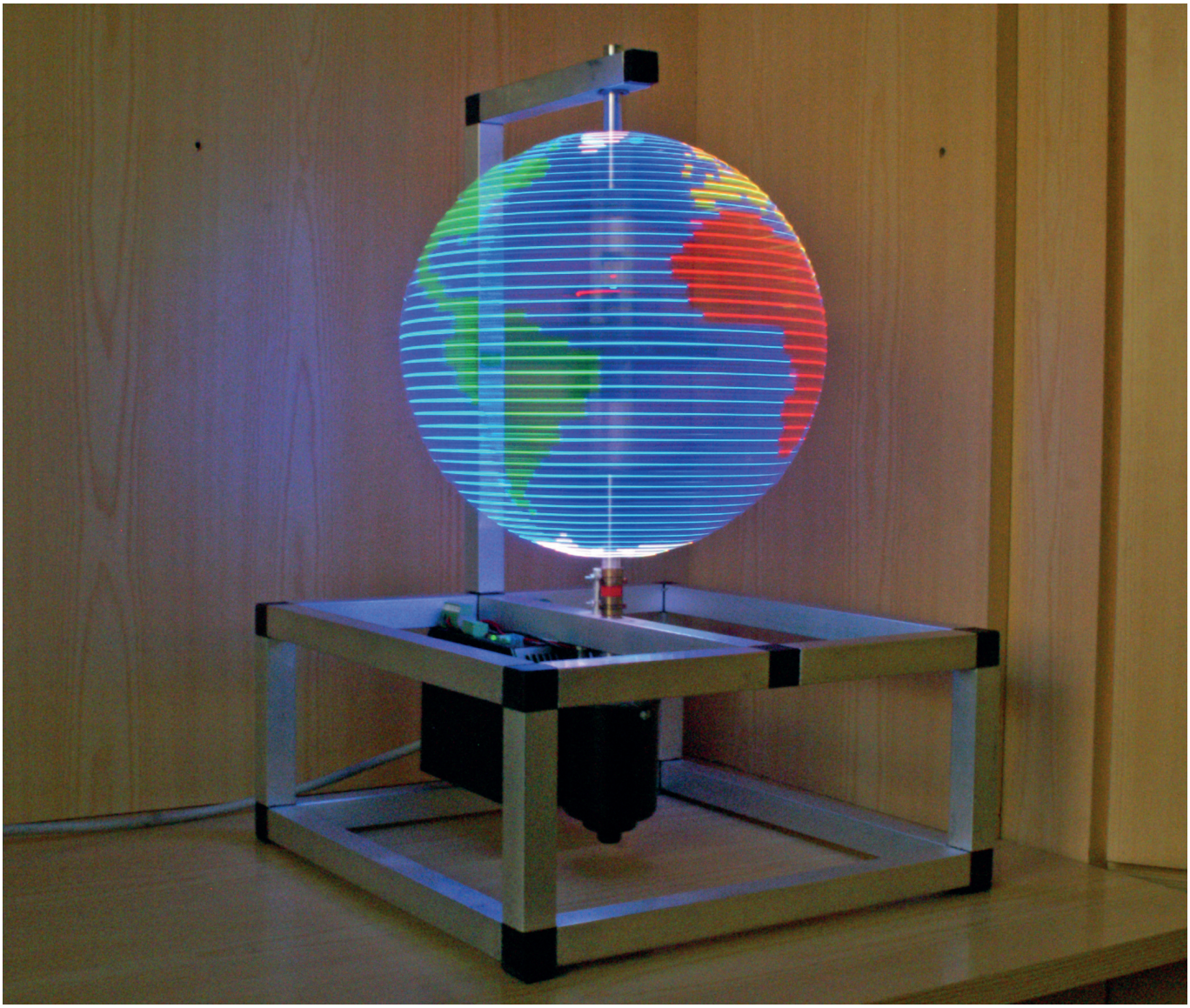


Kulisty, kolorowy wyświetlacz widmowy

Zainspirowany projektami wyświetlaczy widmowych oraz nowoczesnymi LED-ami, postanowiłem zbudować własne urządzenie i wyposażyć je w interesujące funkcje. Obraz powstały w wyniku skanowania mechanicznego nie jest wstanie osiągnąć jakości dostępnej na ekranie monitora czy telewizora, natomiast z powodzeniem może konkurować z reklamami-tablicami LED czy telebimami. Łatwy w wykonaniu, nietypowy kształt wyświetlacza czyni go ciekawym urządzeniem do prezentacji reklam, napisów lub obrazów. Zdalne sterowanie wykorzystujące technologię Bluetooth umożliwia pełną kontrolę nad wyświetlaczem, z poziomu komputera lub telefonu.

Rekomendacje: *wyświetlacz może być atrakcją witryny sklepowej, stoiska targowego, szkolenia lub prezentacji.*

Kulisty wyświetlacz składa się z dwóch części: wirującej linijki diodowej wraz z elementami elektronicznymi oraz stelaża z silnikiem i zasilaczem. Opcjonalnie jest wykorzystywany komputer służący do przesyłania wzorów obrazów, przełączania obrazów wyświetlanych z pamięci urządzenia, tworzenia animacji, sterowania kolorami, a także do bieżącej kontroli parametrów urządzenia.

Za tworzenie czytelnego obrazu odpowiedzialne są ruchome diody zapalające się cyklicznie w sposób zsynchronizowany. Układ migających diod powoduje błyski, które podczas ruchu układają się np. w czy-

telny napis. Ich błyski zostają tworzą obraz dzięki bezwładności oka ludzkiego. W celu osiągnięcia dobrej jakości obrazu i zwiększenia ilości kolorów zastosowano mechaniczny odpowiednik PWM, którego sposób działania szczegółowo opisano w artykule.

Założenia projektowe

Wyświetlacz widmowy ma umożliwić wyświetlenie dowolnego obrazu zapamiętanego w pamięci mikrokontrolera. Ze względu na jedną z planowanych funkcji wyświetlacza, którą jest prezentacja kontynentów kuli ziemskiej, kształt matrycy wyświetlającej będzie sferyczny. Wirująca linijka będzie wyposażona w 48 jasnych diod LED RGB, a wyświetlanie danych, które będzie precyzyjnie zsynchronizowane z jej położeniem, będzie zachodziło 96 razy w ciągu obrotu, co pozwoli na uzyskanie rozdzielczość matrycy 48×96 pikseli. Kulisty wyświetlacz, docelowo wyświetlający obrazy, będzie miał możliwość wyświetlania animacji, składającej się z przełączanych obrazów. Jakość i długości animacji będzie uwarunkowania ilością dostępnej pamięci Flash sterownika wyświetlacza oraz szybkością bezprzewodowej transmisji danych. Przesyłane, przechowywane i analizowane za pomocą mikrokontrolera obrazy będą wcześniej przygotowane i zapisane w odpowiednim formacie. Za wstępną obróbkę obrazów będzie odpowiadała aplikacja napisana w Matlabie.

Do napędzania wirującej linijki diodowej wybrano silnik DC firmy Daewoo o symbolu producenta LM76103118 pochodzący z napędu wycieraczek samochodowych. Jest on zasilany napięciem 12 V. Przy pracy z obciążeniem silnik pobierając prąd 6 A uzyskuje prędkość około 18 obr./s, co pozwala na uzyskanie poprawnego efektu widmowego. W modelu przewidziano zintegrowanie wirującej linijki z kulistą obudową w celu poprawy aerodynamiki, a co za tym idzie, znacznego zwiększenia prędkości obrotowej. Silnik nie umożliwia uzyskania stabilnych obrotów o ściśle określonej liczbie w jednostce czasu, ale jest bardzo łatwy w uruchomieniu i nie wymaga stosowania dodatkowych regulatorów sprzężonych z mikrokontrolerem urządzenia. Za odpowiednią synchronizację wyświetlania odpowiada czujnik zamontowany w urządzeniu i wysyłający do mikrokontrolera informacje o położeniu wirującej linijki diodowej. Sensor, podczas przechodzenia linijki przez pewien punkt będzie wysyłał sygnał, który np. może być obsługiwany przez mikrokontroler za pomocą przerwania.

Ze względu na ruch obrotowy linijki diodowej jest konieczne dostarczenie sygnału diodom, które stale zmieniają położenie. Możliwe byłoby umieszczenie elektroniki sterującej w obudowie urządzenia i wysyłanie impulsów za pomocą złącz obrotowych, lecz takie rozwiązanie skomplikowałoby budowę i zwiększyło gabaryty. Zdecydowano, że sterownik diod świecących będzie zamontowany na wirującej części kulistego wyświetlacza i komunikacja z nim będzie odbywała się bezprzewodowo, natomiast za pomocą złącza stykowego (ślizgowego) będzie przekazywane do układu jedynie zasilanie.

Kulisty wyświetlacz będzie potrzebował do pracy zasilania z dwóch źródeł napięcia. Napięcie o wartości +12 V będzie zasilało silnik, natomiast +5 V układy elektroniczne. Część układów scalonych będzie wymagało zasilania napięciem +3,3 V, jednak dla jego wytworzenia zostaną zaadoptowane gotowe moduły ze stabilizatorami napięcia, które zostaną zintegrowane z elektroniką urządzenia. Idealnym, gotowym i niedrogim rozwiązaniem jest zastosowanie zasilacza komputerowego ATX o mocy 400 W, na przykład KY-400W. Wyświetlacz zostanie wyposażony w wyłącznik napędu i sterownika diod LED.

