanych, pozwalające przygotować kod z poziomu GUI. Po utworzeniu nowego projektu PE automatycznie dołączył do niego jeden lub więcej modułów CPU. W zależności od konfiguracji sprzętowej, PE przygotowuje komponent CPU do pracy, tylko w pamięci RAM lub z opcją zaprogramowania nieulotnej pamięci FLASH. Wszystkie kolejne moduły powinno się dodawać według potrzeb. W tym celu posługujemy się bibliotekami znajdującymi się w oknie Componets Library. Należy zwrócić uwagę, że komponenty te są podzielone na kilka kategorii:



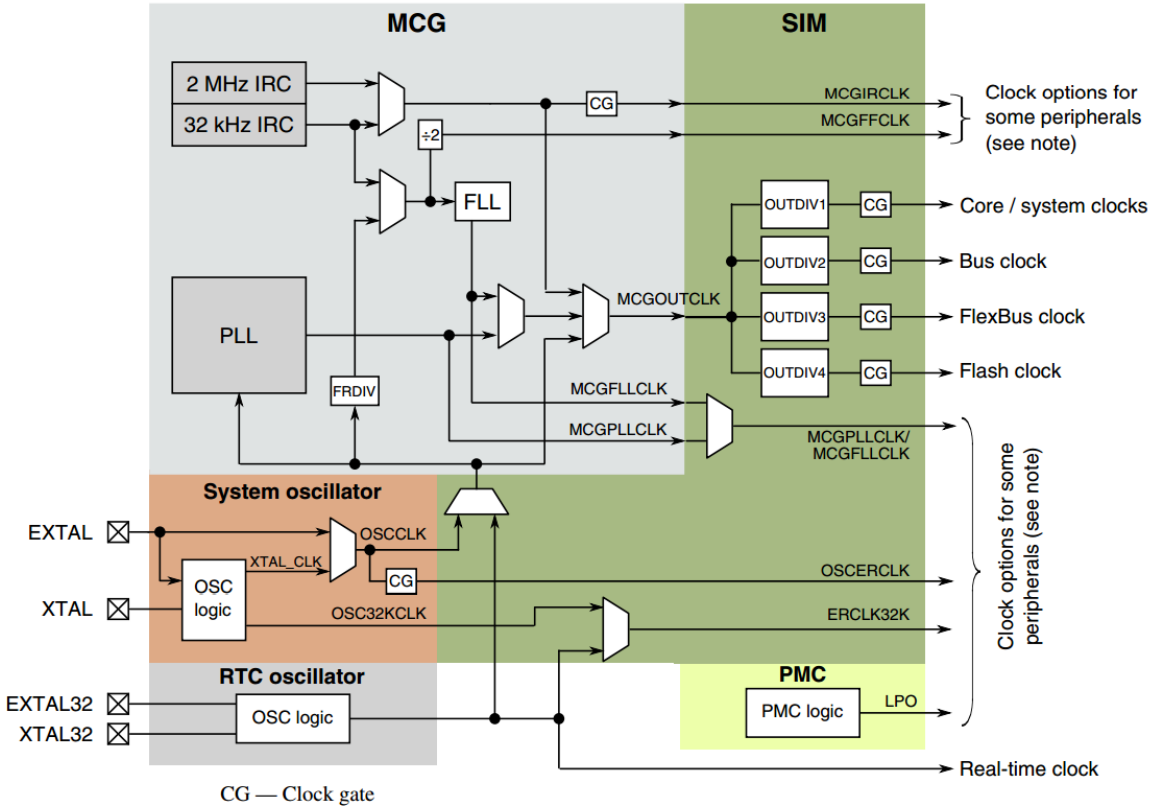
- **Processor External Devices** — są to komponenty, które pozwalają na sterowanie zewnętrznymi urządzeniami podłączonymi do mikrokontrolera, np. czujniki, układy pamięci, wyświetlacze,...
- **Processor Internal Peripherals** — komponenty pozwalające na używanie w programie wszystkich wewnętrznych modułów, zintegrowanych w danym układzie, np. przetworniki, porty I/O, timery,...
- **Logical Device Drivers** — komponenty te, dostarczają warstwę abstrakcji sprzętowej (wewnętrznych bloków mikrokotnrolera). Zostały one stworzone głównie z myślą o zastosowaniu w aplikacjach pracujących pod kontrolą RTOS. Jednakże, mogą być również wykorzystywane do standardowej pracy z płytką ewaluacyjną. Kod tych modułów jest zoptymalizowany pod kątem aplikacji oraz (o ile jest wykorzystywany) dla konkretnego systemu RTOS.
- **Operating systems** — są to komponenty związane z pracą (o ile jest wykorzystywany) systemu RTOS, np. Freescale MQX.
- **SW** — jest to zbiór bibliotek zawierających, np. algorytmy, moduły dziedziczące zasoby sprzętowe lub komponenty utworzone przez użytkownika t.j. TSS do obsługi interfejsu dotykowego.

Po dwukrotnym kliknięciu na wybrany komponent, PE doda go do aktualnie otwartego projektu, następnie moduł ten pojawi się w zakładce *Components - nazwa_projektu*. Klikając dwukrotnie na dodany komponent, otwieramy okno z konfiguracją wybranego modułu. **UWAGA! Jeśli okaże się, że dla zaproponowanych poniżej konfiguracji brakuje pewnych opcji oznacza to, że pojawią się one po przełączeniu widoku w Advanced lub Expert. Zmianę widoku dokonuje się w prawym górnym rogu zakładki Component Inspector.** Należy pamiętać, że każdorazowa zmiana ustawień wymaga ponownego zbudowania projektu. Warto też zwrócić uwagę, że dostępność niektórych opcji, zależy od aktualnej konfiguracji danego bloku. Niestety, konfigurator Processor Expert, dla niektórych modułów, nie umożliwia edycji wszystkich rejestrów/pól. W takich sytuacjach warto posłużyć się modułami **Peripheral**

Initialization, które dostarczają jedynie konfigurację inicjalizacji danego bloku funkcyjnego. Moduły te, możemy znaleźć w oknie *Components library* w katalogu *Processor Internal Peripherals/Peripheral Initialization*.

4 Konfiguracja CPU

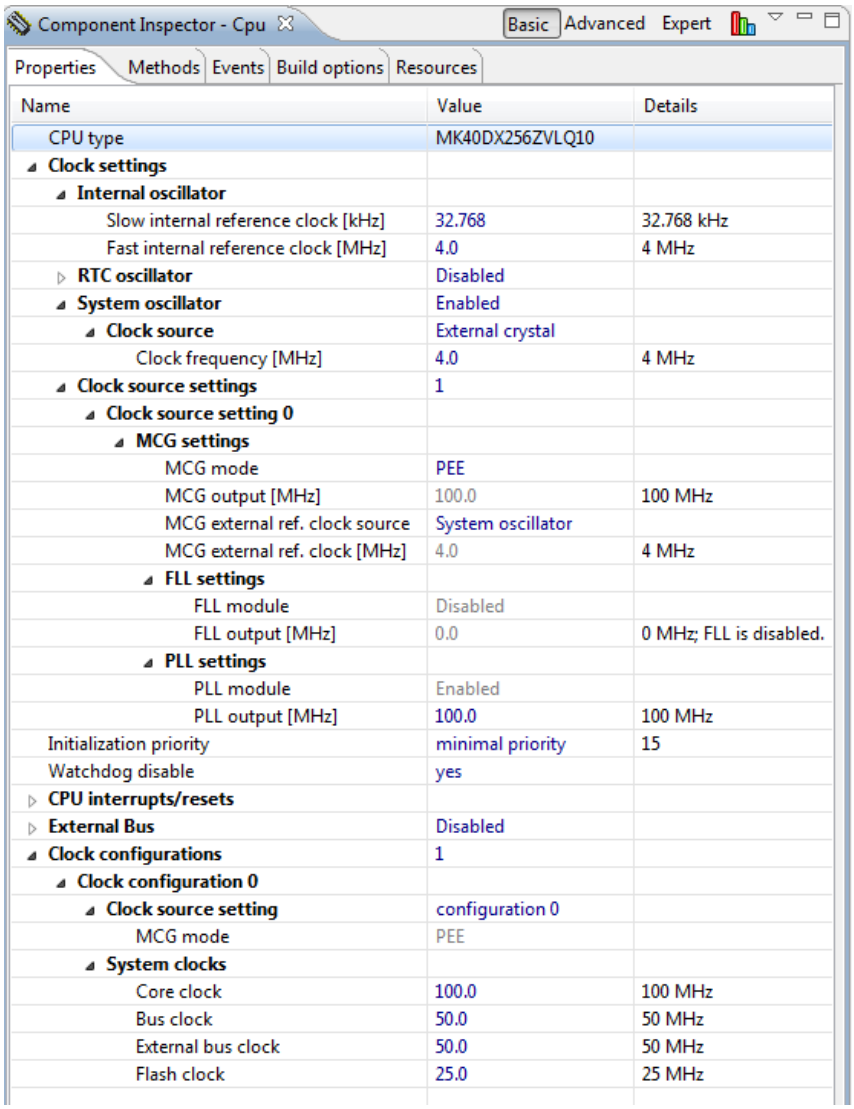
Przed przystąpieniem do konfiguracji CPU, należy zapoznać się z dokumentacją układu [1], w szczególności z modułem MCG (Multipurpose Clock Generator) opisanym w 24 rozdziale, na stronie 549. Warto też przyjrzeć się, w jaki sposób MCG dystrybuuje sygnały zegara dla poszczególnych peryferii. Opis ten oraz schemat blokowy można znaleźć w rozdziale 5 na stronie 195 w wspomnianym dokumencie.



Rysunek 2: Blok dystrybucji sygnału zegarowego

MCG posiada dwa wewnętrzne źródła zegara (Slow IRC ∼32kHz, Fast IRC ∼4MHz) oraz dwa zewnętrzne, pochodzące z bloków System Oscillator oraz RTC oscilator. Ponadto w swej strukturze zwiera pętlę PLL oraz FLL, których można użyć w celu wygenerowania dowolnych (dozwolonych) częstotliwości taktowania. Zatem, układ generowania zegara może pracować w kilku trybach: FEI, FEE, FBI, FBE, PBE, PEE, BLPI, BLPE oraz Stop. Każdy z tych trybów opisany jest w dokumentacji [1] w rozdziale 24 na stronie 564. W ćwiczeniu tym, należy skonfigurować MCG do pracy w trybie PEE (PLL Engaged External). Oznacza to, że zaprzęgamy do pracy generator VCO (Voltage Controlled Oscillator) z pętlą PLL (Phase-locked Loop), który taktowany jest zewnętrznym zegarem, w tym przypadku pochodzącym z zewnętrznego oscylatora kwarcowego 4MHz.

Po utworzeniu nowego projektu, Processor Expert automatycznie doda moduł (bean) CPU, który znajdziemy w oknie Components - nazwa_projektu. Po dwukrotnym kliknięciu otworzy się okno konfiguracyjne (Component Inspector).



Rysunek 3: Konfiguracja CPU

Zakładka Properties

Clock Settings

W pierwszej kolejności musimy skonfigurować zewnętrzne źródło zegara, które zostanie uzyskane przy pomocy bloku System Oscillator. Blok ten jest opisany w dokumentacji układu [1] w rozdziale 25 na stronie 585.

