POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko studenta: Michał Zalewski
Nr albumu: 171671
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Profil: Architektura systemów komputerowych

Imię i nazwisko studenta: Adrian Misiak
Nr albumu: 171600
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Profil: Architektura systemów komputerowych

Imię i nazwisko studenta: Łukasz Mrugała
Nr albumu: 171914
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Profil: Architektura systemów komputerowych

PROJEKT DYPLOMOWY INŻYNIERSKI

Tytuł projektu w języku polskim: Emulator procesora Sharp LR35902

Tytuł projektu w języku angielskim: Emulator of Sharp LR35902 CPU

Opiekun pracy: dr inż. Krzysztof Bikonis
Data ostatecznego zatwierdzenia raportu podobieństw w JSA: 15.12.2020

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Emulator procesora Sharp LR35902

Imię i nazwisko studenta: Michał Zalewski
Data i miejsce urodzenia: 07.04.1998, Gdynia
Nr albumu: 171671

Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: informatyka

Poziom kształcenia: pierwszy
Forma studiów: stacjonarne

Typ pracy: projekt dyplomowy inżynierski

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2019 r. poz. 1231, z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t.j. Dz. U. z 2020 r. poz. 85, z późn. zm.),[1] a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

15.12.2020, Michał Zalewski

Data i podpis lub uwierzytelnienie w portalu uczelnianym Moja PG

**) Dokument został sporządzony w systemie teleinformatycznym, na podstawie §15 ust. 3b Rozporządzenia MNiSW z dnia 12 maja 2020 r. zmieniającego rozporządzenie w sprawie studiów (Dz.U. z 2020 r. poz. 853). Nie wymaga podpisu ani stempla.*

[1] Ustawa z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce:

Art. 312. ust. 3. W przypadku podejrzenia popełnienia przez studenta czynu, o którym mowa w art. 287 ust. 2 pkt 1–5, rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 312. ust. 4. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 5, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o podejrzeniu popełnienia przestępstwa.

STRESZCZENIE

W 1998 roku wydana została konsola GameBoy Color, która osiągnęła znaczny sukces sprzedażowy i jest popularna do dzisiaj. Stosunkowo trudno jednak uzyskać jej fizyczne jednostki, dlatego tworzone są jej emulatory programowe, pozwalające na uruchamianie starych programów na nowoczesnych architekturach. Zauważając, że wiele z nich wymaga stosunkowo dużej mocy obliczeniowej, w ramach tej pracy stworzono emulator dla mniej wydajnych platform komputerowych działających pod kontrolą systemów zgodnych ze standardem POSIX. Rezultat przedstawiono na przykładzie komputera jednoukładowego Raspberry Pi. Praca zawiera opis procesu wytwórczego prowadzącego do stworzenia oprogramowania wraz z instrukcją wdrożeniową na przykładową platformę.

Słowa kluczowe: GameBoy Color, emulacja programowa, Zilog Z80, UNIX, Raspberry Pi.

Dziedzina nauki i techniki, zgodnie z wymogami OECD: 2.2.f Sprzęt komputerowy i architektura komputerów.

ABSTRACT

In 1998 GameBoy Color console was released to the public, with great sales success and is popular to this day. Over time, it has become hard to acquire its physical units, however. That is why its emulators have been developed, allowing old programmes made for it to still be launched on modern architectures. Having noticed that many of these require fairly high computing power, in this project an emulator has been created for lower-end hardware controlled by POSIX compliant systems. Results are presented on Raspberry Pi single board computer. This paper describes the software development process that led to creation of that software, complete with deployment instruction for the sample platform.

Keywords: GameBoy Color, software emulation, Zilog Z80, Unix, Raspberry Pi.

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

CGB/GBC – Game Boy Color
DMA – Direct Memory Access, mechanizm pozwalający na kopiowanie pamięci z jednej lokalizacji do innej bez czynnego udziału procesora
DMG/GB – Game Boy
H-blank – Horizontal Blanking, wygaszanie poziome wyświetlania
HRAM – High RAM, fragment pamięci operacyjnej wewnątrz konsoli
HUD – Head-Up Display, w grach video część interfejsu zawierająca informację o obiektach kontrolowanych przez gracza
MBC – Memory Bank Controller, kontroler banków pamięci
OAM – Object Attribute Memory, pamięć atrybutów obiektów graficznych
PC – Program Counter, licznik programowy; rejestr procesora wskazujący na instrukcję do wykonania
PCB – Printed Circuit Board
SP – Stack Pointer, rejestr procesora przechowujący adres wierzchołka stosu wywołań
V-blank – Vertical Blanking, wygaszanie pionowe wyświetlania
VRAM – Video RAM, pamięć operacyjna układu graficznego
WRAM – Working RAM, główna pamięć operacyjna wewnątrz konsoli

1. WSTĘP I CEL PRACY - MICHAŁ ZALEWSKI

Każda forma rozwoju jest napędzana innowacyjnymi pomysłami i wynalazkami. Często nie zdobywają popularności i zostają zapomniane, ale czasem stają się bardzo rozpowszechnione, dzięki czemu wpisują się na karty historii rozwoju technologicznego i na stałe znajdują miejsce w kulturze. Rozwój ten jednak nie zatrzymuje się, wraz z powstawaniem kolejnych wynalazków wypierane są te istniejące i szybko tracą na popularności. Jednakże, po upływie pewnego czasu, zaobserwować można zjawisko powtórnego zainteresowania przestarzałymi już rozwiązaniami. Może być to związane z nostalgią albo fascynacją ideami, które doprowadziły do zaawansowanych rozwiązań technologicznych, które znane są z codziennego życia.

Niestety w momencie, w którym dana technologia przestaje być aktualna, producent zwykle zaprzestaje jej wytwarzania. Z biegiem czasu sprzęt ulega zniszczeniu i zmniejsza się jego podaż na rynku wtórnym, co w połączeniu z rosnącym zainteresowaniem starszą technologią prowadzi do podwyższenia cen i bardzo ogranicza dostępność dla osób zainteresowanych. Na szczęście postęp technologiczny pozwala na częściowe rozwiązanie tego problemu poprzez użycie programowej emulacji sprzętu.

Dawne konsole do gier są jednym z najbardziej dobitnych przykładów powracającego po latach zainteresowania użytkowników. Spośród nich wyróżnia się wydana w 1998 roku konsola przenośna Game Boy Color, będąca dotychczas jednym z najlepiej sprzedających się systemów do gier. Licząc sumarycznie wyniki sprzedaży GBC i jej bezpośredniego poprzednika, konsolę Game Boy, z którym zachowana została pełna kompatybilność, sprzedane zostało ponad 110 milionów egzemplarzy [1]. Jest to drugi wynik spośród wszystkich konsoli przenośnych w historii.

W obliczu tak dużej popularności nie dziwi fakt, że dotychczas powstało wiele rozwiązań pozwalających na granie w gry z biblioteki konsoli w innych środowiskach. Warto wspomnieć o adapterach Super Game Boy przeznaczonych dla Super Nintendo Entertainment System, które pozwalały na uruchamianie kartridżów przeznaczonych dla GBC i były prawdopodobnie pierwszą taką próbą. Było to jednak rozwiązanie czysto sprzętowe, działanie programów było oparte o układy scalone identyczne z tymi z oryginalnej konsoli, więc nie odbywała się emulacja. W późniejszym czasie powstało wiele nieoficjalnych implementacji programowych wewnętrznej elektronicznej logiki konsoli, których wytworzenie wymagało dużego wysiłku społeczności pasjonatów w zakresie inżynierii odwrotnej działania oryginału. Wybrane emulatory zostały opisane dalej, w podrozdziale *Analiza istniejących rozwiązań*.

W ramach tego projektu inżynierskiego podjęto próbę stworzenia emulatora procesora Sharp LR35902, stanowiącego centralną jednostkę obliczeniową w konsoli Game Boy Color, jak również pozostałych układów sprzętowych konsoli i kartridżów z grami. Za cel postawiono wytworzenie działającego emulatora, na którym możliwe jest uruchamianie gier napisanych na konsolę. Dodatkowym celem było wdrożenie go na urządzeniu opartym o małą platformę komputerową, która pozwala na łatwe użycie emulatora bez konieczności konfiguracji sprzętu. Ważne było również umożliwienie korzystania z wybranego przez użytkownika kontrolera i wyświetlacza lub elementów wbudowanych w urządzenie.

Przedstawiona praca stanowi podsumowanie wysiłku włożonego w realizację wymienionych celów i opis rozwiązań przyjętych, aby była ona możliwa. Dla ułatwienia zrozumienia przyczyn poszczególnych decyzji projektowych, jak również dla uzyskania głębszego wglądu w metody użyte

przy tworzeniu poszczególnych części projektu, poniższy tekst został podzielony na rozdziały uszeregowane według poziomu abstrakcji prowadzonych w nim rozważań – od przeprowadzenia ogólnej analizy do omówienia aspektów implementacji.

W rozdziale *Analiza problemu* Łukasz Mrugała przedstawił dekompozycję problemu emulacji na części funkcjonalne, określił interesariuszy oraz użytkowników systemu, i wyznaczył konkretne cele oraz wymagania pomocne w dalszych rozważaniach. Adrian Misiak w podrozdziale *Analiza istniejących rozwiązań* przedstawił wybrane alternatywne rozwiązania przedstawionego problemu, a w podrozdziale *Różnice między rozwiązaniem wytworzonym a istniejącymi* Michał Zalewski przedstawił cechy wyróżniające system stworzony w ramach projektu
od innych dostępnych.

Rozdział *Architektura systemu* napisany przez Michała Zalewskiego zawiera omówienie wysokopoziomowej architektury emulatora i jego otoczenia programowego i sprzętowego oraz opis interfejsów udostępnianych przez poszczególne komponenty.

Kolejny rozdział pt. *Implementacja* porusza kwestie dotyczące faktycznej realizacji systemu, w tym używane technologie, zaimplementowane moduły, sposoby zarządzania kodem i informacje o przygotowaniu części sprzętowej projektu opisane przez Adriana Misiaka. Dodatkowo w podrozdziale *Opis współpracy między modułami* opisane zostały złożone interakcje między modułami.

W rozdziale *Testy i wyniki* Łukasz Mrugała przedstawił metodologię testowania wytworzonego emulatora z użyciem przykładów oprogramowania napisanego na oryginalną konsolę i podsumował jego wyniki.

W ramach rozdziału *Instrukcja dla użytkownika* Michał Zalewski przedstawił sposób użytkowania systemu oraz omówił jego wdrożenie na docelową platformę.

Rozdział *Podsumowanie* służy zebraniu doświadczeń z procesu tworzenia
systemu i opisaniu:

- problemów, które wystąpiły w trakcie – Łukasz Mrugała,
- osiągniętych sukcesów i pozostawionych niedociągnięć – Adrian Misiak,
- planów i możliwości związanych z dalszym rozwojem emulatora i systemu – Adrian Misiak,
- wniosków wyciągniętych z całości projektu – Michał Zalewski.

2. ANALIZA PROBLEMU

2.1. Analiza wstępna - Łukasz Mrugała

2.1.1. Interesariusze

Tworzenie emulatora istniejącego urządzenia jest specyficznym przypadkiem pod względem analizy oprogramowania. W tym etapie jedynym interesariuszem stawiającym cele biznesowe będzie specyfikacja emulowanej maszyny. Oczywiście przy specyfikowaniu celów funkcjonalnych dołączy do nich zespół deweloperski wraz z opiekunem projektu.

Specyfikacja emulowanej maszyny nie zawsze jest fizycznym dokumentem, ani nawet wcześniej znanym zbiorem informacji. Zamiast tego jest to teoretyczny zbiór wszystkich funkcji konsoli CGB doświadczanych przez użytkownika. W naszym wypadku reprezentuje ją kilka dokumentów sporządzonych przez entuzjastów CGB, które to, w połączeniu z wynikami testów zespołu deweloperskiego, będą w stanie zbliżyć się do ideału w wystarczającym stopniu. Wspomnianymi dokumentami są:

- The Cycle-Accurate Game Boy Docs [10]
- Pan Docs [11]
- Game Boy(TM) CPU Manual [12]
- GameBoy Opcode Summary [13]
- Gameboy CPU (LR35902) instruction set [14]

2.1.2. Cele systemu

Naszym nadrzędnym celem biznesowym jest umożliwienie łatwego uruchamiania programów napisanych dla CGB, dodatkowo zapewniając wybór sposobu wprowadzania wejścia i używanego wyświetlacza. Oprócz tego chcemy wzbudzić zainteresowanie grami retro przez zaoferowanie użytkownikom wygodniejszego doświadczenia w porównaniu do korzystania z już niewydawanego oryginału. Poza tym, chcemy też zwiększyć zainteresowanie od strony deweloperskiej dzięki zapewnianiu im wsparcia.

Chcąc spełnić te cele biznesowe, wyznaczyliśmy odpowiednie cele funkcjonalne. Łatwe uruchamianie gier CGB zamierzamy uzyskać przez emulację działania konsoli CGB na nowoczesnym sprzęcie, zwracając szczególną uwagę na możliwość łatwego zmieniania aktualnie włączonej gry bez potrzeby zmieniania za każdym razem urządzenia pamięci trwałej.

Zwiększenie zainteresowania chcemy osiągnąć przez ułatwienie korzystania z emulatora. Uproszczenie procesu wyboru i włączenia gry zostało już wspomniane, ale do tego chcemy osiągnąć emulację na sprzęcie, który może być przenośny, analogicznie do oryginalnej konsoli. Chcemy też wspierać wiele różnych urządzeń wejścia, a przynajmniej te najpopularniejsze.

Ostatni cel biznesowy zamierzamy osiągnąć przez wbudowanie w program emulujący dodatkowych opcji pozwalających na uzyskanie informacji o występujących błędach i aktualnie wykonywanych instrukcjach.

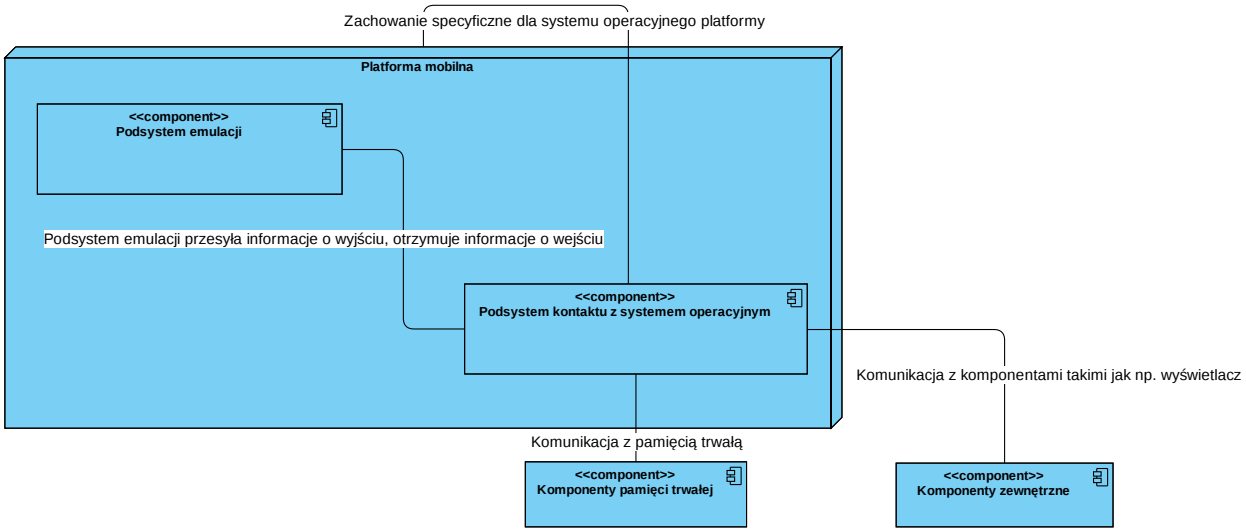
2.1.3. *Użytkownicy*

Wyróżniamy dwa podstawowe typy użytkowników korzystających z naszego systemu. Pierwszym jest użytkownik oprogramowania, który wymaga od niego dwóch rzeczy: działającej emulacji, której widoczny dla niego stan, na który może wpływać w sposób analogiczny do CGB, nie odbiega w znaczący sposób od oryginalnego oprogramowania oraz możliwości zachowania stanu programu w celu późniejszego wznowienia gry. Jego potrzeby mają wyższy priorytet.

Drugim typem użytkownika jest deweloper oprogramowania, chcący wykorzystać nasz system jako część środowiska deweloperskiego do uruchamiania swoich programów na platformę CGB i sprawdzania ich poprawności. W dużej mierze jego potrzeby pokrywają się z potrzebami użytkownika oprogramowania, ponieważ on również wymaga wiernej emulacji, ale dołącza do tego chęć czytelnego i poprawnego systemu rejestracji zdarzeń emulatora. Deweloper potrzebuje dostępu do informacji o błędach i wykonywanych instrukcjach.

2.2. *Wydzielenie i analiza podsystemów - Łukasz Mrugała*

Patrząc na nasze cele można dość szybko dojść do wniosku, że nasz system będzie się dzielił na dwa podstawowe podsystemy - podsystem emulacji i podsystem kontaktu z systemem operacyjnym. Diagram przedstawiający te podsystemy widoczny jest na Rys. 2.1.



