

Konfiguracja przetworników A/C i C/A oraz przerwania sprzętowego

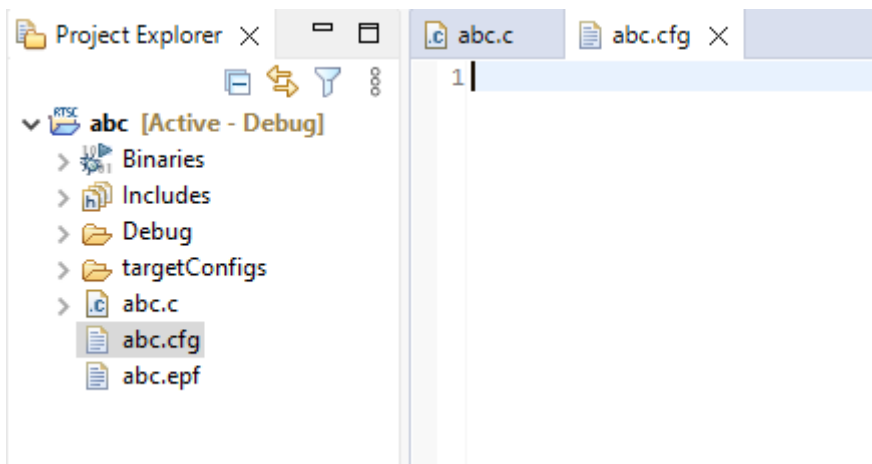
Wykorzystanie funkcji systemu RTOS do obsługi złożonych układów peryferyjnych znacznie skraca czas opracowania i uruchomienia programu. W niniejszym opracowaniu pokazano, w jaki sposób zdefiniować w systemie RTOS proste zadanie (Task) uruchamiane jednorazowo przy starcie programu (nazywane tu zadaniem początkowym). W zadaniu początkowym docelowo umieszczona zostanie procedura inicjalizacji kodeka (zawierającego przetworniki A/C i C/A) poprzez interfejs I2C, procedura inicjalizacji portu mcasp (I2S) wykorzystywanego do transmisji danych z kodekiem oraz odblokowanie przerwania sprzętowego do obsługi kodeka. Tak przygotowany program wraz z prostą zawartością procedury obsługi przerwania sprzętowego obsługującego kodek stanowi bazę do kolejnych ćwiczeń.

1. Rozszerzenie projektu abc o moduł sysbios (jądro systemu RTOS) i jedno zadanie (Task) początkowe

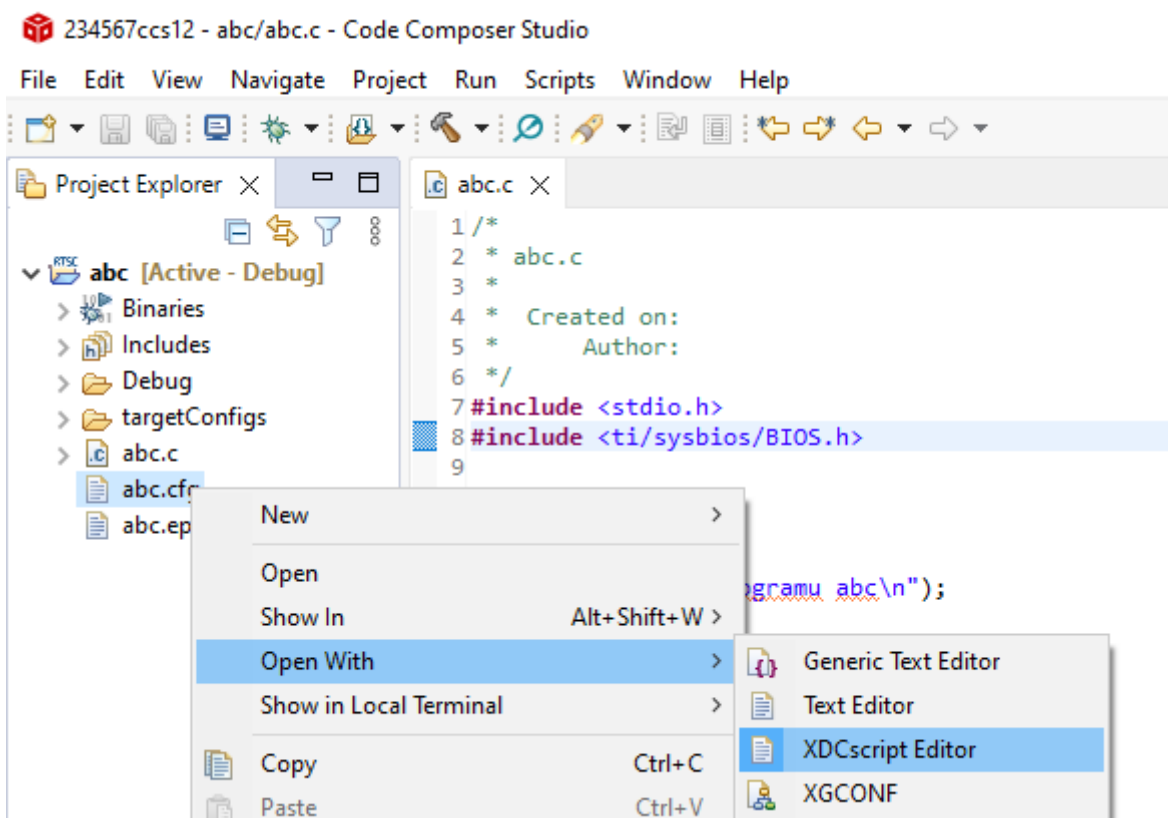
Wykorzystanie funkcji systemu RTOS wymaga załadowania modułu sysbios, co wykonuje się wywołaniem funkcji BIOS_start. Zgodnie z opisem uzyskanym np. w menu Help -> Search -> BIOS_start -> module ti.sysbios.BIOS należy dołączyć moduł w pliku źródłowym poprzez `#include <ti/sysbios/BIOS.h>`:

```
1 /*
2  * abc.c
3  *
4  * Created on:
5  * Author:
6  */
7 #include <stdio.h>
8 #include <ti/sysbios/BIOS.h>
9
10 int main(void)
11 {
12
13     printf("Start programu abc\n");
14     BIOS_start();
15     return (0);
16 }
17
18
```

