Segger Embedded Studio dla STM32 (1)

Opis środowiska, biblioteki, przykłady programów, ekosystem

Wybór środowiska programistycznego jest niezwykle istotną decyzją, którą musi podjąć programista za każdym razem, gdy rozpoczyna nowy projekt. Decyzja ta obejmuje IDE, w którym będzie tworzony kod oraz biblioteki umożliwiające przetwarzanie danych, wykonanie interfejsu użytkownika, a także zawierające stosy komunikacyjne i systemy operacyjne czasu rzeczywistego. W artykule opisano rozwiązania dostarczane przez firmę Segger w postaci środowiska Embedded Studio oraz możliwych do użycia bibliotek. Jako przykład posłuży projekt oscyloskopu przygotowany dla zestawu STM32F746G-DISCO i publikowany na łamach EP 5/2016, 6/2016 i 7/2016, przeniesiony ze środowiska SW4STM32 i systemu FreeRTOS. Segger Embedded Studio i opisywane komponenty są dostępne bezpłatnie do zastosowań niekomercyjnych.

Pracę z narzędziami firmy Segger warto zacząć od pobrania pakietu bibliotek wraz z przykładami skonfigurowanymi dla jednego z dostępnych zestawów ewaluacyjnych. Listę wspieranych modułów można znaleźć na stronie ***http://bit.ly/2VOk2zw***. Znajduje się tam między innymi pakiet dla STM32F746G-Discovery od ST, wraz z bibliotekami wymienionymi w **tabeli 1**.

Komponenty dostępne w pakiecie są w postaci skompilowanych bibliotek i mogą być używane bezpłatnie w projektach niekomercyjnych, natomiast kody źródłowe są udostępniane w wersjach płatnych charakteryzujących się także rozszerzoną funkcjonalnością. Szczegóły dotyczące cen oraz dokładnej funkcjonalności poszczególnych bibliotek dostępne są na stronie producenta – ***www.segger.com***.

Segger Embedded Studio

Kolejnym narzędziem udostępnionym przez firmę Segger jest środowisko Embedded Studio przeznaczone dla systemów Windows, Linux i macOS. Umożliwia ono zarządzanie projektem, kompilowanie oraz debugowanie kodu, a przy tym jest dostępne bezpłatnie do zastosowań niekomercyjnych. Pracę można rozpocząć od jednego z przykładowych projektów, dostępnych w pakiecie ewaluacyjnym lub tworząc nowy projekt i dołączając do niego niezbędne komponenty. Zajmiemy się drugą z tych metod, przedstawiając krok po kroku poszczególne etapy tworzenia projektu i budowania aplikacji dla zestawu STM32F746G-Discovery.

Po uruchomieniu Segger Embedded Studio tworzymy nowy projekt (*File* → *New Project*) i wybieramy opcję *A C/C++ executable*

Poprzednie części kursu i dodatkowe materiały dostępne są na na stronie www.media.avt.pl

Tabela 1. Komponenty dostępne w pakiecie ewaluacyjnym dla STM32F746G-Discovery	
embOS	System operacyjny czasu rzeczywistego
embOS/IP	Stos IP zawierający między innymi DHCP, FTP, CoAP, SMTP
emCompress	Algorytmy kompresji danych
emCrypt	Algorytmy kryptograficzne, m.in. AES, DES, 3DES, MD5, SHA
emFile	System plików z szyfrowaniem dla pamięci NAND, NOR oraz kard SD
emModbus	Stos Modbus
emSecure	Tworzenie i weryfikacja podpisów cyfrowych
emUSB Device	Stos USB-Device
emUSB Host	Stos USB-Host
emWeb	Serwer sieciowy z obsługą, m.in. WebSocket, REST, SSE
emWin	Graficzne interfejsy użytkownika z obsługą paneli dotykowych

for a Cortex-M processor. W kolejnym oknie konfiguratora wybieramy mikrokontroler STM32F746NG, a jako interfejs do debugowania – J-Link SWD. Następne okno pozwala na dołączenie do projektu plików startowych oraz obsługę RTT (*Real-Time Transfer*), służącego do komunikacji z aplikacją za pośrednictwem debuggera. Na koniec można wybrać jeszcze dostępne konfiguracje projektu – Debug oraz Release i zakończyć pracę kreatora.