Rys. 2.1. Szkielet przyszłego diagramu wdrożeniowego systemu stworzony z perspektywy analizy domeny problemowej.

2.2.1. *Podsystem emulacji*

Ten podsystem jest odpowiedzialny za emulację działania wewnętrznych części konsoli CGB. Składają się na niego opisane później komponenty emulacji: pamięci, procesora, przerwań, grafiki, zegara i dźwięku. Ma on działać w sposób niezależny od platformy komputerowej.

2.2.2. Podsystem komunikacji z systemem operacyjnym

Podsystem komunikacji odpowiada za interpretację komend na danych podsystemu emulacji i tłumaczenie ich na akcje specyficzne m. in. dla systemu operacyjnego urządzenia. Ze względu na wyniki analizy problemowej w rozdziale 2 możemy założyć, że zostanie użyty system UNIXowy. Oprócz tego podsystem ten ma komunikować się z urządzeniami zewnętrznymi i pamięcią trwałą trzymającą programy do uruchomienia. Ponadto winien on przekazywać podsystemowi emulacji potrzebne mu dane, wśród nich dane gier CGB. Składają się na niego komponenty: wejścia, wyświetlania grafiki, tworzenia dźwięku, obsługi pamięci zewnętrznej.

2.3. Wydzielenie i analiza komponentów podsystemu emulacji - Łukasz Mrugała

2.3.1. Platforma komputerowa

Wybór specyficznej platformy jest kwestią implementacyjną, ale jesteśmy w stanie nałożyć pewne wymagania na sprzęt, którego szukamy. Chcemy mieć do dyspozycji procesor przynajmniej o rząd wielkości szybszy niż oryginalna konsola [2], pamiętając o tym, że CGB ma do dyspozycji tryb podwójnej prędkości.

Wybierając platformę komputerową potrzebujemy możliwości zainstalowania na niej systemu UNIXowego. Chcemy, by była ona lekka i niewielkich rozmiarów. Patrząc na istniejące konsole, winna ona ważyć poniżej pięciuset gram [15] razem z komponentami zewnętrznymi.

2.3.2. Komponent emulacji pamięci

Komponent ten powinien udostępniać pozostałym komponentom programowym interfejs dostępu do pamięci w sposób analogiczny do oryginalnej konsoli. Musi być on w stanie poprawnie interpretować żądane rejestry i adresy w pamięci CGB i przekładać je na miejsca
w pamięci programowej.

2.3.3. Komponent emulacji zegara

Odpowiedzialny za kontrolowanie stanu globalnego systemu zegarowego taktowanego zegarem procesora i wywoływanie związanych z nim przerwań.

2.3.4. Komponent emulacji procesora

Komponent winien interpretować kod maszynowy pliku programu CGB i przekładać go na akcje na rejestrach i pamięci dające te same efekty. Na etapie analizy nie podejmujemy jeszcze ostatecznej decyzji co do dokładności synchronizacji czasowej naszego systemu. Rozpoznajemy teraz jedynie dwa główne sposoby dokonania tego: dokładność co do cyklu bądź też atomiczne instrukcje. Decyzja co do wyboru sposobu została podjęta na etapie projektowania.

W przypadku dokładności co do cyklu podsystem emulacji będzie odpowiedzialny za informowanie swoich komponentów o upływie czasu co cykl. Komponent emulacji procesora odpowiada wtedy jedynie za działanie jako CPU. Jednakże w wypadku atomiczności instrukcji powinien on też w pewien sposób przekazywać reszcie komponentów informację o tym, ile czasu zajęła dana instrukcja, by umożliwić im synchronizację.

2.3.5. Komponent emulacji przerwań

Komponent ten jest odpowiedzialny za rozpoznanie i obsługę żądań przerwań systemowych CGB. Powinien on udostępniać interfejs zgłaszania ich w sposób przyjazny
dla programisty innym komponentom.

2.3.6. Komponent emulacji grafiki

Odpowiada on za logikę rysowania ekranu na podstawie aktualnej zawartości pamięci. Winien wystawiać interfejs informujący inne komponenty o aktualnym stanie procesora graficznego. Jest to najbardziej skomplikowany komponent w CGB. Tutaj, analogicznie do komponentu emulacji procesora, występuje problem przyjęcia odpowiedniej dokładności.

W oryginalnej konsoli używana jest kolejka FIFO, która przyjmuje poziome odcinki po osiem pikseli każdy. GPU decyduje, co winno się tam znaleźć i umieszcza odpowiednie sekcje obrazu w kolejce[11]. Zamiast tego można przetwarzać bufor ekranu co linia, co zmniejszy dokładność emulacji, ale znacznie uprości implementację i przyśpieszy działanie kodu. Decyzja co do wyboru metody została podjęta na etapie projektowania.

2.3.7. Komponent emulacji dźwięku

Komponent ten jest odpowiedzialny za pobieranie z pamięci informacji o żądanych przez program CGB dźwiękach, interpretacja ich oraz translacja na format odpowiedni dla podsystemu komunikacji z systemem operacyjnym.

2.3.8. Komponent emulacji wejścia

Odpowiada on za wystawienie podsystemowi komunikacji z systemem operacyjnym struktury danych do wypełnienia i następnie na jej podstawie aktualizacja stanu pamięci odpowiedzialnej za wejście w CGB.

2.3.9. Komponent rejestracji zdarzeń

Potrzebujemy również komponentu odpowiedzialnego za udostępnienie deweloperowi informacji o aktualnym stanie emulatora. Winien on spełniać to zadanie za pomocą udostępnienia innym komponentom interfejsu do zgłaszania błędów oraz informacji o swoim stanie.

2.4. Wydzielenie i analiza komponentów podsystemu kontaktu z systemem operacyjnym - Łukasz Mrugała

2.4.1. Komponent wejścia

Komponent odpowiedzialny za pozyskiwanie od systemu operacyjnego informacji o wejściu użytkownika i tłumaczeniu go na format agnostyczny, by przekazać go komponentowi emulacji wejścia. Agnostyczność polega na tym, by komponent emulacji wejścia nie musiał wiedzieć ani przejmować się skąd ani w jaki sposób zostały pozyskane dane. W fazie analizy przyjęto, że będzie to wykonane za pomocą niskopoziomowej biblioteki multimedialnej.

2.4.2. Komponent wyświetlania grafiki

Komponent ten winien ze stabilną częstotliwością komunikować systemowi operacyjnemu, by wyświetlił on zawartość bufora wypełnianego przez komponent emulacji grafiki. W fazie analizy przyjęto, że będzie to wykonane za pomocą niskopoziomowej biblioteki multimedialnej.

2.4.3. Komponent tworzenia dźwięku

Odpowiada za pozyskanie od komponentu emulacji dźwięku informacji o żądanych dźwiękach i przekazaniu systemowi operacyjnemu komendy wytworzenia odpowiedniego dźwięku. W fazie analizy przyjęto, że będzie to wykonane za pomocą niskopoziomowej biblioteki multimedialnej.

2.4.4. Komponent obsługi pamięci zewnętrznej

Ma za zadanie korzystać z interfejsów systemu operacyjnego w celu pobierania z systemu plików danych o dostępnych programach CGB, wybraniu jednego z nich przez użytkownika i przekazania go podsystemowi emulacji. W fazie analizy przyjęto, że będzie to wykonane przez pomocniczy program zewnętrzny.

2.5. Wymagania - Łukasz Mrugała

2.5.1. Wymagania funkcjonalne

Większość naszych wymagań funkcjonalnych składa się na cel funkcjonalny emulacji konsoli CGB, zwykle z komponentem z odpowiadającą nazwą:

- Emulacja działania ekranu
- Emulacja działania przycisków
- Emulacja działania przerwań systemowych
- Emulacja działania działania pamięci
- Emulacja działania CPU
- Emulacja działania GPU
- Emulacja działania DMA - nie posiada własnego komponentu, może być obsłużone przez komponent emulacji pamięci i/lub komponent emulacji grafiki. Ma za zadanie obsługiwać transfery pamięciowe, które mogą odbyć się jedynie przy pewnych stanach GPU.
- Emulacja działania zegara
- Emulacja działania dźwięku

Pozostałymi wymaganiami funkcjonalnymi są wspomniany wcześniej centralny system rejestru zdarzeń z możliwością określenia granularności rejestracji, za który odpowiada komponent rejestracji zdarzeń, interfejs użytkownika konsoli umożliwiający wybór ROMu do załadowania, za który odpowiada komponent obsługi pamięci zewnętrznej, i funkcja wczytywania i usuwania dodatkowych ROMów do systemu, za którą również odpowiada ten komponent.

2.5.2. Wymagania na dane

Podstawowym wymaganiem nałożonym na nasz system jeśli chodzi o dane jest potrzeba przyjmowania plików ROM kompatybilnych z oryginalną konsolą. Pozwoli to na korzystanie z już istniejących zasobów oraz na zapewnienie podobieństwa do innych emulatorów. Oprócz tego, chcielibyśmy, by nasze pliki zapisu pamięci programu również były zgodne
z już istniejącymi emulatorami.

2.5.3. Wymagania jakościowe - Wydajność

Niestety, wiele dokładniejszych wartości pod względem wydajności jest trudnych do ustalenia w sensie numerycznym na etapie analizy. Kwestie takie, jak na przykład dopuszczalny jitter wyświetlanych klatek (w naszym systemie definiowany jako różnica pomiędzy zamierzonym i planowanym interwałem między wyświetlanymi klatkami, a prawdziwym i zmierzonym interwałem między nimi) mogą być określone dopiero po testach na działającym systemie.

Pomimo to jesteśmy w stanie stwierdzić, że nasz emulator musi być w stanie obsłużyć do 2 097 152 instrukcji na sekundę, ze względu na to, że maksymalna częstotliwość zegara CGB to 8 388 608 Hz, a najkrótsza instrukcja zajmuje cztery cykle zegara [11].

Oprócz tego system musi być w stanie rysować przynajmniej 1 382 400 pikseli na sekundę ze względu na oryginalny rozmiar ekranu równy 144 na 160 pikseli i możliwej częstotliwości wyświetlania równej 60 Hz [11].

2.5.4. Wymagania jakościowe - Elastyczność

Nasz system powinien móc być uruchamiany na systemach UNIXowych, a dokładniej na GNU/Linux, ze względu na wnioski wyciągnięte w rozdziale 2.

2.5.5. Wymagania jakościowe - Użyteczność

Kwestia użyteczności dla nas sprowadza się do kwestii łatwości użytku. Podzieliliśmy to wymaganie na trzy części: łatwość wyboru ROMu, wyboru zapisu i wykonania zapisu. Wszystkie zależą przede wszystkim od podsystemu kontaktu z systemem operacyjnym.

2.6. Analiza istniejących rozwiązań - Adrian Misiak

Przeglądając istniejące już emulatory można zauważyć, że emulatorów napisanych konkretnie pod system CGB jest stosunkowo niewiele. Większość z nich obsługuje system CGB jako jeden z wielu dostępnych. Szukając dalej można zobaczyć, że liczba implementacji dostępnych natywnie na systemy Linux jest jeszcze mniejsza. Z tej już całkiem małej puli należy zwrócić uwagę na emulatory o wysokiej precyzji. Owszem, wysoka precyzja jest jak najbardziej pożądana, ale wiążą się z nią wysokie wymagania sprzętowe do płynnej emulacji.

Do emulacji programów na układzie jednopłytkowym chcielibyśmy posiadać emulator natywny dla Linuxa z niewysokimi wymaganiami sprzętowymi, jednocześnie oferujący akceptowalną dokładność emulacji. Chcielibyśmy również mieć do dyspozycji program, któremu można przekazać argumenty przez wiersz poleceń, co by umożliwiło dostosowanie go do umieszczenia w osobno rozwijanym front-endzie. Biorąc pod uwagę takie wymagania, dokładniejszej analizie zostało poddane kilka emulatorów najbardziej polecanych przez Emulation General Wiki [3].

2.6.1. Przegląd emulatora mGBA

Jest to tak naprawdę jedyny emulator godny uwagi z sekcji emulatorów przeznaczonych na urządzenia mobilne o architekturze ARM. Charakteryzuje się dokładną emulacją oraz bardzo dobra wydajnością na słabym sprzęcie. Oprócz CGB emulator obsługuje też GBA, gdzie emulacja CGB i GBA są realizowane jako praktycznie osobne moduły. Emulator obsługuje argumenty z wiersza poleceń oraz posiada wsparcie dla Linuxa. Warto zaznaczyć że jest to oprogramowanie o otwartym źródle.

2.6.2. Przegląd emulatora SameBoy

Dużym plusem jest to, że skupia się na emulacji CGB i robi to w sposób dokładny. Jedyną rzeczą, która może być uciążliwa jest to, że wszystkie operacje wykonywane są poprzez GUI. Samo wczytywanie pliku ROM wymaga użycia graficznego menu lub przeciągnięcia pliku myszą, nie ma opcji wczytywania np. przez wiersz poleceń. Emulator ten przeznaczony jest do samodzielnego działania, nie nadaje się do komponowania go z zewnętrznym systemem lub skryptami. Również posiada wsparcie dla Linuxa i posiada otwarte źródła.

2.6.3. Przegląd emulatora BGB

Posiada bardzo rozbudowane opcje do debugowania, takie jak np. podgląd do pamięci VRAM, podgląd stanu wszystkich rejestrów I/O, tłumaczenie operacji na znanych adresach. Można załadować ROM z wiersza poleceń. Niestety nie posiada oficjalnego wsparcia dla Linuxa. Kolejną wadą jest to, że jego kod źródłowy nie jest udostępniony.

2.6.4. Podsumowanie

Po przeglądzie dostępnych emulatorów doszliśmy do wniosku, że w razie porażki w napisaniu własnego, wykorzystalibyśmy mGBA do osadzenia na komputerze jednopłytkowym. Emulator mGBA jest na tyle elastyczny że łatwo by go było oskryptować lub osadzić w samodzielnie wytworzonym interfejsie, a przede wszystkim działa natywnie na Linuxie nawet na słabym sprzęcie.

Emulator SameBoy również mógłby zostać użyty, jednak osadzenie go w systemie mogłoby być utrudnione ze względu na jego GUI, natomiast BGB nie może być wykorzystany z racji braku wsparcia dla Linuxa. Oczywiście można by skorzystać z warstwy kompatybilności WINE, ale to oznaczałoby dodatkowe narzuty na zasoby, które i tak będą ograniczone.

2.7. Różnice między rozwiązaniem wytworzonym a istniejącymi - Michał Zalewski

Postawione specyficzne cele projektu sprawiają, że większość dostępnych rozwiązań nie jest wystarczająca, aby zrealizować je w pełni. Różne istniejące emulatory spełniają pewne wymagania, które stawiane są wytwarzanemu systemowi, ale żaden z nich, zgodnie z wiedzą zespołu wytwórczego, nie spełnia wszystkich.

Jedną z najważniejszych cech docelowego oprogramowania jest działanie w środowisku o ograniczonych zasobach z wydajnością pozwalającą na płynną emulację z prędkością podobną do oryginalnej konsoli. Tworzony w ramach projektu emulator nie zakłada rozszerzania gamy emulowanych urządzeń o dodatkowe konsole, dzięki czemu możliwe jest zmniejszenie poziomu abstrakcji implementacji i wprowadzenie szczegółowych optymalizacji. Jest to podejście różniące się znacznie od niektórych alternatywnych rozwiązań, jak na przykład tych opartych
o bibliotekę *libretro*.

Używana technologia również została dobrana pod kątem zwiększenia szybkości wykonania. Używana biblioteka do obsługi wejścia/wyjścia użytkownika jest jednym z najszybszych dostępnych rozwiązań [4], a język programowania ma nieduży narzut wydajnościowy. Konfiguracja otoczenia programowego jest również ograniczona do elementów koniecznych dla działania systemu: system operacyjny jest pozbawiony większości zbędnych narzędzi i działa na nim minimalny menedżer okien.

Głównym kryterium oceny działania emulatora jest wierność działaniu oryginalnego urządzenia. Jednak zwykle wraz z dokładnością emulacji rosną wymagania sprzętowe dla środowiska, w którym uruchomiony jest emulator. W ramach projektu zdecydowano się na pewne środki zmniejszające dokładność emulacji (opisane w dalszych rozdziałach), które jednocześnie znacząco zwiększają wydajność i w normalnym działaniu gier nie zmieniają efektu końcowego, który widzi użytkownik.

Istnieją emulatory, które działają z większą dokładnością i dzięki temu są w stanie odtworzyć różne odbiegające od normy zjawiska występujące na oryginalnym urządzeniu w wyniku błędów w kodzie programów, do czego nie będzie zdolne wytworzone rozwiązanie.

Za cel projektu przyjęto stworzenie emulatora, który w łatwy sposób będzie mógł obsłużyć wiele różnych kontrolerów zewnętrznych. Dzięki użyciu mechanizmu z zewnętrznej biblioteki wejścia/wyjścia automatycznie gotowe do użycia są wszystkie kontrolery, które nie używają niestandardowych kodów dla przycisków. Dodatkowo do sterowania może zostać użyta klawiatura, a wykorzystywane przyciski są konfigurowalne za pomocą specjalnego pliku konfiguracyjnego.

