Rysunek 34: Zdjęcie płytki rozwojowej STM32F746G-DISCO

4 Obsługa wyświetlacza graficznego LCD (panelu dotykowego). Wykorzystanie jednostki zmiennopozycyjnej do przetwarzania sygnałów.

Celem tego ćwiczenia jest zapoznanie się z nową platformą z rodziny STM32. Zadaniem studenta jest zapoznać się z obsługą nowych elementów służących do komunikacji z użytkownikiem, takich jak kolorowy wyświetlacz graficzny, ekran dotykowy. Jednocześnie student jest zobowiązany rozsądnie zarządzić czasem procesora, a co za tym idzie priorytetami przerwań. Efektem końcowym tego ćwiczenia powinien być program, który jest wygodny w użyciu (tj. taki, który będzie działał szybko, nie będzie zatrzymywał pracy i będzie stale responsywny).

STM32F746G-DISCO Przejście z płytki rozwojowej ZL27ARM na płytkę STM32F746G-DISCO (Rys. 34) nie jest skomplikowanym procesem. Jednak należy pamiętać, ze rodzina mikrokontrolerów STM32F7 i wyższe nie posiadają już dostosowanych do nich standardowych bibliotek peryferiali. Wyparta ona została przez bibliotekę HAL (*Hardware Abstraction Layer*), która w połączeniu z generatorem kodu CubeMX ma za zadanie usprawnić projektowanie i implementację programów na te mikrokontrolery. W związku z tym, że nauka i oswojenie się z nową biblioteką byłoby czasochłonne, dalsze ćwiczenia będą w dużej mierze oparte o przygotowany wcześniej szablon, tak, aby skupić się na poszerzaniu umiejętności, a nie ich pogłębianiu – ostatcznym celem jest przecież implementacja regulatora, a nie nauka programowania mikrokontrolera.

Wspomniana płytka wyposażona jest w wyświetlacz LCD-TFT o przekątnej 4,3 cala, rozdzielczości 480x272 z pojemnościowym ekranem dotykowym. Złącze zgodne z Arduino Uno V3 zamieszczone po drugiej stronie przydatne będzie do zamontowania na następnych ćwiczeniach dodatkowych przetworników cyfrowo-analogowych. Oryginalnie mikrokontorler zamontowany na tej płytce wyposażony jest w dwa takie przetworniki, lecz nie są one przygotowane do użytku ogólnego – są wykorzystywane do generacji

dźwięku stereo. Posiada on jednak przetwornik analogowo-cyfrowy, który jest jednocześnie podłączony do pinow złącza Arduino.

Do ciekawszych właściwości tego mikrokontrolera, a właściwie jego rdzenia – Cortex®-M7 – należy możliwość obliczeń w arytmetyce zmiennopozycyjnej (pojedynczej precyzji). Jest to znaczące ułatwienie i usprawnienie programów, które tę arytmetykę wykorzystują. W poprzednim mikrokontrolerze korzystanie z dobrodziejstwa arytmetyki na liczbach o zmiennym przecinku odbywało się poprzez programową implementację takich operacji (zajmował się tym dyskretnie kompilator). Teraz do tego posiadamy sprzętowe narzędzie, co pozwoli na sprawniejsze wykonywanie przyszłego kodu regulatora.

Kolejnym przydanym aspektem tej płytki jest dołączony na płytce programator ST-Link. Zmniejsza to wyraźnie ilość sprzętu potrzebnego do programowania takiego mikrokontrolera. Dodatkowo ST-Link jest również wykorzystany w roli debugera, co powinno być dla użytkownika końcowego (niemal) identyczne w działaniu w stosunku do np. programatora/debuggera J-Link.

Biblioteka HAL W dotychczasowych projektach używana była Standardowa Biblioteka Peryferiali. Dla mikrokontrolerów z rodziny STM32F7 nie została ona przygotowana. W jej miejsce pojawia się biblioteka HAL (*Hardware Abstraction Layer*). Przyczyną tej zmiany jest próba dalszego ułatwiania użytkownikom (programistom) generacji kodu. Słowo „generacja" nie zostało użyte przypadkowo – wraz z biblioteką HAL ST wytworzył oprogramowanie służące do automatycznej generacji kodu z pełną konfiguracją wszystkich potrzebnych peryferiali z odpowiednimi ustawieniami. Narzędziem tym jest STM32CubeMX. W tym ćwiczeniu zostanie wykorzystany już wygenerowany kod, dodatkowo jeszcze oczyszczony z powtarzających się fragmentów.