Ze względu na to, że matryca kulistego wyświetlacza opiera się na pracy 48 trójkolorowych diod LED, bardzo trudne byłoby dobranie mikrokontrolera z taką liczbą wyjść, aby każdej diodzie przyporządkować trzy z nich. Także użycie metody demultipleksowania wyjść mikrokontrolera w celu zwiększenia ich liczby byłoby kłopotliwe, gdyż zmuszałoby do połączeń diod w odpowiednich sekwencjach. Znacznie skomplikowałoby to diagnozę uszkodzenia przy ewentualnej usterce. Metoda, która jako jedyna rozwiązuje problem sterowania taką liczbą diod, polega na wysyłaniu danych szeregowo do urządzenia demultipleksującego. Po tym procesie sygnały mają równolegle dotrzeć do diod i zostać przez niewyświetlone. Procesor wykorzystany do realizacji tego zadania został wybrany na podstawie kilku kryteriów. Wybór padł na mikrokontroler firmy ARM. Poza stosunkowo łatwym sposobem obsługi i programowania, ważny był dobór z uwzględnieniem wielkości pamięci Flash. Optymalnym mikrokontrolerem według przeprowadzonej analizy jest STM32F103VET6 z rdzeniem Cortex-M3. Ma on 256 kB pamięci Flash oraz 64 kB pamięci SRAM, co pozwala na przechowywanie i zapisywanie dużej liczby obrazów bez konieczności instalowania pamięci zewnętrznej. Kolejnym atutem wybranego mikrokontrolera jest duża szybkość.

Oprócz mikrokontrolera bardzo ważnym elementem jest układ demultipleksujący dane. Dostępne na rynku układy charakteryzują się różnymi właściwościami, między innymi odmienną szybkością przetwarzania danych i liczbą wyjść. Założono, że rejestr załączający diody LED musi mieć wejście szeregowe, 144 wyjścia równoległe typu zatrzask i mieć możliwość łączenia w łańcuchy. Te warunki zawęziły liczbę możliwych do użycia układów i po analizie właściwości został wybrany układ rejestru przesuwnego z zatrzaskiem 74HC595B1R. Ma on 8 wyjść logicznych i możliwość kaskadowego dołączania kolejnych elementów tego samego typu, przez co liczba wyjść staje się niemal nieograniczona. W projekcie rejestry przesuwne zostaną podzielone na 3 grupy, z których każda ma opowiadać za kolejny kolor wyświetlany przez diody.

Jeszcze jednym bardzo istotnym w modelu kulistego wyświetlacza elementem elektronicznym jest czujnik, który będzie podawał sygnał informujący o położeniu wirującej linijki diodowej. Do jego realizacji wybrano transoptor szczelinowy TCST1103.

W przeprowadzanej analizie i doborze sprzętu nie może zabraknąć elementu, który

Podstawowe informacje:

- Rozdzielczość 48×96 pikseli.
- Kolorowa matryca.
- Wyświetlanie dowolnej, wcześniej przygotowanej grafiki bądź animacji.
- Zdalna obsługa urządzenia wykorzystująca aplikację komputerową pozwalająca na: przeglądanie zapisanych zdjęć z pamięci urządzenia, załączanie bądź wyłączanie poszczególnych składowych kolorów wyświetlacza, przesyłanie zdjęć do urządzenia podczas jego pracy.
- Analiza pracy urządzenia przez zdalny podgląd prędkości obrotowej wirującej linijki diodowej.
- Praca bez ograniczeń czasowych.
- Programowanie mikrokontrolera bez konieczności jego demontażu z urządzenia.
- Debugowanie programu.
- Możliwość rozbudowy modelu i funkcji.

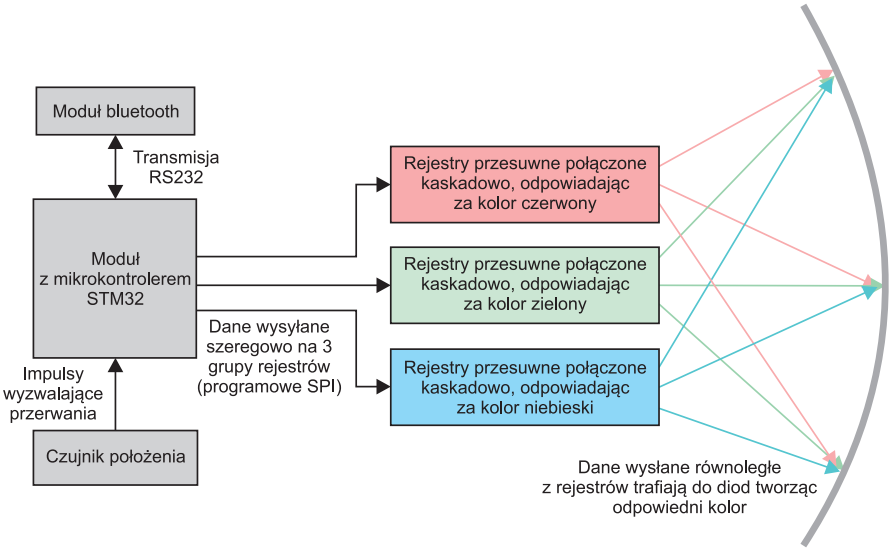
Dodatkowe materiały na FTP:

ftp://ep.com.pl, user: 26526, pass: 841uhx54

- wzory płytek PCB

Projekty pokrewne na FTP:
(wymienione artykuły są w całości dostępne na FTP)

AVT-3060	Wyświetlacz 3D (EdW 5/2013)
AVT-3040	Termometr widmowy (EdW 11/2012)
AVT-1672	Moduł do budowy wyświetlacza tekstowego LED (EP 4/2012)
AVT-2997	Tablica świetlna 32×5 (EdW 1/2012)
AVT-2983	Zegarek naręczny z wyświetlaczem binarno-widmowym (EdW 7/2011)
AVT-5245	Zegar z „widmowym" wyświetlaniem czasu (EP 7/2010)
AVT-5226	Graficzny, modułowy wyświetlacz LED (EP 3/2010)
AVT-2651	Wyświetlacz widmowy (EdW 11/2002)
AVT-3017	Widmo, albo magiczna różdżka EdW 7/2002)



Rysunek 1. Schemat blokowy sterownika diod LED

odpowiada za wyświetlenie obrazu. Diody LED, które będą zastosowane w modelu, muszą charakteryzować się dużą intensywnością świecenia, aby powstały obraz był widoczny i czytelny. Drugim parametrem jest kąt świecenia, który powinien być jak największy, aby wyświetlane informacje były widoczne na całym widocznym obszarze matrycy, a linia wyświetlona przez odpowiednią diodę była możliwie szeroka. Wybór padł na diody LED RGB Super Flux o wymiarach 7,62 mm×7,62 mm w wersji Flat Top o kącie świecenia 160°.

Algorytm sterowania diodami

Schemat blokowy sterownika diod LED pokazano na **rysunku 1**. Działanie sferycznej matrycy polega na wyświetlaniu obrazu zapisanego w pamięci mikrokontrolera. To wyświetlanie musi być zsynchronizowane z czasem trwania pojedynczego obrotu. Wyświetlane dane są wysyłane pakietami. Pakiet zawiera 48 bitów, czyli 6 bajtów danych, które trafią do jednej z trzech grup rejestrów przesuwnych. Po zapełnieniu wszystkich rejestrów danymi nastąpi przepisanie zawartości rejestrów na wyjścia układu scalonego za pomocą pojedynczego impulsu sterującego zatrzaskiem wyjściowym i wyświetlenie informacji przez czas tak długi, jak przewiduje to procedura synchronizująca.

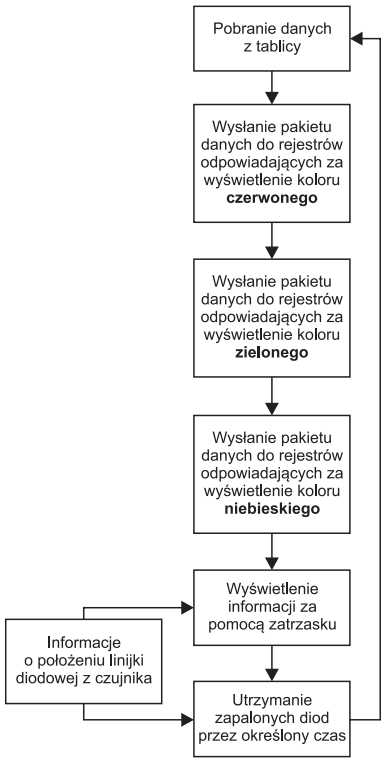
Wykaz elementów

Rezystory: (SMD 1206)
R1: 1 kΩ
R2…R147: 180 Ω
Kondensatory:
C1: 1500 µF/16 V
Półprzewodniki:
IC1…IC18: 74HC595D (SO16)
U$1…U$48: dioda LED RGB
U$50: TCST1103 (czujnik szczelinowy IR)
Inne:
Moduł Bluetooth BTM222
Moduł z mikrokontrolerem STM32F103
Silnik 12 V DC (opis w tekście)

Jeden bajt danych jest zapamiętywany przez jeden rejestr przesuwny aż do momentu odbioru kolejnej porcji danych. Cały proces obejmuje 288 pakietów, którym odpowiada wyświetlenie obrazu na sferycznej matrycy wyświetlacza. Po zakończeniu proces jest powtarzany od początku.

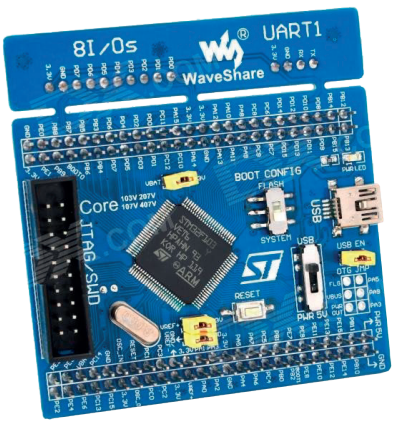
Opracowany sposób wysyłania danych wzorowany na interfejsie SPI opiera się na użyciu jedynie 5 wyjść portu procesora, co umożliwia użycie pozostałych wyprowadzeń do obsługi innych elementów przy ewentualnej rozbudowie modelu. Trzy pierwsze wyjścia mikrokontrolera: PA0, PA2 i PA3 są połączone z wejściami SER pierwszych rejestrów przesuwnych z trzech kolejnych grup odpowiadających za działanie poszczególnych kolorów. Połączenia te odpowiadają za wysyłanie szeregowo danych, binarnych zer i jedynek. Układy 74HC595B1R w swoich grupach są połączone kaskadowo – wyjście Q7' poprzedzającego jest połączone z wejściem SER kolejnego. Wyście PA6 mikrokontrolera jest połączone będzie równolegle z wejściami SRCLK wszystkich rejestrów przesuwnych. Odpowiadać ma za przekazanie sygnału zegarowego, którego zbocze narastające pojawi się po wysłaniu każdej danej z mikrokontrolera. Zadaniem tych impulsów jest przesuwanie danych na kolejne przerzutniki wewnątrz rejestrów przesuwnych. Wyjście PA4 będzie połączone także równolegle z wejściami RCLK wszystkich układów 74HC595B1R. Połączenie to ma odpowiadać za włączenie tzw. zatrzasku, czyli przeniesienia wartości danych z rejestrów pamięci na wyjścia układów.

