Politechnika Wrocławska
Katedra Cybernetyki i Robotyki

Sterowniki Robotów

Prymitywny Animowany Wyświetlacz
Opis projektu PAW

Paulina Porczyńska - 218484
Juliusz Stanisław Tarnowski - 215197

Tydzień parzysty, środa 11:15, grupa 7

Prowadzący: Wojciech Domski

17 czerwca 2017

Spis treści

1 Opis Projektu **2**
1.1 Temat 2
1.2 Cel Projektu 2

2 Specyfikacja **2**
2.1 Środowisko i język programowania 2
2.2 Opis funkcjonalności 2
2.3 Budowa architektury systemu 3
2.3.1 Biblioteki matematyczne 3
2.3.2 Klasa koloru 3
2.3.3 Klasy figur 3
2.3.4 Klasa Scena 4
2.4 Diagram Klas 4

3 Konfiguracja mikrokontrolera **4**
3.1 Konfiguracja pinów 4
3.2 Peryferia 6

4 Opis działania programu **8**

5 Zadania niezrealizowane **10**
5.1 Wykorzystanie z SDRAM 10

1 Opis Projektu

1.1 Temat

Animacje na LCD (płytka STM32F429I-DISC0) Biblioteka obsługująca rysowanie prymitywów (linia, prostokąt, trójkąt, koło, tekst); buforowanie obrazu. Wykonanie serii animacji na wyświetlaczu graficznym.

1.2 Cel Projektu

Głównym zadaniem jest napisanie biblioteki pozwalającej na generowanie obrazów na ekranie LCD płytki STM32F429i.

W ramach projektu napisaliśmy funkcje pozwalające rysować proste kształty (koła, linie itp. wypełnione lub nie). Korzystaliśmy z gotowych bibliotek pozwalających na podstawową obsługę i konfigurację wyświetlacza oraz z programów dedykowanych tym płytkom: STM32Cube oraz System Workbench for STM32.

Finałem naszego projektu jest seria animacji pokazujących możliwości biblioteki oraz działanie wyświetlacza.

2 Specyfikacja

2.1 Środowisko i język programowania

Płytką na której pracowaliśmy jest STM32F429 Discovery. Posiada wbudowany kolorowy wyświetlacz LCD o rozdzielczości 320x240 pixeli. Jest również wyposażona w 2 MB Flash, 8 MB SDRAM. Wyświetlacz jest dotykowy, jednak nie korzystamy z tej jego funkcjonalności.

Biblioteka powstała za pomocą dedykowanego oprogramowania w systemie operacyjnym Windows 7.

- STM32Cube
- System Workbench for STM32
- STM32 ST-Link Utility

Językiem, w którym programowaliśmy jest **C++**. Jest to język uniwersalny, przeznaczony do pisania programów na różne platformy, dzięki temu nasz kod powinno być można w miarę łatwo dostosować do sterowania podobnym ekranem LCD za pomocą innej płytki. Chcieliśmy wykorzystać obiektowość do stworzenia logicznego systemu klas obiektów i funkcji, tak, aby po zakończeniu projektu inne osoby mogły łatwo zrozumieć nasz kod i rozbudowywać go we własnym zakresie.

2.2 Opis funkcjonalności

Biblioteka przede wszystkim obsługuj wyświetlanie figur oraz wydarzeń z nimi związanych. Figury są płaskie, jednak zbudowaliśmy bibliotekę tak, aby mogła

kiedyś obsługiwać przestrzeń trójwymiarową. Z punktu widzenia wydajności dokłada to wiele obliczeń, ale czyni projekt ciekawszym, znacząco zwiększając jego potencjalne możliwości. Figury składają się z prymitywnych kształtów: punktów, linii i okręgów. Każdy prymityw posiada własny kolor. Figury są umieszczane na scenie, która pozwala na obsługę wydarzeń. Wydarzeniem jest przesuwanie oraz obracanie się figury, ale architektura klas pozwala również na łatwą implementację zmiany koloru, wykrycie przyciśnięcia figury na panelu dotykowym, lub nawet kolizje. Oczywiście można poruszać elementami na różnych poziomach: można obracać i przesuwać wszystko na raz z poziomu sceny, można manipulować jedną figurą lub pojedynczym prymitywem w figurze.