Warto mieć na uwadze, że wraz ze wprowadzaniem kolejnych uproszczeń i ujednoliceń w kodzie, twórcy i użytkownicy takiego kodu narażają się na pojawiające się redundancje. Tak jak biblioteka SPL nie jednokrotnie powtarza te same operacje na rejestrach, tak i biblioteka HAL generuje powtórzenia wywołań funkcji SPL. Tutaj należy zaznaczyć, że wiele funkcji ze Standardowej Biblioteki Peryferiali wchodzi w skład biblioteki HAL. Co więcej nierzadko biblioteka HAL wprowadza jedynie nową nazwę funkcji znanej z SPL. Najważniejszą jednak różnicą między wykorzystaniem SPL a HAL jest struktura projektu. W dotychczasowych projektach wszystko było konfigurowane wprost – HAL ma na celu oddzielenie warstwy sprzętowej od aplikacji, bibliotek i stosów. Pozwala to na łatwiejszy rozwój aplikacji bez uzależnienia od konkretnej płytki wykorzystanej w projekcie.

Biblioteka HAL jak i SPL posiadają swoich zwolenników i przeciwników – tutaj nie będą omawiane szczegółowo cechy obu podejść. Warto jednak mieć na uwadze, że o ile projekt jest w fazie testów – użycie biblioteki HAL jest jak najbardziej wskazane. W przypadku jednak gdy produkt jest gotowy i zaczyna się optymalizacja wydajności – wtedy warto posprzątać trochę kod, który został wygenerowany automatycznie.

Omówienie funkcji biblioteki HAL zostanie ograniczone do minimum. W tym oraz kolejnych ćwiczeniach oczekuje się od studenta umiejętności dobierania parametrów konfiguracji (o ile to konieczne) na podstawie komentarzy w kodzie źródłowym bibliotek oraz własnej domyślności. Wiele nazw wciąż jest intuicyjnych (w szczególności stałych), a modyfikacja parametrów konfiguracji i uzupełnianie jej pojedynczych pól jest zadaniem, które było realizowane na każdym z dotychczasowych ćwiczeń – zakłada się więc, że umiejętność ta została opanowana przez studenta.

Wyświetlacz LCD z ekranem dotykowym W ramach tego ćwiczenia wykorzystana zostanie biblioteka do obsługi wyświetlacza LCD oraz ekranu dotykowego dostarczona wraz z jednym z przykładów przez ST. Zagłębienie się w kod tej biblioteki nie jest celem tego ćwiczenia – należy skupić się na jej jak najlepszym wykorzystaniu. W tym ćwiczeniu będzie wykorzystanych kilka kluczowych funkcji do rysowania na wyświetlaczu:

- `BSP_LCD_SetTextColor` – funkcja do ustawienia koloru tekstu i koloru rysowanych pixeli,
- `BSP_LCD_SetBackColor` – funkcja do ustawienia koloru tła tekstu,
- `BSP_LCD_DisplayStringAt` – funkcja do wyświetlania tekstu w podanym miejscu na wyświetlaczu,
- `BSP_LCD_DrawLine` – funkcja do rysowania linii,
- `BSP_LCD_DrawPixel` – funkcja do rysowania pojedynczego pixela,
- `BSP_LCD_FillRect` – funkcja do rysowania wypełnionego prostokąta (zniechęca się do korzystania z tej funkcji, gdyż bez zapewnienia mechanizmu synchronizacji może nie generować spodziewanych efektów),
- `BSP_LCD_GetXSize` – funkcja zwracająca szerokość ekranu,
- `BSP_LCD_GetYSize` – funkcja zwracająca wysokość ekranu.

Argumenty przyjmowane przez te funkcje są oczywiste – nie będą tutaj objaśniane.

