FT800 w „szponach” Arduino

W Internecie krąży wiele mitów dotyczących wyświetlaczy ze sterownikiem a właściwie akceleratorem graficznym FT800 firmy FTDI. Książki poświęcone wyświetlaczom traktują FT8xx po macoszemu, co gorsza, zarówno turtoriale w Internecie jak i książki opisują skomplikowany sposób pisania oprogramowania ignorując całkowicie istnienie EVE edytora przygotowanego przez firmę FTDI. Sam układ FT800 jest tani (ok 25zł) oferując bardzo duże możliwości. Dostępne w sklepach wyświetlacze w obudowie z płytką sterownika i konwerterem poziomów są tańsze niż podobne rozwiązania firmy NEXTION oferując przy okazji większe możliwości. W artykule można będzie znaleźć informacje jak szybko uzyskać spektakularne efekty na małym mikrokontrolerze dostępnym na płytce ArduinoUNO.

Artykuł przeznaczony jest dla wszystkich, którzy chcieliby w prosty sposób uzyskać zaawansowane efekty na wyświetlaczu o dużej rozdzielczości. Nagrania efektów można obejrzeć na Youtube https://www.youtube.com/playlist?list=PLdtkbzWTUVMl3Sam8Fj1lh3ec1CdCRSXm a stworzenie każdego z nich zajęło nie więcej jak 30 minut. W artykule pominięte zostaną szczegóły techniczne i dokładne opisy komend. Artykuł jest przeznaczony dla osób chcących bez wnikania w szczegóły, w praktyczny i szybki sposób obsłużyć wyświetlacz w swojej aplikacji.

Co będzie potrzebne?
Arduino UNO, wyświetlacz ze sterownikiem FT800 oraz kilka przewodów połączeniowych. Do odsłuchu efektów audio potrzebny będzie zestaw głośników komputerowych lub wzmacniacz audio oraz przewody połączeniowe.

Pierwsze próby.
Przykłady będą uruchamiane na ArduinoUNO oraz wyświetlaczu FT800 5cali o rozdzielczości 480x272 pikseli https://elty.pl/pl/p/Wyswietlacz-LCD-5-480x272-z-panelem-dotykowym-sterowanianie-SPI%2C-kontoler-FT800-/1196 Umieszczam link do sklepu ponieważ nie znalazłem kompletnych wyświetlaczy ze sterownikiem w innych sklepach w Polsce więc nie należy traktować linku jak reklamy lecz wskazówki. Wyświetlacz podłączamy z Arduino według tabelki:

Arduino UNO	Wyświetlacz		Wzmacniacz audio
	Numer pinu	Nazwa	
D8	8	PD	-
D9	7	INT	-
D10	6	CS	-
D11	5	MOSI	-
D12	4	MISO	-
D13	3	SCK	-
+5V	1	VCC	-
GND	2	GND	-
-	9	PWM	Audio IN
-	10	GND	GND

Fotografie (fotografie „01.JPG” do „07.JPG” - coś tam da się wybrać) powinny rozwiać wszelkie wątpliwości. Przewody powinny być krótkie, najlepiej aby nie przekraczać 15..20cm. W razie problemów z komunikacją nie należy korzystać z taśmy tylko pojedynczych przewodów i puścić je „luzem”. Do testu posłuży program:

```
#include <EEPROM.h>
#include <SPI.h>
#include <Wire.h>
#include <FT_VM800B35.h>

FT800IMPL_SPI FTImpl(FT_CS_PIN, FT_PDN_PIN, FT_INT_PIN);
void setup()
{
  FTImpl.Init(FT_DISPLAY_RESOLUTION);
  FTImpl.SetDisplayEnablePin(FT_DISPENABLE_PIN);
  FTImpl.SetAudioEnablePin(FT_AUDIOENABLE_PIN);
  FTImpl.DisplayOn();
  FTImpl.AudioOn();

  FTImpl.DLStart();
  FTImpl.Finish();
  FTImpl.DLStart();
  FTImpl.DLEnd();
  FTImpl.Finish();
}

void loop()
{
    FTImpl.DLStart();
    FTImpl.ColorRGB(0, 255, 255);
    FTImpl.Cmd_Text(200, 133, 28, FT_OPT_CENTERX, "Napis testowy");

    FTImpl.DLEnd();
    FTImpl.Finish();
}
```