Ewentualni następcy będą mogli rozbudować bibliotekę o wiele możliwości - wczytywanie tekstur, obsługę panelu dotykowego, implementację prostej fizyki, wczytywanie animacji lub gifów i inne. Naszym celem jest zapewnienie architektury, która to umożliwi. Chcemy, aby obie części projektu były pod tym względem wygodne, dlatego wydarzenia zdefiniowane są dosyć abstrakcyjnie, a interfejs sprzętu umożliwia komunikację również z pominięciem sceny.

2.3 Budowa architektury systemu

2.3.1 Biblioteki matematyczne

Napisaliśmy biblioteki **PAW_Vector.h** i **PAW_Matrix.h** pozwalające operować na wektorach i macierzach 4d. Posiadają one przeciążenia wszystkich niezbędnych operatorów, łącznie z operatorem indexowania, co pozwala na intuicyjne odwołania do elementów jak *macierz['x']['z']* czy *wektor['u']*. Przeciążone zostało też mnożenie macierz * wektor i macierz * macierz. Dodatkowo mamy metodę tworzącą podstawową macierz obrotu. Obydwie klasy są skończone i zawiera wszystkie operacje matematyczne jakich prawdopodobnie możemy potrzebować.

2.3.2 Klasa koloru

Klasa **PAW_Color** przetrzymuje informacje o kolorach. W szczególności zwraca kod koloru na podstawie podanych wartości T, R, G, B oraz ma metodę mieszającą kolory na podstawie ich wagi, która będzie mogła być wykorzystywana w algorytmach antyaliasingowych. W przyszłości będzie można dodać obsługę przeźroczystości.

2.3.3 Klasy figur

Jest to zbiór klas dziedziczących po klasie wirtualnej **PAW_Primitive**, w chwili obecnej: **punkt**, **linia**, **koło** - podstawowe prymitywy, z których możemy składać inne figury. Klasa wirtualna **PAW_Figure** jest kontenerem na prymitywy tworzące razem większy kształt. Na takiej grupie będzie się wykonywać jednakowe przekształcenia aby zmieniać całą figurę. Na przykład: płatek śniegu jest złożony z trzech obiektów klasy linia. Wszystkie trzy są elementami obiek-

tu klasy figura, dzięki czemu wiemy, że tworzą razem jeden kształt i należy je przekształcać w ten sam sposób.

2.3.4 Klasa Scena

Jest to klasa przetrzymująca informacje o scenie: kolor tła, listę wszystkich figur znajdujących się na scenie, rozdzielczość obrazu. Jest odpowiedzialna za rysowanie obiektów na wyświetlaczu (metoda *display*) oraz ma metody pozwalające na identyczny obrót/translacje/zmianę koloru wszystkich figur na raz.

2.4 Diagram Klas

Rysunek 1 przedstawia diagram najlepiej ilustrujący wykonaną przez nas pracę.

3 Konfiguracja mikrokontrolera

3.1 Konfiguracja pinów

Rysunek 2 oraz poniższa tabela przedstawiają konfiguracje pinów na mikrokontrolerze. Rysunek 3 przedstawia ustawienia zegarów. Skonfigurowane zostały piny odpowiedzialne za obsługę wyświetlacza oraz pamięci SDRAM i DMA2D, które ostatecznie nie są wykorzystywane.

