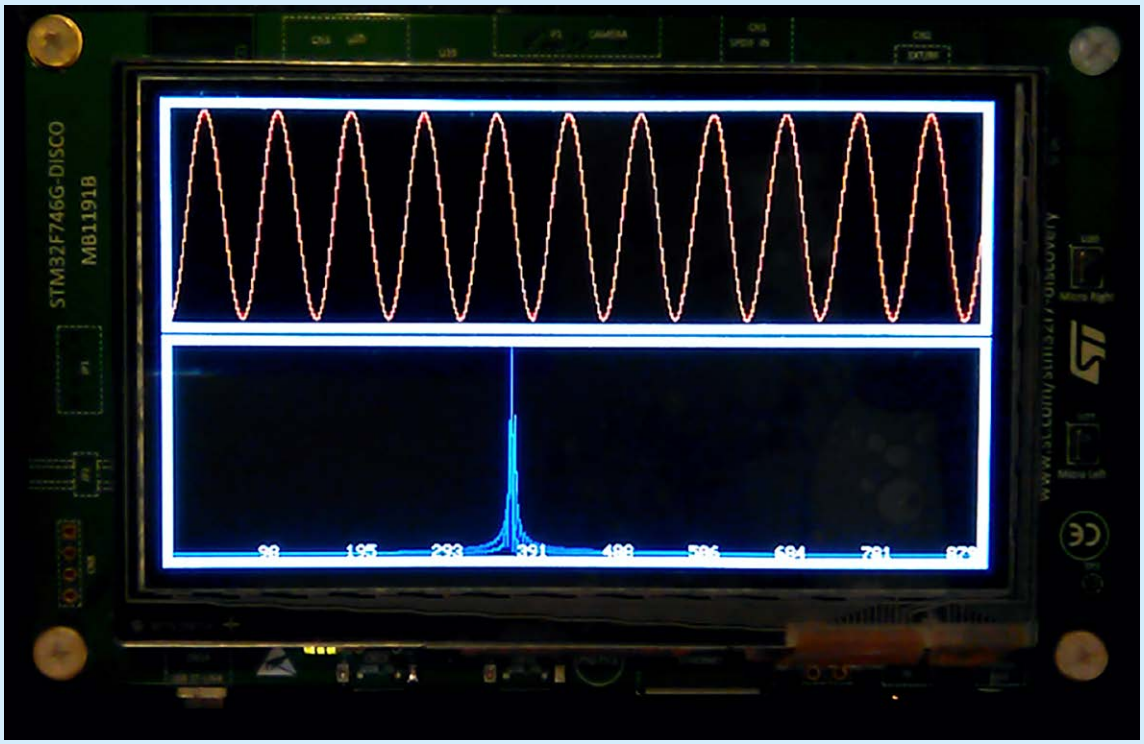

Podstawy programowania STM32F746G-DISCO (3)

Jak zbudować oscyloskop z FFT z użyciem STM32F746G-DISCO

W ostatniej części artykułu poświęconego aplikacji próbkującej i wyświetlającej sygnał z wejścia liniowego zostaną omówione pakiet BSP, biblioteki graficzna STemWin, matematyczna ARM CMSIS DSP oraz moduł do wykrywania podstawowych gestów wykonanych przez użytkownika na panelu dotykowym.

BSP, czyli *Board Support Package*, jest zbiorem sterowników oraz interfejsów przygotowanych dla określonego mikrokontrolera oraz towarzyszących mu zewnętrznych peryferiów. BSP dostarczany wraz z biblioteką Cube od ST, umożliwia łatwą i szybką konfigurację zestawu ewaluacyjnego STM32F746G-DISCOVERY i skupienie się na tworzeniu aplikacji. Konfiguracja biblioteki jest wykonywana w pliku *stm32f7xx_hal_conf.h*. W przykładzie sprowadza się ona do włączenia odpowiednich modułów w sekcji *Module Selection*.

Do poprawnego działania, BSP musi dostać informację o czasie, za pośrednictwem funkcji inkrementującej wewnętrzne liczniki. Najłatwiej można to zrobić wywołując funkcję *HAL_IncTick()* w regularnych odstępach czasu generowanych przez przerwania sprzętowe. Z uwagi na to, że w projekcie jest używany system FreeRTOS, który może informować aplikację o każdorazowym przerwaniu od zegara systemowego, funkcja *HAL_IncTick()* jest wywoływana wewnątrz funkcji

```
void vApplicationTickHook( void )
{
    HAL_IncTick();
}
```

Funkcja *vApplicatioTickHook* została zdefiniowana w pliku *main.c* i jest wywoływana z częstotliwością zegara systemowego, ustawianego parametrem *configTICK_RATE_HZ*, pod warunkiem, że wartość *configUSE_TICK_HOOK* jest ustawiona na 1. Oba te parametry powinny być zdefiniowane w pliku *FreeRTOSConfig.h*.