System oscillator: wybieramy *Enabled* - na płytce zestawu laboratoryjnego znajduje się rezonator kwarcowy, o częstotliwości 4MHz (Kwikstik) lub 8MHz (TWRK-K40),

Clock source: wybieramy *External crystal*,

Clock frequency [MHz]: wpisujemy *4* lub *8*,

Clock source settings - konfigurujemy tylko jedno źródło zegara systemowego,

Clock source setting 0 - wybieramy pierwsze źródło,

Teraz możemy skonfigurować MCG do pracy z pętlą PLL tak, aby uzyskać maksymalną dopuszczalną częstotliwość pracy układu (w tym przypadku 100MHz).

MCG settings

MCG mode: wybieramy *PEE* (PLL Engaged External), czyli pętlę PLL pracującą z zew. źródłem zegara,

MCG external ref. clock source: wybieramy *System oscillator*, czyli uprzednio skonfigurowane zew. źródło zegara

PLL settings

PLL output [MHz]: wpisujemy *100*.

Przechodzimy teraz do konfiguracji częstotliwości pracy całego systemu (rdzenia, magistrali, pamięci). Powyższe źródło pracuje z częstotliwością 100MHz. Jest to maksymalna częstotliwość pracy tego układu. W dokumentacji układu [1] w rozdziale 5 na stronie 197 możemy sprawdzić dopuszczalne częstotliwości pracy poszczególnych bloków układu. Zgodnie z powyższym, dokonujemy następującej konfiguracji:

Clock configurations 1 - mamy skonfigurowane tylko jedno źródło

Clock configuration 0 - wybieramy pierwsze źródło

Clock source settings: - wybieramy *configuration 0*,

System clocks

Core clock: wpisujemy *100*,

Bus clock: wpisujemy *50*,

External bus clock: wpisujemy *50*,

FLASH clock: wpisujemy *25*.

Po wykonaniu powyższej konfiguracji można załadować program do mikrokontrolera. Dzięki temu, zweryfikujemy czy projekt został poprawnie utworzony oraz czy debuger komunikuje się z układem docelowym. Code Warrior zweryfikuje także, czy debuger ma zainstalowany aktualny firmware. Jeśli nie, zaproponuje jego aktualizację.

5 Konfiguracja GPIO

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Port I/O. Następnie, wybieramy moduł BitIO, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu. Konfigurację należy przeprowadzić, podobnie jak w przypadku CPU, tzn. w oknie Component Inspector. Każdą wybraną linię, należy odpowiednio skonfigurować, np. jako wejście lub wyjście, nadać jej adekwatną nazwę, parametry pracy oraz wybrać minimalny zestaw przydatnych funkcji do jej obsługi. Dokładny opis konfiguracji poszczególnych portów, jako linie GPIO, można znaleźć w dokumentacji układu [1] w rozdziale 11 na stronie 271.

zakładka Properties

Component name: nadajemy nazwę adekwatną dla wybranej linii, np. dla diody *LED*, dla klawisza *KEY*,

Pin for I/O: wybieramy linię, do której podłączona jest dioda LED lub klawisz,

Direction: w zależności od funkcji jaką będzie pełnić wybieramy *Output* lub *Input*,

Initialization - parametry startowe,

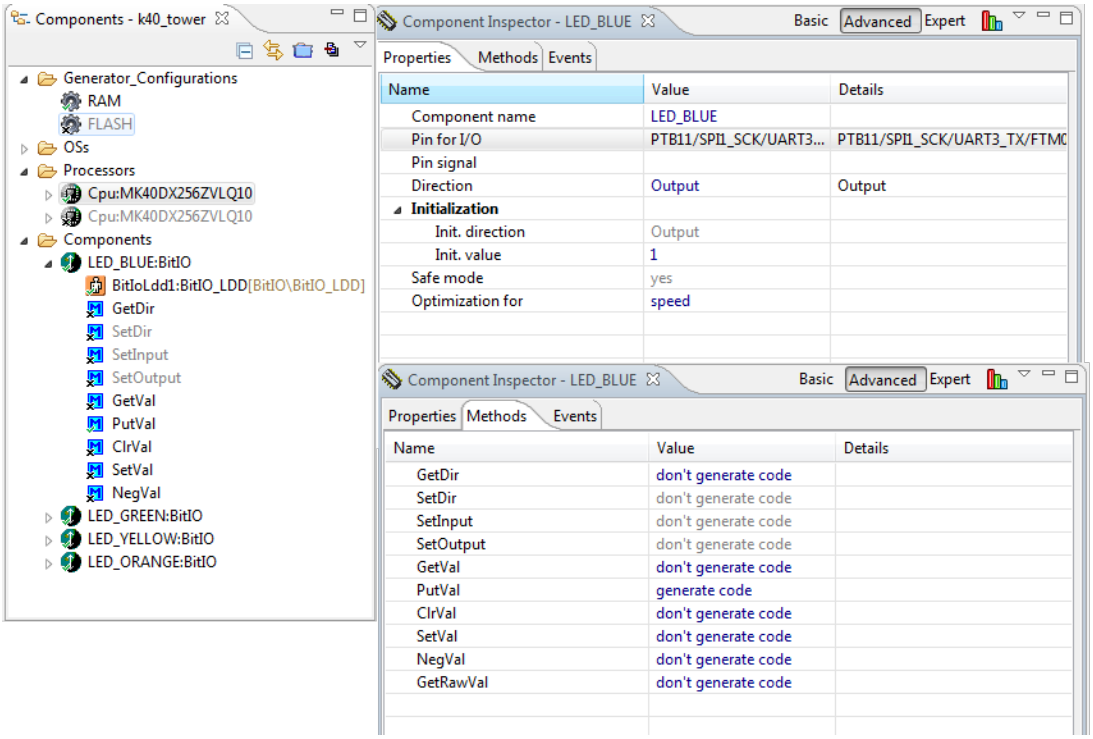
Init value: należy wybrać wartość, którą przyjmie linia po resecie (tylko dla wyjścia).

zakładka Methods

Wybieramy minimalny zestaw funkcji potrzebnych do obsługi danej linii. Warto zwrócić uwagę, że jeśli wybrana linia została skonfigurowana jako wyjście, to warto odznaczyć (don't generate code) wszystkie funkcje, które służą do odczytywania wartości na linii. W takim przypadku można wygenerować, np. tylko funkcję *PutVal*.

zakładka Events

Moduł ten nie generuje żadnych zdarzeń/przerwań.



Rysunek 4: Przykład konfiguracji GPIO do sterowania diodą LED

Poniżej zaprezentowano program demo, którego efektem działania są cztery mrugające diody LED, znajdujące się na płytce zestawu Freescale Tower TWR-K40X256 [8].

```
/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
    int i=0;
    bool togg=FALSE;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
    for(;;) {
        LED_BLUE_PutVal(togg);
        LED_GREEN_PutVal(togg);
        LED_YELLOW_PutVal(togg);
        LED_ORANGE_PutVal(togg);
        for(i=0;i<=1000000;i++) { } // delay
        if (togg) togg=FALSE; else togg=TRUE;
    }

  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
```

UWAGA: Konfigurator Processor Expert nie zapewnia pełnej edycji wszystkich rejestrów konfiguracyjnych portów.

Jeśli chcemy, aby dana linia pracowała, np. jako wejście z wewnętrznym rezystorem pullup, należny ręcznie skonfigurować rejestr **PORTx_PCRn**. Opis rejestru możemy znaleźć w dokumentacji [1] układu w rozdziale 11 na stronie 280. Poniżej przedstawiono przykład włączenia górnych rezystorów podciągających.

```
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  PORTC_PCR7|=PORT_PCR_PE_MASK;     // Internal pulling resistor is enabled
  PORTC_PCR7|=PORT_PCR_PS_MASK;     // Internal pull-up resistor type
```

6 Konfiguracja zewnętrznego przerwania ExIRQ

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Interrupts. Następnie, wybieramy moduł ExtInt, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu. Konfigurację należy przeprowadzić w oknie Component Inspector. Dokładny opis konfiguracji zewnętrznych przerwań można znaleźć w dokumentacji [1] układu w rozdziale 11 na stronie 271.

zakładka Properties

Component name: nadajemy nazwę modułu, np. dla przełącznika typu switch *MyKey*,
Pin: wybieramy linię, do której podłączony jest switch, np. *PTC5*,
Generate interrupt on: wybieramy zbocze/zbocza, na którym ma wystąpić przerwanie,

zakładka Methods

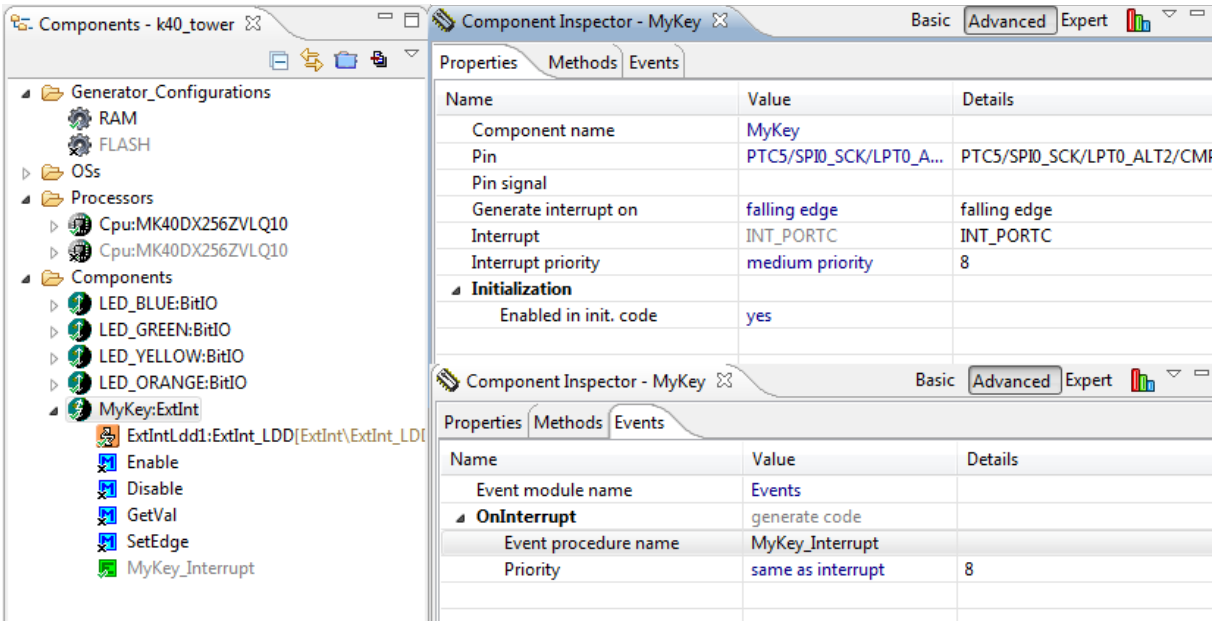
Wybieramy minimalny zestaw funkcji do obsługi modułu, np. włączenie i wyłączenie przerwania.

zakładka Events

Event module name: pozostawiamy *Events* - jest to nazwa modułu, w którym Processor Expert umieści funkcję obsługującą konfigurowane przerwanie,
OnInterrupt

Event procedure name: wpisujemy nazwę funkcji, która zostanie wywołana podczas przerwania, np. *MyKey_Interrupt*.

Konfiguracja przerwania jest już zakończona. W celu zwizualizowania efektu jego działania, można posłużyć się diodą LED oraz wygenerowanymi w tym celu funkcjami. W zakładce *Files*, aktualnie otwartego projektu, należy odszukać moduł, w którym Processor Expert umieścił funkcję wywoływaną przez skonfigurowane przerwanie. W tym przypadku będzie to plik *Event.c* znajdujący się w katalogu *Sources*. Po odszukaniu funkcji *MyKey1*, możemy posłużyć się zaproponowanym poniżej kodem.