Po uruchomieniu powinniśmy zobaczyć napis https://youtu.be/lizoIF8-1t4.
To proste demo zostało stworzone programem EVE edytor, który znajduje się w materiałach dodatkowych. Najnowsza wersja programu dostępna jest na stronie producenta https://www.ftdichip.com/.
Po zainstalowaniu i uruchomieniu programu, w zakładce „Devices” po prawej stronie okna programu, wybieramy typ układu (plik „001.png”) i rozdzielczość wyświetlacza (plik „002.png”). Następnie w domyślnej zakładce „Toolbox” po lewej stronie, rozwijamy listę „Widgets” po czym klikamy lewym przyciskiem myszy na „Text”. trzymając przycisk przesuwamy wskaźnik myszy do okna po prawej stronie ekranu w miejsce, w którym ma się on wyświetlić. Pozycję napisu można skorygować używając myszy lub klikając na zakładkę „Properties” po prawej stronie okna (plik „003.png”) co oznaczono czerwoną elipsą. Zieloną wskazano parametry jakie można zmodyfikować, jak pozycja poziomo, pionowa, wielkość fontów, treść napisu, centrowanie w osi poziomej i pionowej. Punktem odniesienia dla operacji centrowania nie jest ekran tylko środek napisu, który jest oznaczony czerwonym kursorem na ekranie. Jak działa centrowanie najprościej sprawdzić zaznaczając bądź odznaczając poszczególne opcje. W jaki sposób zmienić kolor napisu? Nie ma opcji związanej z kolorem czcionki. Aby zmienić kolor obiektu (nie ważne czy czcionki, linii, itp), po lewej stronie rozwijamy listę „Graphics State”, klikamy na „Color RBG” i przeciągamy do okna wyświetlacza (plik „004.png”) po czym klikamy na kolorowy prostokąt a w oknie, które pojawi się wybieramy interesującą nas barwę (plik „005.png”). Niestety, napis nie zmienił barwy. Spostrzegawczy Czytelnicy, z pewnością zauważyli, że w oknie mieszczącym się na dole okna programu, za każdym razem gdy dodamy jakiś obiekt, pojawiają się kolejne wpisy (plik „006.png”) a po skasowaniu obiektu (należy go wskazać i nacisnąć DEL) wpis jest kasowany. Warto wiedzieć, że okno to nie służy tylko do podglądu ale także do edycji. Można w nim zmieniać

parametry komendy koprocesora graficznego wpływając na miejsce umieszczenia napisu, jego położenie czy barwę. W oknie widzimy, że komenda „COLOUR_RGB” znalazła się za „CMD_TEXT” dlatego nie wpłynęła na kolor napisu. Aby zmiana barwy znalazła się przed napisem trzeba napis skasować po czym dodać dzięki czemu znajdzie się na końcu listy. Jest jednak prostszy sposób. Wystarczy myszką zaznaczyć cały wiersz z komendą zmiany koloru (plik „007.png”) po czym używając CTRL+X skasować go a przy pomocy CTRL+V umieścić przed CMD_TEXT. Efekt tych działań można zobaczyć na ekranie (plik „008.png”). Pozostaje wygenerować kod dla Arduino. W tym celu w menu „Export” wskazujemy „Arduino project” i „VB800B35” (plik „009.png”). Po chwili zostanie otworzony program „ArduinoIDE” z wygenerowanym programem” (plik „010.png”). Po wskazaniu płytki UNO (plik „011.png”) oraz portu COM (w moim przypadku jest to COM34) (plik „012.png”) wystarczy program skompilować i wgrać do Arduino. Niestety, w moim przypadku nie było zbyt dobrze (fotografia „9.JPG” - wyświetlacz odbija wszystko, bez filtru polaryzacyjnego ciężko zrobić dobre foto), jak to niektórzy mawiają: ZONK. Porównując ekran w EVE edytorze z tym co na wyświetlaczu widać, że jest ustawiona zła, prawdopodobnie za mała rozdzielczość wyświetlacza. Próby wyboru innych płytek w opcjach eksportu spełzły na niczym. Postanowiłem więc obejrzeć zawartość pliku „FT_VM800B35.h”. Dlaczego akurat tego? Powód jest prosty (plik „013.png”). A gdzie go szukać? Arduino trzyma biblioteki w dwóch miejscach (plik „014.png”) i (plik „015.png”) w ”Moje dokumenty” (ten katalog może mieścić się w innej lokalizacji). Interesujący nas plik znajduje się w „Moje dokumenty/arduino/libraries/FTDI_FT800” a w nim to czego szukamy (plik „016.png”). W razie problemu z odnalezieniem katalogu z plikami, można przeszukać zawartość komputera pod kątem pliku „FT_VM800B35.h”. W pliku znajdziemy informacje o pinach do których podłączone są linie sterujące:

```
//macros specific to FT800 hardware
/* Macros used for CS, PDN, INT pins for SPI - default values */
#define FT_CS_PIN                                   10
#define FT_PDN_PIN                                  8
#define FT_INT_PIN                                  9
```

oraz wybraną rozdzielczość:

```
/* Macros related to display dimensions */
#define FT_DISPLAYWIDTH             FT_DISPLAY_HSIZE_QVGA
#define FT_DISPLAYHEIGHT    FT_DISPLAY_VSIZE_QVGA
#define FT_DISPLAY_RESOLUTION    FT_DISPLAY_QVGA_320x240
```

która jak widać nie zgadza się z tą ustawioną w EVE edytorze. Zmieniłem zawartość pliku na:

```
#define FT_DISPLAYWIDTH             480
#define FT_DISPLAYHEIGHT            272
#define FT_DISPLAY_RESOLUTION    FT_DISPLAY_WQVGA_480x272
```

