Ćwiczenie 3
Wyświetlacz TFT MI0283QT-9A

Bibliografia

[1] Karta katalogowa mikrokontrolera LM4F232H5QD - http://www.ti.com/lit/ds/symlink/tm4c123gh6pge.pdf
[2] Instrukcja obsługi płyty EasyMxPROv7 http://www.mikroe.com/downloads/get/1812/easymx_pro_v7_stellaris_manual_v102.pdf
[3] Strona wiki Texas Instruments dotycząca płyty EasyMxPROv7 http://processors.wiki.ti.com/index.php/EasyMxPRO
[4] Instrukcja obsługi ROM LM4F232H5QD http://www.ti.com/lit/ug/spmu270a/spmu270a.pdf
[5] Instrukcja obsługi biblioteki TivaWare Graphics Library http://www.ti.com/lit/ug/spmu300a/spmu300a.pdf

Wymagania:

- Znajomość środowiska CodeComposerStudio, umiejętność utworzenia projektu, zarządzania, uruchomienia, ustawienia opcji linkera oraz assemblera
- Znajomość podstawowych funkcji z klasy GPIO (ćwiczenie 1)
- Ogólna wiedza dotycząca biblioteki TivaWare Graphics Library – baza teoretyczna

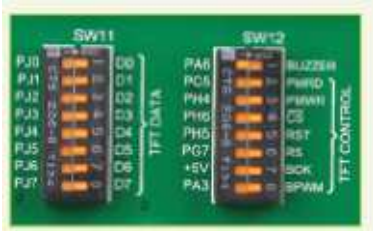
Część I – Wyświetlacz TFT

Wprowadzenie

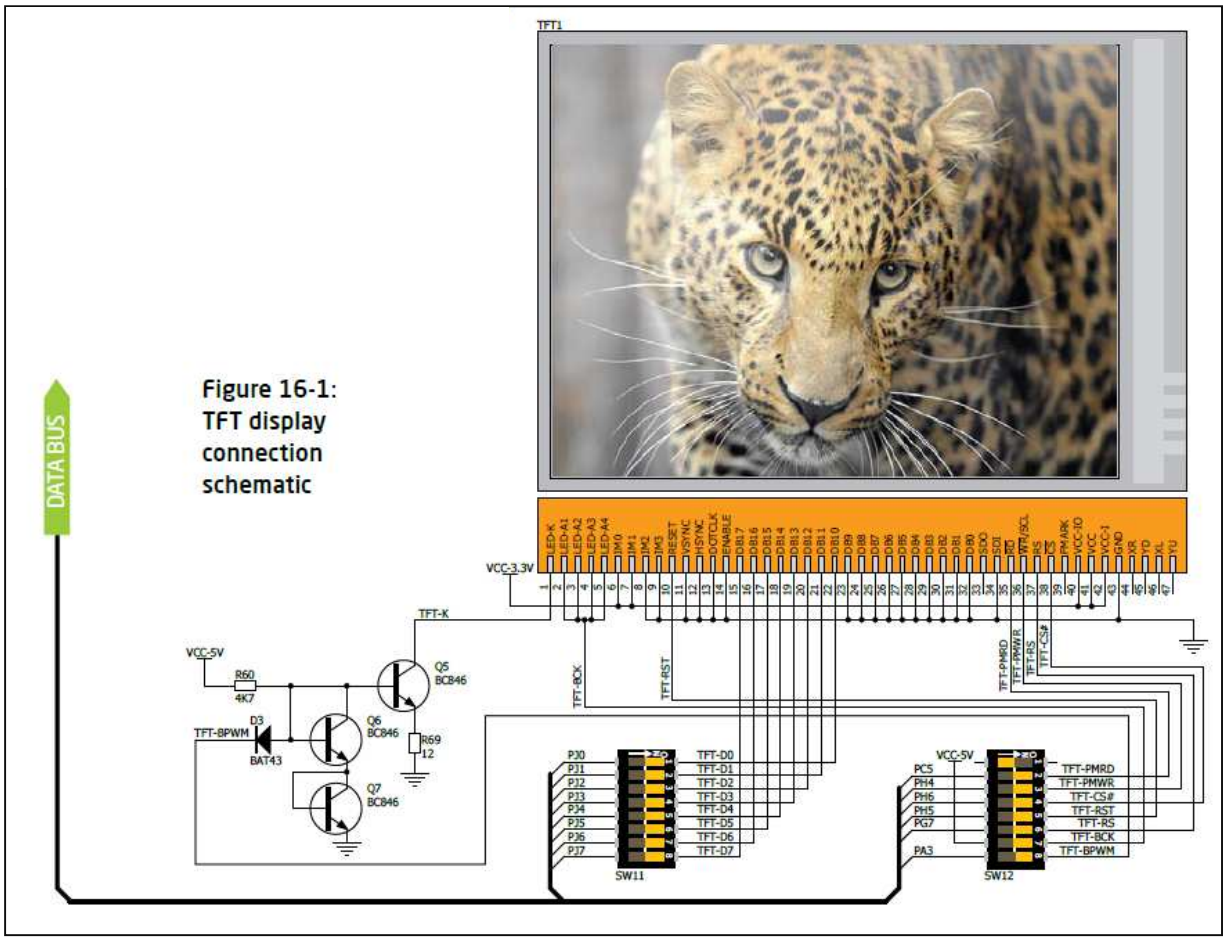
Jeden z istotnych elementów wyposażenia płyty EasyMx PRO™ v7 for Stellaris® ARM® stanowi kolorowy ekran TFT (ang. thin-film-transistor liquid-crystal display) o średnicy 2.83 cala wraz z panelem dotykowym. Ekran ten jest podświetlany diodami LED i charakteryzuje się rozdzielczością 320x240 pikseli. Według specyfikacji każdy piksel może wyświetlać 262144 różnych kolorów (18 bitów). W nowszych płytach (produkowanych po 20 marca 2013r.) wyświetlacz MI0283QT-2 wraz ze sterownikiem HX8347D został zastąpiony nowszym rozwiązaniem w postaci ekranu MI0283QT-9A wraz z kontrolerem ILI9341; jest to istotne w momencie wybierania odpowiednich bibliotek.

Rysunek 1. Zdjęcie ekranu TFT typu MI0283QT-2

Wyświetlacz włącza się za pomocą przełączników SW11.1 - SW11.8 oraz SW12.2 – SW12.6. Podświetlenie można włączyć na dwa sposoby. Pierwszy to użycie przełącznika SW12.7, które włączy podświetlenie z maksymalną wartością jasności. Drugi to wykorzystanie sygnału PWM dzięki czemu można dostosować jasność ekranu. Aby umożliwić sterowanie jasnością poprzez sygnał PWM należy ustawić przełączniki SW12.7 i SW12.8 na pozycję włączoną.



Schemat podłaczenia wyświetlacza został przedstawiony na rysunku poniżej:



Rysunek 2. Schemat podłączenia wyświetlacza

Wybrane paremetry wyświetlacza MI0283QT-9A zostały przedstawione w tabeli 1, pochodzącej ze specyfikacji wyświetlacza

(http://www.mikroe.com/downloads/get/1975/tft_320_240_mi0283qt_9a_v1_3_spec.pdf)

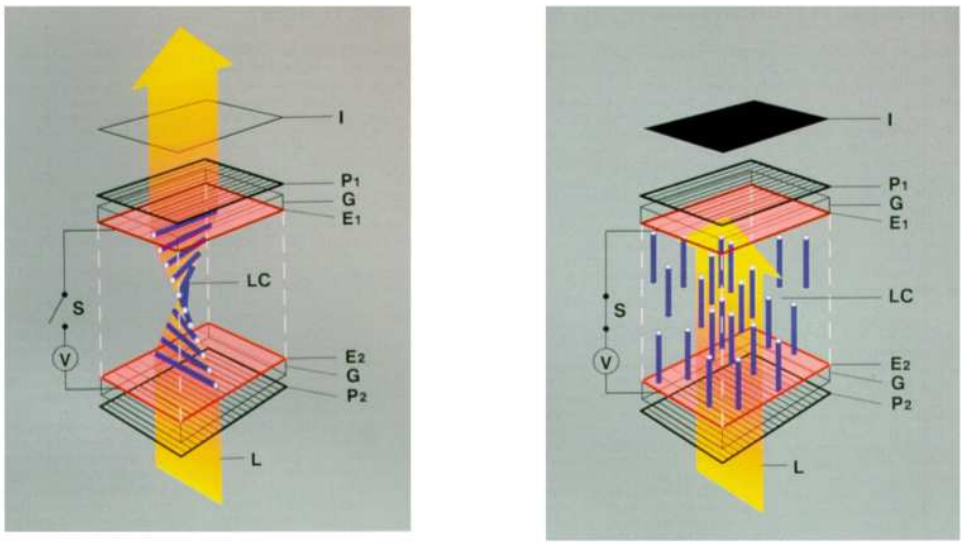
Table 1. Wybrane parametry wyświetlacza MI0283QT-9A

