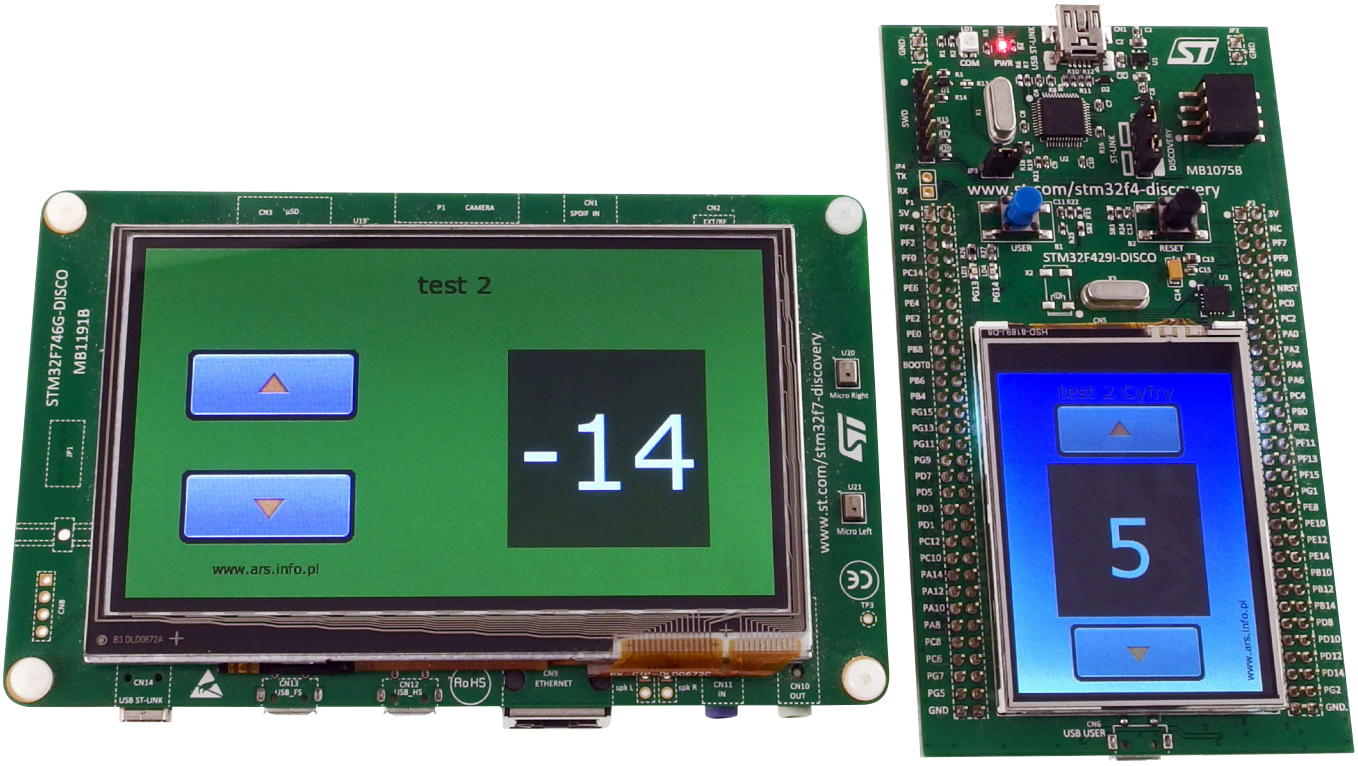


TouchGFX środowisko graficzne dla kontrolerów STM32

System procesorowy z wbudowanym ekranem, potrzebuje do obsługi grafiki programistycznego interfejsu użytkownika zwanego skrótowo GUI. W przypadku projektów z zaawansowanymi funkcjami, zamiast samodzielnie tworzyć taki interfejs od podstaw, wygodniej jest skorzystać z gotowych rozwiązań. TouchGFX to darmowy, także dla zastosowań komercyjnych, interfejs graficzny dla urządzeń z kontrolerami z rodziny STM32. Artykuł poświęcony jest podstawom tworzenia GUI przy pomocy TouchGFX w powiązaniu z innymi narzędziami dla kontrolerów STM32.