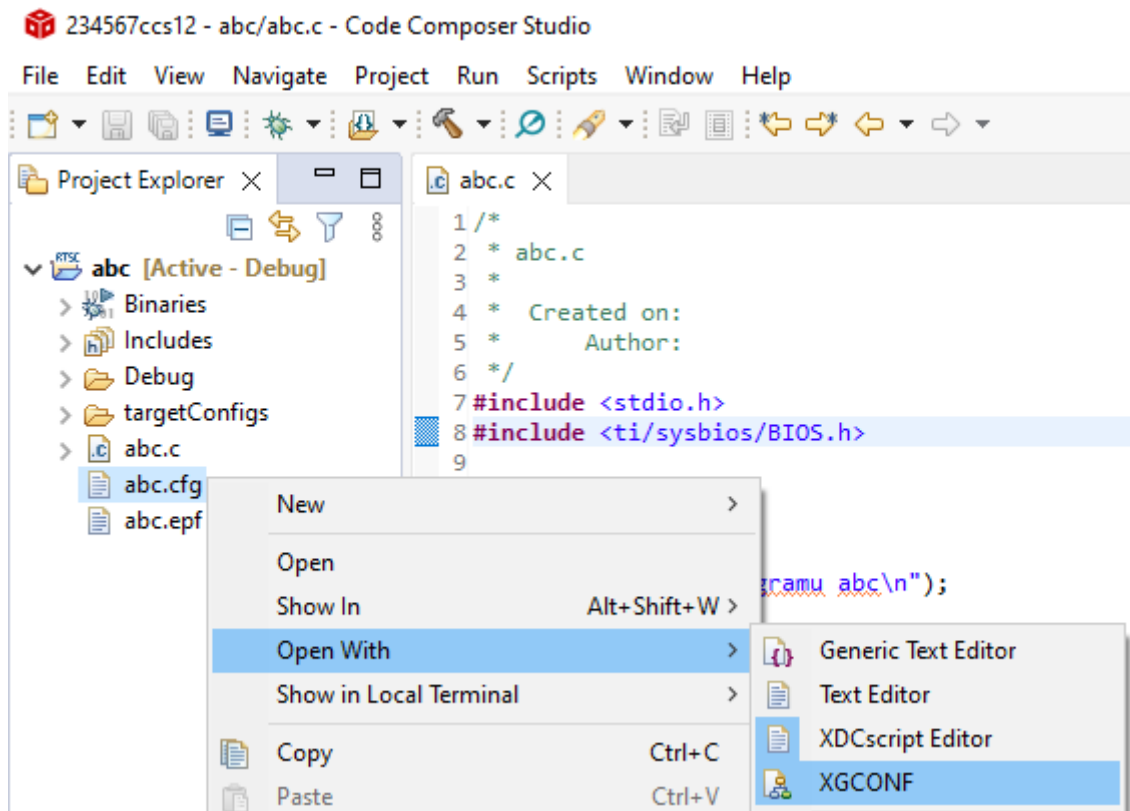
Trzeba jeszcze włączyć moduł sysbios w pliku konfiguracyjnym RTOS projektu, czyli abc.cfg. Dwukrotny klik myszki na nazwie abc.cfg w 'Project Explorer' otwiera ten plik do edycji w trybie tekstowym (początkowo plik ten jest pusty):



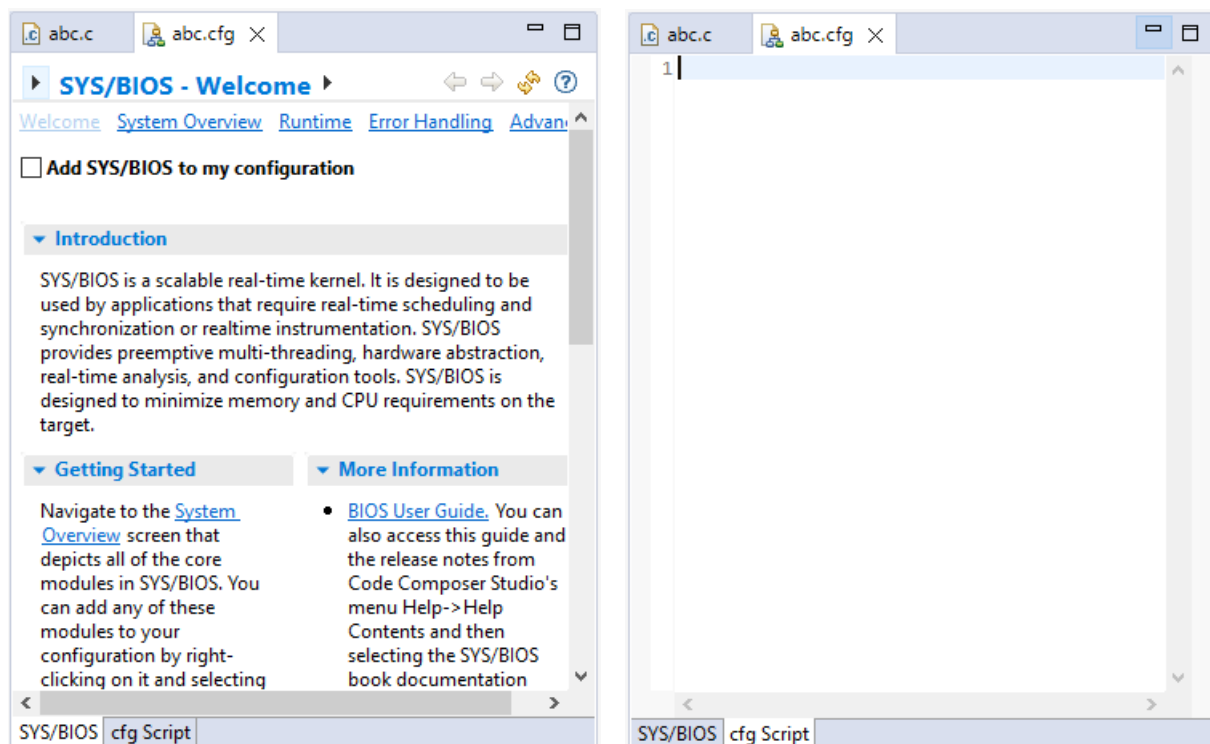
Taki sposób konfiguracji wymaga jednak posługiwania się odpowiednimi poleceniami na podstawie dokumentacji RTOS. Ten sam efekt uzyskujemy przy wciśnięciu prawego klawisza myszy na nazwie pliku i wybraniu 'Open With' -> 'XDCscript Editor':



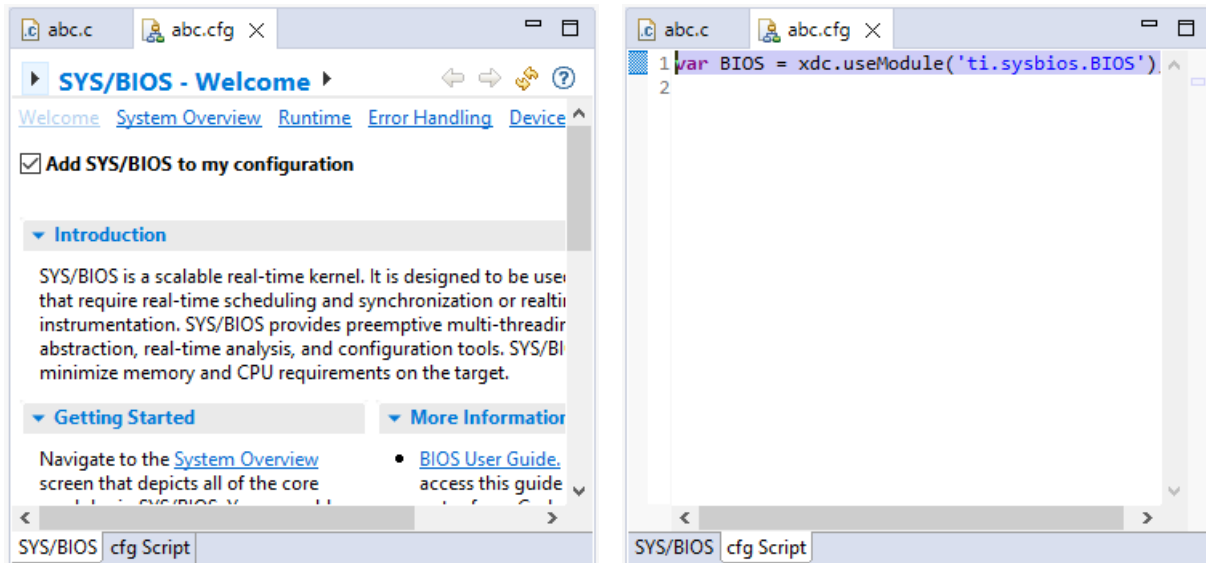
Znacznie wygodniejszym jest wykorzystanie edytora XGCONF:



