POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka
SPECJALNOŚĆ: Inżynieria systemów informatycznych (INS)

PROJEKT INŻYNIERSKI

Obsługa paneli RGB z wykorzystaniem Raspberry Pi i magistrali SPI lub I^2C

RGB panels control over SPI or I^2C interface and Raspberry Pi

AUTOR:
Mikalai Barysau

PROWADZĄCY PROJEKT:
dr inż. Tomasz Surmacz

OCENA PROJEKTU:

WROCŁAW 2014

Spis treści

Wstęp

Rosnąca liczba urządzeń elektronicznych w segmencie budżetowym, wzrastające możliwości obliczeniowe komputerów i coraz mniejsze rozmiary procesorów, płyt głównych, źródeł zasilania i nośników danych umożliwiają stworzenie produktów, które stosunkowo niedawno wymagały znaczących inwestycji finansowych i zazwyczaj były mało mobilne.

Obecny świat elektroniczny sprzyja powstaniu licznych "inteligentnych" systemów i oddzielnych urządzeń. Niektóre z tych urządzeń już teraz stały elementami naszego codziennego otoczenia, życia bez których nie możemy sobie wyobrazić. Spad kosztów, wzrost mocy jednostek obliczeniowych i gwałtowne zwiększenie rynku urządzeń elektronicznych powoduje coraz większe zapotrzebowanie społeczności na nowe rozwiązania sprzętowe i programowe.

Warto również zwrócić uwagę na to, że świat systemów operacyjnych również bardzo się rozwinął w ciągu ostatnich 20 lat. Jakiś czas temu najbardziej rozpowszechnionym system operacyjnym był system *Microsoft Windows*. Jednak po upływie czasu systemy, oparte na UNIX-ie, takie jak *GNU/Linux* i *OSX*, również zdobyły dużą popularność spośród użytkowników PC, co w tej chwili wymaga od twórców oprogramowania wyboru technologii, które pozwolą na stworzenie narzędzi, nie wymagających od użytkownika zainstalowania dodatkowych bibliotek, konfigurowania odpowiedniego środowiska czy jakiejkolwiek ingerencji w proces funkcjonowania dostarczonego produktu. Dla użytkownika końcowego najważniejszym jest to, żeby produkt działał w sposób, od niego oczekiwany.

Dlatego rola programisty i inżyniera, który jest twórcą tego produktu, polega nie tylko na rozwiązaniu konkretnego problemu technicznego, ale również na przemyśleniu tego, czy dany produkt jest intuicyjny w obsłudze, o ile to jest możliwe, czy jest on sprawny i przetestowany, czy użytkownik końcowy nie będzie w stanie zakłócić działanie programu przez nieodpowiednie korzystanie z dostępnych funkcjonalności.

Stworzenie takiego produktu jest zadaniem nietrywialnym i czasochłonnym. Opracowanie scenariuszy działania tworzonego systemu, przeprowadzenie analizy przypadków użycia, dobór odpowiednich technologii i efektywnych metod obliczeniowych jest niezbędną cześcią procesu tworzenia oprogramowania o wysokiej jakości.

Rozdział 1

Cel pracy

Celem danej pracy było stworzenie biblioteki do sterowania panelami RGB za pomocą *Raspberry Pi* przez interfejs *SPI* oraz aplikacji graficznej, umożliwiającej kalibrowanie kolorów i wykonanie symulacji działania systemu na zwykłym komputerze bez konieczności podłączenia sprzętu.

Wykorzystany w danej pracy sprzęt wymaga posiadanie pewnej wiedzy technicznej, dotyczącej zasad funkcjonowania pamięci układu *Texas Intruments tlc5947* specyfiki oraz opanowania technik programowania niskopoziomowego.

Stworzona biblioteka ma na celu znacznie ułatwić sterowanie podłączonym panelem LED RGB. Użytkownik biblioteki operuje na obiektach struktury *RGB* i funkcjach, które pozwalają zdefiniować lub pobrać kolor diody o wybranym numerze porządkowym. Dzięki temu nie jest konieczne dokładne zapoznanie się z modelem pamięci sterownika. Dla korzystania z biblioteki potrzebna jest jedynie podstawowa wiedza z programowania w języku *C*.

Z kolei rozwinięcie biblioteki, lub dostosowanie jej do innego modelu pamięci, będzie wymagało bardziej zaawansowanej wiedzy z dziedziny programowania niskopoziomowego. Nie powinno to, jednak, być dużym problem, ponieważ jedną z najważniejszych zasad tego projektu było bardzo szczególne dokumentowanie kodu, co pozwala osobie, nie posiadającej niezbędnej wiedzy o układzie *tlc5947*, w stosunkowo krótkim czasie zapoznać się z zasadami operowania na pamięci, definiowania kolorów i wyciągania danych z rejestru przesuwnego, wykorzystanego w danym układzie.

Rozdział 2

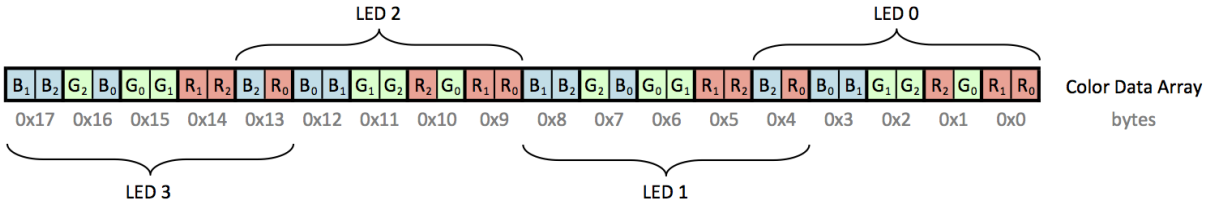
Opis sprzętu

2.1 Układ *Texas Intruments tlc5947*

Przy wykonaniu danej pracy został wykorzystany sterownik *Texas Intruments tlc5947* – 24 kanałowy sterownik diod LED RGB[6]. Każda z trzech barw odpowiada kolorowi czerwonemu, zielonemu oraz niebieskiemu.

W środku sterownika znajduje się 36-bajtowy rejestr przesuwny. Każdej z ośmiu diod RGB przysługuje 4.5 bajta pamięci, co oznacza że wartość odpowiadająca każdemu kolorowi może mieć wartość w przedziale od *0* do *4095*[6].