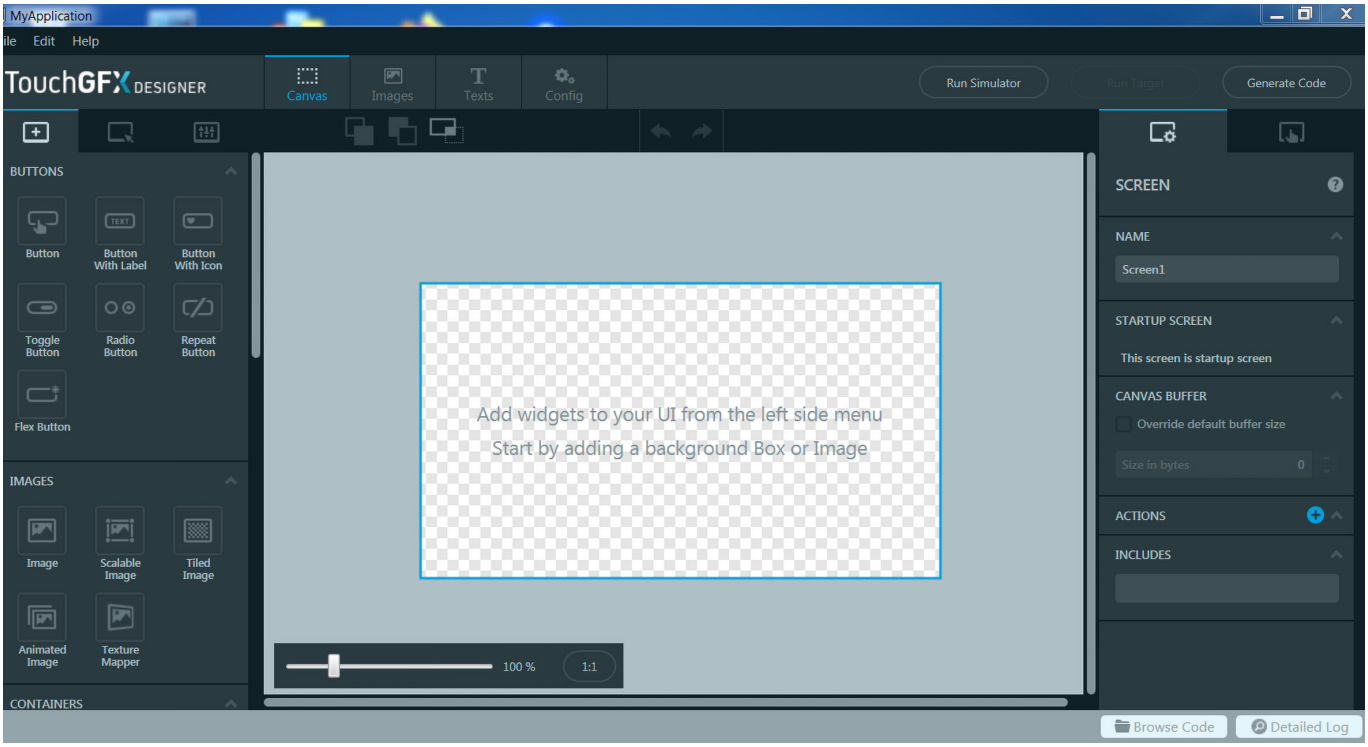
W połowie 2018 roku firma ST przejęła duńską firmę Draupner Graphics producenta TouchGFX, wysokiej jakości biblioteki graficznej przeznaczonej dla systemów mikroprocesorowych. Tym samym dla 32-bitowych kontrolerów z rodziny STM32 stało się ogólnie dostępne drugie obok STemWin środowisko GUI. Najważniejsze cechy biblioteki TouchGFX:

- jest przystosowana do współpracy z różnymi kontrolerami rodziny STM32, wyświetlaczami TFT o różnej rozdzielczości, wyposażonymi w kontrolery dotyku dla ekranów rezystancyjnych i pojemnościowych,
- obsługuje przeźroczystość, animacje, różne alfabety narodowe w tym polski,
- jest napisana w języku C++. Przy pomocy prostych procedur pośredniczących, biblioteka może także współpracować z funkcjami w czystym C,
- biblioteka pracuje pod nadzorem systemu operacyjnego czasu rzeczywistego,
- częścią biblioteki jest narzędzie TouchGFX Designer. Metodą „przeciągnij i upuść” pozwala tworzyć GUI z podglądem na ekranie jego rzeczywistego wyglądu. Designer umożliwia symulację działania GUI a na koniec generuje kod obsługi grafiki, umieszczany w programie użytkownika.