Pierwszą czynnością, którą należy wykonać przed skorzystaniem ze sterowników jest wywołanie funkcji inicjalizacyjnej *HAL_Init();*. Jest ona wywoływana na samym początku funkcji main. Bezpośrednio po niej, w funkcji *SystemClock_Config*, konfigurowane są zegary mikrokontrolera. Jest ona zdefiniowana w pliku *main.c* i wykorzystuje funkcje biblioteczne *HAL_RCC_OscConfig* oraz *HAL_RCC_ClockConfig*.

Teraz można już przystąpić do konfiguracji peryferiów. Na początek w zadaniu *GUI_Task* są inicjalizowane

Listing 1. Aby otrzymać informacje o zakończeniu połowy i całego transferu DMA należy przedefiniować dwie funkcje oznaczone jako __weak

```
__weak void BSP_AUDIO_IN_TransferComplete_CallBack(void)
__weak void BSP_AUDIO_IN_HalfTransfer_CallBack(void)
W projekcie są one zdefiniowanie w pliku main.c jako:
void BSP_AUDIO_IN_HalfTransfer_CallBack(void) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xTaskNotifyFromISR(appGlobals.fftTaskId, TASK_EVENT_DMA_HALF_DONE, eSetBits,
&xHigherPriorityTaskWoken);
    xTaskNotifyFromISR(appGlobals.signalTaskId, TASK_EVENT_DMA_HALF_DONE, eSetBits,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

void BSP_AUDIO_IN_TransferComplete_CallBack(void) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
   xTaskNotifyFromISR(appGlobals.fftTaskId, TASK_EVENT_DMA_DONE, eSetBits,
&xHigherPriorityTaskWoken);
    xTaskNotifyFromISR(appGlobals.signalTaskId, TASK_EVENT_DMA_DONE, eSetBits,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
```

zewnętrzna pamięć RAM oraz panel dotykowy. Sterownik panelu dotykowego wymaga podania rozmiaru wyświetlacza w pikselach:

```
BSP_SDRAM_Init();
BSP_TS_Init(LCD_GetXSize(),
LCD_GetYSize());
```

Dopiero wywołaniu drugiej z tych funkcji, można uruchomić licznik programowy odpowiedzialny za odczyt danych z panelu dotykowego. Ostatnią funkcją BSP wywoływana w tym zadaniu jest *__HAL_RCC_CRC_CLK_ENABLE();*, Włącza ona zegar modułu CRC procesora, który jest wymagany przez bibliotekę graficzną. Dalsza część konfiguracji peryferiów znajduje się w zadaniu *Signal_Task*. Po otrzymaniu powiadomienia o gotowości interfejsu graficznego, wywoływane są dwie funkcje:

```
BSP_AUDIO_IN_Init(INPUT_DEVICE_
INPUT_LINE_1, DEFAULT_AUDIO_IN_VOLUME,
DEFAULT_AUDIO_IN_FREQ);
BSP_AUDIO_IN_Record((uint16_t*)
appGlobals.dmaBuffer, DMA_BUFFER_LENGTH);
```

Pierwsza z nich konfiguruje zewnętrzny przetwornik audio, tak aby korzystał z wejścia liniowego (*INPUT_DEVICE_INPUT_LINE_1*), domyślnego poziomu głośności wynoszącego 64 (na 100) oraz domyślnego próbkowania 16 kHz. Druga funkcja uruchamia pomiary z wykorzystaniem DMA. Pierwszym argumentem jest tablica przeznaczona na próbki sygnału, drugim zaś jej długość. W przykładzie długość tablicy wynosi 16384 elementy, co daje 8192 próbki, ponieważ przetwornik próbkuje i umieszcza w tablicy dane z dwóch kanałów wejściowych.