Rysunek 5: Przykład konfiguracji ExIRQ wyzwalanego klawiszem.

UWAGA: Aby przerwania zewnętrzne pracowały prawidłowo, konieczne jest zastosowanie rezystorów podciągających. Niestety, konfigurator Processor Expert nie umożliwia edycji wszystkich pól rejestrów konfiguracyjnych. W związku z tym, należny dokonać ręcznego włączenia rezystorów podciągających.

Poniżej zaprezentowano program demo, którego efektem działania jest włączenie i wyłączenie diody LED znajdującej się na płytce zestawu Freescale Tower TWR-K40X256 [8].

```
/*
** ===================================================================
**     Event       :  MyKey_Interrupt (module Events)
**
**     Component   :  MyKey [ExtInt]
**     Description :
**         This event is called when an active signal edge/level has
**         occurred.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void MyKey_Interrupt(void)
{
  /* Write your code here ... */
    static bool togg;

    LED_BLUE_PutVal(togg);
    if (togg) togg=FALSE; else togg=TRUE;

}
```

```
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  PORTC_PCR5|=PORT_PCR_PE_MASK;     // Internal pulling resistor is enabled
  PORTC_PCR5|=PORT_PCR_PS_MASK;     // Internal pull-up resistor type

  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
```

7 Konfiguracja cyklicznego przerwania

Mikrokontroler K40X256VLQ100 posiada kilka źródeł cyklicznych przerwań. Są to m.in.: trzy uniwersalne, wielokanałowe, konfigurowalne moduły czasowo-licznikowe FTM, cztery moduły przerwań cyklicznych PIT, a także generator nisko prądowy LPTMR. Dokładny opis konfiguracji modułu FTM można znaleźć w dokumentacji [1] układu w rozdziale 37 na stronie 895. Natomiast opis konfiguracji PIT znajduje się w rozdziale 38 na stronie 1031.

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Timer. Następnie, wybieramy moduł TimerInt, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu. Konfigurację należy przeprowadzić w oknie Component Inspector.

zakładka Properties

Component name: nadajemy nazwę modułu, np. *MyClock*,
Periodic interrupt source: wybieramy *PIT_LDVAL0*,
Interrupt service/event

Interupt period: wpisujemy czas pomiędzy przerwaniami, np. *250ms* - 4 przerwania na sekundę.

zakładka Methods

Wybieramy minimalny zestaw funkcji potrzebnych jedynie do zmiany częstotliwości przerwań. Pozostawiamy tylko te funkcje, które dla wybranego zakresu mają sens użycia.

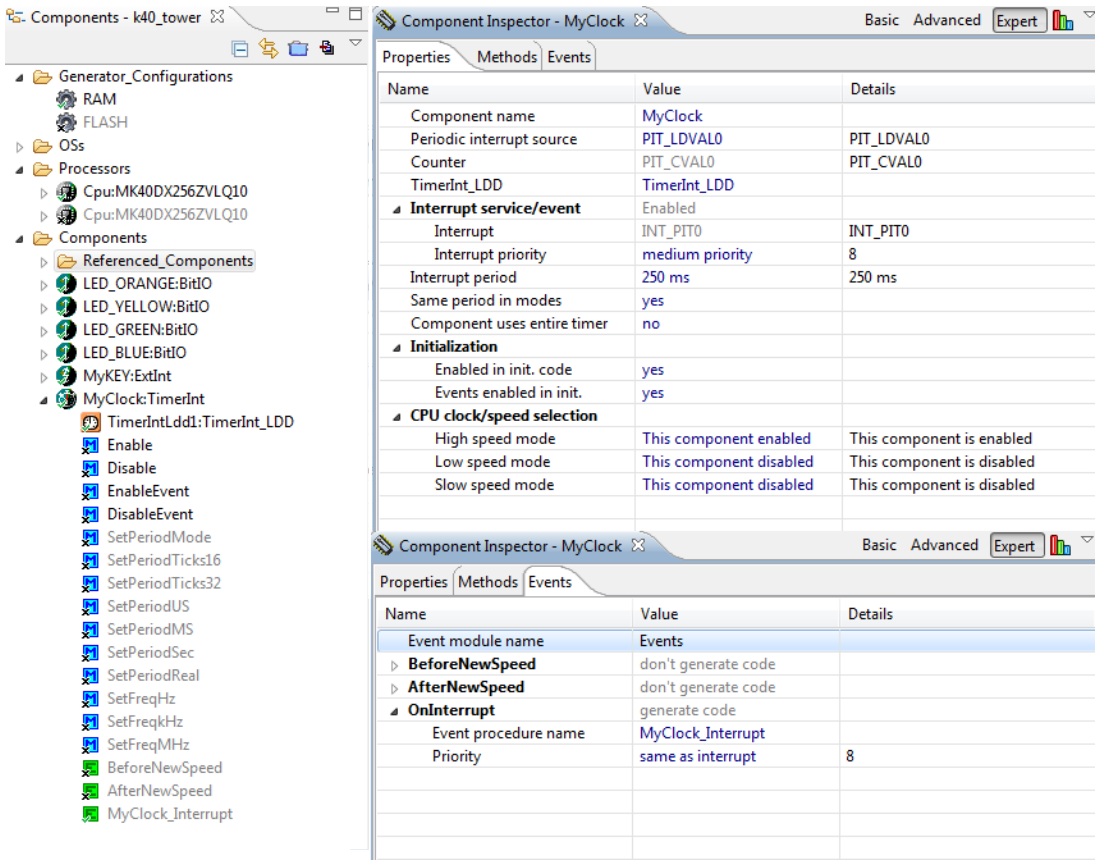
zakładka Events

Event module name: pozostawiamy *Events* - jest to nazwa modułu, w którym Processor Expert umieści funkcje obsługujące konfigurowane przerwanie,
OnInterrupt

Event procedure name: wpisujemy nazwę funkcji, którą wywoła konfigurowane przerwanie, np. *MyClock_Interrupt*.

Konfiguracja przerwania jest już zakończona. W celu zwizualizowania efektu jego działania, można posłużyć się diodą LED oraz wygenerowanymi w tym celu funkcjami. W zakładce *Files*, aktualnie otwartego projektu, należy odszukać moduł, w którym Processor Expert umieścił funkcję wywoływaną przez skonfigurowane przerwanie. W tym przypadku będzie to plik

Event.c znajdujący się w katalogu *Sources*. Po odszukaniu funkcji *MyClock* możemy posłużyć się zaproponowanym poniżej kodem.



Rysunek 6: Przykład konfiguracji PIT o częstotliwości 4Hz.

Poniżej zaprezentowano program demo, którego efektem działania jest mrugająca dioda LED znajdująca się na płytce zestawu Freescale Tower TWR-K40X256 [8].

```
/*
** ===================================================================
**     Event       :  MyClock_Interrupt (module Events)
**
**     Component   :  MyClock [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the component is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void MyClock_Interrupt(void)
{
  /* Write your code here ... */
      /* Write your code here ... */
        static bool togg;

        if (togg) togg=FALSE; else togg=TRUE;
        LED_ORANGE_PutVal(togg);
}
```

8 Generator sygnału PWM

Do generowania sygnału PWM posłużymy się jednym z kanałów modułu FlexTimer FTM. Moduł ten, jest zbudowany w oparciu o, stosowany od wielu lat w mikrokontrolerach 8-biotywch, blok TPM. Został on nieco rozbudowany i wzbogacony o dodatkowe funkcje przeznaczone do sterowania napędami, oświetleniem, przetwarzaniem zasilania. Producent do pewnego stopnia zapewnia kompatybilność nowego FTM z TPM. O konfiguracji i trybach pracy FTM można przeczytać w dokumentacji [1] układu w rozdziale 37 na stronie 895. Poniżej przedstawiono przykład konfiguracji modulatora PWM o wybranych parametrach (częstotliwość i wypełnienie). Do wizualizacji działania można posłużyć się diodą LED lub głośnikiem (BUZZER).

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Timer. Następnie, wybieramy moduł PWM, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

Component names: wpisujemy nazwę modułu, np. *PWM_LED*, *PWM_MOTOR* lub *PWM_BUZZER*,
PWM or PPG device: wpisujemy nazwę modułu oraz kanału, który zamierzamy użyć. UWAGA! Każdy kanał ma z góry przypisaną linię I/O. PE automatycznie wybierze urządzenie, jeśli od razu przejdziemy do wyboru linii,
Output pin: wybieramy linię, do której podłączono LED, silnik lub głośnik,
Interrupt service/event: nie używamy przerwań, zatem pozostawiamy *Disabled*,
Period: wpisujemy okres sygnału PWM,
Starting pulse width: wpisujemy początkowe wypełnienie,
Initial polarity: wybieramy polaryzację *low*.

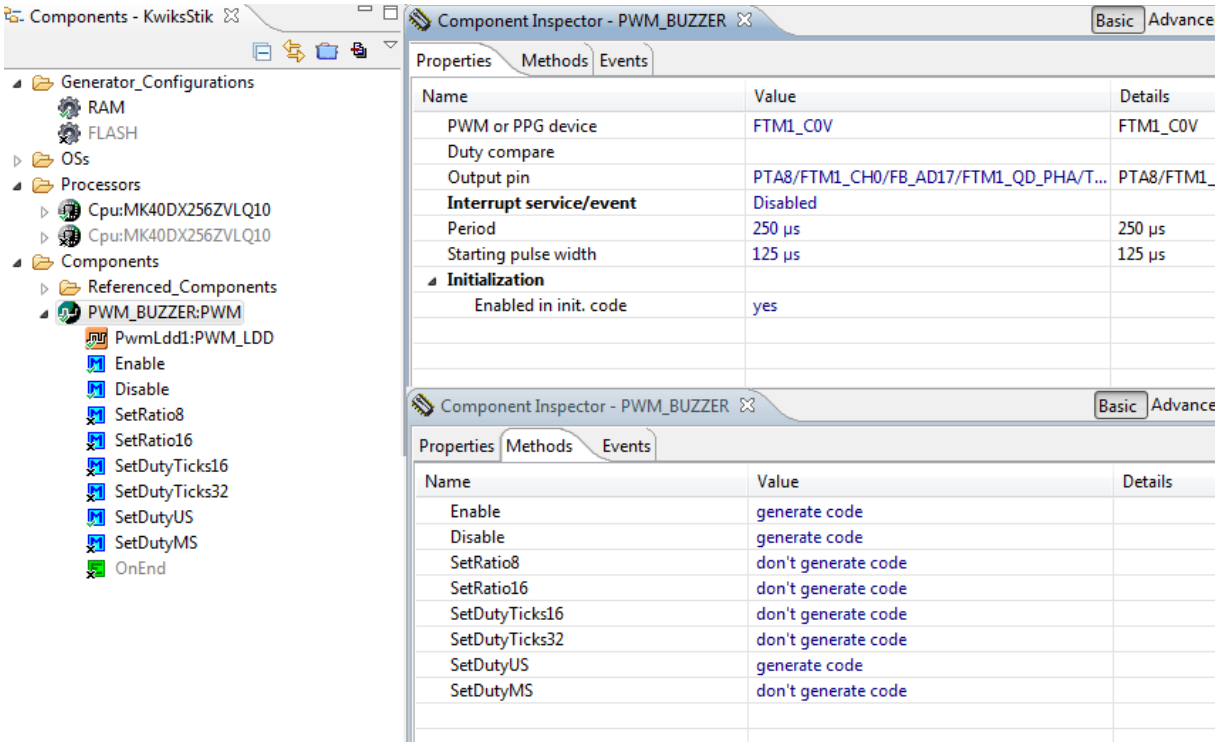
zakładka Methods

Wybieramy minimalny zestaw funkcji, potrzebnych do sensownego zadawania wypełnienia konfigurowanego generatora sygnału PWM. Jeżeli wypełnienie podajemy w μs<1ms, to nie ma sensu generować funkcji do zadawania wypełnienia w ms. Natomiast warto również dodać funkcje włączające i wyłączające generator.

zakładka Events

Powyższa konfiguracja nie generuje zdarzeń/przerwań.