Co ważne, pakiet TouchGFX jest już w znacznym stopniu zintegrowany z innymi narzędziami używanymi do pisania oprogramowania dla kontrolerów ST.

Integracja TouchGFX z narzędziami ST

Dla większości użytkowników, kontakt z kontrolerami STM32 rozpoczyna się od płyty ewaluacyjnej na której można testować własne programy. Niektóre z nich, z serii DISCOVERY przystosowane są do współpracy z ekranem graficznym. Płyty wraz z oprogramowaniem do generacji kodu STM32CubeMX czy pakietem programistycznym STM32CubeIDE stanowią jednolite środowisko projektowe. Rzeczą naturalną była integracja TouchGFX z pozostałymi elementami



Rysunek 1. Wygląd TouchGFX Designer-a

środowiska. Korzystając z STM32CubeMX i TouchGFX da się wygenerować działający kod z zaprojektowanym GUI dla wyświetlacza, jednak trzeba się przy tym trochę naklikać.

Należy zacząć od zainstalowania TouchGFX Designera, który będzie służył do projektowania i testowania budowanego GUI oraz wygeneruje kod do osadzenia w programie głównym. Pakiet instalacyjny można pobrać z (link 1). Na dzień pisania artykułu do wyboru były dwie wersje: 4.10.0 i 4.12.3 różniące się nieco możliwościami, można pobrać i zainstalować równolegle obie. Po rozpakowaniu i uruchomieniu pliku nastąpi standardowa procedura instalacyjna. Najwygodniej zainstalować oprogramowanie TouchGFXDesigner-a zgodnie z sugestią w katalogu głównym twardego dysku. Warto także odświeżyć instalację STM32CubeMX do najnowszej wersji co najmniej 5.4.0.

Po zainstalowaniu TouchGFXDesigner nadaje się od razu do pracy. Po jego uruchomieniu można wybierać płytę na której pracujemy, rodzaj wyświetlacza, projektować interfejs graficzny, przetestować go na symulatorze, a nawet zaprogramować posiadaną płytę, co pozwoli obejrzeć na docelowym wyświetlaczu przygotowane GUI. Jednak integracja kodu odpowiedzialnego za GUI z pozostałą częścią oprogramowania np. obsługą urządzeń zewnętrznych i wewnętrznych interfejsów kontrolera jest łatwiejsza, gdy działanie TouchGFXDesigner-a zostaje wywołane podczas tworzenia projektu przy pomocy STM32CubeMX.

Przykład projektu dla płyty STM32F746G-DISCO

Teraz krok po kroku pokażę jak stworzyć program, którego działanie polega na przełączaniu pomiędzy dwoma ekranami. Program jest prosty i będzie funkcjonował bez konieczności dopisania własnych procedur, wszystko zostanie wyklikane.

Zaczynamy od otwarcia STM32CubeMX i wskazania płyty dla której ma powstać nowy projekt. W tym przypadku będzie to STM32F746G-DISCO. Należy zgodzić się na inicjalizację z predefiniowanymi ustawieniami. Po utworzeniu nowego projektu należy zablokować interfejs eternetowy klikając kolejno:

Pinout & Configuration → Connectivity → ETH → Mode: Disable

Następnie należy przejść do zakładki Project Manager i podać wybraną nazwę dla projektu oraz miejsce gdzie ma być zapisany na dysku. Dalej trzeba określić środowisko programistyczne dla którego ma zostać wygenerowany kod. W przykładzie będzie to SW4STM32 więc kolejne ustawienia na tej zakładce będą wyglądały tak:

Project Manager → Application Structure: Advanced

Project Manager → Tolchain/IDE: SW4STM32, Generate under root: v

Project Manager → Firmvare Package Name and Version: STM32Cube FW_F7 V1.15.0

Wybieramy opcję GENERATE CODE, a po zakończeniu zapisu wygenerowanego kodu można przejść do działań związanych z wyświetlaczem i GUI. Zaznaczamy:

Pinout & Configuration → Middleware → GRAPHICS a następnie Graphics Framework: TouchGFX, Display Interface: Display Parallel Interface using LTDC.