Użycie DMA wymaga jeszcze zdefiniowania przerwań, aby otrzymać informację o zakończeniu pomiaru. Na początek trzeba dodać bezpośrednią obsługę przerwania od DMA. Odpowiednia funkcja została zdefiniowana w pliku stm32f7xx_it.c:

```
void AUDIO_IN_SAIx_DMAx_IRQHandler(void)
{
    HAL_DMA_IRQHandler(haudio_in_sai.
hdmarx);
}
```

Jest to w rzeczywistości zwykła funkcja obsługi przerwania DMA, co można sprawdzić w pliku stm32746g_discovery_audio.h:

```
#define AUDIO_IN_SAIx_DMAx_IRQHandler
DMA2_Stream7_IRQHandler.
```

Funkcja ta z kolei informuje o fakcie wystąpienia przerwania bibliotekę BSP, za pomocą wywołania funkcji *HAL_DMA_IRQHandler*. Jako argument przyjmuje ona wskaźnik na strukturę *SAI_HandleTypeDef*, znajdującą się w *stm32746g_discovery_audio.c*. Należy ją więc zadeklarować jako *extern* w pliku *stm32f7xx_it.c*:

```
extern SAI_HandleTypeDef haudio_in_sai;
```

Dalsza część przetwarzania przerwania znajduje się już w bibliotece. Aby otrzymać informacje o zakończeniu połowy i całego transferu DMA należy przedefiniować dwie funkcje oznaczone jako *__weak* w pliku *stm32746g_discovery_audio.c*, co pokazano na **listingu 1**. Operacje wykonywane wewnątrz tych funkcji zostały już omówione w poprzedniej części, dotyczącej systemu FreeRTOS. Istotne jest to, że przekazują one informację o końcu transferu, lub jego połowy do zadań *Signal_Task* oraz *FFT_Task*.

Wywołanie ostatniej z używanych funkcji BSP znajduje się w obsłudze licznika programowego (*TouchPanel_TimerCallback*), omawianej także w poprzedniej części. Funkcja ta pobiera aktualny stan panelu dotykowego i zapisuje w strukturze typu *TS_StateTypeDef*, zawierającej m. in. liczbę wykrytych punktów oraz ich współrzędne *BSP_TS_GetState(&tsState);*.

STemWin

Biblioteka graficzna STemWin zawiera zbiór komponentów przydatnych przy tworzeniu interfejsów graficznych w oparciu o wyświetlacze graficzne. Jest ona niezależna od typu użytego mikrokontrolera, wyświetlacza oraz systemu operacyjnego (w tym jego braku).

Biblioteka wymaga kilku zewnętrznych plików konfiguracyjnych zapewniających poprawną obsługę posiadanego sprzętu. Pliki te znajdują się w katalogach *src*, *in* oraz *inc* projektu i zostały wzięte z przykładu *StemWin_HelloWorld* biblioteki *STM32Cube* pobranej ze strony *http://goo.gl/ytv1Fv*. Poza niżej wymienionymi plikami STemWin nie potrzebuje żadnej dodatkowej konfiguracji.

- **GUIConf.h** zawiera podstawową konfigurację biblioteki StemWin (m. in. domyślną czcionkę, włączenie menadżera okien).
- **GUIConf.c** definiuje funkcję *GUI_X_Config* odpowiedzialną za alokowanie pamięci.
- **LCDConf.h** udostępnia strukturę *LCD_LayerPropTypedef* reprezentującą warstwę interfejsu graficznego i przechowującą takie dane jak rozmiar wyświetlacza i liczbę bajtów przypadającą na jeden piksel.
- **LCDConf.c** definiuje podstawowe parametry i funkcje obsługi wyświetlacza LCD, z których korzysta biblioteka STemWin.

Przed utworzeniem interfejsu graficznego, biblioteka musi zostać zainicjalizowana. Z uwagi na wykorzystanie

Listing 2. W projekcie zaimplementowano interfejs graficzny służący do prezentacji danych, oparty o dwa komponenty typu GRAPH

```
appGlobals.signalGraph = GRAPH_CreateEx(0, 0, LCD_GetXSize(), LCD_GetYSize()/2, WM_HBKWIN, WM_CF_
SHOW, 0, GUI_ID_GRAPH0);
GRAPH_SetBorder(appGlobals.signalGraph, 5, 5, 5, 5);
appGlobals.signalGraphData = GRAPH_DATA_YT_Create(GUI_RED, SIGNAL_LENGTH, NULL, 0);
GRAPH_AttachData(appGlobals.signalGraph, appGlobals.signalGraphData);

appGlobals.fftGraph = GRAPH_CreateEx(0, LCD_GetYSize()/2, LCD_GetXSize(), LCD_GetYSize()/2, WM_
HBKWIN, WM_CF_SHOW, 0, GUI_ID_GRAPH1);
GRAPH_SetBorder(appGlobals.fftGraph, 5, 5, 5, 5);
appGlobals.fftGraphScale = GRAPH_SCALE_Create(LCD_GetYSize()/2-12, GUI_TA_LEFT, GRAPH_SCALE_CF_
HORIZONTAL, 50);
GRAPH_SCALE_SetFactor(appGlobals.fftGraphScale, (2.0*DEFAULT_AUDIO_IN_FREQ/DMA_BUFFER_LENGTH));
GRAPH_AttachScale(appGlobals.fftGraph, appGlobals.fftGraphScale);
appGlobals.fftGraphData = GRAPH_DATA_YT_Create(GUI_BLUE,FFT_LENGTH, NULL, 0);
GRAPH_AttachData(appGlobals.fftGraph, appGlobals.fftGraphData);
```