Dodatkowym utrudnieniem dla użytkownika stanowi zastosowanie w rejestrze przesuwnym układu *Texas Intruments tlc5947* modelu pamięci, polegającym na uporządkowaniu danych w innej kolejności, niż tego może oczekiwać użytkownik. Schemat modelu pamięci jest przedstawiony na rysunku 2.1



Rysunek 2.1 Kolejność danych, odpowiadająca dwóm diodom, w rejestrze przesuwnym sterownika *Texas Intruments tlc5947*

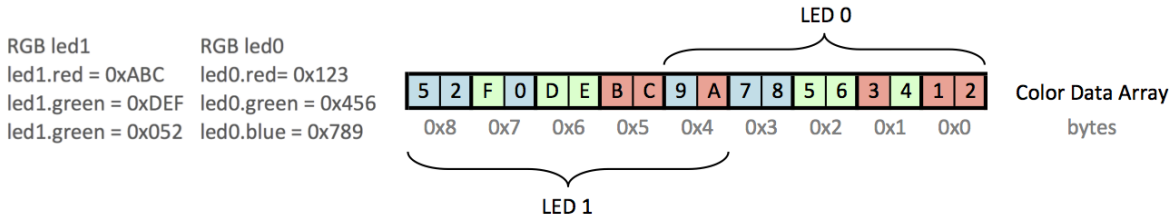
Litery $R_xG_xB_x$ i kolor komórek odpowiadają barwom oddzielnej diody. Indeksy $_x$ oznaczają kolejność liczb szesnastkowych, odpowiadających kodom poszczególnych kolorów. Adresy poszczególnych bajtów wypisane są szarą czcionką.

Następny przykład (rys. 2.2) ilustruje kolejność danych dla 2 diod w dziewięciobajtowym bloku pamięci.

2.2 Mikrokomputer *Raspberry Pi*

Raspberry Pi – komputer jednopłytkowy, stworzony przez *Raspberry Pi Foundation*. Mikrokomputer oparty jest na układzie *Broadcom BCM2835 SoC*, który składa się z procesora *ARM1176JZF-S 700 MHz*, *VideoCore IV GPU* i 256 megabajtów (MB) pamięci RAM. Na potrzeby projektu na urządzeniu został zainstalowany system *Raspbian*.

Konfiguracja programowa mikrokomputera została omówiona w rozdziale 3.



Rysunek 2.2 Kolejność danych, odpowiadających dwóm diodom, w rejestrze przesuwnym sterownika *Texas Intruments tlc5947*

Konfiguracja sprzętowa *Raspberry Pi model B*, który został użyty w danym projekcie, wygląda następująco:

SoC	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM)
CPU	700 MHz ARM1176JZF-S core (ARM11 family)
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decode
Pamięć (SDRAM)	256 MB (współdzielona z GPU) 256 MB (współdzielona z GPU)
Porty USB 2.0	2
Nośnik danych	złącze kart SD / MMC / SDIO MicroSD
Połączenia sieciowe	10/100 Ethernet (RJ45)
Pozostałe złącza	8 x GPIO, UART, szyna I^2C , szyna SPI z dwiema liniami CS, +3,3 V, +5 V, masa
Zasilanie	700 mA (3,5 W)
Źródło zasilania	5 V przy pomocy złącza MicroUSB, ewentualnie za pomocą złącza GPIO
Wymiary	85,60 × 53,98 mm
Waga	45 g

Rozdział 3

Opis wybranych technologii

Wybór języka, który będzie głównym narzędziem do realizacji projektu, należy dokonywać biorąc pod uwagę specyfikę projektu, zasoby sprzętowe, przypadki użycia produktu, który powinien powstać, wymagania wydajnościowe oraz funkcjonalne. Niemniej ważny jest czas, którym dysponuje programista.

Dany projekt jest rozwiązaniem zarówno sprzętowym, jak i programowym. Dlatego wybrane technologie muszą spełniać zestaw określonych wymagań, oraz gwarantować możliwość dalszego utrzymania i rozwoju produktu.

Po przeanalizowaniu najbardziej popularnych obecnie technologii, zostały wybrane języki *C* i *C++* z wykorzystaniem zestawu bibliotek *Qt framework*.

3.1 Biblioteka do obsługi układu *Texas Intruments tlc5947*

Język *C* jest sprawdzonym narzędziem, które pozwala na bardzo szczegółowe manipulacje pamięcią oraz bezpośrednią kontrolę nad złożonością i wydajnością implementowanych funkcji i algorytmów. Język *C++*, z kolei, oprócz wsparcia niskopoziomowych operacji również pozwala na implementację interfejsów obiektowych i graficznej powłoki do sterowania programem oraz reprezentacji wyników działania i komunikacji z użytkownikiem.

3.2 Aplikacja graficzna *LedSimulator* do symulacji działania systemu

Zaletą języków *C/C++* jest duża wydajność i szybkość wykonywania skompilowanego kodu (pod warunkiem poprawnego posługiwania się możliwościami tych języków, systemu operacyjnego, rejestrami procesora, pamięcią operacyjną i innymi zasobami sprzętowymi). Od momentu stworzenia języków *C* i *C++* została opracowana liczna grupa bibliotek, dających prawie nieograniczone możliwości do tworzenia oprogramowania. Swobodny dostęp do dokumentacji, tutoriali, projektów z otwartym kodem źródłowym i ogromna społeczność programistów *C/C++* jest gwarancją tego, że napotkane problemy techniczne ne zablokują rozwoju produktu i nie pozostawią programistę sam na sam z ich rozwiązaniem.

Jako narzędzie do tworzenia GUI został wybrany *Qt framework*, gdyż pozwala on w bardzo wygodny sposób tworzyć interfejsy graficzne użytkownika. Oprócz tego *Qt frame-*

work posiada wyjątkowo dobrą dokumentację, która jest wbudowana w IDE *Qt Creator* i również jest dostępna na stronie internetowej projektu *Qt:*

```
http://qt-project.org/doc/
```

W dokumentacji do framework-u można znaleźć szczegółowe opisy funkcji bibliotecznych i także wiele przykładowych fragmentów kodu, które znacznie ułatwiają rozumienie mechanizmu działania sygnałów i slotów, zasad działanie elementów interfejsu, kontenerów i innych narzędzi bibliotecznych. Kolejną zaletą użycia framework-u *Qt* jest możliwość stworzenia wieloplatformowej aplikacji graficznej bez konieczności utrzymania kilku wersji kodu dla różnych systemów operacyjnych.