Większość emulatorów pozwala na łatwe użycie klawiatury i dopasowanie używanych klawiszy do własnych upodobań, ale podłączenie kontrolera wymaga często korzystania z dodatkowych narzędzi tłumaczących wejście z niego na przyciśnięcia klawiszy na klawiaturze lub nieintuicyjnej konfiguracji.

Ważnym oczekiwaniem zespołu projektowego była możliwość rozwijania oprogramowania w środowisku innym niż wdrożeniowe ze względu na wygodę i możliwości sprzętu. Wybór technologii obsługującej interakcję z użytkownikiem i systemem operacyjnym był obarczony ryzykiem niewystarczającej wydajności. Oba te fakty zostały uwzględnione w analizie wymagań i projektowaniu, w wyniku czego zdecydowano się na korzystanie z minimalnej liczby zewnętrznych zależności.

Używana jest jedna wieloplatformowa biblioteka multimedialna i mechanizmy udostępniane przez system operacyjny zgodne ze standardem POSIX. Dzięki temu wytworzony kod jest w dużym stopniu przenośny między różnymi systemami operacyjnymi i platformami sprzętowymi. Dodatkowo, jeżeli zaistniałaby potrzeba użycia emulatora w środowisku, które nie jest aktualnie obsługiwane warstwowa architektura zapewnia, że do przeniesienia go będą wymagane zmiany tylko w określonych miejscach w kodzie adaptujące nowe narzędzia do potrzeb emulatora. Jest to tym łatwiejsze, że udostępniony jest kod źródłowy.

Wiele istniejących rozwiązań opiera swoje działanie na konkretnych technologiach tracąc przez to na rozszerzalności. Nie każdy twórca emulatora udostępnia jego kod źródłowy, więc nawet gdy jego architektura pozwalałaby na rozszerzenie go o obsługę dodatkowych środowisk, to nie dojdzie do tego, jeżeli nie będzie to leżeć w interesie twórcy.

Należy również wspomnieć o unikalnej funkcji implementowanej przez stworzony emulator, jaką jest automatyczne kolorowanie gier z oryginalnego DMG. Jest to funkcja, która występowała w Game Boy Color, a nie jest implementowana w większości emulatorów, więc jest czynnikiem wyróżniającym wytworzony program.

3. ARCHITEKTURA SYSTEMU - MICHAŁ ZALEWSKI

3.1. Koncepcja

Na podstawie analizy domeny problemowej projektu opracowana została architektura systemu. W pierwszym kroku podzielono wymagane funkcje na te, które są realizowane w ramach programu emulatora i te, które wymagają osobnych komponentów usługowych.

W ramach emulatora wydzielono dwie zasadnicze warstwy: warstwę logiki emulacji oraz warstwę interakcji z użytkownikiem. Pierwsza z nich odpowiada za odwzorowanie działania układów elektronicznych wewnętrznej logiki oryginalnej konsoli, a druga stanowi adapter między logiką emulacji, a środowiskiem wykonywania programu. Dodatkowo w emulatorze istnieją komponenty pomocnicze pomagające w rozwijaniu emulatora. Uproszczony podział komponentów programowych został przedstawiony na rysunku 3.1.

Przedstawiony podział komponentów pomiędzy warstwę odpowiadającą za samą emulację działania konsoli i warstwę współpracującą ze sprzętem umożliwia łatwą zmianę otoczenia programowego emulatora (używanych bibliotek, sprzętu, systemu operacyjnego). Nie wymaga ona dostosowywania komponentów odpowiadających za emulację, a ogranicza się tylko do komponentów warstwy interakcji z użytkownikiem. Oznacza to, że portowanie emulatora na inne platformy powinno być łatwe i pozbawione ryzyka wprowadzania błędów w emulacji.

Takie funkcje systemu jak zarządzanie katalogiem i uruchamianie gier, oraz dodawanie nowych tytułów z urządzenia zewnętrznego pozostawiono do zrealizowania poza programem emulatora, w samodzielnych komponentach.

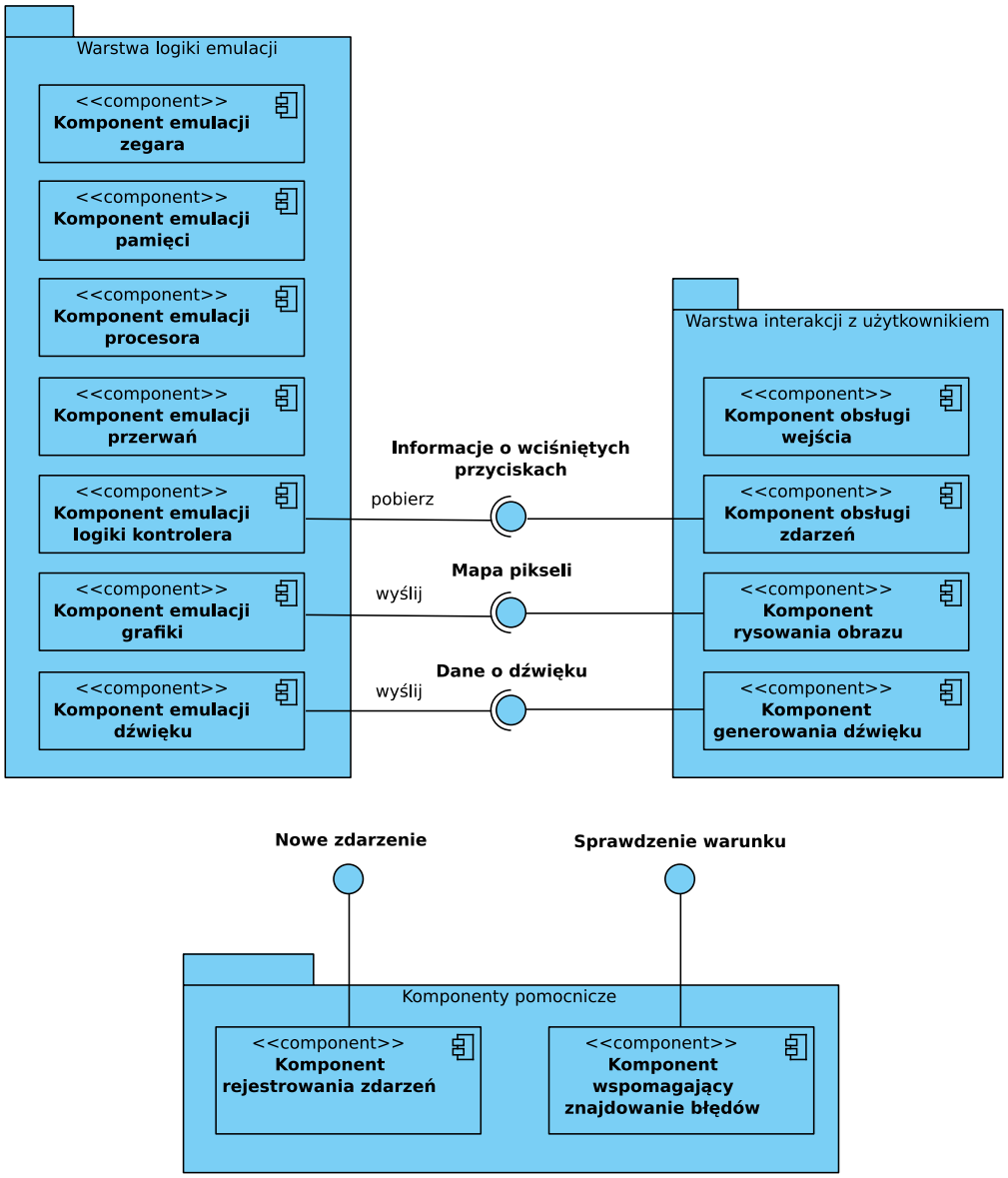
Dokonany został również podział na komponenty sprzętowe, przy czym całość programowej implementacji mieści się w ramach pojedynczego mikrokomputera, a do komponentów zewnętrznych należą: kontroler, ekran, głośnik i pamięć zewnętrzna do przenoszenia plików gier do systemu. Ogólne działanie wszystkich komponentów systemu zostało przedstawione
na rysunku 3.2.

3.2. Główne warstwy architektoniczne emulatora

3.2.1. Warstwa logiki emulacji

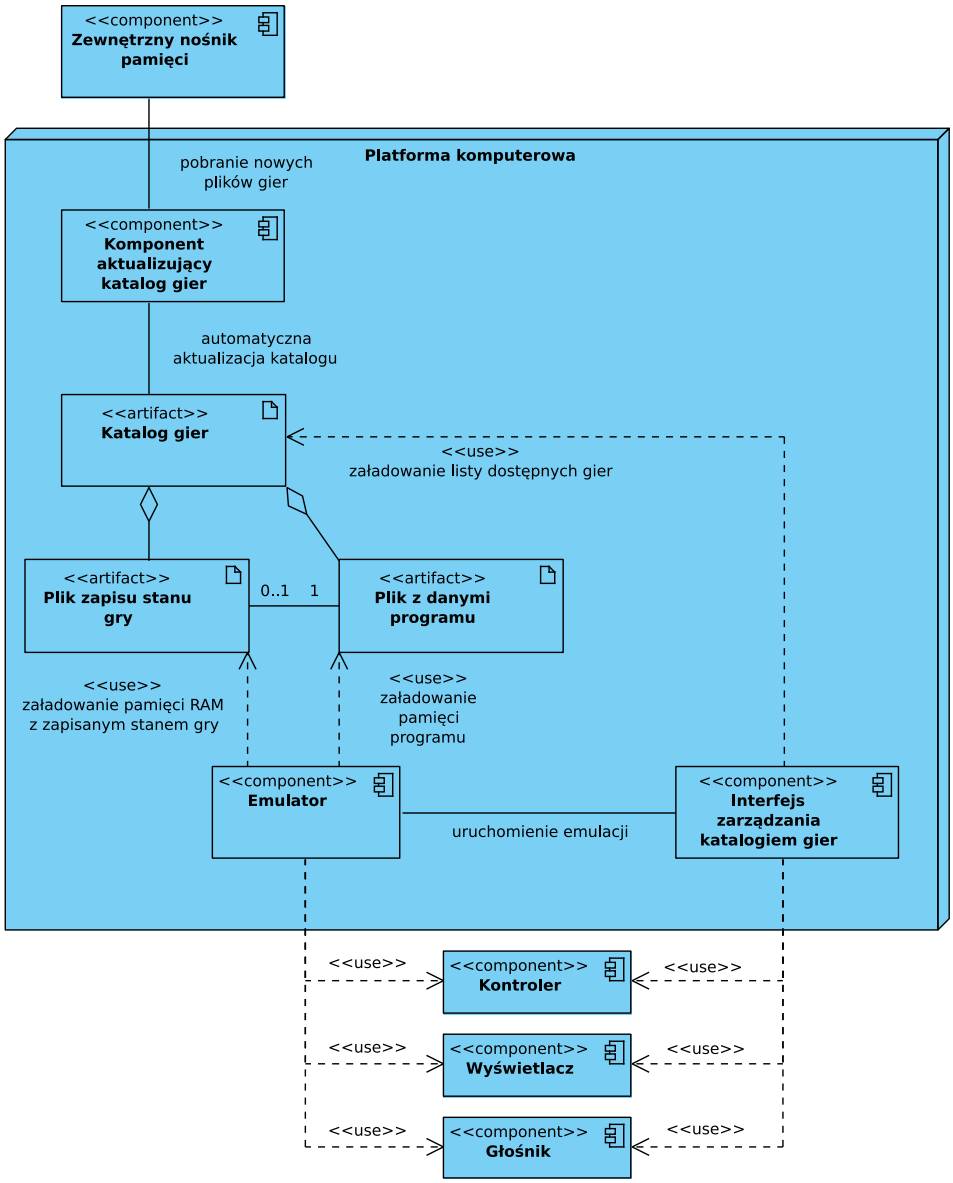
Warstwa logiki emulacji odpowiada za wierne odtworzenie efektów działania wewnętrznej logiki konsoli. Dodatkowo z przyjętego sposobu zapisu plików gier (w postaci surowej pamięci ROM zgranej z kartridża) wynika konieczność emulacji układu Memory Bank Controller (MBC), oryginalnie obecnego wewnątrz kartridża, która również realizowana jest w tej warstwie jako część emulacji mapowania pamięci.

Warstwa działa z dokładnością do wykonanej instrukcji procesora. Decyzja ta skutkuje zmniejszeniem dokładności emulacji, ale jednocześnie znacznie zmniejsza ilość wykonywanych obliczeń [2]. Umożliwia to płynne działanie emulatora w środowisku o ograniczonej mocy obliczeniowej, co jest kluczowe dla zrealizowania celów projektu.



Rys. 3.1. Diagram komponentów programowych emulatora. Dla czytelności pominięto wykorzystanie interfejsów udostępnianych przez komponenty pomocnicze – są one dostępne we wszystkich innych komponentach.

W praktyce oznacza to, że jeden krok emulacji odpowiada liczbie cykli, która jest wymagana do wykonania aktualnej instrukcji przez procesor. Komponenty, które wymagają synchronizacji czasowej z zegarem procesora, w każdym kroku są informowane o liczbie cykli, jaką teoretycznie zajęło wykonanie ostatniej instrukcji przez procesor.



Rys. 3.2. Diagram wdrożenia systemu w środowisku docelowym z wyróżnionymi komponentami. Komponenty programowe stanowiące części emulatora zostały przedstawione jako pojedynczy komponent *Emulator*.

Komponent emulacji pamięci

Komponent emulacji pamięci odpowiada za zarządzanie wewnętrzną pamięcią RAM konsoli, do której zalicza się pamięć:

- VRAM – *Video RAM*, pamięć, w której zapisane są fragmenty grafiki odczytywane przez układ graficzny (m. in. fragmenty tła i obrazków elementów świata gry),
- HRAM – *High RAM*, pamięć dostępna do użytku procesora. Pierwotnie był zwykle używany do przechowywania stosu wywołań,
- WRAM – *Working RAM*, robocza pamięć RAM konsoli, do wykorzystania przez program,
- OAM – *Object Attribute Memory*, tablica atrybutów obrazków wykorzystywana do rysowania obrazu.

Dodatkowo obsługiwana jest pamięć, która występuje oryginalnie na kartridżu.
Składają się na nią:

- ROM – pamięć zawierająca kod wykonywanego programu,
- RAM – dodatkowa pamięć operacyjna dostępna do użycia przez program. Jeżeli kartridż zawiera wewnętrzną baterię, to zawartość tej pamięci nie zmienia się między uruchomieniami konsoli, co pozwala na zapisywanie w niej stanu gry.
  Nie występuje we wszystkich kartridżach.

Zarówno pamięć ROM, jak i RAM kartridża może zostać zwielokrotniona poprzez wykorzystanie mechanizmu tzw. banków pamięci. Dostępny obszar pamięci jest wtedy podzielony na równe części, z których tylko jedna jest dostępna w danym momencie dla procesora konsoli (wyjątkiem jest zawsze dostępny bank 0 pamięci ROM). Zmiana dostępnego banku odbywa się poprzez wpisanie numeru żądanego banku pod odpowiedni adres w mapie pamięci. Po tym układ MBC zaczyna mapować zawartość nowego banku po otrzymaniu adresu w pamięci. Ilość i rozmiar banków pamięci zależy od użytego w kartridżu układu MBC.

Wewnętrzna pamięć WRAM i VRAM również korzysta z podobnego sposobu podziału na banki. W tym wypadku jednak dostępna ilość i rozmiar banków są znane z góry, ponieważ pamięci te są częścią konsoli. W wypadku konsoli GB dostępne są dwa banki WRAM po 4 KB i jeden bank VRAM o rozmiarze 8 KB, natomiast w GBC dostępnych jest 8 banków WRAM po 4 KB i dwa banki VRAM po 8 KB.

Komponent pamięci jest odpowiedzialny za obsługę banków pamięci kartridża. Przy inicjalizacji wczytuje z banku 0 pamięci ROM, z nagłówka programu, bajt opisujący typ kartridża. Na jego podstawie określana jest wersja układu MBC, z której korzystał kartridż i alokowana jest pamięć, która będzie odpowiadała bankom pamięci ROM i RAM kartridża.

Poza obsługą poszczególnych przestrzeni pamięci, komponent emulacji pamięci zajmuje się również delegowaniem zapisów i odczytów na adresach należących do przestrzeni portów wejścia/wyjścia. Adresy te odpowiadają rejestrom w innych modułach sprzętowych oryginalnej konsoli. W architekturze emulatora za obsługę poszczególnych adresów również odpowiadają odpowiednie dla nich komponenty, które rejestrują się do obsługi danego adresu przy inicjalizacji.

Część adresów z przestrzeni portów wejścia/wyjścia jest jednak ściśle związana z komponentem emulacji pamięci i tenże komponent rejestruje dla nich własną obsługę. Zapisy i odczyty na tych adresach pozwalają na ustawianie i sprawdzenie używanego banku WRAM i VRAM, jak również kontrolowanie mechanizmów bezpośredniego dostępu do pamięci (DMA).