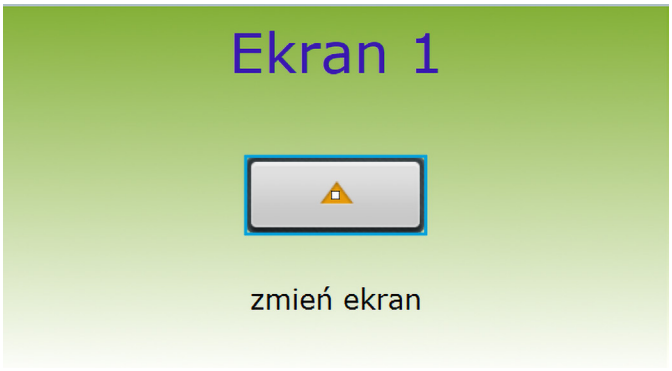
Trzeba określić, do którego portu jest podłączone wyprowadzenie XRES wyświetlacza, dla płyty STM32F746G-DISCO będzie to PI.12. Klikamy w nastepującej kolejności:

Configuration → Platform Settings → LCD Reset Pin XRES: PI12[LCD_DISP[RK043FN048H-CT672B_DISP]].

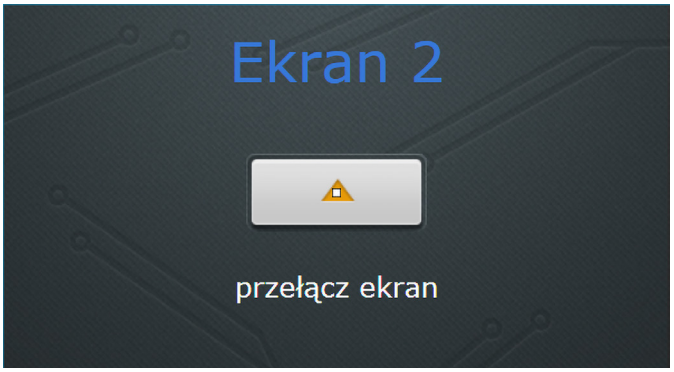
Żeby możliwe było uruchamianie TouchGFX Designer-a z STM32CubeMX, trzeba podać ścieżkę dostępu do pliku wykonywalnego TouchGFX Designer-4.12.3.exe lub TouchGFX Designer-4.10.0.exe jeżeli będziemy korzystali ze starszej wersji:

Configuration → TouchGFX → Location: c:\TouchGFX\4.12.3\designer\TouchGFXDesigner-4.12.3.exe.

W zakładce Configuration → TouchGFX uaktywnia się klawisz Execute, którego naciśnięcie spowoduje przejście do TouchGFX Designer-a. Po otwarciu, gotowy do pracy z nowym projektem TouchGFX Designer wygląda jak na **rysunku 1**. Centralny pulpit służy do układania graficznych komponentów tworzonej strony naszego GUI. Z lewej strony otwarta jest zakładka z dostępnymi komponentami (widgetami). Z prawej strony widać zakładkę służącą do ustawiania parametrów użytych komponentów. W momencie wywołania TouchGFX Designer wygenerował wstępnie pliki, które dodał do katalogów stworzonych wcześniej przez STM32CubeMX. Jeden z wygenerowanych plików: nazwa_projektu\TouchGFX\nazwa_projektu.touchgfx (nazwa_projektu należy zastąpić podaną w STM32CubeMX nazwą generowanego projektu), należy ręcznie wyedytować i zmienić linię



Rysunek 2. Wygląd ekranu 1 z przykładu pierwszego



Rysunek 3. Wygląd ekranu 2 z przykładu pierwszego

```
"PostGenerateCommand": "touchgfx update_project
--project-file=../.cproject ",
```

na:

```
"PostGenerateCommand": " echo test ".
```

Tworzenie projektu nowego ekranu zazwyczaj rozpoczyna się od ustawienia tła, którym może być obrazek z pliku o formacie .bmp lub .png. Obrazek trzeba wcześniej przygotować w dowolnym programie graficznym. Wymiary obrazka powinny być takie same jak pozioma i pionowa rozdzielczość wyświetlacza płyty, w tym przypadku 480×272 pikseli. Po wybraniu widgetu image, należy go ułożyć w lewym górnym rogu pulpitu. W zakładce parametrów otwieramy listę STYLE wybierając opcję No Style. Poniżej w opcjach IMAGE klikamy na '+' żeby wskazać na katalog z wykonanym plikiem tła i wczytać go do listy używanych w projekcie GUI plików graficznych. Wskazujemy na wczytany obrazek, który po kliknięciu powinien pojawić się na pulpicie jako tło ekranu.