Poniżej zaprezentowano program demo, którego efektem działania jest cykliczne wydawanie dźwięku o częstotliwości 4kHz przez głośnik znajdujący się na płytce Freescale KwikStik [7].



Rysunek 7: Przykład konfiguracji generatora sygnału PWM.

```
/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
    bool togg=FALSE;
    int i=0;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  for(;;) {

      if(togg)
          {
              togg=FALSE;
              PWM_BUZZER_SetDutyUS(20);
          } else {
              togg=TRUE;
              PWM_BUZZER_SetDutyUS(200);
          }
      for (i=0;i<=1000000;i++) {}
  }
```

9 Konfiguracja przetwornika A/C

Pomiaru sygnałów analogowych możemy dokonać jednym z dwóch przetworników ADC. Mikrokontroler, znajdujący się na stanowisku laboratoryjnym, pozwala na 8,10,12,14,16-bitowy pomiar 24 sygnałów analogowych. Dokładny opis konfiguracji ADC można znaleźć w dokumentacji [1] układu w rozdziale 32 na stronie 767. Przed przystąpieniem do tego zadania, należy posłużyć się schematem zestawu laboratoryjnego.

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Converter/ADC. Następnie, wybieramy moduł ADC, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

Component name: nadajemy nazwę modułu, np. *MyADC*,
A/D converter: wybieramy jeden z dwóch modułów np. *ADC1*,
Interrupt service/event: pozostawiamy bez zmian *Enabled*,
A/D channels: wybieramy liczbę kanałów pomiarowych używając klawiszy +/-,
A/D channel (pin): wybieramy linię pomiarową w zależności od wybranego źródła,
Mode select: pozostawiamy *Single Ended*, druga opcja *Differential* oznacza pomiar różnicowy i wymagane są wtedy dwie linie pomiarowe,
A/D resolution: wybieramy rozdzielczość pomiaru tak, aby precyzja pomiaru była sensownie wykorzystana,
Conversion time: wpisujemy czas pojedynczego pomiaru.

zakładka Methods

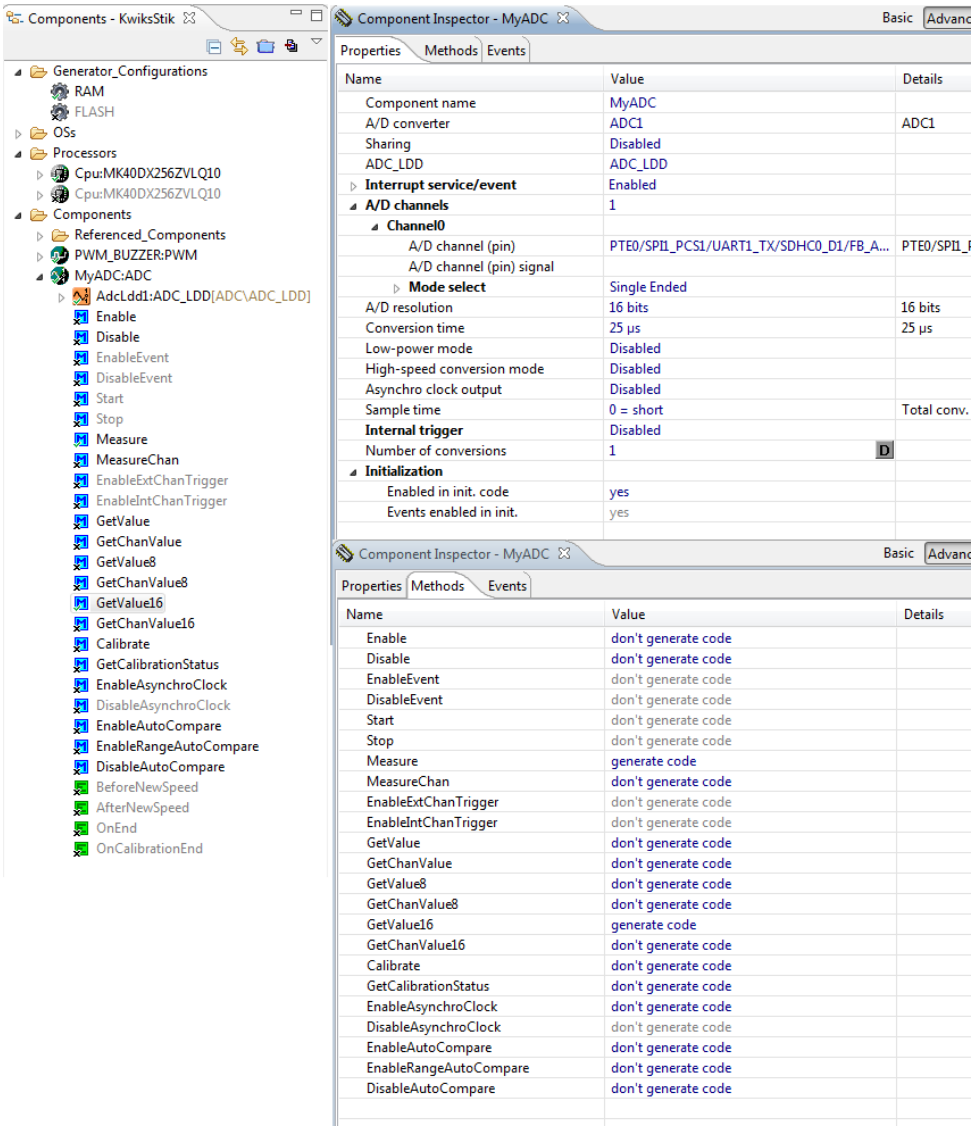
Wybieramy minimalny zestaw funkcji, potrzebnych jedynie do zainicjalizowania pomiaru oraz odczytania zmierzonej wartości.

zakładka Events

Powyższa konfiguracja nie generuje zdarzeń/przerwań.

W celu wykonania pojedynczego pomiaru oraz odczytu jego wartości należy, posłużyć się wygenerowanymi funkcjami. Warto zwrócić uwagę na to, że funkcja *MyAnalog_GetValue16* zwraca wynik (8,10,12,14-bitowy) w postaci 16-bitowej zmiennej z wartością wyrównaną do lewej. Poniżej zaproponowano przykładowy kod.

Poniżej zaprezentowano program demo, którego efektem działania jest pomiar wartości napięcia analogowego na lini B51 szyny PRIMARY modułu KwikStik [7].



Rysunek 8: Przykład konfiguracji jednego kanału pomiarowego.

```
/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
    word measure=0;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  for(;;) {
      MyADC_Measure(TRUE);  // wait for result
      MyADC_GetValue16(&measure);
  }
```

10 Konfiguracja przetwornika C/A

Generowania sygnału analogowego możemy dokonać jednym z dwóch przetworników DAC. Mikrokontroler, znajdujący się na stanowisku laboratoryjnym, pozwala na generowanie sygnału analogowego z rozdzielczością 12-bitów. Dokładny opis konfiguracji DAC można znaleźć w dokumentacji [1] układu w rozdziale 34 na stronie 851. Przed przystąpieniem do tego zadania, należy posłużyć się schematem zestawu laboratoryjnego. **UWAGA! Dla tego układu brakuje w PE modułu typu CPU Internal Peripherials, w związku z tym posłużymy się modułem typu LLD (Logical Device Drivers)**. W oknie Componets Library/Logical Device Drivers należy wybrać katalog Converter. Następnie, wybieramy moduł DAC, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

Component name: nadajemy nazwę modułu, np. *MyDAC*,
D/A converter: wybieramy jeden z dwóch modułów np. *DAC0*,
Interrupt service/event: wybieramy *Disabled* - brak przerwań,
Output pin: pozostawiamy *yes*,
D/A channel (pin): pozostawiamy jedyną dla wybranego modułu dostępną linię *DAC0_OUT*,
Init value: wpisujemy wartość początkową na wyjściu DAC,
D/A resolution: pozostawiamy jedyną dostępną w tym układzie opcję *12 bits*,
Data mode: wybieramy sposób, w jaki będą podawane dane do przetwornika,
Voltage reference source: wybieramy *external*,
Initialization
Auto initialization: UWAGA! Moduły typu LDD mają domyślnie wyłączoną autoinicjalizację, w zależności od potrzeb można ją włączyć w tryb automatyczny (w trakcie startu) lub ręcznie (po starcie) - wybieramy *yes*.

zakładka Methods

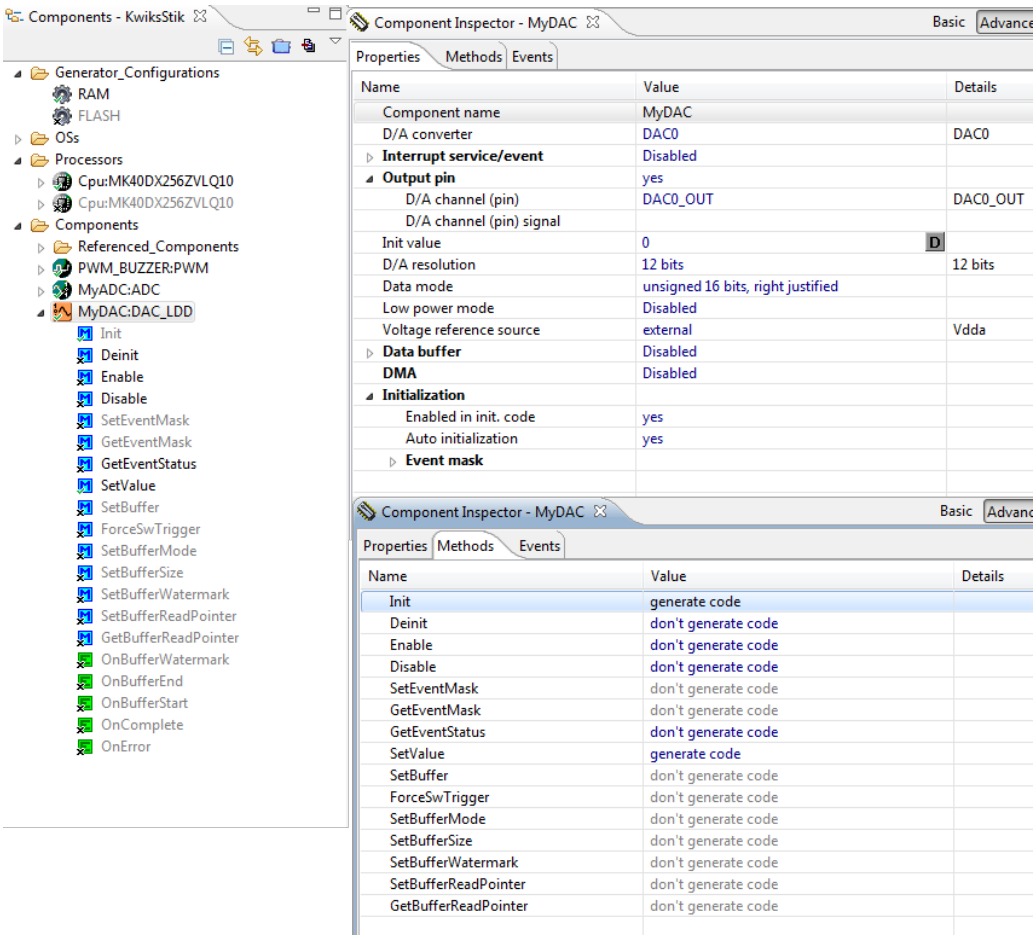
Wybieramy minimalny zestaw funkcji, potrzebnych jedynie do zainicjalizowania przetwornika oraz zadawania żądanej wartości na wyjściu analogowym.

zakładka Events

Powyższa konfiguracja nie generuje zdarzeń/przerwań.