Game Boy Color używa dwóch rodzajów mechanizmu DMA: Object Attribute Memory (OAM) DMA i HDMA. OAM DMA pozwala na kopiowanie zawartości pamięci ROM lub RAM do pamięci atrybutów obrazków. W czasie trwania transferu procesor ma dostęp tylko do pamięci HRAM. HDMA natomiast pozwala na kopiowanie zawartości pamięci z ROM kartridża, RAM kartridża i WRAM do określonego fragmentu VRAM. Może się to odbywać w dwóch trybach: ogólnego użytku, gdzie całość pamięci jest kopiowana od razu, lub poprzez kopiowanie porcji pamięci w okresach pomiędzy rysowaniem kolejnych linii. Komponent emulacji pamięci musi imitować to zachowanie w sposób, który z punktu widzenia wykonywanego programu nie będzie odbiegał od działania oryginalnej konsoli.

Komponent pozwala na dostęp do wszystkich wymienionych wyżej obszarów pamięci poprzez podanie adresu w liniowej przestrzeni adresowej identycznej z urządzeniem oryginalnym (przedstawionej w tabeli 3.1). Oznacza to, że musi on posiadać wewnętrzną logikę, która będzie przeprowadzała odpowiednie mapowanie adresów do określonych banków pamięci i obsługi rejestrów w innych komponentach.

Tabela 3.1. Mapa logicznej przestrzeni adresowej w Game Boy Color

Adresy	Opis
0x0000-0x3FFF	Bank 0 pamięci ROM kartridża
0x4000-0x7FFF	Bank 1-N pamięci ROM kartridża
0x8000-0x9FFF	VRAM
0xA000-0xBFFF	Bank 0-N pamięci RAM kartridża
0xC000-0xCFFF	Bank 0 pamięci WRAM
0xD000-0xDFFF	Bank 1-7 pamięci WRAM
0xE000-0xFDFF	Nieużywana pamięć
0xFE00-0xFE9F	OAM
0xFEA0-0xFEFF	Nieużywana pamięć
0xFF00-0xFF7F	Porty wejścia/wyjścia
0xFF80-0xFFFE	HRAM
0xFFFF	Rejestr kontrolny przerwań procesora

Komponent emulacji procesora

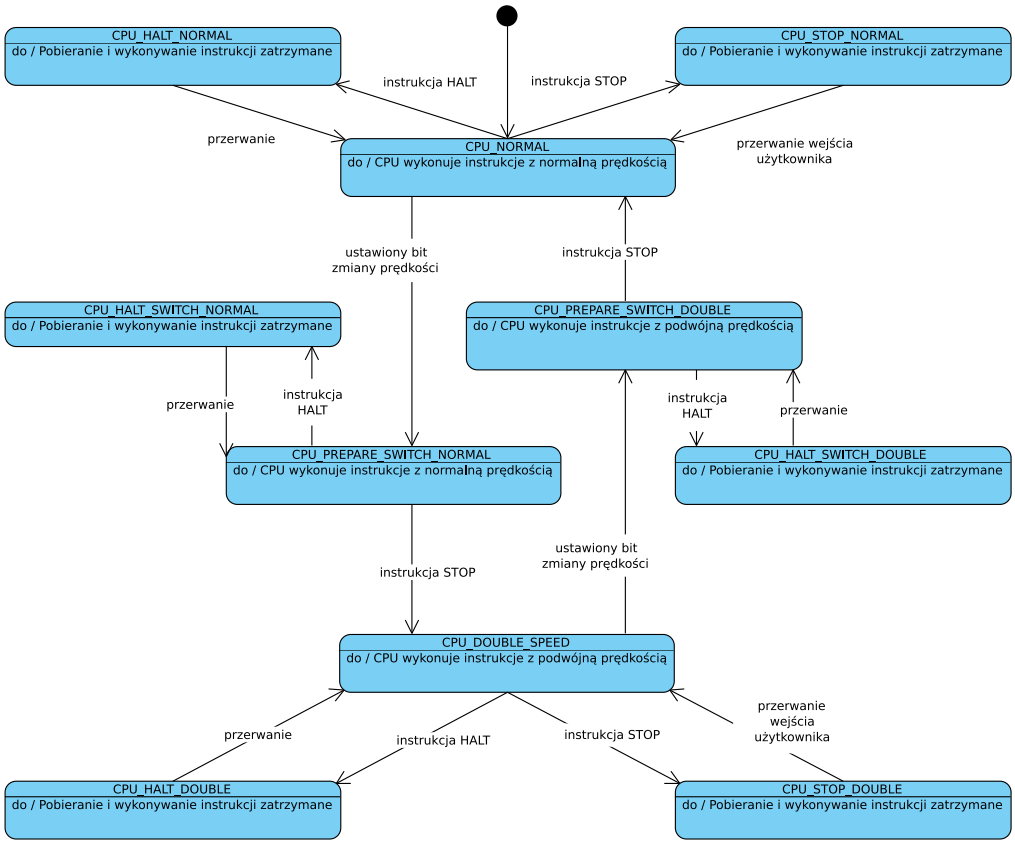
Emulacja procesora w systemie koncepcyjnie sprowadza się do pobrania instrukcji spod adresu podanego w rejestrze licznika programu (z użyciem komponentu emulacji pamięci), wykonania instrukcji, inkrementacji licznika programu (PC) i zwróceniu wartości: liczby cykli procesora, które wykorzystałby oryginalny procesor na wykonanie instrukcji. Ta następnie jest wykorzystywana przez kolejne komponenty do synchronizacji czasowej emulacji. Takie podejście pozwala na realizację emulacji działania oryginalnego procesora z dokładnością co do instrukcji.

Podczas wykonywania programu istotne jest przechowywanie stanu rejestrów procesora pomiędzy wykonywanymi instrukcjami, ponieważ jest w nich zapisany stan wykonywanego programu. W związku z tym komponent emulacji procesora posiada własną pamięć poświęconą na przechowywanie stanu rejestrów. Dodatkowo, przy uruchomieniu konsoli poszczególne rejestry procesora przyjmują standardowe wartości zależne od typu konsoli, co pozwala wykonywanym programom na określenie środowiska, w którym są uruchamiane. Komponent inicjalizuje stan rejestrów procesora zgodnie z emulowanym modelem konsoli.

Instrukcjami dla których zachowanie procesora różni się od standardowego są STOP i HALT. Obie instrukcje wprowadzają procesor w tryb oszczędności energii, w którym nie są wykonywane żadne instrukcje. Wyjście z tego trybu jest możliwe po wystąpieniu przerwania procesora, które jest wtedy obsługiwane, po czym wykonywana jest kolejna instrukcja
po instrukcji zatrzymania (STOP/HALT).

Instrukcja STOP jest także używana do zmiany tryby pracy GBC między trybem normalnej, a podwójnej szybkości, jeżeli został uprzednio ustawiony odpowiedni bit w rejestrze kontrolującym szybkość. Rejestr ten jest obsługiwany przez komponent emulacji procesora i zarejestrowany w komponencie emulacji pamięci pod odpowiednim adresem.

W ten sposób instrukcje STOP i HALT wprowadzają dodatkowe stany, w których może znajdować się procesor. Można przedstawić mechanikę działania emulacji procesora jako maszynę stanów, w której przejścia między stanami są wywoływane przez instrukcje STOP i HALT oraz przerwania procesora (zob. rysunek 3.3).



Rys. 3.3. Maszyna stanów komponentu emulacji procesora związana z użyciem instrukcji HALT i STOP oraz użyciem mechanizmu podwójnej prędkości dostępnym w GBC.

Komponent emulacji przerwań

Komponent emulacji przerwań ma trzy główne zadania:

- udostępnienie innym komponentom warstwy logiki emulacji interfejsu pozwalającego na zgłaszanie przerwań,
- wykorzystanie interfejsu komponentu emulacji procesora do wywołania odpowiedniej procedury obsługi przerwania
- obsługa rejestrów kontrolujących działanie przerwań.

Konsola GBC korzysta z pięciu rodzajów przerwań: po zakończeniu rysowania ekranu (rozpoczęcie okresu V-blank), przy przejściu między trybami pracy układu graficznego, przy przepełnieniu rejestru zegara, przy zakończeniu transferu seryjnego i przy wciśnięciu przycisku przez użytkownika. Jako że wszystkie te zdarzenia dotyczą różnych komponentów, komponent emulacji przerwań dostarcza im centralny punkt, do którego mogą zgłosić wystąpienie zdarzenia, które mają wywołać przerwanie.

Po zgłoszeniu przerwania komponent za pośrednictwem interfejsu komponentu emulacji CPU wpisuje adres aktualnej instrukcji na wierzchołek stosu wywołań (wskazywanego przez rejestr SP procesora), a do rejestru PC wpisuje odpowiedni adres procedury obsługi przerwania, co z perspektywy procesora wygląda jak wywołanie zwykłej procedury.

Emulowany procesor pozwala wykonywanemu programowi na wyłączenie przerwań procesora i na odczyt aktualnie ustawionych przerwań. Odbywa się to poprzez rejestry Interrupts Enable (IE, włączanie/wyłączanie przerwań) i Interrupt Flags (IF, aktualnie ustawione przerwania) zmapowane na adresy w przestrzeni pamięci. Obsługa tych rejestrów leży po stronie komponentu emulacji przerwań, który rejestruje się w tym celu przez interfejs komponentu emulacji pamięci.

Komponent emulacji zegara

Konsola GBC udostępnia wykonywanemu programowi układ zegarowy pozwalający mu na dostęp do rejestru zegarowego taktowanego zegarem procesora oraz konfigurowalnych rejestrów pochodnych. Komponent emulacji zegara odpowiada za odtworzenie tego zachowania w emulatorze.

Za pomocą interfejsu komponentu emulacji pamięci komponent jest rejestrowany do obsługi odczytywania i zapisywania odpowiednich rejestrów. Aktualna wartość rejestrów jest przechowywana wewnątrz komponentu i obliczana w każdym kroku emulacji na podstawie liczby cykli procesora, którą zajęła ostatnia instrukcja.

Dodatkowo w momencie przepełnienia rejestru pochodnego wysyłane jest przerwanie procesora. Odbywa się to przy użyciu interfejsu udostępnianego przez komponent emulacji przerwań.

Komponent emulacji grafiki

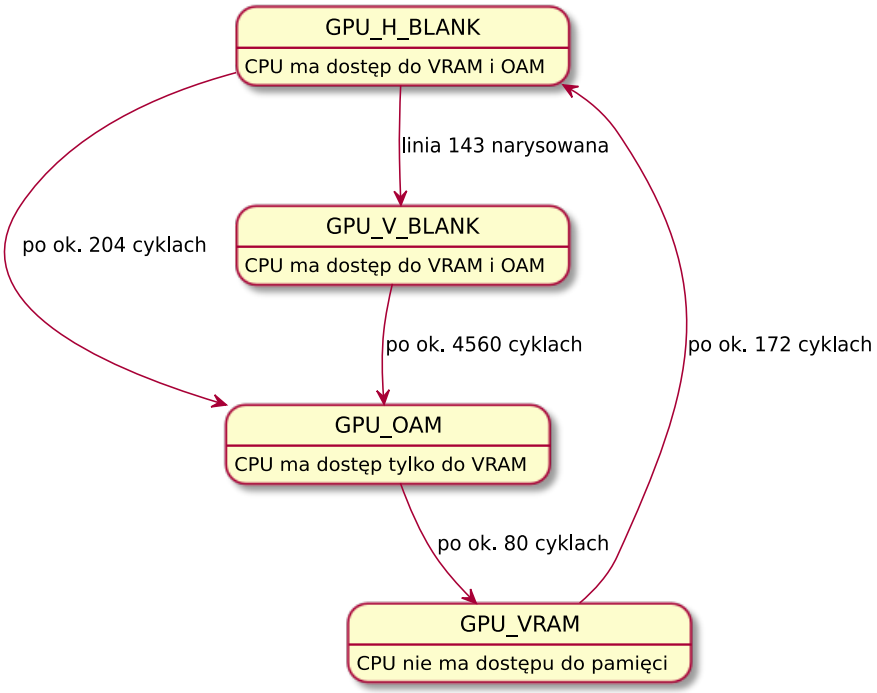
Logika rysowania grafiki konsoli Game Boy Color jest jednym z najbardziej zaawansowanych aspektów jej działania. Pobiera dane z pamięci VRAM, OAM i pamięci palet kolorów dla tła i sprite'ów – małych obiektów graficznych, takich jak np. postacie w grze. Jej wyjściem jest mapa pikseli z narysowanymi w odpowiednich miejscach fragmentami tła, okna gry oraz sprite'ów.

Oryginalnie zawartość ekranu rysowana jest w sposób ciągły, kolejne piksele od lewej strony ekranu do prawej, a kolejne linie od góry do dołu. Wyróżniane są cztery
tryby pracy układu graficznego:

- skanowanie OAM – przed rysowaniem linii. W tym czasie w oryginalnej konsoli pamięć OAM jest przeszukiwana pod kątem sprite'ów znajdujących się w linii, która będzie rysowana. W tym trybie CPU nie ma dostępu do OAM.
- rysowanie linii na podstawie OAM, VRAM i palet kolorów – w tym trybie niemożliwe jest odczytywanie pamięci OAM ani VRAM przez CPU.
- wygaszanie poziome (Horizontal blanking) – po narysowaniu pojedynczej linii. CPU ma dostęp do całej pamięci video.
- wygaszanie pionowe (Vertical blanking) – po narysowaniu całego ekranu. CPU ma dostęp do całej pamięci video.

Przejścia pomiędzy poszczególnymi trybami występują po upływie określonego czasu (przedstawione na Rys. 3.4).

Emulacja rysowania w tym komponencie ma dokładność ograniczoną do jednej linii. Oznacza to, że rysowanie odbywa się dopiero, gdy upływa czas przeznaczony na narysowanie pojedynczej linii (przy przejściu do stanu H-blank) i jest ona rysowana w całości. Komponent emulacji grafiki posiada własny bufor o wymiarach odpowiadających wymiarom ekranu konsoli, w którym zapisywane są na bieżąco kolory każdego piksela.



Rys. 3.4. Diagram maszyny stanów pamięci GPU pokazujący, do których części pamięci video ma dostęp CPU w różnych momentach emulacji GPU

Podczas rysowania wykorzystywane są dane z pamięci VRAM i OAM, do których dostęp odbywa się za pośrednictwem interfejsu komponentu emulacji pamięci.

Po zakończeniu rysowania całego ekranu (przy przejściu do stanu V-blank) zawartość bufora znajdującego się w komponencie jest przekazywana do komponentu rysowania obrazu, który na jej podstawie prezentuje go użytkownikowi.

Poza wpisywaniem danych do pamięci VRAM i OAM program kontroluje działanie układu graficznego i uzyskuje informacje o jego stanie poprzez szereg rejestrów kontrolnych. Ich stan, zapisany w komponencie, jest używany i aktualizowany w trakcie kroku emulacji. Dostęp do nich z poziomu programu jest realizowany przez rejestrację obsługi określonych adresów w pamięci z użyciem interfejsu komponentu emulacji pamięci.

Przy przejściu do V-blank jest również zgłaszane odpowiednie przerwanie procesora z użyciem interfejsu udostępnianego przez komponent obsługi przerwań.

Komponent emulacji dźwięku

Zadaniem komponentu emulacji dźwięku jest obsługa rejestrów sterujących wytwarzaniem dźwięku i dedykowanej pamięci RAM przeznaczonej do zapisu dowolnego docelowego kształtu fali dźwiękowej. Podczas każdego kroku emulacji dane te powinny być przetwarzane, aby uzyskać wartości fali dźwiękowej dla czasu odpowiadającego czasowi wykonania najdłuższej instrukcji w docelowej częstotliwości próbkowania. Następnie wartości te powinny zostać przesłane do komponentu generowania dźwięku wraz z liczbą cykli zegara procesora, które teoretycznie zajmuje wykonanie ostatniej instrukcji przez CPU.

W ramach projektu komponent emulacji dźwięku został zrealizowany tylko w szczątkowej formie. Zrealizowana została obsługa rejestrów kontrolnych układu dźwiękowego wystarczająca, aby z punktu widzenia wykonywanego programu układ wydawał się funkcjonalny, lecz żadne dane wpisywane przez program do rejestrów i dedykowanej pamięci nie są wykorzystywane do wytwarzania dźwięku.

Komponent emulacji logiki kontrolera

Zadania komponentu emulacji logiki kontrolera sprowadzają się do obsługi odczytu i zapisu dla rejestru kontrolera przez wykonywany program. Przy każdym kroku emulacji komponent odpytuje komponent obsługi zdarzeń o nowe zdarzenia wejścia użytkownika, po czym aktualizuje wewnętrzną reprezentację informacji o wciśniętych klawiszach kontrolera.

Jest również zarejestrowany w module emulacji pamięci do obsługi rejestru kontrolera. Dzięki temu program jest w stanie wybrać, czy zwracana będzie informacja o wciśniętych klawiszach kierunkowych, czy funkcyjnych przez wpisanie odpowiedniej wartości do rejestru, po czym możliwe jest odczytanie i wykorzystanie wejścia od użytkownika w programie.

Dodatkowo, po otrzymaniu informacji o wciśnięciu przycisku, komponent ustawia przerwanie odpowiadające wejściu użytkownika za pomocą interfejsu udostępnianego przez komponent emulacji przerwań.

3.2.2. Warstwa interakcji z użytkownikiem

Zadaniem warstwy interakcji z użytkownikiem jest zaprezentowanie użytkownikowi interfejsu, przez który może komunikować się z emulatorem używając komponentów sprzętowych oraz interakcja z systemem operacyjnym. Jej komponenty nie zajmują się emulacją działania oryginalnej konsoli, tylko dostarczają dane wejściowe i informacje z systemu operacyjnego do warstwy emulacji i przekazują jej wyjścia do interfejsu użytkownika.

