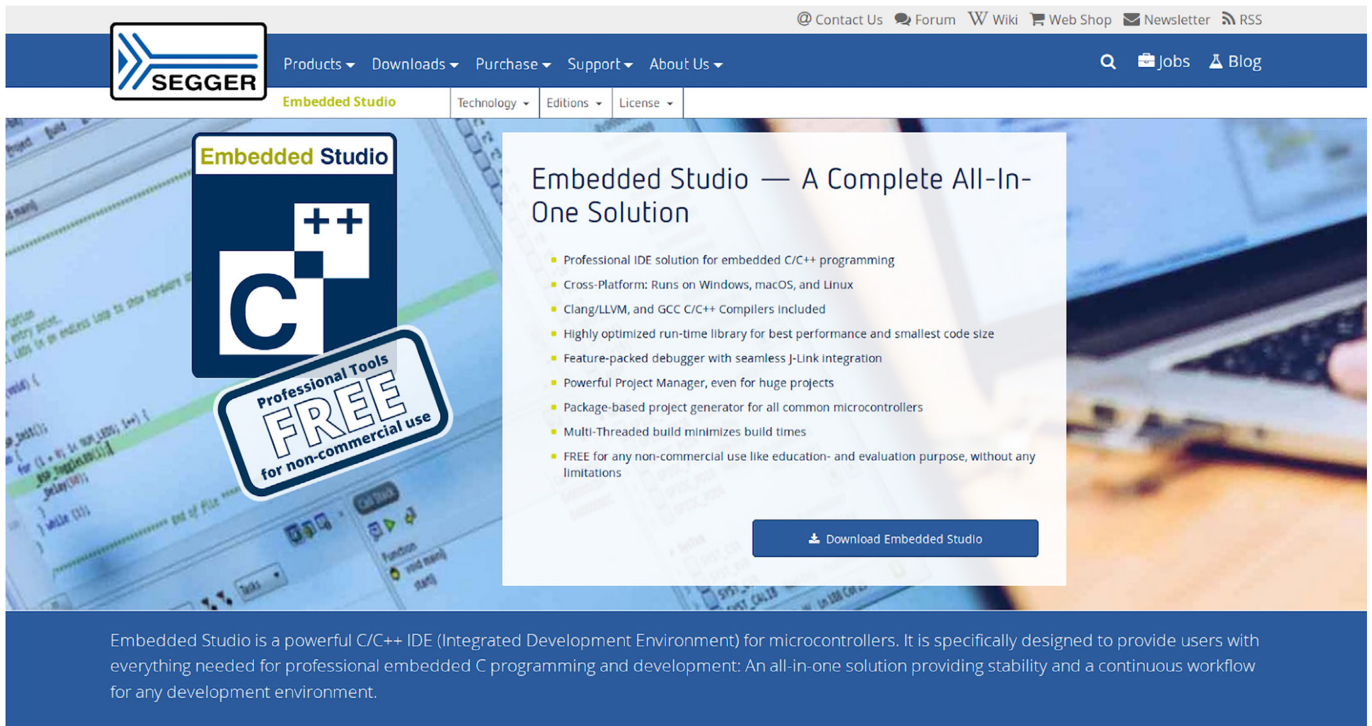


Segger Embedded Studio dla STM32 (2)

Biblioteki i przykłady programów

Prezentujemy interesującą alternatywę dla głównych trendów, która powinna zainteresować programistów systemów embedded bazujących na mikrokontrolerach: IDE+pakiet bibliotek firmy Segger. Do celów ewaluacyjnych – bezpłatne!

W wersji projektu przedstawionego miesiąc temu obsługa panelu dotykowego była realizowana przez napisany do tego celu moduł *mtouch*, który cyklicznie pobierał dane z kontrolera panelu dotykowego i umieszczał je w kolejce przetwarzanej przez jedno z zadań interfejsu graficznego. W tej wersji użyjemy jednego z komponentów biblioteki emWin, integrującego zdarzenia z panelu dotykowego z interfejsem użytkownika.

W bibliotece emWin dostępne są dwa sposoby obsługi paneli dotykowych. Pierwszy z nich – *Pointer Input Device,* pozwala na implementację interfejsu na bazie wskaźnika, którym użytkownik może klikać na poszczególne elementy, takie jak przyciski lub suwaki. Ze względu na swoją specyfikę metoda ta może być również używana do przyłączenia myszy lub joysticka do systemu.