Na **rysunku 2** pokazano schemat blokowy procedury obsługi kulistego wyświetlacza. Program realizujący postawiony cel musi wykonywać się jak najszybciej. Wymuszają to ramy czasowe oraz duża liczba wysyłanych danych, a także pozostałe funkcje, które muszą być zrealizowane w tym samym czasie. Dla zwiększenia szybkości działania bity danych są przesyłane kolejno na wszystkie trzy grupy rejestrów, po każdym impulsie odpowiadającym za przesunięcie bitów. Po wystawieniu wszystkich danych odpowiadających za powstanie obrazu na jednej linii wystąpi jeden sygnał dla zatrzasku, który steruje wszystkie 18 rejestrów przesuwnych i powoduje zaświecenie się LED. W tym wypadku optymalizacja polega na trzykrotnym zmniejszeniu liczby impulsów odpowiedzialnych za przesunięcie danych oraz użyciu tylko jednego sygnału wyzwalającego na trzy grupy rejestrów sterujących kolorami: czerwonym, zielonym i niebieskim.

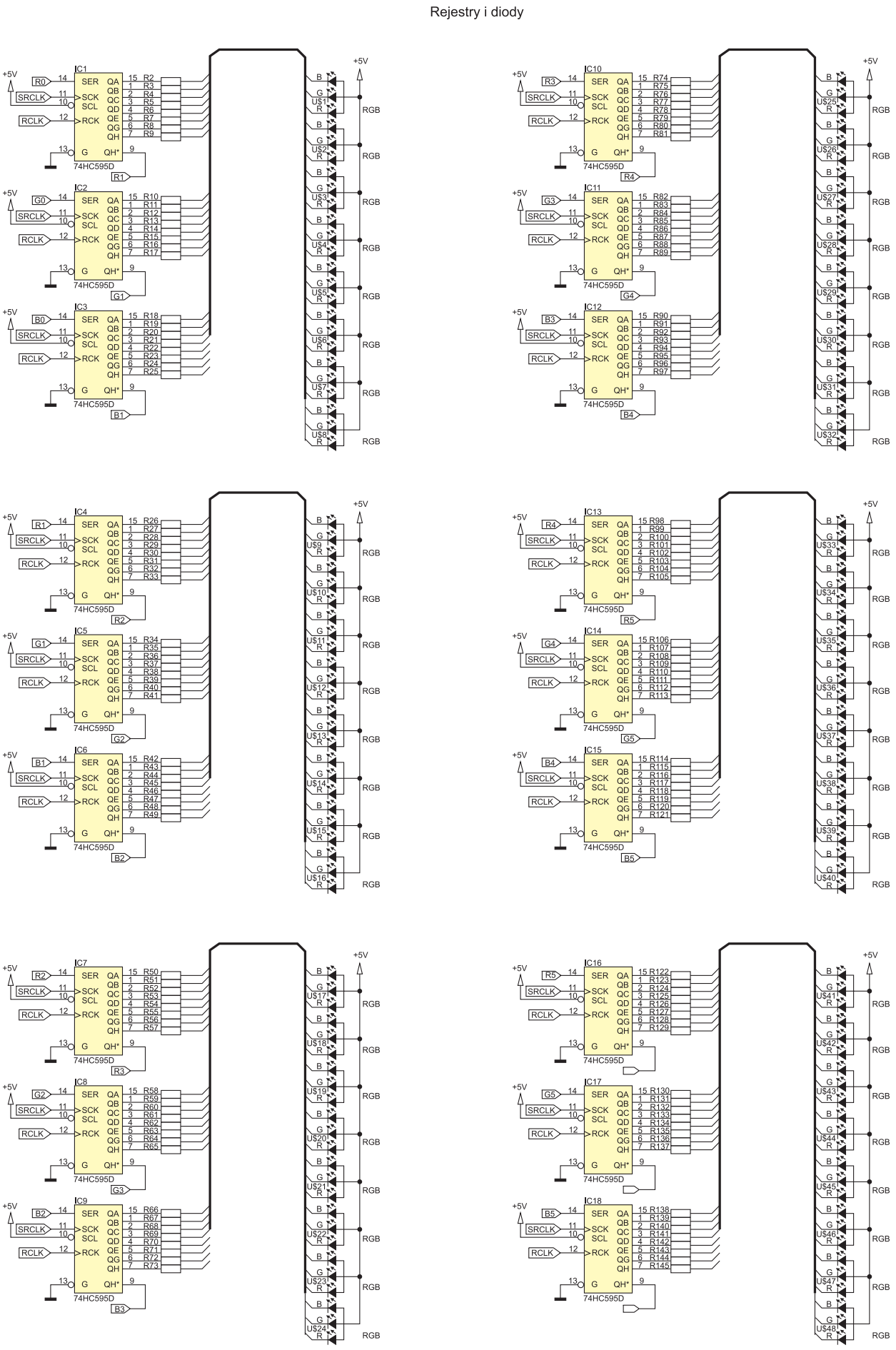


Rysunek 2. Schemat blokowy obsługi wyświetlacza kulistego

Przesyłane dane są generowane na podstawie bitmapy obrazu wcześniej zapisanego w rozdzielczości 48×96 pikseli i poddanego konwersji. Każdy piksel jest składową trzech kolorów i jest reprezentowany przez



Fotografia 3. Płytka uruchomieniowa z STM32



Rysunek 4. Schemat ideowy wyświetlacza. Przedstawiono dwie z sześciu grup rejestrów przesuwnych

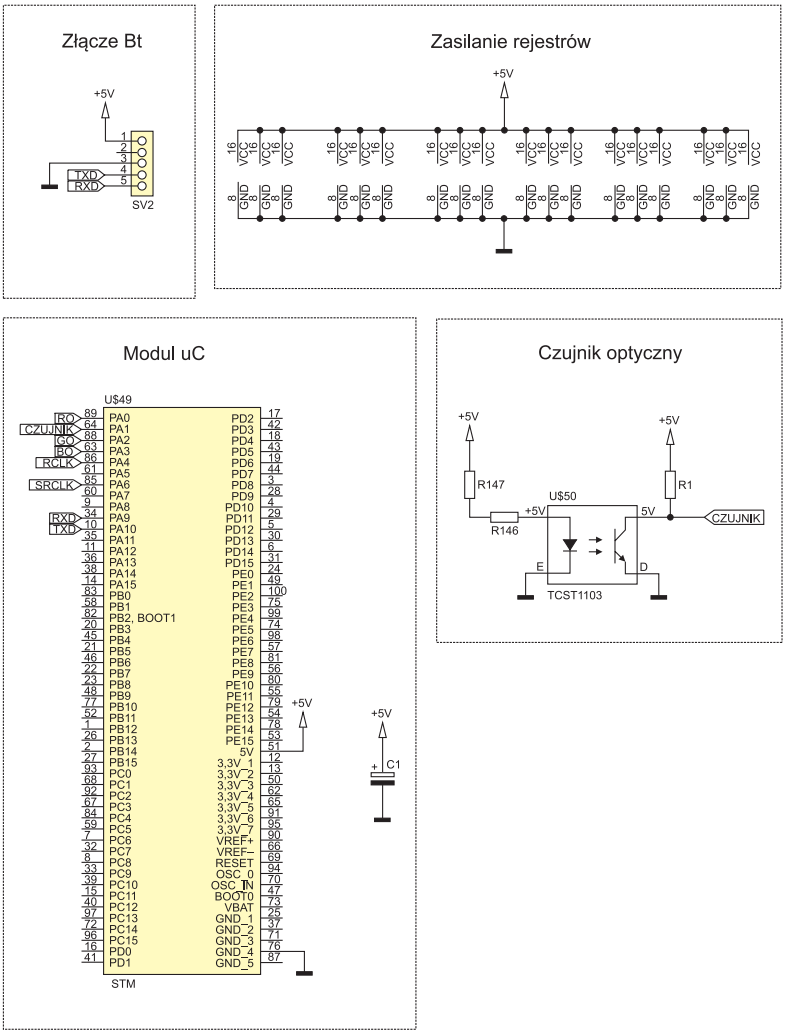
pojedynczy punkt na sferycznej matrycy modelu. Wyświetlenie danych umożliwiają odpowiednio napisane pętle w programie, a za synchronizację jest odpowiedzialne przerwanie od czujnika położenia i odpowiedni podprogram.