Jeżeli podczas konfiguracji do projektu zostały dodane wszystkie pliki, to możemy teraz zbudować i uruchomić przykładową aplikację wypisującą w terminalu debuggera napis *Hello World*. Kompilację możemy uruchomić, wybierając opcję z menu *Build* lub wciskając klawisz F7. Po jej zakończeniu otrzymujemy informację o wykorzystaniu pamięci Flash i RAM przez utworzoną aplikację.

Przed uruchomieniem aplikacji na płytce STM32F746G-DISCO musimy się na chwilę zatrzymać i przygotować debugger, ponieważ środowisko Segger Embedded Studio współpracuje jedynie z interfejsami J-Link. Istnieje jednak możliwość zmiany firmware'u na wbudowany w zestaw STM32F746G-DISCO interfejsie ST-Link, tak aby współpracował z naszym środowiskiem. Szczegółowe instrukcje dotyczące zmiany znajdują się na stronie firmy Segger pod adresem ***http://bit.ly/2RpQb26***. Zmiana firmware'u jest odwracalna. Niestety, procedura zmiany oprogramowania interfejsu ST-Link może zostać przeprowadzona tylko za pomocą systemu Windows. Po przeprogramowaniu debuggera możemy uruchomić skompilowaną wcześniej aplikację na zestawie STM32F746G-DISCO za pomocą opcji *Debug → Go* lub klawisza F5. Podczas debugowania mamy dostęp m.in. do: pracy krokowej, podglądu rejestrów, stosu wywołań i wartości zmiennych.

System operacyjny embOS

embOS jest systemem operacyjnym czasu rzeczywistego przeznaczonym do użycia w systemach wbudowanych. System jest

Tabela 2. Zużycie pamięci w bajtach przez komponenty systemu

Komponent	Zużycie pamięci
Zadanie (TCB, Task Control Block)	36
Licznik (Software timer)	20
Muteks (Mutex, Resource semaphore)	16
Semafor (Counting semaphore)	8
Skrzynka pocztowa (Mailbox)	24
Kolejka (Queue)	32
Zdarzenie (Event)	12

wielozadaniowy, a zadania są realizowane zgodnie z przydzielonym priorytetem. Zaimplementowano w nim opcjonalne wywłaszczanie oraz algorytm round-robin dla zadań o jednakowym priorytecie. Dla celów komunikacji i synchronizacji wbudowano weń takie mechanizmy, jak: semafory, kolejki, zdarzenia, skrzynki pocztowe. Wymagania dotyczące potrzebnej pamięci zależą od liczby użytych komponentów systemowych. Minimalne wymagania systemu to około 1700 bajtów pamięci Flash i 70 bajtów pamięci RAM. Dodatkowe wymagania pamięci RAM dla poszczególnych komponentów wymieniono w **tabeli 2**.

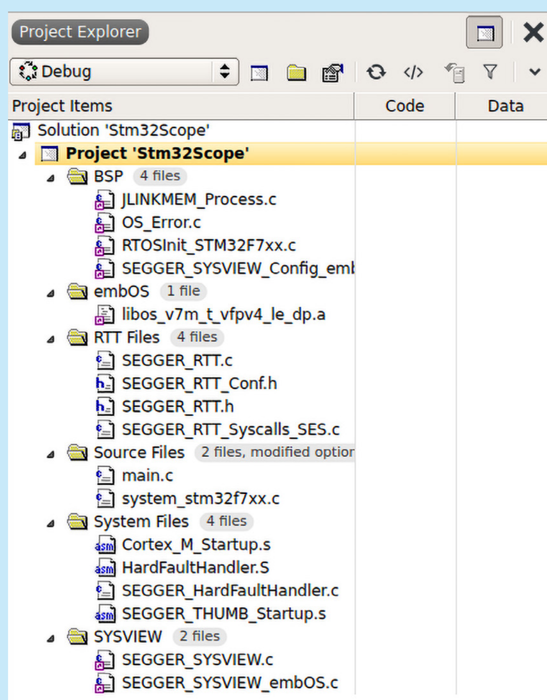
Skompilowane źródła systemu embOS oraz niezbędne nagłówki znajdują się w pakiecie ewaluacyjnym dla STM32F746G-DISCO, w katalogu OS. Katalog ten możemy skopiować do utworzonego projektu i dodać bibliotekę do kompilacji. W tym celu musimy kliknąć prawym przyciskiem myszy na nazwę projektu i wybrać opcję *Add Existing File…*