Element	Wartość	Jednostka
Typ LCD	TFT/Transmissive/Positive	
Obszar aktywny	43.2 x 57.6	mm^2
Liczba punktów	240 (RGB) x 320	
Powierzchnia piksela	0.18 x 0.18	mm^2
Sterownik	ILI9341	
Kolory	262K	
Zużycie energii	220	mw
Napięcie wejściowe	2.8	V
Waga	24.3	g

Podstawy technologii TFT LCD

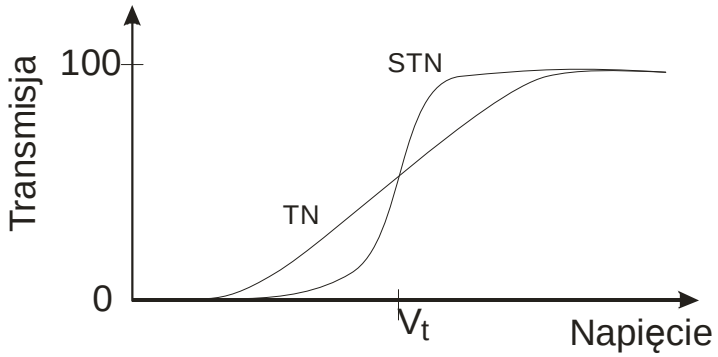
Wyświetlacz TFT LCD, thin-film-transistor liquid-crystal display , jest to wyświetlacz ciekłokrystaliczny wykonany z wykorzystaniem technologii cienkowarstwowych tranzystorów (thin-film-transistors) o tzw. aktywnej matrycy. Wyświetlacz składa się bowiem z siatki (matrycy) aktywnych elementów (pikseli), których liczba sięga tysięcy lub nawet milionów. Każdy taki piksel jest sterowany za pomocą dołączonego do niego tranzystora cienkowarstwowego, dzięki czemu wybrany piksel może być albo włączony albo wyłączony.

W samej technologii LCD wykorzystuje się ciekłe kryształy, które są sterowane napięciem (przez pojęcie *ciekły kryształ* rozumiemy stan skupienia materii, w którym posiada ona zarówno właściwości cieczy, np. zdolność do płynięcia, ale także własności kryształu, np. silne uporządkowanie cząsteczek). Zwykle stosuje się dwa typy ciekłych kryształów – wykonane w technologii TN (twisted nematic – dającej obrót o 90 stopni) oraz STN (super twisted nematic, dającej obrót o 270 stopni). Przykładowa ilustracja działania ciekłego kryształu TN pokazana została na rysunku poniżej, na którym z lewej strony światło nie zostaje zablokowane przez odpowiednią płytkę polaryzującą (piksel jasny), zaś z prawej strony - dzięki dołączonemu napięciu światło zostaje zablokowane (piksel ciemny).



Rysunek 3. Schemat działania wyświetlacza TN, opis skrótów E – elektroda, LC – ciekły kryształ, G – płytka szklana, P – płytka polaryzacyjna; (Uprawnienia GNU Licence - Creative Commons Attribution-Share Alike 3.0 Unported)

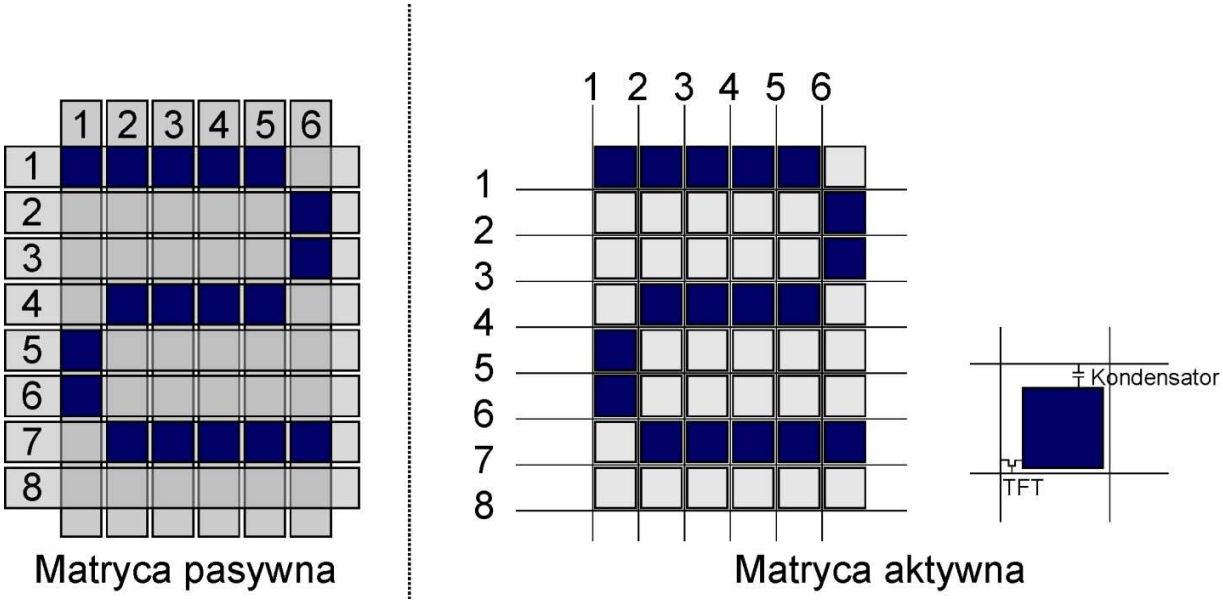
Zastosowanie obrotu większego niż 90 stopni pociąga za sobą polepszenie jakości obrazu, a także zwiększenie kąta, pod którym można oglądać obraz. Dodatkowo w przypadku kryształów STN charakterystyka pomiędzy mocą przepuszczoną a przyłożonym do elektrod napięciem jest znacznie bardziej stroma, dzięki czemu można zwiększyć częstotliwość przełączania pomiędzy stanami włączonym i wyłączonym (charakterystyka ta zobrazowana została na rys. 4, pochodzącym z http://www.mc2.chalmers.se/pl/lc/engelska/applications/Displays.html)



Rysunek 4. Zależność pomiędzy mocą transmitowaną (w zakresie od 0 do 100%) a przyłożonym napięciem w przypadku kryształów TN oraz STN