PERIPHERALS	MODES	FUNCTIONS	PINS
FMC:SDRAM	SDCKE1+SDNE1	FMC_SDCKE1	PB5
FMC:SDRAM	SDCKE1+SDNE1	FMC_SDNE1	PB6
FMC:SDRAM	4 banks	FMC_BA0	PG4
FMC:SDRAM	4 banks	FMC_BA1	PG5
FMC:SDRAM	12 bits	FMC_A0	PF0
FMC:SDRAM	12 bits	FMC_A1	PF1
FMC:SDRAM	12 bits	FMC_A2	PF2
FMC:SDRAM	12 bits	FMC_A3	PF3
FMC:SDRAM	12 bits	FMC_A4	PF4
FMC:SDRAM	12 bits	FMC_A5	PF5
FMC:SDRAM	12 bits	FMC_A6	PF12
FMC:SDRAM	12 bits	FMC_A7	PF13
FMC:SDRAM	12 bits	FMC_A8	PF14
FMC:SDRAM	12 bits	FMC_A9	PF15
FMC:SDRAM	12 bits	FMC_A10	PG0
FMC:SDRAM	12 bits	FMC_SDCLK	PG8
FMC:SDRAM	12 bits	FMC_SDNCAS	PG15
FMC:SDRAM	12 bits	FMC_SDNRAS	PF11
FMC:SDRAM	12 bits	FMC_SDNWE	PC0
FMC:SDRAM	12 bits	FMC_A11	PG1
FMC:SDRAM	16 bits	FMC_D0	PD14
FMC:SDRAM	16 bits	FMC_D1	PD15
FMC:SDRAM	16 bits	FMC_D2	PD0
FMC:SDRAM	16 bits	FMC_D3	PD1
FMC:SDRAM	16 bits	FMC_D4	PE7
FMC:SDRAM	16 bits	FMC_D5	PE8
FMC:SDRAM	16 bits	FMC_D6	PE9
FMC:SDRAM	16 bits	FMC_D7	PE10
FMC:SDRAM	16 bits	FMC_D8	PE11
FMC:SDRAM	16 bits	FMC_D9	PE12
FMC:SDRAM	16 bits	FMC_D10	PE13
FMC:SDRAM	16 bits	FMC_D11	PE14
FMC:SDRAM	16 bits	FMC_D12	PE15
FMC:SDRAM	16 bits	FMC_D13	PD8
FMC:SDRAM	16 bits	FMC_D14	PD9
FMC:SDRAM	16 bits	FMC_D15	PD10
FMC:SDRAM	16-bit byte enable	FMC_NBL0	PE0
FMC:SDRAM	16-bit byte enable	FMC_NBL1	PE1

PERIPHERALS	MODES	FUNCTIONS	PINS
LTDC	RGB666 (18 bits)	LTDC_R2	PC10
LTDC	RGB666 (18 bits)	LTDC_R3	PB0
LTDC	RGB666 (18 bits)	LTDC_R4	PA11
LTDC	RGB666 (18 bits)	LTDC_R5	PA12
LTDC	RGB666 (18 bits)	LTDC_R6	PB1
LTDC	RGB666 (18 bits)	LTDC_R7	PG6
LTDC	RGB666 (18 bits)	LTDC_G2	PA6
LTDC	RGB666 (18 bits)	LTDC_G3	PG10
LTDC	RGB666 (18 bits)	LTDC_G4	PB10
LTDC	RGB666 (18 bits)	LTDC_G5	PB11
LTDC	RGB666 (18 bits)	LTDC_G6	PC7
LTDC	RGB666 (18 bits)	LTDC_G7	PD3
LTDC	RGB666 (18 bits)	LTDC_B2	PD6
LTDC	RGB666 (18 bits)	LTDC_B3	PG11
LTDC	RGB666 (18 bits)	LTDC_B4	PG12
LTDC	RGB666 (18 bits)	LTDC_B5	PA3
LTDC	RGB666 (18 bits)	LTDC_B6	PB8
LTDC	RGB666 (18 bits)	LTDC_B7	PB9
LTDC	RGB666 (18 bits)	LTDC_HSYNC	PC6
LTDC	RGB666 (18 bits)	LTDC_VSYNC	PA4
LTDC	RGB666 (18 bits)	LTDC_CLK	PG7
LTDC	RGB666 (18 bits)	LTDC_DE	PF10
RCC BYPASS	Clock Source	RCC_OSC_IN	PH0
RCC BYPASS	Clock Source	RCC_OSC_OUT	PH1
SPI5	Full-Duplex Master	SPI5_MISO	PF8
SPI5	Full-Duplex Master	SPI5_MOSI	PF9
SPI5	Full-Duplex Master	SPI5_SCK	PF7

3.2 Peryferia

Poniższe tabele przedstawiają dokładniejszy opis konfiguracji ważniejszych peryferiów.