W wersji ewaluacyjnej mamy dostęp jedynie do trzech skompilowanych wersji embOS:

1. **libos_v7m_t_vfpv4_le_sp.a** – wersja release, ze sprawdzaniem stosu i profilowaniem,
2. **libos_v7m_t_vfpv4_le_r.a** – wersja release,
3. **libos_v7m_t_vfpv4_le_dp.a** – wersja debug z profilowaniem.

Do przykładu została wybrana wersja **libos_v7m_t_vfpv4_le_dp.a**, skompilowana z flagą kompilatora DEBUG=1. Wybrana biblioteka musi być także zaznaczona w pliku *OS_Config.h*, poprzez zdefiniowanie odpowiednio *OS_LIBMODE_SP*, *OS_LIBMODE_R* lub *OS_LIBMODE_DP*. Oprócz samej biblioteki do projektu trzeba także dodać ścieżkę do plików nagłówkowych „../OS/Inc”, klikając prawym przyciskiem myszy na nazwę projektu w oknie Project Explorer, wybierając opcję *Options…* i modyfikując pole *Preprocessor → User Include Directories*. Oprócz biblioteki embOS i jej nagłówków musimy dodać do projektu kilka innych plików znajdujących się w pakiecie ewaluacyjnym Seggera i w bibliotece STM32Cube. Potrzebne pliki i katalogi zostały umieszczone w **tabeli 3**.

W tym miejscu warto wspomnieć o zarządzaniu plikami w środowisku Embedded Studio. Wszystkie pliki dodane do projektu są wyświetlane po lewej stronie, w oknie nazwanym Project Explorer. Dodatkowo ścieżki plików w projekcie nie muszą odpowiadać lokalizacji plików na dysku, dzięki czemu możemy dowolnie grupować



Rysunek 1. Przykładowe pliki dodane do projektu

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

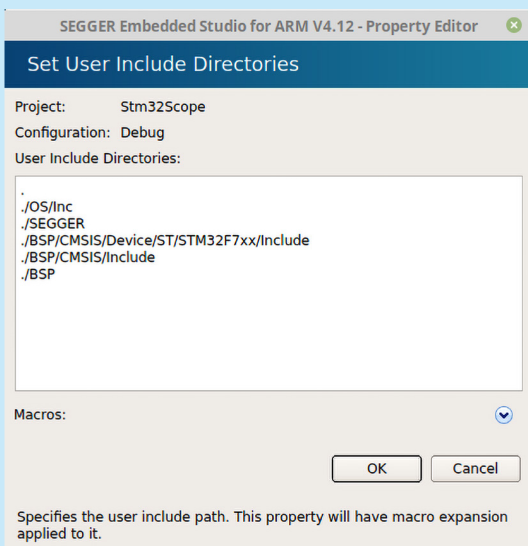
źródła znajdujące się bezpośrednio w katalogu projektu jak i poza nim. W prezentowanym przykładzie wszystkie niezbędne pliki zostały skopiowane do katalogu projektu, aby po rozpakowaniu można go było zbudować bez dodatkowych zależności.

Podobnie jak w wypadku biblioteki embOS, wszystkie pliki źródłowe z tabeli 3, musimy dodać do projektu, korzystając z opcji *Add Existing File*, natomiast ścieżki do lokalizacji plików nagłówkowych powinny znaleźć się w ustawieniach projektu, w polu *User Include Directories*. Przykładową konfigurację pokazano na **rysunkach 1** i **2**. Po dodaniu wszystkich ścieżek i źródeł musimy jeszcze zdefiniować symbol STM32F746xx dla biblioteki CMSIS. Możemy to zrobić w ustawieniach projektu w polu *Preprocessor → P reprocessor Definitions*.