W podobny sposób jak widget image, umieszczamy na pulpicie widget Text Area. Na zakładce parametrów w polu Text wpisujemy „Ekran 1". Zmieniając parametry dobieramy wielkość czcionki, kolor i położenie. Umieszczamy na pulpicie widget Button. W opcjach STYLE można wybrać jeden z predefiniowanych kształtów dla przycisku albo, tak jak w przypadku tła, przygotować obrazki z wyglądem przycisku naciśniętego i zwolnionego a następnie na listach Pressed Image i Released Image wskazać położenie obrazków. W momencie naciskania i zwalniania klawisza umieszczonego na ekranie, obrazki będą automatycznie zastępowane. W opcjach NAME zmieniamy nazwę przycisku na „button1". Przygotowany ekran powinien wyglądać podobnie jak na **rysunku 2**.

Przechodzimy do zakładki Screens (z lewej strony pulpitu). Naciskając na '+' inicjujemy drugi ekran, który wypełniamy widgetami w podobny sposób jak poprzedni. Na **rysunku 3** pokazano przykładowy wygląd drugiego ekranu.

Na koniec zaprojektujemy zdarzenia jakie ma wywołać naciśnięcie button1 na ekranie 1 i button2 na ekranie 2. W tym celu należy otworzyć zakładkę INTERACTIONS z prawej obok zakładki gdzie edytuje się parametry komponentów. Klikamy na przycisk Add Interaction. Z kolejnych list wybieramy:

Trigger: Button is clicked

Choose clicked source: button1

Action: Change screen a poniżej wskazać jako ekran docelowy Screen2

Ponownie klikamy na przycisk Add Interaction i projektujemy zdarzenie dla button2. Jako ekran docelowy wskazujemy Screen1.

Żeby sprawdzić czy wszystko zostało poprawnie zaprojektowane uruchamiamy symulator przyciskiem Run Simulator. Na ekranie komputera powinien wyświetlić się zaprojektowany ekran 1 a po wskazaniu kursorem przycisku i kliknięciu lewym przyciskiem myszy powinien na jego miejscu pojawić się ekran 2. Jeżeli wszystko działa zgodnie z oczekiwaniem klikamy przycisk Generate Code a po pojawieniu się komunikatu o pomyślnym utworzeniu kodu, zamykamy TouchGFX Designer.

W STM32CubeMX wybieramy opcję GENERATE CODE i przechodzimy do edytora naszego środowiska programistycznego – w tym opisie będzie to SW4STM32. Teraz trzeba dostosować wygenerowany przez oba narzędzia kod tak, aby poprawnie pracował a kompilacja odbywała się bez błędów. Ponieważ TouchGFX Designer dołącza w generowanym kodzie odwołania do symulatora trzeba je wyłączyć z kompilacji. W SW4STM32 wskazuje się plik lub katalog i wywołując opcję:

Properties → C/C++ Build → Settings → Exclude resource from build: [v] zaznacza opcję wyłączenia.

Podaję lokalizację katalogów i plików do wyłączenia:

TouchGFX: simulator

TouchGFX → generated: simulator

Midlewares → ST → Touchgfx → touchgfx → framework → source → platform → hal: simulator

Midlewares → ST → Touchgfx → touchgfx → framework → include → platform → hal: simulator

oraz:

Midlewares → ST → Touchgfx → touchgfx → framework → source → platform → driver → touch: SDL2TouchControler.cpp

Midlewares → ST → Touchgfx: touchgfx_backup

Po usunięciu w taki sposób wszystkich błędów kompilacji, płyta powinna dać się zaprogramować i wyświetlony zostanie ekran 1. Jednak klawisz na ekranie nie będzie działał bo w naszym programie brakuje podłączenia do sterowników ekranu dotykowego. Aby uruchomić dotyk będziemy potrzebowali plików z firmware package STM32Cube_FW_F7_V1.15.0. Spakowany plik można pobrać stąd (link 2). Dla zachowania porządku pliki będą kopiowane grupami do utworzonych w drzewie projektu podfolderów, które zostaną dodane do folderu

Project Explorer → nazwa_projektu → Drivers.