3.3 Programowa konfiguracja *Raspberry Pi*

Jako środowisko systemowe na *Raspberry Pi* został zainstalowany *Raspbian* – system operacyjny *GNU/Linux* dla mikrokomputerów *Raspberry Pi*, oparty na dystrybucji *Debian*. System jest dostępny do ściągnięcia z oficjalnej strony projektu *Raspbian*:

```
http://www.raspbian.org/
```

lub z oficjalnej strony projektu *Raspberry Pi*:

```
http://www.raspberrypi.org/downloads/
```

Komunikacja z urządzeniami peryferyjnymi przez interfejs *SPI* odbywa się za pomocą sterownika *SPI*, wbudowanego w jądro systemu operacyjnego. Przed rozpoczęciem komunikacji między *Raspberry Pi* i urządzeniem peryferyjnym należy odpowiednio skonfigurować system operacyjny. Domyślnie sterownik *spi* znajduje się na "czarnej liście" modułów jądra systemu i dlatego nie jest ładowany podczas startu systemu. Aby dodać go do autostartu systemu należy otworzyć plik */etc/modprobe.d/raspi-blacklist.conf* w edytorze tekstowym i zakomentować linię (umieścić znak '#' na początku linii), w której znajduje się nazwa sterownika *spi-bcm2708*

```
1 # blacklist spi and i2c by default (many users don't need them)
2 # blacklist spi-bcm2708
3 blacklist i2c-bcm2708
```

Listing 3.1 Zmodyfikowany plik */etc/modprobe.d/raspi-blacklist.conf*

Również należy zmodyfikować */etc/modules*, usuwając znak '#' na początku linii, w której znajduje się nazwa modułu *spi-dev*:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.
#
# sound devices
snd-bcm2835
# SPI devices
spi-dev
# I2C devices
# i2c-dev
# 1-Wire devices
# w1-gpio
# 1-Wire thermometer devices
# w1-therm
```

Listing 3.2 Zmodyfikowany plik */etc/modules*

Po wprowadzeniu zmian należy zrestartować *Raspberry Pi* (np. za pomocą wykonania polecenia *reboot* w linii komend systemu operacyjnego *Raspbian*).

Po ponownym załadowaniu można sprawdzić, czy moduł *SPI* został załadowany. W tym celu należy skorzystać z polecenia *lsmod*.

```
Module                  Size  Used by
snd_bcm2835            12808  0
snd_pcm                74834  1 snd_bcm2835
snd_seq                52536  0
snd_timer              19698  2 snd_seq,snd_pcm
snd_seq_device          6300  1 snd_seq
snd                    52489  5 snd_seq_device,snd_timer,snd_seq,snd_pcm,
    snd_bcm2835
snd_page_alloc          4951  1 snd_pcm
spidev                  5136  0
spi_bcm2708             4401  0
```

Listing 3.3 Wynik wykonania polecenia *lsmod*

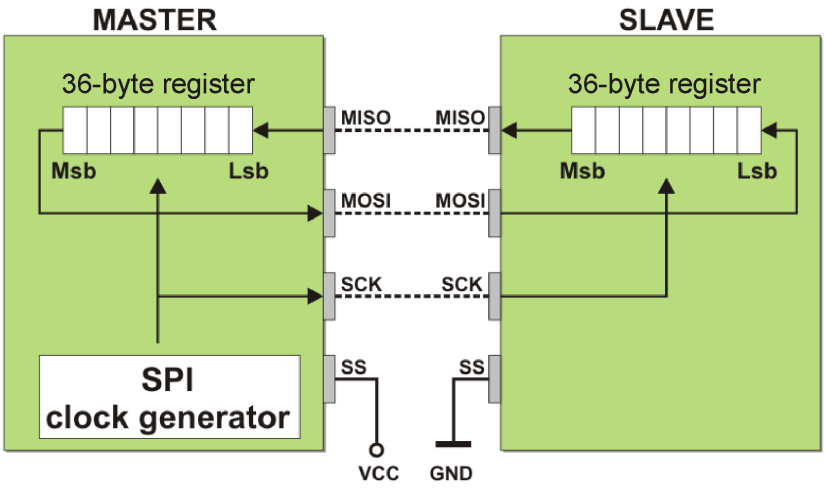
3.4 Interfejs komunikacyjny *SPI*

Interfejs *SPI (Serial Peripheral Interface)* umożliwia komunikację pomiędzy mikrokomputerem i urządzeniem peryferyjnym. Szeregowa transmisja danych odbywa się za pomocą trzech kanałów[5]:

- ***MOSI*** *(Master Out / Slave In)* - dane od jednostki nadrzędnej do podporządkowanej
- ***MISO*** *(Master In / Slave Out)* - dane od jednostki podporządkowanej do nadrzędnej
- ***SCK*** *(Serial Clock)* - zegar synchronizujący transmisję

Aktywacją wybranego urządzenia peryferyjnego odbywa się za pomocą dodatkowej linii ***SS*** *(Slave Select)*.

Na rysunku 3.1 jest umieszczony schemat komunikacji nędzy urządzeniem nadrzędnym i pojedynczym urządzeniem podporządkowanym.



Rysunek 3.1 Schemat komunikacji przez interfejs *SPI*

Po załadowaniu modułów jądra jest możliwa komunikacja z urządzeniem peryferyjnym przez interfejs *SPI*.

Rozdział 4

Implementacja

4.1 Biblioteka do obsługi układu *Texas Intruments tlc5947*

Biblioteka do sterowania układem składa się z trzech plików:

- *tlc5947_ controller.h*
- *tlc5947_ controller.c*
- *RGB.h*

Plik *RGB.h* zawiera strukturę *RGB*, która składa się z trzech pól:

- `uint64_t red`
- `uint64_t green`
- `uint64_t blue`

Każde pole służy do przechowywania wartości odpowiedniego koloru. Struktura została przeniesiona do oddzielnego pliku w cela dalszej możliwości korzystania z niej bez konieczności dołączania całej biblioteki.

Plik nagłówkowy *tlc5947_ controller.h* zawiera makro definicje i nagłówki trzech funkcji, które stanowią interfejs zewnętrzny (API) biblioteki:

- `int setLedRGB(uint32_t ledNumber, RGB rgbSet, uint8_t* tab)`
- `RGB getLedRGB(uint8_t ledNumber, uint8_t* tab)`
- `void printLedDataArray(uint8_t* tab)`

Również pllik *tlc5947_ controller.h* zawiera krótki komentarz, który wyjaśnia zasadę indeksowania diod w rejestrze przesuwnym układu *Texas Instruments tlc5947*.

Plik *tlc5947_controller.c* oprócz implementacji interfejsu zawiera również implementacje funkcji statycznych, które nie są widoczne dla użytkownika biblioteki, natomiast są wykorzystywane w mechanizmach adresacji i komunikacji z układem *tlc5947*. Lista zaimplementowanych funkcji wygląda nastepująco:

- `static void cleanRGB(RGB* rgb)`
- `static uint64_t getLedIndex(uint8_t ledNumber)`
- `static uint64_t compileEvenLedPattern(RGB rgbSet)`
- `static uint64_t compileOddLedPattern(RGB rgbSet)`
- `static int insertLedRgb(uint32_t ledNumber, uint64_t RGBpattern, uint8_t* tab)`
- `RGB getLedRGB(uint8_t ledNumber, uint8_t* tab)`
- `int setLedRGB(uint32_t ledNumber, RGB rgbSet, uint8_t* tab)`
- `void printLedDataArray(uint8_t* tab)`

W dalszej części tego rozdziału przedstawiony jest szczegółowy opis funkcji z przykładami kodu.

4.1.1 cleanRGB

Funkcja służy do czyszczenia obiektu struktury *RGB*.

Nazwa funkcji	*cleanRGB*
Funkcja statyczna	tak
Parametry wejściowe	***RGB**** *rgb* – wskaźnik na obiekt struktury *RGB*
Parametry wyjściowe	–

4.1.2 getLedIndex

Funkcja służy do obliczania indeksu bloku pamięci diody w rejestrze przesuwnym. Dzięki tej funkcji użytkownik biblioteki nie musi uwzględniać modelu pamięci rejestru przesuwnego układu *tlc5947*, korzystając z niego tak samo, jak ze zwykłej tablicy bajtowej.

Nazwa funkcji	*getLedIndex*
Funkcja statyczna	tak
Parametry wejściowe	***uint8_t*** *ledNumber* – numer diody
Parametry wyjściowe	***uint64_t*** *ledAddress* – indeks początku bloku pamięci diody w rejestrze przesuwnym

4.1.3 compileOddLedPattern

Funkcja służy do kompilacji fragmentu pamięci dla diody o numerze nieparzystym, który będzie zawierał uporządkowane w odpowiedniej kolejności wartości dla kolorów czerwonego, zielonego i niebieskiego. Przykład działania funkcji: W przypadku, gdy Parametr wejściowy ***RGB*** *rgbSet* ma następujące wartości:

- rgbSet.red = 0x123
- rgbSet.green = 0x456
- rgbSet.blue = 0x789

Funkcja zwróci wartość 64-bitową, która stanowi następujący fragment pamięci:

[00 00 00 89 67 45 23 01]

Otrzymany w wyniku fragment jest później wkładany do tablicy *txdata*, a następnie przesyłany do rejestru przesuwnego. Otrzymany fragment pamięci jest wkładany do tablicy *txdata* za pomocą operacji logicznej *OR*. Wartości dla odpowiednich kolorów znajdują się na właściwych pozycjach, przy czym reszta fragmentu pamięci jest wypełniona zerami. Dzięki temu wartości kolorów dla innych diod, które znajdują się w tablicy, nie ulegają modyfikacji.

Operację na pamięci w większości wykonywane są za pomocą operatorów logicznych *AND*, *OR* oraz przesunięć bitowych. W kodzie biblioteki znajduje się szczegółowy opis zmian, które odbywają się w pamięci.