Druga z metod, nazwana *MultiTouch support,* jest przeznaczona wyłącznie dla paneli dotykowych i zawiera między innymi rozpoznawanie gestów, za których pomocą użytkownik jest w stanie przesuwać, powiększać, pomniejszać i obracać elementy wyświetlane na ekranie. W przykładzie użyjemy tej właśnie metody, wspomagając się dodanym wcześniej sterownikiem panelu dotykowego z biblioteki STM32Cube.

Konfiguracja biblioteki emWin zawiera domyślnie inicjalizację modułu PID (*Pointer Input Device*), który jest uruchamiany w funkcji **LCD_X_DisplayDriver** w pliku *./GUI/Setup/STM32F746_ST_STM32F746G_Discovery/LCDConf.c*. Możemy go wyłączyć, usuwając z niej wywołanie **PID_X_Init**. W zamian za to musimy dodać kod do obsługi interfejsu *MultiTouch* składający się z kilku warstw.

Pierwszą warstwę obsługi panelu dotykowego stanowi przerwanie generowane przez kontroler FT5336. Jest ono wywoływane za każdym razem, gdy zostanie wykryte dotknięcie panelu dotykowego. Do konfiguracji kontrolera i wspomnianego przerwania na pinie GPIOI 13 służą funkcje **BSP_TS_Init** i **BSP_TS_ITConfig** wywoływane wewnątrz funkcji **guiTask** po skonfigurowaniu wszystkich komponentów interfejsu graficznego.

Za obsługę przerwania GPIO jest odpowiedzialna funkcja **EXTI15_10_IRQHandler**. Znajdują się w niej wspomniane wcześniej wywołania informujące system embOS o aktywnym przerwaniu oraz funkcja biblioteki STM32Cube – **HAL_GPIO_EXTI_IRQHandler**, odpowiedzialna za wyzerowanie flagi przerwania pochodzącego od odpowiedniego pinu i wywołująca zdefiniowaną w pliku *main.c* funkcję zwrotną **HAL_GPIO_EXTI_Callback**. Jej jedynym zadaniem jest powiadomienie zadania **tsTask** o dostępności nowych danych wygenerowanych przez kontroler panelu dotykowego. Zadanie tsTask jest drugą z warstw przetwarzania danych, jednak zanim przejdziemy do jego omówienia, warto zrozumieć ogólną zasadę działania modułu *MultiTouch* biblioteki emWin.

Biblioteka emWin jest w stanie wykryć kilka podstawowych gestów wykonywanych przez użytkownika na panelu dotykowym. Są to:

przesuwanie (*panning*), przybliżanie i oddalanie (*zooming*) oraz obracanie (*rotating*). Gesty te są wykrywane na podstawie dostarczanych do biblioteki danych opisujących sekwencję ruchu na panelu dotykowym. Do przekazywania danych do biblioteki służy funkcja **GUI_MTOUCH_StoreEvent**, przyjmująca dwa argumenty – wskaźniki na struktury **GUI_MTOUCH_EVENT** oraz **GUI_MTOUCH_INPUT**. Obie struktury pokazano na **listingu 2**.

Pierwszy z argumentów zawiera, między innymi, liczbę punktów w tablicy (*NumPoints*), będącej drugim argumentem wywołania oraz warstwę interfejsu (*LayerIndex*). Z kolei wspomniana tablica zawiera listę punktów, w których został dotknięty panel. Punkty o jednakowym polu *Id* dotyczą tego samego zdarzenia, dzięki czemu biblioteka może rozpoznać gesty składające się z więcej niż jednego punktu. Oprócz tego struktura zawiera także pole **Flags** przyjmujące wartości:

- **GUI_MTOUCH_FLAG_DOWN** – początek ruchu.
- **GUI_MTOUCH_FLAG_UP** – koniec ruchu.
- **GUI_MTOUCH_FLAG_MOVE** – ruch.