Schemat układu sterującego linijką diodową

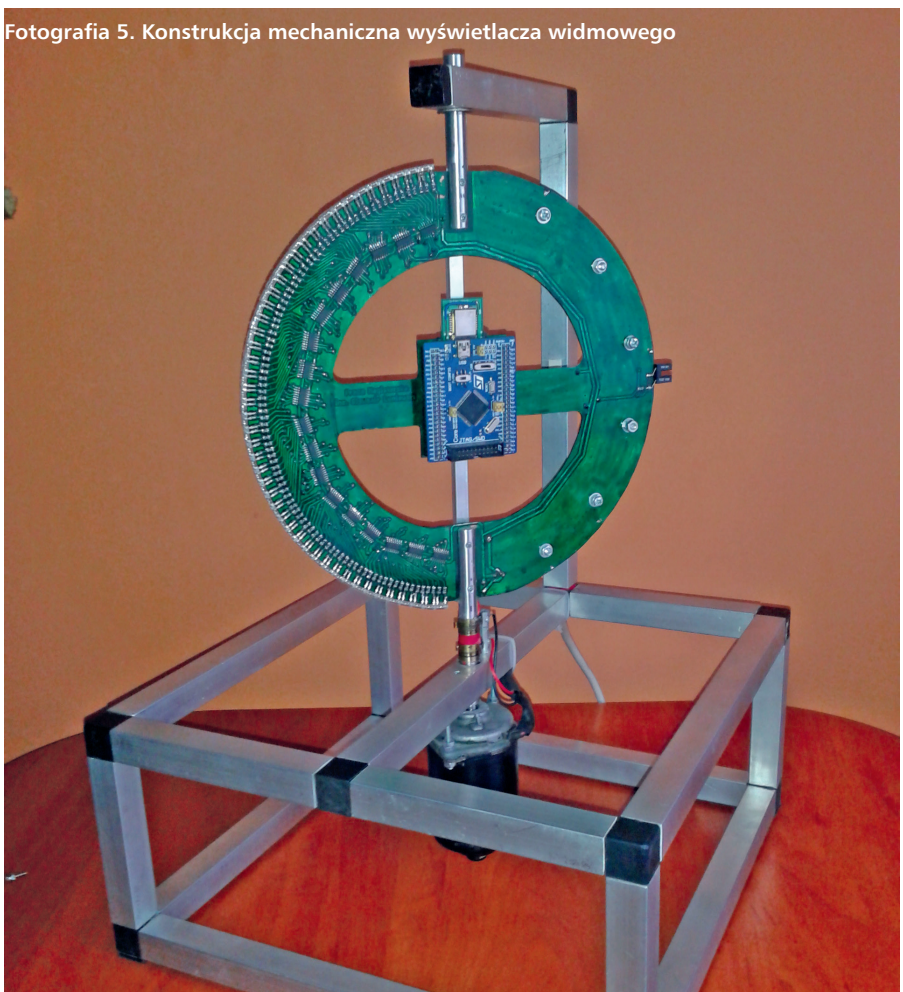
„Sercem" sterownika jest moduł z mikrokontrolerem STM32 taktowanym częstotliwością 72 MHz. Pokazano go na **fotografii 3**. Jednym z jego dwóch podstawowych zadań jest odbieranie impulsów z czujnika i sterowanie rejestrami przesuwnymi. Każdy rejestr kontroluje 8 wyjść. Wszystkie mają wydajność prądową 20 mA, co przy napięciu 5 V jest wystarczające do zasilenia zastosowanych diod LED. Diody są zabezpieczone przed zbyt dużym prądem rezystorami. Informacja o położeniu kątowym jest ustalana na podstawie sygnału z transoptora szczelinowego. Wraz z dodatkowymi elementami generuje on sygnał logiczny o poziomie niskim lub wysokim w zależności od tego, czy zainstalowana przeszkoda znajdzie się w obszarze wiązki światła emitowanej przez diodę i odbieranej przez fototranzystor czujnika.

Drugim zadaniem mikrokontrolera jest wymiana danych z modułem Bluetooth, który umożliwia bezprzewodowe sterowaniem modelem. Moduł jest wyposażony w inwertery z układem Schmitta, zarówno na połączeniu nadawania jak i odbierania danych. Możliwa jest zatem komunikacja z urządzeniem na poziomach logicznych 3,3 V oraz 5 V.

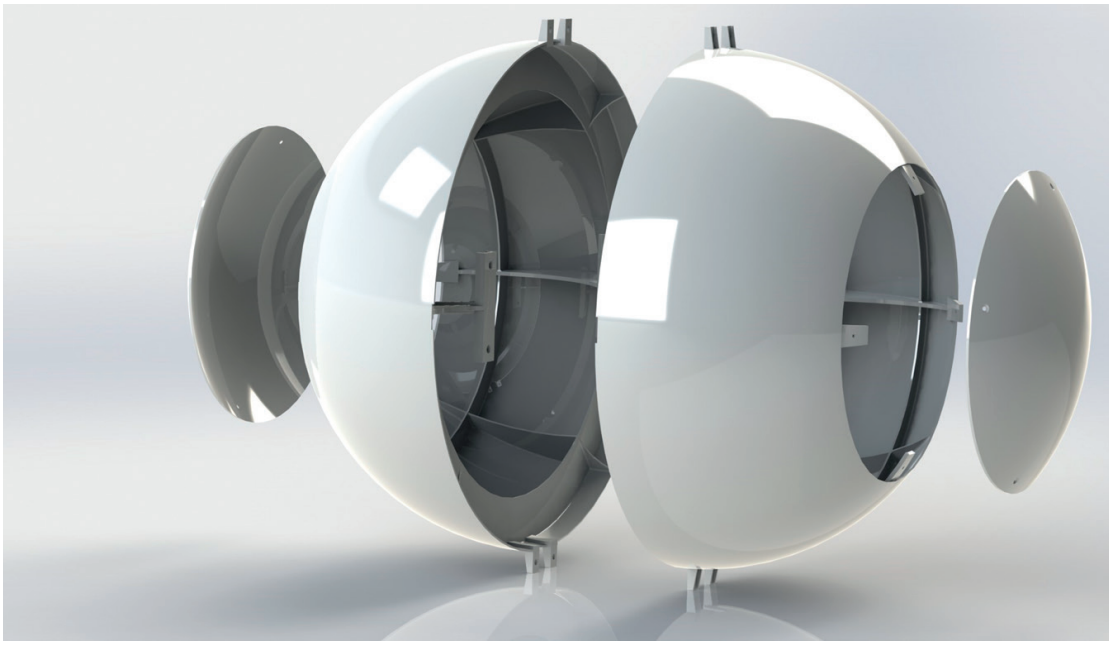
Na **rysunku 4** pokazano schemat ideowy sterownika wyświetlacza. Zawiera on część układów rejestrów przesuwnych wraz z połączeniami. Sterowanie rejestrem przesuwnym 74HC595B1R odbywa się poprzez 4 wejścia (dwa zegarowe, jedno danych i jedno zerowania), nie licząc wyprowadzeń zasilania. Wejście SCLR (zerowanie rejestru) w trybie pracy układu musi być ustawione. Poziom niski na tym doprowadzeniu służy jedynie do przesunięcia rejestru. W projekcie na to wejście podano na stałe logiczną jedynkę. Doprowadzenie G pozwala na odbieranie danych i jest aktywne przy poziomie niskim, toteż połączono je bezpośrednio z masą układu. Kolejne wejścia sterowane są w sposób dynamiczny za pomocą mikrokontrolera. Na wejścia SER (szeregowego wprowadzenia danych) pierwszych rejestrów z trzech grup opowiadających za kolory RGB, jest podawany sygnał z mikroprocesora. Rejestry w grupach połączone są kaskadowo i wejścia SER kolejnych rejestrów połączone są z wyjściami QH' (wyjścia szeregowego danych) poprzednich. Kontroler wysyła dane trzema pakietami 48-bitowymi do 6 rejestrów w każdej grupie. Wysyłane dane przesuwane są na kolejne przerzutniki wewnątrz rejestrów i za



Rysunek 4. c.d.

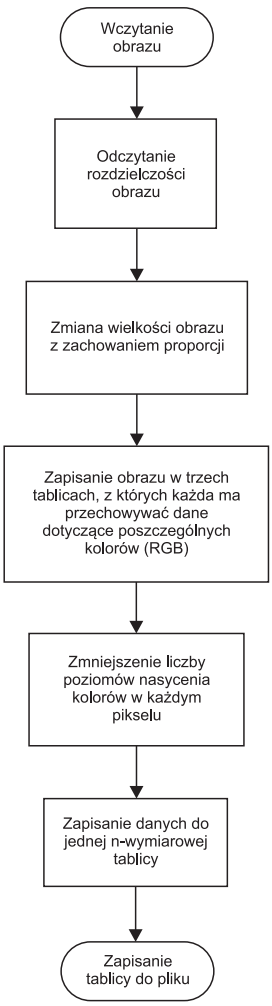


Fotografia 5. Konstrukcja mechaniczna wyświetlacza widmowego



Rysunek 6. Aerodynamiczna obudowa wirującej części wyświetlacza

pomocą wyjścia QH' do kolejnych. Wygląda to w ten sposób, że pierwszy wysłany bit trafia na QH (8 wyjście) 6 rejestru, a 48 bit zapisze się na QA (1 wyście) 1 rejestru. Przesuwanie danych jest możliwe dzięki in-



Rysunek 7. Algorytm działania aplikacji przetwarzającej obraz w programie Matlab

nemu sygnałowi z kontrolera, obsługującego wejścia SRCLK wszystkich 74HC595B1R.

Wejście SRCLK jest sygnałem zegarowym rejestru przesuwnego. Gdy jego poziom zmieni się z niskiego na wysoki, to zawartość rejestru przesuwnego jest przesuwana o 1 bit w prawo, natomiast na pozycji najmłodszego bitu jest zapamiętywany poziom występujący na wejściu SER. Wejście RCLK steruje wyprowadzaniem danych z rejestru przesuwnego na wyjścia równoległe układu scalonego. Musi ono być wyzerowana podczas zapisywania informacji do rejestru. Poziom wysoki powoduje zapisanie danych do rejestru pamięci, po czym stan wyjściowy pojawia się na pinach QA…QH wszystkich rejestrów.

Konstrukcja mechaniczna

Model mechaniczny – pokazano go na **fotografii 5** – składa się z dwóch głównych części. Pierwszą jest statyw urządzenia, który wykonano z rur aluminiowych o przekroju kwadratowym. Do połączenia elementów metalowych zastosowano łączniki kątowe wytworzone z tworzywa sztucznego. Szkielet modelu składa się z podstawy oraz obręczy. Podstawa jest prostopadłościanem, w którym zamontowano napęd wirującej linijki diodowej oraz zasilacz. Część ta odpowiada za utrzymanie modelu w odpowiedniej pozycji oraz za tłumienie drgań. Górna obręcz szkieletu podtrzymuje część ruchomą wyświetlacza. Zamontowano na niej tuleję ślizgową minimalizująca tarcie linijki diodowej oraz element będący przesłoną dla czujnika optycznego. Drugim głównym elementem części mechanicznej modelu wyświetlacza jest wirująca ramka z linijką diodową. Opiera się ona na specjalnej obręczy wyposażonej w tuleję i jest wprawiona w ruch obrotowy za pomocą silnika DC. Jej zadaniem jest przemieszczanie diod po kole z określoną prędkością. Część wirująca jest odpowiedzialna również za utrzymanie elektroniki układu, a tworzy ją specjalnie zaprojektowana płyta PCB.