Warstwa ta jest uzależniona od konkretnych, wykorzystywanych w danej platformie rozwiązań, takich jak zewnętrzne biblioteki, czy mechanizmy systemu operacyjnego. Potencjalne przeniesienie emulatora na inną platformę będzie się wiązało z dostosowaniem komponentów tej warstwy do udostępnianych w niej mechanizmów.

Komponent obsługi zdarzeń

Wszystkie informacje przekazywane do systemu z otoczenia po jego inicjalizacji przyjmują postać asynchronicznych zdarzeń. Istotne dla procesu emulacji są zdarzenia związane z wejściem użytkownika – wciskanymi przyciskami – i zdarzenia czasowe występujące w regularnych interwałach, które pozwalają na synchronizację działania komponentów systemu.

Komponent obsługi zdarzeń odpowiada za obsługę opisanych wyżej zdarzeń i dostarcza interfejsy, przez które pozostałe komponenty mogą sprawdzić, czy wystąpiło dane zdarzenie w czasie ich normalnego, synchronicznego wykonywania. W szczególności udostępniane są informacje o zdarzeniach odpowiadających wciśnięciu przycisku przez użytkownika oraz o wystąpieniu zdarzenia czasowego synchronizującego wyświetlanie ramek na ekranie.

Do pobierania tego rodzaju zdarzeń musi zostać użyta dodatkowa zewnętrzna biblioteka. Po otrzymaniu informacji o wejściu użytkownika od biblioteki dane na jego temat są przekazywane komponentowi obsługi wejścia, który po przetworzeniu ich zwraca wewnętrzną reprezentację możliwą do późniejszego wykorzystania przez komponent emulujący logikę kontrolera. Dane przechowywane są w komponencie obsługi zdarzeń i udostępniane za pomocą interfejsu. Do momentu ich odczytania informacje o kolejnych zdarzeniach są dopisywane. Oznacza to, że każdy rodzaj zdarzenia (każdy wciśnięty klawisz) jest rejestrowany co najwyżej raz pomiędzy odczytaniami danych o zdarzeniach.

W celu realizacji funkcjonalności zdarzenia czasowego rysowania klatki również wymagane jest użycie zewnętrznego mechanizmu. Komponent rejestruje się w bibliotece, żeby ta wywołała jego funkcję po określonym czasie. Gdy zostanie wywołana, zapisywana jest informacja o zdarzeniu i komponent rejestruje swoją funkcję po raz kolejny. Informacja o zdarzeniu czasowym jest udostępniana przez interfejs komponentu obsługi zdarzeń i nie podlega buforowaniu – przy odczycie widoczne jest tylko czy od ostatniego odczytania wystąpiło zdarzenie czasowe bez określania krotności jego wystąpienia.

Komponent obsługi wejścia

Komponent obsługi wejścia pełni funkcję pomocniczą wobec komponentu obsługi zdarzeń. Udostępniony jest interfejs pozwalający na przekazanie informacji z zewnętrznej biblioteki na temat zdarzenia wejścia użytkownika. Przekazane dane są konwertowane do wewnętrznego formatu używanego przez komponent emulacji logiki kontrolera. Dane w takim formacie są zwracane do komponentu obsługi zdarzeń.

W tym komponencie zawarta jest logika dla różnych kontrolerów użytkownika, takich jak klawiatura, kontroler wbudowany lub zewnętrzny.

Komponent rysowania obrazu

Rysowanie obrazu na ekranie odbywa się z określoną przy starcie systemu częstotliwością. Jego synchronizacja jest niezależna od głównej pętli emulacji logiki konsoli. Po określeniu wartości koloru dla wszystkich pikseli na ekranie komponent emulacji układu graficznego zgłasza ten fakt do komponentu rysowania obrazu. W tym momencie komponent, przy użyciu interfejsu komponentu obsługi zdarzeń, sprawdza czy od ostatniego wywołania wystąpiło zdarzenie czasowe synchronizacji wyświetlania. Jeżeli nie, ten komunikat jest ignorowany i kontrola jest zwracana do emulacji grafiki. Jeżeli tak, zawartość wewnętrznego bufora kolorów pikseli z komponentu emulacji grafiki jest kopiowana, po czym zostaje narysowana na ekranie. Do rysowania wymagane jest użycie zewnętrznej biblioteki graficznej.

Poza samym rysowaniem komponent odpowiada za inicjalizację biblioteki, stworzenie okna, w którym będzie rysowany obraz i alokację zasobów potrzebnych do skopiowania bufora pikseli, jak również zasobów biblioteki koniecznych do rysowania obrazu.

Komponent generowania dźwięku

W związku z rezygnacją z realizacji emulacji dźwięku konsoli komponent generowania dźwięku nie został stworzony w ramach prac nad projektem. Jednakże założenia projektowe odnośnie funkcjonowania warstwy interakcji z użytkownikiem jasno sugerują jaką rolę pełniłby w systemie i w jaki sposób powinien funkcjonować w swoim otoczeniu.

Tak jak wszystkie pozostałe elementy systemu współpracujące ze sprzętem, również komponent generowania dźwięku wykorzystywałby zewnętrzną bibliotekę, aby wytworzyć dźwięk. Jego zasadniczym zadaniem byłoby odbieranie informacji o fali dźwiękowej od komponentu emulacji dźwięku i dostosowywanie jej do formatu oczekiwanego przez bibliotekę. W tym celu wymagane byłoby udostępnienie interfejsu do przesyłania surowych danych dźwiękowych
z warstwy emulacji.

3.2.3. Komponenty pomocnicze emulatora

Komponenty pomocnicze nie stanowią warstwy w standardowym rozumieniu tego słowa, ponieważ oferowane przez nie funkcje są dostępne dla wszystkich pozostałych komponentów. Ich zadaniem jest wspomaganie procesu wytwarzania emulatora, ułatwianie znajdowania błędów i określania przyczyn ich wystąpienia.

Komponent rejestrowania zdarzeń

Komponent rejestrowania zdarzeń dostarcza dostępny w całym systemie interfejs pozwalający na zapisywanie istotnych informacji o działaniu systemu. Informacje te mają nadany poziom istotności, który pozwala rozróżnić zdarzenia, które są częścią normalnego funkcjonowania systemu, te, które odbiegają od normy, lecz są automatycznie korygowane przez system i takie, które są krytycznym i nienaprawialnym odstępstwem on oczekiwanego działania.

Udostępnione są dwie formy przedstawiania informacji: standardowa, w której informacja zostaje wypisana dokładnie tak, jak została podana, lub szczegółowa, w której są przedstawione dodatkowe informacje o stanie krytycznych dla wykonywania programu komponentów.

Zależnie od wybranego poziomu istotności informacja zostaje wypisana na standardowe wyjście lub standardowe wyjście dla błędów systemu operacyjnego. Umożliwia to dowolne przekierowanie strumienia wyjścia w zależności od potrzeb użytkownika bez ingerencji
w kod programu.

Komponent wspomagający znajdowanie błędów

Kolejnym komponentem, którego interfejsy są dostępne w całym systemie jest komponent wspomagający znajdowanie błędów. Udostępnia on mechanizm tzw. asercji, czyli założeń co do poprawnego działania systemu przyjmowanych w różnych miejscach implementacji. Gdy założenie jest spełnione działanie systemu pozostaje niezmienione, natomiast gdy nie jest spełnione, działanie jest przerywane i, za pośrednictwem komponentu rejestrowania zdarzeń, zostają wypisane informacje pomagające w diagnozie i naprawieniu zaistniałego problemu.

Komponent udostępnia również interfejs ułatwiający wypisywanie instrukcji pobieranych przez procesor, co pozwala na odczytanie historii wykonywanego programu i jego interpretację. Dzięki temu łatwiej jest stwierdzić z czego wynikają występujące błędy i wykryć problemy w logice emulacji układów konsoli.

3.3. Komponenty usługowe poza emulatorem

W przedstawianej architekturze rola emulatora została ograniczona do emulacji samego procesu rozgrywki. Przypomina to działanie oryginalnej konsoli, która również nie zawiera żadnych elementów interfejsu użytkownika i bezpośrednio po włączeniu uruchamiana jest na niej gra. Jednakże w określonym wcześniej modelu użytkowania opisywanego systemu wymagane są pewne dodatkowe możliwości.

Użytkownik musi być w stanie dodać nowe gry do systemu oraz uruchomić wybraną grę. Ponadto musi również istnieć możliwość wyłączenia systemu gdy nie jest używany. Te funkcje są realizowane przez dodatkowe komponenty działające w ramach systemu
operacyjnego środowiska wdrożeniowego.

3.3.1. Interfejs zarządzania katalogiem gier

Komponent zarządzania katalogiem gier jest uruchamiany od razu po zakończeniu inicjalizacji systemu operacyjnego. Jest to wymagane, aby ograniczyć aktywność użytkownika w ramach systemu operacyjnego do funkcji udostępnianych przez rozwijany system.

Interfejs użytkownika umożliwia wybranie konkretnej gry z kolekcji, uruchomienie jej, zmianę jej nazwy i metadanych oraz usunięcie pozycji.

3.3.2. Komponent aktualizujący katalog gier

Dodawanie nowych gier jest realizowane poprzez podłączenie zewnętrznego nośnika pamięci zawierającego pliki gier do platformy komputerowej. Po podłączeniu komponent aktualizujący jest automatycznie uruchamiany przez odpowiednio skonfigurowane mechanizmy systemu operacyjnego. Jego zadaniem jest zamontowanie urządzenia w systemie i skopiowanie nowych plików gier do wyznaczanego katalogu przechowującego kolekcję. Po tej operacji możliwe jest odłączenie urządzenia zewnętrznego.

3.4. Komponenty sprzętowe

Jako że jednym z celów projektu jest wytworzenie urządzenia, na którym będzie docelowo działał emulator, do elementów architektury systemu należą również komponenty sprzętowe, które umożliwiają użytkownikowi korzystanie z systemu. Opisanie ich jako część architektury jest istotne z punktu widzenia komponentów mających bezpośrednią styczność ze środowiskiem programowym, tj. komponentów warstwy interakcji z użytkownikiem emulatora i komponentów usługowych funkcjonujących poza nim, ponieważ ograniczają zasób możliwych do wykorzystania rozwiązań technologicznych takich jak biblioteki, czy mechanizmy udostępniane przez system operacyjny. Ważne jest również dopasowanie wydajności implementacji do możliwości obliczeniowych docelowego sprzętu.

3.4.1. Platforma komputerowa

System jest wdrażany w obrębie pojedynczej platformy komputerowej, która pozwala na przenoszenie konsoli i zasilenie jej bez ciągłego podłączenia do sieci elektrycznej. Jest wystarczająco mała, żeby zmieścić się w poręcznej obudowie.

Platforma działa pod kontrolą procesora, który pozwala na używanie nowoczesnego systemu operacyjnego, co umożliwia wykorzystanie rozwijanych aktualnie bibliotek do interakcji programu z użytkownikiem. Ważne jest również, że procesor ma wystarczającą moc obliczeniową, żeby

uruchomić emulator i komponenty pomocnicze i pozwolić na ich płynne działanie, w przypadku emulatora przynajmniej z prędkością równą oryginalnej konsoli.

Od platformy komputerowej wymagane jest posiadanie portów pozwalających podłączyć dużą część dostępnych w powszechnym użyciu zewnętrznych kontrolerów i wyświetlaczy. Dodatkowo platforma posiada interfejsy konieczne do podłączenia kontrolera i wyświetlacza stanowiącego zintegrowane części systemu. Dostępny jest również port pozwalający na podłączenie standardowych typów nośników pamięci przenośnej.

3.4.2. Kontroler

System posiada wbudowany kontroler w postaci przycisków odpowiadających przyciskom oryginalnej konsoli. Przyciski te są podłączone do platformy w sposób pozwalający na identyfikację ich przez system operacyjny jako urządzenia wejścia użytkownika. Biblioteka wykorzystywana przez komponent obsługi zdarzeń nasłuchuje na zdarzenia wejścia z przycisków i po wykryciu zdarzenia wywołuje procedurę obsługi z komponentu.

Możliwe jest również podłączenie dodatkowego zewnętrznego kontrolera, którego obsługa jest identyczna do wbudowanych przycisków.

3.4.3. Wyświetlacz

Zintegrowany wyświetlacz stanowi część docelowego systemu. Domyślnie wyświetlana jest na nim zawartość ekranu generowana przez system operacyjny. Zastosowanie odpowiednich ustawień wyświetlania pozwala na użycie całego ekranu zarówno w interfejsie zarządzania katalogiem gier, jak i podczas działania emulatora, dzięki czemu użytkownik nie ma dostępu do innych elementów systemu operacyjnego i zapewniona jest spójność interfejsu użytkownika.

Możliwe jest podłączenie zewnętrznego wyświetlacza za pomocą standardowego portu. W tej sytuacji wbudowany wyświetlacz przestaje pokazywać zawartość ekranu, która jest za to rysowana na zewnętrznym wyświetlaczu korzystając z automatycznych
mechanizmów systemu operacyjnego.

3.4.4. Głośnik

W związku z ograniczeniem zakresu projektu głośnik nie stanowi części systemu. W razie ewentualnej kontynuacji prac nad projektem możliwe jest dodanie głośnika, który byłby obsługiwany przez system operacyjny. Możliwe byłoby również podłączenie zewnętrznego urządzenia audio za pomocą standardowego portu. W takiej sytuacji wbudowany głośnik powinien zostać wyłączony, a dźwięk powinien być przesyłany do urządzenia zewnętrznego.

3.4.5. Zewnętrzny nośnik pamięci

Nośnik pamięci nie stanowi zintegrowanej części systemu, ale jest ważnym elementem przeprowadzającym interakcje z systemem. Jest on podłączany przez standardowy port dla tego typu urządzeń. Jego pamięć jest sformatowana według jednego z obsługiwanych systemów plików, a pliki z zawartością pamięci ROM gier są umieszczone w katalogu nadrzędnym
i mają ustalone rozszerzenia.

4. IMPLEMENTACJA

4.1. Technologie - Adrian Misiak

Celem projektu, oprócz napisania samego emulatora, jest umieszczenie go na mikroplatformie komputerowej. Postawienie takiego celu nałożyło dodatkowe ograniczenia i nakreśliło priorytety, które były brane pod uwagę podczas doboru technologii. Dodatkowo sam wybór sprzętu skutkował potrzebą użycia pewnych rozwiązań. Dlatego oprócz przedstawienia technologii programowych użytych w ramach projektu, fragment zostanie poświęcony opisaniu w skrócie użytych rozwiązań sprzętowych i nadaniu kontekstu naszym decyzjom.

4.1.1. Rozwiązania oprogramowania

C11 – Cały kod składający się bezpośrednio na nasz emulator został napisany w języku C. Wybór tego języka zamiast C++ został dokonany ze świadomością, że rezygnujemy z możliwości obiektowych, które by nam oferował. Uzasadnieniem podjętej przez nas decyzji była możliwość uzyskania szybszego programu o niższym wykorzystaniu zasobów, co u nas stanowi najważniejszy priorytet. Oczywiście istnieje wiele innych języków programowania o konkurencyjnej szybkości. Decyzja na język C zapadła za zgodą całego zespołu projektowego, jako technologia, w której wszyscy mają jakieś doświadczenie. Jako pewien dodatkowy element, o którym warto wspomnieć przy kwestii wyboru języka jest to, że do naszego projektu zaadoptowaliśmy pewne części stylu programowania jądra Linux [6].

SDL2 – Biblioteka przeznaczona do tworzenia gier lub programów multimedialnych. Daje dostęp do urządzeń wejścia oraz sprzętu graficznego. W początkowych fazach projektu zamiast niej wykorzystywana była biblioteka Allegro 5. Już w trakcie projektu pojawiły się pewne problemy z wydajnością emulatora i po przejrzeniu pracy porównującej szybkości pomiędzy bibliotekami: SDL, Allegro oraz SFML [4] okazało się, że przejście na SDL2 może przynieść korzyści. Ponieważ sposób korzystania z obu bibliotek jest do siebie zbliżony oraz mając na uwadze fakt, że wszystkie operacje związane z grafiką działy się w osobnej warstwie, postanowiliśmy przepisać ten fragment korzystając z nowej biblioteki. Dodatkowo SDL2 oferuje Controller API do obsługi kontrolerów, za pomocą którego obsługa różnych urządzeń peryferyjnych wejścia jest łatwiejsze od odpowiednika – Joystick Routines z Allegro 5.

Pthreads – Biblioteka POSIX udostępniająca funkcje do zarządzania wątkami. W naszym przypadku została użyta do oddelegowania wypisywania logów do osobnego wątku. Zostało to wykonane w ramach testu, którego wynikiem było lekkie przyśpieszenie działania systemu. Planowano również przeniesienie logiki rysowania na osobny wątek, aby jeszcze bardziej przyśpieszyć działanie emulatora. Na obecną chwilę nie zostało to zaimplementowane, ponieważ okazało się to niekrytyczne i zespół projektowy wolał skupić się na bardziej pilnych problemach. W przyszłości planujemy odstąpić od korzystania z tej biblioteki na rzecz API zarządzania wątkami udostępnionego przez SDL2.

