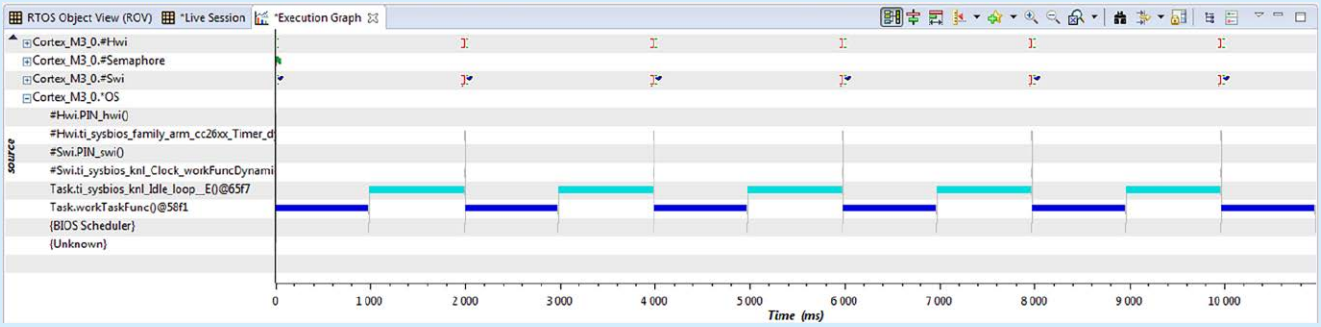


Systemy dla Internetu Rzeczy (5)

System operacyjny czasu rzeczywistego TI-RTOS – pierwszy program

Podstawowym czynnikiem efektywnego programowania systemów wbudowanych jest zapewnienie pracy w czasie rzeczywistym. Szczególnie istotne, ale i trudne, jest to w przypadku stosowania wielordzeniowych układów scalonych typu SOC (system on chip). Z taką sytuacją mamy do czynienia w przypadku układu CC2650 SensorTag firmy Texas Instruments. Dlatego producent przygotował wersję systemu operacyjnego czasu rzeczywistego TI-RTOS ściśle powiązanego z biblioteką użytkową oraz stosem komunikacyjnym.

W poprzednich odcinkach kursu został omówiony zestaw CC2650 SensorTag [1], jego użytkowanie [2] oraz moduły rozszerzeń Debug DevPack, Display DevPack (LCD screen) i LED Audio DevPack [3]. Moduł CC1310 LaunchPad został omówiony w kolejnym odcinku [4]. W tym odcinku kursu zostanie pokazany system operacyjny czasu rzeczywistego TI-RTOS dla serii układów scalonych CC13xx/CC26xx. Przedstawione praktyczne ćwiczenie dotyczy zestawu CC2650 SensorTag.

Zestaw CC2650 SensorTag jest dostarczany z fabrycznie zaprogramowanym programem o wdzięcznej nazwie DEMO. Program jest zbudowany z zastosowaniem systemu operacyjnego TI-RTOS oraz stosu Bluetooth LE ver.4.2 [1, 2, 3].

Dokumentacja

Linki do aktualnych wersji podstawowej dokumentacji można znaleźć na stronie TI-RTOS [7]. Opis rdzenia systemu jest zamieszczony w dokumencie *SYS/BIOS (TI-RTOS Kernel) User's Guide* [8]. Opis ogólny systemu jest zamieszczony w dokumencie *TI-RTOS 2.20 User's Guide* [9]. Opis wersji dla procesorów serii CC26xx jest zamieszczony w *TI-RTOS 2.20 for CC13xx/CC26xx SimpleLink Getting Started Guide* [7] oraz w *CC2640/CC2650 Bluetooth low energy Software Developer's Guide* [9].

Podstawowym źródłem dokumentacji zainstalowanej lokalnie w komputerze PC wersji systemu TI-RTOS jest plik *Documentation_Overview_cc13xx_cc26xx.html* zamieszczony w ścieżce *C:\ti\tirtos_cc13xx_cc26xx_2_20_01_08\docs*. Pliki opisu mają format pdf oraz htm. Jest też dokumentacja drajwerów peryferyjnych w formacie Doxygen. W pliku są również odnośniki do stron internetowych. Na portalu społecznościowym *TI E2E Community* znajduje się bardzo przydatna strona *CC2640/CC2650 Getting Started and FAQ* [11]. Jest ona często aktualizowana i zawiera odpowiedzi na najczęściej zadawane pytania.

Warsztaty *SimpleLink Academy*

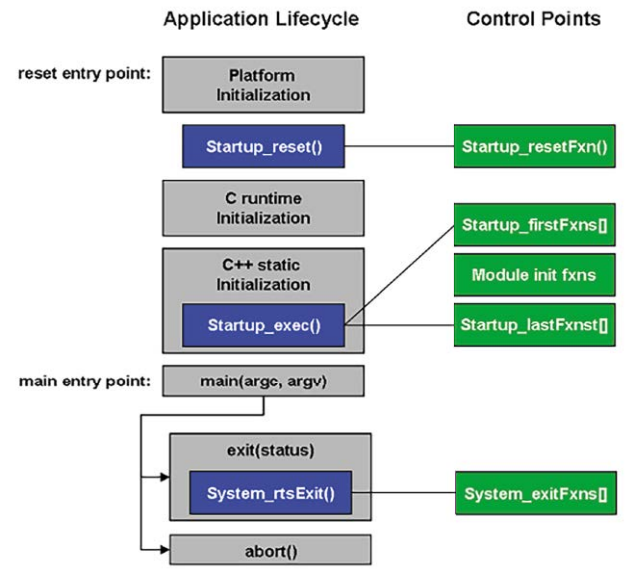
Bardzo ciekawą pomocą dla każdego, kto zaczyna pracować z procesorami rodziny CC13xx/CC26xx, są ćwiczenia warsztatowe *SimpleLink Academy* [15]. Dostępnych jest wiele ćwiczeń z dokładnym opisem oraz kodem źródłowym. Dla wielu ćwiczeń jest udostępniony zapis wideo.