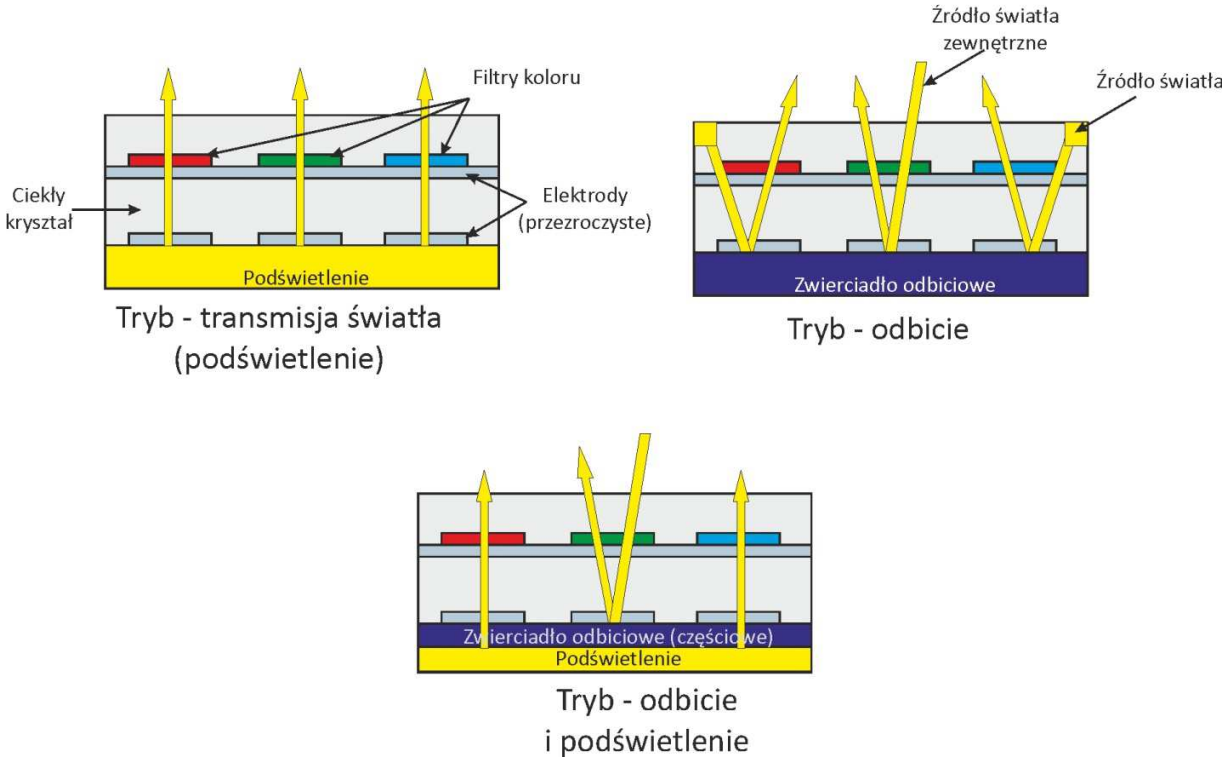
Poza kryształami TN czy STN istnieje także wiele innych rozwiązań, pozwalających polepszyć działanie wyświetlaczy ciekłokrystalicznych.

Wyświetlacze ciekłokrystaliczne mogą posiadać strukturę pasywna albo aktywną. W przypadku tej pierwszej (Passive Matrix LCD – PMLCD), napięcie do wybranego piksela jest kierowane poprzez dostarczenie napięcia do odpowiedniej kolumny i rzędu w macierzy pikseli (patrz rys. 5). Rozwiązanie to charakteryzuje się prostota produkcji, jednak również długim czasem reakcji. Dzięki zastosowaniu aktywnych matryc (Active Matrix LCD – AMLCD), każdy piksel jest sterowany za pomocą cienkowarstwowego tranzystora oraz kondensatora (rys. 5). W przypadku zastosowania wyświetlaczy kolorowych wykorzystywane są polaryzatory do odpowiednich kolorów.



Rysunek 5. Porównanie technologii z aktywną i pasywną matrycą pikseli

Warto także wspomnieć, że wyświetlacze mogą pracować w dwóch trybach w zależności od źródła pochodzenia światła – może ono pochodzić albo z otoczenia (reflective mode, wymagane użycie luster), albo z dołożonego dedykowanego źródła światła (transmittive mode). Istnieje także możliwość połączenia tych trybów (rys. 6).



Rysunek 6. Trzy tryby pracy wyświetlaczy LCD

Używana jest także nazwa: „positive" oraz „negative", określających sposób wyświetlania. W przypadku tych pierwszych, kiedy piksel jest wyłączony wówczas jest przezroczysty, natomiast kiedy jest włączony, jest nieprzezroczysty. Ma to zastosowanie w przypadku kiedy np. wyświetlane są czarne litery na białym tle w miejscach o dużej intensywności światła zewnętrznego. Piksele włączone są nieprzezroczyste więc będą widoczne jako czarne, większość zaś będzie wyłączona i widoczna jako biała. W przypadku wyświetlaczy „negative LCD" sytuacja jest odwrotna, mają one zastosowanie w sytuacji, kiedy intensywność światła zewnętrznego jest niewielka, wówczas istotne jest zapewnienie silnego podświetlenia wyświetlacza. W takim przypadku włączone piksele będą przezroczyste pozwalając na przenikane światła w kierunku użytkownika.

Interfejs 8080

Typowo procesory obsługujące wyświetlacze TFT LCD komunikują się za pomocą interfejsów równoległych Intel 8080 oraz Motorola 6800. W przypadku wyświetlacza użytego na płycie ewaluacyjnej Stellaris EasyMxPro v7, zastosowany został równoległy 8-bitowy interfejs 8080 oraz dodatkowe linie sterujące. Płyta posiada także sterownik podświetlenia, który umożliwia regulację jasności za pomocą sygnału PWM.

Przez interfejs 8080 rozumiemy sposób kontroli przepływu danych stosowany pierwotnie przez firmę INTEL. W protokole tym stosowane są następujące linie komunikacyjne (UWAGA, oznaczenia linii i ich znaczenie różni się w zależności od producenta chipu, a także wybranej wersji interfejsu):

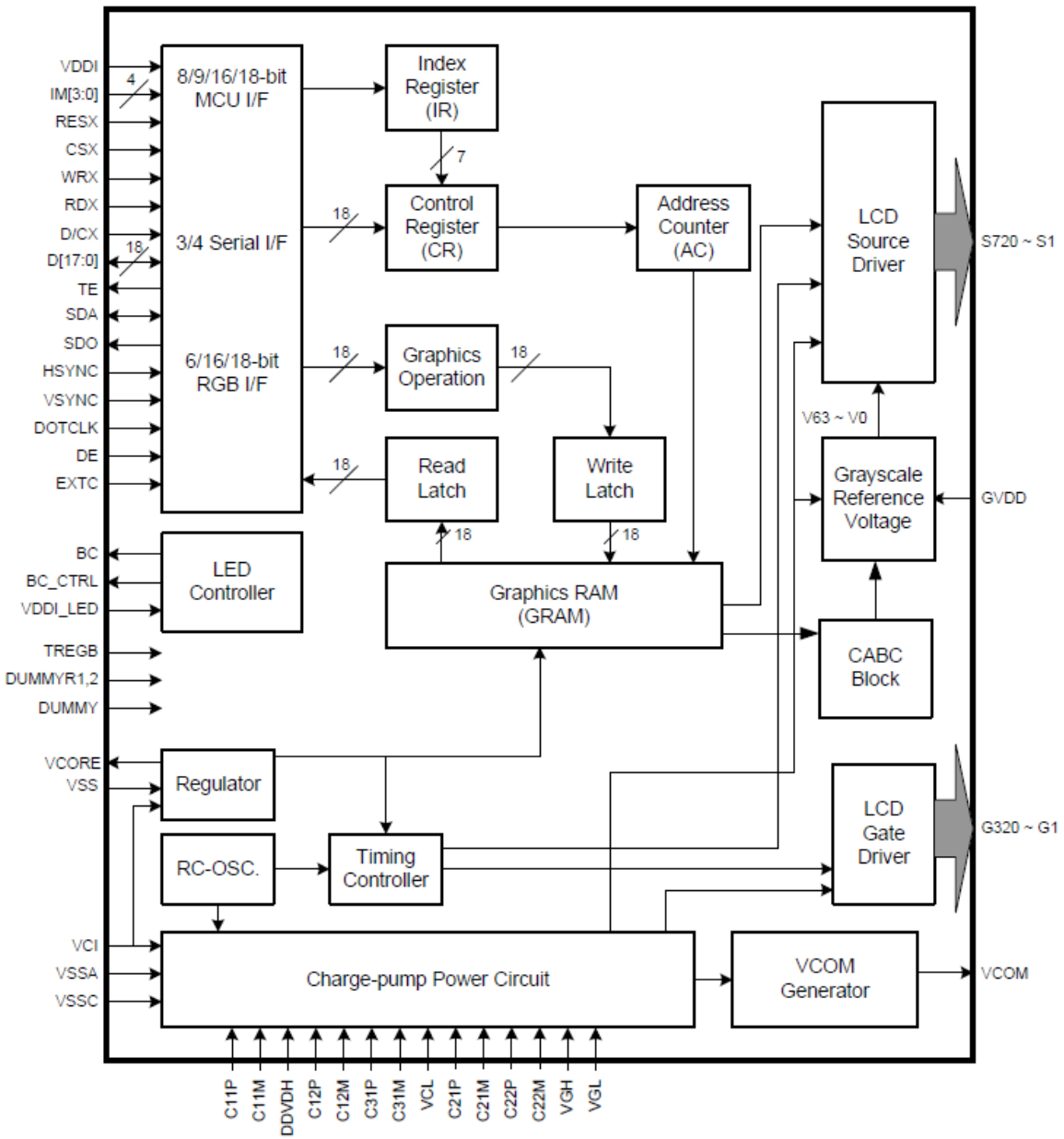
- Osiem dwukierunkowych linii komunikacyjnych D0-D7

- Linia E (Enable) – zezwolenie na wykonanie operacji
- Linia R/W# (Read/Write) – linia sterująca zapisem albo odczytem danych
- Linia D/C# - wybór pomiędzy danymi a instrukcjami sterującymi
- Linia C/S# (Chip Select) – wybór układu

Jak wspomniano, na płycie EasyMxPro v 7.0 został zastosowany sterownik wykonany w technologii SOC firmy ILITEK ILI9341 (dokumentacja sterownika, „a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color", dostępna jest pod adresem https://www.displaytech-us.com/sites/default/files/driver-ic-data-sheet/ILI9341_DS_V1.10_20110415.pdf

Sterownik

Schemat sterownika przedstawia rysunek 7.