Za odpowiednie przygotowanie danych jest odpowiedzialne wspomniane wcześniej zadanie *tsTask*. Po otrzymaniu notyfikacji z przerwania od pinu GPIOI 13 pobiera ono dane z kontrolera za pomocą funkcji **BSP_TS_GetState**. W otrzymanej strukturze znajdują się współrzędne wszystkich wykrytych punktów, które są następnie tłumaczone na struktury danych wymaganych przez bibliotekę emWin, według poniższego schematu:

- Jeżeli wcześniej nie był wykonywany ruch, to jest generowane zdarzenie **GUI_MTOUCH_FLAG_DOWN**.
- Jeżeli wykryto kolejne przerwanie w serii, to jest generowane zdarzenie **GUI_MTOUCH_FLAG_MOVE**.
- Jeżeli był wykonywany ruch, a kolejne przerwanie nie zostało odebrane w określonym czasie, to jest generowane zdarzenie **GUI_MTOUCH_FLAG_UP**.
- Jeżeli liczba wykrytych punktów jest różna od liczby punktów z poprzedniego przerwania, to są generowane zdarzenia **GUI_MTOUCH_FLAG_UP**, a następnie **GUI_MTOUCH_FLAG_DOWN**, kończące analizę poprzedniego gestu i rozpoczynające nową sekwencję.

Opisany algorytm jest zaimplementowany w kodzie zadania zamieszczonym na **listingu 3**. Na końcu jego pętli głównej znajduje się dodatkowe opóźnienie 10 ms (funkcja *OS_Delay*) ograniczające częstotliwość odbierania danych z kontrolera, ponieważ kolejne wykrywane przez niego informacje nadpisują poprzednie.

Za konwersję danych pomiędzy sterownikiem kontrolera panelu dotykowego a biblioteką emWin jest odpowiedzialna funkcja **storeMtouchEvent**, przedstawiona na **listingu 4**. Po przekazaniu danych do biblioteki graficznej przechodzimy do trzeciej warstwy – obsługi gestów wykonanych na poszczególnych elementach interfejsu.

Przyjrzyjmy się teraz konfiguracji samej biblioteki emWin na potrzeby obsługi panelu dotykowego oraz konfiguracji komponentów graficznych, reagujących na wykryte gesty. Po pierwsze, musimy dodać inicjalizację modułu *MultiTouch* oraz włączyć wykrywanie gestów. Są za to odpowiedzialne dwie funkcje: **GUI_MTOUCH_Enable** oraz **WM_GESTURE_Enable**, wywoływane tuż po inicjalizacji biblioteki emWin za pomocą funkcji **GUI_Init**. Dodatkowo, komponenty interfejsu mające reagować na gesty musimy utworzyć razem z flagą **WM_CF_GESTURE**.

W przykładzie została ona dodana do domyślnych flag menedżera okien ustawianych przez funkcję **WM_SetCreateFlags**. Dzięki temu wszystkie tworzone okna będą w stanie odbierać gesty wykonywane na panelu dotykowym. Wykryte gesty można odebrać za pomocą wywołania zwrotnego rejestrowanego za pomocą funkcji **WM_SetCallback**. Przyjmuje ona uchwyt na wybrane okno oraz wspomniane wywołanie. W przykładzie oba okna zawierające wykresy są obsługiwane przez funkcję **graphCallback**. Cała opisana konfiguracja jest wykonywana na początku funkcji implementującej zadanie **guiTask**.

Listing 2. Typy danych reprezentujące zdarzenia na panelu dotykowym

```
typedef struct
{
    int             LayerIndex;
    unsigned        NumPoints;
    GUI_TIMER_TIME  TimeStamp;
    PTR_ADDR        hInput;
} GUI_MTOUCH_EVENT;

typedef struct
{
    I32 x;
    I32 y;
    U32 Id;
    U16 Flags;
} GUI_MTOUCH_INPUT;
```