```
1  //compiling green color
2  hex = 0;                      //     hex = [00 00 00 00 00 00 00 00]
3  hex = green & hex1Mask;       //     hex = [00 00 00 00 00 00 04 00]
4  hex = hex << 3 * POSITION;    //     hex = [00 00 00 00 00 40 00 00]
5  pattern = pattern | hex;      // pattern = [00 00 00 00 00 40 23 01]
```

Listing 4.1 Fragment kodu, w którym przedstawiony jest komentarz ilustrujący efekt wykonywania operacji na pamięci

Nazwa funkcji	*compileEvenLedPattern*
Funkcja statyczna	tak
Parametry wejściowe	***RGB*** *rgbSet* – obiekt struktury *RGB*, który zawiera wartości dla odpowiednich kolorów
Parametry wyjściowe	***uint64_t*** *pattern* – spreparowany fragment pamięci, który następnie zostanie włożony do rejestru przesuwnego

4.1.4 compileEvenLedPattern

Funkcja służy do kompilacji fragmentu pamięci dla diody o numerze parzystym, który będzie zawierał uporządkowane w odpowiedniej kolejności wartości dla kolorów czerwonego, zielonego i niebieskiego. Przykład działania funkcji: W przypadku, gdy Parametr wejściowy ***RGB*** *rgbSet* przyjmuje następujące wartości:

- rgbSet.red = 0x123
- rgbSet.green = 0x456
- rgbSet.blue = 0x789

Funkcja zwróci wartość 64-bitową, która stanowi następujący fragment pamięci:

[00 00 00 90 78 56 34 21]

Otrzymany w wyniku fragment jest później wkładany do tablicy *txdata*, a następnie przesyłany do rejestru przesuwnego. Otrzymany fragment pamięci jest wkładany do tablicy *txdata* za pomocą operacji logicznej *OR*. Wartości dla odpowiednich kolorów znajdują się na właściwych pozycjach, przy czym reszta fragmentu pamięci jest wypełniona zerami. Dzięki temu wartości kolorów dla innych diod, które znajdują się w tablicy, nie ulegają modyfikacji.

Operację na pamięci w większości wykonywane są za pomoća operatorów logicznych *AND*, *OR* oraz przesunięć bitowych. W kodzie biblioteki znajduje się szczegółowy opis zmian, które odbywają się w pamięci.