Rysunek 7. Schemat sterownika wyświetlacza

Biblioteki

Aby możliwa była sprawna obsługa wyświetlacza z poziomu języka C/C++ konieczne jest zainstalowanie odpowiednich bibliotek, w tym przypadku pliki te – dostępne pod ogólną nazwą *grlib* - dostępne są jako fragment biblioteki TivaWare opisanej już w ćwiczeniu nr I. Wszystkie wykorzystywane przy obsłudze wyświetlacza pliki są po instalacji umieszczone w katalogu <*KATALOG_INSTALACYJNY/grlib*, gdzie <KATALOG_INSTALACYJNY> to miejsce, do którego wgrane zostały biblioteki TivaWare (np. *C:/TI/TivaWare*). Ścieżka dostępu do pliku wykorzystywanego podczas linkowania jest następujaca <*KATALOG_INSTALACYJNY/grlib/ccs/Debug/grlib.lib* i analogicznie do tego, jak to miało miejsce w przypadku biblioteki GPIO, należy poinformować Linker o miejscu wgrania biblioteki. W tym celu

należy otworzyć okno dialogowe z własnościami projektu, wybrać opcje linkera i dodać odpowiednią bibliotekę (CCS Build -> ARM Linker -> File Search Path).

W katalogu *<KATALOG_INSTALACYJNY/grlib* dostępne są pliki .c oraz .h, w których zdefiniowane zostały różnorodne funkcje, pozwalające na swobodne używanie wyświetlacza. Przykładami takich plików są: *line.c* albo *rectangle.c* pozwalające odpowiednio na rysowanie linii lub elementów prostokątnych.

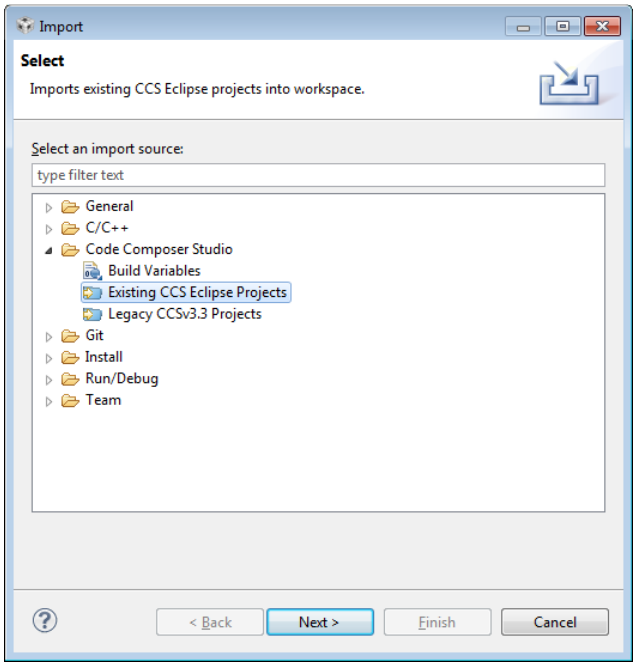
Jednak aby poprawnie skomunikować się z kontrolerem wyświetlacza (ILI9341) konieczne jest wykorzystanie odpowiedniego sterownika. Sterownik taki (ili9341_240x320x262K.c) został przygotowany przez pracowników Texas Instruments i dostępny jest na stronach internetowych pracowników prowadzących zajęcia.

Przebieg ćwiczenia

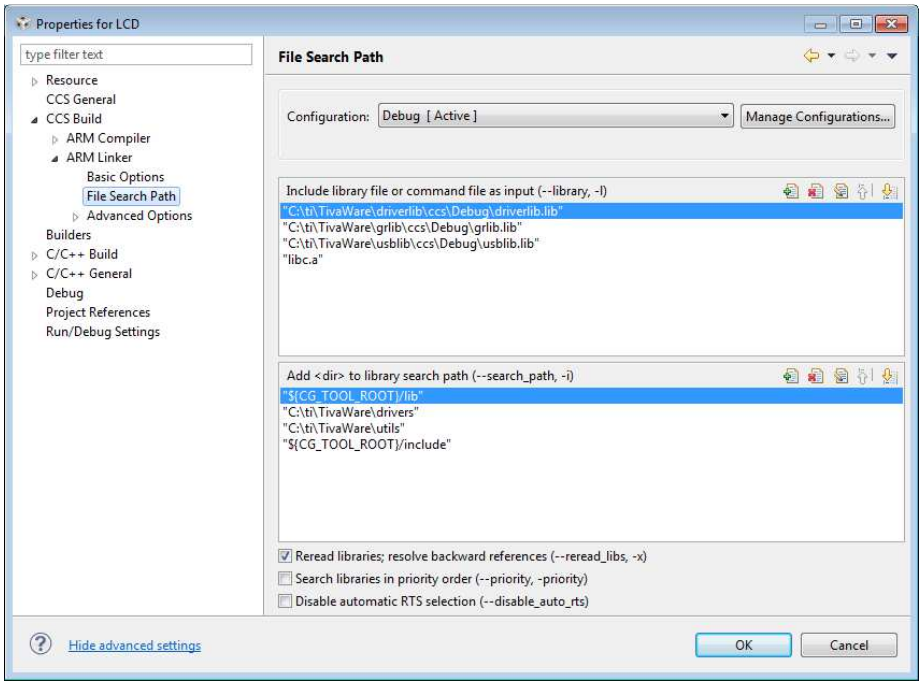
Program 1

Pierwszy program ma zapoznać z podstawami obsługi wyświetlacza. W tym celu należy ściągnąć ze strony internetowej odpowiednie pliki (Lab_2.rar). Następnie należy postępować zgodnie z krokami poniżej.

1. W programie CCS należy zaimportować pliki projektu. W eksploratorze projektów należy kliknąć prawym przyciskiem myszy i wybrać opcję „Import”, a następnie „Code Composer Studio -> Existing CCS Eclipse Project”.
2. W oknie Import CCS Eclipse project poprzez naciśnięcie przycisku „Brwose…” odszukać ścieżkę do importowanego projektu, zaznaczyć opcję „copy project into workspace” (o ile nie znajduje się wcześniej w folderze aktualnej przestrzeni roboczej) i nacisnąć przycisk „Finish”



Należy zwrócić uwagę, że w projekcie występują dodatkowe biblioteki (opisane pokrótce wcześniej), które należało dołączyć do opcji linkera. Bieżące ustawienia można sprawdzić klikając sprawdzając odpowiednie właściwości projektu („CCS Build -> ARM Linker ->File Search Path”)