Poniżej przedstawiono przykład użycia przetwornika DAC. Warto zwrócić uwagę, że funkcje (Methods) modułów typu LDD wymagają podania wskaźnika na strukturę danych urządzenia UserDataPtr. Wykorzystując zaprojektowane, z myślą o systemach RTOS, moduły typu LDD, do standardowej pracy z płytką ewaluacyjną można w tym miejscu podać wartość NULL.

Poniżej zaprezentowano program demo, którego efektem działania jest generowanie sinusoidalnego sygnału analogowego na linii B32 szyny PRIMARY modułu KwikStik [7].



Rysunek 9: Przykład konfiguracji przetwornika DAC.

```
/* User includes (#include below this line is not maintained by Processor Expert) */
#include <math.h>

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
    float time=0;
    int i,tmp;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  for(;;) {
      time+=0.001;
      tmp=2000+(int)(sinf(time)*2000);
      (void)MyDAC_SetValue(NULL,(LDD_DAC_TData)tmp);
      for(i=0;i<=10000;i++) {}
  }
```

11 Konfiguracja interfejsu UART

Zastosowany na zajęciach laboratoryjnych mikrokontroler, posiada 6 kanałów komunikacji UART. Poniższy przykład prezentuje, jak skonfigurować jeden z nich w celu przesłania danych, np. do komputera PC. Dokładny opis konfiguracji UART można znaleźć w dokumentacji [1] układu w rozdziale 48 na stronie 1337. Przed przystąpieniem do tego zadania, należy posłużyć się schematem zestawu laboratoryjnego.

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Communication. Następnie, wybieramy moduł AsynchroSerial, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

Component name: nadajemy nazwę modułu, np. *MyUART*,
Channel: wybieramy jeden z 5 kanałów UART,
UWAGA! Przerwania nie są konieczne, zwłaszcza jeśli używamy komunikacji tylko do nadawania pojedynczych znaków. Przerwania należny włączyć, jeśli chcemy CPU był natychmiast informowany o nadejściu znaku lub w sytuacji, kiedy chcemy użyć programową kolejkę FIFO. Niezbędne jest również określenie wielkości buforów nadawczego lub/i odbiorczego.
Interrupt service/event: wybieramy *Enabled*,

Input buffer size: jeżeli zezwoliliśmy na przerwania, to wpisując wartość większą od 0 utworzymy kolejkę dla odbieranych znaków (danych),

Output buffer size: jeżeli zezwoliliśmy na przerwania, to wpisując wartość większą od 0 utworzymy kolejkę dla nadawanych znaków (danych),
Settings - wybieramy parametry transmisji,

Parity: bit parzystości,

Width: długość znaku, zwykle *8* bitów,

Receiver: jeżeli korzystamy tylko z wysyłania, możemy odbiornik wyłączyć,

RxD: wybieramy linię odbiorczą,

Transmitter: jeżeli korzystamy tylko z odbierania, możemy nadajnik wyłączyć,

TxD: sprawdzamy/wybieramy linię nadawczą,

RxD: wybieramy linię nadawczą,

Baudrate: wpisujemy prędkość transmisji,

Break signal: pozostawiamy *Disabled*.

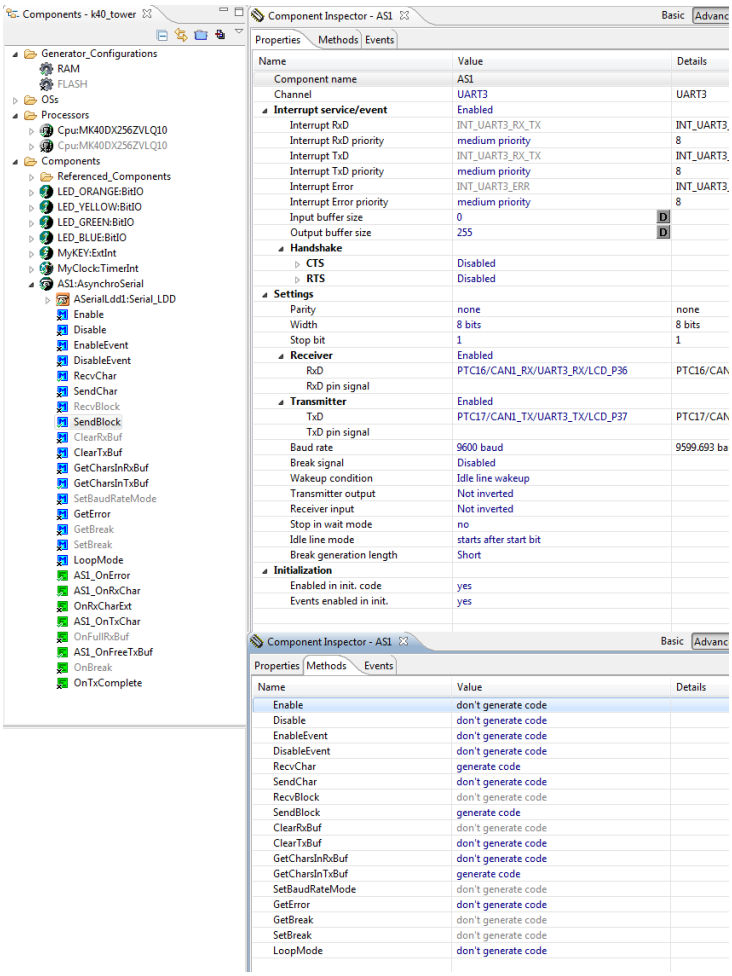
zakładka Methods

Wybieramy minimalny zestaw funkcji, potrzebnych jedynie do zainicjalizowania transmisji oraz pobierania danych z buforów.

zakładka Events

Pozostawiamy bez zmian. W zależności od powyższych ustawień, PE wygeneruje przerwania na potrzeby działania kolejki FIFO oraz informujące o odebraniu/wysłaniu pojedynczego znaku.

Poniżej zaprezentowano program demo, którego efektem jest wysyłanie do komputera napisu *"Alive nr 0"*. Każdy wysłany zestaw danych zawiera numer kolejności.



Rysunek 10: Przykład konfiguracji interfejsu UART.

```
/* User includes (#include below this line is not maintained by Processor Expert) */
#include <string.h>

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
    byte text[40];
    int i;
    int counter=0;
    word tmp;

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */

  (void)memset(text,0,sizeof(text));

  for(;;) {
      counter++;
      sprintf(text,"Alive nr %4d\n",counter);
      AS1_SendBlock(text,14,&tmp);
      for (i=0;i<=1000000;i++) {}
  }
```

12 Konfiguracja interfejsu SPI

Zastosowany na zajęciach laboratoryjnych mikrokontroler posiada 3 bloki komunikacji SPI. Poniższy przykład prezentuje, jak skonfigurować jeden z nich w celu obsługi urządzeń zewnętrznych takich jak sensory, przetworniki, ekspandery,... Dokładny opis konfiguracji SPI można znaleźć w dokumentacji [1] układu w rozdziale 46 na stronie 1251. Przed przystąpieniem do tego zadania, należy posłużyć się schematem zestawu laboratoryjnego.

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Communication. Następnie, wybieramy moduł SynchroMaster, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

Component name: nadajemy nazwę modułu, np. *MySPI*,
Channel: wybieramy jeden z 3 bloków, np. *SPI0*,
UWAGA! Przerwania nie są konieczne, zwłaszcza jeśli używamy komunikacji tylko do nadawania pojedynczych danych. Przerwania należny włączyć, jeśli chcemy użyć programowej kolejki FIFO. W takiej sytuacji niezbędne jest określenie wielkości bufora nadawczego lub/i odbiorczego.
Interrupt service/event: zezwalamy na przerwania z tego modułu *Enabled*,

Input buffer size: jeżeli zezwoliliśmy na przerwania, to wpisując wartość większą od 0 utworzymy kolejkę dla odbieranych danych,

Output buffer size: jeżeli zezwoliliśmy na przerwania, to wpisując wartość większą od 0 utworzymy kolejkę dla nadawanych danych,
Settings - wybieramy parametry transmisji,

Width: wpisujemy szerokość pojedynczej danej w bitach,

Input pin: pozostawiamy *Enabled*,

Pin: wybieramy linię odbiorczą MISO,

Input pin: pozostawiamy *Enabled*,

Pin: wybieramy linię nadawczą MOSI,

Clock pin: pozostawiamy *Enabled*,

Pin: wybieramy linię zegara CLK,

Slave select pin: wybieramy *Enabled*,

Pin: wybieramy linię wyboru Slave Select,

Clock edge: odpowiada ustawieniom CPHA - wybieramy na którym zboczu sygnału zegarowego ma nastąpić pobranie stanu linii danych,

Shift clock rate: wpisujemy czas trwania impulsu zegara, czyli prędkość transmisji, np. dla ∼1MHz czas ten powinien wynosić $\sim 1\mu$s,

Delay between chars: wpisujemy czas odstępu pomiędzy wysyłanymi danymi,

CS to CLK delay: wpisujemy czas odstępu pomiędzy wyborem urządzenia, a startem sygnału zegarowego,

CLK to CS delay: wpisujemy czas odstępu pomiędzy końcem sygnału zegarowego, a zwolnieniem urządzenia,

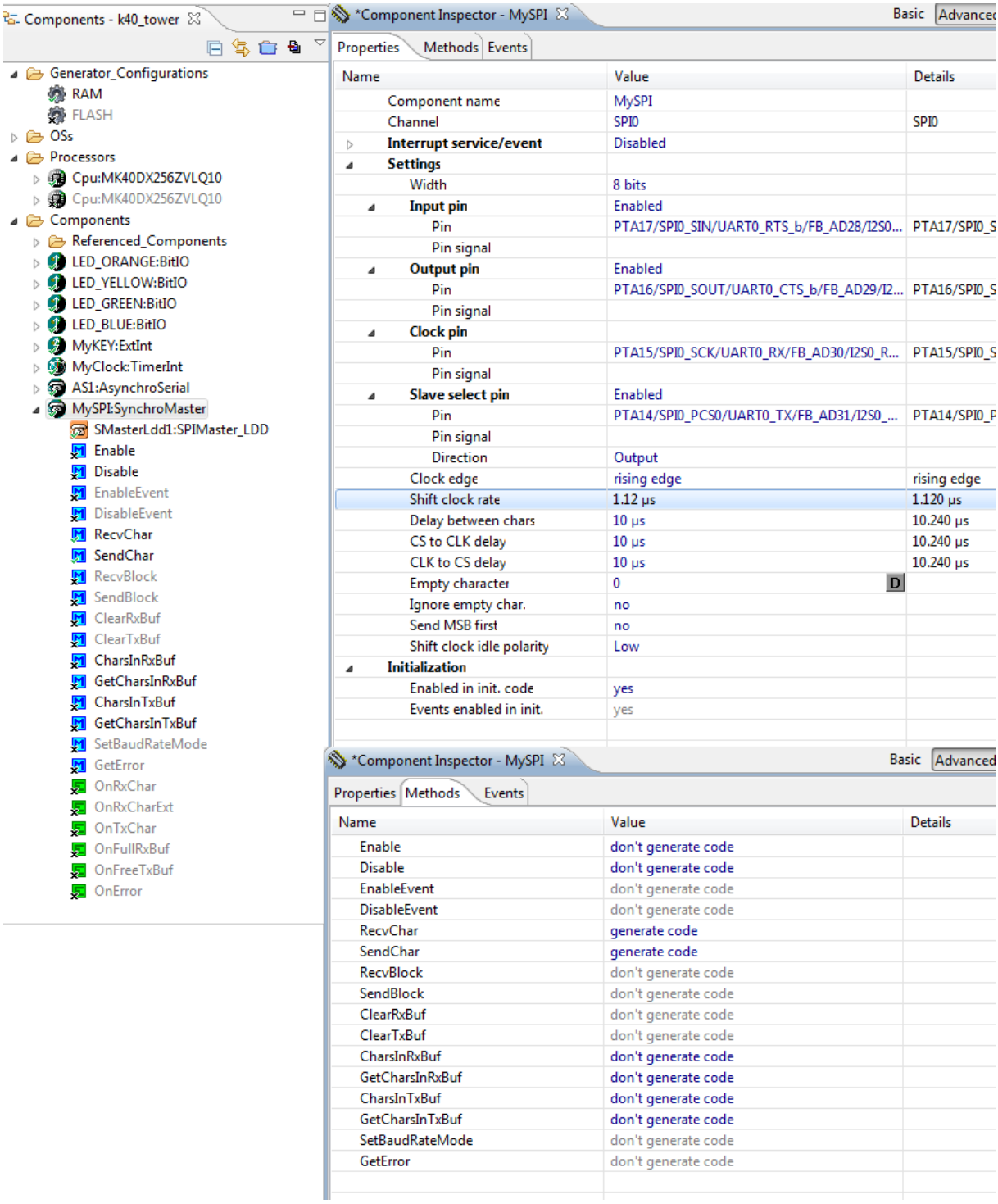
Shift clock idle polarity: odpowiada ustawieniom CPOL, czyli polaryzacją sygnału zegara.