Aby powstał efekt widmowy reprezentujący obraz o dobrej jakości o liczbie kolorów większej niż podstawowa gama 6 barw, konieczne jest uzyskanie dużej prędkości obrotowej linijki diodowej. Skonstruowana wirująca cześć kulistego wyświetlacza ma spore opory powietrza, co przeciwdziała ruchowi. Dla zmniejszenia oporów powietrza i podwyższenia szybkości wirowania zaprojektowano czteroelementową obudowę o kulistym kształcie, która została wydrukowana na drukarce 3D z materiału ABS. Płaszcz obudowy jest cienki, co przekłada się na jej niewielki ciężar, zaś konstrukcję usztywnią odpowiednio rozlokowane żebra. Jej projekt pokazano na **rysunku 6**.

Do zasilania wirującej linijki zastosowano złącze **ślizgowe składające się z** dwóch pierścieni. Ze względu na możliwość wystąpienia przerwy kontaktu w złączu obrotowym przez czas rzędu mikrosekund zastosowano podwójny zestaw szczotek na każdy pierścień, a także umieszczono w wirującej części urządzenia i połączono równolegle ze źródłem zasilania kondensator **o dużej pojemności (1,5 mF)**. Zapobiega on zaburzeniom pracy mikrokontrolera i psuciu efektu widmowego.

Algorytm sterowania bezprzewodowego

Prototyp wyposażono we wstępnie skonfigurowany moduł Bluetooth BTM222. Umożliwiają to komendy AT, za pomocą których należy: wyłączyć echo i odpowiedzi AT z modułu, a także ustawić prędkość transmisji, nazwę oraz tryb pracy jako sla-

ve. Po wykonanej konfiguracji moduł jest gotowy do pracy, a łącze radiowe Bluetooth staje się odpowiednikiem kablowego połączenia z interfejsem RS232. W mikrokontrolerze transmisję umożliwia uniwersalny port USART. Do realizacji połączenia użyto jedynie dwóch linii. Nadawanie – oznaczone TxD – jest połączone z wyprowadzeniem PA9 mikrokontrolera, zaś odbiór danych – RxD – jest przyporządkowany jest PA10 (USART1 mikrokontrolera STM32F103).

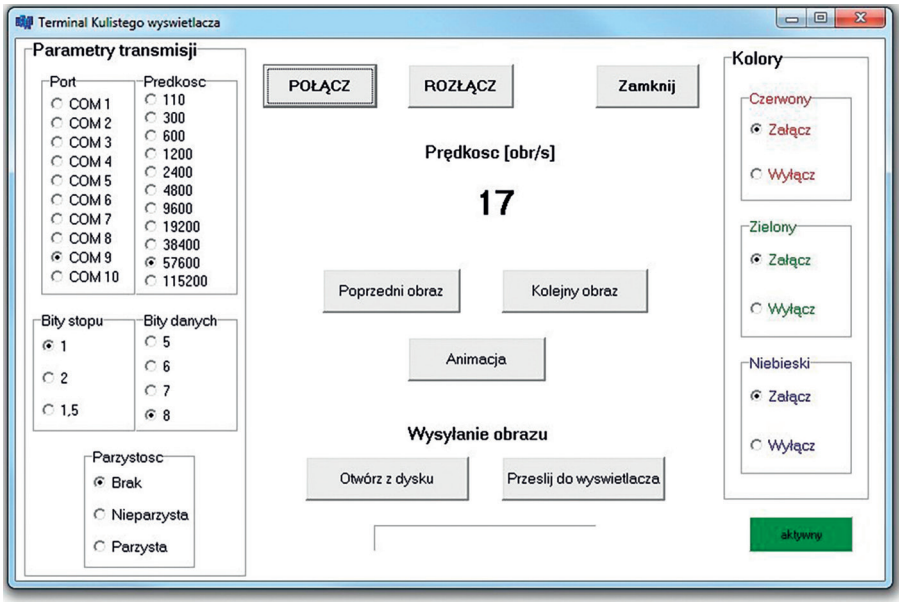
Obsługa transmisji jest realizowana za pomocą przerwań. Nie jest używana sprzętowa kontrola przepływu za pomocą sygnałów CTS i RTS.

Przygotowanie wyświetlanych obrazów i „mechaniczny PWM"

Właściwe działanie kulistego wyświetlacza wymaga odpowiednio przygotowanych danych, które zostaną zobrazowane za pomocą diod na widmowej matrycy. W tym celu została przygotowano aplikację w środowisku Matlab, której zadaniem jest konwertowanie obrazu w formacie BMP na postać wewnętrzną, która może sterować diodami LED. Algorytm funkcjonowania tej aplikacji pokazano na **rysunku 7**, natomiast wygląd jej okna na **rysunku 8**. Pierwszym etapem procesu przetwarzania danych jest wczytanie obrazu zapisanego w formacie BMP z wykorzystaniem funkcji *imread*. Struktura formatu zawiera nagłówek pliku, nagłówek mapy bitowej, tablicę kolorów składającą się z rekordów RGB oraz ciąg bajtów zawierających dane o obrazie. Z punktu widzenia programu wykorzystywane będą jedynie tablice kolorów, z których w drugim etapie procesu przetwarzania zostanie odczytany rozmiar obrazu.

Rozdzielczość obrazu może być dowolna. Ze względu na to, że wyświetlacz kulisty ma rozdzielczość 96×48 pikseli, aplikacja musi dopasować obraz do wielkości matrycy. Zmienia go do pożądanych wymiarów za pomocą funkcji *imresize*, której argumentami są liczba pikseli w poziomie i pionie oraz metoda interpolacji wartości nasycenia barw. W celu zachowania proporcji rozdzielczość pionowa jest ustawiona na 48, natomiast pionowa jest obliczana na podstawia formatu analizowanego obrazu. Z najbardziej znanych metod interpolacji, jakimi są metoda najbliższego sąsiada, metoda interpolacji dwuliniowej oraz metoda interpolacji dwukwadratowej, wybrano pierwszą ze względu na stosunkowo prosty algorytm i ostry obraz wynikowy. Metoda najbliższego sąsiada (w Matlabie – bilinear) jest metodą, dla której nową wartość piksela określa się biorąc pod uwagę wartość pikseli w jego sąsiedztwie.

Kolejnym etapem aplikacji jest zapisanie przetwarzanego obrazu w trzech tablicach, z których każda przechowuje składową odpowiadającą za kolory: czerwony, zielony i niebieski. Tablice te standardowo mające dla



Rysunek 8. Wygląd okna aplikacji przetwarzającej obraz

każdego piksela 256 poziomów jasności muszą zostać przetworzone do 4 poziomów. Aby uzyskać odpowiedni odcień danego koloru wyświetlacz będzie zaświecał i gasił diodę w momencie jej jednakowego położenia w kolejnych obrotach wirującej ramki. Intensywność barwy będzie proporcjonalna do czasu świecenia diody względem czasu działania całego układu. Dla przykładu, na drugim poziomie intensywności dioda będzie zapalana w co drugim obrocie. Metodę można nazwać mechanicznym PWM, czyli mechaniczną modulacją szerokości impulsów o stałej częstotliwości i amplitudzie. Częstotliwość będzie proporcjonalna do prędkości wirowania linijki diodowej, natomiast amplituda odpowiada napięciu zasilania diod. Sposób ten umożliwi uzyskanie obrazu o 64 kolorowej palecie barw.

Ostatnim etapem programu przetwarzającego obraz jest zapisanie wyodrębnionych we wcześniejszym kroku tablic w jednej, 4-wymiarowej tablicy o formacie *dane[4][3][96][6]*. Definiuje ona 4 elementy odpowiadające za odcień, w której każdy element składa się z tablicy 3 elementów odpowiadający za kolor, a każdy element tej tablicy składa się dwuwymiarowej tablicy przechowującej dane położenia pikseli. Przygotowana tablica zostaje zapisana do pliku z rozszerzeniem *.txt*. Po przesłaniu danych obrazu do pamięci mikrokontrolera, program sterujący odczytuje informacje i wyświetla je na matrycy kulistego wyświetlacza.

Program sterujący mikrokontrolerem

Program napisano w języku C za pomocą środowiska TrueSTUDIO firmy Atollic. Wykorzystano przy tym programator/debugger zgodny z ST-Link oraz bibliotekę *STM32F10x Standard Peripherals Library v3.4.0* udostępnioną przez firmę STMicroelectronics. Za jej pomocą skonfi-