System TI-RTOS

TI-RTOS (dawna nazwa DSP/BIOS) jest systemem operacyjnym czasu rzeczywistego ze skalowalnym jądrem (poprzednia nazwa SYS/BIOS) [5, 13]. System zawiera wiele dołączonych elementów w tym: stosy protokołów komunikacyjnych, obsługę komunikacji międzyrdzeniowej, sterownik urządzeń peryferyjnych i zarządzanie zasilaniem. System jest dostarczany z pełnymi źródłami i nie wymaga licencji.

Systemu TI-RTOS można używać z różnymi środowiskami programowymi. Dostarczany jest w wersjach dla różnych układów scalonych produkcji TI. System nie jest instalowany razem ze środowiskiem CCS. Należy go pobrać i zainstalować oddzielnie [5]. Dla procesorów serii CC26xx jest wersja TI-RTOS 2.20 for CC13xx/CC26xx [7]. Pakiet programowy CC26xxWare został zintegrowany z tą wersją systemu. Gdy jednocześnie jest stosowany stos Bluetooth LE, to należy pobrać i zainstalować pakiet SDK [6]. Zawiera on zarówno stos BLE, jak i odpowiednią wersję systemu TI-RTOS.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **39483**, pass: **5ikr2a2ku**



Rysunek 1. Sekwencja obsługi funkcji *main()* przez system TI-RTOS [16]

TI-RTOS jest wielowątkowym systemem operacyjnym czasu rzeczywistego z wywłaszczaniem. Ściśle z nim związane są narzędzia do synchronizacji i szeregowania (XDCTools) [9, 12].

Jądro systemu zarządza czterema oddzielnymi poziomami wątków:

- Przerwania sprzętowe (Hardware interrupt service routines (ISRs)) oraz Timer.
- Przerwania programowe (Software interrupt routines) oraz Clock.
- Zadania (Tasks).
- Funkcje tła (Background idle functions).

Zadanie systemu TI-RTOS może być w jednym z kilku stanów:

- Running: zadanie właśnie jest wykonywane.
- Ready: zadanie jest zaszeregowane do wykonania.
- Blocked: wykonywanie zadania jest zawieszone.
- Terminated: wykonywanie zadania jest zakończone.
- Inactive: zadanie jest na liście zadań nieaktywnych.

Tylko jedno zadanie może być wykonywane. Nawet jeśli jest to zadanie tła. Wykonywane zadanie może zostać zablokowane na semaforze. Może też samo zakończyć swoje działanie. W obu przypadkach procesor przełącza się na oczekujące zadanie gotowe o najwyższym priorytecie. Każde zadanie ma funkcję inicjalizacji oraz funkcję callback.

Sekwencja startowa systemu TI-RTOS

Sekwencja startowa jest logicznie podzielona na dwie fazy. Na **rysunku 1** jest pokazana sekwencja obsługi funkcji *main()* przez, system TI-RTOS [16]. Pokazane są specyficzne punkty, w których mogą być wywołane funkcje dostarczane przez użytkownika.

Funkcja *main()* w pliku *main.c* foldera roboczego projektu środowiska IDE (np. CCSv7) jest punktem startowym czasu wykonania. W tym punkcie inicjalizowane są sterowniki przy zablokowanych przerwaniach. Inicjalizowane jest też zarządzanie zasilaniem oraz są tworzone zadania. Na koniec włączane jest zezwolenie na obsługę przerwań, uruchamiane jest szeregowanie zadań przez jądro systemu TI-RTOS – wywoływana jest funkcja *BIOS_start()*, która nie wraca nigdy do funkcji *main()*.

Sekwencja startowa przed funkcją *main()*

Jest ona w całości sterowana przez pakiet XDCtools:

1. Bezpośrednio po zakończeniu stanu reset procesora wykonaj specyficzną dla tego procesora sekwencję inicjalizacji.
2. Wykonaj funkcję reset dostarczoną przez użytkownika.
3. Wykonaj cinit() w celu inicjalizacji środowiska języka C.
4. Wykonaj dostarczone przez użytkownika „first functions”.
5. Wykonaj funkcje inicjalizacyjne.
6. Wykonaj dostarczone przez użytkownika „last functions”.
7. Wykonaj pinit().
8. Wywołaj funkcję main().

Sekwencja po funkcji *main()*

Jest sterowana przez jądro systemu TI-RTOS. Rozpoczyna się po wywołaniu funkcji *BIOS_start()*.

1. Wykonaj dostarczone przez użytkownika „startup functions”. Jeśli obiekty Timers zostały utworzone statycznie, to teraz zostanie wykonana ich statyczna inicjalizacja. Jeśli zostały skonfigurowane do automatycznego wystartowania, to zostaną uruchomione.
2. Włączane jest zezwolenie na obsługę przerwań sprzętowych.
3. Włączane jest zezwolenie na obsługę przerwań programowych (SWI).
4. Szeregowane są zadania. Jeśli nie ma żadnego zadania do uruchomienia, to rozpoczynana jest funkcja tła.

Ćwiczenie *TI-RTOS Basic – Lab 1*

Ćwiczenie pokazuje praktycznie (krok po kroku) używanie podstawowych elementów systemu operacyjnego czasu rzeczywistego TI-RTOS. Jest ono wzorowane na ćwiczeniu TI-RTOS Basic w portalu *SimpleLink Academy* [15]. Zostały jednak wprowadzone liczne zmiany i rozszerzenia.

Wymagania sprzętowe

- Do pracy potrzebny jest zestaw CC2650 SensorTag z dołączonym modułem rozszerzeń Debug DevPack, połączony z komputerem kablem USB-A USB-Micro.
- Komputer PC z systemem operacyjnym Windows 7 (lub nowszym).

Ćwiczenie można też wykonywać z użyciem modułu uruchomieniowego CC2650 LanuchPad lub modułu CC2650EM Evaluation Module z płytą SmartRF06 Evaluation Board.