systemu operacyjnego, czynność ta musi być wykonana wewnątrz jednego z zadań – w tym przypadku *GUI_Task*, od razu po opisywanej wcześniej inicjalizacji BSP.

Pierwszą funkcją *STemWin*, którą należy wywołać jest GUI_Init. Wszystkie pozostałe funkcje biblioteczne mogą zostać wywołane dopiero po niej. Wyjątek stanowi jedynie funkcja *WM_SetCreateFlags* służąca do ustawiania domyślnych flag podczas tworzenia nowych okien interfejsu. W przykładzie jest to tylko jedna flaga – *WM_CF_MEMDEV*. Powoduje ona, że odrysowywanie obrazu jest wykonywane w buforze w pamięci, a nie bezpośrednio na wyświetlaczu. Dopiero po przygotowaniu bufora, jest on wyświetlany, co zapobiega migotaniu obrazu. Ostatnim krokiem inicjalizacji jest wyczyszczenie wyświetlacza:

```
WM_SetCreateFlags(WM_CF_MEMDEV);
GUI_Init();
GUI_Clear();
```

W projekcie zaimplementowano bardzo prosty interfejs graficzny służący do prezentacji danych. Został on oparty o dwa komponenty typu GRAPH – reprezentujące wykresy. Są one tworzone w zadaniu *GUI_Task*, co pokazano na **listingu 2**. Jako pierwszy tworzony jest wykres sygnału za pomocą funkcji *GRAPH_CreateEx*. Pierwsze dwa argumenty to pozycja lewego górnego rogu okna wykresu na ekranie, kolejne dwa to jego rozmiar. Następnie należy podać okno, do którego dodany zostanie wykres – makro *WM_HBKWIN* zwraca uchwyt do okna *Desktop* będącego podstawowym oknem każdego interfejsu graficznego. Następne na liście argumentów są flagi wykorzystywane podczas tworzenia wykresu. *WM_CF_SHOW* oznacza wyświetlenie okna od razu po jego utworzeniu. Wartości flag podawane w tym polu są wspólne dla wszystkich typów okien, w przeciwieństwie do flag podawanych w przedostatnim argumencie. W projekcie nie są wykorzystywane dodatkowe flagi komponentu. Ostatni argument to *ID*, które musi być unikalne w obrębie komponentów znajdujących się w obrębie jednego okna. Wartość *GUI_ID_GRAPH0* jest zdefiniowana w pliku bibliotecznym *GUI.h*. W razie konieczności definiowania własnych identyfikatorów, powinny one mieć wartości większe od *GUI_ID_USER (0x800)*. Opisywana funkcja zwraca uchwyt do komponentu wykresu przechowywanego w zmiennej globalnej.

Kolejna funkcja – *GRAPH_SetBorder* tworzy marginesy wokół pola wykresu. Pierwszym argumentem jest uchwyt do wcześniej utworzonego komponentu. Cztery kolejne parametry oznaczają grubość marginesów: lewego, górnego, prawego i dolnego, wyrażoną w pikselach.

Po utworzeniu okna wykresu należy dołączyć do niego jeden, lub więcej obiektów danych. Każdy z nich tworzony jest za pomocą funkcji *GRAPH_DATA_YT_Create*. Tworzy ona dane reprezentujące m. in. przebiegi czasowe, w których współrzędna X oznacza numery kolejnych próbek, a współrzędna Y – ich wartości. Lista argumentów zawiera kolejno: kolor, maksymalną liczbę punktów na ekranie, tablicę zawierającą dane początkowe oraz jej długość. W tym przypadku dane zostaną dopiero pobrane, dlatego zamiast tablicy podawany jest wskaźnik pusty NULL. Na koniec można dodać dane do okna wykresu za pomocą funkcji *GRAPH_AttachData*, przyjmującej uchwyty do okna wykresu oraz obiektu danych.