Listing 3. Zadanie przetwarzające dane z kontrolera panelu dotykowego

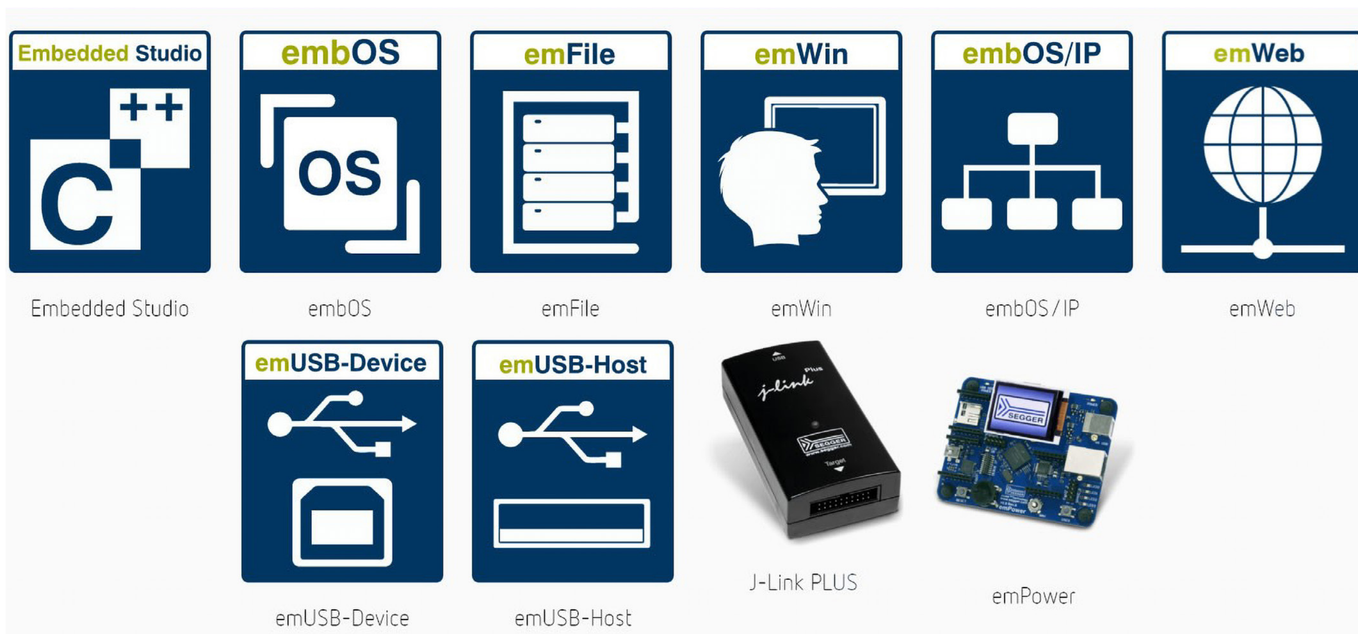
```
void tsTask(void)
{
    OS_TASK_EVENT event;
    int moving = 0;
    int lastPoints = 0;

    while (1)
    {
        event = OS_WaitEvent_Timed(0xFFFFFFFF, 100);
        if (event == 0 && moving > 0)
        {
            moving = 0;
            storeMtouchEvent(NULL, GUI_MTOUCH_FLAG_UP);
        } else if (event != 0)
        {
            TS_StateTypeDef state;
            if (BSP_TS_GetState(&state) != TS_OK)
            continue;
            if (state.touchDetected < 1 || state.touchDetected > 2)
            continue;
            if (lastPoints != state.touchDetected)
            {
                moving = 0;
                lastPoints = state.touchDetected;
                storeMtouchEvent(NULL, GUI_MTOUCH_FLAG_UP);
            }
            if (moving > 0)
            {
                storeMtouchEvent(&state, GUI_MTOUCH_FLAG_MOVE);
            } else
            {
                moving = 1;
                storeMtouchEvent(&state, GUI_MTOUCH_FLAG_DOWN);
            }
            OS_Delay(10);
        }
    }
}
```

Listing 4. Konwersja danych z panelu dotykowego na format wymagany przez bibliotekę emWin

```
void storeMtouchEvent(TS_StateTypeDef* tsState, int eventType)
{
    GUI_MTOUCH_EVENT event;
    event.LayerIndex = 0;
    event.TimeStamp = OS_GetTime();
    GUI_MTOUCH_INPUT input[2];
    if (eventType == GUI_MTOUCH_FLAG_UP)
    {
        event.NumPoints = 2;
        input[0].Flags = GUI_MTOUCH_FLAG_UP;
        input[1].Flags = GUI_MTOUCH_FLAG_UP;
    } else if ((eventType == GUI_MTOUCH_FLAG_DOWN)
             || (eventType == GUI_MTOUCH_FLAG_MOVE))
    {
        event.NumPoints = tsState->touchDetected;
        input[0].Id = 0;
        input[0].x = tsState->touchX[0];
        input[0].y = tsState->touchY[0];
        input[0].Flags = eventType;
        input[1].Id = 0;
        input[1].x = tsState->touchX[1];
        input[1].y = tsState->touchY[1];
        input[1].Flags = eventType;
    }
}
```

Informacje dotyczące wykrytych gestów pojawiają się w funkcji zwrotnej **graphCallback** jako argument **WM_MESSAGE *pMsg**, podobnie jak wszystkie pozostałe wiadomości związane z oknem. W związku z tym jest konieczne filtrowanie otrzymanych wiadomości po wartości pola **pMsg → MsgId**, które dla gestów ma wartość **WM_GESTURE**. Wszystkie inne zdarzenia są przekazywane do domyślnej obsługi realizowanej przez funkcję biblioteczną **GRAPH_Callback**. Sama obsługa wykrytych gestów, podobnie jak w oryginalnym przykładzie, sprowadza się do wygenerowania powiadomień dla zadań **signalTask** oraz **fftTask**, które z kolei są odpowiedzialne za przesuwanie i skalowanie wykresów.