, skompilowałem, wgrałem do Arduino i sukces, który można zobaczyć na filmie https://youtu.be/lizoIF8-1t4. No tak, ale napis na filmie porusza się. Tak, bo zmodyfikowałem trochę program, konkretnie pętlę główną:

```
void loop()
{
  int16_t static x = -60;

  uint32_t static t;
  if ( millis() > t ) {
    t = millis() + 5;

    FTImpl.DLStart();
    FTImpl.ColorRGB(0, 255, 255);
    FTImpl.Cmd_Text(x, 133, 28, FT_OPT_CENTERX, "Napis testowy");

    FTImpl.DLEnd();
    FTImpl.Finish();

    if( ++x > 550 ) x = -60;
```

```
  }
}
```

Efekt: https://youtu.be/cNcEKuQl4Ow. Proszę zauważyć, że w programie nie ma delay, opóźnienie jest realizowane z wykorzystaniem funkcji „millis”. Program dla Arduino i zmodyfikowany plik „FT_VM800B35.h” znajduje się w materiałach dodatkowych w katalogu „Program Arduino/L01”. Warto zapoznać się z przykładami, które można znaleźć w katalogu (plik „017.png”). Postanowiłem zmodyfikować efekt:

```
void loop()
{
  int16_t static x = -60;

  uint32_t static t;
  if ( millis() > t ) {
    t = millis() + 20;

    uint8_t static r, g = 255, b, d = 50;
    if ( --d == 0 ) {
      d = 50;
      r = random(0, 255);
      g = random(0, 255);
      b = random(0, 255);
    }

    digitalWrite(7, 0);
    FTImpl.DLStart();
    FTImpl.ColorRGB(r, g, b);
    FTImpl.Cmd_Text(x, 133, 28, FT_OPT_CENTERX, "Napis testowy");

    FTImpl.DLEnd();
    FTImpl.Finish();
    digitalWrite(7, 1);

    if ( ++x > 550 ) x = -60;
  }
}
```

Teraz co sekundę napis zmienia kolor. Czas zająć się czymś bardziej zaawansowanym. Może wyświetlić obrazek? Najpierw należy przygotować obrazek. Można użyć obrazka stworzonego programem graficznych czy fotografii. Obrazek należy stworzyć czy przeskalować do wymiarów jakie chcemy uzyskać na ekranie. Na początek wybrałem sobie logo AVT, które skopiowałem ze strony wydawnictwa. Po wybraniu w EVE edytorze nowego projektu należy wybrać z lewej strony zakładkę „Content” po czym naciosnąć „ADD” (plik „018.png”). W oknie wyboru należy wskazać plik z obrazkiem. Po wczytaniu obrazek zostanie wyświetlony z prawej strony okna programu (plik „019.png”). W dolnej części okna widać zajętość pamięci układu FT800. Aby uniknąć męczącego klikania proponuję wkleić w okno „Coprocessor” poniższe komendy:

```
CLEAR(1,1,1)
BITMAP_HANDLE(0)
BITMAP_SOURCE(0)
BITMAP_LAYOUT(ARGB1555, 210, 105)
BITMAP_SIZE(NEAREST, BORDER, BORDER, 105, 105)
BEGIN(BITMAPS)
VERTEX2II(165, 133, 0, 0)
END()
```

W efekcie na ekranie pojawi się logo AVT (plik „020.png”). Co oznaczają kolejne komendy?
CLEAR - Ustawia stan początkowy układu FT8xx.
BITMAP_HANDLE - „Uchwyt” do obrazka (podobnie jak uchwyt do pliku), jego identyfikator. Może przyjmować wartości od 0 do 15. Każdy obrazek/obiekt musi mieć inny identyfikator.
BITMAP_SOURCE - Adres obrazka w pamięci FT800. Skąd dowiedzieć się jaki obrazek ma adres? W zakładce „Content” należy w oknie wskazać obrazek (plik „021.png”) co oznaczono czerwona elipsą. Wtedy z prawej strony, pod obrazkiem, znajdziemy informacje o jego adresie