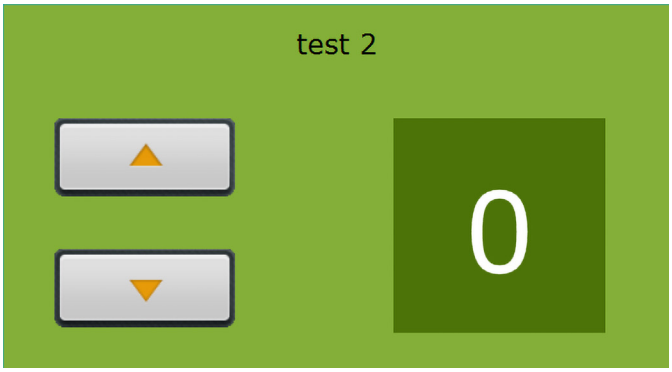
Z folderu z firmware package ...\Drivers\BSP\STM32746G-Discovery\

Listing 1. Zmiany w pliku: STM32F7TouchController.cpp

```
UZUPEŁNIĆ:
#include „STM32F7TouchController.hpp"
/* USER CODE BEGIN BSP user includes */
#include „stm32746g_discovery_ts.h"
/* USER CODE END BSP user includes */


USUNĄĆ BLOK KOMENTARZA:
void STM32F7TouchController::init(){
    /* USER CODE BEGIN F4TouchController_init */
    /* Add code for touch controller Initialization */
    BSP_TS_Init(LCD_GetXSize(), LCD_GetYSize());
    /* USER CODE END F4TouchController_init */
}

bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y){
    /* USER CODE BEGIN  F4TouchController_sampleTouch  */
    TS_StateTypeDef state = { 0 };
    BSP_TS_GetState(&state);
    if (state.touchDetected){
        x = state.touchX[0];
        y = state.touchY[0];
        return true;
    }
    return false;
    /* USER CODE END F4TouchController_sampleTouch  */
}
```



Rysunek 4. Wygląd ekranu z przykładu drugiego

kopiuj do utworzonego w drzewie projektu ... Drivers → BSP → src pliki:

stm32746g_discovery_sdram.c
stm32746g_discovery_ts.c
stm32746g_discovery.c

Z folderu z firmware package ...\Drivers\BSP\STM32746G-Discovery\

kopiuj do utworzonego w drzewie projektu ... Drivers → BSP → inc pliki:

stm32746g_discovery_sdram.h
stm32746g_discovery_ts.h
stm32746g_discovery.h

Z folderu z firmware package ...\Drivers\BSP\Components\ft5336\

kopiuj do utworzonego w drzewie projektu ... Drivers → Components → ft5336 pliki:

ft5336.c
ft5336.h

Z folderu z firmware package ...\Drivers\BSP\Components\Common\

kopiuj do utworzonego w drzewie projektu ... Drivers → Components → Common plik:

ts.h

Należy podać ścieżki dostępu do utworzonych podkatalogów dla obu kompilatorów MCU GCC Compiler i MCU G++ Compiler:

Drivers\BSP
Drivers\BSP\inc
Drivers\Components
Drivers\Components\Common
Drivers\Components\ft5336

Na koniec należy podłączyć pliki sterowników do plików GUI. W tym celu należy otworzyć w edytorze naszego IDE plik TouchGFX → target: STM32F7TouchController.cpp i wykonać zmiany zaznaczone na żółto w **listingu 1**. Od tego momentu po kompilacji i zaprogramowaniu płyty, przyciski na wyświetlaczu powinny działać w taki sam sposób jak podczas testów na symulatorze TouchGFX Designer-a.

Przykład projektu z dodanym kodem użytkownika

Poprzedni przykład był bardzo prosty, ale też niewiele potrafił zrobić. Stwórzmy trochę bardziej skomplikowany projekt, który będzie wymagał dopisania odrobiny dodatkowego kodu użytkownika. W projekcie będzie jeden ekran z dwoma klawiszami. Naciśnięcie klawiszy będzie zwiększało lub zmniejszało stan licznika, którego zawartość zostanie wyświetlana na ekranie.