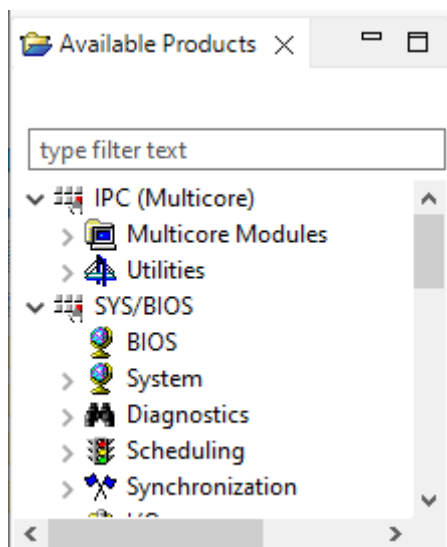
W ten sposób uzyskujemy możliwość edycji ustawień dla modułu sysbios w sposób graficzny (zakładka SYS/BIOS) lub w tekstowy (zakładka cfg Script):



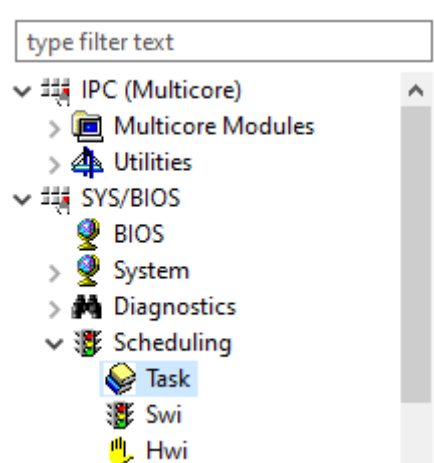
Dodanie modułu sysbios polega na zaznaczeniu 'Add SYS/BIOS to my configuration' i w ten sposób narzędzie XGCONF wprowadza odpowiedni tekst do pliku abc.cfg:



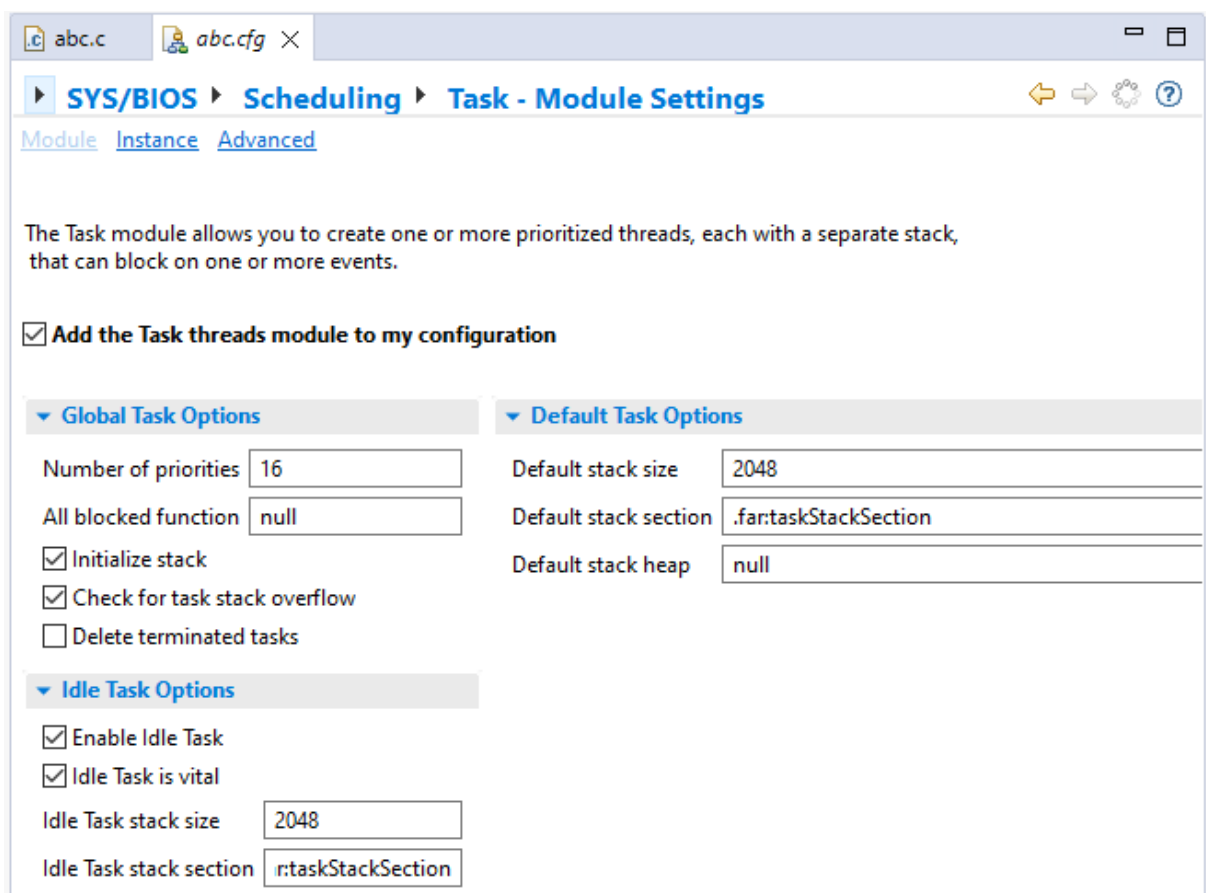
Narzędzie XGCONF pozwala również na wybór ustawień modułu sysbios poprzez kartę 'Available Products' dostępną po lewej stronie (jeśli okienko 'Available Products' nie jest widoczne – np. po resecie widoku – można go dodać poprzez menu View -> Other -> 'Show View' -> RTSC -> 'Available Products' -> Open):