Python 3 – Język wysokiego poziomu. Oprócz samego emulatora potrzebowaliśmy funkcjonalności wczytywania plików ROM. Nie musi ona być zoptymalizowana ponieważ jest stosowana stosunkowo rzadko i nie przebiega w trakcie samej emulacji. Uznaliśmy więc że skorzystamy z języka Python do napisania prostego skryptu pozwalającego na kopiowanie plików ROM z podłączonej pamięci USB do wyznaczonego katalogu w pamięci urządzenia.

udev – System plików dostępny na Linuxie pozwalający na zarządzanie urządzeniami. W skład udev wchodzi demon udevd, który nasłuchuje na zdarzenia uevent wysyłane przez jądro systemu w momencie dodania/usunięcia nowego urządzenia. Można definiować jakie akcje mają być podjęte przy danym zdarzeniu za pomocą reguł. Używając prostej reguły jesteśmy w stanie automatycznie kopiować nowe pliki ROM poprzez wywołanie naszego skryptu napisanego w Pythonie od razu po podłączeniu urządzenia z pamięcią.

Raspberry Pi OS Lite – System operacyjny instalowany na układzie jednopłytkowym. Wybór był mocno związany z opisanym dalej wyborem sprzętu. Domyślna wersja Desktop posiada wiele zbędnych programów oraz środowisko graficzne. Ograniczyliśmy się do systemu w wersji Lite i dodaliśmy do niego tylko niezbędne elementy aby zwolnić jak najwięcej zasobów. Do wyświetlania emulatora zainstalowaliśmy
lekki menedżer okien Openbox.

EmulationStation – Konfigurowalny front-end dla emulatorów, który został napisany z wsparciem dla urządzeń Raspberry Pi, dzięki czemu idealnie nadaje się do naszego rozwiązania. Służy nam do wyboru oraz uruchamiania gier. Wydzielenie interfejsu do osobnej aplikacji pozwoliło nam na pisanie emulatora w taki sposób, aby mógł się całkowicie skupić na emulacji, pomijając prezentowanie menu użytkownikowi.

4.1.2. Rozwiązania sprzętowe

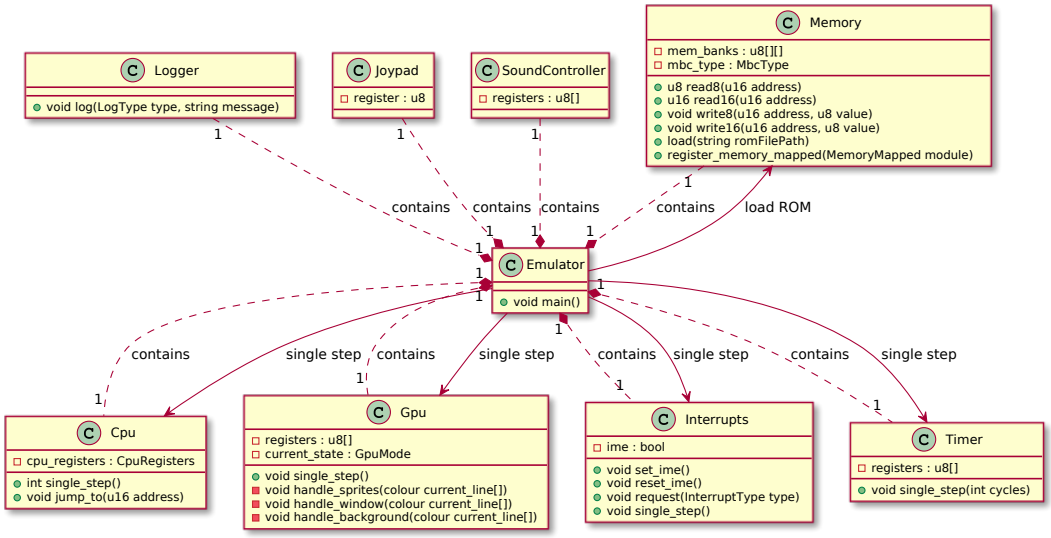
Raspberry Pi 4B 4GB RAM – Komputer jednoukładowy, posiada 4 rdzeniowy procesor o architekturze ARM z taktowaniem 1.5GHz. Wybraliśmy model posiadający 4 GB pamięci, ponieważ różnica cenowa między tym samym modelem o pamięci 2 GB nie była znacząca. Wybrany model jest stosunkowo nowy (wydany w roku 2019), dzięki czemu posiada wydajne podzespoły. Postąpiliśmy tak, aby dać sobie trochę miejsca na błędy. Docelowo emulator powinien być coraz wydajniejszy i jeżeli okaże się wystarczająco wydajny, niewykluczona będzie próba przeniesienia naszego systemu na starsze modele.

Adafruit PiTFT 2.8" – Chcemy aby układ był przenośny, więc zdecydowaliśmy się zamontować mały ekran. Wybrany model posiada rozdzielczość 320x240. Nie udało się znaleźć ekranu o rozdzielczości odpowiadającej oryginalnemu wyświetlaczu konsoli CGB (160x144). Montaż i korzystanie z ekranu poprzez Raspberry Pi jest proste. Korzystamy z portów GPIO na układzie oraz skryptów instalacyjnych udostępnionym przez Adafruit [7]. Ekran posiada z boku kilka przycisków, które później mogą zostać wykorzystane do wywoływania dodatkowych funkcji emulatora. Oprócz małego wyświetlacza do układu można podłączyć kolejny wyświetlacz poprzez złącze HDMI.

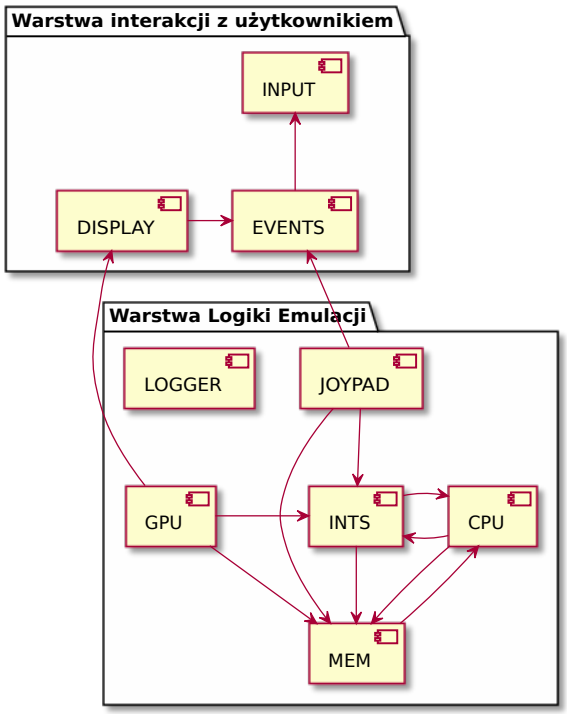
PiGRRL 2.0 Custom Gamepad PCB – Aby układ mógł być przenośny oprócz wyświetlacza uznaliśmy za potrzebne dodanie wbudowanego kontrolera. Przewidujemy też opcję korzystania z klawiatury albo popularnych kontrolerów do gry, ale chcieliśmy, aby cały system mógł być używany samoistnie. Uznaliśmy za stosowne użyć do tego celu płytki PCB utworzonej w ramach projektu PiGRRL 2. Posiada ona dobrą integrację z wybranym wyświetlaczem, połączenie również odbywa się poprzez porty GPIO. Płytka posiada więcej przycisków niż wymagane dla samego emulatora, co daje późniejsze możliwości wprowadzenia dodatkowych funkcji emulacyjnych.

4.2. *Logiczny szkielet aplikacji - moduły - Adrian Misiak*

Korzystając z wcześniejszej analizy zidentyfikowaliśmy najważniejsze byty i dokumentowaliśmy je za pomocą klas analitycznych, a później komponentów. Są to jednak modele na wysokim poziomie abstrakcji. Technologia, którą wybraliśmy jednoznacznie sugeruje pisanie aplikacji w paradygmacie proceduralnym. W takim przypadku mapowanie klas analitycznych, które są łatwo przekładalne na obiektowe klasy projektowe nie jest proste dla naszego projektu.



Rys. 4.1. Skrót analitycznego diagramu klas. Podkreśla występowanie jednej instancji każdej znaczącej klasy.



Rys. 4.2. Diagram przedstawiający ogólne spojrzenie na moduły i relacje między nimi. Dla czytelności powiązania do modułu LOGGER zostały pominięte, ponieważ wszystkie inne elementy korzystają z niego.

Robiąc przegląd naszych wcześniejszych analiz zauważyliśmy, że praktycznie wszystkie znaczące klasy analityczne w naszym systemie 4.1 są dosyć rozległe i można je traktować jak samoistne komponenty, oraz że byłyby realizowane w postaci singletona. Podjęta została więc decyzja projektowa, że podczas implementacji naszego systemu klasy analityczne przełożone na komponenty zostaną implementowane w postaci modułów. Moduły oraz połączenia między nimi przedstawione zostały na diagramie 4.2. Strzałki na diagramie oznaczają, że dany moduł może wywoływać metody elementu, na który wskazuje.

Tutaj, tak jak i na wcześniejszych etapach projektowych, wyróżnione zostały dwie warstwy. Założenie było takie, aby potencjalne przeniesienie emulatora na inną platformę wiązało się z jak najmniejszym narzutem dodatkowej pracy. Warstwa logiki emulacji jest odpowiedzialna za samą emulację i powinna w jak najmniejszym stopniu zależeć od sprzętu, na jakim się znajduje. Natomiast warstwa interakcji z użytkownikiem będzie wysoce zależna od zastosowanych bibliotek zewnętrznych do operowania z grafiką i wejściem. Przy takich założeniach proces portowania emulatora powinien sprowadzić się do podmiany modułów w warstwie interakcji na zgodne z nową platformą i minimalnych zmian, jeżeli już jakichś, w punktach styku między warstwami.

Moduły będą realizować funkcjonalności zgodne z założeniami przedstawionymi w architekturze systemu. Podczas prac implementacyjnych pojawiło się kilka dodatkowych modułów. W ramach uściślenia przedstawione i pokrótce opisane zostaną moduły znajdujące się w systemie w czasie pisania tej pracy, a nieposiadające swojego własnego komponentu
w architekturze systemu.

Tabela 4.1. Podsumowanie faktycznie implementowanych modułów

Nazwa	Implementuje komponent
Planowane odgórnie	
CPU	Komponent emulacji procesora
GPU	Komponent emulacji grafiki
INTS	Komponent emulacji przerwań
MEM	Komponent emulacji pamięci
JOYPAD	Komponent emulacji logiki kontrolera
TIMER	Komponent emulacji zegara
SOUND	Komponent emulacji dźwięku
DISPLAY	Komponent rysowania obrazu
INPUTS	Komponent obsługi wejścia
EVENTS	Komponent obsługi zdarzeń
INPUTS	Komponent obsługi wejścia
LOGGER	Komponent rejestrowania zdarzeń
DEBUG	Komponent wspomagający znajdowanie błędów
Powstałe w trakcie implementacji	
ROM	Komponent emulacji pamięci
REGS	Komponent emulacji procesora
SYS	BRAK

W specjalnej sekcji *Powstałe w trakcie implementacji* w tabeli 4.1 zostały wymienione moduły, którym nie można w oczywisty sposób przypisać komponentu, jaki realizują. Wynika to z tego, że pełnią rolę pomocniczą i uzupełniają funkcję pozostałych modułów. Należą do nich:

ROM – Udostępnia funkcje odczytywania metadanych z nagłówka pliku ROM. Tak naprawdę wspomaga działanie modułu MEM.

REGS – Udostępnia funkcję odpowiedniego ustawienia wartości początkowych rejestrów wykorzystywanych przez CPU.

SYS – Wczytuje parametry wejściowe programu i je interpretuje. Nie jest powiązany bezpośrednio z żadnym komponentem emulacji. Wykorzystywany jest w głównej funkcji programu.

4.2.1. Główna pętla programu

Do pełnego zrozumienia działania emulatora należy jeszcze wspomnieć jak wykorzystane są moduły w głównym programie. Na listingu 4.1 przedstawiono uproszczony schemat funkcji *main* emulatora. Pominięte zostało wczytywanie parametrów wejściowych i przekazywanie ich do funkcji przygotowujących dane moduły. Schemat działania programu jest następujący:

1. Moduł CPU emuluje wykonanie jednej instrukcji. Zwraca liczbę cykli która powinna upłynąć w ramach wykonania tej instrukcji.
2. Pozostałe moduły w każdym kroku przyjmują liczbę cykli podaną przez CPU i za jej pomocą synchronizują swoje działanie.

Listing 4.1: Uproszczony schemat głównej pętli programu

```
// Zalaczenie naglowkow wszystkich modulow
#include "cpu.h"
#include "gpu.h"
// etc.

int main()
{
        // Sekcja prepare, wywolywanie metod przygotowujacych moduly
        cpu_prepare();
        gpu_prepare();
        // etc.

        int cycles_delta = 0;
        while(cycles_delta != -1 && !display_get_closed_status()) {
                cycles_delta = cpu_single_step();
                gpu_step(cycles_delta);
                // etc.
        }

}
```

Warunkiem wyjścia z programu jest ustawienie statusu wyjścia ekranu na zamknięty za pomocą zdarzenia zamknięcia okna lub wciśnięcia odpowiedniego przycisku. Drugim warunkiem jest zwrócenie liczby -1 jako upłyniętych cyklów przez moduł CPU. Sytuacja ta występuje w przypadku, kiedy procesor napotka niezdefiniowany kod operacji. Może się to wydarzyć, ponieważ kod operacji jest pobierany z pamięci jako jeden bajt, potencjalnie jest to 256 wartości. Procesor LR35902 wykorzystuje tylko 245 z tych możliwości, czyli zostaje 11 kodów które nie stanowią poprawnej instrukcji. Z punktu widzenia emulacji oznacza to możliwe uszkodzenie pamięci lub błędne wykonywanie programu, w takim wypadku emulator zostaje wyłączony.

4.3. Schemat fizyczny - klasy i pliki - Adrian Misiak

Skutkiem zastąpienia klas modułami w naszym projekcie jest schemat fizyczny, za pomocą którego można w łatwy sposób odczytać schemat logiczny. Wynika to ze sposobu, w jaki jest zbudowany moduł. Postanowiliśmy, że będzie składał się on z:

- jednego pliku .c nazwanego tak jak moduł.
- dowolnej ilości plków nagłówkowych .h zawierających w swojej nazwie nazwę modułu.

Takie ograniczenia nie stanowią wszystkiego, co potrzebne do wytworzenia prawidłowego modułu. Dlatego w ramach zespołu zostały ustalone pewne zasady konstrukcji modułu obowiązujące wszystkich członków zespołu. Sprecyzowanie jasnych zasad było potrzebne, aby projekt pozostawał zarządzalny w późnych etapach implementacyjnych:

- Wszystkie funkcje modułu powinny posiadać przedrostek "**<NAZWAMODUŁU>_**"
- Wszystkie funkcje modułu, które mogą być używane przez inne moduły, powinny posiadać swoją deklarację w pliku nagłówkowym.
- Wszystkie funkcje modułu, które nie powinny być widoczne dla innych modułów powinny być zadeklarowane jako statyczne oraz dodatkowo posiadać przedrostek "_" w nazwie.
- Chcąc ograniczyć/rozszerzyć dostęp do pewnych funkcji modułu innemu elementowi należy dodać nowy plik nagłówkowy. Nazwa powinna sugerować zastosowanie funkcji udostępnianych i oczywiście zawierać nazwę modułu.

W podanych fragmentach kodu: 4.2 oraz 4.3 przedstawiono fragmenty modułu INTS skonstruowanego zgodnie z wyżej przedstawionymi zasadami.

Listing 4.2: Część pliku nagłówkowego modułu INTS

```
// Ustawia wartosc rejestru Interrupt Master Enable
void ints_set_ime(void);
// Zeruje wartosc rejestru Interrupt Master Enable
void ints_reset_ime(void);
// Ustawia stan poczatkowy modulu
void ints_prepare(void);
// Wywolanie procedury sprawdzenia wystapienia przerwania
void ints_check(void);
// Procedura pozwalajaca innym modulom zglaszac przerwania
void ints_request(enum ints_interrupt_type interrupt);
```

Listing 4.3: Część ciała pliku .c modułu INTS

```
static void _ints_write_handler(a16 addr, u8 data)
{
        // Implementacja
}

void ints_request(enum ints_interrupt_type interrupt)
{
        // Implementacja
}
```

Zasady tworzenia i korzystania z modułów nie są pilnowane przez żadne narzędzia, zostały jedynie spisane w projektowym wiki. Dlatego do efektywnego wytwarzania dużą uwagę, przynajmniej w początkowych fazach projektu, skupiono na wzajemnym wykonywaniu przeglądów kodu. Pozwalało to na wczesne wykrywania naruszeń założeń projektowych. W końcowych fazach projektu wszyscy członkowie byli już przyzwyczajeni do naszych wewnętrznych reguł i takich naruszeń nie było prawie wcale.

Plusem takiego schematu konstrukcji modułów jest to, że bardzo łatwo odtworzyć logiczny szkielet aplikacji. Wystarczy potraktować pliki .c jako moduły, a występujące w nich dyrektywy **#include** jako powiązania między modułami.