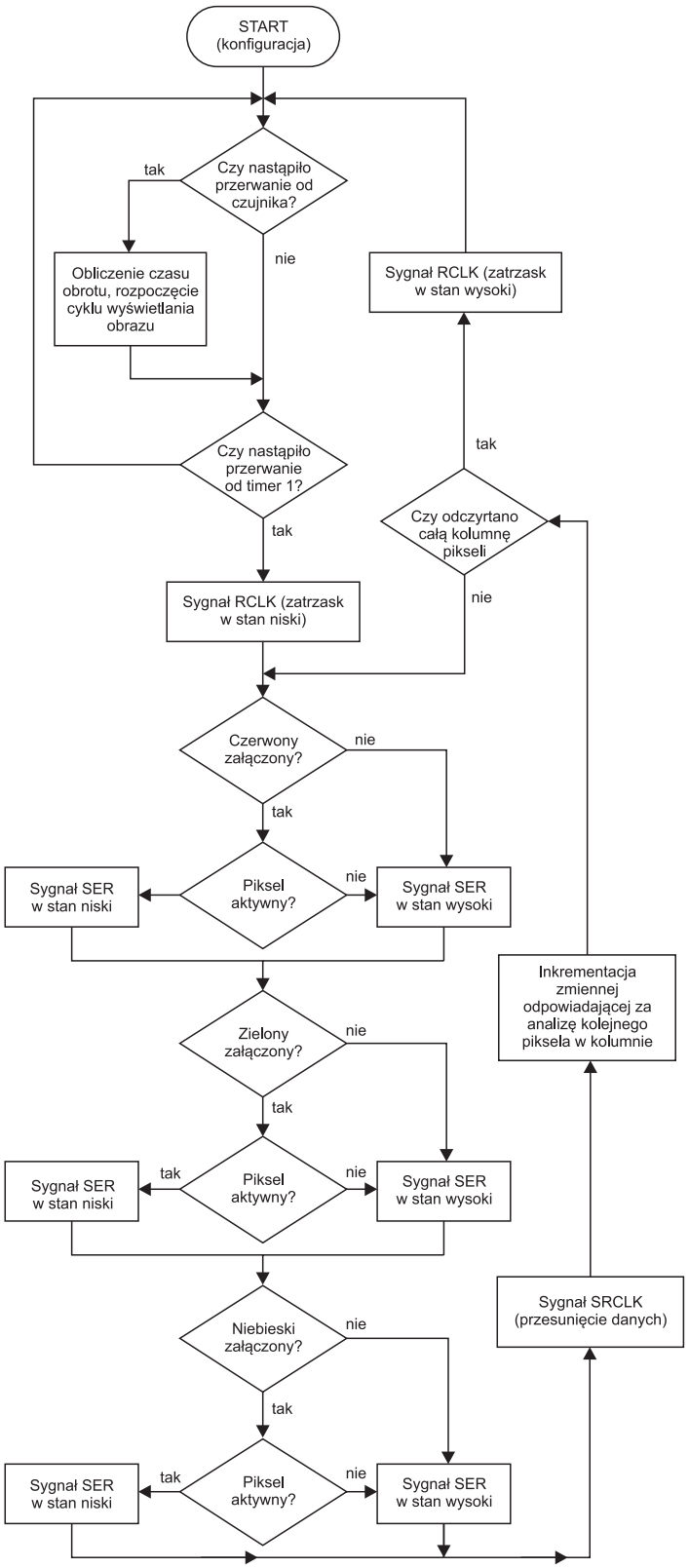
Listing 1. Konfiguracja sygnału taktującego

```
ErrorStatus HSEStartUpStatus;
// Reset ustawień RCC, przywrócenie wartości domyślnych
RCC_DeInit();
// Włączenie HSE
RCC_HSEConfig(RCC_HSE_ON);
// Oczekiwanie aż HSE będzie gotowy
HSEStartUpStatus = RCC_WaitForHSEStartUp();
if(HSEStartUpStatus == SUCCESS)
  {
   FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
// zwłoka dla pamięci Flash
   FLASH_SetLatency(FLASH_Latency_2);
// Ustalenie źródeł i częstotliwości taktowań poszczególnych magistral
   RCC_HCLKConfig(RCC_SYSCLK_Div1);
   RCC_PCLK2Config(RCC_HCLK_Div1);
   RCC_PCLK1Config(RCC_HCLK_Div2);
// PLLCLK = 8MHz * 9 = 72 MHz
   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
// Wlączenie PLL
   RCC_PLLCmd(ENABLE);
// Oczekiwanie aż PLL poprawnie sie uruchomi
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
// PLL będzie źródłem sygnału zegarowego
   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
// Oczekiwanie aż PLL będzie sygnałem zegarowym systemu
   while(RCC_GetSYSCLKSource() != 0x08);
// Włączenie taktowanie GPIOB i TIM2
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1 | RCC_APB2Periph_GPIOA |
   RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO, ENABLE);
   }
```

gurowano odpowiednie układy peryferyjne mikrokontrolera. Proces konfiguracji odbywa się tuż po zasileniu modułu w głównej funkcji programu *main()*.

Algorytm działania programu sterującego mikrokontrolerem pokazano na **rysunku 9**. Na początku zostają odpowiednio ustawione rejestry odpowiadające za sygnał taktujący mikrokontrolerem – **listing 1**. W programie sygnałem taktującym (zegarowym) jest sygnał pobierany z powielacza wykonanego na wewnętrznym generatorze PLL, który powiela częstotliwość rezonansową rezonatora HSE. Do wyprowadzeń generatora przebiegów szybkich jest dołączony rezonator o częstotliwości 8 MHz. Pozwala to na osiągniecie maksymalnej, katalogowej częstotliwości taktowania rdzenia mikrokontrolera 72 MHz.

W dalszej części – **listing 2** – następuje konfigurowanie wejść i wyjść. Linie portu A: PA0…PA4, PA6 pracują jako wyjścia wysyłając dane do rejestrów przesuwnych. Dodatkowo, PA9 obsługuje wysyłanie danych od modułu Bluetooth. Wejściami są: sygnał z czujnika doprowadzony do linii PA1 oraz kanał odbioru danych USART do linii PA10. Właściwa konfiguracja następuje po włączeniu sygnału zegarowego dla portu I/O za pomocą wywołania funkcji *RCC_APB2PeriphClockCmd()*. Następnym etapem jest ustalenie, jaką rolę mają pełnić konfigurowane wyprowadzenia oraz szybkość ich pracy.



Rysunek 9. Algorytm sterowania procesem wyświetlania danych

W dalszej części następuje konfiguracja sprzętowego kontrolera przerwań NVIC (**listing 3**), w której są ustalane priorytety przerwań oraz jest konfigurowany Timer 1 (**listing 4**). Jego zadaniem jest wywoływanie przerwania realizującego wysyłanie danych do rejestrów przesuwnych. Przerwanie zostaje wywoływane w okresie 1/96 czasu trwania obrotu linijki diodowej. Czas ten zostaje mierzony za pomocą drugiego licznika (Timer 2). Przygotowanie licznika do pracy składa się z dwóch etapów. W pierwszym następuje konfigurowanie układu podstawy czasu, do której należy ustalenie wartości wpisywanych do rejestru przepełniania i preskalera, a także ustalenie trybu pracy oraz przez jaką wartość będzie dzielona częstotliwość sygnału zegarowego timera. W drugim etapie zostaje skonfigurowany kanał licznika. Z czterech dostępnych użyty będzie kanał pierwszy, pracujący w trybie *TIM_OCMode_Timing*. Opcja oznacza, że urządzenie peryferyjne TIM1 będzie generowało przerwanie w momencie, gdy timer doliczy do wartości równej zawartości pola *TIM_Pulse* struktury inicjującej. Poziomem aktywnym będzie poziom wysoki, wybrany w polu *TIM_OCPolarity*. Ostatnim ustawianym polem jest *TIM_OutputState*, któremu nadano wartość *TIM_OutputState_Enable*. Sygnał z timera będzie wykorzystywany do sterowanie innym elementem systemu, kontrolerem przerwań.

Zadaniem drugiego licznika Timer 2 będzie mierzenie okresu sygnału pochodzącego od czujnika (**listing 5**). W tym trybie licznik zaczyna liczyć, gdy wystąpi zbocze narastające na wejściu wyzwalającym. Dzięki temu można mierzyć częstotliwość sygnału wejściowego oraz odstępy czasu pomiędzy kolejnymi obrotami linijki diodowej. Konfiguracja timera w pierwszym etapie polega na ustawienia podstawy czasu. Zachodzi ona analogicznie do ustawień w liczniku 1. W drugiej części zaś następuje ustawienie timera do realizacji funkcji wyzwalania/zatrzymania sygnałem zewnętrznym za pomocą funkcji *TIM_ICInit()*. Jest ona rozbudowana i daje możliwość wyboru kanału oraz rodzaju zbocza, na które reagować ma układ. Ze względu na to, że timer jest wyposażony w preskaler i filtr wejściowy, konfiguracja daje możliwość ustalenia w jaki sposób mają one pracować. W programie preskaler nie zostaje wykorzystany, ponieważ zbocze narastające wywołuje przerwanie.

Konfiguracja interfejsu UART (**listing 6**) polega na ustawieniu podstawowych parametrów transmisji. Proces rozpoczyna się od ustawieniu prędkości transmisji na wartość 57600 kbps w polu *USART_BaudRate* struktury inicjującej. Ramce danych zostaje przypisana długość 8 bitów bez bitu parzystości i z jednym bitem stopu. Pole *USART_HardwareFlowControl_None* wyłącza sprzętową kontrolę przepływu danych (sygnały RTS i CTS), ponieważ moduł Bluetooth wyposażony jest jedynie w sygnały Tx i Rx. Ostatnią czynnością inicjalizacji jest zadeklarowanie nadawania i odbioru danych przez interfejs.

Po zakończeniu procesu konfiguracji układów peryferyjnych program przechodzi do realizacji dalszej części funkcji głównej, oraz funkcji wywoływanych przerwaniami (**listing 7**). Podprogram, którego zadaniem jest obsługa zdarzeń wywołanych sygnałem z czujnika przedstawiony jest na **listingu 8**.

Działanie układu polega na ciągłym monitorowaniu linii PA1, do której dołączono czujnik optyczny. Licznik pracuje bez przerwy, a po wykryciu zbocza narastającego na linii PA1 jest wywoływana funkcja obsługi przerwania. Jej zadaniem jest wyzerowanie flagi przerwania, odczytanie przechwyconej wartości licznika kanału 2, oraz wyzerowanie licznika Timer 2. Przechwycona wartość licznika zostaje podzielona przez 96 (liczbę linii obrazu do wyświetlenia) i przekazana w postaci zmiennej globalnej dla licznika 1. Dodatkowo, w podprogramie następuje inkrementacja zmiennej *a*, która powoduje wyświetlenie innej tablicy obrazu danego odcienia po każdym obrocie. Wyzerowana jest także wartość zmiennej *c*, która w czterowymiarowej tablicy obrazu np. *grafika[a][b][c][d]*, odpowiada za rozdzielczość poziomą. Ostatnią częścią funkcji przerwania jest przeliczenie i wysłanie liczby obrotów linijki diodowej za pomocą UART do stacji zdalnego sterowania.

Kolejną częścią kodu programu jest funkcja wywoływana za pomocą przerwania licznika 1. Wywołanie następuje cyklicznie co 1/96 czasu trwania obrotu wirującej linijki diodowej, czyli czas obliczony i przechowywany w zmiennej *CC1*. Zadaniem podprogramu jest sprawdzenie co jest źródłem przerwania i dla którego kanału, wyzerowanie bitu przerwania, ustawienie rejestru przepełnienia, a także realizacja wyświetlenia jednej linii obrazu. Na listingu 7 zostało przedstawione ciało podprogramu.