```
1 //compiling blue color
2 hex = 0;                    //     hex = [00 00 00 00 00 00 00 00]
3 hex = blue & hex1Mask;      //     hex = [00 00 00 00 00 00 07 00]
4 hex = hex << 5 * POSITION;  //     hex = [00 00 00 00 70 00 00 00]
5 pattern = pattern | hex;    // pattern = [00 00 00 00 70 56 34 12]
```

Listing 4.2 Fragment kodu, w którym przedstawiony jest komentarz ilustrujący efekt wykonywania operacji na pamięci

Nazwa funkcji	*compileEvenLedPattern*
Funkcja statyczna	tak
Parametry wejściowe	***RGB*** *rgbSet* – obiekt struktury *RGB*, który zawiera wartości dla odpowiednich kolorów
Parametry wyjściowe	***uint64_t*** *pattern* – spreparowany fragment pamięci, który następnie zostanie włożony do rejestru przesuwnego

4.1.5 insertLedRgb

Funkcja służy do obliczania indeksu bloku pamięci diody w rejestrze przesuwnym. Dzięki tej funkcji użytkownik biblioteki nie musi uwzględniać modelu pamięci rejestru przesuwnego układu *tlc5947*, korzystając z niego tak samo, jak ze zwykłej tablicy bajtowej.

Nazwa funkcji	*insertLedRgb*
Funkcja statyczna	tak
Parametry wejściowe	• ***uint32_t*** *ledNumber* – numer diody, której dotyczy działanie • ***uint64_t*** *RGBpattern* – spreparowany 64-bitowy fragment pamięci, zawierający odpowiednie wartości kolorów • ***uint8_t**** *tab* – wskaźnik na tablicę *txdata*, która zostanie później przesłana do rejestru przesuwnego
Parametry wyjściowe	***int*** – wartość *0* mówi o skutecznym wykonaniu operacji; w przypadku błędu zwracana jest wartość *-1*

4.1.6 getLedRGB

Funkcja zwraca obiekt struktury ***RGB***, który zawiera wartości kolorów wybranej diody. Wartości są pobierane z tablicy *txdata*, i wyciągane za pomocą operatorów logicznych *AND*, *OR* oraz przesunięć bitowych. W kodzie biblioteki znajduje się szczegółowy opis zmian, które odbywają się w pamięci.