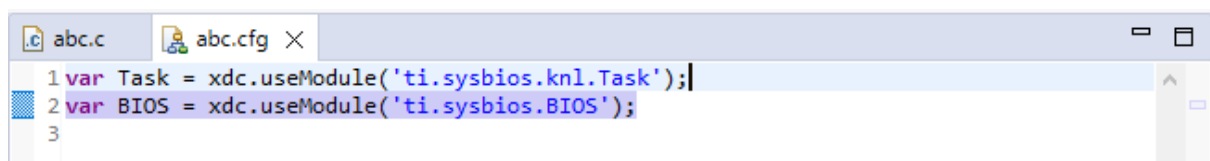
W celu konfiguracji zadania (Task) początkowego rozwijamy gałąź 'Scheduling' i wybieramy 'Task':



Następnie wracamy do edytora i zakładki SYS/BIOS i włączamy opcję 'Add the task threads to my configuration':

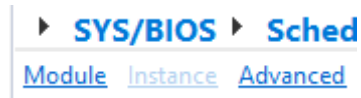


W pliku abc.cfg pojawia się odpowiednie polecenie konfiguracyjne:

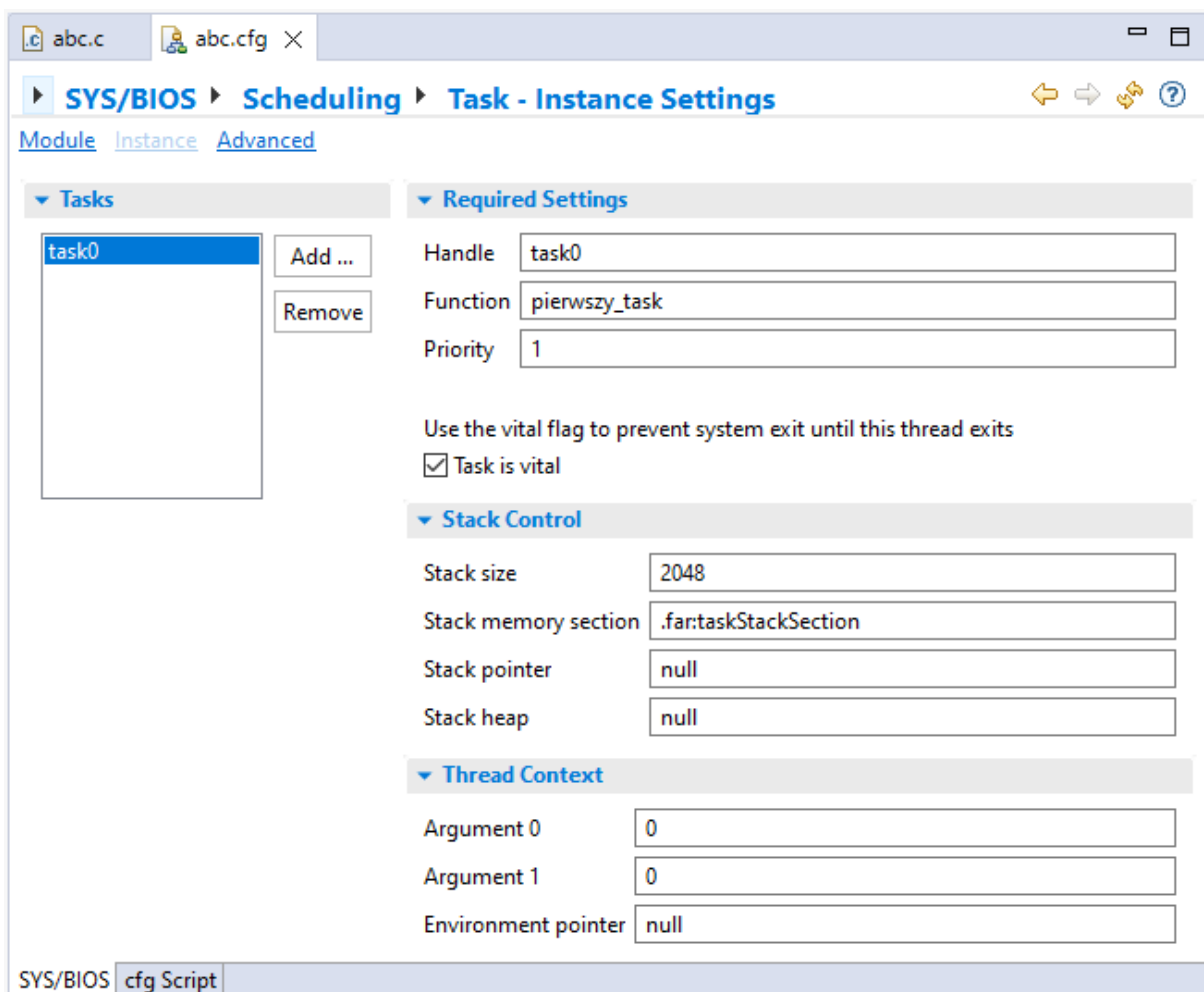


Uwaga: każde zadanie obsługiwane jest przez swoją własną stertę (stack). Dla prostego zadania (jak w tym przypadku) wielkość sterty można pozostawić domyślną. Dla bardziej złożonych zadań należy wielkość sterty zwiększyć w przypadku jej przepełnienia, które jest sygnalizowane komunikatem przy uruchomieniu, jeśli włączona jest opcja 'Check for task stack overflow'.

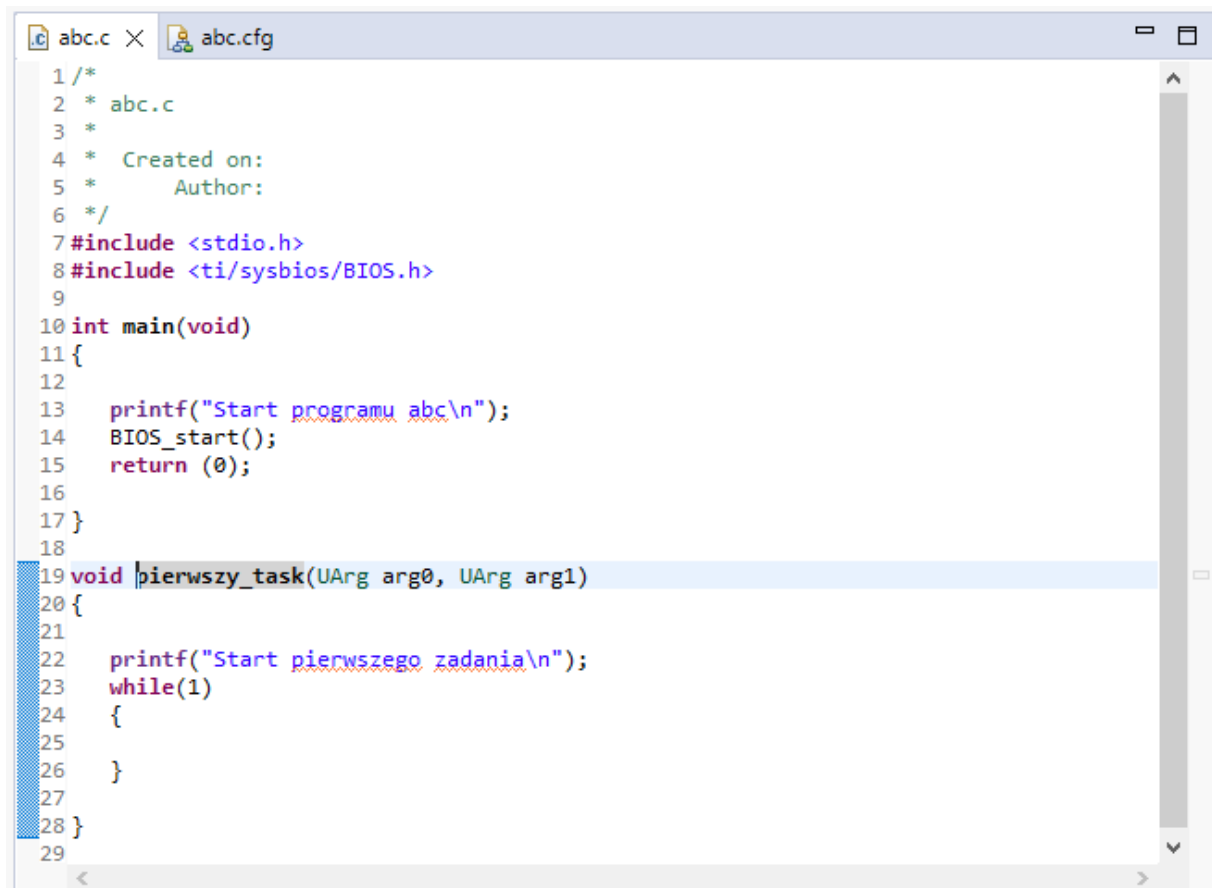
Następnie wybieramy w górnej części 'Instance':