zakładka Methods

Wybieramy minimalny zestaw funkcji, potrzebnych jedynie do zainicjalizowania transmisji oraz pobierania danych z buforów.

zakładka Events

Pozostawiamy bez zmian.



Rysunek 11: Przykład konfiguracji SPI 8-bit.

13 Konfiguracja interfejsu I^{2}C

Zastosowany na zajęciach laboratoryjnych mikrokontroler posiada 2 kanały komunikacji I^{2}C. Poniższy przykład prezentuje, jak skonfigurować jeden z nich w celu obsługi urządzeń zewnętrznych takich jak sensory, przetworniki, ekspandery,... Dokładny opis konfiguracji I^{2}C można znaleźć w dokumentacji [1] układu w rozdziale 47 na stronie 1303. Przed przystąpieniem do tego zadania, należy posłużyć się schematem zestawu laboratoryjnego.

W oknie Componets Library/CPU Internal Peripherials należy wybrać katalog Communication. Następnie wybieramy moduł InternalI2C, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

Component name: nadajemy nazwę modułu, np. *MyI2C*,
Channel: wybieramy jeden z 2 kanałów, np. *I2C1*,
Mode selection: - wybieramy *MASTER*,
Interrupt service/event: *Disabled*, wyłączamy przerwania,
Polling trials: *2000*, ilość prób na sprawdzenie potwierdzenia ACK,
Automatic stop condition: *no*, ze względu na różną długość odczytywanych/zapisywanych danych, warunek stopu będziemy generować ręcznie,
MASTER mode
Initialization
Address mode: wybieramy 7-bitowy tryb adresowania,
Target slave address init: wpisujemy adres urządzenia docelowego, można go zmienić w trakcie działania programu,
Data and Clock
SDA pin: wybieramy linię danych,
SCL pin: wybieramy linię sygnału zegara,
Internal frequency (multiplier factor): wybieramy częstotliwość zegara bloku komunikacji,
Bits 0-2 of Frequency divider register: wybieramy dzielnik powyższej częstotliwości,
Bits 3-5 of Frequency divider register: wybieramy dzielnik powyższej częstotliwości,
SCL frequency - ostatecznie obserwujemy uzyskaną częstotliwość pracy interfejsu. Warto zwrócić uwagę, czy jest ona dopuszczalna przez producenta układu, z którym zamierzamy się komunikować.

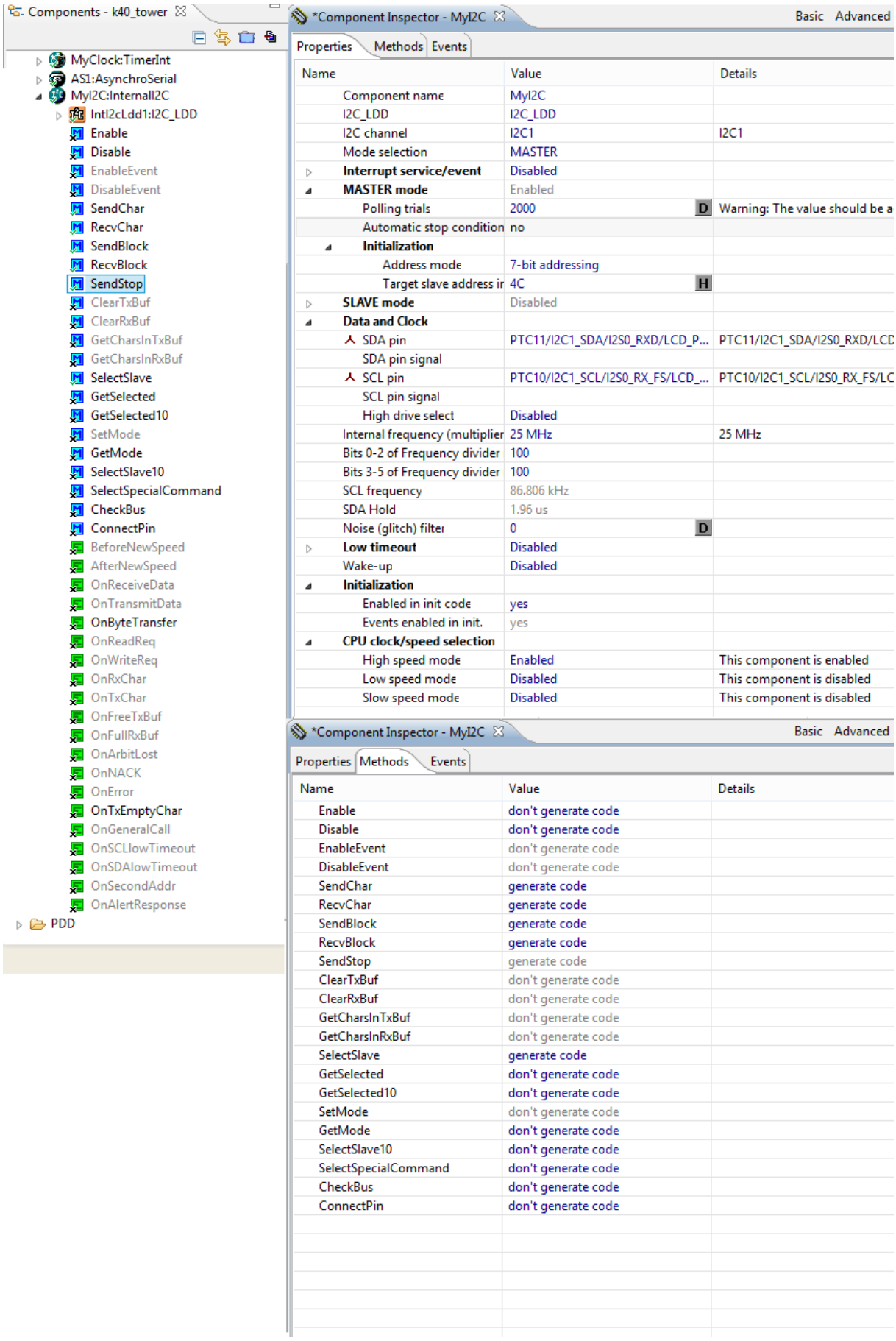
zakładka Methods

Wybieramy minimalny zestaw funkcji, potrzebnych jedynie do zainicjalizowania transmisji oraz pobierania danych z buforów. Należy zwrócić uwagę na funkcję *SendStop()*, gdyż w powyższej konfiguracji nie używamy automatycznego generowania warunku stopu.

zakładka Events

Nie używamy przerwań.

Poniżej zaprezentowano program demo, którego efektem jest odczytanie pomiarów z akcelerometru MMA7660, znajdującego się na płytce TWR-K40X256 [8].



Rysunek 12: Przykład konfiguracji interfejsu I2C.

```
/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
int i;
byte send_frame[2];
byte receive_frame[3];
word tmp;

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  ////////////////////////////////////////////
  // see Document Number: MMA7660FC
  // page 16
  // set register address
  // 0x07: Mode Register
  send_frame[0] = 0x07;
  // Set MODE to 1 - Active Mode
  //       IAH  IPP SCPS  ASE  AWE  TON  --- MODE
  // 0x01    0    0    0    0    0    0    0    1
  send_frame[1] = 0x01;
  MyI2C_SendBlock(send_frame,2,&tmp);
  MyI2C_SendStop();

  for(;;) {
      // delay
      for (i=0;i<100000;i++){}
      // set to register address
      // 0x00: XOUT 6-bit output value X
      MyI2C_SendChar(0x00);
      // get 3 data registers XOUT, YOUT, ZOUT
      MyI2C_RecvBlock(receive_frame,3,&tmp);
      MyI2C_SendStop();
  }
```

14 Konfiguracja dekodera kwadraturowego

Mikrokontroler K40X256VLQ100 posiada trzy timery FTM (Flex Timer Module). Są to trzy uniwersalne, wielokanałowe, konfigurowalne moduły czasowo-licznikowe FTM0, FTM1 oraz FTM2. Pierwszy, posiada 8 kanałów, kolejne jedynie po dwa kanały. Moduły FTM1 oraz FTM2 producent przewidział z myślą o pracy jako dekodery kwadraturowe. Naturalnie, wszystkie moduły mogą pracować także jako IC, OC, PWM... Dokładny opis konfiguracji modułu FTM można znaleźć w dokumentacji [1] układu w rozdziale 37 na stronie 895. Natomiast opis konfiguracji TPM w trybie QDEC na stronie 940.