Proces wysyłania danych opiera się na działaniu funkcji przerwań oraz pętlach *for*. Pierwsza z pętli znajdująca się wewnątrz pozostałych zajmuje się odczytaniem ciągu 8 bitów ze stałej zapisanej w tablicy danych w postaci szesnastkowej. Następnie przekazuje odpowiednie poziomy logiczne na wyjście: PA0 dla koloru czerwonego, PA2 dla koloru zielonego i PA3 dla koloru niebieskiego. W konsekwencji zapełnią one jeden z rejestrów przesuwnych w każdej grupie. W pętli po każdym przejściu zostaje zmieniona maska, przez przesuniecie jedynki o jedno miejsce w prawo, począwszy od stanu 10000000. Bierze ona udział w działaniu logicznego iloczynu wraz ze stałą z tablicy *obraz[a][b][c][d]*. Dalej, zmienna wynikowa zastaje porównana z maską. Jeśli wynikiem tych operacji jest 1, na wyjście zostaje wyzerowane, jeśli zaś „0” to ustawione. Aby dane zostały przesunięte wewnątrz rejestrów i zrobiły miejsce kolejnym, po każdym przejściu następuje również wysłanie jedynki logicznej za pomocą wyprowadzenia PA6.

Pętla wykonuje się 8 razy, a więc tyle, ile bitów jest do wysłania dla każdej grupy rejestrów z jednej stałej tablicowej, a następnie program z niej wychodzi. Dodatkowo, w pętli jest sprawdzany stan załączenia poszczególnych kolorów przed każdym wysłanym bitem. Załączenie bądź wyłączenie kolorów dokonuje się zdalnie. Ze względu na to, że pętle są umieszczone jedna w drugiej, automatycznie po wyjściu realizowane są rozkazy z pętli zewnętrznej, która wymusza zmianę stałej z tablicy na kolejną w tym samym wierszu. Po wysłaniu wszystkich stałych

Listing 2. Konfiguracja wejść i wyjść

```
GPIO_InitTypeDef GPIO_InitStructure;
// PA1 wejściem drugiego kanału TIM2, wejście czujnika
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, & GPIO_InitStructure);
// Konfiguracja wyjść sterujacych rejestrami przesuwnymi
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_
Pin_4 | GPIO_Pin_6 ;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, & GPIO_InitStructure);
// Konfiguracja PA9 jako Tx - nadawanie danych do bluetooth
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
// Konfiguracja PA10 jako Rx-odbiór danych od bluetooth
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Listing 3. Konfiguracja kontrolera przerwań

```
NVIC_InitTypeDef NVIC_InitStructure;
// Właczenie przerwania od TIM1
NVIC_InitStructure.NVIC_IRQChannel = TIM1_CC_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
// Właczenie przerwania od TIM2
NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
// Włączenie przerwania od USART1
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Listing 4. Konfiguracja licznika 1

```
// Konfiguracja układu podstawy czasu
TIM_TimeBaseStructure.TIM_Period = 65535;
TIM_TimeBaseStructure.TIM_Prescaler = 360; //fclk = 72MHz/360 = 200kHz
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
// Konfiguracja kanału 1
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CC1;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC1Init(TIM1, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Disable);
// Włączenie przerwań od kanałów
TIM_ITConfig(TIM1, TIM_IT_CC1, ENABLE);
// Włączenie timera
TIM_Cmd(TIM1, ENABLE);
```

Listing 5. Konfiguracja interfejsu USART

```
USART_InitTypeDef USART_InitStructure;
// Konfiguracja USART
USART_InitStructure.USART_BaudRate = 57600;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_
None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART1, &USART_InitStructure);
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
USART_Cmd(USART1, ENABLE);
```

Listing 6. Konfiguracja licznika 2
```
// Konfiguracja układu podstawy czasu
TIM_TimeBaseStructure.TIM_Period = 65535;
TIM_TimeBaseStructure.TIM_Prescaler = 360; //fclk = 72MHz/360 = 200kHz
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
// Konfiguracja kanalu 2 jako „input capture”
TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
// Reakcja na zbocze narastające
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
// Nie uwzględnianie preskalera, każde zdarzenie jest uwzględniane
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
// Nie używanie filtra
TIM_ICInitStructure.TIM_ICFilter = 0;
TIM_ICInit(TIM2, &TIM_ICInitStructure);
// Włączenie przerwania od TIM2
TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
// Włączenie timera
TIM_Cmd(TIM2, ENABLE);
```

Listing 7. Funkcja obsługi przerwania od TIMER 1
```
void TIM1_CC_IRQHandler(void)
{
//sprawdzenie statusów przerwań od poszczególnych kanałów
  if (TIM_GetITStatus(TIM1, TIM_IT_CC1) != RESET)
  {
  // Wyzerowanie flagi przerwania
    TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
  //Wyzerowanie licznika TIM1
    TIM_SetCompare1(TIM1, TIM_GetCapture1(TIM1) + CC1);
  //Wyświetl kolejną linię obrazu przy następnym przerwaniu
    c++;
  //--------WYŚWIELTENIE 1 LINII OBRAZU-------
    GPIO_ResetBits(GPIOA, GPIO_Pin_4);     //Poziom niski na RCLK
    for (d=0 ; d<6 ; d++)
    {
        for (k=0 ; k<8 ; k++)
        {
            z = (128)>>k;
            if (czerwony)  //sprawdź czy załączony kolor
            {
                if ((mapa[a][0][c][d] & z) == z)  //czerwony
                GPIO_ResetBits(GPIOA, GPIO_Pin_0);
                else GPIO_SetBits(GPIOA, GPIO_Pin_0);
            }
            else GPIO_SetBits(GPIOA, GPIO_Pin_0);
            if (zielony)  //sprawdź czy załączony kolor
            {
                if ((mapa[a][1][c][d] & z) == z)  //zielony
                GPIO_ResetBits(GPIOA, GPIO_Pin_2);
                else GPIO_SetBits(GPIOA, GPIO_Pin_2);
            }
            else GPIO_SetBits(GPIOA, GPIO_Pin_2);
            if (niebieski)  //sprawdź czy załączony kolor
            {
                if ((mapa[a][2][c][d] & z) == z)    //niebieski
                GPIO_ResetBits(GPIOA, GPIO_Pin_3);
                else GPIO_SetBits(GPIOA, GPIO_Pin_3);
            }
            else GPIO_SetBits(GPIOA, GPIO_Pin_3);
            //Poziom wysoki na SRCLK
            GPIO_SetBits(GPIOA, GPIO_Pin_6);
            //Poziom niski na SRCLK
            GPIO_ResetBits(GPIOA, GPIO_Pin_6);
        }
    }
    //Poziom wysoki na RCLK
    GPIO_SetBits(GPIOA, GPIO_Pin_4);
  }
}
```

Listing 8. Funkcja obsługi przerwania pochodzącego od czujnika
```
void TIM2_IRQHandler(void)
{
  // Jezeli przerwanie od kanalu 2
  if (TIM_GetITStatus(TIM2, TIM_IT_CC2) == SET)
  {
    // Wyzerowanie flagi przerwania
    TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
    // Odczytanie wartości rejestru CCR kanału 2
    cnt = TIM_GetCapture2(TIM2);
    //Wartość do jakiej zliczać ma timer1
    CC1 = cnt / 96 ;
    //zmień odcień koloru po przejściu czujnika
    a++;
    if (a>3) a=0;   //po przejściu czujnika rozpocznij wyświetlanie
    //obrazu od pierwszej linii
    c=0;
    // Zeruj licznik TIM2
    TIM_SetCounter(TIM2, 0);
    //Wyślij wartość ilości obrotów przez UART
    TxBuf[0]=(uint8_t)(200000/cnt);
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
  }
}
```

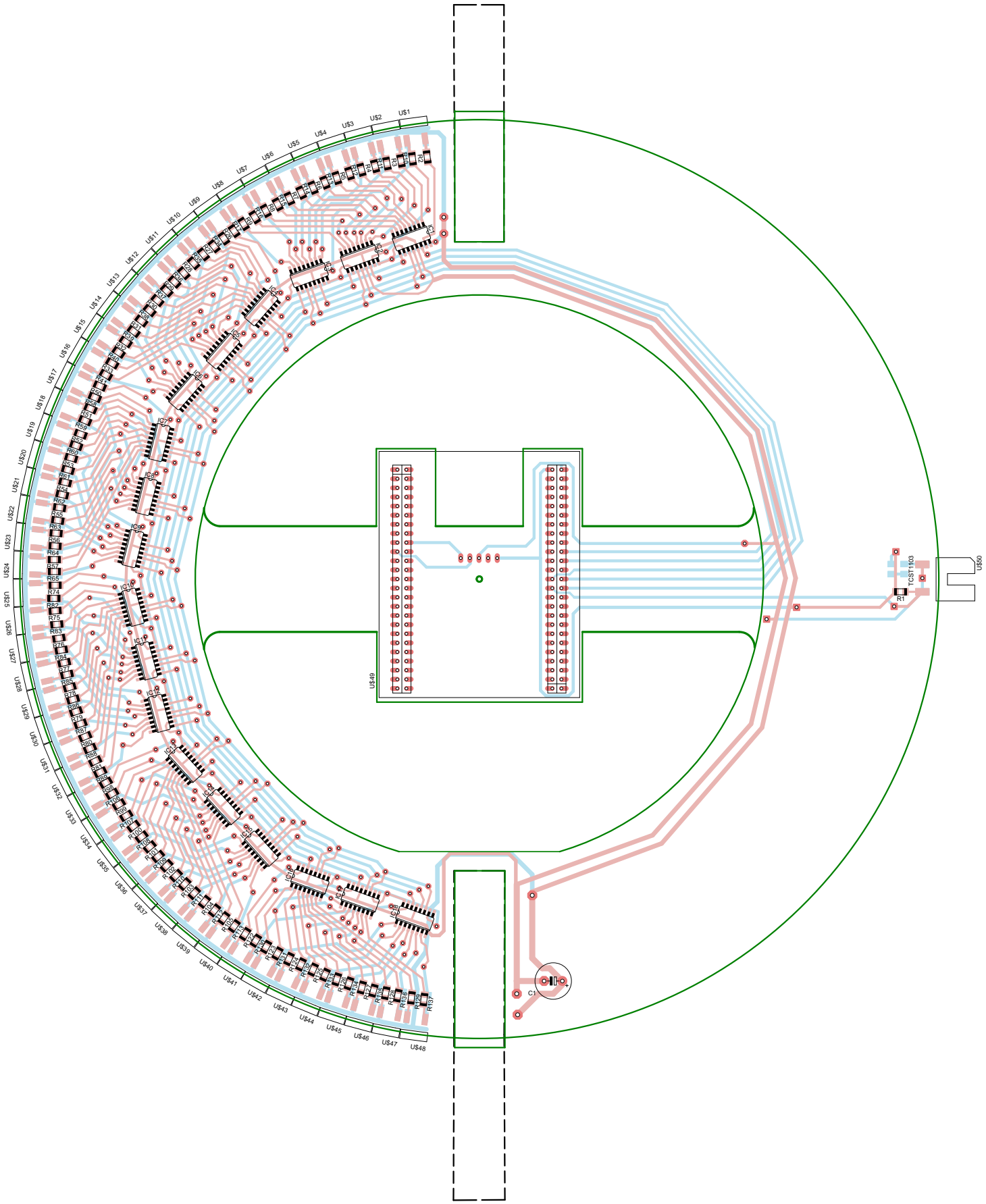
z danego wiersza dla trzech kolorów, następuje użycie zatrzasku powodującego wyświetlenie linii danych na matrycy wyświetlacza. Dodatkowo tuż przed wyjściem z podprogramu zachodzi inkrementacja zmiennej *c* odpowiadającej za przejście do kolejnej kolumny.