W tym momencie mamy już wszystko potrzebne do uruchomienia prostego przykładu, którego kod źródłowy został przedstawiony na **listingu 1**. Konfiguracja systemu w funkcji main wykorzystuje funkcje zdefiniowane w dodanych do projektu plikach z pakietu Evaluation Software, a następnie tworzy pojedyncze zadanie. Na koniec jest wywoływana funkcja *OS_Start*, która uruchamia planistę systemowego odpowiedzialnego za uruchamianie zadań. Jedyne zadanie w przykładzie wypisuje w pętli „Hello Test Task!" na terminal debuggera. Funkcja OS_Delay realizuje opóźnienie zależnie od konfiguracji zegara systemowego, którego okres w przykładzie wynosi 1 ms.

Warto zwrócić uwagę na sposób tworzenia zadań za pomocą makra OS_CREATETASK. Przyjmuje ono wskaźnik do struktury OS_TASK, nazwę zadania, wskaźnik na funkcję implementującą zadanie, priorytet oraz wskaźnik na stos. Istotne jest, aby ten ostatni był dostatecznie duży, ponieważ to z niego będzie korzystało zadanie podczas wykonywania. Możemy teraz skompilować przykład i uruchomić go na płytce STM32F746G-DISCO.

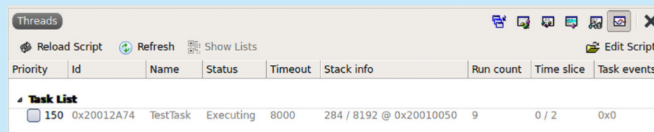
Na koniec tego rozdziału zobaczymy jeszcze, w jaki sposób można zdobyć informację o zdefiniowanych zadaniach i wykorzystywanej przez nie pamięci. Do tego celu potrzebujemy skryptu *embOSPluginSES.js*, znajdującego się w katalogu BSP/ST/STM32F746_STM32F746G_Discovery/Setup w pakiecie ewaluacyjnym. Ścieżkę do niego musimy podać w opcjach projektu, w polu *Debugger → Threads Script File* (w przykładzie jest to *$(ProjectDir)/embOSPluginSES.js*). Teraz, podczas debugowania kodu, za każdym razem, kiedy zatrzymamy program, w oknie Threads (dostępnym w menu



Rysunek 2. Konfiguracja ścieżek dla plików nagłówkowych

Tabela 3. Pliki potrzebne do uruchomienia przykładu dla systemu embOS

Segger Evaluation Software
Segger/Global.h
Segger/Segger.h
Segger/Segger_SYSVIEW.c
Segger/Segger_SYSVIEW.h
Segger/Segger_SYSVIEW_ConfDefaults.h
Segger/Segger_SYSVIEW_embOS.c
Segger/Segger_SYSVIEW_embOS.h
Segger/Segger_SYSVIEW_Int.h
BSP/ST/STM32F746_STM32F746G_Discovery/Setup/JLINKMEM_Process.cq
BSP/ST/STM32F746_STM32F746G_Discovery/Setup/OS_Error.c
BSP/ST/STM32F746_STM32F746G_Discovery/Setup/RTOSInit_STM32F7xx.c
BSP/ST/STM32F746_STM32F746G_Discovery/Setup/Segger_SYSVIEW_Conf.h
BSP/ST/STM32F746_STM32F746G_Discovery/Setup/Segger_SYSVIEW_Config_embOS.c
STM32Cube_FW_F7
Drivers/CMSIS/Device/ST/STM32F7xx/Include
Drivers/CMSIS/Include
Projects/STM32746G-Discovery/Templates/Src/system_stm32f7xx.c



Rysunek 3. Informacja o zadaniach w systemie embOS

View → More Debug Windows → Threads lub po wciśnięciu klawiszy Ctrl+Alt+H) zobaczymy listę zadań wraz z takimi informacjami jak ich stan oraz zużycie pamięci, co zostało pokazane na **rysunku 3**.