DMA2D	
mode:	Activated
Parameter Settings	
Transfer Mode	Memory to Memory
Color Mode	RGB565
Output Offset	0
Foreground layer Configuration	
DMA2D Input Color Mode	RGB565
DMA2D ALPHA MODE	No modification of the alpha channel value

FMC	
Clock and chip enable:	SDCKE1+SDNE1
Internal bank number:	4 banks
Address:	12 bits
Data:	16 bits
Byte enable:	set
SDRAM 1	
Bank SDRAM	bank 2
Number of column address bits	8 bits
Number of row address bits	12 bits
CAS latency	3 memory clock cycles
Write protection	Enabled
SDRAM common clock	2 HCLK clock cycles
SDRAM common burst read	Disabled
SDRAM common read pipe delay	2 HCLK clock cycles
SDRAM timing in memory clock cycles	
Load mode register to active delay	16
Exit self-refresh delay	16
Self-refresh time	16
SDRAM common row cycle delay	16
Write recovery time	16
SDRAM common row precharge delay	16
Row to column delay	16

LTDC	
Display Type:	RGB666 (18 bits)
Parameter Settings	
Synchronization for Width	
Horizontal Synchronization Width	8
Horizontal Back Porch	7
Active Width	320
Horizontal Front Porch	6
HSync Width	7
Accumulated Horizontal Back Porch Width	14
Accumulated Active Width	334
Total Width	340
Synchronization for Height	
Vertical Synchronization Height	4
Vertical Back Porch	2
Active Height	480
Vertical Front Porch	2
VSync Height	3
Accumulated Vertical Back Porch Height	5
Accumulated Active Height	485
Total Height	487
Signal Polarity	
Horizontal Synchronization Polarity	Active Low
Vertical Synchronization Polarity	Active Low
Data Enable Polarity	Active Low
Pixel Clock Polarity	Normal Input
BackGround Color	
Red	0
Green	0
Blue	0

4 Opis działania programu

Przedstawiamy tutaj sposób stworzenia prostej animacji oraz jej wyświetlanie na ekranie LCD. Rysunek 4 przedstawia jedną klatkę tej animacji.

- Połączenie prymitywów w figury, które będą wyświetlane, dobranie odpowiedniego miejsca wyświetlania. Tutaj funkcja tworząca pawie pióro.

```
void PAW_MakeFeatherFromFigure(PAW_Figure &this_figure)
{
//duze oko
  //polozenie srodka, promien, wypelnienie, kolor
  PAW_Circle big_cir1(PAW_Vector(120, 90, 0, 1),
    35, true, PAW_Color(255, 0, 255, 0));
```

```
  PAW_Circle big_cir2(PAW_Vector(120, 90, 0, 1),
    20, true, PAW_Color(255, 0, 0, 255));
//srednie oko
  PAW_Circle mid_cir1(PAW_Vector(120, 90+50, 0, 1),
    30, true, PAW_Color(255, 0, 255, 0));
  PAW_Circle mid_cir2(PAW_Vector(120, 90+50, 0, 1),
    15, true, PAW_Color(255, 0, 0, 255));
//male oka
  PAW_Circle s_cir1(PAW_Vector(120, 90+50+40, 0, 1),
    25, true, PAW_Color(255, 0, 255, 0));
  PAW_Circle s_cir2(PAW_Vector(120, 90+50+40, 0, 1),
    15, true, PAW_Color(255, 0, 0, 255));
  PAW_Circle s_cir3(PAW_Vector(120, 90+50+40+30, 0, 1),
    20, true, PAW_Color(255, 0, 255, 0));
  PAW_Circle s_cir4(PAW_Vector(120, 90+50+40+30, 0, 1),
    10, true, PAW_Color(255, 0, 0, 255));
  PAW_Circle s_cir5(PAW_Vector(120, 90+50+40+30+30, 0, 1),
    20, true, PAW_Color(255, 0, 255, 0));
  PAW_Circle s_cir6(PAW_Vector(120, 90+50+40+30+30, 0, 1),
    10, true, PAW_Color(255, 0, 0, 255));
//wpisanie ich do figury, kolejnosc jest istotna
  this_figure.push(big_cir1);
  this_figure.push(big_cir2);
  this_figure.push(mid_cir1);
  this_figure.push(mid_cir2);
  this_figure.push(s_cir1);
  this_figure.push(s_cir2);
  this_figure.push(s_cir3);
  this_figure.push(s_cir4);
  this_figure.push(s_cir5);
  this_figure.push(s_cir6);
}
```