Wymagania programowe

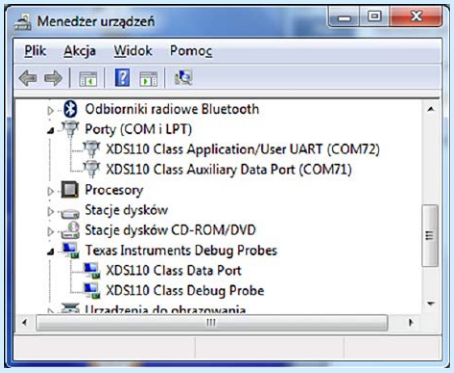
- Zainstalowany program CCS v7 (lub nowszy).
- Zainstalowany program BLE-STACK V2.2.1 (Support for CC2640/CC2650).
  Instalowany jest:
  - stos BLE w ścieżce *C:\ti\simplelink\ble_sdk_2_02_01_18*;
  - biblioteki systemu operacyjnego TI-RTOS w ścieżce *C:\ti\tirtos_cc13xx_cc26xx_2_20_01_08*;
  - biblioteki zewnętrzne tego systemu w ścieżce *C:\ti\xdctools_3_32_00_06_core*.
- Zainstalowane źródła pakietu *SimpleLink Academyv1.11* w ścieżce *C:\ti\simplelink_academy_01_11_00_0000*.

Istotna jest powyższa kolejność instalowania oprogramowania. Po zainstalowaniu każdego pakietu należy uruchomić CCSv7. Pozwala to na zbudowanie przez CCSv7 bazy projektów przykładowych dostarczanych przez pakiet.

Ładowanie projektu

1. Do zestawu CC2650 SensorTag dołącz moduł rozszerzeń Debug DevPack (dokładny opis w [2]). Połącz go z komputerem kablem USB-A USB-Micro. Na płytce modułu Debug DevPack zaczyna świecić dioda LED, co sygnalizuje, że moduł jest gotowy do pracy.
2. Otwórz Menadżer urządzeń i czekaj, aż zostaną zainstalowane wszystkie drajwery sprzętowe (**rysunek 2**)

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5kc7a2ku



Rysunek 2. Stan po zainstalowaniu drajwerów XDS110

3. Uruchom program CCSv7. Dwukrotnie kliknij na jego ikonę.
4. W oknie *Workspace Launcher* wpisz ścieżkę do folderu roboczego, np. <C:\home_dir**work_ART5**>. Kliknij *OK*.
5. Obserwuj informacje na pasku w prawym dolnym rogu. Dotyczą one ładowania modułów środowiska Eclips oraz sprawdzania dostępności aktualizacji. Najlepiej zaczekać na zakończenie tych prac.
6. Zamknij okno *Updates Avilable* (jeśli się pojawi) lub wykonaj aktualizację.
7. Może się też pojawić okno *Install Discoverd Products*. Należy wtedy kliknąć *Finish* i ponownie wystartować CCS. Jest to konieczny krok po to, aby CCS widział zainstalowane oprogramowanie.
8. Otwórz okno *Resource Explorer* z menu *View* → *Resource Explorer Classic*.
9. W zakładce *TI Resource Explorer* rozwiń listę SimpleLink Academy 1.11 → TI-RTOS → Projects → Lab 1.
10. Kliknij na linię ***CC2650 SensorTag***.

Po prawej stronie zostanie wyświetlona instrukcja, jak postępować w czterech krokach (**rysunek 3**).

Krok 1 – Importuj przykładowy projekt do CCS

11. Kliknij na odnośnik kroku 1. Po załadowaniu projektu zostanie pokazana w oknie *Project Explorer* linia projektu ***tirtos_lab1_cc2650stk*** *[Active – Debug]*. Otwierane są także w oknach edycji pliki *lab1-main.c* oraz *lab1-main-solution.c*.
12. Kliknij na zakładkę *TI Resource Explorer*. Zielony znaczek √ pojawi się na prawo od odnośnika kroku 1.

Jeśli znaczek się nie pojawi, należy kliknąć w zakładce *TI Resource Explorer* na linię *CC2650 SensorTag*.

Krok 2 – Zbuduj projekt *tirtos_lab1_cc2650stk*

13. W celu zbudowania projektu wybierz z menu *Project* → ***Clean***.
14. W oknie *Clean* kliknij na *OK*.

Pojawią się okna „Console” i „Problems”. Podczas budowania postęp wykonania jest pokazywany w oknie „Progress Information” oraz w oknie „Console”.

Na końcu w oknie „Console” zostanie pokazana informacja o pomyślnym wygenerowaniu pliku kodu ‘Finished building target: tirtos_lab1_cc2650stk.out’

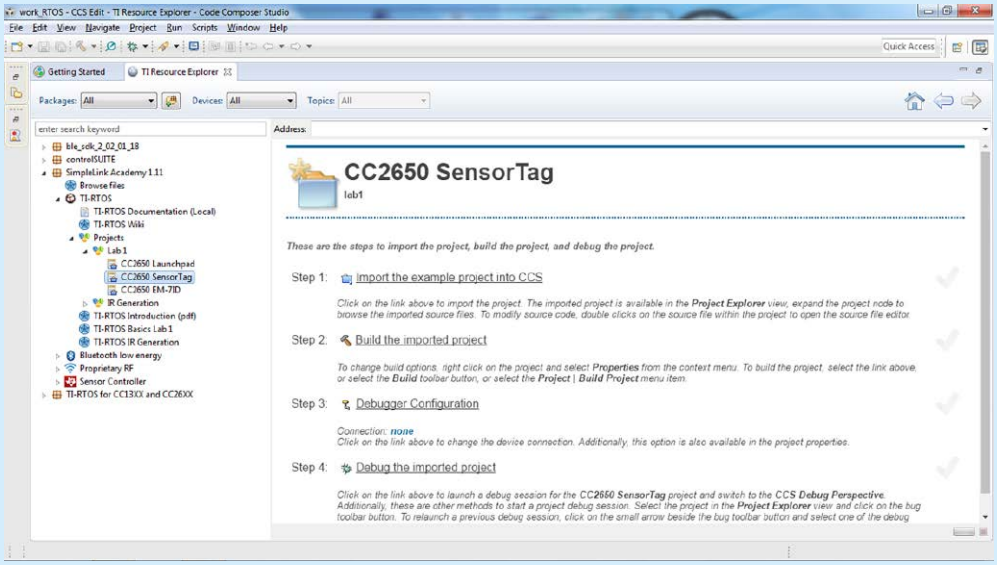
Zielony znaczek √ pojawi się na prawo od odnośnika kroku.