Do obsługi ekranu dotykowego wymagane jest utworzenie specjalnej struktury:

```
TS_StateTypeDef TS_State;
```

której zawartość będzie odświeżana za każdym razem, gdy sobie tego zażyczy użytkownik. Do odświeżenia zawartości użyć należy funkcji `BSP_TS_GetState`, której argumentem jest wskaźnik na instancję wspomnianej struktury. Do przydatnych atrybutów tej struktury należą:

- `touchDetected` – atrybut przechowujący liczbę wykrytych jednoczesnych dotknięć ekranu,
- `touchX` – tablica przechowująca współrzędną poziomą dla każdego z wykrytych jednoczesnych dotknięć ekranu (początek osi znajduje się z lewej strony ekranu),
- `touchY` – tablica przechowująca współrzędną pionową dla każdego z wykrytych jednoczesnych dotknięć ekranu (początek osi znajduje się u góry ekranu),

Ekran dotykowy komunikuje się z użyciem protokołu I^2C i posiada wiele dodatkowych możliwości (jak np. wykrywanie gestów), lecz ewentualne zgłebienie tego tematu pozostawia się zainteresowanym. W związku z uproszczoną obsługą ekranu dotykowego, zamiast oczekiwania na przerwanie sygnalizujące zmianę stanu ekranu, będzie on regularnie odpytywany o stan. Wystarczająco duża częstotliwość sprawdzania powinna pozwolić na równie wygodną obsługę programu, co wykorzystanie przerwań.

Przy obsłudze ekranu należy zwrócić uwagę na współdzielenie zasobów – w tym przypadku będzie to wyświetlacz. Wystąpienie przerwania w momencie gdy wyświetlany jest obraz może spowodować, że zostanie wyświetlone nie to co powinno lub przynajmniej nie tak jak powinno. Warto użyć w tym przypadku semafora lub monitora, które pozwolą uzyskać wyłączny dostęp do wyświetlacza przez tylko jedną funkcję.

Jednostka do obliczeń zmiennoprzecinkowych Jednostka do obliczeń zmiennoprzecinkowych wykorzystywana jest automatycznie przez kompilator, dzięki czemu kod programu korzystający z niej nie różni się niczym od tego, który jej nie używa. W ramach tego ćwiczenia przeprowadzony zostanie test w jaki sposób działa program z użyciem i bez użycia tej jednostki – powinien zostać zaobserwowany wyraźny spadek wydajności mikrokontrolera.

4.1 Przebieg laboratorium

Celem ćwiczenia jest zapoznanie się z nową platformą z rodziny STM32. Student ma za zadanie przede wszystkim zapoznać się z obsługą wyświetlacza i ekranu dotykowego, z jednoczesnym przećwiczeniem umiejętności organizacji priorytetów przerwań i zadań tak, aby zaimplementowany program był wygodny w użyciu. W tym celu należy zaprojektować i wykonać aplikację w taki sposób aby:

- podzielić wyświetlacz na dwie części: lewą, która będzie związana z rysowaniem fraktalu Mandelbrota oraz prawą, gdzie będzie rysowany wykres wartości napięcia na jednym z pinów mikrokontrolera w funkcji czasu,
- każda z części powinna posiadać na górze nagłówek (np. `MANDELBROT` i `INPUT PLOT`),
- każda z części powinna posiadać na dole przycisk modyfikujący obraz wyświetlany w odpowiadającej mu części:
  - przycisk pod fraktalem powinien być bistabilny (tj. jego stan przełącza się z każdym jego naciśnięciem) – jego przełączenie powinno powodować narysowanie fraktalu w kolorach odpowiednio od czarnego do zielonego lub od białego do czerwonego,
  - przycisk pod wykresem powinien być monostabilny i jego wciśnięcie powinno powodować wyczyszczenie wykresu i rozpoczęcie ponownego zbierania danych tak, jak po starcie programu,
- przerysowanie fraktalu nie może uniemożliwiać interakcji użytkownika – w szczególności powinno być możliwe jednoczesne przerysowywanie fraktalu i czyszczenie wykresu,
- główną cechą tego interfejsu ma być jego funkcjonalność, a nie wygląd – jest to sprawa drugorzędna z powodu ograniczonego czasu ćwiczenia.

Większość implementacji związanej z konfiguracją peryferiów jest gotowa – do studenta należy implementacja „logiki" programu. Zakłada się, że przy skończonej ilości czasu student potrafiłby odtworzyć własnoręcznie kod służący do konfiguracji. Zostało to wykonane wcześniej, aby studentowi oszczędzić pisania tej (nudnej) części programu.