- Stworzenie obiektu klasy scena, wybranie koloru tła

```
PAW_Scene superscena(PAW_Color(0xff,255,255,255));
```

- Dodanie figur do listy elementów na Scenie

```
PAW_Figure f1, f2, f3, t1;
PAW_MakeFeatherFromFigure(f1);
PAW_MakeFeatherFromFigure(f2);
PAW_MakeFeatherFromFigure(f3);
PAW_MakeBody(t1);
superscena.push(f1);
```

```
superscena.push(f2);
superscena.push(f3);
superscena.push(t1);
```

- W pętli: zmiana położenia poszczególnych figur i prymitywów oraz wyświetlenie całości.

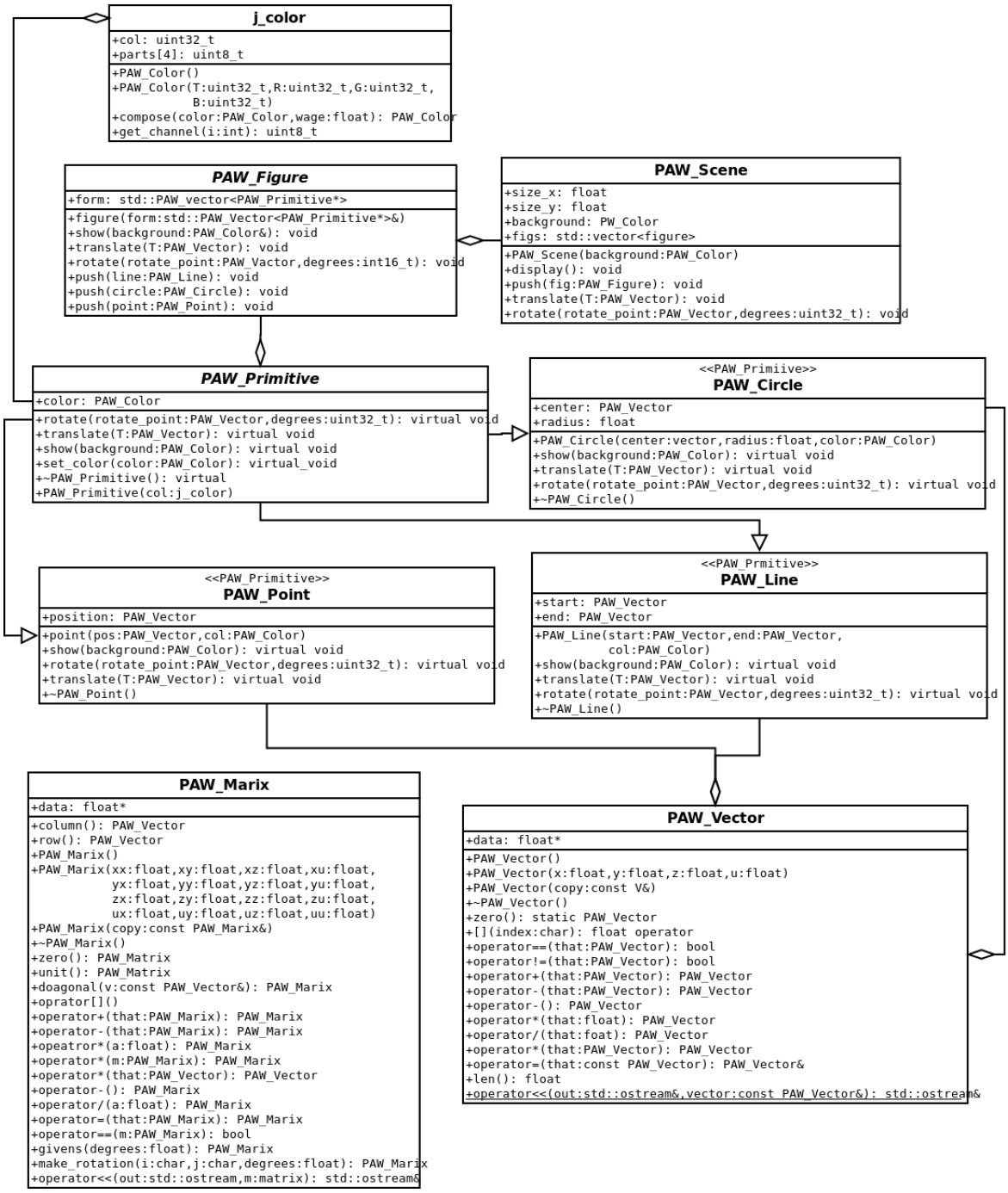
```
//obroc figury wzgledem jakiego punktu i o jaki kat
superscena.figs[0].rotate(PAW_Vector(120,290,0,0), a);
superscena.figs[1].rotate(PAW_Vector(120,290,0,0), b);

uint8_t i = 150;
while(i--) {
  superscena.display(); //wyswietl

  // nastrosz piorka
  superscena.figs[0].rotate(PAW_Vector(120,290,0,0), a);
  superscena.figs[1].rotate(PAW_Vector(120,290,0,0), b);
  a = -1 * a;
  b = -1 * b;
  HAL_Delay(1000); //delay do ladnego wyswietlania
}
```