Fragment funkcji graphCallback przetwarzający gesty znajduje się na **listingu 5**. O ile gesty związanie z przesuwaniem nie wymagają dalszego komentarza, o tyle warto wyjaśnić zachowanie funkcji w przypadku przybliżania i oddalania.

Struktura danych **WM_GESTURE_INFO** znajdująca się wewnątrz **WM_MESSAGE** zawiera pole współczynnika skali (*Factor*). Jest on ustawiany na arbitralną wartość wynoszącą 100 na początku gestu, o którym świadczy flaga **WM_GF_BEGIN**. Ta wartość jest modyfikowana przez bibliotekę odpowiednio do szybkości wykonywania gestu przez użytkownika i używana w funkcji **graphCallback** do generowania opóźnienia, bez którego skalowanie odbywa się zbyt szybko. Za wielkość opóźnienia są odpowiedzialne odpowiednio dobrane progi, dzięki którym jest możliwe wysłanie powiadomienia dopiero po zmianie współczynnika o żądaną wartość.

Listing 5. Przetwarzanie wykrytych gestów w funkcji zwrotnej graphCallback

```
static int lastZoomingFactor;
OS_TASK *destTask;

if (pMsg->MsgId == WM_GESTURE)
{
    if (pMsg->hWin == appGlobals.fftGraph) destTask = &appGlobals.fftTaskId;
        else if (pMsg->hWin == appGlobals.signalGraph) destTask = &appGlobals.signalTaskId;
            else return;
    WM_GESTURE_INFO *info = (WM_GESTURE_INFO *)pMsg->Data.p;

    if ((info->Flags & WM_GF_PAN) == WM_GF_PAN)
    {
        if ((info->Point.x) > 0) OS_SignalEvent(TASK_EVENT_CHANGE_VIEW_MOVE_RIGHT, destTask);
            else if ((info->Point.x) < 0) OS_SignalEvent(TASK_EVENT_CHANGE_VIEW_MOVE_LEFT, destTask);
    } else if ((info->Flags & (WM_GF_ZOOM | WM_GF_BEGIN)) == (WM_GF_ZOOM | WM_GF_BEGIN))
    {
        // Set the arbitrary initial factor for zoom event.
        info->Factor = 100;
        lastZoomingFactor = 100;
    } else if (((info->Flags & (WM_GF_ZOOM)) == WM_GF_ZOOM) && (info->Factor != 0))
    {
        // The arbitrary factor change threshold for more gesture inertia.
        if ((lastZoomingFactor - info->Factor) > 25)
        {
            OS_SignalEvent(TASK_EVENT_CHANGE_VIEW_ZOOM_OUT_X, destTask);
            lastZoomingFactor = info->Factor;
        }
                if ((lastZoomingFactor - info->Factor) < -25)
        {
            OS_SignalEvent(TASK_EVENT_CHANGE_VIEW_ZOOM_IN_X, destTask);
            lastZoomingFactor = info->Factor;
        }
    }
}
```

Na koniec

Na obsłudze panelu dotykowego kończy się opis migracji projektu oscyloskopu dla zestawu STM32F746G-DISCO do środowiska Embedded Studio oraz biblioteki firmy SEGGER. Pomimo znaczących różnic w stosunku do środowiska SW4STM32, Embedded Studio jest na tyle intuicyjne, że po krótkim zaznajomieniu się z jego możliwościami konfiguracja projektu nie stanowi problemu. Podobnie wygląda kwestia bibliotek embOS i emWin, które stanową cenną pomoc w budowaniu aplikacji ze złożonym graficznym interfejsem użytkownika.

Krzysztof Chojnowski

REKLAMA

MEDIA ELEKTRONIKA PRAKTYCZNA

Aby skorzystać z materiałów dodatkowych dołączonych do numeru, należy:
1. Wejść na stronę **www.media.avt.pl**
2. Zarejestrować się lub zalogować
3. Wybrać wydanie „Elektroniki Praktycznej", które ma trafić do biblioteki osobistej
4. Odpowiedzieć na proste pytanie dotyczące bieżącego numeru
5. Pobrać pliki