(zielona elipsa). Poniżej można odczytać informacje o jego rozmiarze i rozdzielczości. Warto zapamiętać wartość parametru „Stride” (pomarańczowa elipsa).
BITMAP_LAYOUT - określa format obrazka i jego rozmiary. Pierwszy to użyty format. Można go zmienić wybierając z listy (plik „022.png”). L1, L4, L8 to obraz monochromatyczy (L1) lub w odcieniach szarości (L4, L8). Efekt wyboru jest widoczny natychmiastowo na ekranie tak samo jak i wielkość obrazka po kompresji. AVR mega328 użyty w ArduinoUNO nie ma zbyt wiele pamięci, dlatego trzeba wybrać format taki, który zapewnia jak najlepszy stosunek rozmiaru obrazka do jakości. Drugi parametr komendy to wspomniany „Stride”, trzeci rozdzielczość pionowa.
BITMAP_SIZE - Ostatnie dwa parametry nie wymagają chyba tłumaczenia, dwa pierwsze najprościej przetestować. Dla przypomnienia, klikamy w wiersz z interesująca nasz komendą (plik „023.png”) po czym po prawej stronie pokaże się okno, w którym możemy „wyklikać” ustawienia.
BEGIN - Informujemy o wykonywaniu operacji na obiekcie „BEGIN(BITMAP)”, jak łatwo się domyśleć, na bitmapie.
VERTEX2II - Komenda koprocesora. Pierwsze dwa parametry to położenie bitmapy. Kolejny, „handle” wspomniany wcześniej uchwyt, ostatni, „cell” zostawiamy 0.
END - Koniec operacji na obiekcie. Zanim pojawi się END mogą być wykonane na obiekcie operacje takie jak obroty czy zmiana rozmiaru.
Teraz można wygenerować obrazek i … i błąd (plik „024.png”). Nie ma się co przejmować, wystarczy „prog_uchar” zamienić na „int8_t”, skompilować i .... kolejny błąd (plik „025.png”). Tym razem brakuje const przed „static PROGMEM”. Po zmianie linii na „const static PROGMEM uint8_t IMAGES_LOGO_AVT[] = {” program kompiluje się poprawnie i działa.
Wskazówka!
W nazwie pliku (pomijam kropkę i rozszerzenie) nie należy używać znaków, które nie są akceptowane w nazwach etykiet w języku C/C++ (nie języku Arduino, taki NIE ISTNIEJE, podobnie jak procesor czy mikrokontroler Arduino). Jeśli w nazwie pliku znajda się niedozwolone znaki (spacja, polskie znaki, itp), trzeba będzie w wielu miejscach programu poprawiać nazwy etykiety.
Z czego wynikają błędy? EVE edytor był pisany gdy kompilator miał inne wymagania, nowsza wersja „czepia” się braku const dla danych umieszczanych w FLASH. Dzięki temu jest bardziej zgodny z kompilatorami dla innych mikrokontrolerów.

Teraz podniesiemy poprzeczkę, dodamy kolejny obrazek. Spodobał mi się robot na stronie AVT. Wczytujemy jak poprzedni (plik „026.png”). Po lewej stronie widać kolejny plik, po prawej można odczytać adres w pamięci, rozdzielczość i stwierdzić, że 30% pamięci jest już zajęte. Podobnie jak w poprzednim przypadku, nie będziemy klikać tylko dokleimy w oknie koprocesora komendy:

```
BITMAP_HANDLE(1)
BITMAP_SOURCE(22052)       // Adres grafiki w pamięci FT800
BITMAP_LAYOUT(ARGB1555, 354, 162)
BITMAP_SIZE(NEAREST, BORDER, BORDER, 177, 162)
BEGIN(BITMAPS)
VERTEX2II(16, 33, 1, 0)
END()
```

Naszym oczom ukaże się (plik „027.png”). Po wygenerowaniu kodu, niezbędnych poprawkach:

```
const static PROGMEM uint8_t IMAGES_LOGO_AVT[] = {
const static PROGMEM uint8_t IMAGES_ROBOT[] = {
```

Stwierdzimy, że program nie mieści się w mikrokntrolerze (plik „028.png”). Cóż, do zabawy z bitmapami, 32kB to za mało. Widać, że nie wiele pamięci zabrakło. W takim razie, zgodzimy się na mniejszą jakość obrazka (plik „029.png”). Ponowne generowanie projektu, poprawki w pliku źródłowym, kompilacja i udało się! Nawet zostało 13% wolnej pamięci, niech więc obrazki poruszają się. Mała zmiana kodu:

```
void loop()
{
  int16_t static x1, y2;
  bool static d1, d2;
```

```
  uint32_t static t;
  if ( millis() > t ) {
    t = millis() + 20;

    FTImpl.DLStart();
    FTImpl.BitmapHandle(0);
    FTImpl.BitmapSource(0);
    FTImpl.BitmapLayout(FT_ARGB1555, 210, 105);
    FTImpl.BitmapSize(FT_NEAREST, FT_BORDER, FT_BORDER, 105, 105);
    FTImpl.Begin(FT_BITMAPS);
    FTImpl.Vertex2ii(x1, 133, 0, 0);
    FTImpl.End();
    FTImpl.BitmapHandle(1);
    FTImpl.BitmapSource(22052);
    FTImpl.BitmapLayout(FT_ARGB4, 354, 162);
    FTImpl.BitmapSize(FT_NEAREST, FT_BORDER, FT_BORDER, 177, 162);
    FTImpl.Begin(FT_BITMAPS);
    FTImpl.Vertex2ii(160, y2, 1, 0);
    FTImpl.End();

    FTImpl.DLEnd();
    FTImpl.Finish();

    if ( ! d1 ) {
      if ( ++x1 > 350 ) d1 ^= 1;
    }
    else{
      if ( --x1 < 2 ) d1 ^= 1;
    }

    if ( ! d2 ) {
      if ( ++y2 > 250 ) d2 ^= 1;
    }
    else{
      if ( --y2 < 2 ) d2 ^= 1;
      }

  }
}
```

i efekt widoczny na nagraniu https://youtu.be/DtxmG75tOOc. Program dostępny jest w materiałach dodatkowych w katalogu „L02”.