5 Zadania niezrealizowane

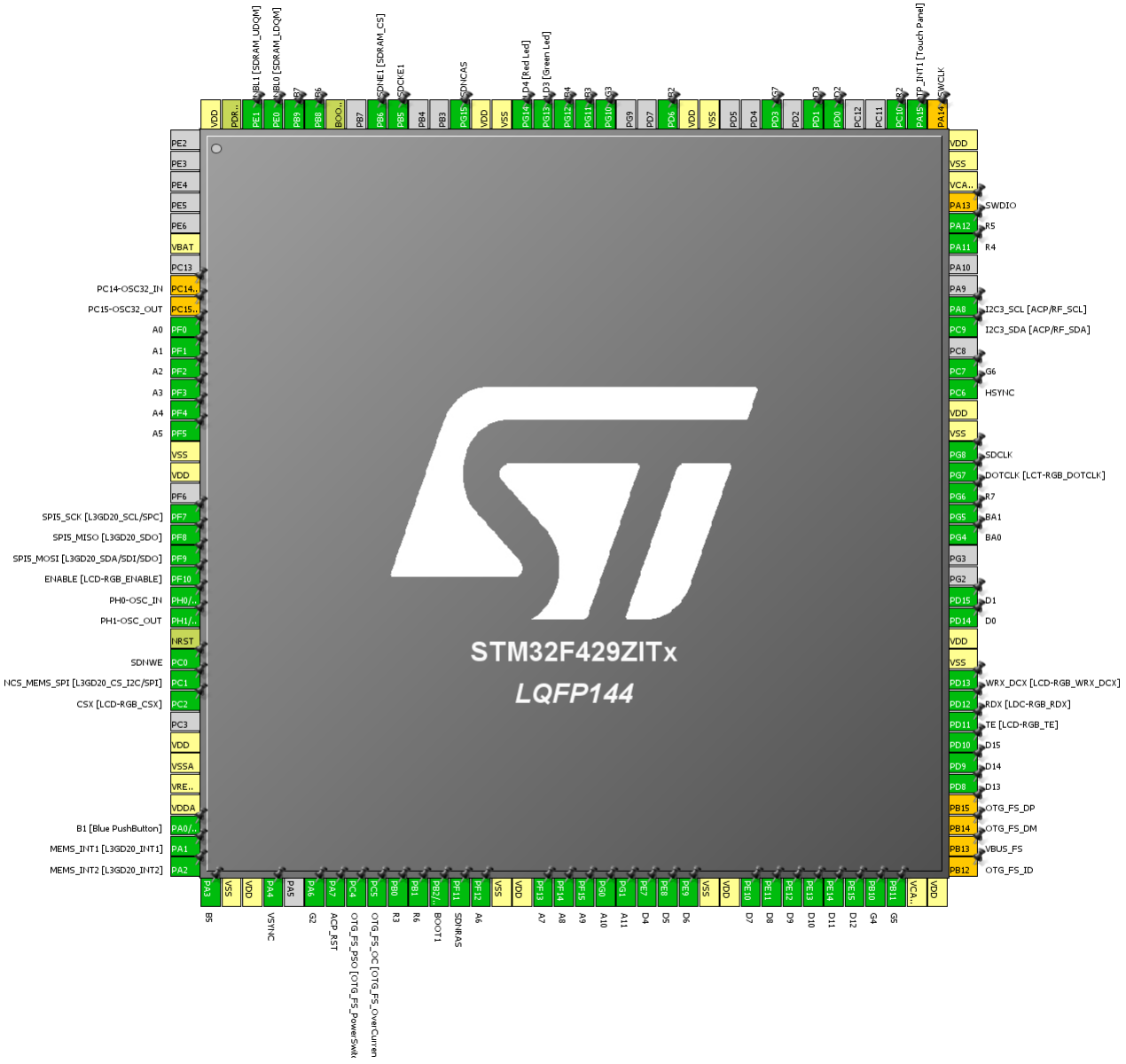
5.1 Wykorzystanie z SDRAM

Chcieliśmy zapewnić płynną animację, bez konieczności oglądania rysowanie każdej kolejnej figury. Do tego niezbędne jest korzystanie z RAMu oraz DMA2D. W tym momencie wszystko wyświetlane jest na bieżąco, co powoduje migotanie ekranu, brak wyraźnego obrazu, szybkie męczenie się oczu i generalnie wygląda źle. Obsługa RAMu