Niestety Processor Expert nie posiada modułu do konfiguracji FTM do pracy w trybie QDEC. W związku z tym, należy samodzielnie przeprowadzić konfigurację rejestrów. Poniżej znajduje się przykład implementacji.

```
34 int main(void)
35 /*lint -restore Enable MISRA rule (6.3) checking. */
36 {
37   /* Write your local variable definition here */
38     int i;
39     int tmp=0;
40     char text[9];
41   /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
42   PE_low_level_init();
43   /*** End of Processor Expert internal initialization.                    ***/
44
45   //////////////////////////
46   // QUAD FTM1 Init
47   //////////////////////////
48
49    // Enable the clock for FTM1
50    SIM_SCGC6 |= SIM_SCGC6_FTM1_MASK;
51
52    // All registers including the FTM-specific registers (second set of registers) are available for use with no restrictions.
53    //FTM1_MODE |= FTM_MODE_FTMEN_MASK;
54
55    // Enable the counter to run in the BDM mode
56    //FTM1_CONF |= FTM_CONF_BDMMODE(3);
57
58    // Load the Modulo register and counter initial value
59    FTM1_MOD = 4095;
60    FTM1_CNTIN = 0;
61
62    // Configuring FTM for quadrature mode
63    FTM1_QDCTRL |= FTM_QDCTRL_QUADEN_MASK;
64
65    // If set it operates in Step-Dir mode, if NOT it operates in Channel_A-Channel_B mode
66    //FTM1_QDCTRL |= FTM_QDCTRL_QUADMODE_MASK;
67
68    // Set the pulse prescaler: 0-1,1-2,2-4,3-8,4-16,5-32,6-64,7-128
69    FTM1_SC |= FTM_SC_PS(2);
70
71    // Configuring the input pins:
72    PORTA_PCR8 = PORT_PCR_MUX(6); // FTM1 CH0
73    PORTA_PCR9 = PORT_PCR_MUX(6); // FTM1 CH1
74
75    //////////////////////////
76    // LCD Init Section
77    //////////////////////////
78    SLCDModule_Init();
79
80    //////////////////////////
81    // Main loop
82    //////////////////////////
83    for(;;) {
84        tmp=FTM1_CNT;
85        (void)sprintf(text,"%4d", tmp);
86        SLCD_WriteString(text, 0 ,SMALL_FONT);
87    }
```

15 Konfiguracja interfejsu dotykowego TSI

Interfejs TSI (Touch Sensing Input) to zintegrowany, m.in. w mikrokontrolerach z rodziny K-40, blok peryferyjny umożliwiający obsługę pojemnościowych klawiatur bezstykowych. Oprócz sprzętu wbudowanego w strukturę mikrokontrolerów Kinetis, producent opracował również biblioteką programistyczną TSS (Touch Sensing Software) [4], która przeznaczona jest do obsługi modułu TSI. Moduł TSI wraz biblioteką TSS stanowi kompletne rozwiązanie, umożliwiające projektantom budowanie interfejsów użytkownika, wykorzystujących klawiatury pojemnościowe. Poniższy przykład prezentuje, jak skonfigurować układ klawiatury umieszczony na płytce laboratoryjnej. Dokładny opis konfiguracji TSI można znaleźć w dokumentacji [1] układu w rozdziale 52 na stronie 1623. Przed przystąpieniem do tego zadania, należy zapoznać się z rozmieszczeniem elektrod na dostarczonej płytce laboratoryjnej.

W oknie Componets Library/SW/Tools Library należy wybrać moduł TSS_Library, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu. UWAGA! Oprogramowanie TSS nie jest dostarczone ze środowiskiem CW. Należy je pobrać ze strony Freescale i zainstalować jako dodatek.

zakładka Properties

Number of Electrodes wpisujemy liczbę elektrod niezbędnych do pełnej obsługi interfejsu dotykowego, znajdującego się na płytce laboratoryjnej - w tym celu używamy klawiszy +/-,

Electrode0 - czynność tą powtarzamy dla każdej elektrody,

Sensing Method: wybieramy dostępną, dla danej elektrody, metodę, np. *TSI Module* lub *GPIO Method*,

TSI Channel: wybieramy linię, do której podłączono elektrodę,

...

Następnie, należny zdefiniować liczbę kontrolek. Możliwe kontrolki to: KEYPAD, SLIDER, ROTARY, AROTARY, ASLIDER. MATRIX. UWAGA! Niektóre kontrolki wymagają użycia kilku elektrod, np. kontrolka typu suwak wymaga dwóch elektrod.

Number of Controls: wpisujemy liczbę kontrolek używając klawiszy +/-,

Control0 - tą czynność powtarzamy dla każdej kontrolki,

Control Type: wybieramy jaki to jest typ kontrolki,

Number of Electrodes: wpisujemy liczbę użytych elektrod dla danej kontrolki,

Struct Name: wpisujemy nazwę struktury, w której będą przechowywane dane o zdarzeniach danej kontrolki,

...

Sensor Settings

Ponadto należy pamiętać, że gdy kiedykolwiek wybierzemy metodę pomiaru *GPIO Method*, musimy dla TSS skonfigurować dodatkowy licznik.

Timers

HW Timer

Timer Name: Wybieramy jeden z wolnych bloków FTM np *FTM2_CNT*.

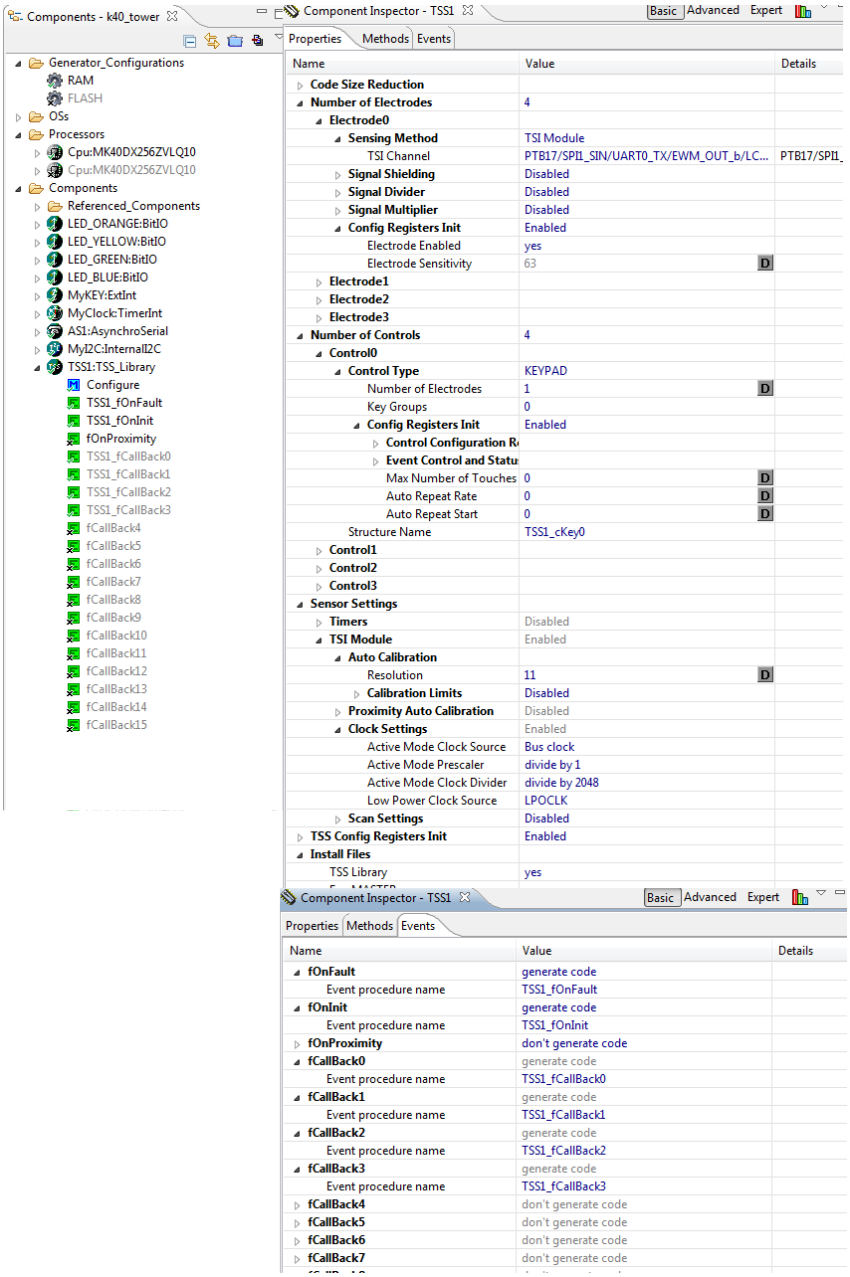
zakładka Methods

Funkcję *Configure* pozostawiamy bez zmian.

zakładka Events

W zakładce tej, należy zwrócić uwagę na nazwy funkcji przerwań, które zostały przypisane poszczególnym kontrolkom. Deklaracje tych funkcji można odnaleźć w pliku *Events.c*

W zależności od powyższych ustawień, PE wygeneruje kompletną konfigurację oraz ustawi odpowiednie przerwania, na potrzeby działania poszczególnych kontrolek. Po zakończonej konfiguracji konieczne jest ręczne wywołanie, w głównej pętli programu, funkcji *Configure();* oraz cykliczne wywoływanie funkcji przetwarzania modułu TSS *TSS_Task*. Następnie, należy uzupełnić funkcje przerwań o odpowiednie akcje, które im towarzyszą. Możliwa jest również konfiguracja, w której użyjemy kilka elektrod dla jednej kontrolki. Dla powyższego przykładu, może to być jedno wspólne zdarzenie dla wszystkich czterech elektrod. Należy wtedy, w polu **Number of Electrodes** wpisać liczbę większą od 1. Poniżej przedstawiono przykład dla czterech elektrod skonfigurowanych jako cztery kontrolki typu KEYPAD. Każda z nich zgłasza niezależne zdarzenie.

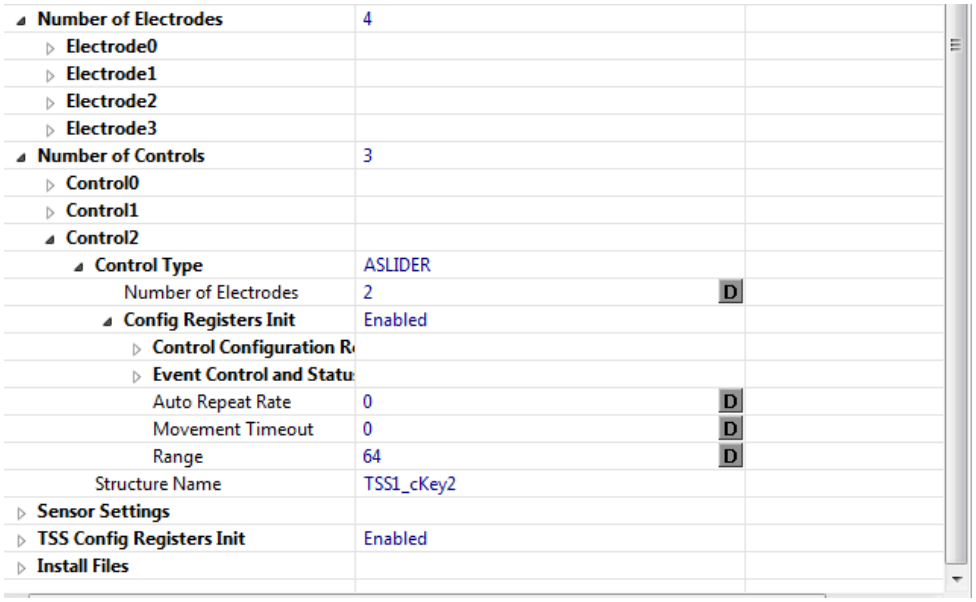


Rysunek 13: Przykład konfiguracji interfejsu dotykowego.

```c
/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  Configure();

  /* Write your code here */
  /* For example: for(;;) { } */

  for(;;) {
      TSS_Task();
  }

  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
```

```c
** ===================================================================
**     Event       :  TSS1_fCallBack0 (module Events)
**
**     Component   :  TSS1 [TSS_Library]
**     Description :
**         Callback definition for Control 0. This event is enabled
**         only if Control 0 is enabled.
**         The default CallBack Name is automatically generated with
**         automatic prefix update by current Component Name. User can
**         define own name, but then the automatic name update is not
**         functional.
**     Parameters  :
**         NAME            - DESCRIPTION
**         u8ControlId     - Valid unique Identifier of
**                           the Control which generated the CallBack
**                           function. This Id can be used for finding
**                           of Callback's source Control.
**     Returns     : Nothing
** ===================================================================
*/
void TSS1_fCallBack0(TSS_CONTROL_ID u8ControlId)
{
  UINT8 u8Event; /* 8 bits local variable used to store the event information */
    static bool togg;

  while (!TSS_KEYPAD_BUFFER_EMPTY(TSS1_cKey0))  /* While unread events are in the buffer */
  {
    TSS_KEYPAD_BUFFER_READ(u8Event,TSS1_cKey0); /* Read the buffer and store the event in the u8Event variable */

    /* Write your code here ... */

    if (togg) togg=FALSE; else togg=TRUE;
    LED_ORANGE_PutVal(togg);
    (void) u8Event;
  }

  (void) u8ControlId;
  return;
}
```

Kolejny przykład przedstawia, jak przy pomocy TSS skonfigurować ASLIDER. Należy zwrócić uwagę, że ASLIDER wykorzystuje dwie elektrody. Ponadto, wymaga dodatkowo określenia zakresu (w przykładzie wpisano 64).