3. Należy przeanalizować kod programu dostępny w pliku main.c

```
1.  //*****************************************************************************
2.  // hello.c - Simple hello world example.
3.  //
4.  // Maciej Kucia July 2013
5.  //
6.  // This is part of revision 1.0 of the EK-LM4F232 Firmware Package.
7.  //*****************************************************************************

8.  #include <stdint.h>
9.  #include <stdbool.h>
10. #include "inc/hw_memmap.h"
11. #include "driverlib/fpu.h"
12. #include "driverlib/sysctl.h"
13. #include "driverlib/rom.h"
14. #include "driverlib/pin_map.h"
15. #include "driverlib/uart.h"
16. #include "grlib/grlib.h"
17. #include "drivers/ili9341_240x320x262K.h"
18. #include "utils/uartstdio.h"
19. #include "driverlib/gpio.h"

20. //*****************************************************************************
21. //
22. // TI logo in array form (1 bit per pixel)
23. //
24. //*****************************************************************************
25. const unsigned char TI_logo[(16*2)+5] = {
26. IMAGE_FMT_1BPP_UNCOMP, 16,0, 16,0,
27. 0x0f,0xff,0x0f,0xff,0x0f,0xf3,0x0f,0xfa,0x0f,0x8f,0x0f,0x89,0x81,0x99,0x
    81,0x19,0x03,0x09,0x07,0x89,0x07,0xdf,0x3f,0xee,0x7f,0xf0,0xff,0xf8,0xff
    ,0xf9,0xff,0xfb};
```

```
28. //     1,1,1,1,0,0,0,0, 1,1,1,1,1,1,1,1,
29. //     1,1,1,1,0,0,0,0, 1,1,1,1,1,1,1,1,
30. //     1,1,1,1,0,0,0,0, 1,1,0,0,1,1,1,1,
31. //     1,1,1,1,0,0,0,0, 0,1,0,1,1,1,1,1,
32. //     1,1,1,1,0,0,0,0, 1,1,1,1,0,0,0,1,
33. //     1,1,1,1,0,0,0,0, 1,0,0,1,0,0,0,1,
34. //     1,0,0,0,0,0,0,1, 1,0,0,1,1,0,0,1,
35. //     1,0,0,0,0,0,0,1, 1,0,0,1,1,0,0,0,
36. //     1,1,0,0,0,0,0,0, 1,0,0,1,0,0,0,0,
37. //     1,1,1,0,0,0,0,0, 1,0,0,1,0,0,0,1,
38. //     1,1,1,0,0,0,0,0, 1,1,1,1,1,0,1,1,
39. //     1,1,1,1,1,1,0,0, 0,1,1,1,0,1,1,1,
40. //     1,1,1,1,1,1,1,0, 0,0,0,0,1,1,1,1,
41. //     1,1,1,1,1,1,1,1, 0,0,0,1,1,1,1,1,
42. //     1,1,1,1,1,1,1,1, 1,0,0,1,1,1,1,1,
43. //     1,1,1,1,1,1,1,1, 1,1,0,1,1,1,1,1
44. //*****************************************************************************
45. //
46. // Print some text to the display.
47. //
48. //*****************************************************************************
49. int
50. main(void)
51. {
52. tContext sContext;
53. tRectangle sRect;
54. //
55. // Enable lazy stacking for interrupt handlers. This allows floating-point
56. // instructions to be used within interrupt handlers, but at the expense of
57. // extra stack usage.
58. //
59. ROM_FPULazyStackingEnable();
60. //
61. // Set the clocking to run directly from the crystal.
62. //
63. ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
    |
                    SYSCTL_OSC_MAIN);
64. //
65. // Initialize the display driver.
66. //
67. ILI9341_240x320x262K_Init();
68. //
69. // Initialize the graphics context.
70. //
71. GrContextInit(&sContext, &g_sILI9341_240x320x262K);
72. //
73. // Print some text
74. //
75. GrContextFontSet(&sContext, g_psFontCm12);
76. GrStringDrawCentered(&sContext, "Hello", -1,
                    GrContextDpyWidthGet(&sContext) / 2,
```

```
                    (GrContextDpyHeightGet(&sContext) / 2),
                    0);

    GrContextForegroundSet(&sContext, ClrCrimson);

    GrContextFontSet(&sContext, g_psFontCm20b);
    GrStringDrawCentered(&sContext, "EUP", -1,
                    GrContextDpyWidthGet(&sContext) / 2,
                    20 + (GrContextDpyHeightGet(&sContext) / 2),
                    0);

    //
    // Draw TI logo
    //
    sRect.i16XMin = 0;
    sRect.i16YMin = 220;
    sRect.i16XMax = 319;
    sRect.i16YMax = 239;
    GrContextBackgroundSet(&sContext, ClrWhite);
    GrContextForegroundSet(&sContext, ClrRed);
    GrRectFill(&sContext, &sRect);
    GrImageDraw(&sContext, TI_logo, 200, 222);

    GrContextForegroundSet(&sContext, ClrWhite);
    GrContextFontSet(&sContext, g_psFontFixed6x8);
    GrStringDraw(&sContext, "Texas", -1,218,222,0);
    GrStringDraw(&sContext, "Instruments", -1,218,230,0);

    //
    // Flush any cached drawing operations.
    //
    GrFlush(&sContext);


    //
        // We are finished. Hang around doing nothing.
        //
        unsigned long ii = 1;
        while(1)
        {
        //
        // Draw RGB rectangles
        //
       sRect.i16XMin = 20;
       sRect.i16YMin = 20;
       sRect.i16XMax = 50;
       sRect.i16YMax = 50;
//      unsigned long temp_color = ColorTranslate(void *pvDisplayData, ii);
       ii=ii+0.01;
       GrContextForegroundSet(&sContext, ii);
       GrRectFill(&sContext, &sRect);

       ROM_SysCtlDelay(ROM_SysCtlClockGet()/2);

       GrContextForegroundSet(&sContext, ClrGreen);
       GrRectFill(&sContext, &sRect);
```

```
119.     ROM_SysCtlDelay(ROM_SysCtlClockGet()/2);

120.     GrContextForegroundSet(&sContext, ClrBlue);
121.     GrRectFill(&sContext, &sRect);

122.     ROM_SysCtlDelay(ROM_SysCtlClockGet()/2);
123.      }
124.      }
```

Opis wybranych fragmentów kodu:

Fragment 1, linie 25-27

```
25. const unsigned char TI_logo[(16*2)+5] = {
26. IMAGE_FMT_1BPP_UNCOMP, 16,0, 16,0,
27. 0x0f,0xff,0x0f,0xff,0x0f,0xf3,0x0f,0xfa,0x0f,0x8f,0x0f,0x89,0x81,0x99,0x81,0x1
    9,0x03,0x09,0x07,0x89,0x07,0xdf,0x3f,0xee,0x7f,0xf0,0xff,0xf8,0xff,0xf9,0xff,0
    xfb};
```

Zostaje tutaj utworzona tablica (typ const unsigned char), pozwalająca na zdefiniowanie obrazka piksel po pikselu. Sam obrazek ma rozmiar 16 na 16 pikseli (czyli razem 32 bajty), zaś tablica dodatkowo zawiera pięć pól opisujących parametry obrazka. Są to:

- `IMAGE_FMT_1BPP_UNCOMP – Image format, 1 bit per pixel, uncompressed`
- `16, 0, – usWidth (szerokość obrazka, typ short zawierający 2 bajty, litery us oznaczają unsigned short)`
- `16, 0, – usHeight (wysokość obrazka, typ short zawierający 2 bajty)`

Warto zwrócić uwagę, że format tablicy jest zgodny ze strukturą *tImage*, proponowaną w bibliotece grlib.h dla procesora (zobacz s 16, opis biblioteki TivaWare Graphics Library). Z tego powodu definicja obrazka piksel po pikselu występuje jako ostatnia.

Fragment 3, linia 59

```
59. ROM_FPULazyStackingEnable();
```

Istnieją trzy istotne funkcje związane z przetwarzaniem zmiennoprzecinkowym (floating-point) *FPUStackingEnable(void)*, *FPULazyStackingEnable(void)* oraz *FPUStackingDisable(void)*. W funkcjach tych FPU oznacza Floating-Point Unit, czyli jednostkę odpowiedzialną za działania zmiennoprzecinkowe. Do operacji zmiennoprzecinkowych wykorzystywane są rejestry s0-s15. W przypadku wywołania funkcji *FPUStackingEnable(void)*, podczas obsługi przerwania rejestry s0 do s15 są odkładane na stos i przywracane przy powrocie z procedury obsługi przerwania. Wywołanie drugiej funkcji, *FPULazyStackingEnable(void),* rezerwuje na stosie miejsce do przechowania rejestrów s0-s15, jednak są one tam przepisywane tylko w przypadku, kiedy operacje zmiennoprzecinkowe są wykonywane w przerwaniu (co spowodowałoby nadpisanie rejestrów). Taka konfiguracja pozwala skrócić czas opóźnienia spowodowanego przez przerwanie w sytuacji, kiedy nie są wykorzystywane operacje zmiennoprzecinkowe wewnątrz przerwania. Ostatnia funkcja wyłącza możliwość przechowywania na stosie rejestrów s0-s15.