15. Jeśli znaczek się nie pojawił, to trzeba w zakładce *TI Resource Explorer* kliknąć na linię projektu *CC2650 SensorTag*.

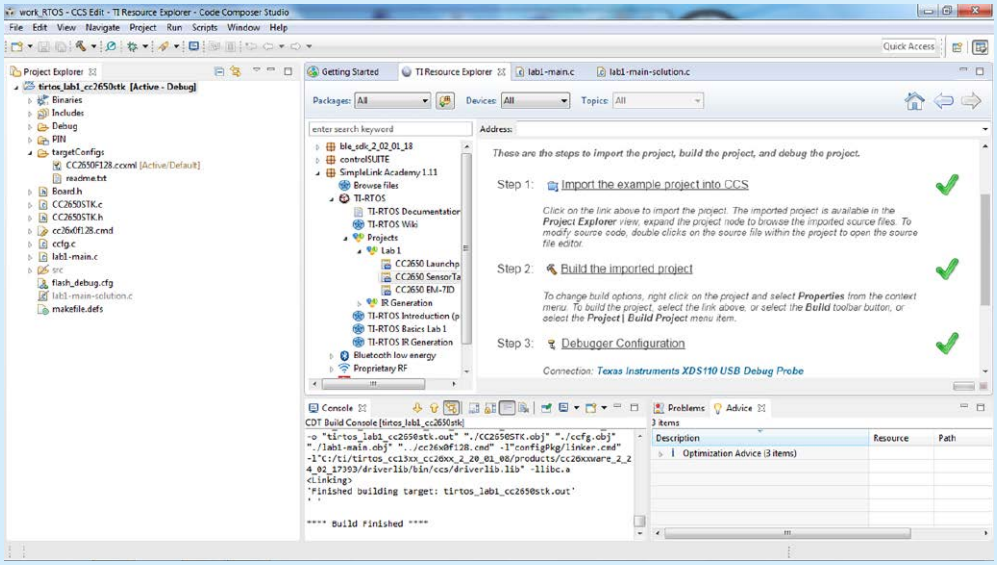
Uwaga! Polecenie Clean powoduje usunięcie wszystkich plików pośrednich projektu i zbudowanie całego projektu od początku. Pozwala to na uniknięcie kłopotów związanych z niepełną zgodnością wersji różnych bibliotek związanych z projektem.

Krok 3 – Wybierz typ emulatora sprzętowego

16. Może się okazać, że w projekcie już został wcześniej zdefiniowany typ emulatora. Wtedy zielony znaczek przy odnośniku kroku 3 jest ustawiony (**rysunek 4**), ale trzeba sprawdzić



Rysunek 3. Ładowanie projektu *TI-RTOS Basic - Lab 1*



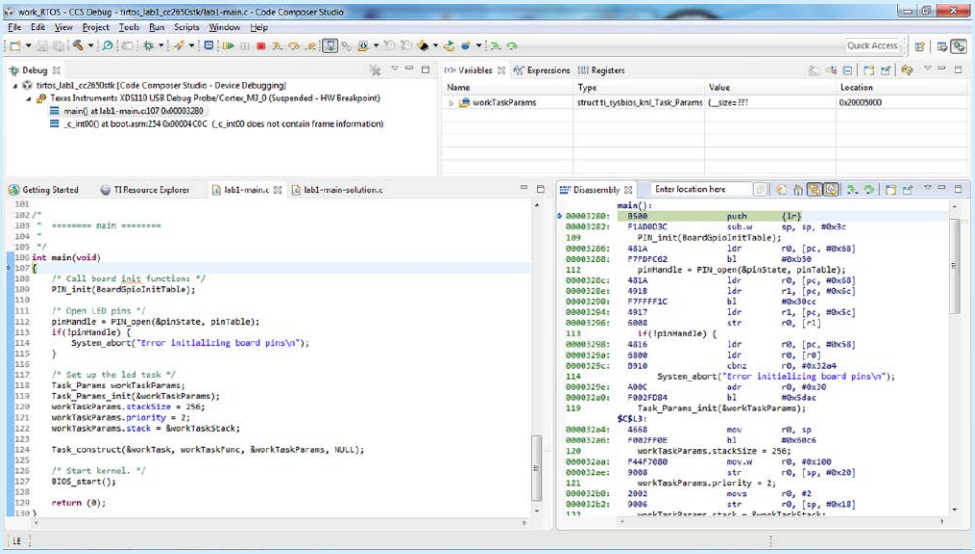
Rysunek 4. Stan projektu po wykonaniu budowania

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5kc7a2ku

poniżej odnośnika, jaki jest wybrany typ emulatora, i w razie niezgodności zmienić go na prawidłowy.

Jeśli typ emulatora nie został wcześniej zdefiniowany, to kliknij na odnośnik kroku 3. Z listy wybierz *„Texas Instruments XDS110 USB Debug Probe”*. Kliknij *OK*. Zielony znaczek √ pojawi się na prawo od odnośnika kroku.

17. W oknie *„Project Explorer”* rozwiń drzewo projektu ***tirtos_lab1_cc2650stk*** oraz pozycję ***targetConfigs*** node. Został zdefiniowany plik CC2650F128.ccxml oraz ustawiony jako [Active/Default].