Projekt ekranu pokazany został na **rysunku 4**. Tło i klawisze wstawiamy w sposób opisany w poprzednim przykładzie. Nadajemy nazwę jednemu z klawiszy „buttonUp" a drugiemu „buttonDown". Okienko do wyświetlania aktualnego stanu licznika stworzone zostało z kolejnego obrazka z ciemnym tłem w kształcie prostokąta 150×150 pikseli. Na nim osadzony został widget Text Area o nazwie „textCounter". W zakładce parametrów usuwamy zaznaczenie opcji Auto-size i powiększamy zajmowany przez niego obszar tak aby wypełniał ciemniejszą podkładkę. W polu Typography wybieramy czcionkę Large Verdana 40 px a opcję Alignment ustawiamy na justowanie do środka. W opcji Text wpisać <d> co oznacza, że widget będzie dokonywał konwersji przypisanej zmiennej na cyfry dziesiętne. Naciskamy przycisk '+' co uaktywni zakładkę WILDCARD1, którą otwieramy. Na zakładce w polu Initial value wpisujemy 0 i zaznaczamy opcję Use wildcard buffer, pozostawiając domyślny rozmiar buforu.

Teraz dostosujemy do naszych potrzeb właściwości typograficzne czcionki użytej do wyświetlania zawartości licznika. Na belce funkcyjnej ponad pulpitem klikamy napis Texts, a potem wybieramy zakładkę Typographies. Powiększamy użytą do wyświetlania licznika czcionkę Large Verdana, zwiększając wymiar Size do 80. W polu Wildcard Ranges wpisujemy 0...9, będzie to zakres znaków dziesiętnych dopuszczonych do wyświetlenia. W polu Wildcard Characters wpisujemy myślnik '–' który będzie służył jako znak liczb ujemnych.

Dodajemy interakcję dla buttonUp uzupełniając kolejne pola:

Trigger: Button is clicked
Choose clicked source: buttonUp
Action: Call new virtual function
Function Name: buttonUpClicked
Interaction Name: InteractionButtonUp

Dodajemy interakcję dla buttonDown uzupełniając pola:

Trigger: Button is clicked
Choose clicked source: buttonDown
Action: Call new virtual function
Function Name: buttonDownClicked
Interaction Name: InteractionButtonDown

Tak jak w wcześniejszym przykładzie generujemy kod, usuwamy wywołujące błędy kompilacji odwołania do symulatora, dodajemy sterowniki ekranu dotykowego. Teraz pozostało dopisać fragment kodu powodującego zmianę stanu wyświetlanego licznika po naciśnięciu klawiszy. TouchGFX Designer wykonał już część pracy, deklarując wirtualne funkcje do obsługi naciśnięć klawiszy, nadając im nazwy które wpisaliśmy wcześniej w interakcjach: buttonUpClicked i buttonDownClicked. Można to podejrzeć w pliku TouchGFX → generated → gui_generated → src → nazwa ekranu_screen: Nazwa ekranuViewBase.cpp
gdzie utworzone zostały puste funkcje. Można je przysłonić własnymi funkcjami użytkownika, które umieścimy, w także już wygenerowanych i przeznaczonych do tego plikach. W pliku

TouchGFX → gui → include → gui → nazwa ekranu_screen: nazwa ekranuView.hpp

w sekcji public deklarujemy nazwy dodawanych funkcji, a w sekcji protected deklarujemy zmienną, w której będzie przechowywana zawartość licznika:

```
public:
 virtual void buttonUpClicked();
 virtual void buttonDownClicked();
protected:
```

Listing 2. Definicje funkcji do umieszczenia w pliku: nazwa ekranuView.cpp

```
void Screen1View::buttonUpClicked(){
    counter++;
    if (counter>16) counter =-16;
    Unicode::snprintf(textCounterBuffer, TEXTCOUNTER_SIZE, „%d", counter);
    //Invalidate text area, which will result in it being redrawn in next tick
    textCounter.invalidate();
}

void Screen1View::buttonDownClicked(){
    counter--;
    if (counter<-16) counter =16;
    Unicode::snprintf(textCounterBuffer, TEXTCOUNTER_SIZE, „%d", counter);
    //Invalidate text area, which will result in it being redrawn in next tick
    textCounter.invalidate();
}
```

int counter;

W pliku TouchGFX → gui → src → gui → nazwa ekranu_screen: nazwa ekranuView.cpp umieszczamy definicje naszych funkcji tak jak na **listingu 2**, które będą wywoływane po naciśnięciu odpowiednich przycisków. Wyświetlana zawartość licznika będzie się zmieniała w zakresie od –16 do 16.