i opcją 'Add..' dodajemy nowy task i jego nazwę (tutaj 'pierwszy_task'):

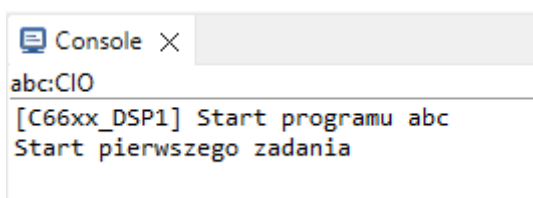


Następuje odpowiednia modyfikacja pliku abc.cfg. Zauważmy, że z tym zadaniem związane są dwa argumenty ('Thread Context' -> Argument 0, Argument 1), co uwzględniamy przy definicji funkcji pierwszy_task w pliku źródłowym języka C:



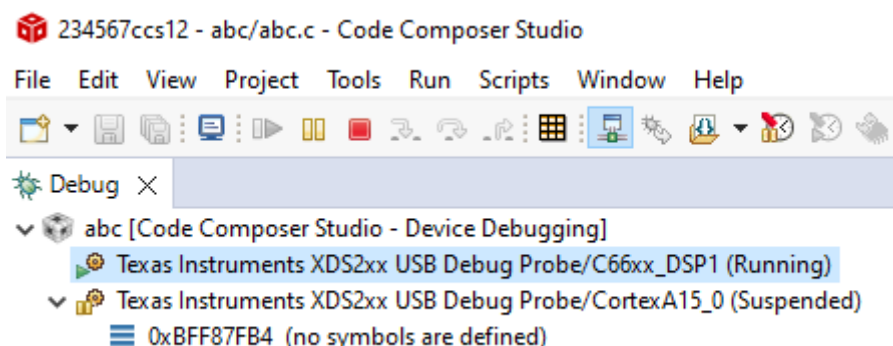
```
1 /*
2  * abc.c
3  *
4  * Created on:
5  * Author:
6  */
7 #include <stdio.h>
8 #include <ti/sysbios/BIOS.h>
9
10 int main(void)
11 {
12
13     printf("Start programu abc\n");
14     BIOS_start();
15     return (0);
16 }
17
18
19 void pierwszy_task(UArg arg0, UArg arg1)
20 {
21
22     printf("Start pierwszego zadania\n");
23     while(1)
24     {
25
26     }
27 }
28 }
29
```



Następnie kompilujemy program i jeśli nie ma błędów sprawdzamy jego działanie:

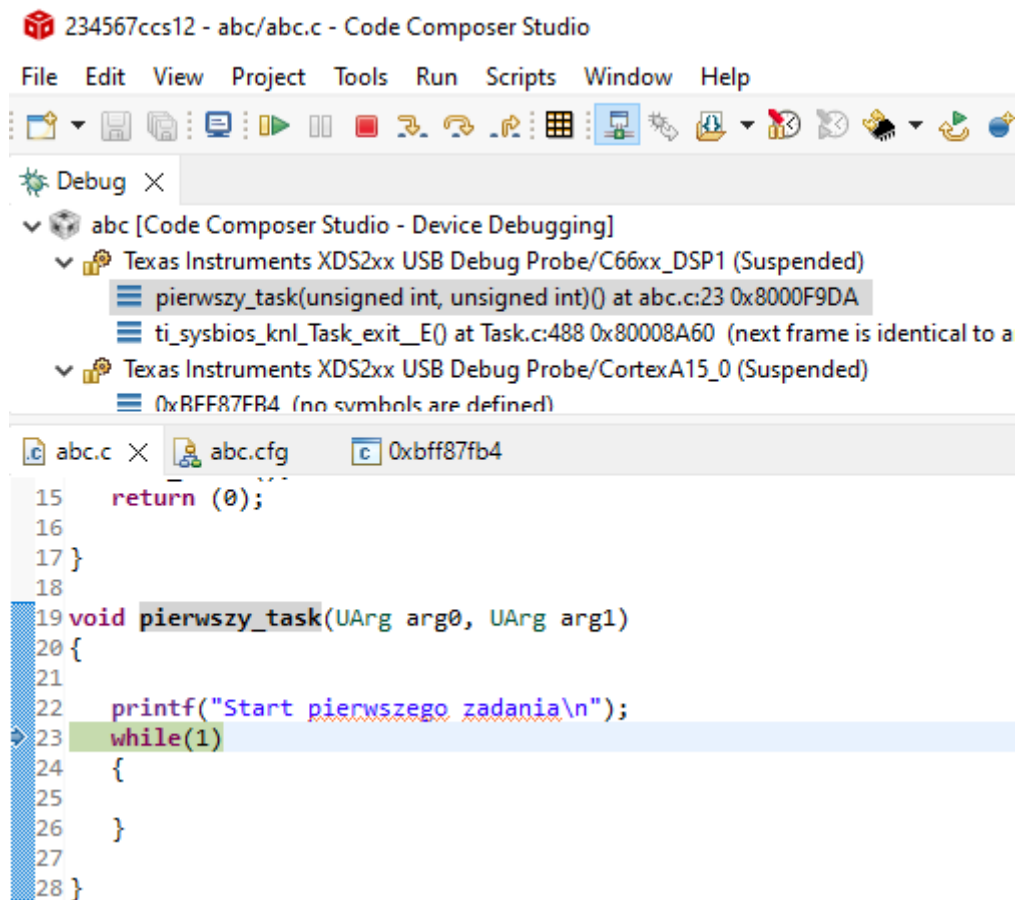


```
Console X
abc:CIO
[C66xx_DSP1] Start programu abc
Start pierwszego zadania
```