Fragment 3, linia 67

```
67. ILI9341_240x320x262K_Init();
```

Inicjalizacja sterownika – dzięki wywołaniu tej funkcji utworzone zostają struktury typowe dla użytego na płycie ewaluacyjnej wyświetlacza i jego sterownika - g_sILI9341_240x320x262K. Dzięki temu w kolejnych liniach można dokonać inicjalizacji samego wyświetlacza, tj. przypisać utworzoną strukturę do zmiennej *sContext*, opisującą parametry całego wyświetlacza i wyświetlanego obrazu (kontekstu).

Fragment 4, linia 75

```
75. GrContextFontSet(&sContext, g_psFontCm12);
```

Dzięki wykonaniu funkcji *GrContextFontSet* ustalany jest bieżący krój czcionki wykorzystywany przez zmienną *sContext*. Warto wspomnieć, że pełna lista dostępnych czcionek została umieszczona w rozdziale 15 dokumentu [5].
UWAGA: BRAK USTAWIENIA CZCIONKI SPOWODUJE BŁĄD PODCZAS WYKONYWANIA PROGRAMU, NIE MA DOMYŚLNIE USTAWIONEJ CZCIONKI

Fragment 5, linia 76

```
76. GrStringDrawCentered(&sContext, "Hello", -1,
                         GrContextDpyWidthGet(&sContext) / 2,
                         (GrContextDpyHeightGet(&sContext) / 2),0);
```

Funkcja *GrStringDrawCentered* ma za zadania wyświetlenie na ekranie napisu określonego drugim argumentem funkcji. Definicja funkcji jest następująca:

```
#define GrStringDrawCentered(psContext, pcString, i32Length, i32X, i32Y, bOpaque)
```

gdzie pierwszy element to wskaźnik do struktury kontekstu, drugi to wskaźnik do obiektu przechowującego tekst do wyświetlenia, i32Length to liczba znaków w tekście (-1 oznacza, że cały test ma być wyświetlony, zezwalając na potencjalne obcięcie liter), i32X oraz i32Y to pozycja tekstu na ekranie (względem której następuje wyśrodkowywanie), natomiast ostatnia zmienna bOpaque określa, czy tło litery ma być wyświetlone, czy pozostawiony ma być bieżący kolor tła. Należy zwrócić uwagę na sposób opisu zmiennych, i32X oznacza zmienną typu całkowitego o długości 4 bajtów (Int32).

Fragment 6, linia 87-90

```
87. GrContextBackgroundSet(&sContext, ClrWhite);
88. GrContextForegroundSet(&sContext, ClrRed);
89. GrRectFill(&sContext, &sRect);
90. GrImageDraw(&sContext, TI_logo, 200, 222);
```

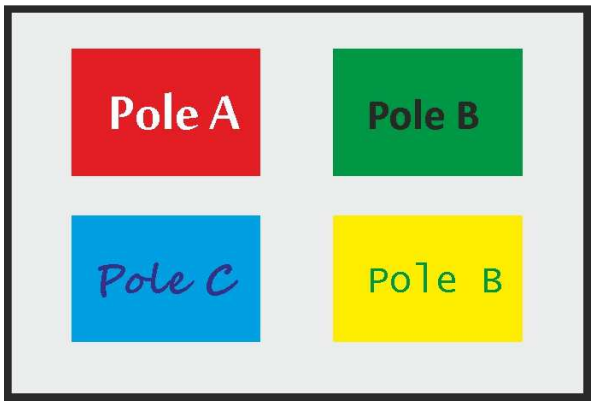
Funkcje: *GrContextBackgroundSet* oraz *GrContextForegroundSet* mają za zadanie ustawienie kolorów tła oraz pierwszego planu. W przypadku wyświetlania grafiki, w której kolor na każdym pikselu jest określony za pomocą jednego bitu, obraz tworzony jest właśnie za pomocą kolorów tła i pierwszego planu. W sytuacji, kiedy dany piksel ma wartość 1, wówczas wyświetlany jest kolor pierwszego planu, kiedy zaś 0 – wówczas tła. W przypadku wykorzystania palet kolorów (kiedy np. kolory RGB na jednym pikselu są określane za pomocą 4 bitów) należy zapoznać się z wymaganiami biblioteki.

W linii 89 następuje wypełnienie kolorem wskazanego obszaru prostokątnego, natomiast wykonanie instrukcji z linii 90 skutkuje wyświetleniem grafiki TI_logo, gdzie liczby 200 oraz 222 są współrzędnymi względem lewego górnego narożnika obszaru prostokątnego, w którym ma

być renderowny obrazek. Funkcja sama rozpoznaje, w jakiej postaci został zapisany obraz (tzn. za pomocą jakiej liczby bitów reprezentowane są kolory na jednym pikselu.

Zadanie 1

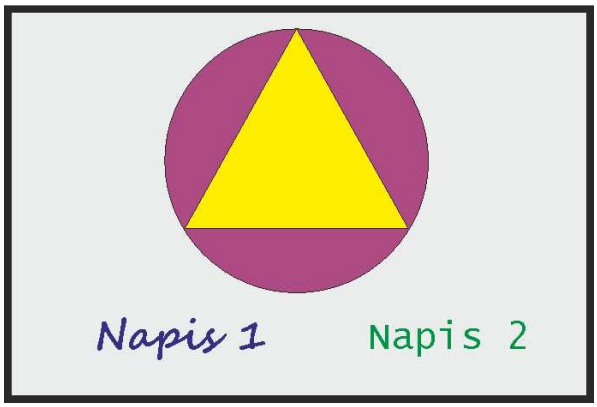
Bazując na Programie 1 przeanalizowanym powyżej należy napisać program, który na wyświetlaczu przedstawi następującą strukturę:



Każdy prostokąt powinien być innego koloru, podobnie każdy napis powinien być przedstawiony z wykorzystaniem innej czcionki.

Zadanie 2

Bazując na Programie 1 przeanalizowanym powyżej należy napisać program, który środku wyświetlacza narysuje koło o zadany kolorze, wewnątrz koła trójkąt o innym kolorze, zaś pod tymi figurami umieszczony będzie napis informujący o wykorzystanych kolorach (patrz rysunek poniżej). Trójkąt nie musi być idealnie wpasowany w koło. Jedną z możliwości utworzenia trójkąta jest utworzenie obiektu typu Image lub tablicy znakowej, jak to miało miejsce w przykładzie z logo firmy TexasInstruments.



Uwaga:

Aby stworzyć własny obrazek jak powyżej należy albo narysować trójkąt za pomocą linii o zmieniającej się długości (linie są rysowane jedna pod drugą), albo stworzyć własny obraz w programie graficznym.

Poniżej przedstawiona została metoda utworzenia własnego obrazka i jego wykorzystanie w naszym projekcie.

1. Otwórz program graficzny GIMP (lub inny), utwórz obrazek o rozmiarze 320 na 240 pikseli lub mniejszym, w zależności od wymaganej wielkości obrazka. Pamiętaj o fizycznych rozmiarach wyświetlacza.
2. Zmień tryb obrazka na indeksowany (`Obraz -> Tryb -> Indeksowany`) i wybierz maksymalną liczbę kolorów palety na 16. Zapisz zmiany.
3. Utworzony rysunek należy wyeksportować do pliku `obraz.pnm`. Rozszerzenie pnm (ang. *portable anymap*) reprezentuje obrazy przenoszone pomiędzy różnymi urządzeniami.

   Jeżeli ktoś nie chce korzystać z programu GIMP, można także wykorzystać dostępne w Internecie konwertery pomiędzy innymi graficznymi formatami a formatem pnm

4. Otwórz linie poleceń i przejdź do katalogu, gdzie umieszczone są biblioteki TivaWare (np. `C:\TI\TivaWare`), a następnie otwórz katalog *`tools\bin`* (tj. `C:\TI\TivaWare\tools\bin`). Wykorzystując program pnmtoc.exe zamień wygenerowany wcześniej plik obraz.pnm na plic obraz.c. Wykorzystaj do tego komendę: `pnmtoc –c obraz.pnm > obraz.c`
5. Dodaj utworzony plik `obraz.c` do swojego projektu (kliknij prawym przyciskiem na nazwę projektu i wybierz „*Add files…*”)
6. Otwórz plik `obraz.c` (jeżeli występują błędy w pliku, mogą być one związane z formatowaniem; aby je wyeliminować otwórz program Notatnik i przekopiuj tam zawartość pliku `obraz.c`, następnie nadpisz plik `obraz.c`). U góry pliku, gdzie zdefiniowany jest obraz w formacie zrozumiałym dla kompilatora, dodaj następujące linie:

   ```
   #include <stdint.h>
   #include <stdbool.h>
   #include <grlib/grlib.h>
   ```

7. W pliku głównym `main.c` dodaj linię:

   ```
   extern const uint8_t g_pui8Image[];
   ```

8. W celu wyświetlenia obrazka wykorzystaj metody przedstawione w celu wyświetlenia znaku firmy TexasInstruments, np. `GrImageDraw(&sContext, g_pui8Image, 0, 0);`

Zadanie 3

Wykorzystując inne dostępne funkcje z biblioteki GrLib stwórz dowolny układ graficzny (np. składający się z linii, kół, prostokątów etc.)