Oba wykresy: sygnału i FFT są tworzone w ten sam sposób, jednak do drugiego z nich dodawana jest jeszcze skala. Jest ona tworzona za pomocą funkcji *GRAPH_SCALE_Create*. Jej pierwszym argumentem jest pozycja na oknie wykresu – ustawiana na 12 pikseli od jego dolnej krawędzi. Za pomocą drugiego argumentu można ustawić przesunięcie tekstu do lewej, prawej, lub jego wyśrodkowanie (gdyby skala dotyczyła osi Y byłoby to przesunięcie tekstu do góry, do dołu, lub wyśrodkowanie). Kolejny argument, to wybór osi poziomej (*GRAPH_SCALE_CF_HORIZONTAL*) lub pionowej (*GRAPH_SCALE_CF_VERTICAL*). Ostatni argument to odległość pomiędzy kolejnymi wartościami w pikselach.

Powyższe wywołania kończą tworzenie interfejsu graficznego w obrębie zadania *GUI_Task*. Ma ono jednak jeszcze jedną istotną funkcję. Jest nią cykliczne wywoływanie funkcji *GUI_Delay*, przyjmującej jako argument wartość opóźnienia oraz odpowiedzialnej za aktualizację interfejsu graficznego w razie jakichkolwiek zmian.

Rysowanie wykresów odbywa się w zadaniach *Signal_Task* i *FFT_Task*. W obu przypadkach wygląda ono podobnie, jednak w zadaniu *FFT_Task* jest zmieniana także skala, dlatego właśnie to zadanie posłuży do omówienia aktualizacji danych.

Wykresy aktualizowane są w pętli głównej zadania po otrzymaniu jednej z dwóch notyfikacji: *TASK_EVENT_DMA_HALF_DONE* lub *TASK_EVENT_DMA_DONE*. Otrzymanie jej oznacza, że dane w jednej połówce bufora są gotowe do przetworzenia. W pierwszej kolejności wykonywane są obliczenie transformaty oraz skalowanie (omówione w następnym punkcie), a wynikowe dane trafiają do bufora: *appGlobals.fftOutput*. Dalsze kroki zaprezentowano na **listingu 3**.

W pętli for wybierane są punkty, które powinny zostać wyświetlone. Nie wybierane są one od początku bufora danych, ponieważ brany jest od uwagę offset wynikający z możliwego przesunięcia początku wykresu, dodatkowo dzielony przez współczynnik jego skali. Ostateczna lista punktów jest wpisywana do tablicy appGlobals.fftDisplay o stałej długości równej szerokości pola wykresu.

Listing 3. Wyświetlenie wyników przetwarzania
```
for (idx = 0; idx < FFT_LENGTH; idx++)
    appGlobals.fftDisplay[idx] = appGlobals.fftOutput[idx+displayOffsetX/displayScaleX];
GRAPH_DetachData(appGlobals.fftGraph, appGlobals.fftGraphData);
GRAPH_DATA_YT_Delete(appGlobals.fftGraphData);
appGlobals.fftGraphData = GRAPH_DATA_YT_Create(GUI_BLUE, FFT_LENGTH, appGlobals.fftDisplay, FFT_LENGTH);
GRAPH_AttachData(appGlobals.fftGraph, appGlobals.fftGraphData);
```

Listing 4. Funkcje odpowiedzialne za zmianę offsetu
```
if(notificationValue & TASK_EVENT_CHANGE_VIEW_MOVE_LEFT) {
    if(displayOffsetX + displayScaleX*(FFT_LENGTH+10)<SIGNAL_SAMPLES) {
        displayOffsetX+= 10*displayScaleX;
        GRAPH_SCALE_SetOff(appGlobals.fftGraphScale, -1*displayOffsetX/displayScaleX);
    }
}

if(notificationValue & TASK_EVENT_CHANGE_VIEW_MOVE_RIGHT) {
    if(displayOffsetX >= 10*displayScaleX) {
        displayOffsetX-=10*displayScaleX;
        GRAPH_SCALE_SetOff(appGlobals.fftGraphScale, -1*displayOffsetX/displayScaleX);
    }
}
```