Program działa na zasadzie oczekiwania na przerwania. W momencie wywłaszczenia procesora przez podprogram realizujący obsługę czujnika, następuje wyzerowanie zmiennej *c*, a także zmiana odcienia wyświetlania kolorów za pomocą zmiennej *a*. Cały proces jest powtarzany nieskończoną liczbę razy, co podczas pracy kulistego wyświetlacza powoduje powstanie efektu widmowego, jakim jest pojawienie się obrazu na powierzchni sfery.

Funkcja obsługi przerwana od portu szeregowego sprawdza po wywołaniu czy przerwanie pochodzi od części nadawczej, czy też od odbiorczej. Jeśli dane zostały odebrane przez mikrokontroler, to ich odczyt z rejestru danych przychodzących USART odbywa się za pomocą funkcji *USARTReceiveData()*. Zwracana wartość jest zapisywana do tablicy bufora odbiorczego *RxBuf[]*. Następnie jest wykonywane sprawdzenie czy ostatni odebrany znak jest kodem znaku nowej linii (0x0D). Jeżeli tak, to następuje powrót z indeksowaniem tablicy na początek i ustawienie flagi informującej o nadejściu polecenia. Od indeksu tablicy w chwili sprawdzania warunku końca polecenia jest odejmowana wartość 1, co wynika z faktu, że podczas odczytywania danych z rejestru odbiorczego portu szeregowego jest używana postinkrementacja. W związku z tym w momencie sprawdzania końca komunikatu, indeks wskazuje na element o jedną pozycję dalej, aniżeli byłoby to oczekiwane. Teraz pozostaje dekodowanie łańcucha znaków zawartego w tablicy *RxBuf[]*.

Przerwanie od nadawczej części portu szeregowego jest generowane natychmiast po jego włączeniu, jeśli tylko rejestr danych do wysłania jest pusty. Podobnie jak dla odbierania, w pierwszej kolejności zachodzi sprawdzenie czy przerwanie faktycznie pochodzi od nadajnika. Służy do tego funkcja USART *GetITStatus()*. Następnie, wywołując funkcję *USART SendData()* i przesyłając jej w argumentach, o który UART chodzi oraz bajt informacji, wysyłana jest porcja danych. Sprawdzanie końca danych przeznaczonych do wysyłki odbywa się w identyczny sposób, jak w przypadku odbierania informacji. Na uwagę zasługuje jeszcze funkcja wyłączająca przerwanie od nadajnika UART. Jeśliby nie wyłączyć tego przerwania, to byłoby ono generowana bez przerwy i te same dane byłyby wysyłane „w kółko”.

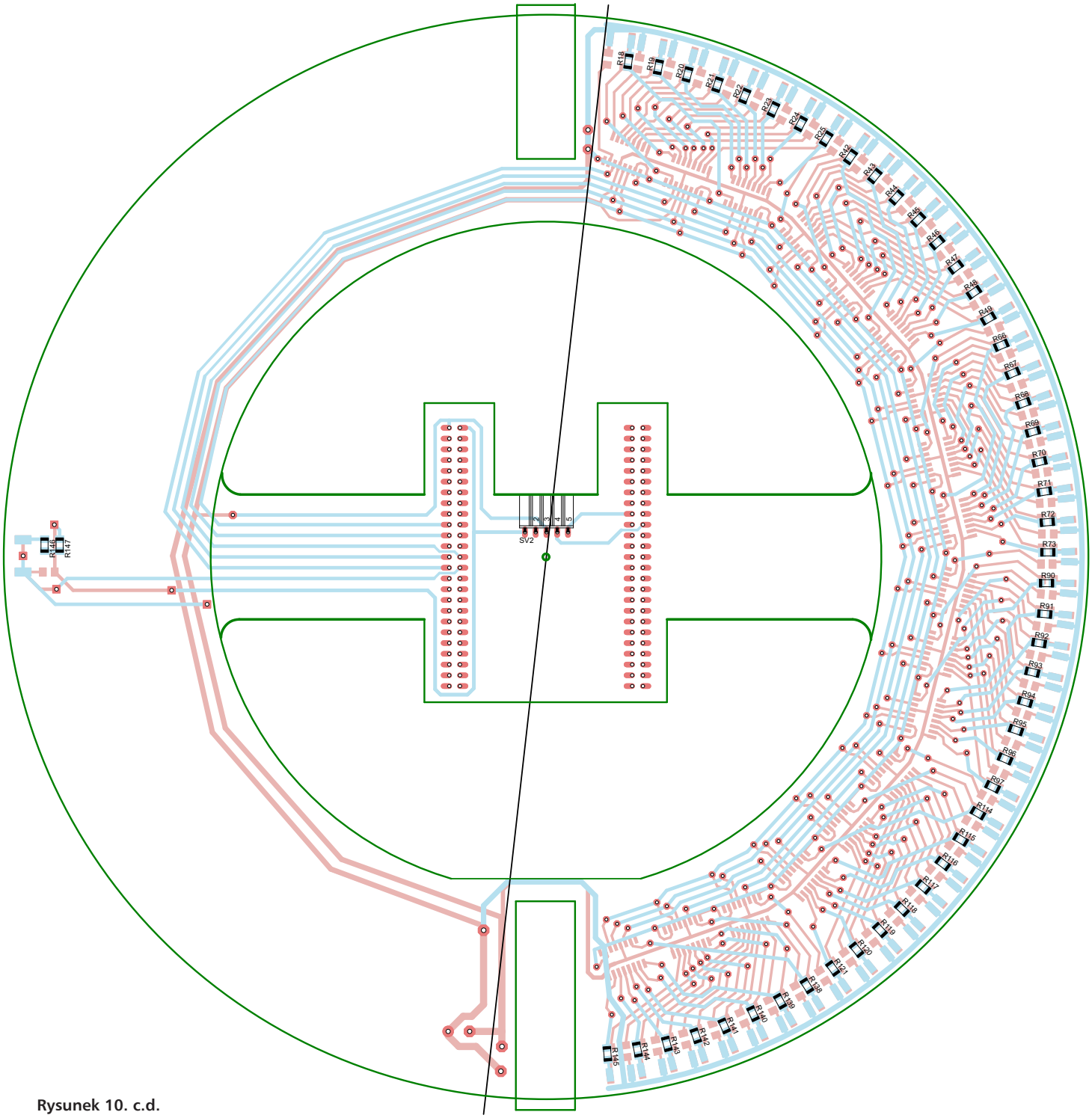
W momencie oczekiwania na przerwania program wraca do głównej funkcji programu main(). Po konfiguracji układów peryferyjnych wchodzi ona w nieskończoną pętle while(1), w której zachodzi analiza i przetwarzanie odebranych ciągów znaków z interfejsu szeregowego.



Rysunek 10. Schemat montażowy płytki wyświetlacza

Podsumowanie

Schemat montażowy płytki wyświetlacza widmowego pokazano na rysunku 10. Jej montaż mowego przysporzyło pewnych problemów, Początkowo uruchomienie wyświetlacza widktórych podczas tworzenia koncepcji nie brałem pod uwagę. Duży opór powietrza wirującej linijki diodowej i trudność precyzyjnego wyważenia, to podstawowe mankamenty konstrukcji mechanicznej. Dodatkowo, brak możliwości debugowania mikrokontrolera, ponieważ został on umieszczony na wirującej



Rysunek 10. c.d.

Listing 9. Funkcja obsługi przerwania od USART

```
void USART1_IRQHandler(void)
{
  if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
  {
    RxBuf[RxIndex++] = USART_ReceiveData(USART1);
    if(RxBuf[RxIndex-1] == 0x0D)
    {
        odebrano_polecenie = 1;
        RxIndex = 0;
    }
  }
  if(USART_GetITStatus(USART1, USART_IT_TXE) != RESET)
  {
    USART_SendData(USART1, TxBuf[TxIndex++]);
    if(TxBuf[TxIndex-1] == 0x0D)
    {
        USART_ITConfig(USART1, USART_IT_TXE, DISABLE);
        TxIndex = 0;
    }
  }
}
```

płytce – tu musiałem posłużyć się interfejsem USART i połączeniem Bluetooth. Zastosowany moduł BTM222 działa właściwie mimo ciągłej zmiany położenia wywołanej szybkimi ruchami wyświetlacza. Powstały obraz wywołany efektem widmowym jest czytelny i kolorowy. Stabilna konstrukcja, mocny silnik i złącze obrotowe wykorzystujące podwójne szczotki sprawiają, że wyświetlacz działa właściwie bez ograniczeń czasowych. Pozwala wyświetlać dowolne obrazy napisy i proste animacje zarówno statycznie jak i poruszające się po okręgu z dowolną prędkością.

Sławomir Łuniewski
luniewski.s@gmail.com