4.4. Opis współpracy między modułami - Łukasz Mrugała

O ile zostało już wytłumaczone działanie poszczególnych modułów systemu, dobrze jest spojrzeć na występujące między nimi nietrywialne interakcje, by móc go lepiej zrozumieć.

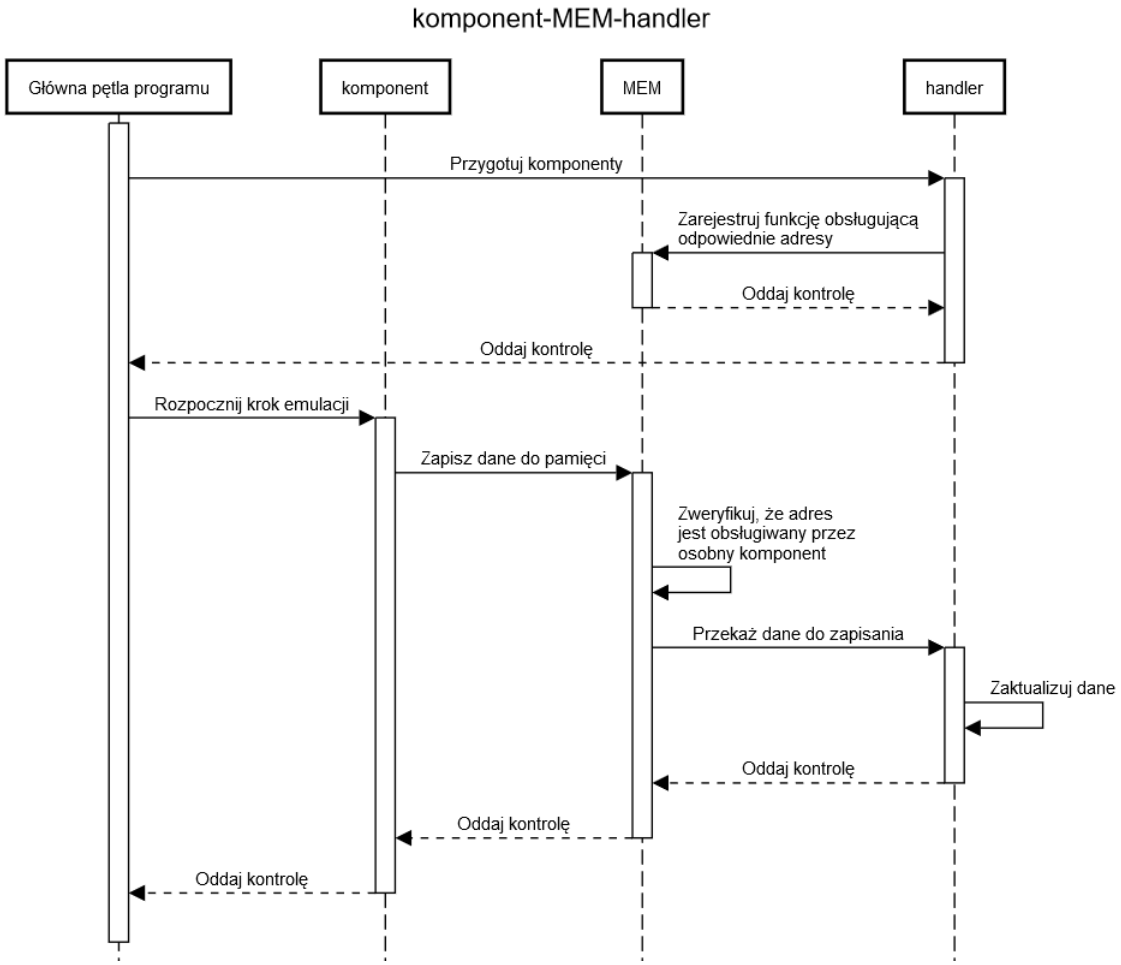
4.4.1. Obsługa adresów przez moduły inne niż MEM

CGB wykorzystywało część ze swoich adresów w pamięci jako rejestry, rozszerzając gamę używaną normalnie przez CPU. Wiele z nich było mocno związanych z pewnym modułem, jak na przykład adres $FFFF, zwany rejestrem przerwań, który był wykorzystywany przez moduł INTS.

W procesie tworzenia emulatora musieliśmy więc rozwiązać problem wynikający z tego faktu. Te dodatkowe rejestry często miały ze sobą związane pewne ograniczenia co do możliwości ich odczytu oraz zapisu, bądź też dodatkowe interakcje z innymi rejestrami. Przykładem mogą być tu rejestry BGPI i BGPD - gdy do BGPD zapisano wartość, była ona też zapisywana pod adres wskazywany przez rejestr BGPI.

Trzymanie całej logiki interakcji międzyrejestrowej i ograniczeń nań nałożonych w module MEM nie wydawało się na poprawne, gdy wiele z nich wynikało ze związania ze szczególnym modułem. Dlatego też zdecydowaliśmy się na rozwiązanie pokrótce przedstawione
na diagramie 4.3.

Dowolny komponent może, w swojej funkcji przygotowującej (_prepare()), zarejestrować pewne własne funkcje jako handlery (funkcje obsługujące) określonych adresów w module MEM. Wtedy, gdy MEM otrzymuje od któregoś z modułów w trakcie działania emulatora komendę zapisu pod dany adres, wpierw sprawdza on, czy zostały już zarejestrowane do niego handlery. Jeśli tak, przesyła on dane do zarejestrowanej funkcji dzięki zapisanym wskaźnikom do funkcji. Dzięki temu to moduł związany z rejestrem dodatkowym odpowiada za jego logikę.



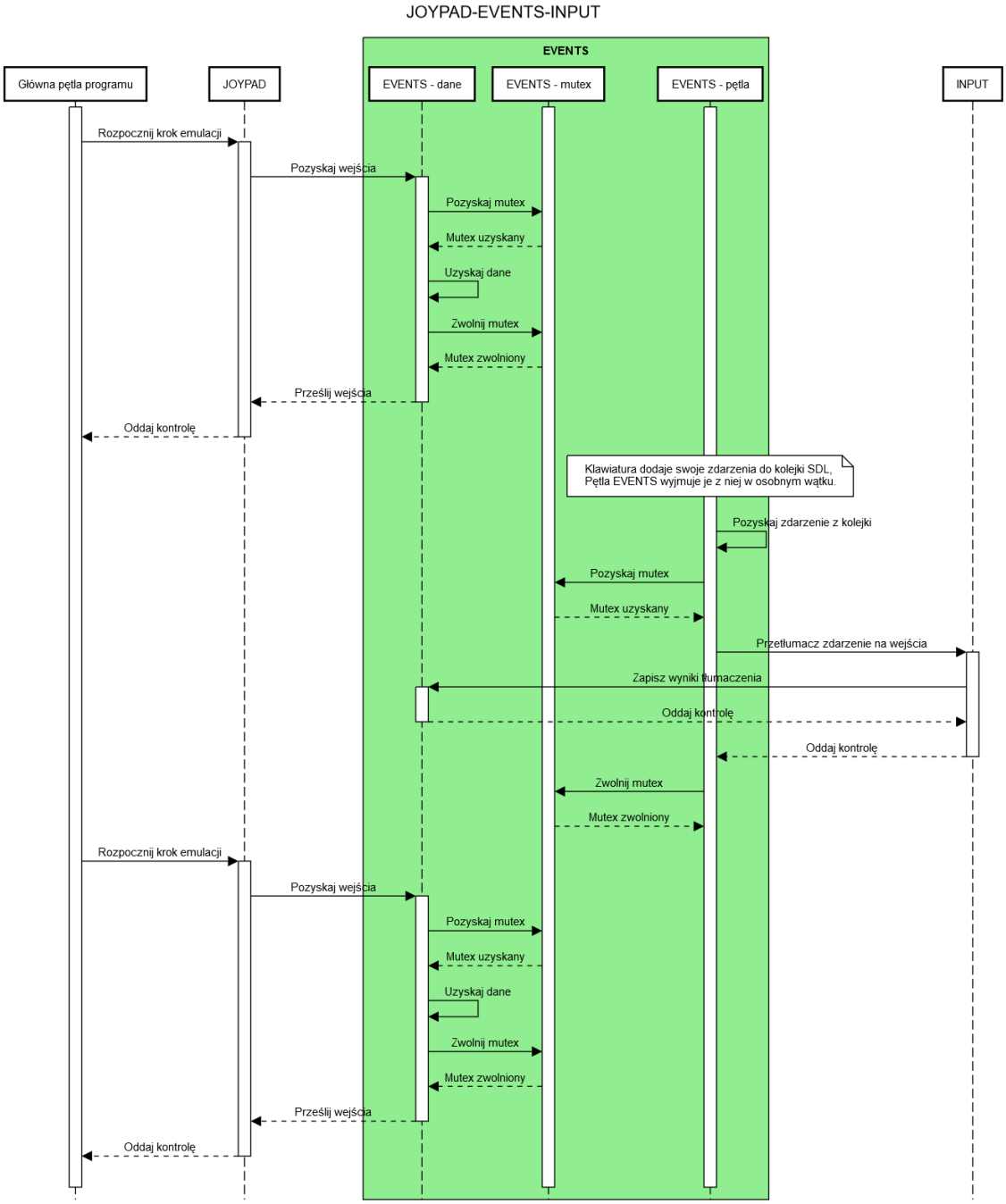
Rys. 4.3. Diagram interakcji pokazujący rejestrację i wykonanie funkcji odpowiadających za logikę zapisu/odczytu pamięci specyficznej dla innego modułu niż MEM.

4.4.2. Pobieranie zdarzeń wejścia

Pozyskiwanie wejścia od użytkownika w systemie interaktywnym często może sprawiać pewne problemy. W naszym wypadku było to zapewnienie poczucia responsywności systemu, jednocześnie nie używając architektury kierowanej zdarzeniami (*Event-driven architecture*).

Moduł JOYPAD odpowiada za przemianę niezależnych od platformy danych wejścia specyficznych dla naszego emulatora na odpowiednie wartości pamięci w nim oraz wywołanie wymaganych przerwań systemowych. We wcześniejszej wersji programu, wykorzystującej technologię Allegro 5, pozyskiwał on je bezpośrednio z modułu INPUT, który posiadał własną kolejkę zdarzeń klawiatury i innych urządzeń wejścia.

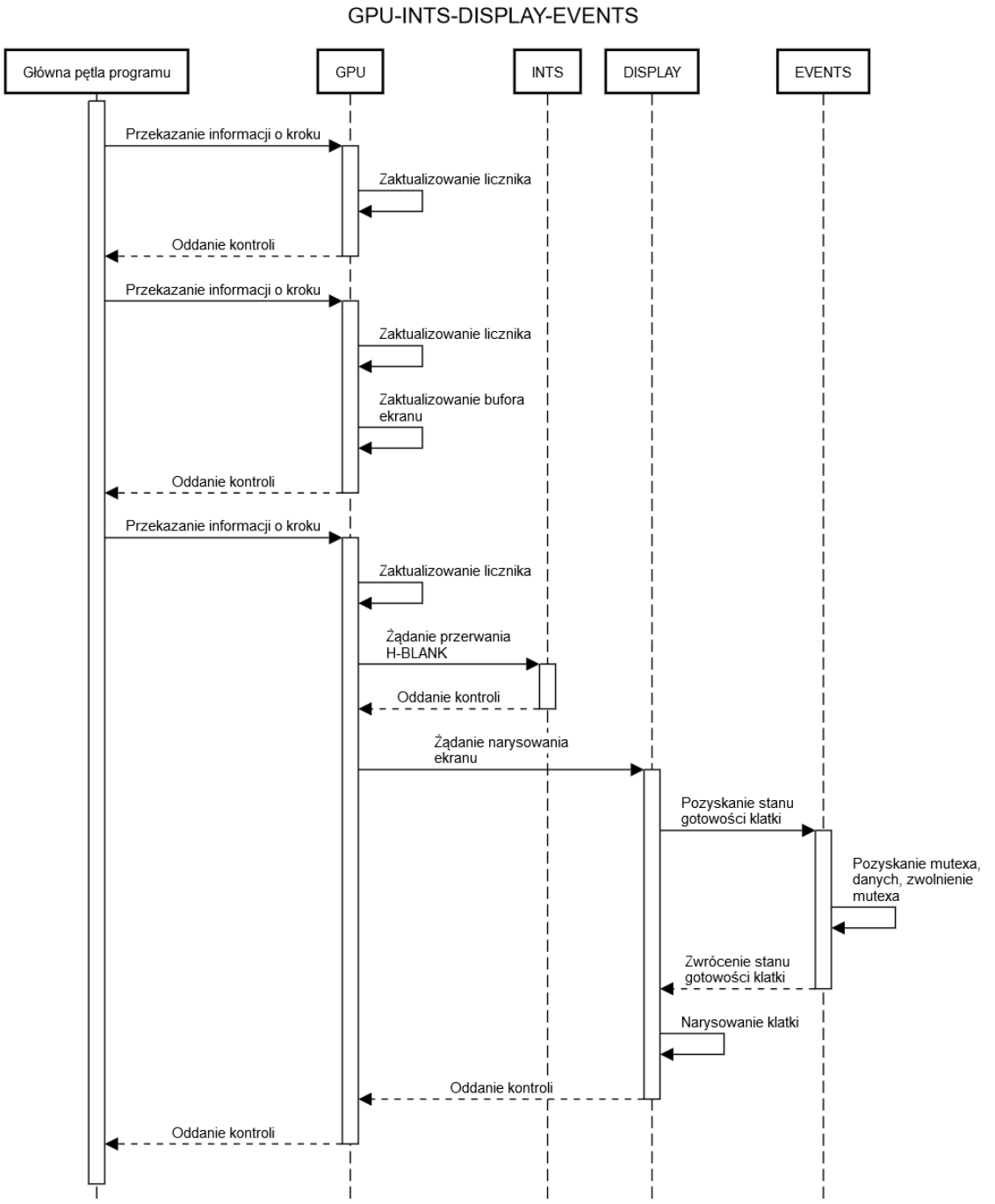
Ze względu na naszą decyzję, by użyć biblioteki SDL, istnieje pojedyncza kolejka zawierająca wszystkie zdarzenia asynchroniczne. Z tego względu moduł JOYPAD pozyskuje aktualne informacje z modułu EVENTS. Sam EVENTS przechowuje poprawny stan danych, ponieważ działa on na osobnym wątku, w którym, gdy zdejmie on z kolejki zdarzenie urządzenia wejścia, wysyła on je modułowi INPUT do przetłumaczenia. Zapobiegamy wyścigom używając mutexów dostarczanych przez SDL. Sytuację aktualną przedstawia diagram 4.4.



Rys. 4.4. Diagram interakcji pokazujący współdziałanie modułów JOYPAD, INPUT i działającego na osobnym wątku EVENTS w celu pozyskania i przetłumaczenia wejścia na dane w pamięci emulatora.

4.4.3. Generowanie i wyświetlanie obrazu

Ostatnią z nietrywialnych interakcji w naszym systemie jest sposób radzenia sobie z wyświetlaniem kolejnych klatek emulowanego programu przez nasz emulator. Jest ona przedstawiona na diagramie 4.5.



Rys. 4.5. Diagram interakcji pokazujący działania modułów GPU, INTS, DISPLAY i EVENTS w celu aktualizacji stanu procesora graficznego oraz wyświetlania nowych klatek z bufora w odpowiednim czasie.

Moduł GPU posiada specjalny wewnętrzny licznik pilnujący, kiedy powinien narysować kolejną linię ekranu. W momencie, gdy się to zdarzy, aktualizuje on fragment swojego bufora ekranu. Obsługuje on też rejestr LY, dzięki czemu wie, kiedy kończy rysować całość ekranu. W tym momencie informuje on moduł emulacji przerwań i przekazuje modułowi DISPLAY komendę narysowania ekranu. DISPLAY po uzyskaniu tej komendy wpierw sprawdza, czy moduł EVENTS otrzymał zdarzenie zegarowe mówiące, że należy już narysować klatkę. Jeśli i GPU, i EVENTS się zgadzają, DISPLAY rysuje ekran na podstawie bufora przekazanego przez GPU.

4.5. Implementacja sprzętowa - Adrian Misiak

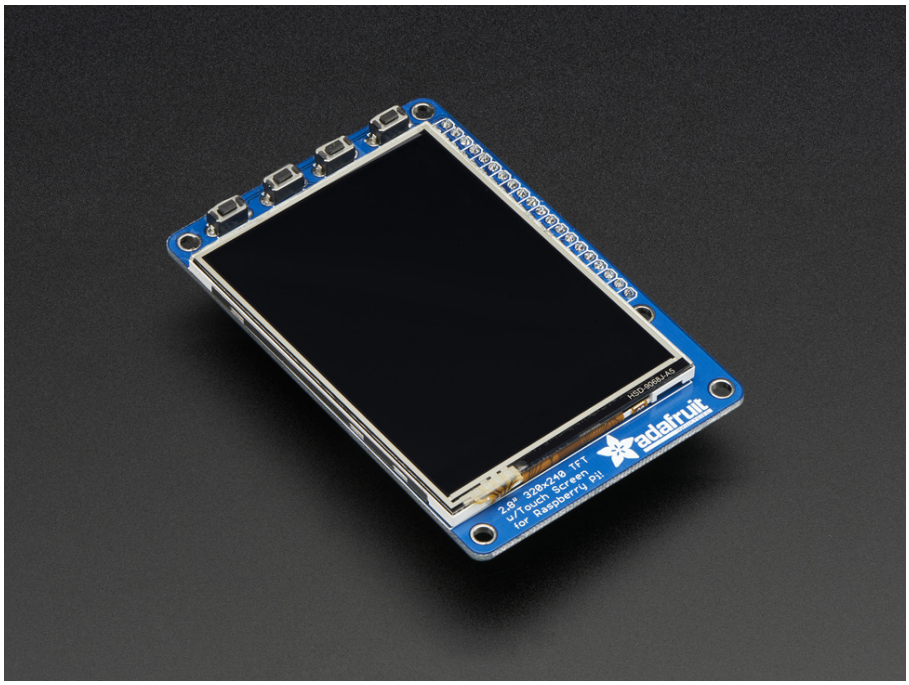
W ramach kompletności opisu prac wykonanych w ramach powstawania tego projektu zostanie przedstawiony proces łączenia komponentów sprzętowych w jedną
platformę zdolną do emulacji.