Jak się można było przekonać, dwa niewielkie obrazki mimo wybrania oszczędnych formatów zapełniły prawie całą pamięć ArduinoUNO. Czy trzeba więc sięgać po płytki z większymi mikrokontrolerami? Niekoniecznie. Jeśli obrazki rysowane są programem bitmapowym można ograniczyć liczbę kolorów i zastosować format RGB332 lub ARGB2. W przypadku fotografii można skorzystać z JPEG. Ze względu na to, że wyświetlanie JPEG nie jest takie proste, zwłaszcza jeśli obrazków ma być więcej, zostawię to zagadnienie na inna okazję podobnie jak i własne czcionki.

Warto podejrzeć w lekcji L02b (katalog „L02b”) jak zrealizowano efekt, w którym logo AVT porusza się raz nad obrazkiem robota innym razem pod https://youtu.be/X7yhAjnDWFE.

Nadszedł czas aby wyświetlacz przemówił.

FT800 posiada jedno wyjście audio. Może odtwarzać dźwięk syntezowany jak i sample. Można regulować głośność dźwięku w 256 krokach. Wyjście dźwiękowe to sygnał PWM i powinien być filtrowany przez filtr dolnoprzepustowy ale do prób można go podać bezpośrednio na wejście wzmacniacza. W pierwszej demonstracji posłużę się przykładem zaczerpniętym ze strony producenta. Po skompilowaniu pliku „L03.ino” i wgraniu do płytki, w pierwszej kolejności należy skalibrować ekran dotykowy dotykając trzech kropek, które będą pojawiać się na ekranie. Po

skalibrowaniu ekranu naciśnięciu przycisku „Play” towarzyszy dźwięk. Głośność, częstotliwość i rodzaj dźwięku można zmienić w funkcji „Sound()” zmieniając poniższy fragment kodu:

```
/* Assign the sound parameters - experiment audio engine by modifying the below parameters */
Vol = 255;//volume of the sound played
Pitch = FT_MIDI_C_6;//midi note to be played - freq
Sound = FT_PIANO;//synthesized sound to be played
```

Definicje instrumentów i dźwięków znajdują się w pliku „FT800.h” znajdującym się w „ścieżka/Arduino/librafies/FTDI_FT800/hardware/FT800/”. Aby odciążyć czytelników od jego szukania znajduje się w katalogu z lekcjami.

Wyświetlacz miał mówić a jak na razie popiskuje. Zaraz temu zaradzimy. Trzeba zacząć od przygotowania dźwięków. Na początku opiszę standardy jakie potrafi odtwarzać układ Ft800:

- Sample8-bit Próbki dźwiękowe 8-bit. Standard używany w Amigach i na kartach Covox komputerów C-64 i PC.
- µlaw Kompresja dynamiki z 12-bit do 8-bit na zasadach podobnych do działania preemfazy w radiu FM. Kodowanie używane w telefonii cyfrowej ISDN i VoIP (kodek G711).
- ADPCM Kompresja 8-bit na 4-bit. Zapisywana jest jedna próbka 8-bit następnie 4-bit informujące o różnicy w głośności pomiędzy poprzednio zakodowanym poziomem głośności.

Do obróbki sampli najlepiej posłużyć się programem Audacity. Gotowe sample należy skonwertować do formatu RAW, 8-bit ze znakiem. Kiedyś taka opcja dobrze działała w Audacity, teraz jest ale nie daje wyboru parametrów. Skorzystałem z „Switch Audio Converter” do poprania https://www.nchsoftware.com/howto/convert/wav_to_raw_files.html oraz w materiałach dodatkowych. Po uruchomieniu programu i wskazaniu pliku WAV wybieramy format zapisu (plik „030.png”) po czym klikamy na „Options” (czerwona elipsa) wybierając parametry (plik „031.png”). Najważniejsze to dźwięk mono i format „8 bit signed”. Czestotliwość zależnie od wymaganej jakości i dostępnej pamięci w mikrokotrolerze może zmieniać się od 8 do 22kHz. Wygenerowany plik RAW trzeba skonwertować do pliku ascii. W tym celu skorzystałem z programu HEX2BIN. Aby ułatwić sobie zadanie przygotowałem plik BAT:

```
bin2c -o ciemnosc.h ciemnosc.raw
pause
```

Po jego uruchomieniu plik „ciemonosc.raw” zostanie skonwertowany i zapisany pod nazwą „ciemnosc.h”. W pliku należy dokonać małej zmiany aby mógł być używany z AVR.

```
const unsigned char nazwa_pliku_raw[
```

należy zamienić na:

```
const PROGMEM unsigned char nazwa_pliku_raw[
```

W kodzie źródłowym trzeba dołączyć plik:

```
#include „nazwa_pliku_raw.h”
```

Teraz będzie już z górki. Co prawda w bibliotekach FT800 nie znalazłem funkcji przesyłającej próbki do pamięci FT800 ale łatwo ja napisać:

```
 FTImpl.StartWrite(FT_RAM_G + GPUAddr);
 for ( uint32_t x=0; x<len; x++) {
   //Ft_Gpu_Hal_Transfer8(host,*buffer);
   FTImpl.Transfer(pgm_read_byte_near(plik)); plik++;
 }
 FTImpl.EndTransfer();
```