Niestety biblioteka STemWin uniemożliwia dodanie wielu punktów jednocześnie bez odświeżania wykresu po każdym z nich, dlatego obiekt danych *appGlobals.fftGraphData* jest przed każdą aktualizacją usuwany funkcjami *GRAPH_DetachData* i *GRAPH_DATA_YT_Delete* oraz tworzony na nowo za pomocą omawianej już funkcji *GRAPH_DATA_YT_Create* – tym razem jednak przekazywana jest to niej tablica z wartościami początkowymi. Nowo utworzony obiekt danych jest ponownie dodawany do okna wykresu za pomocą wywołania *GRAPH_AttachData*. We wszystkich funkcjach używane są komponenty utworzone wcześniej w zadaniu *GUI_Task*. Pozostałe notyfikacje służą do aktualizacji skalowania i przesunięcia wykresu. Pierwsze dwie odpowiedzialne są za zmianę offsetu (**listing 4**),

Po otrzymaniu notyfikacji *TASK_EVENT_CHANGE_VIEW_MOVE_LEFT*, offset jest zwiększany po wcześniejszym sprawdzeniu, czy nie zostanie przekroczona wartość maksymalna wynikająca z liczby zebranych próbek. Wartość przesunięcia zależna jest od aktualnego współczynnika skali przechowywanego w zmiennej displayScaleX. Po obliczeniu przesuwana jest także skala wyświetlana pod wykresem. Z uwagi na to, że wykonuje ona automatyczne skalowanie, przekazywana do niej wartość przesunięcia jest najpierw dzielona przez współczynnik skali. Drugi warunek, odpowiedzialny za przesunięcie wykresu w prawo, odpowiednio zmniejsza offset po sprawdzeniu czy nowa wartość nie będzie mniejsza od zera, a następnie w ten sam sposób modyfikuje przesunięcie skali.

Obsługa zmiany współczynnika skali została przedstawiona na **listingu 5**. W przypadku zbliżenia, pierwszym krokiem jest sprawdzenie czy współczynnik skali może zostać zmniejszony. Następnie modyfikowany jest offset, tak aby punkt środkowy wykresu pozostał bez zmiany. Mnożenie przez wartość *scaleFactor*, definiowaną na początku kodu zadania:

```
float32_t scaleFactor =
2.0*DEFAULT_AUDIO_IN_FREQ/
DMA_BUFFER_LENGTH;
```

jest spowodowana początkową inicjalizacją skali wykresu w zadaniu *GUI_Task* tą samą wartością. Wynika ona z podziałki 2 Hz na jeden piksel przy największym zbliżeniu wykresu. Współczynnik skali jest następnie zmniejszany, a offset dzielony przez *scaleFactor*. Obliczone wartości przekazywane są do funkcji modyfikujących skalę i przesunięcie *GRAPH_SCALE_SetFactor* oraz *GRAPH_SCALE_SetOff*. Podczas oddalania wykresu, pierwszy warunek sprawdza, czy zmiana skali nie spowoduje wykroczenia poza maksymalna liczbę zbieranych próbek. Następnie, analogicznie do zbliżania, modyfikowane są przesunięcie i współczynnik skali, jednak tym razem konieczne jest jeszcze sprawdzenie czy nie nastąpiło przekroczenie dozwolonych wartości spowodowane zmianą offsetu. Jeżeli tak się stanie, jest on dodatkowo modyfikowany. Na koniec aktualizowana jest skala pod wykresem.

ARM CMSIS DSP

ARM CMSIS DSP jest biblioteką zawierającą zbiór funkcji matematycznych przydatnych zarówno przy

Listing 5. Obsługa zmiany współczynnika skali
```
if(notificationValue & TASK_EVENT_CHANGE_VIEW_ZOOM_IN_X) {
    if(displayScaleX>1) {
        displayOffsetX*=scaleFactor;
        displayOffsetX += FFT_LENGTH*displayScaleX/4;
        displayScaleX--;
        displayOffsetX /= scaleFactor;

        GRAPH_SCALE_SetFactor(appGlobals.fftGraphScale,  scaleFactor*displayScaleX);
        GRAPH_SCALE_SetOff(appGlobals.fftGraphScale, -1*displayOffsetX/displayScaleX);
    }
}

if(notificationValue & TASK_EVENT_CHANGE_VIEW_ZOOM_OUT_X) {
    if((FFT_LENGTH-1)*(displayScaleX+1)<SIGNAL_SAMPLES) {
        displayOffsetX *= scaleFactor;
        displayOffsetX -= FFT_LENGTH * displayScaleX / 2;
        displayScaleX++;
        displayOffsetX /= scaleFactor;
        if(displayOffsetX<0) displayOffsetX=0;
        else if(displayScaleX*(displayOffsetX+FFT_LENGTH)>=SIGNAL_SAMPLES)
        displayOffsetX = SIGNAL_SAMPLES/displayScaleX-FFT_LENGTH;
        GRAPH_SCALE_SetFactor(appGlobals.fftGraphScale,  scaleFactor*displayScaleX);
        GRAPH_SCALE_SetOff(appGlobals.fftGraphScale, -1*displayOffsetX/displayScaleX);
    }
}
```