W okienku 'Debug' komunikat 'Running' informuje o działającym programie na rdzeniu DSP:



Program można zatrzymać poprzez  (Suspend), ale zatrzymanie nastąpi w losowym miejscu (wskazanym przez wskaźnik ) – w tym przypadku łatwo jednak przewidzieć, że zatrzymanie nastąpi w nieskończonej pętli while pierwszego zadania:



234567ccs12 - abc/abc.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

Debug X

abc [Code Composer Studio - Device Debugging]

✓ Texas Instruments XDS2xx USB Debug Probe/C66xx_DSP1 (Suspended)

pierwszy_task(unsigned int, unsigned int)() at abc.c:23 0x8000F9DA


ti_sysbios_knl_Task_exit__E() at Task.c:488 0x80008A60 (next frame is identical to a

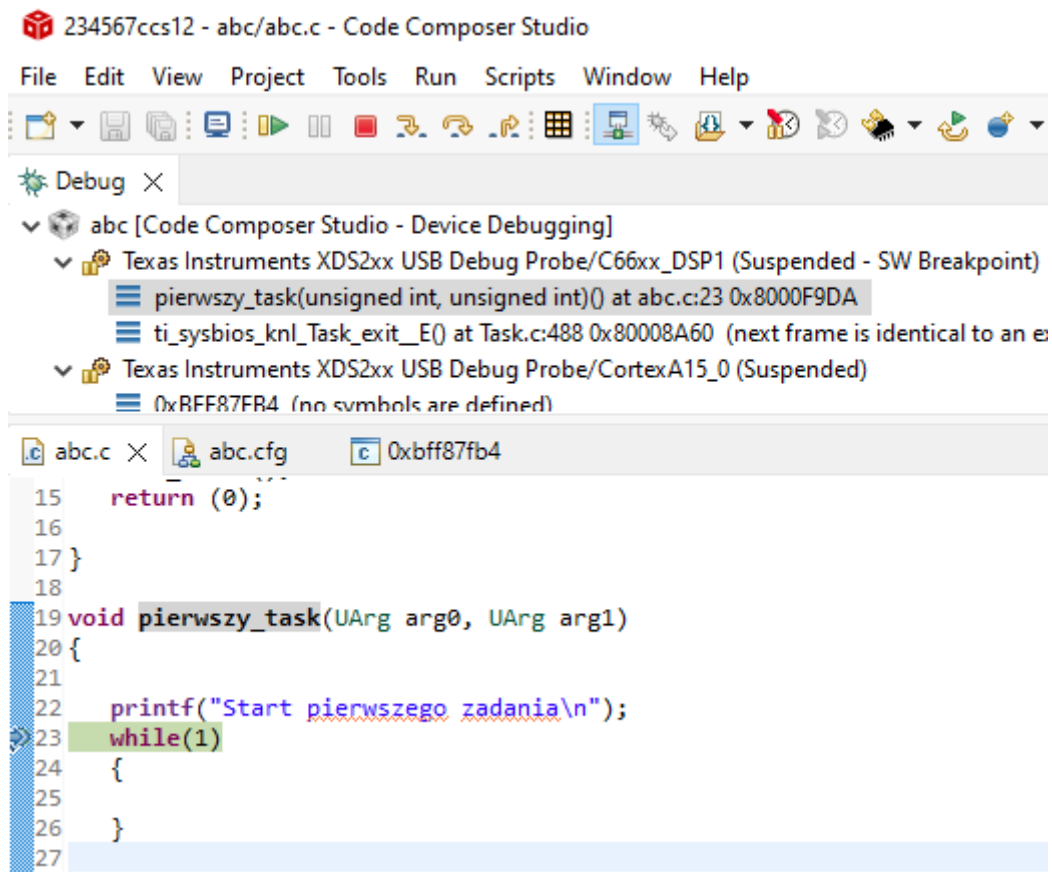
✓ Texas Instruments XDS2xx USB Debug Probe/CortexA15_0 (Suspended)

0xBFF87FB4 (no symbols are defined)

abc.c X abc.cfg 0xbff87fb4

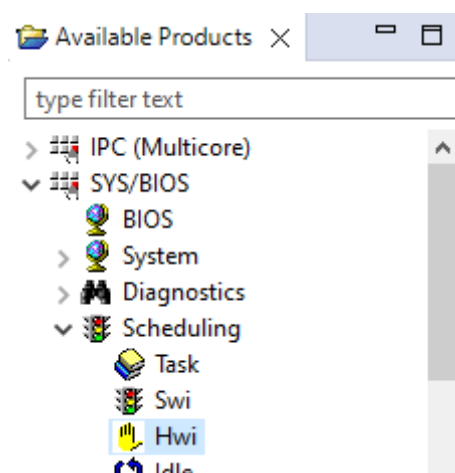
```
15 return (0);
16
17 }
18
19 void pierwszy_task(UArg arg0, UArg arg1)
20 {
21
22 printf("Start pierwszego zadania\n");
23 while(1)
24 {
25
26 }
27
28 }
```

Kontynuacja programu następuje poprzez  (Resume, F8). Zatrzymanie jest możliwe również w sposób bardziej kontrolowany (co do miejsca zatrzymania) poprzez wstawienie breakpointa, np. w linii 23 (dwukrotny klik myszy na lewym marginesie) programu. W oknie Debug mamy wtedy komunikat dla rdzenia DSP 'Suspended – SW Breakpoint':



2. Rozszerzenie programu abc o obsługę przetworników A/C i C/A (kodeka) z wykorzystaniem systemu przerwania sprzętowych

W pierwszym kroku dodajemy i konfigurujemy potrzebne moduły z modułu sysbios. Włączamy moduł HWI:





► **SYS/BIOS** ► **Scheduling** ► **Hwi - Module Settings**

[Module](#) [Instance](#) [Advanced](#)

The Hwi module provides a portable interface to define and synchronize with h service routines. Application code that restricts itself to using this module for h synchronization can be used without change in any SYS/BIOS supported system

☒ **Add the portable Hwi management module to my configuration**

[Device-specific Hwi Support](#)

▼ **Dispatcher**

- ☒ Enable interrupt nesting
- ☒ Enable software interrupt support
- ☒ Enable Task support
- ☒ Enable IRP tracking

▼ **Stack Management**

- ☒ Initialize stack
- ☒ Check for stack overflow

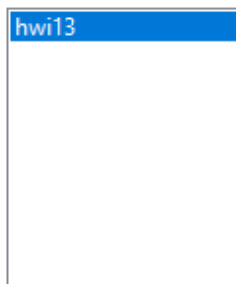
Definiujemy parametry dla przerwania nr 13: handle hwi13, nazwa procedury obsługi przerwania_rcv, priorytet 13, Identyfikator zdarzeń 94 (na podstawie dokumentacji procesora AM5729 przypisującej interfejsowi szeregowemu mscsp1 taki numer zdarzenia), wyłączony 'Enable at startup' (ze względu na konieczność wcześniejszej konfiguracji portu mscsp1 i samego kodeka):



► **SYS/BIOS** ► **Scheduling** ► **Hwi - Instance Settings**

[Module](#) [Instance](#) [Advanced](#)

▼ **Portable Hwis**



Add ...