zagwarantowała by nam możliwość wrzucania całych kolejnych klatek animacji na raz. Wtedy, nawet jeśli procesor pozwoli nam na 1FPS, to efekt końcowy był by nieporównywalnie lepszy i przyjemniejszy dla oka.

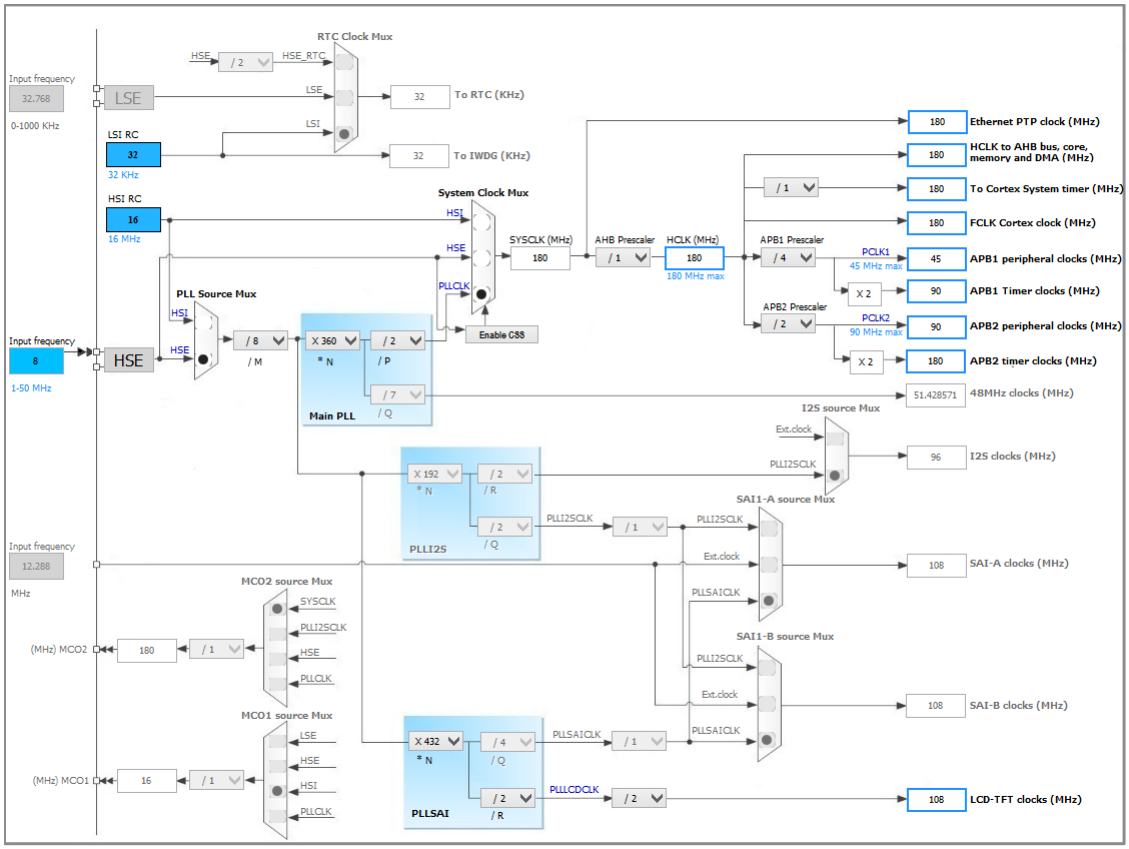
Ostatecznie udało nam się uruchomić ram oraz poprawnie zapisywać i odczytywać z niego dane, jednak nie zrobiliśmy interfaceu niezbędnego do zintegrowania go z naszą biblioteką.