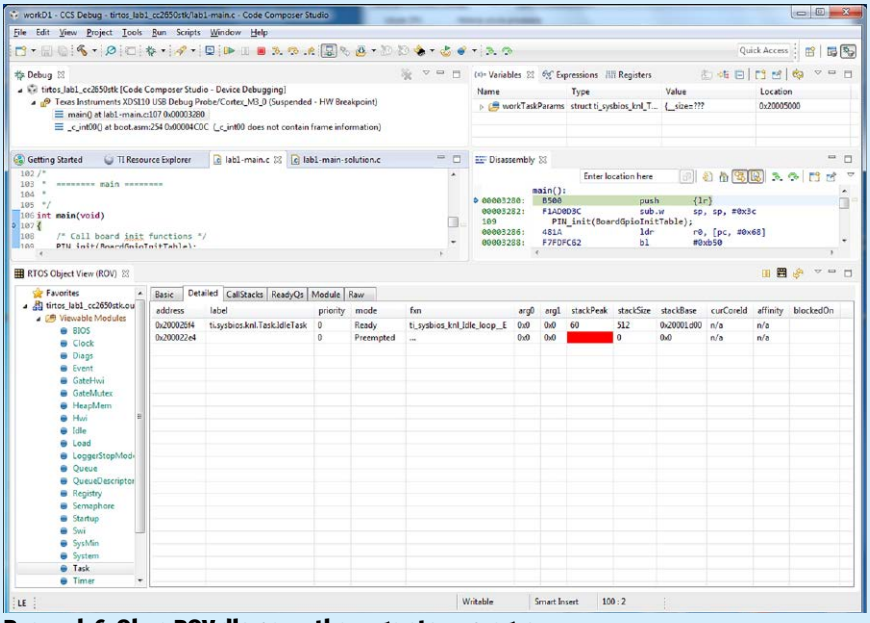
Rysunek 5. Stan projektu po wykonaniu polecenia debugowania

Krok 4 – Debuguj projekt tirtos_lab1_cc2650stk

18. Kliknij na odnośnik kroku 4.
19. Czekaj, aż kursor w oknie edycyjnym *lab1-main.c* zostanie umieszczony w pierwszej linii funkcji *main()*.

Zostało wykonanych kilka operacji:

- Otworzenie sesji debugowej oraz otworzenie perspektywy *CCS Debug*.
- Połączenie debuggera poprzez emulator XDS110 i port JTAG z układem scalonym CC2650.
- Zbudowanie narzędzi zewnętrznych XDCtools systemu TI-RTOS.
- Załadowanie (zaprogramowanie) kodu z pliku *.out do wewnętrznej pamięci Flash układu scalonego CC2650.
- Wyprowadzenie układu CC2650 ze stanu RESET i uruchomienie wykonania kodu załadowanego programu (preambuła języka C).
- Zatrzymanie wykonywania załadowanego programu na pułapce umieszczonej w pierwszej linii kodu (asemblerowego) tego programu.



Rysunek 6. Okno ROV dla początkowego stanu programu

20. Z menu *View* → *Disassembly* otwórz okno *Disassembly* (**rysunek 5**).

W oknie *Disassembly* można zobaczyć aktualną zawartość licznika rozkazów (PC). Niebieska strzałka pokazuje na pierwszą instrukcję asemblerową funkcji main(). Kod binarny z pamięci wewnętrznej układu CC2650 jest odczytywany, dekodowany (disasemlerowany) i instrukcje asemblerowe są pokazywane w oknie *Disassembly*. Jest to najlepszy (i chyba jedyny skuteczny) sposób na kontrolowanie stanu połączenia debuggera (CCS) z układem scalonym oraz stanu wykonywanego programu.

Wgląd w projekt tirtos_lab1_cc2650stk

21. Sprawdź okno edycyjne pliku ***lab1-main.c***.

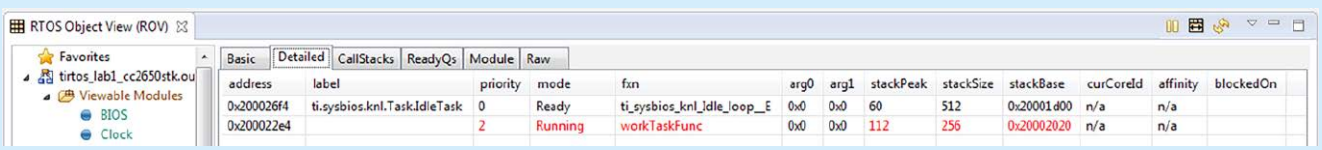
System TI-RTOS dostarcza wiele sterowników peryferyjnych dla procesorów CC26xx. Do sterowania wyprowadzeniami układu procesorowego używany jest sterownik PIN driver.

Najpierw trzeba dołączyć pliki nagłówkowe.

```
/* TI-RTOS Header files */
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
* Pin driver handles */
static PIN_Handle pinHandle;
/* Global memory storage for a PIN_Config table */
static PIN_State pinState;
```

Potem trzeba zdefiniować strukturę, gdzie definiowane są wyprowadzenia: kierunek pracy, poziom logiczny, podciąganie oraz moc sterowania.

```
/*
```



Rysunek 7. Okno ROV po zatrzymaniu pracy programu

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5kc7a2ku

```
 * Initial pin configuration table
 *  - LEDs Board_LED0 & Board_LED1 are off after the pin table is initialized.
 *  - Button is set to input with pull-up. On SmartRF06, BUTTON0 is UP, for the
 *    others it's the LEFT button.
 */
PIN_Config pinTable[] = {
    Board_LED0 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    Board_BUTTON0 | PIN_INPUT_EN | PIN_PULLUP,
    PIN_TERMINATE
};
```

W ramach funkcji *main()* należy najpierw inicjalizować wszystkie wyprowadzenia. A potem otworzyć wyprowadzenia używane w projekcie.

```
/* Call board init functions */
PIN_init(BoardGpioInitTable);
/* Open LED pins */
pinHandle = PIN_open(&pinState, pinTable);
if(!pinHandle) {
  System_abort(„Error initializing board pins\n");
  }
```

Uzyskany uchwyt zostanie użyty do zmiany stanu wyprowadzeń w czasie pracy programu.

```
PIN_setOutputValue(pinHandle, Board_LED1, 1);
```

Kolejnym działaniem w projekcie jest utworzenie i uruchomienie zadania (workTask) systemu TI-RTOS. Wymaga to zdefiniowania struktury dostępu oraz stosu tego zadania.

```
Task_Struct workTask;
static uint8_t workTaskStack[256];
```