Remove

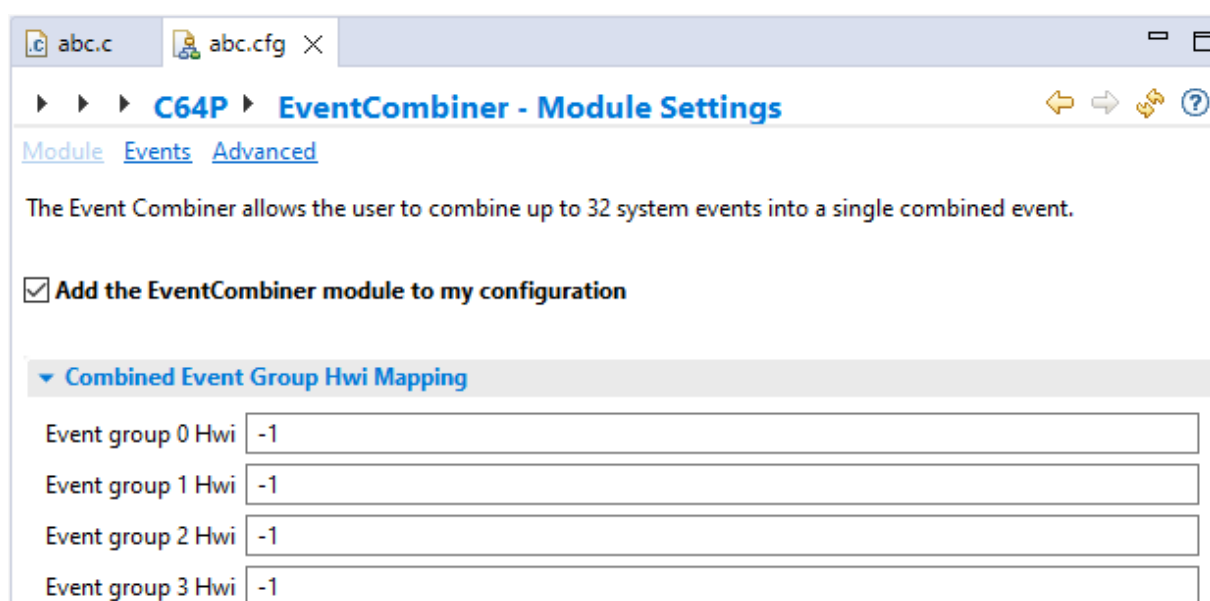
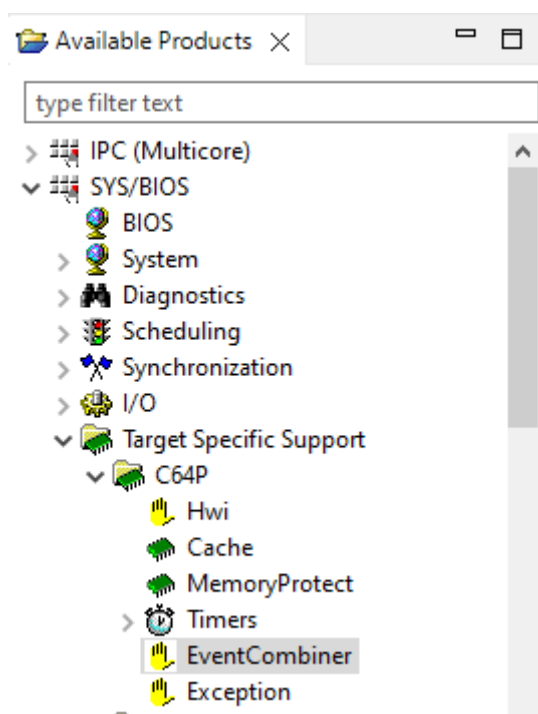
▼ **Required Settings**

Handle
ISR function
Interrupt number

▼ **Additional Settings**

Argument passed to ISR function
Interrupt priority
Event Id
☐ Enable at startup
Masking options

Dodajemy moduł 'konfiguratora zdarzeń' ('Event Combiner') w gałęzi 'Target Specific Support\C64P':



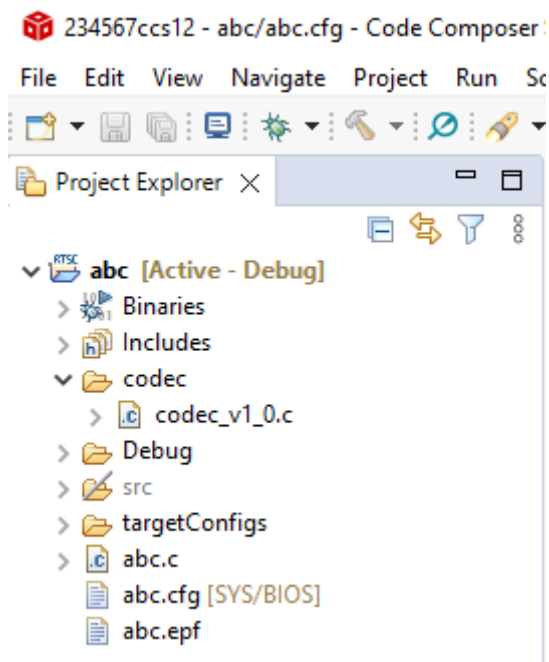
Z modułu sysbios są to już wszystkie potrzebne moduły. Warto zatem dodać kilka wierszy komentarzy w pliku abc.cfg w celu uporządkowania jego zawartości (tutaj są to wiersze 1-2, 7-9, 13-15):

```
abc.c *abc.cfg X
1 /* ===== General configuration ===== */
2
3 var Task = xdc.useModule('ti.sysbios.knl.Task');
4 var BIOS = xdc.useModule('ti.sysbios.BIOS');
5 var Hwi = xdc.useModule('ti.sysbios.hal.Hwi');
6 var EventCombiner = xdc.useModule('ti.sysbios.family.c64p.EventCombiner');
7
8 /* ===== Tasks configuration ===== */
9
10 var task0Params = new Task.Params();
11 task0Params.instance.name = "task0";
12 Program.global.task0 = Task.create("&pierwszy_task", task0Params);
13
14 /* ===== Interrupt configuration ===== */
15
16 var hwi0Params = new Hwi.Params();
17 hwi0Params.instance.name = "hwi13";
18 hwi0Params.enableInt = false;
19 hwi0Params.priority = 13;
20 hwi0Params.eventId = 94;
21 Program.global.hwi13 = Hwi.create(13, "&przerwanie_rcv", hwi0Params);
22
```