Część II - Dżojstik

Jak już wspomniano w laboratorium nr 1 (dot. GPIO), jednym z dostępnych elementów płyty EasyMx Pro 7 jest dżojstik (Navigation Switch), pozwalający na realizację pięciu typów zdarzeń: przechylenie dżojstika w lewo, w prawo, do góry albo w dół, a także przyciśnięcie (na środku). Każde zdarzenie odpowiada fizycznemu przyciśnięciu jednego z pięciu guzików. W celu uaktywnienia funkcjonalności dżojstika należy przestawić odpowiednie przełączniki na pozycję „+" (w górę), są to odpowiednio SW2.1 (dla naciśnięcia dżojstika w lewo), SW2.8 (w górę), SW5.3 (w dół), SW5.4 (w prawo), o SW8.6 (przyciśnięcie) – zobacz Rysunek 8. Dodatkowo, guziki dżojstika są podłączone do następujących pinów mikrokontrolera: PB0, PB7, PE4, PE5, PH2, dla odpowiednio kierunków „do góry", „w lewo", „w prawo", „W dół" oraz „przyciśnięcie".

UWAGA UWAGA UWAGA

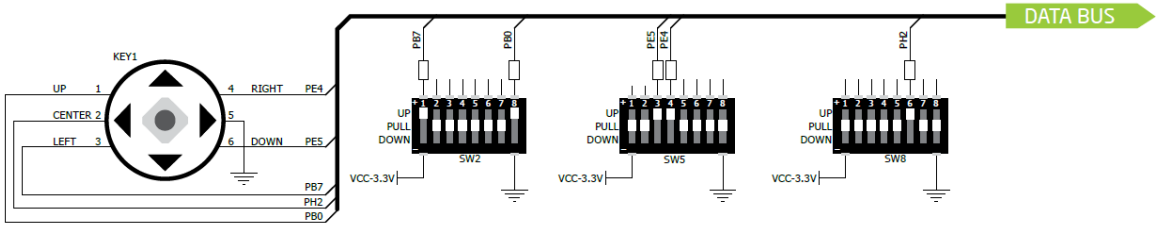
UWAGA 1: przełącznik SW2 jest oznaczony na płycie jako PORT B, SW5 – jako PORT E, oraz SW8 – jako port H.

UWAGA 2: Pin PB7 jest podłączony do mikrokontrolera przez pin PK7.

UWAGA 3: naciśnięcie dowolnego ze wspomnianych przycisków skutkuje podłączeniem wybranego pinu do masy (zera), co pozwala na wykrycie zdarzenia przez mikroprocesor

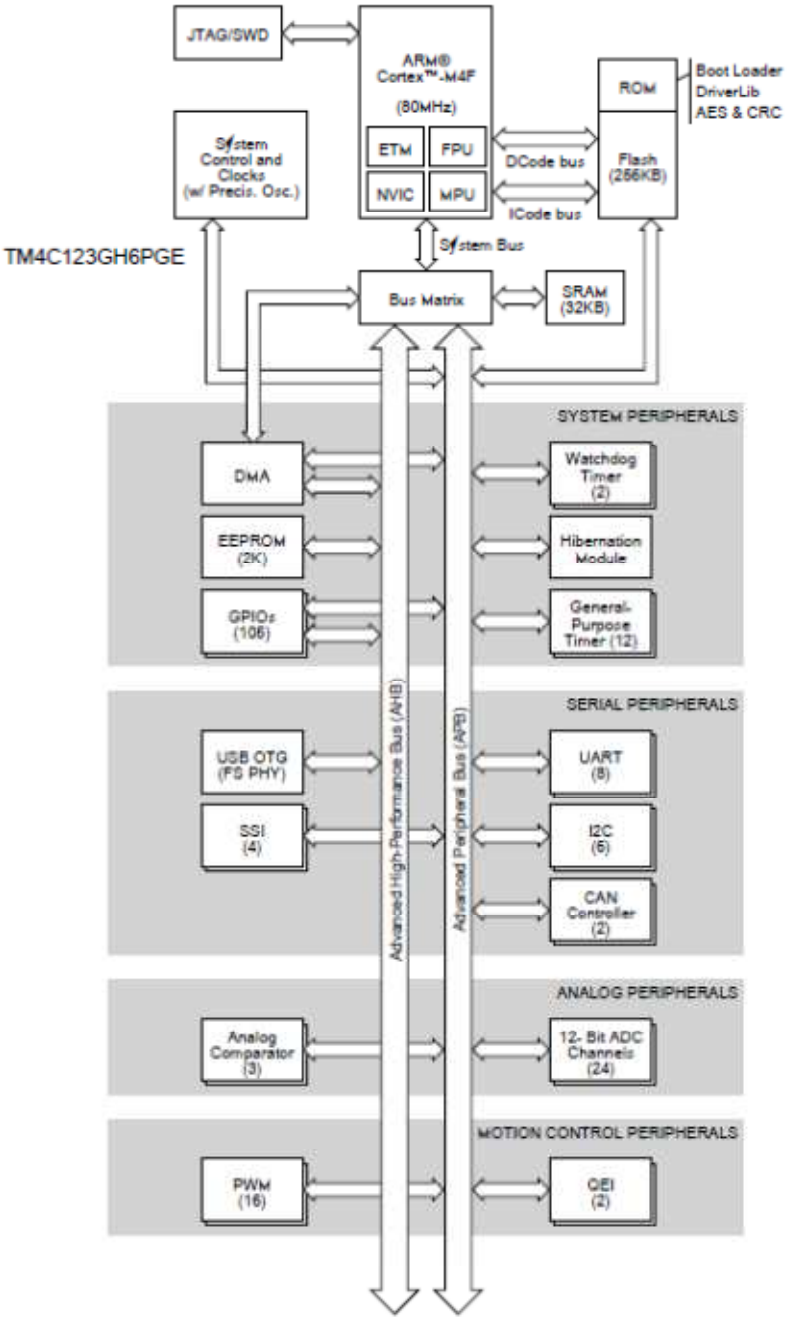
UWAGA 4: do obsługi dżojstika należy korzystać z bibliotek GPIO, omawianych na laboratorium nr 1.

Schemat podłączenia dżojstika został przedstawiony na rysunku poniżej:



Rysunek 8. Schemat podłączenia dżojstika

W przypadku obsługi dżojstika należy dodatkowo zwrócić uwagę na architekturę procesora, a w szczególności na obecność dwóch szyn, dzięki którym urządzenia zewnętrzne (GPIO) są sterowane z poziomu mikrokontrolera. Są to magistrale AHB (Advanced High-Performance Bus) oraz APB (Advanced Periperal Bus). Magistrala APB zapewnia kompatybilność wstecz ze starszymi komponentami, natomiast magistrala AHB jest nowszym rozwiązaniem zapewniającym większa efektywność przesyłania danych. Magistrala AHB jest sterowana za pomocą rejestru GPIOHBCTL (s. 258 pozycji [3]). Dostęp do portów K, L, M, M oraz P jest zagwarantowany tylko poprzez magistralę AHB.