Listing 6. Obliczanie FFT jest wykonywane wewnątrz zadania FFT_Task
```
for(idx=0; idx<SIGNAL_SAMPLES; idx++)
    appGlobals.fftInput[idx] = appGlobals.dmaBuffer[idx*2];
arm_rfft_fast_f32(&fftInit, appGlobals.fftInput, appGlobals.fftOutput, 0);
arm_abs_f32(appGlobals.fftOutput, appGlobals.fftOutput, SIGNAL_SAMPLES);
```

Listing 7. Funkcja scaleAxisFloat
```
static void scaleAxisXFloat(float32_t* tab, uint32_t len, uint32_t scale) {
  uint32_t index;
  for (index = 0; index < len/scale; index++)
    arm_mean_f32(tab + (index * scale), scale, tab+index);
}
```

Listing 8. Funkcja scaleAxisYFloat
```
static void scaleAxisYFloat(float32_t* tab, uint32_t len) {
  float32_t min;
  float32_t max;
  uint32_t index;

  arm_max_f32(tab, len, &max, &index);
  arm_min_f32(tab, len, &min, &index);

  for (index = 0; index < len; index++)
    tab[index] = (tab[index]-min)*GRAPH_RANGE_Y/(max-min)+GRAPH_OFFSET_Y;
}
```

podstawowych operacjach matematycznych, jak i przy podstawowym przetwarzaniu sygnałów. Biblioteka została skompilowana w kilku różnych konfiguracjach:

- **libarm_cortexM7lfdp_math.a** – little endian, zmiennoprzecinkowa podwójnej precyzji,
- **ibarm_cortexM7lfsp_math.a** – little endian, zmiennoprzecinkowa pojedynczej precyzji,
- **libarm_cortexM7l_math.a** – little endian, stałoprzecinkowa.

Pliki można znaleźć w katalogu *Drivers/CMSIS/Lib/GCC* pakietu STM32Cube dla Cortex M7. Po dodaniu biblioteki do projektu należy pamiętać o dodaniu symbolu ARM_MATH_CM7 w konfiguracji.

Biblioteka zawiera szereg podstawowych funkcji matematycznych działających na całych tablicach, co znacząco ułatwia implementację bardziej złożonych algorytmów. Listę wszystkich funkcji wraz z dokumentacja można znaleźć na stronie *http://goo.gl/a3WFDq*. W przykładzie funkcje biblioteczne zostały użyte do dwóch celów – skalowania wykresów oraz obliczenia FFT.

Obliczanie FFT jest wykonywane wewnątrz zadania *FFT_Task*, po otrzymaniu notyfikacji o gotowości danych w buforze DMA – **listing 6**. W pierwszym kroku wartości próbek sygnału są kopiowane do tablicy typu float32_t. Wybierana jest co druga próbka, ponieważ w tablicy appGlobals.dmaBuffer znajdują się dane z obu kanałów. Obliczenia wykonywane są tylko dla pierwszego z nich – drugi jest ignorowany.

Skopiowane dane przekazywane są do funkcji *arm_rfft_fast_f32* obliczającej transformatę. Pierwszym jej argumentem jest struktura przechowująca parametry transformaty, utworzona na początku zadania *FFT_Taks*:

```
arm_rfft_fast_instance_f32 fftInit;
arm_rfft_fast_init_f32(&fftInit,
SIGNAL_SAMPLES);
```

Jedynym parametrem, który należy podać przy jej tworzeniu jest liczba punktów, z których będzie obliczane FFT. Kolejne argumenty funkcji *arm_rfft_fast_f32*, to tablica zawierająca próbki sygnału i tablica przeznaczona na wynik obliczeń, również typu *float32_t*. Na końcu znajduje się flaga mówiąca o tym, czy ma być obliczana transformata odwrotna.

Aby otrzymać widmo amplitudowe, należy obliczyć moduł otrzymanego wyniku. Służy do tego funkcja *arm_abs_f32*, pobierająca tablice wejściową, wyjściową oraz ich długość. W przykładzie wyniki są zapisywane w tym samym miejscu, co dane wejściowe, dlatego pierwsze dwa argumenty są takie same.