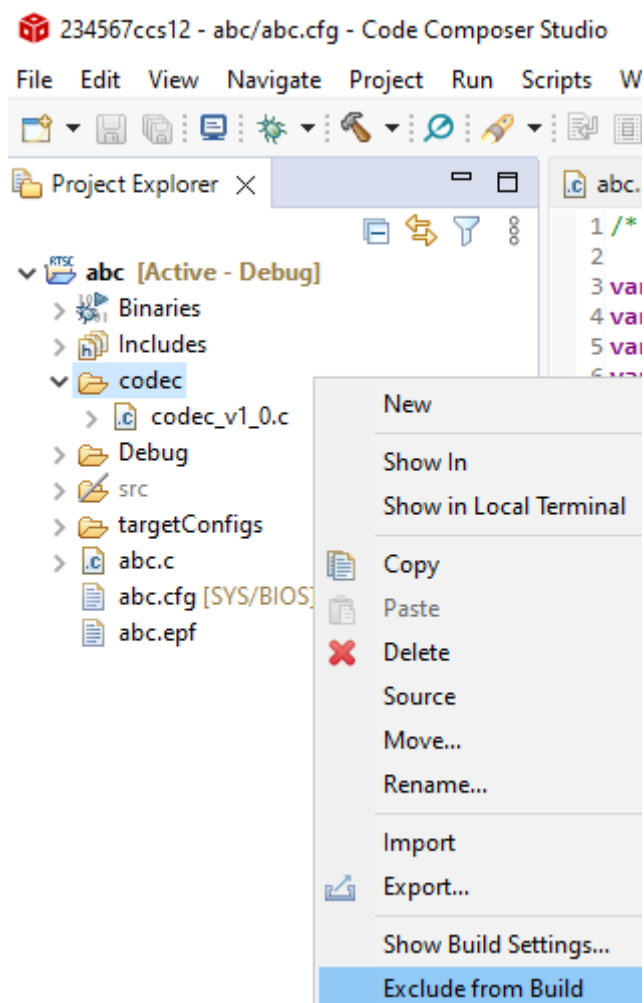
Na koniec tego pliku (od wiersza 23) przenosimy zawartość pliku `konf_i2c.cfg` z katalogu `c:\ti\bba:`

```
abc.c abc.cfg X
1 /* ===== General configuration ===== */
2
3 var Task = xdc.useModule('ti.sysbios.knl.Task');
4 var BIOS = xdc.useModule('ti.sysbios.BIOS');
5 var Hwi = xdc.useModule('ti.sysbios.hal.Hwi');
6 var EventCombiner = xdc.useModule('ti.sysbios.family.c64p.EventCombiner');
7
8 /* ===== Tasks configuration ===== */
9
10 var task0Params = new Task.Params();
11 task0Params.instance.name = "task0";
12 Program.global.task0 = Task.create("&pierwszy_task", task0Params);
13
14 /* ===== Interrupt configuration ===== */
15
16 var hwi0Params = new Hwi.Params();
17 hwi0Params.instance.name = "hwi13";
18 hwi0Params.enableInt = false;
19 hwi0Params.priority = 13;
20 hwi0Params.eventId = 94;
21 Program.global.hwi13 = Hwi.create(13, "&przerwanie_rcv", hwi0Params);
22
23 /* ===== Driver configuration ===== */
24
25 /* Load the Osal package */
26 var osType = "tirtos";
27 var Osal = xdc.loadPackage('ti.osal');
28 Osal.Settings.osType = osType;
29
30 /*use CSL package*/
31 var socType = "am572x";
32 var Csl = xdc.loadPackage('ti.csl');
33 Csl.Settings.deviceType = socType;
34
35 /* Load the i2c package */
36 var I2C = xdc.loadPackage('ti.drv.i2c');
37 I2C.Settings.socType = socType;
38
```

Następnie do podkatalogu projektu kopiujemy podkatalog c:\ti\bbai\codec z plikiem codec_v1_0.c. W oknie 'Project Explorer' powinno to dać efekt następujący:



Dla katalogu codec zaznaczamy opcję 'Exclude from Build' (co spowoduje, że w programie trzeba wyraźnie zaznaczyć dołączenie pliku z tego katalogu za pomocą komendy #include):



Ikonka katalogu w 'Project Explorer' będzie w efekcie tej zmiany przekreślona: ✂

Następnie uzupełniamy program o wiersze: 9-13, 17, 24-30, 37-40. Wiersze 18, 36 z printf zaleca się ująć w komentarz (na początkowym etapie uruchomienia można je zostawić działające):

```
*abc.c × abc.cfg
1 /*
2  * abc.c
3  *
4  * Created on:
5  * Author:
6  */
7 #include <stdio.h>
8 #include <ti/sysbios/BIOS.h> // for BIOS_start()
9 #include <xdc/cfg/global.h> // for Hwi_enableInterrupt(13)
10
11 #include "codec/codec_v1_0.c"
12
13 int probka;
14
15 int main(void)
16 {
17     Pins_config();
18     //printf("Start programu abc\n");
19     BIOS_start();
20     return (0);
21 }
22
23
24 void przerwanie_rcv()
25 {
26     probka = Read_mcaspl_rcv();
27
28     Write_mcaspl_xmt(probka);
29     Restart_mcaspl_if_error(); // It must be the last line of INT
30 }
31
32
33 void pierwszy_task(UArg arg0, UArg arg1)
34 {
35
36     //printf("Start pierwszego zadania\n");
37     Config_i2c_and_codec();
38
39     Config_and_start_mcaspl(); // these 3 lines keep together
40     Hwi_enableInterrupt(13);
41     while(1)
42     {
43
44     }
45 }
46 }
47
```

Zawartość procedur main i pierwszy_task wskazuje, że konfiguracja pinów dla kodeka następuje przed uruchomieniem BIOS_start(), natomiast konfiguracja portu i2c i zaprogramowanie kodeka oraz konfiguracja i uruchomienie portu macasp1 do obsługi

przesyłania próbek z i do kodeka odbywa się w zadaniu pierwszym, ponieważ procedury te wymagają w pełni uruchomionej warstwy BIOS poleceniem BIOS_start(). Na końcu zadania pierwszego (kiedy zaprogramowany jest kodek i uruchomiony port mcasp1) następuje odblokowanie przerwania nr 13 z przypisaną procedurą przerwanie_rcv. Przerwanie to odbiera (wiersz 26) jedną 16-bitową próbkę kanału lewego (część starsza danej 32 bitowej) oraz jedną 16-bitową próbkę kanału prawego (część młodsza danej 32 bitowej) i wysyła te próbki do przetwornika C/A (wiersz 28).

Program jest gotowy do kompilacji, poprawienia ewentualnych błędów i uruchomienia. Program przenosi sygnał z wejścia (przetwornik A/C) na wyjście (przetwornik C/A), a sprawdzenie odbywa się z wykorzystaniem generatora i oscyloskopu. W kolejnym etapie warto również sprawdzić zachowanie się programu po założeniu breakpointa w wierszu 28.