Fraktal W tym ćwiczeniu jako przykład wykorzystania obliczeń w arytmetyce zmiennoprzecinkowej użyty zostanie fraktal – zbiór Mandelbrota. Algorytm jego rysowania znajduje się poniżej:

```
void mandelbrot(){
    static int Px = 0;
    static int Py = 0;
    static float x = 0;
    static float xtemp = 0;
    static float y = 0;
```

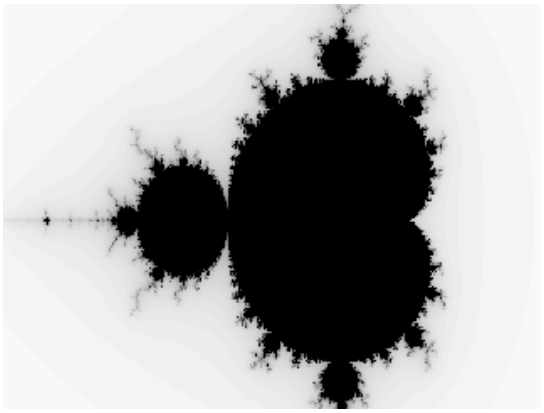
```
    static float x0 = 0;
    static float y0 = 0;
    static uint8_t iteration = 0;

    // obszar rysowany zbioru Mandelbrota
    static float xmin = -0.1011-0.01;
    static float xmax = -0.1011+0.01;
    static float ymin =  0.9563-0.01;
    static float ymax =  0.9563+0.01;

    // for(Px=nr pierwszej kolumny pixeli; Px<=nr ostatniej kolumny pixeli; ++Px)
    for(;;++Px){ // TODO
        // for(Py=nr pierwszego wiersza pixeli; Px<=nr ostatniego wiersza pixeli; ++Py)
        for(;;++Py){ //TODO
            // wyznaczyc x0 i y0 tak, aby:
            // x0 = xmin <=> Px = indeks pierwszej kolumny pikseli;
            // x0 = xmax <=> Px = indeks ostatniej kolumny pikseli;
            // y0 = ymin <=> Py = indeks pierwszego wiersza pikseli;
            // y0 = ymax <=> Py = indeks ostatniego wiersza pikseli;

            x0 = 0.0; // TODO
            y0 = 0.0; // TODO
            // algorytm rysujacy zbior Mandelbrota
            x = 0.0;
            y = 0.0;
            iteration = 0;
            while ((x*x + y*y < 2*2) && (iteration < 0xFF)) {
                xtemp = x*x - y*y + x0;
                y = 2*x*y + y0;
                x = xtemp;
                iteration = iteration + 1;
            }
            // rysowanie piksela w punkcie Px, Py,
            // o kolorze zaleznym od wartosci iteration (0x00-0xFF)
            // TODO
        }
    }
}
```

Ponieważ umiejscowienie rysunku nie jest jednoznacznie zdefiniowane – do programisty należy zadanie odpowiedniego przeskalowania obszaru rysowania (wyznaczenie wartości `x0` i `y0`) i wyznaczenia jego granic (wypełnienie pętli `for`). Oczywiście na koniec należy zaimplementować również rysowanie poszczególnych pikseli w zależności od zmiennej `iteration`. Wynik rysowania z użyciem powyższego kodu powinien być identyczny (z dokładnością do koloru) do widocznego na Rys. 35.



Rysunek 35: Zbiór Mandelbrota w skali szarości

Algorytm ten wykorzystany został jako test szybkości obliczeń wykonywanych przez mikrokontroler. Wielokrotne operacje na liczbach zmiennoprzecinkowych, wykonywane dla każdego z pikseli wyświetlacza, generują znaczne opóźnienie między rozpoczęciem a zakończeniem obliczeń. Opóźnienie to jest widoczne gołym okiem, co powoduje, że konieczna jest implementacja, która pozwoli na jednoczesne rysowanie czasochłonnego obrazu i obsługę pozostałej części interfejsu graficznego przez użytkownika. Jednak jedną z najważniejszych cech tego testu jest to, że bardzo ładnie on wygląda na kolorowych wyświetlaczach.