W przykładowym kodzie odtwarzającym sample (katalog „L04”) ładowanie dźwięku połączone jest z jego odtwarzaniem:

```
void PlayAudioWav( const uint8_t *plik, uint32_t len, uint16_t freq, uint32_t GPUAddr, bool wait) {
 FTImpl.StartWrite(FT_RAM_G + GPUAddr);
 for ( uint32_t x=0; x<len; x++) {
   FTImpl.Transfer(pgm_read_byte_near(plik)); plik++;
 }
 FTImpl.EndTransfer();
```

```
  FTImpl.PlayAudio( 255, 0, 8000, GPUAddr, len, 0);

  if ( wait ) {
    while ( FTImpl.Read( REG_PLAYBACK_PLAY) == 1) ; // Zaczekaj na koniec odtwarzania
  }
}
```

Aby odtworzyć dźwięk należy wywołać funkcję przekazując adres danych w pamięci FLASH, ich długość, częstotliwość odtwarzania, adres w pamięci FT800, pod który zostaną zapisane dane i z spod, którego będą odtwarzane oraz czy funkcja ma czekać na koniec odtwarzania dźwięku czy nie. Wywołanie dla sampli „ciemnosc" wygląda jak poniżej:

```
PlayAudioWav( ciemnosc_raw, ciemnosc_raw_size, 8000, 0, 0);
```

Na koniec zmagań z dźwiękami poruszę dwa zagadnienia:
Co zrobić gdy sample zapisane są z próbkowaniem 8kHz 8-bit a nadal brakuje pamięci w mikrokontrolerze? Kompresja MP3 odpada ponieważ FT800 jej nie obsługuje a AVR jest zbyt wolny aby dekompresję obsłużyć on-line. Rozwiązaniem jest kodowanie ADPCM ale w chwili pisania artykułu nie udało się znaleźć konwertera, któr generowałby pliki, które są odtwarzane przez FT800.

Jak odtwarzać dźwięki dłuższe niż mieszczą się w FT800? Najpierw trzeba rozwiązać problem małej pamięci AVR. Problemy są podobne jak w przypadku grafik i do zabaw z samplami najlepiej skorzystać z ARM, ESP32, ESP8266 ostatecznie płytki ArduinoMega2560.
Przykładowy dźwięk „ciemnosc" trwający 1,8sekundy ładowany jest do FT800 w czasie 42ms. Można więc w pamięci FT800 stworzyć dwa banki pamięci. Do jednego ładuje się fragment do odtworzenia a w tym czasie inny jest odtwarzany. FT800 może wygenerować przerwanie po zakończeniu odtwarzania, więc w przerwaniu wystarczy nakazać odtworzyć kolejny fragment a w programie głównym zacząć ładowanie dźwięku do banku, który już jest nieużywany.

Ekran dotykowy.

Prezentacje możliwości ekranu dotykowego rozpoczniemy od umieszczenia na nim widgetu „keys" ale najpierw zmienimy kolor tła (plik „032.png"). Następnie umieszczamy „Keys" (plik „033.png") we właściwościach zmieniając napis na „1234" oraz wyróżniając przycisk „1" (plik „034.png"). Pozostaje wygenerować projekt i sprawdzić czy działa. Jeśli wyświetla się to czego oczekujemy mała zmiana pętli loop:

```
void loop()
{
  uint32_t static tim;
  char static z = '0';


  if ( millis() > tim ) {
    tim = millis() + 1000;
    if ( ++z > '4' ) z = '1';

    FTImpl.DLStart();
    FTImpl.ClearColorRGB(143, 143, 143);
    FTImpl.Clear(1, 1, 1);
    FTImpl.Cmd_Keys(107, 77, 160, 36, 29, z, "1234");

    FTImpl.DLEnd();
    FTImpl.Finish();
  }
}
```

Teraz co sekundę zmieniony będzie wyróżniony przycisk. Teraz dodamy obsługę ekranu dotykowego. W tym celu dodajemy funkcję calibrate:

```
void Calibrate()
```

```
{
	FTImpl.DLStart();
	FTImpl.ClearColorRGB(64,64,64);
	FTImpl.Clear(1,1,1);
	FTImpl.ColorRGB(0xff, 0xff, 0xff);
	FTImpl.Cmd_Text((FT_DISPLAYWIDTH/2), (FT_DISPLAYHEIGHT/2), 27, FT_OPT_CENTER,
"Dotknij kolejno pulsujacych punktow.");
	FTImpl.Cmd_Calibrate(0);

	FTImpl.Finish();
}
```