Dalej definiowana jest funkcja callback

```
Void workTaskFunc(UArg arg0, UArg arg1)
{
   while (1) {
     /* Do work */
     doWork();
    /* Wait a while, because doWork should be a periodic thing, not continuous.*/
     CPUdelay(24e6);
   }
}
```

W funkcji *main()* ustawiane są parametry zadania oraz jest ono tworzone przy użyciu wywołania funkcji Task_construct.

```
/* Set up the led task */
Task_Params workTaskParams;
Task_Params_init(&workTaskParams);
workTaskParams.stackSize = 256;
workTaskParams.priority = 2;
workTaskParams.stack = &workTaskStack;
Task_construct(&workTask, workTaskFunc, &workTaskParams, NULL);
```

Definiowane jest tylko jedno zadanie system operacyjnego TI-RTOS z funkcją workTaskFunc jako ciało zadania. Jest to pętla nieskończona, która realizuje przełączanie stanu diody LED (w funkcji *doWork*) oraz opóźnienie czasowe.

```
void doWork(void)
{
  PIN_setOutputValue(pinHandle, Board_LED1, 1);
  FakeBlockingSlowWork();
  PIN_setOutputValue(pinHandle, Board_LED1, 0);
}
```

Wywołanie na końcu funkcji *main()* funkcji *BIOS_start* powoduje uruchomienie działania systemu operacyjnego TI-RTOS. Sterowanie nie wraca już do funkcji *main()*.

Ostatnią linią kodu w funkcji *main()* jest polecenie powrotu

```
return (0);
```

Jest ono dodane po to, aby w przypadku niepowodzenia wywołania funkcji *BIOS_start* program nie „poszedł w maliny". Przy dosyć skomplikowanej strukturze projektu taki scenariusz jest możliwy.

Okno RTOS Object View

Okno RTOS Object View (lub w skrócie ROV) pozwala na wgląd w stan obiektów systemu TI-RTOS [17]. Informacja jest aktualizowana poprzez łącze JTAG po zatrzymaniu pracy procesora.

22. Otwórz ROV z menu *Tools* → *RTOS Object View (ROV)*.

Otwierana jest tabelka z informacją o obiektach systemu TI-RTOS.

23. Zamknij okno *Console*. W zakładce *RTOS Object View (ROV)* rozwiń drzewo *tirtos_lab1_cc2650stk.out*. Wybierz pozycję *Task* i otwórz zakładkę *Detailed* (**rysunek 6**).

W oknie można zobaczyć, które zadanie jest aktualnie wykonywane (Running), które jest zablokowane (Blocked), a które jest gotowe (Ready).

Obecnie tylko dla zadania tła (IdleTask) możemy sprawdzić wielkość stosu (512) i jego maksymalny rozmiar (60).

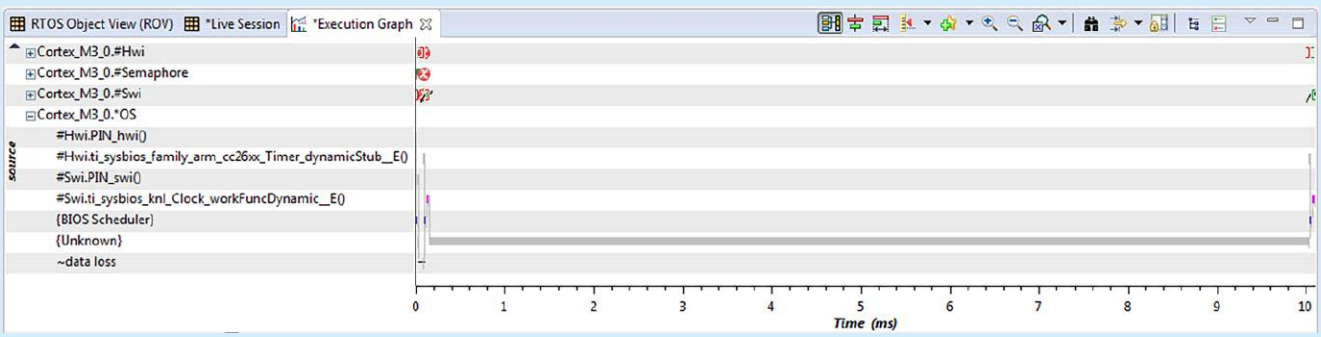
Uruchom program tirtos_lab1_cc2650stk

24. Na pasku narzędzi perspektywy *CCS Debug* kliknij na przycisk *Resume*.

Program zaczyna pracować. Dioda LED1 (czerwona) zestawu CC2650STK SensorTag zapala się na 1 sekundę z 1- jednosekundową przerwą.

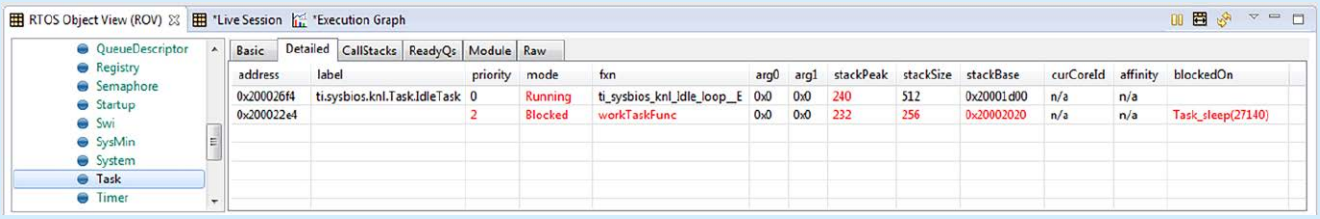
25. Po pięciokrotnym zaświeceniu diody LED kliknij na przycisk *Suspend* (Halt/Pause). Spowoduje to zatrzymanie działania programu.
26. Zobacz stan okna *RTOS Object View (ROV)* – **rysunek 7**.

W oknie można zobaczyć, że zadanie *workTaskFunc* jest aktualnie wykonywane (Running), maksymalny rozmiar stosu wynosi 112 przy wielkości stosu (256). Czyli nie ma obaw o wystąpienie przepełnienia stosu.