```
1  //getting red color
2  hex = pattern & (0x0F);          //         hex = [00 00 00 00 00 00 00 01]
3  hex = hex << 2 * POSITION;       //         hex = [00 00 00 00 00 00 01 00]
4  ledRGB.red = ledRGB.red | hex; // ledRgb.red = [00 00 00 00 00 00 01 00]
5  hex = 0;                         //         hex = [00 00 00 00 00 00 00 00]
```

Listing 4.3 Fragment kodu, w którym przedstawiony jest komentarz ilustrujący efekt wykonywania operacji na pamięci

Nazwa funkcji	*getLedRGB*
Funkcja statyczna	nie
Parametry wejściowe	• ***uint32_t*** *ledNumber* – numer diody, której dotyczy działanie • ***uint64_t*** *RGBpattern* – spreparowany 64-bitowy fragment pamięci, zawierający odpowiednie wartości kolorów • ***uint8_t**** *tab* – wskaźnik na tablicę *txdata*, która zostanie później przesłana do rejestru przesuwnego
Parametry wyjściowe	***RGB*** *ledRGB* – obiekt struktury ***RGB***, który zawiera odpowiednie wartości kolorów czerwonego, zielonego i niebieskiego wybranej diody

4.1.7 setLedRGB

Funkcja służy do ustawiana koloru wybranej diody. Korzysta ona z funkcji statycznych biblioteki: *compileEvenLedPattern*, *compileOddLedPattern*, *insertLedRgb*

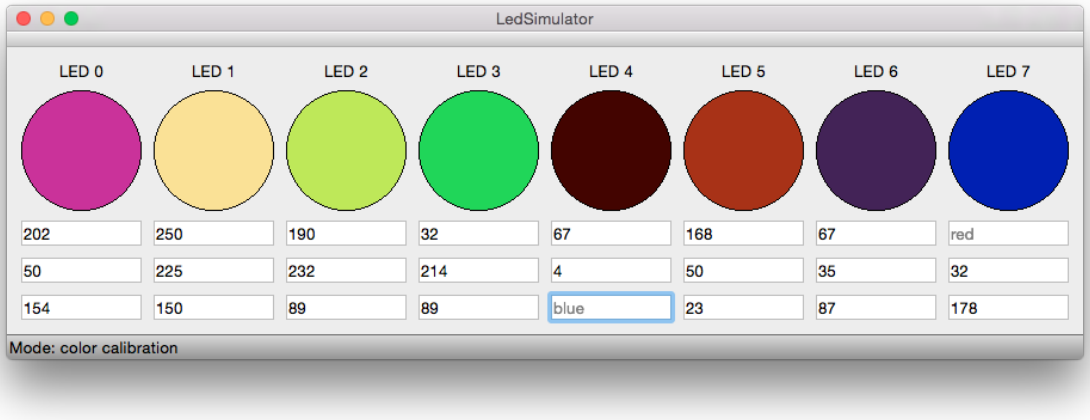
Nazwa funkcji	*setLedRGB*
Funkcja statyczna	nie
Parametry wejściowe	• ***uint32_t*** *ledNumber* – numer diody, której dotyczy działanie • ***RGB*** *rgbSet* – obiekt struktury ***RGB***, który zawiera odpowiednie wartości kolorów czerwonego, zielonego i niebieskiego • ***uint8_t**** *tab* – wskaźnik na tablicę *txdata*, która zostanie później przesłana do rejestru przesuwnego
Parametry wyjściowe	***int*** *result* – wartość *0* mówi o skutecznym wykonaniu operacji; w przypadku błędu zwracana jest wartość *-1*

4.1.8 printLedDataArray

Funkcja służy do wyświetlania zawartości tablicy *txdata* na standardowym wyjściu.

Nazwa funkcji	*printLedDataArray*
Funkcja statyczna	nie
Parametry wejściowe	***uint8_t**** *tab* – wskaźnik na tablicę *txdata*, która zawiera wartości kolorów wszystkich diod
Parametry wyjściowe	–

4.2 Aplikacja graficzna *LedSimulator*



Rysunek 4.1 Aplikacja graficzna *LedSimulator*: przykładowe kolory

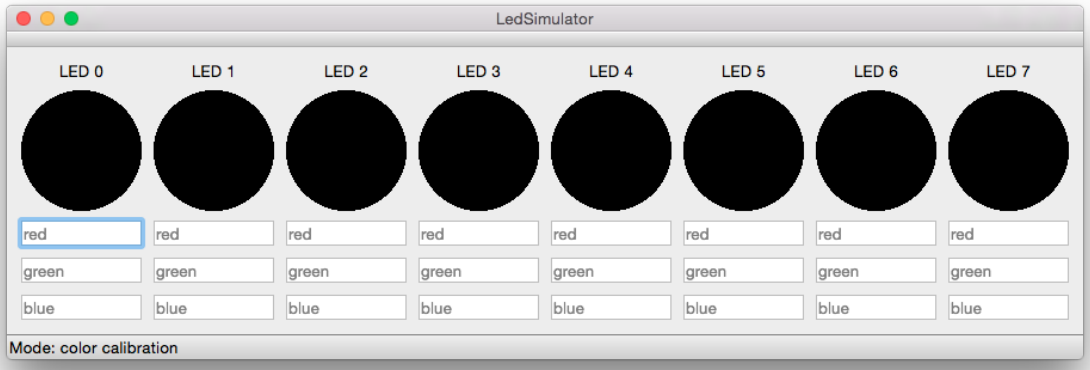
Na podanym zrzucie ekranu (rys. 4.1) jest przedstawione okno główne aplikacji graficznej *LedSimulator*. Aplikacja znajduje się w trybie *Color calibration*. Pokazany jest układ zawierający 8 diod RGB. Diody są ponumerowane od *0*. Każdej diodzie przysługują 3 pola tekstowe:

- *red*
- *green*
- *blue*

W każde pole użytkownik może wprowadzić wartość od *0* do *255*. Wpisywane wartości są sprawdzane za pomocą wyrażeń regularnych, dzięki czemu nie jest możliwe wprowadzenie innych znaków, niż cyfry, oraz liczb, większych *niż 255*.

Kolory diod są odświeżane w momencie zmiany wartości w odpowiednim polu. Domyślną wartością, przechowywaną w każdym polu, jest *0*, co oznacza brak koloru (rys. 4.2).

Aplikacja może pracować w dwóch trybach (rys. 4.3):

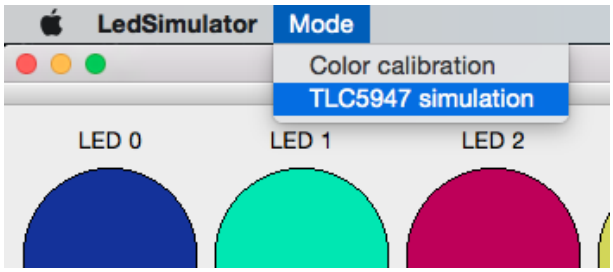


Rysunek 4.2 Aplikacja graficzna *LedSimulator*: domyślny stan diod

- *Color calibration*
- *TLC5947 simulation*

Tryb *Color calibration* został opisany wyżej.

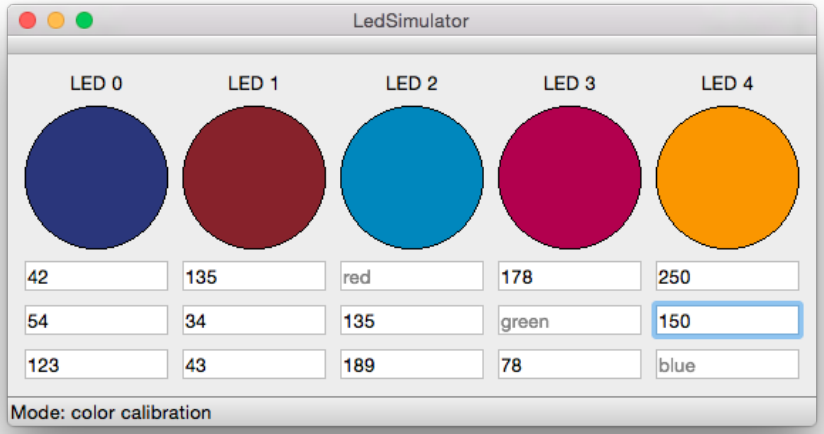
Tryb *TLC5947 simulation* służy do testowania działania biblioteki w przypadku braku dostępu do układu *Texas Instruments TLC5947*. Przy wyborze tego trybu uruchomiony zostanie scenariusz, umieszczony w funckcji *runLedScenario*.



Rysunek 4.3 Wybór trybu działania aplikacji *LedSimulator*

Aplikacja *LedSimulator* może być skonfigurowana odpowiednio dla rożnych układów diod. Jak widać na rysunku 4.4, aplikacja może symulować działanie układu o dowolnej liczbie diod.

Również jest możliwe ułożenie diod w kilka rzędów. Na rysunku 4.5 przedstawiony jest widok aplikacji, skonfigurowanej dla symulowania działania układu diod 2x4



Rysunek 4.4 Aplikacja *LedSimulator*, skonfigurowana dla symulowania działania układu 5 diod

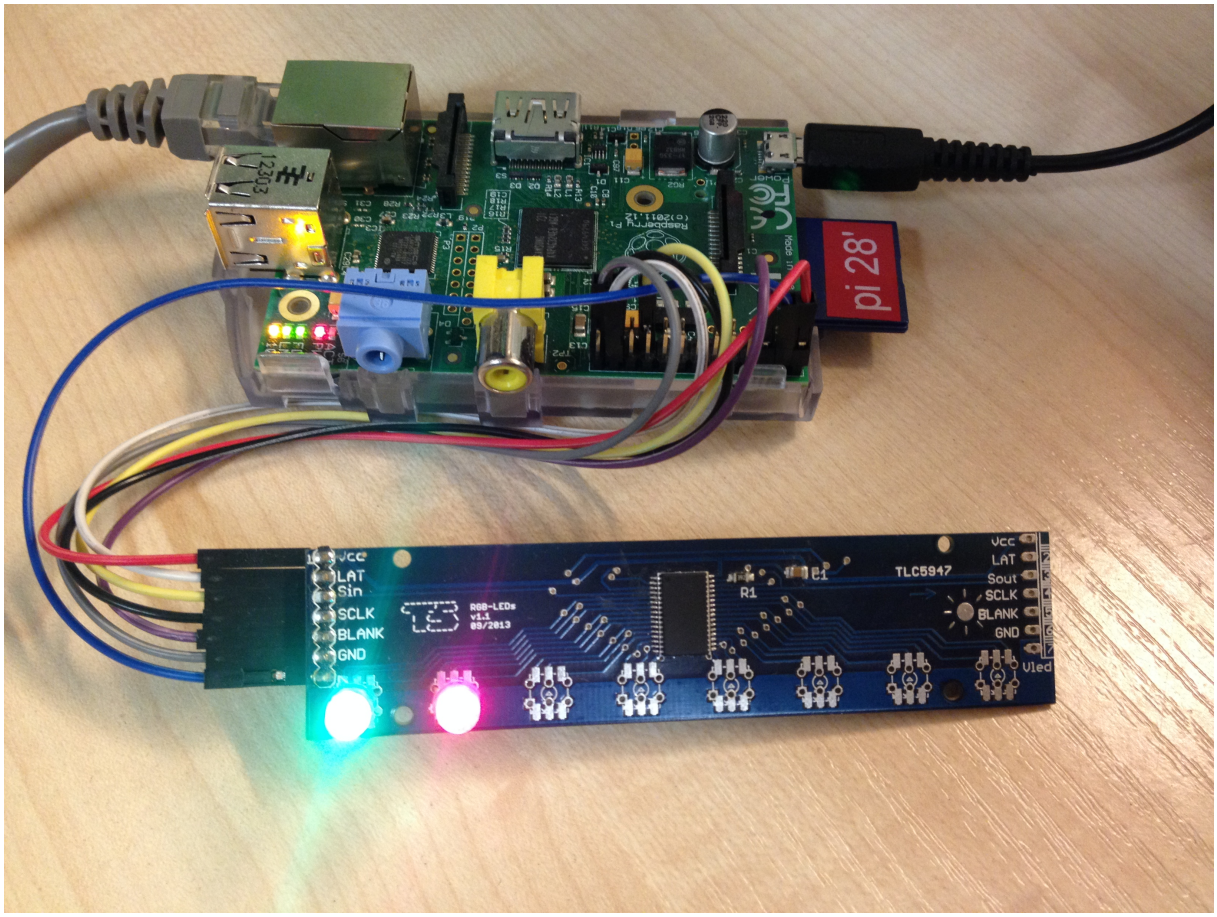
Rysunek 4.5 Aplikacja *LedSimulator*, skonfigurowana dla symulowania działania układu 2x4 diod

Rozdział 5

Przykłady uruchomienia projektu

5.1 Zastosowanie biblioteki *tlc5947_ controller* do sterowania układ 8 diod za pomocą *Raspberry Pi*

Na rysunku 5.1 jest przesdtawiony działający układ *tlc5947* z zamontowanymi 2 diodami, podłączony do mikrokomputera *Raspberry Pi*.



Rysunek 5.1 Układ *tlc5947* i mikrokomputer *Raspberry Pi*

Niżej jest przedstawiony fragment kodu, który pozwala na sterowanie układem *tlc5947* za pomocą napisanej biblioteki.