i dodajemy jej wywołanie na koniec setup(). Po modyfikacji programu pętli głównej:

```
void loop()
{
  char static z = '0';
  uint32_t static tagval;

  sTagXY sTagxy;
  FTImpl.GetTagXY(sTagxy);


  uint32_t static prev;
  if ( prev != sTagxy.tag ) {
    prev = sTagxy.tag;
    Serial.println( sTagxy.tag );
  }


  FTImpl.DLStart();
  FTImpl.ClearColorRGB(143, 143, 143);
  FTImpl.Clear(1, 1, 1);

  //FTImpl.Tag('k');
  FTImpl.Cmd_FGColor(0x040FAD); //Kolor przycisku
  if( sTagxy.tag >= '1' && sTagxy.tag <= '4' ){
    z = sTagxy.tag;
  }
  FTImpl.Cmd_Keys(107, 77, 160, 36, 29, z, "1234");

  if (sTagxy.tag == 'p' ){
    tagoption = FT_OPT_FLAT;
    FTImpl.Cmd_FGColor(0xFFFF00); //Kolor przycisku
  }
  else{
    FTImpl.Cmd_FGColor(0x040FAD); //Kolor przycisku
  }
  FTImpl.Tag('p');
  //COLOR_RGB(255, 255, 127); // kolor tekstu
  FTImpl.Cmd_Button(131, 160, 120, 36, 27, 0, "Play");

  FTImpl.DLEnd();
  FTImpl.Finish();
}
```

Skomplikowaniu i uruchomieniu wyświetli się ekran kalibracji, po której ukarze się ekran z przyciskami „radiowymi” 1234 oraz przycisk „Play”. Naciskając przyciski „1234” powodujemy ich przełączenie, naciskając „Play” zmieniamy jego kolor. Wszystko to można zobaczyć na filmie https://youtu.be/eoUznvN-pic. Teraz wyjaśnię działanie programu i wygodną funkcjonalność tagów, która uwalnia programistę od kontrolowania położenia i kształtu obiektu, który jest wskazywany na ekranie. Funkcja „ FTImpl.GetTagXY(sTagxy);” zwraca nr tagu obiektu. Jak definiuje się numer tagu będzie napisane niebawem. Fragment „if(sTagxy.tag >= '1' && sTagxy.tag <= '4') z = sTagxy.tag;” zmienia stan zmiennej „z”. Gdy zwracany tag ma wartość w zakresie kodów ASCII

odpowiadających znakom cyfr 1..4. Funkcja „FTImpl.Cmd_FGColor(0x040FAD); „ zmienia kolor obiektu. W EVE osiągalna jest w menu (plik „035.png”). Kolejna funkcja, znana już „FTImpl.Cmd_Keys(107, 77, 160, 36, 29, z, "1234");” umieszcza przyciski „1234”. Tym razem, zmienna „z” nie jest modyfikowana co sekundę ale przez rezultat funkcji zwracającej numer wskazanego tagu. W konsekwencji, zależnie od wskazanego obiektu, zwracany jest tag kodu ASCII cyfry 1...4 a funkcja „Cmd_Keys” uaktywnia wybrany przycisk. W przypadku widgetów „Keys”, tag jest taki sam jak kod ASCII znaku. Inaczej jest w przypadku innych obiektów, tym trzeba nadać unikalny numer tagu, różny od tagów „Cmd_Keys”. Może to być dowolna liczba z zakresu 1..255. Fragment:

```
if (sTagxy.tag == 'p' ){
  FTImpl.Cmd_FGColor(0xFFFF00);
}
else{
  FTImpl.Cmd_FGColor(0x040FAD);
}
```

reaguje na tag o wartości kodu ASCII małej litery „p”. Zależnie od tego, czy tag jest aktywny czy nie, kolor tła obiektu jest ustawiany na niebieski bądź żółty:

```
FTImpl.Cmd_Button(131, 160, 120, 36, 27, 0, "Play");
```

Jak nadać numer tagu? Przed umieszczeniem obiektu należy wywołać funkcję:

```
FTImpl.Tag();
```

Argumentem jest ID tagu. W przykładowym kodzie jest to kod litery „p”:

```
FTImpl.Tag('p');
//COLOR_RGB(255, 255, 127); // kolor tekstu
FTImpl.Cmd_Button(131, 160, 120, 36, 27, 0, "Play");
```

Warto wiedzieć, że tag będzie obowiązywał do momentu odwołania. Naciśnięcie jakiegokolwiek obiektu umieszczonego za tagiem będzie traktowane tak jak wskazanie pierwszego z nich. Cały program znajduje się w katalogu „L05”.

Kalibracja ekranu dotykowego.

Nie jest tajemnicą, że ekran dotykowy trzeba skalibrować ale czy trzeba to robić po każdym uruchomieniu programu? Nie, nie trzeba, wystarczy raz a efekt kalibracji zapamiętać. Rejestry kalibracyjne można odczytać funkcją:

```
for (char x = 0; x < 6; x++) {
  touch[x] = FTImpl.Read32( REG_TOUCH_TRANSFORM_A + x * 4);
}
```

zapisać:

```
for (char x = 0; x < 6; x++) {
  FTImpl.Write32( REG_TOUCH_TRANSFORM_A + x * 4, touch[x]);
}
```

Naturalnie dane należy przechowywać w pamięci nieulotnej.