Rysunek 8. Okno *Execution Graph* po zatrzymaniu pracy programu

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5kc7a2ku



Rysunek 9. Okno ROV po zatrzymaniu pracy programu w trakcie uśpienia

Execution graph

Okno Execution graph pokazuje informacje o stanie systemu TI-RTOS podczas pracy [14].

Lista wątków jest pokazana w lewej kolumnie. Uwidoczniona jest aktywność wątków zdefiniowanych przez użytkownika (HWI, SWI, Semaphore), jego zadań (Task) oraz wątków systemu operacyjnego. Kolorowa linia pokazuje, kiedy wątek jest aktywny. Dokładniejszy opis jest zamieszczony w dokumencie [14].

27. Z menu wybierz *Tools* → *RTOS Analyzer* → *Execution Analysis*.
28. W oknie *Analysis Configuration* zaznacz tylko Execution Graph, pozostaw resztę ustawień bez zmian i naciśnij przycisk *Start*.
29. W nowym oknie zakładki *Execution Graph* rozwiń widoczność wątków *Cortex_M3_0.*OS* (**rysunek 8**).

Jest tylko jedno aktywne zadanie *workTaskFunc*. Jak można zobaczyć w zakładce *Live Session*, zostało ono zaliczone do klasy zdarzeń o nazwie Unknown. Zadanie tła nie jest nigdy uaktywniane podczas pracy programu. Oznacza to, że jedno zadanie zawłaszcza cały czas procesora. Brak wyświetlania stanu zadania *workTaskFunc* jest sytuacją raczej błędną. Poprzednia wersja środowiska i systemu TI-RTOS go pokazywała.

Śpij dobrze

System TI-RTOS umożliwia zawieszenie działania zadania na określony czas. Służy do tego funkcja *Task_sleep(numTicks [tick])*. Blokuje ona działanie zadania na określoną liczbę okresów zegara RTOS (tick).

Aby uzyskać taki efekt, zostanie zamieniona funkcja *CPUdelay* na funkcję *Task_sleep* z czasem 1000 ms (1e6 mikrosekund).

30. Przejdź do perspektywy *CCS Edit*.
31. Otwórz plik *lab1-main-solution.c*. Dwukliknij na linię z jego nazwą.
32. Skopiuj linie 109-110.

```
/* Sleep */
Task_sleep(1000 * (1000 / Clock_tickPeriod));
```

33. Otwórz plik *lab1.c*.
34. Znajdź linie 97-98

```
/* Wait a while, because doWork should be a periodic thing, not continuous.*/
 CPUdelay(24e6);
```

35. Zamień linie 97–98 na nową skopiowaną zawartość.
36. Skopiuj linię 45 z pliku *lab1-main-solution.c*.

```
#include <ti/sysbios/knl/Clock.h>
```

37. Wstaw ją do pliku *lab1.c* po linii 44 jako nową linię 45.

```
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
```

Znak gwiazdki przed nawą pliku na pasku tytułowym zakładki okna edycyjnego pliku *lab1.c* sygnalizuje, że w pliku zostały wykonane zmiany i aktualna wersja nie jest zapisana do pliku. Na początku wykonywania polecenia budowania projektu wszystkie zmodyfikowane okna edycyjne są zapisywane do plików.

38. Wykonaj budowanie projektu. Użyj przycisku *Build* . Nie używaj przycisku *Debug* .
39. W oknie *Reload…* kliknij *Yes*, aby załadować (zaprogramować) nowy kod wynikowy do procesora.
40. Czekaj, aż kursor w oknie edycji *lab1-main.c* zostanie umieszczony w pierwszej linii funkcji *main()*.
41. Przejdź do perspektywy *CCS Debug*.
42. Na pasku narzędzi perspektywy *CCS Debug* kliknij na przycisk *Resume* .
43. Po pięciokrotnym zaświeceniu diody LED kliknij na przycisk *Suspend* (Halt/Pause). Spowoduje to zatrzymanie działania programu.
44. Zobacz stan okna *RTOS Object View (ROV)* (**rysunek 9**).

Zadanie *workTaskFunc* jest zablokowane przez polecenie *Task_sleep*.

45. Zobacz wykres w oknie zakładki *Execution Graph* (**rysunek 10**).

Aby zmienić skalę czasową wykresu, kliknij na pole opisu (Time) i pokręć kółkiem myszki.

W oknie wyraźnie widać naprzemienną pracę zadania *workTaskFunc* oraz systemowego zadania tła.

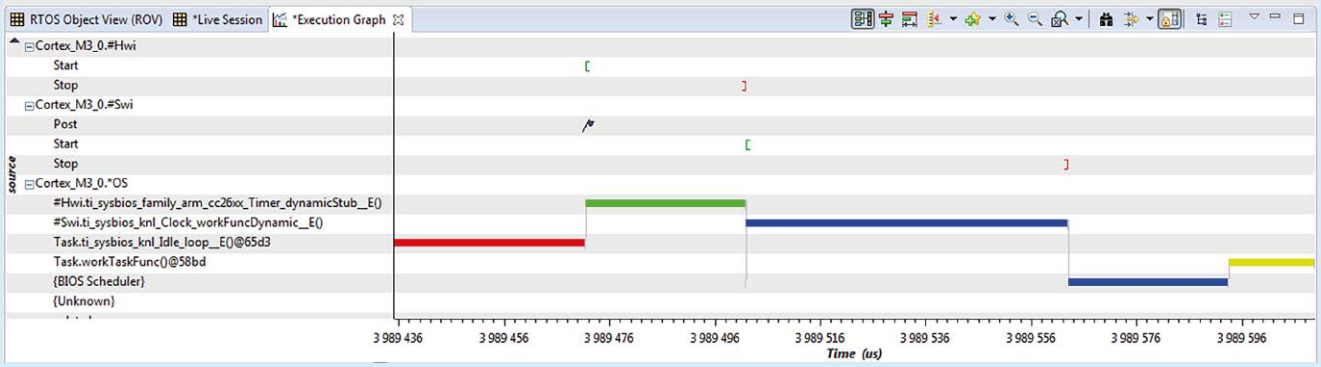
Symbole stosowane na wykresie okna *Execution Graph*

Przerwania sprzętowe. Nawiasy [oraz] oznaczają początek i koniec pracy wątku. Jeśli wystąpi utrata danych, wtedy pokazywana jest ikonka .