Rysunek 14: Przykład konfiguracji dla kontrolki ASLIDER.

```
/*
** ===================================================================
**     Event       :  TSS1_fCallBack2 (module Events)
**
**     Component   :  TSS1 [TSS_Library]
**     Description :
**         Callback definition for Control 2. This event is enabled
**         only if Control 2 is enabled.
**         The default CallBack Name is automatically generated with
**         automatic prefix update by current Component Name. User can
**         define own name, but then the automatic name update is not
**         functional.
**     Parameters  :
**         NAME            - DESCRIPTION
**         u8ControlId     - Valid unique Identifier of
**                           the Control which generated the CallBack
**                           function. This Id can be used for finding
**                           of Callback's source Control.
**     Returns     : Nothing
** ===================================================================
*/
void TSS1_fCallBack2(TSS_CONTROL_ID u8ControlId)
{
  UINT8 u8Event; /* 8 bits local variable used to store the event information */
    static bool togg;

    LED_ORANGE_PutVal(TSS1_cKey2.Position<=16);
    LED_YELLOW_PutVal(TSS1_cKey2.Position>16 && TSS1_cKey2.Position<=32);
    LED_GREEN_PutVal(TSS1_cKey2.Position>32 && TSS1_cKey2.Position<=48);
    LED_BLUE_PutVal(TSS1_cKey2.Position>48 && TSS1_cKey2.Position<=64);


  (void) u8ControlId;
  return;
}
```

16 Konfiguracja wyświetlacza LCD

Mikrokontrolery z rodziny K-40 wyposażone są w sprzętowy sterownik wyświetlacza LCD. Wbudowany kontroler umożliwia bezpośrednie podłączenie wyświetlaczy typu segmentowego do wyprowadzeń układu. Posiada on aż 52 linie, które mogą pracować jako 8 - typu backplane lub/i 44 - typu frontplane (w rozkładzie 8x40 lub 4x44). Ostatecznie pozwala na sterowanie maksymalnie 320 segmentami. Konfiguracji kontrolera LCD dokonuje się dla konkretnego modelu wyświetlacza. KwikStik [7] wyposażony jest w graficzny wyświetlacz LCD (37x8 punktów), składający się z 306 segmentów. Natomiast zestaw TWR K40X256 [8] wyposażony jest w moduł TWRPI-SLCD z wyświetlaczem numerycznym (18:88), składającym się z 28 segmentów. Opis obydwu modeli można znaleźć w dokumentacji zestawów [7, 8].

Dokładny opis konfiguracji bloku LCD można znaleźć w dokumentacji [1] układu w rozdziale 34 na stronie 851. Przed przystąpieniem do tego zadania, należy posłużyć się schematem zestawu laboratoryjnego.

W przypadku wyświetlaczy LCD posłużymy się modułem typu LLD (Logical Device Drivers). W oknie Componets Library/Logical Device Drivers należy wybrać katalog Display. Następnie, wybieramy moduł SegLCD_LDD, który po dwukrotnym kliknięciu znajdzie się w zakładce Components - nazwa_projektu aktualnie otwartego projektu.

zakładka Properties

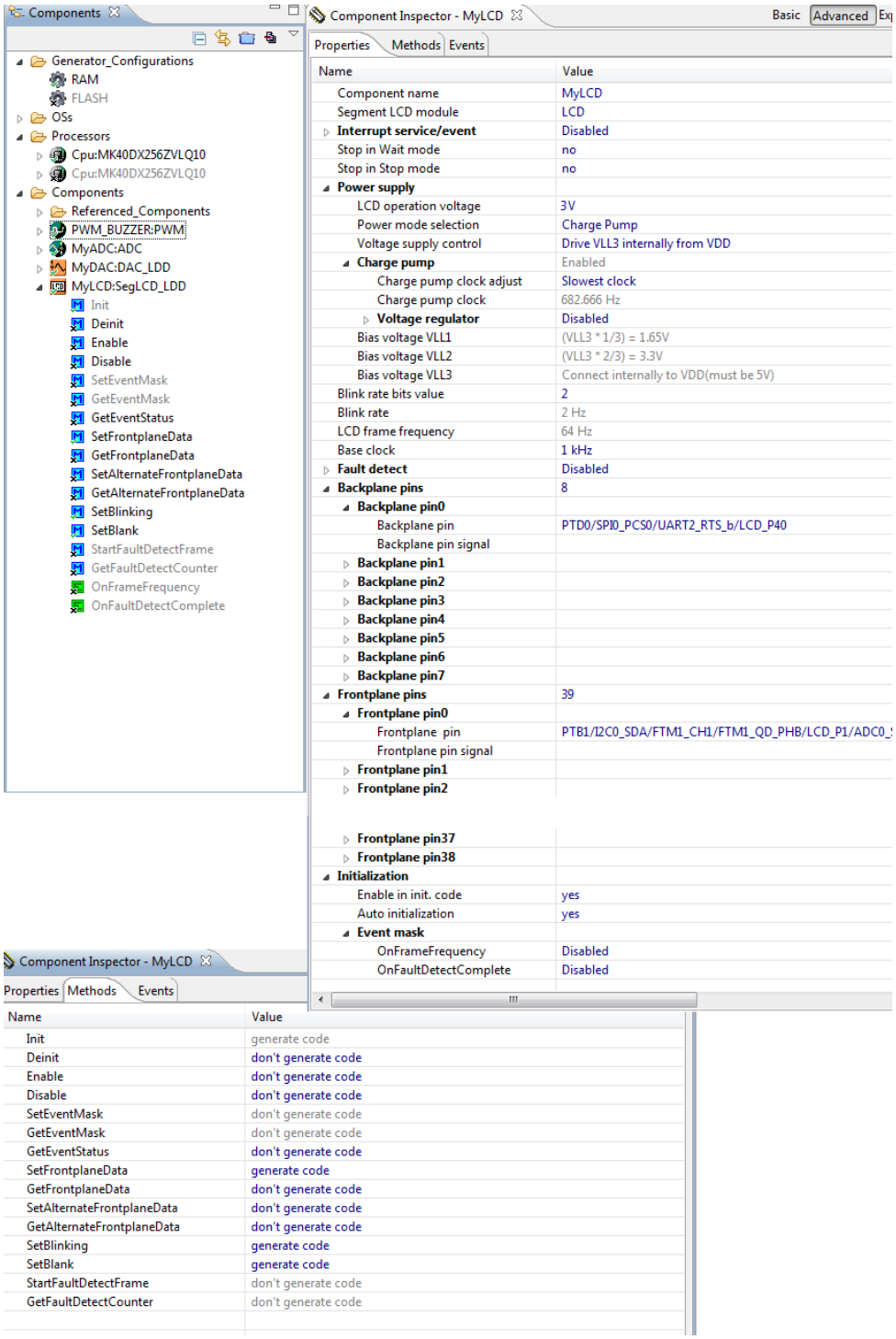
Component name: nadajemy nazwę modułu, np. *MyLCD*,
Interrupt service/event: wybieramy *Disabled* - brak przerwań,
Power supply
LCD operation voltage: wybieramy *3V*,
Power mode selection: pozostawiamy *Charge Pump*,
Voltage supply control: wybieramy *Drive VLL3 internally from VDD*, jest to tryb w jakim zasilany jest LCD (rozdział 53.4.4 na stronie 1692 w dokumentacji układu [1]),
Blink rate bits value: wpisujemy *2*, przy pomocy tego dzielnika ustawiamy częstotliwość mrugania wyświetlacza,
Blink rate: w tym miejscu wyświetli się częstotliwość mrugania (zależy od *Base clock* oraz *Blink rate bits*),
Blink rate: w tym miejscu wyświetli się częstotliwość odświeżania, (zależy od *Base clock*),
Base clock: wpisujemy częstotliwość zegara bazowego *1kHz*,
Backplane pins: wpisujemy odpowiednio dla wyświetlacza, np. *8*,
Backplane pin0 - tą czynność powtarzamy dla wszystkich linii,
Backplane pin: wybieramy odpowiednią linię,
...
Frontplane pins: wpisujemy odpowiednio dla wyświetlacza, np. *39*,
Frontplane pin0 - tą czynność powtarzamy dla wszystkich linii,
Frontplane pin: wybieramy odpowiednią linię,
...
Initialization,
Auto initialization: wybieramy *yes*.

zakładka Methods

Wybieramy minimalny zestaw funkcji, potrzebnych jedynie do zainicjalizowania wyświetlacza oraz zaświecania segmentów.

zakładka Events

Powyższa konfiguracja nie generuje zdarzeń/przerwań.



Rysunek 15: Przykład konfiguracji LCD.

```
/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
    int i=0;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */

  // write on LCD one horizontal line in the middle (4th row)
  for(i=1;i<=37;i++) { MyLCD_SetFrontplaneData(NULL, i, 8);}

  // turn ON freescale logo
  MyLCD_SetFrontplaneData(NULL, 0, 16);


  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY TH
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END ProcessorExpert */
/*
```

Powyższa konfiguracja wraz z przykładami może posłużyć jedynie do wyświetlania pojedynczych segmentów. Do pracy z tego typu wyświetlaczami warto przygotować odpowiedni zestaw funkcji w postaci plików źródłowych lub nagłówkowych dostarczających gotowych funkcji, np. do wyświetlania napisów. Dla wspomnianych już wyświetlaczy (KwikStik, TWRPI-SLCD) opracowano biblioteki umożliwiające kompletną inicjalizację, wyświetlanie napisów oraz liczb. Biblioteki te można znaleźć na stronie kursu:
http://lirec.ict.pwr.wroc.pl/ jkedzier/index.php/courses/6.html

Biblioteka dla KwikStik:
http://lirec.ict.pwr.wroc.pl/~jkedzier/download/laborki/cw3/SLCD_KWIKSTIK.zip

Biblioteka dla TWRPI-SLCD:
http://lirec.ict.pwr.wroc.pl/~jkedzier/download/laborki/cw4/SLCD_TWRPI.zip

Literatura

[1] *K40 Sub-Family Reference Manual*, Document Number: K40P144M100SF2RM, Rev. 3, 4 Nov 2010, Freescale Semiconductors Inc.

[2] *CodeWarrior Development Studio for Microcontrollers V10.x Getting Started Guide*, Revised: January 12, 2011, Freescale Semiconductors Inc.

[3] *Touch Sensing Software Users Guide*, Document Number: TSSUG, rev. 5, 08/2012

[4] *CodeWarrior for Microcontrollers V10.x Processor Expert User Manua*, Help version 4.24, Freescale Semiconductor, Inc.

[5] *www.freescale.com*, Freescale Semiconductors Inc.

[6] *www.arm.com*, Architecture for the Digital World

[7] KwikStik-K40, User's Manual, Rev. 1, Freescale Semiconductors Inc.

[8] TWR-K40X256 Tower Module, User's Manual, Rev. 1, Freescale Semiconductors Inc.