Rysunek 9. Schemat blokowy procesora

Program 2

Uruchomić program CCS oraz korzystając z szablonu utworzonego podczas Ćwiczenia nr 1 utworzyć nowy projekt. Własności projektu powinny umożliwić dostęp do odpowiednich katalogów, w których umieszczone są biblioteki i sterowniki do wyświetlacza. W głównym pliku projektu należy umieścić poniższy fragment kodu, który odpowiedzialny jest za wyświetlenie napisu „GORA" w dwóch różnych kolorach w zależności od tego, czy dżojstik został naciśnięty w kierunku „w górę" czy też nie.

```
(…)
    1. SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);

    2. //
    3. // Set all GPIOB pins as inputs
    4. //

    5. GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PINS_ALL);

    6. //    GPIOPinTypeGPIOInput(GPIO_PORTH_BASE, GPIO_PINS_ALL);

    7. GrContextFontSet(&sContext,g_psFontCm16b);
    8. GrContextForegroundSet(&sContext, ClrBlue);
    9. int temp_var = 0;
    10. while(1){
              temp_var = GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_0);
              if (temp_var == 1)
              {
                 GrContextForegroundSet(&sContext, ClrBlue);
                 GrStringDrawCentered(&sContext, "GORA", -1,
                                 GrContextDpyWidthGet(&sContext) / 2,
                                 40 + (GrContextDpyHeightGet(&sContext) / 2),
                                 0);
              }
              else
              {
                 GrContextForegroundSet(&sContext, ClrRed);
                 GrStringDrawCentered(&sContext, "GORA", -1,
                                          GrContextDpyWidthGet(&sContext) / 2,
                                          40 + (GrContextDpyHeightGet(&sContext) /
                                          2),
                                          0);
              }
        }
(…)
```

W przypadku pozostałych portów do obsługi dżojstika należy zwrócić uwagę, że domyślnie są one obsługiwane w trybie AHB.

UWAGA UWAGA UWAGA

UWAGA1: Jeżeli dany port wymaga użycia trybu AHB należy odpowienio poinformować o tym fakcie kompilator stosując instrukcje analogiczne do poniższej:

```
GPIOPinTypeGPIOInput(GPIO_PORTH_AHB_BASE, GPIO_PIN_2);
```

Wyświetlacz LCD wykorzystuje także port H w trybie AHB. W przypadku wykorzystania wyświetlaczna razem z dżojstkiem nie wolno uaktywniać jako wejście wszystkich pinów portu H. Poniższa instrukcja jest błędna i spowoduje "zawieszenie się" program.

```
GPIOPinTypeGPIOInput(GPIO_PORTH_AHB_BASE, GPIO_PINS_ALL);
```

UWAGA2: Ustawienie portów powinno następować po ustaleniu częstotliwości pracy układu.

UWAGA3: Jeżeli wykorzystanie portu AHB nie jest możliwe lub wskazane, należy odpowiednio zmodyfikować progra. Zanim ustawi się poszczególne porty jako wejście albo jako wyjście, należy wyłączyć obsługe magistrali AHB dla tych portów, nastepnie włączyć obsługę magistrali APB oraz sprawdzić, czy zmiany zostały zapisane, a dopiero w nastepnej kolejności ustawić poszczególne porty jako wejście lub wyjście.

```
    SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOH);
    SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOD);
    SysCtlGPIOAHBDisable(SYSCTL_PERIPH_GPIOH);
    SysCtlGPIOAHBDisable(SYSCTL_PERIPH_GPIOB);

    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOD);
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOH);
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOK);

    _Bool gzd = true;
    while(gzd)
    {
    if(SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB) &&
             SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE) &&
             SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOH))
                 gzd = false;
    else
                 gzd = true;
    }
GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_0);
GPIOPinTypeGPIOInput(GPIO_PORTK_BASE, GPIO_PIN_7);
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, 0xf0);
GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_4|GPIO_PIN_5);
GPIOPinTypeGPIOInput(GPIO_PORTH_BASE, GPIO_PIN_2);
```

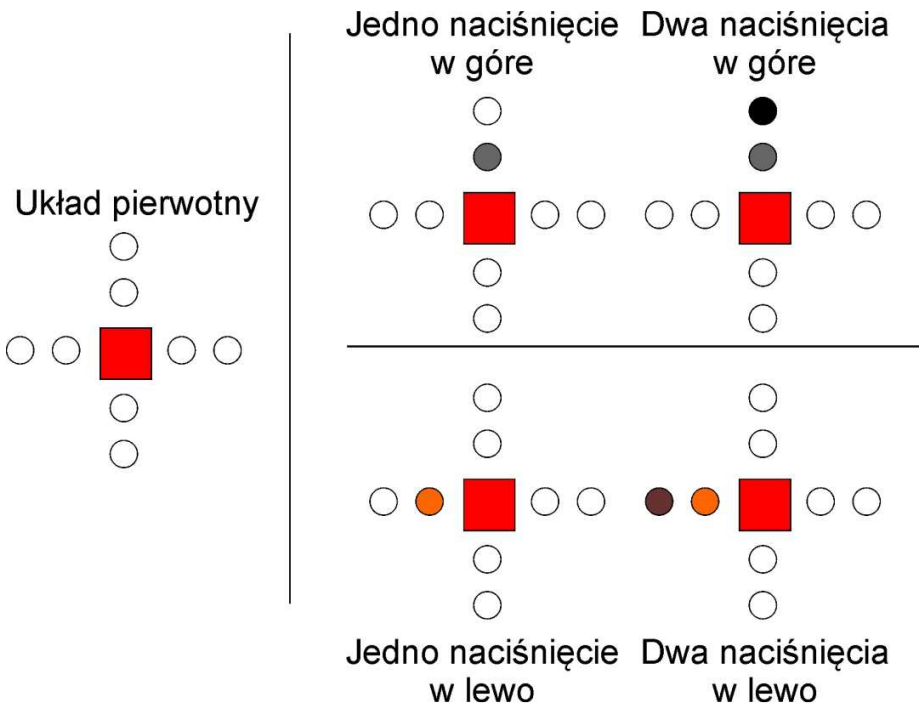
Zadanie 4

Napisać program, w którym w zależności od przyciśniętego guzika w dżojstiku wykonane będą pewne proste procedury (np. zapalenie diody, zmiana tła, zmiana koloru itd.)

.

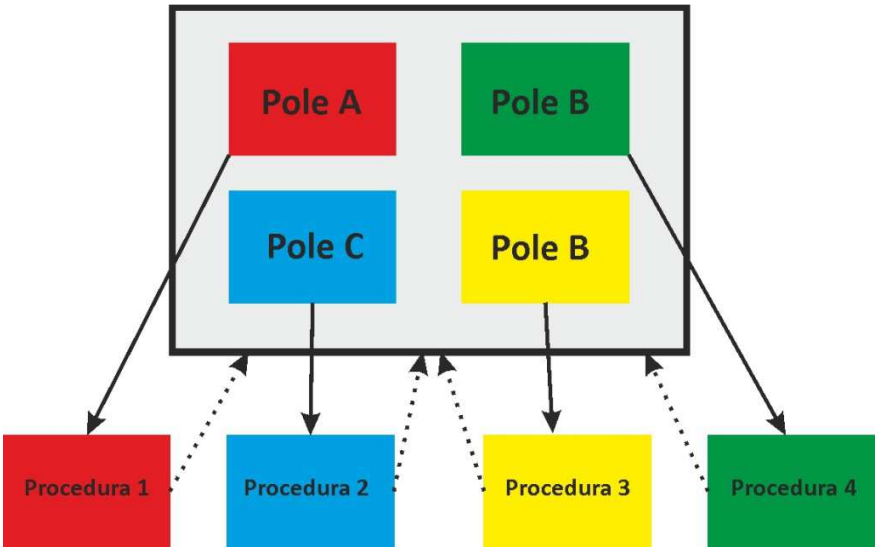
Zadanie 5

Napisać program, w którym w zależności od przyciśniętego guzika w dżojstiku zmieni wypełnienie odpowiedniego pola – przykładowa funkcjonalność znajduje się na rysunku poniżej.



Zadanie 6

Korzystając z programu napisanego w ramach zadania 1, należy napisać program „MENU”, w którym użytkownik za pomocą dżojstika porusza się po czterech widniejących na wyświetlaczy polach, a następnie przyciskając dżojstik wybiera wskazane właśnie pole i wykonuje przypisaną do tego pola prostą procedurę (np. zapalenie diody). Wykonanie procedury powinno spowodować zmianę obrazu pokazanego na wyświetlaczu. Jednocześnie ponowne naciśnięcie dżojstika powinno powodować powrót do menu głównego.


Pole A
Pole B
Pole C
Pole B
Procedura 1
Procedura 2
Procedura 3
Procedura 4