Ostatni przykład z lekcji 6 uzupełnijmy „Silderem” (plik „036.png), regulatorem „Dial” (plik „037.png) i dwoma polami liczbowymi (plik „038.png). Aby ułatwić identyfikację pól można nadać im wartości początkowe, ja wybrałem 5 i 30. Wygenerowany kod należy uzupełnić o funkcje odczytu ekranu dotykowego oraz dodać tagi. W przykładowym kodzie „Dial” otrzymał tag „d”, Silder tag „s”, pozostałe obiekty tag o wartości 1. Nie wyłączam tagów, ponieważ raz wyłączone nie mogą być ponownie załączone. Lepiej wybrać nieużywaną wartość dzięki czemu w kolejnym obiektom, w razie potrzeby można nadać tagi. Aby odczytać miejsce, w którym wskazano obiekt, trzeba zdefiniować strefę objętą kontrolą funkcją „Cmd_Track „. W przypadku Slider'a będą to jego rozmiary:

```
FTImpl.Tag('s');
FTImpl.Cmd_Slider(30, 225, 326, 20, 0, valSlider, 255 );
FTImpl.Cmd_Track(30, 225, 326, 20, 's');
```

Trochę inaczej jest w przypadku gałki. Wtedy wskazujemy obszar wielkości jednego piksela w

lewym górnym rogu obiektu:

```
FTImpl.Tag('d');
FTImpl.Cmd_Dial(379, 76, 42, 0, valGalka );
FTImpl.Cmd_Track(379, 76, 1, 1, 'd');
```

Numer obiektu i zwracana wartość znajduje się w rejestrze REG_TRACKER. W bitach 0..7 zwracany jest numer tagu, w bitach 16..31 wartość z zakresu 0..65535. Po przeskalowaniu można przypisać odpowiednie położenie suwaka czy gałki lub ją wyświetlić. Zmodyfikowany kod dostępny jest w katalogu „L06" a przykład działania na filmie https://youtu.be/kOc6_apOY-k.

Miłe dodatki.

W czasie wykonywania długotrwałych operacji przez mikrokontroler, na przykład wczytywania dużego pliku z karty SD, na ekranie może pojawić się animowany wskaźnika zajętości. Kiedyś to była klepsydra, teraz na topie są inne. Dostępne są cztery jego rodzaje (plik „039.png) (plik „040.png) (plik „041.png) (plik „042.png) każdy w wielu rozmiarach, przy czym na wyświetlaczu 480x272 sens mają wartości z zakresu 0..2. Spinner uaktywnia funkcja:

```
FTImpl.Cmd_Spinner(217, 148, 3, 1);
```

Co ważne, po uruchomieniu spinner'a mikrokontroler nie jest angażowany w jego animowanie. Przykładowy program znajduje się w katalogu „L07". Efekt działania programu można obejrzeć na Youtube:

https://youtu.be/MW7wGF1pnwo
https://youtu.be/Cbeul6HFAF4
https://youtu.be/VAHgxhqD1LU
https://youtu.be/imQBeMGv2fY
https://youtu.be/FHSB3tUYI-M

Kolejnym dodatkiem jest wygaszacz ekranu, którym może być grafika poruszająca się po ekranie (plik „043.png). W przeciwieństwie do przykładu z lekcji 2 w tym przypadku, tak jak i w przypadku spinner'a, mikrokontroler nie jest angażowany w animację, wszystko realizuje makro. Efekty można zobaczyć na https://youtu.be/a_hPWu0teCg. Program dostępny w katalogu „L08".

Na zakończenie.

Starałem się aby kod był prosty i przejrzysty bez zbędnych dodatków.
Temat obsługi akceleratora FT800 został tylko „liźnięty". Aby szczegółowo omówić możliwości FT800 należałoby o każdym zagadnieniu napisać artykuł o objętości porównywalnej z powyższym artykułem. Jeżeli Czytelnicy są zainteresowaniu dłuższym, bardziej szczegółowym opisaniem możliwości układu, proszę o e-maile do redakcji w tej sprawie. W e-mailach proszę pisać jakie zagadnienia należy poruszyć i na jakie platformy sprzętowe bo jak wykazał artykuł mały AVR nie za bardzo nadaje się do przechowywania dużej ilości danych. Problemu nie rozwiązuje zewnętrzna karta SD czy pamięć DataFlash a to ze względu na powolny interfejs SPI i brak DMA.
W moim odczuciu, można wykorzystać małe AVR o ile zrezygnuje się z grafik lub ograniczy ich rozmiar. Duże AVR, jakkolwiek niekorzystne cenowo, to jednak popularne w płytkach ArduinoMega2560 pozwolą na większe grafiki i ograniczoną ilość samplowanych dźwięków. Xmega mają niekorzystną cenę w stosunku do ARM i nie widzę sensu ich używania i nie tylko ja co widać po projektach w czasopismach i na forach internetowych gdzie królują AVR i ARM. Co zaś do ARM, to ograniczeń nie ma ale czy jest sens używania FT800 w połączeniu z ARM? Nie prościej i co ważne taniej podłączyć wyświetlacz bezpośrednio do ARM?

SaS, AVT
sas@elportal.pl