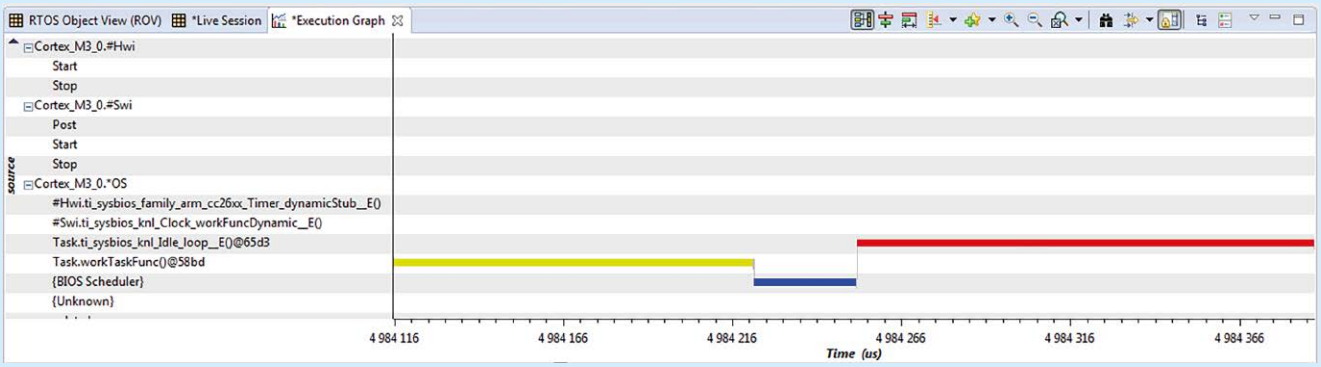
Przerwania programowe. Ikonka informuje, że wątek został aktywowany (post). Nawiasy [oraz] oznaczają początek i koniec pracy wątku.

Rysunek 10. Okno *Execution Graph* po zatrzymaniu pracy programu w trakcie uśpienia

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5kc7a2ku



Rysunek 11. Początek pracy zadania *workTaskFunc*



Rysunek 12. Koniec pracy zadania *workTaskFunc*

46. W oknie zakładki *Execution Graph* rozciągnij wykres w okolicach przełączania z systemowego zadania tła do zadania *workTaskFunc* (**rysunek 11**).

Zakończenie zliczania czasu uśpienia powoduje zgłoszenie przerwania sprzętowego. Wątek przerwania sprzętowego aktywuje funkcję zegara. Po jego zakończeniu startuje praca wątka przerwania programowego (zegarowego) i następnie sterowanie jest przekazywane do programu szeregującego zadania (scheduler). Zadanie *workTaskFunc* jest gotowe i zostaje uruchomione.

47. W oknie zakładki *Execution Graph* rozciągnij wykres w okolicach przełączania z systemowego zadania *workTaskFunc* do zadania tła (**rysunek 12**).

Zadanie *workTaskFunc* po wywołaniu funkcji *Task_sleep* zostaje zablokowane i program szeregujący zadania uruchamia zadanie tła.

W następnym odcinku spróbujemy jeszcze bardziej poprawić pracę programu.

Henryk A. Kowalski
kowalski@ii.pw.edu.pl

Literatura:

[1] Systemy dla Internetu Rzeczy (1): Zestaw CC2650 SensorTag, „Elektronika Praktyczna", 12/2016
[2] Systemy dla Internetu Rzeczy (2): Użytkowanie zestawu CC2650 SensorTag, „Elektronika Praktyczna", 1/2017
[3] Systemy dla Internetu Rzeczy (3): Użytkowanie zestawu CC2650 SensorTag, „Elektronika Praktyczna", 2/2017
[4] Systemy dla Internetu Rzeczy (4): Zestaw CC1310 LaunchPad, „Elektronika Praktyczna", 3/2017
[5] TI-RTOS: Real-Time Operating System (RTOS) for Microcontrollers (MCU) ***https://goo.gl/1nU5Ua***
[6] BLE-STACK V2.2.1 (Support for CC2640/CC2650) v2.2.1, 28-OCT-2016 ***https://goo.gl/84HQNI***
[7] TI-RTOS 2.20 for CC13xx/CC26xx SimpleLink Getting Started Guide (SPRUHU7D.pdf), 17 Jun 2016 ***https://goo.gl/5VksQk***
[8] SYS/BIOS (TI-RTOS Kernel) v6.46 User's Guide (SPRUEX3Q.pdf), 16 Jun 2016 ***https://goo.gl/3cbzBR***
[9] CC2640/CC2650 Bluetooth low energy Software Developer's Guide (SWRU393D.pdf) 15 October 2016 ***https://goo.gl/J4499t***
[10] TI-RTOS 2.20 User's Guide (SPRUHD4M.pdf), 17 Jun 2016 ***https://goo.gl/L8tOFo***
[11] CC2640/CC2650 Getting Started and FAQ, 2016 Oct 31 ***https://goo.gl/lBTXOI***
[12] TI-RTOS (TI WIKI) ***https://goo.gl/Luh1oW***
[13] Category:SYSBIOS (TI WIKI) ***https://goo.gl/L08fy9***
[14] System Analyzer User's Guide (SPRUH43.pdf) (ver.F March 2014) ***https://goo.gl/dCXuHg***
[15] SimpleLink Academy (v1.11 – November 4th 2016) ***https://goo.gl/ez4FjV***
[16] How applications boot, run, and shutdown, The Eclipse Foundation, 2008 ***https://goo.gl/lDR4FF***
[17] Runtime Object Viewer, Using ROV for Eclipse-Based Debugging ***https://goo.gl/Cg69ek***

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5ikr2a2ku

www.ep.com.pl/kap