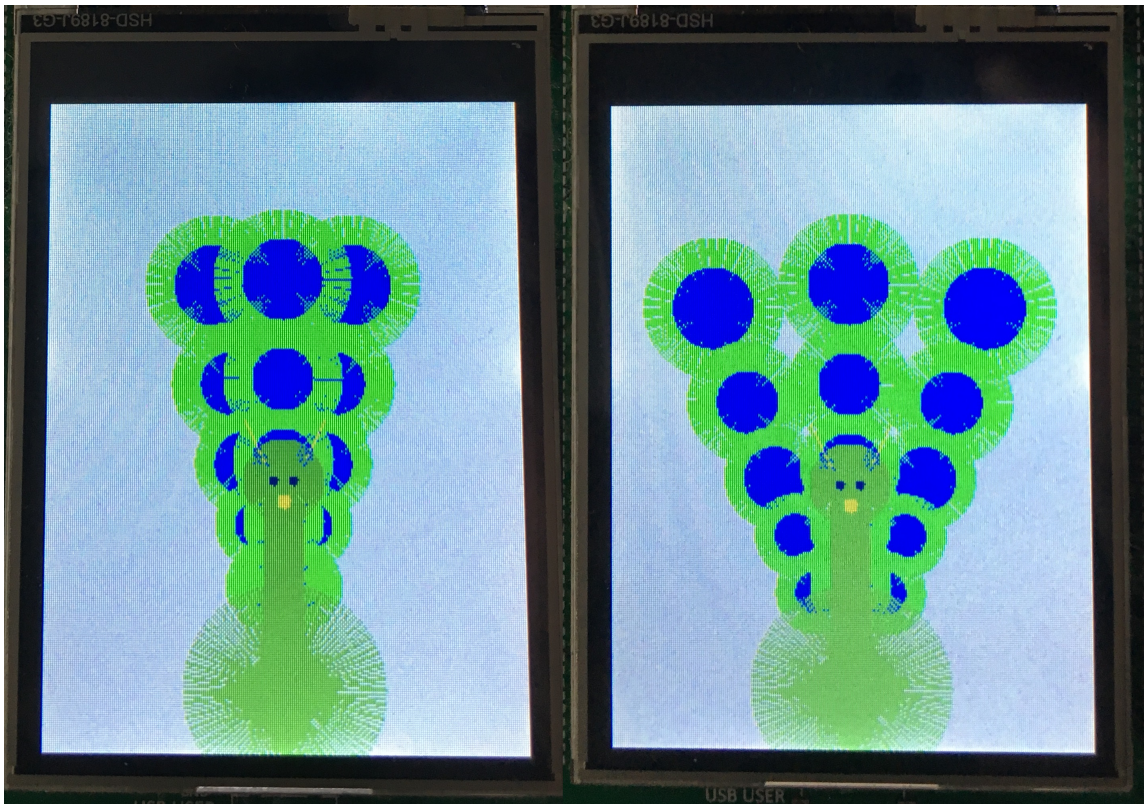
Rysunek 1: Architektura systemu



Rysunek 2: Konfiguracja wyprowadzeń



Rysunek 3: Konfiguracja zegarów



Rysunek 4: Prosty Paw złożony z prymitywów