Migracja z FreeRTOS na embOS

Mimo różnic w implementacji, przenoszenie aplikacji pomiędzy systemami FreeRTOS i embOS jest stosunkowo proste, ponieważ zasada obsługi zadań, jak i komunikacji pomiędzy nimi pozostaje taka sama. Odpowiadające sobie typy danych i wywołania obu systemów operacyjnych użyte w przykładzie zostały przedstawione w **tabeli 4**. W tabeli pominięto wywołania związane z obsługą kolejek i liczników programowych, ponieważ nie zostały one użyte w kodzie programu po migracji. Tabela pomija również kwestie inicjalizacji sprzętu – w systemie embOS jest za to odpowiedzialna funkcja *OS_InitHW()*, zdefiniowana w pakiecie ewaluacyjnym i użyta w kodzie na listingu 1. Konfiguruje ona m.in. kontroler przerwań NVIC, zegar systemowy oraz pamięci podręczne. W przypadku systemu FreeRTOS inicjalizacja części peryferii oraz biblioteki HAL odbywa się zaraz na początku funkcji main, niezależnie od systemu, natomiast SysTick jest konfigurowany w ramach kodu zależnego od platformy i znajdującego się w pliku port.c.

Komentarza wymaga kwestia zdarzeń i notyfikacji. Oba mechanizmy działają analogicznie i polegają na przesłaniu do zadania wartości 32-bitowej z innego zadania lub przerwania. Przekazywana wartość jest najczęściej maską bitową, na której zakodowane są zdarzenia, dzięki czemu możliwe jest jednoczesne odebranie wielu zdarzeń. Oba systemy umożliwiają oczekiwanie na zdarzenie (lub notyfikację) przez określoną ilość czasu. Warto także zwrócić uwagę na to, że system FreeRTOS ma funkcje przeznaczone do wywoływania wyłącznie w zadaniach lub w przerwaniach. W embOS dostępna jest jedna funkcja, którą można wywołać niezależnie od kontekstu.

W przypadku tworzenia zadań można zauważyć dwie zasadnicze różnice. System FreeRTOS sam zarządza przydzieloną mu pulą pamięci, która jest wykorzystywana m.in. na stosy tworzonych zadań.

Poprzednie części kursu i dodatkowe materiały dostępne są na na stronie www.media.avt.pl

Tabela 4. Porównanie interfejsu w systemach FreeRTOS i embOS	
embOS	**FreeRTOS**
Inicjalizacja systemu i uruchomienie planisty.	
`void OS_InitKern(void);` `void OS_Start(void);`	`void vTaskStartScheduler(void);`
Tworzenie zadania, typ uchwytu i prototyp funkcji implementującej zadanie.	

`void OS_CREATETASK(`
`  OS_TASK* pTask,`
`  char* pName,`
`  void* pRoutine,`
`  OS_PRIO Priority,`
`  void* pStack);`
	`BaseType_t xTaskCreate(`
	`  TaskFunction_t pvTaskCode,`
	`  const char * const pcName,`
	`  unsigned short usStackDepth,`
	`  void *pvParameters,`
	`  UBaseType_t uxPriority,`
	`  TaskHandle_t *pxCreatedTask);`

`OS_TASK`	`TaskHandle_t`
`void task(void);`	`void vTask(void const *arg);`
Obsługa zdarzeń/notyfikacji	

`void OS_SignalEvent(`
`  OS_TASK_EVENT Event,`
`  OS_TASK* pTask);`
	`BaseType_t xTaskNotifyGive(`
	`  TaskHandle_t xTaskToNotify);`
	
	`BaseType_t xTaskNotify(`
	`  TaskHandle_t xTaskToNotify,`
	`  uint32_t ulValue,`
	`  eNotifyAction eAction );`
	
	`BaseType_t xTaskNotifyFromISR(`
	`  TaskHandle_t xTaskToNotify,`
	`  uint32_t ulValue,`
	`  eNotifyAction eAction,`
	`  BaseType_t *pxHigherPriorityTaskWoken );`