Projekty dla płyty STM32F429I-DISCO

W spsób podobny do opisanego, można wygenerować projekt z użyciem biblioteki TouchGFX dla innych płyt rozwojowych firmy ST. Różnice będą polegały na zmianie sposobu sterowania wyświetlaczy i zastosowanych kontrolerów dotyku. Na przykładzie płyty STM32F429I-DISCO pokażę co trzeba zmienić.

Po inicjacji nowego projektu w STM32CubeMX dla STM32F429I-DISCO należy:

ustawić częstotliwość zegara wyświetlacza na 6 MHz:

Clock Configuration → LCD-TFT clocks.

Włączyć SPI5 w trybie Half Duplex Master. Po wyborze środowiska graficznego TouchGFX i włączeniu wyświetlacza należy w opcjach Configuration → Platform Settings ustawić:

Chip Select: PC2[CSX[LCD-RGB_CSX]]

WRX High: PD13[WRX_DCX[LCD-RGB_ WRX_DCX]]

RDX High: PD12[RDX[LCD_RGB_RDX]]

SPI_PIN: SPI:Half-Duplex Master SPI5

Po przygotowaniu projektu ekranów i wygenerowaniu plików, przechodzimy do edytora w używanym środowisku programistycznym. Tak jak w poprzednich przykładach wyłączamy z kompilacji katalogi i pliki powodujące wystąpienie błędów, np. odwołania do symulatora. Na koniec instalujemy pliki do obsługi kontrolera dotyku. Potrzebne pliki znajdziemy w firmware package STM32Cube_FW_F4_V1.24.0 do pobrania stąd (link 3). W drzewie katalogów projektu dodajemy podkatalog DRV_F429I:

Drivers → BSP → DRV_F429I

Z folderu z firmware package ...\STM32Cube_FW_F4_V1.24.0\Drivers\BSP\STM32F429I-Discovery\ kopiujemy pliki:

stm32f429i_discovery_io.c

stm32f429i_discovery_io.h

stm32f429i_discovery_ts.c

Listing 3. Zmiany jakie należy wykonać w pliku: STM32F4TouchController.cpp

```
DOPISAĆ:
/* USER CODE BEGIN BSP user includes */
#include <stm32f429i_discovery_ts.h>
/* USER CODE END BSP user includes */

ODKOMENTOWAĆ:
void STM32F4TouchController::init(){
    /* USER CODE BEGIN F4TouchController_init */
    /* Add code for touch controller Initialization*/
    BSP_TS_Init(LCD_GetXSize(), LCD_GetYSize());
    /* USER CODE END  F4TouchController_init  */
}

bool STM32F4TouchController::sampleTouch(int32_t& x, int32_t& y){
    /* USER CODE BEGIN  F4TouchController_sampleTouch  */
    TS_StateTypeDef state;
    BSP_TS_GetState(&state);
    if (state.TouchDetected){
        x = state.X;        //state.x na state.X
        y = state.Y;        //state.y na state.Y
        return true;
    }
    return false;
 /* USER CODE END F4TouchController_sampleTouch */
}
```

stm32f429i_discovery_ts.h

stm32f429i_discovery.c

stm32f429i_discovery.h

do utworzonego podkatalogu DRV_F429I.

Dodajemy ścieżkę dostępu do utworzonego podkatalogu dla obu kompilatorów MCU GCC Compiler i MCU G++ Compiler. W pliku TouchGFX → target: STM32F4TouchController.cpp wykonujemy zmiany takie jak zaznaczono na **listingu 3**. W pliku Drivers → BSP → DRV_F429I: stm32f429i_discovery.c komentujemy całą definicję funkcji:

```
//void LCD_Delay(uint32_t Delay)
//{
//  HAL_Delay(Delay);
//}
```

Ryszard Szymaniak, EP

Link 1: ***http://bit.ly/2uuZaEK***
Link 2: ***http://bit.ly/2vgeMwf***
Link 3: ***http://bit.ly/2w1miet***