Poza obliczeniem FFT, biblioteka CMSIS DSP została także użyta do skalowania. Oś X jest modyfikowana, odpowiednio do ustawionego współczynnika skali, w funkcji *scaleAxisFloat* pokazanej na **listingu 7**. Oblicza ona średnią arytmetyczną punktów w oknie o długości scale i wpisuje je do kolejnych elementów tablicy wejściowej. Okno jest za każdym razem przesuwane dokładnie o swoja długość. W ten sposób realizowane jest przybliżanie o oddalanie wykresu na ekranie. Uśrednione wartości są następnie skalowane w osi Y tak, aby wypełniały całą wysokość pola wykresu. Jest to realizowane przez funkcję *scaleAxisYFloat* (**listing 8**). Funkcja wyszukuje najpierw wartości minimalnej i maksymalnej w tablicy danych za pomocą funkcji *arm_max_f32* oraz *arm_min_f32*. Obie funkcje maja identyczną listę argumentów: tablica wejściowa, jej długość, wskaźnik, pod który zostanie wpisana szukana wartość oraz wskaźnik gdzie zostanie zapisany jej indeks w tablicy. Po znalezieniu wartości min i max, cała tablica jest skalowana, tak aby wartości mieściły się w nowym przedziale *<GRAPH_OFFSET_Y; GRAPH_OFFSET_Y+GRAPH_RANGE_Y>*, wynikającym z ograniczenia pola wykresu.

Te same operacje uśredniania i skalowania są wykonywane w zadaniu *Signal_Task*, jednak tam używane są wersje funkcji bibliotecznych działające na liczbach całkowitych.

mtouch

Ostatnim opisywanym elementem projektu jest moduł do rozpoznawania gestów wykonanych na panelu dotykowym. Udostępnia on dwie funkcje:

```
void MTOUCH_AddTouchData(MTOUCH_
TouchData_p touchData);
void MTOUCH_GetGesture(MTOUCH_
GestureData_p gestureData);
```

Pierwsza z nich powinna być wywoływana cyklicznie, ponieważ za jej pośrednictwem moduł dostaje informacje o aktualnym stanie panelu dotykowego przekazywanego w strukturze:

```
typedef struct {
  uint8_t points;
  uint16_t x[2];
```

```
  uint16_t y[2];
} MTOUCH_TouchData_s;
```

Zawiera ona liczbę punktów (obsługiwane są zero, jeden, lub dwa) oraz ich współrzędne. Druga z powyższych funkcji oblicza gest wykryty na podstawie ostatnio przekazanych danych dotyczących dotyku. Gest opisany jest przez strukturę:

```
typedef struct {
  MTOUCH_GestureID gesture;
  uint16_t origin_x;
  uint16_t origin_y;
} MTOUCH_GestureData_s;
```

Pierwszym polem struktury jest typ gestu:

```
typedef enum {
    NONE,
    TOUCH,
    MOVE_UP,
    MOVE_DOWN,
    MOVE_LEFT,
    MOVE_RIGHT,
    ZOOM_IN_X,
    ZOOM_OUT_X,
    ZOOM_IN_Y,
    ZOOM_OUT_Y,
} MTOUCH_GestureID;
```

Kolejne dwa pola to współrzędne określające położenie gestu na panelu. Jest to konieczne, aby aplikacja mogła zdecydować, do którego komponentu interfejsu graficznego się on odnosi.

Gest jest wykrywany na podstawie dwóch ostatnio przekazanych struktur *MTOUCH_TouchData_s*. Najpierw sprawdzane jest czy liczba punktów zmieniła się, czy pozostała jednakowa. W pierwszym przypadku zapamiętywane są współrzędne początkowe gestu – punkt dotyku, lub średnia w przypadku dwóch punktów. Gest oznaczany jest jako *TOUCH*. Jeżeli natomiast liczba punktów pozostała bez zmian obliczany jest kierunek największego przesunięcia gestów jednopunktowych (*MOVE_RIGHT*, *MOVE_LEFT*, *MOVE_UP*, *MOVE_DOWN*) lub zmiana odległości między punktami dla gestów dwupunktowych (*ZOOM_IN_X*, *ZOOM_OUT_X*, *ZOOM_IN_Y*, *ZOOM_OUT_Y*). Kod modułu znajduje się w plikach *mtouch.c* oraz *mtouch.h*.

Krzysztof Chojnowski

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 77642, pass: 3220ppmm