`S_TASK_EVENT OS_WaitEventTimed(`
`  OS_TASK_EVENT EventMask,`
`  OS_TIME TimeOut);`
 	
`OS_TASK_EVENT OS_WaitEvent(`
`  OS_TASK_EVENT EventMask);`
	`uint32_t ulTaskNotifyTake(`
	`  BaseType_t xClearCountOnExit,`
	`  TickType_t xTicksToWait);`
	
	`BaseType_t xTaskNotifyWait(`
	`  uint32_t ulBitsToClearOnEntry,`
	`  uint32_t ulBitsToClearOnExit,`
	`  uint32_t *pulNotificationValue,`
	`  TickType_t xTicksToWait );`

Alokacja pamięci – statyczna, lub dynamiczna, jest wykonywana przez jeden z dostępnych alokatorów. EmbOS z kolei wymaga jawnego podania bloku pamięci, w którym zostanie umieszczony stos zadania. Druga różnica widoczna jest w prototypach funkcji implementujących zadanie – FreeRTOS pozwala na przekazanie argumentu w postaci wskaźnika typu void*, podczas gdy w systemie embOS zadania nie mają argumentów.

Ostatnią kwestią, o której należy pamiętać jest obsługa przerwań. O ile w przypadku systemu FreeRTOS nie były wymagane dodatkowe kroki, to w embOS musimy na początku każdego przerwania wywołać funkcję *OS_EnterInterrupt()*, natomiast na końcu – *OS_LeaveInterrupt()*. Jest to konieczne, aby poinformować system o aktywnym przerwaniu.

Po wykonaniu niezbędnych zmian w oryginalnym kodzie trzeba jeszcze dodać kolejne ścieżki i pliki w ustawieniach projektu. Znajdują się one w **tabeli 5**. Są to pliki niezbędne do obsługi peryferii mikrokontrolera, pamięci SDRAM, wyświetlacza oraz panelu dotykowego. Na potrzeby przykładu użyte zostały sterowniki dostarczane razem z biblioteką STM32Cube. Do projektu musimy dodać także wymienioną w tabeli bibliotekę matematyczną ARM. Wymaga ona zdefiniowania symbolu ARM_MATH_CM7 w ustawieniach projektu (*Preprocessor* → *Preprocessor Definitions*). Pozwala on na dodanie odpowiednich plików nagłówkowych z biblioteki CMSIS.

Ostatnią kwestią, którą poruszymy w tym rozdziale, jest tryb kompilacji dla operacji zmiennoprzecinkowych. Parametry kompilacji wszystkich komponentów i naszej aplikacji muszą być zgodne, co oznacza, że musimy dostosować się do biblioteki embOS. Interesujące nas parametry kompilacji znajdują się w jej nazwie: v7m (Cortex-M3/M4/M4F/M7/M7F), vfpv4 (VFPv4 z programową obsługą operacji zmiennoprzecinkowych), le (little endian). Wynika z tego, że w opcjach projektu, w polu *Code Generation* → *ARM FP ABI Type* należy ustawić wartość *SoftFP*. Z tego samego powodu do projektu została dodana biblioteka matematyczna w wersji niewymagającej sprzętowej obsługi operacji zmiennoprzecinkowych.

Biblioteka graficzna

W oryginalnym projekcie do stworzenia interfejsu graficznego została użyta biblioteka STemWin. Zmiana na bibliotekę emWin nie niesie ze sobą żadnych trudności, ponieważ API użytych w projekcie komponentów jest identyczne. Jedyne, co musimy zrobić, to dodać skompilowane źródła biblioteki i jej pliki nagłówkowe z pakietu ewaluacyjnego do naszego projektu. Wspomniane pliki znajdują się w katalogu GUI, w którym znajdziemy kilka wersji biblioteki:

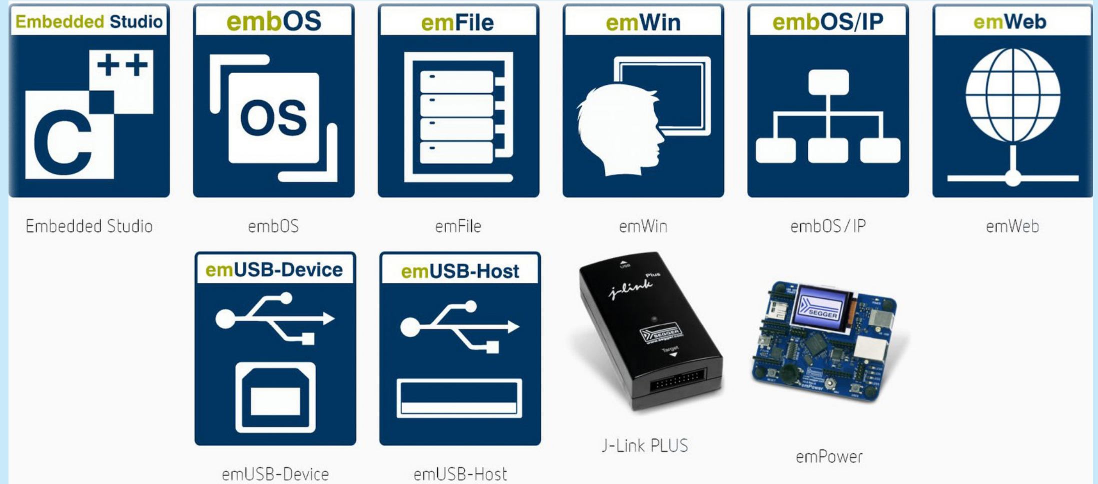
- **libGUI_ABGR_v7m_t_vfpv4_le_d.a** – wersja debug z formatem kolorów ABGR,
- **libGUI_ABGR_v7m_t_vfpv4_le_r.a** – wersja release z formatem kolorów ABGR,

Listing 1. Przykładowy kod korzystający z biblioteki embOS

```
#include <stdio.h>
#include <stdlib.h>
#include „stm32f746xx.h"
#include „RTOS.h"

static OS_TASK testTaskId;
static OS_STACKPTR int testTaskStack[2048];

static void testTask(void)
{
    while(1)
    {
        printf(„Hello Test Task!\n");
        OS_Delay(1000);
    }
}

void main(void)
{
    OS_InitKern();
    OS_InitHW();
    OS_CREATETASK(&testTaskId, „TestTask", testTask, 150, testTaskStack);
    OS_Start();
}
```


Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

- **libGUI_v7m_t_vfpv4_le_d.a** – wersja debug z formatem kolorów ARGB,
- **libGUI_v7m_t_vfpv4_le_r.a** – wersja release z formatem kolorów ARGB.



W przykładzie została użyta pierwsza z nich i dodana za pomocą opcji *Add Existing File*. Oprócz niej musimy także dodać odpowiednie ścieżki do plików nagłówkowych i pliki źródłowe konfigurujące bibliotekę do pracy z systemem embOS i wyświetlaczem znajdującym się na płytce STM32F746G-DISCO:

```
./GUI/Inc
./GUI/OS/GUI_X_embOS.c
./GUI/Setup/STM32F746_ST_STM32F746G_Discovery/
./GUI/Setup/STM32F746_ST_STM32F746G_Discovery/
BSP_GUI.c
./GUI/Setup/STM32F746_ST_STM32F746G_Discovery/
GUIConf.c
./GUI/Setup/STM32F746_ST_STM32F746G_Discovery/
LCDConf.c
```