```
 #include <math.h>
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include "RPI_SPI_TLC5947/tlc5947_lib/tlc5947_controller.h"


#define TIMER 100000


static void pabort(const char *s)
{
  perror(s);
  abort();
}

static const char *device = "/dev/spidev0.0";
static uint8_t mode;
static uint8_t bits = 8;
static uint32_t speed = 200000;
static uint16_t delay;
int factor = 4096/256;


uint8_t txdata[SIZE] = {
//  rrrrrrr-ggggggg--bbbbbbb/rrrrrrrr-ggggggggg-bbbbbbb/
  0x00, 0x00, 0x00, 0xff, 0x00, 0x00, 0xff, 0x00, 0x00, // right
  0xff, 0xf0, 0x00, 0xff, 0x00, 0xff, 0x00, 0xff, 0x00, //  ^
  0xff, 0xf0, 0x00, 0xff, 0x00, 0x00, 0x00, 0x0f, 0xff, //  |
  0x00, 0x0f, 0xff, 0x00, 0xff, 0xf0, 0x00, 0x00, 0x00  // left
};


static void write36(int fd)
{
  write(fd, txdata, 36);
}

static void initspi(int fd)
{
  int ret = 0;

/** spi mode **/
  ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
  if (ret == -1)
    pabort("can't set spi mode");

  ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
  if (ret == -1)
```

```
57      pabort("can't get spi mode");
58
59  /** bits per word **/
60    ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
61    if (ret == -1)
62      pabort("can't set bits per word");
63
64    ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
65    if (ret == -1)
66      pabort("can't get bits per word");
67
68  /** max speed hz **/
69    ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
70    if (ret == -1)
71      pabort("can't set max speed hz");
72
73    ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
74    if (ret == -1)
75      pabort("can't get max speed hz");
76
77  }
78
79  void cleanAllLeds()
80  {
81    memset(txdata, 0, SIZE);
82  }
83
84  int main(int argc, char *argv[])
85  {
86    int fd;
87
88    fd = open(device, O_RDWR);
89
90    if (fd < 0)
91      pabort("can't open device");
92
93    initspi(fd);
94
95    printf("--------------------------------------\n");
96    printf("spi mode: %d\n", mode);
97    printf("bits per word: %d\n", bits);
98    printf("max speed: %d Hz (%d KHz)\n", speed, speed / 1000);
99    printf("--------------------------------------\n");
100
101   cleanAllLeds();
102
103   RGB rgb7;
104   rgb7.red   = 0xff;
105   rgb7.green = 0x0;
106   rgb7.blue  = 0xf;
107
108   RGB rgb6;
109   rgb6.red   = 0x0;
110   rgb6.green = 0xff;
111   rgb6.blue  = 0xf;
112
113   int i=0;
114   for( ;i<10; ++i)
115   {
```

```
116     setLedRGB(6, rgb6, txdata);
117     setLedRGB(7, rgb7, txdata);
118     usleep(TIMER);
119     write36(fd);
120     printLedDataArray(txdata);
121
122     setLedRGB(7, rgb6, txdata);
123     setLedRGB(6, rgb7, txdata);
124     usleep(TIMER);
125     write36(fd);
126     printLedDataArray(txdata);
127   }
128
129
130   close(fd);
131   printf("close FD\n");
132
133   return;
134 }
```

Listing 5.1 Sterowanie układem emphtlc5947 za pomocą biblioteki *tlc5947_controller*

5.2 Uruchomienie aplikacji w środowisku *Qt creator*

Procedura instalacji i konfigurowania środowiska została przeprowadzona w systemie operacyjnym *Ubuntu Linux*. Do uruchomienia programu *LedSymulator* należy ściągnąć środowisko *Qt Community Edition* (aktualna wersja 5.4), w skład którego wchodzą IDE *Qt creator* i zestaw bibliotek *Qt*. Środowisko jest dostępne do ściągnięcia ze strony:

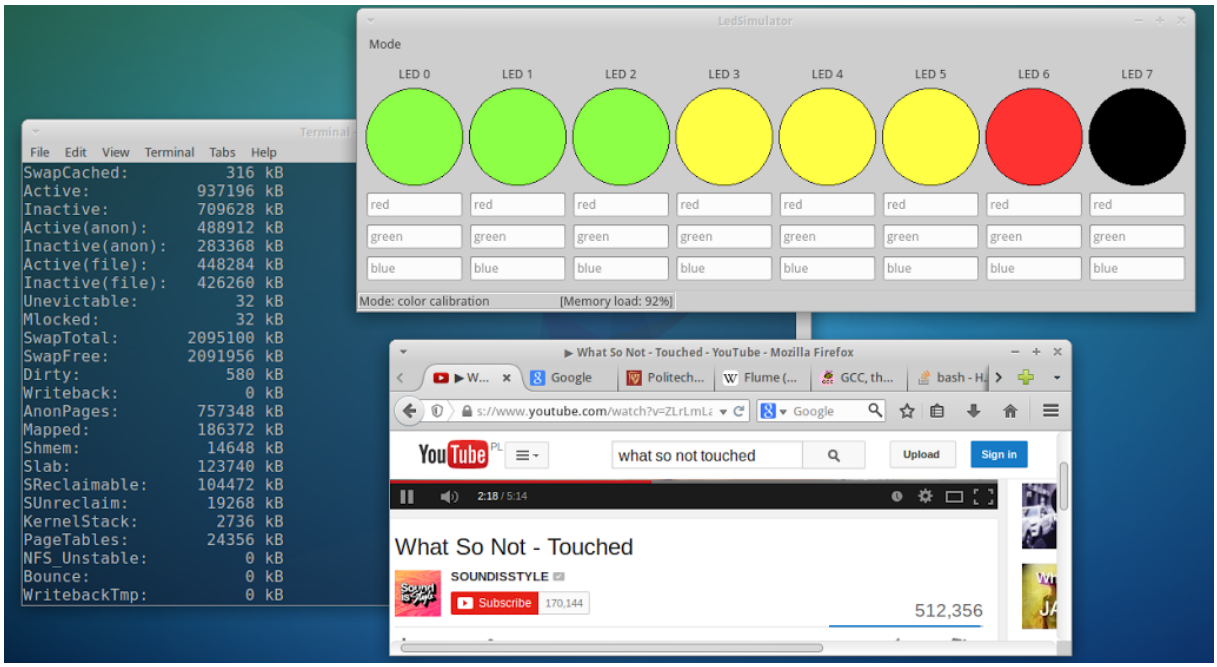
```
http://qt-project.org
```

Aktualny link do ściągnięcia:

```
http://download.qt-project.org/official_releases/online_installers/
qt-opensource-linux-x64-online.run
```

Po zainstalowaniu środowiska należy zaimportować projekt w IDE *Qt creator*. Projekt jest skonfigurowany do współpracy z biblioteką do obsługi układu *tlc5947* w pełni gotowy do kompilacji i uruchomienia.

Na poniższym rysunku jest przedstawiona aplikacja *LedSimulator*, uruchomionej w systemie Ubuntu Linux. Aplikacja działa w trybie symulacji. Scenariusz symulacji polega na wizualizacji ilości zajętej pamięci operacyjnej.



Rysunek 5.2 Wizualizacja ilości zajętej pamięci RAM w aplikacji *LedSimulator*

Podsumowanie

Zgodnie ze wstępnymi założeniami, wynikiem projektu inżynierskiego jest biblioteka *tlc5947_controller*, napisana w języku C oraz aplikacja graficzna *LedSimulator*, stworzona z wykorzystaniem framework-u *Qt*. Celem nadrzędnym było stworzenie wygodnego narzędzia do sterowania układem *Texas Intruments tlc5947* oraz napisanie szczegółowej dokumentacji. Biblioteka pozwala na ustawienie koloru wybranej diody, pobieranie aktualnego koloru i wyświetlenie stanu rejestru przesuwnego na standardowym wyjściu. Użycie funkcji, udostępnionych w API biblioteki, nie wymaga głębokie wiedzy na temat interfejsów wymiany danymi, lub zasad działania układów elektronicznych. Użytkownik ma do dyspozycji abstrakcyjną tablicę danych oraz strukturę *RGB*, które pełnią funkcją pośredniczącą pomiędzy użytkownikiem biblioteki i układem *tlc5947*.

Potencjalne rozbudowanie biblioteki nie powinno być problematyczne ze względu na dużą ilość komentarzy i przykładów w kodzie źródłowym. Działanie biblioteki i aplikacji graficznej zostało sprawdzone na kilku systemach operacyjnych (*Microsoft Windows*, *Ubuntu Linux* oraz *OSX*). Zastosowanie framework-u *Qt* znacznie przyspieszyło pracę nad częścią graficzną projektu i pozwoliło na stworzenie rozwiązania wieloplatformowego.

Dzięki dużej popularności wybranych technologii, bogatej dokumentacji i aktywnemu udziału promotora, podczas pracy nad danym projektem nie napotkano żadnych problemów, które następnie nie zostały rozwiązane.

Wszystkie cele i założenia projektu zostały spełnione.

Spis rysunków

Literatura

[1] *Loading I2C, SPI and 1-Wire drivers on the Raspberry Pi under Raspbian wheezy.* https://www.modmypi.com/blog/loading-i2c-spi-and-1-wire-drivers-on-the-raspberry-pi-under-raspbian-wheezy (data dostępu: 09.12.2014).

[2] *Qt Documentation.* http://doc.qt.io/ (data dostępu: 09.12.2014).

[3] *Raspberry Pi Hardware Documentation. SPI.* http://www.raspberrypi.org/documentation/hardware README.md (data dostępu: 09.12.2014).

[4] *Using The SPI Interface.* http://www.raspberry-projects.com/pi/programming-in-c/spi/using-the-spi-interface (data dostępu: 09.12.2014).

[5] *Broadcom BCM2835 ARM Peripherals.* 2012. PDF.

[6] *Texas Instruments TLC5947: Datasheet.* 2014. 24-Channel, 12-Bit PWM LED Driver with Internal Oscillator (Rev. A), PDF.

[7] G. Henderson. *Understanding SPI on the Raspberry Pi.* https://projects.drogon.net/understanding-spi-on-the-raspberry-pi (data dostępu: 09.12.2014).

[8] B. W. Kernighan, D. M. Ritchie. *The C Programming Language.* wydanie 2, 1995.

[9] B. Stroustrup. *The C++ Programming Language.* wydanie 4, 2013.