Kolejną kwestią, którą musimy się zająć, jest konfiguracja linkera. W opisywanym projekcie biblioteka emWin pracuje w trybie *Memory Device*, co oznacza, że wszystkie operacje są wykonywane w buforze w pamięci mikrokontrolera, a gotowy obraz jest wysyłany do wyświetlacza. Dzięki temu można ograniczyć transmisję danych pomiędzy mikrokontrolerem a wyświetlaczem, a także uniknąć efektu migotania podczas rysowania kolejnych warstw obrazu, takich jak bitmapy, tekst i inne elementy interfejsu użytkownika. Podejście to niesie ze sobą konieczność zdefiniowania wystarczająco dużego bufora w pamięci, reprezentującego wyświetlany obraz. Znajduje się on w pliku **GUIConf.c** i ma atrybut **section („.GUI_RAM")**. Oznacza to, że powinien zostać umieszczony w sekcji pamięci o nazwie *.GUI_RAM*, której definicja, wraz z pozostałymi obszarami pamięci, znajduje się w pliku *flash_placement.xml*. Znajdziemy go w pakiecie ewaluacyjnym, w katalogu *BSP/ST/STM32F746_STM32F746G_Discovery/Setup*, skąd możemy go skopiować do swojego projektu i dodać w ustawieniach w polu *Linker* → *Section Placement File*, które będzie dostępne po zmianie linkera z Segger na GNU (pole *Linker* → *Linker*). Drugim z plików, który będzie nam potrzebny, jest STM32F746NG_MemoryMap.xml, znajdujący się w tym samym katalogu. Zawiera on definicje wszystkich dostępnych w systemie pamięci RAM oraz Flash. Podobnie jak poprzednio, powinniśmy go skopiować do projektu i dodać w ustawieniach, w polu *Build* → *Memory Map File*.

Zmiana linkera niesie ze sobą także potrzebę zmiany plików startowych. Niezbędne pliki znajdziemy w pakiecie ewaluacyjnym:

- ./BSP/ST/STM32F746_STM32F746G_Discovery/Setup/thumb_crt0.s
- ./BSP/ST/STM32F746_STM32F746G_Discovery/Setup/DeviceSupport/STM32F7x6_v1r1_Vectors.s
- ./BSP/ST/STM32F746_STM32F746G_Discovery/Setup/DeviceSupport/STM32F7xx_Startup.s

Zawierają one kod rozruchowy oraz definicje wektorów przerwań. Tak jak zwykle powinniśmy je dodać do projektu, jednocześnie usuwając obecnie używane:

```
Segger_Thumb Startup.s
Cortex_M_Startup.s
```

Nowe pliki startowe definiują funkcję Reset_Handler, która jest jednocześnie punktem wejściowym do aplikacji. Należy poinformować o tym linker, wpisując jej nazwę w ustawieniach projektu w polu *Linker* → *Entry Point*, zastępując dotychczasową wartość.

Ostatnią rzeczą, jaką powinniśmy zrobić, jest dodanie konfiguracji zegara. Funkcję konfiguracyjną – SystemClock_Config możemy skopiować z jednego z przykładów znajdujących się w bibliotece STM32Cube. W opisywanym przykładzie użyta została funkcja z pliku main.c znajdującego się w katalogu *Projects/STM32746G-Discovery/Applications/StemWin/StemWin_MemoryDevice/Core/Src* i wywołana na samym początku funkcji main(), przed inicjalizacją systemu embOS.

Krzysztof Chojnowski

Tabela 5. Pliki sterowników z biblioteki STM32 HAL używane w projekcie

STM32Cube_FW_F7
Drivers/CMSIS/Lib/GCC/libarm_cortexM7l_math.a
Drivers/BSP/STM32746G-Discovery
Drivers/BSP/STM32746G-Discovery/stm32746g_discovery_audio.c
Drivers/BSP/STM32746G-Discovery/stm32746g_discovery_sdram.c
Drivers/BSP/STM32746G-Discovery/stm32746g_discovery_ts.c
Drivers/BSP/STM32746G-Discovery/stm32746g_discovery.c
Drivers/BSP/Components
Drivers/BSP/Components/wm8994/wm8994.c
Drivers/BSP/Components/ft5336/ft5336.c
Drivers/STM32F7xx_HAL_Driver/Inc
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_cortex.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_dma.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_gpio.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_i2c.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_pwr.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_pwr_ex.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_rcc.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_rcc_ex.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_sai.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_hal_sdram.c
Drivers/STM32F7xx_HAL_Driver/Src/stm32f7xx_ll_fmc.c
Projects/STM32746G-Discovery/Templates/Inc/stm32f7xx_hal_conf.h

Poprzednie części kursu i dodatkowe materiały dostępne są na na stronie www.media.avt.pl