Sprzętem, który zastosowaliśmy jako platformę komputerową jest wymieniony wcześniej układ Raspberry Pi przedstawiony na zdjęciu 4.6. Posiada 4 porty USB, które będziemy wykorzystywać do podłączenia zewnętrznego nośnika pamięci zawierającego pliki z kopiami pamięci ROM gier przeznaczonych na konsole GBC/GB. Do tych portów również podłączane są różnego rodzaju kontrolery. Układ posiada dwa wejścia Micro HDMI, za pomocą których można wyświetlić kopię obrazu emulatora na dowolnym ekranie. Z bardziej istotnych, niewymienionych jeszcze elementów jest zestaw pinów GPIO 2 x 20, za pomocą którego podłączony został wyświetlacz.

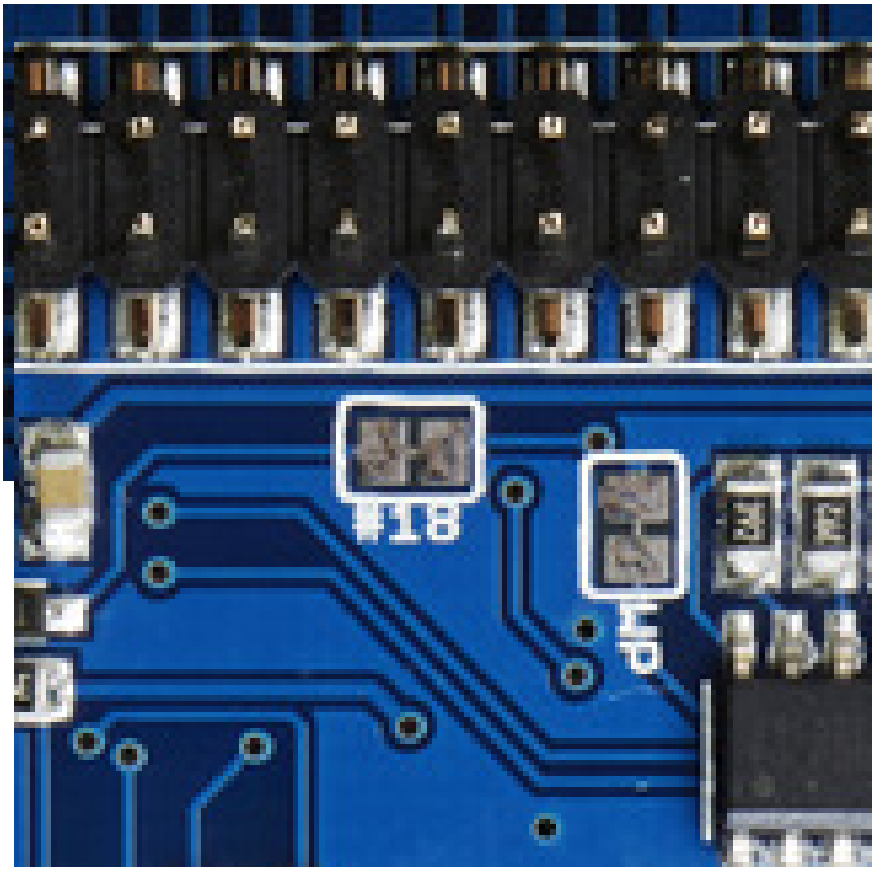
Rys. 4.6. Platforma komputerowa Raspberry Pi 4 model B z 4 GB pamięci RAM.

Wyświetlacz Adafruit PiTFT widoczny na zdjęciu 4.7 zajmuje wszystkie 40 pinów, ale za to udostępnia kolejne 40, dzięki czemu mamy miejsce, aby podłączyć wbudowany kontroler. Przed podłączeniem kontrolera do wyświetlacza należy jednak wykonać modyfikację na wyświetlaczu. Pin 18 układu wyświetlającego jest odpowiedzialny za sterowanie podświetlaniem (przedstawiony w źródle [8]), kolidowałoby to z płytką PCB wykorzystaną jako kontroler. Aby temu zaradzić należy ręcznie przeciąć zworkę lutowniczą oznaczoną **Lite #18** na układzie wyświetlacza (widoczną na zdjęciu 4.8).

Przygotowanie samego kontrolera wiązało się z przylutowaniem gniazda żeńskiego 2x20 pinów oraz 10 przycisków do płytki PCB przedstawionej na zdjęciu 4.9. Pozostają 4 nieużyte miejsca na przyciski typu "Bumper". Obecnie nie mamy dla nich żadnego zastosowania ponieważ emulator potrzebuje tylko 8 przycisków, ale pozostaje możliwość dodania ich jeżeli by zaszła taka potrzeba związana na przykład z rozszerzeniem funkcjonalności.

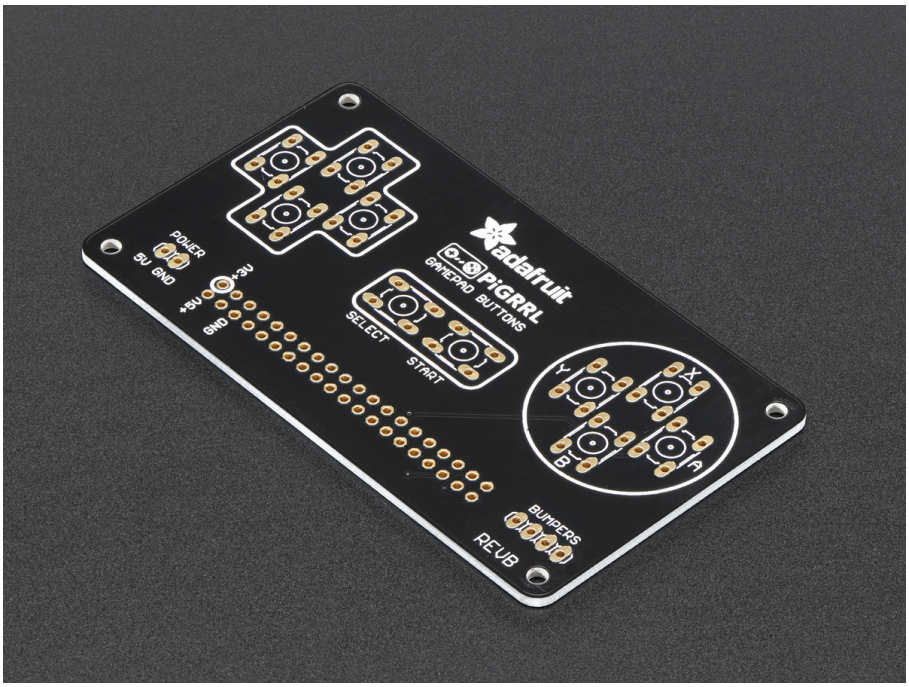


Rys. 4.7. Wyświetlacz rezystywny Adafruit PiTFT 2.8" o rozdzielczości 320x240, numer modelu P2298B.



Rys. 4.8. Ścieżka #18 znajdująca się z tyłu układu wyświetlacza, do poprawnej integracji z kontrolerem należy ją przeciąć.

Na obecną chwilę platforma nie posiada żadnego zasilania wewnętrznego, do działania układ Raspberry Pi musi być podłączone do prądu. Z powodu braków implementacyjnych zestaw dźwiękowy również został pominięty.



Rys. 4.9. Płytka PCB kontrolera wyprodukowana przez firmę Adafruit. Zaprojektowana na potrzeby projektu PiGRRL 2.0.

5. TESTY I WYNIKI - ŁUKASZ MRUGAŁA

Jak wiadomo, nawet najlepsze teorie i obliczenia nie zastąpią praktyki. Dlatego też zanalizowany i zaprojektowany system należy jeszcze zaimplementować, a implementację przetestować, by móc znaleźć problemy i usprawnić rozwiązanie.

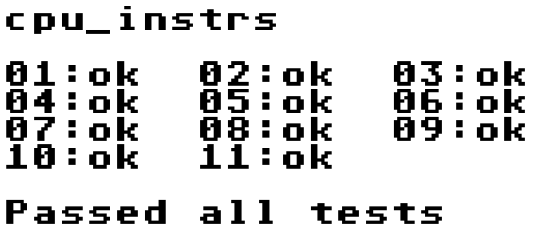
W naszym wypadku nie chcemy testować pełnej zgodności z dokumentacją, ponieważ już na etapie projektowania założyliśmy, że jej nie zachowamy, by mieć możliwość umieszczenia emulatora na przenośnej platformie komputerowej. Nadal jednak, o ile to możliwe, powinniśmy ocenić, czy komponenty, które mogą być testowane automatycznie z pewną dozą sukcesu, zgadzają się nią. Do takiej oceny używa się m. in. testowych ROMów blargga [5], których wyniki przedstawimy poniżej. Ważniejszą metryką jest dla nas wybranie pewnej liczby popularnych gier na emulowany system, które musimy na naszym emulatorze uruchomić i ręcznie sprawdzić ich zgodność. Wyniki testów zostały zamieszczone w tabeli 5.1.

Nie ma twardych zasad rozróżniania niezgodności działania ROMów na emulatorze w porównaniu z oryginalnym sprzętem. Każdy tytuł, w celu uzyskania pożądanych miar, był testowany przez pięć minut nieprzerwanej gry, włączając w to animację początkową. Jeśli przed danym testem uzyskaliśmy informację o błędzie w późniejszym etapie gry, segment testów mógł zostać podwojony. Kod emulatora, na jakim odbywało się testowanie był lekko zmodyfikowanym kodem produkcyjnym, do którego dodano możliwości mierzenia jittera oraz czasu jałowego. W tej pracy ustaliliśmy cztery miary mówiące nam o jakości emulacji danego tytułu. Są to:

- **Start** - Odpowiada na pytanie, czy gra jest w stanie się uruchomić. NIE, jeśli program emulacji się zatrzyma bądź wpadnie w nieskończoną pętlę bez udziału użytkownika. TAK w innym wypadku.
- **Różnice** - Subiektywna miara, czy emulowana gra posiada widoczne różnice w porównaniu z oryginałem, wyłączając niezaimplementowane komponenty, takie jak SOUND. N/D, jeśli wartość uruchomienia to NIE. BRAK, jeżeli nie udało się pozyskać odpowiedniego porównania (Nie zdobyliśmy oryginalnego kartridża do zaobserwowania oryginalnej gry sami ani nie znaleźliśmy nagrania pokazującego grę na oryginalnej konsoli). NIE, jeżeli nie zauważono różnic. MIN, jeżeli różnice są kosmetyczne, trudne do zauważenia, nie wpływają na grę i satysfakcję użytkownika. MAX, jeżeli różnice są znaczne,
  w szczególności gdy uniemożliwiają postęp.
- **Czas jałowy** - W kodzie testowym występuje funkcja wait_clock(), która jest wywoływana na końcu każdego cyklu instrukcyjnego, by zachować zgodność emulatora z oryginałem względem czasu przeznaczonego na instrukcję. Korzystając z narzędzia perf, możemy dowiedzieć się, ile czasu działania emulator spędził w tej funkcji. Gry działające wolno będą odwrotnie skorelowane do tej wartości procentowej. Wartości powyżej 15% zwykle
  zapewniają płynność rozgrywki.

- **Jitter (średni i maksymalny)** - W kodzie testowym wykorzystujemy funkcję clock_gettime(), by móc porównać moment wyświetlenia klatki z momentem, w którym powinna ona była zostać wyświetlona, biorąc pod uwagę zdarzenia takie jak np. wyłączenie rysowania. Fluktuacje w liczbie renderowanych klatek na sekundę zwykle oznaczają mniejszą satysfakcję z działania emulatora i niewystarczającą jego wydajność. Według naszego testowania, wartości poniżej 10 000 000 ns powinny być niezauważalne.

5.1. ROMy blargga



Rys. 5.1. Zrzut ekranu naszych wyników testów CPU blargga.

Ze względu na nasze cele projektowe, użycie wszystkich ROMów testowych blargga nie było potrzebne. Przez atomiczność naszych instrukcji wszystkie testy sprawdzające dokładność działań do cyklu musiały zakończyć się niepowodzeniem. Dlatego też użyliśmy jedynie ROMów sprawdzających poprawność implementacji instrukcji CPU (Wyniki widoczne na Rys. 5.1). Wszystkie odniosły sukces stosunkowo wcześnie w czasie tworzenia naszego emulatora.

5.2. Wyniki i ich opis

5.2.1. Opisy znalezionych błędów

Powszechnym błędem będący powodem oceny MIN w kategorii Różnice jest ukrywanie sprite'ów będących zbyt blisko lewej bądź górnej krawędzi ekranu. Błąd ten, bez żadnych dodatkowych niuansów, był zauważalny we wszystkich tytułach, w których można było przemieścić sprite blisko tych granic ekranu, ale dzięki odpowiednio wcześniejszemu testowaniu byliśmy w stanie go naprawić.

Super Mario Land 2: 6 Golden Coins po użyciu jednej z maszyn do hazardu staje się niezdatny do gry.

W grze The Legend of Zelda: Link's Awakening górna krawędź ekranu mruga na biało.

Animacja startowa w grze Donkey Kong jest błędna. Po zakończeniu danej planszy, okienko ze spisem wyników Mario nie wyświetla się.

Pokémon Pinball napotyka naruszenie ochrony pamięci po wyborze języka.

Tabela 5.1. Podsumowanie wyników testowania

Nazwa	Wydanie	Start	Różnice	Czas jałowy	Średni jitter [ns]	Maks. jitter [ns]
Tytuły GB						
Tetris	World A	TAK	NIE	42,1%	5 968 748	277 003 776
Pokémon Red	U E	TAK	NIE	37,0%	4 417 619	56 316 064
LoZ: Link's Awakening	U	TAK	MIN	41,6%	2 605 314	39 351 200
DK	World A	TAK	MIN	34,4%	4 810 907	329 703 584
SML 2: 6 Golden Coins	U E	TAK	MAX	42,4%	3 524 685	146 680 384
SML	J U E	NIE	N/D	N/D	N/D	N/D
Tytuły CGB						
DWM	U E	TAK	NIE	41,5%	5 737 374	192 731 520
Pokémon Yellow	U	TAK	NIE	36,5%	3 978 295	51 619 128
Pokémon Silver	U E	TAK	NIE	33,8%	3 508 327	88 458 136
Ghosts'n Goblins	U E	TAK	NIE	25,2%	8 943 004	45 297 184
DK Country	U E	TAK	NIE	24,3%	7 871 395	45 175 656
Shantae	U	TAK	NIE	21,6%	4 358 924	26 252 528
Pokémon TCG	USA	TAK	NIE	16,3%	9 447 700	58 297 272
Bionic Commando 2	U	TAK	NIE	16,0%	4 751 193	53 045 888
LoZ: OoA	U	TAK	NIE	15,7%	9 688 621	67 353 296
GTA 2	U	TAK	NIE	14,7%	4 330 010	173 611 520
Mario Tennis	E	TAK	NIE	14,0%	9 119 716	61 390 136
Tetris DX	World	TAK	MIN	37,6%	1 877 495	82 759 080
SMB Deluxe	U B	TAK	MIN	26,1%	8 639 787	65 271 456
Yu-Gi-Oh! 4: KD	J	TAK	MIN	17,0%	4 121 518	301 181 248
Wario Land 3	World	TAK	MAX	25,7%	8 408 619	66 151 120
Metafight EX	U E	TAK	MAX	15,1%	6 926 115	55 557 496
Pokémon Pinball	E	TAK	MAX	N/D	N/D	N/D
Kirby Tilt 'n' Tumble	U	NIE	N/D	N/D	N/D	N/D
G&WG 3	U E	NIE	N/D	N/D	N/D	N/D
Aladdin	U	NIE	N/D	N/D	N/D	N/D

Tabela 5.2. Zestawienie znaczeń skrótów użytych w tym rozdziale

Skrót	Pełny tekst
Bionic Commando 2	Bionic Commando: Elite Forces
DK	Donkey Kong
DWM	Dragon Warrior Monsters
E	Europa (w kontekście wydań gier CGB)
G&WG	Game & Watch Gallery
J	Japonia (w kontekście wydań gier CGB)
LoZ	The Legend of Zelda
OoA	Oracle of Ages
SMB	Super Mario Bros.
SML	Super Mario Land
TCG	Trading Card Game
U	Stany Zjednoczone (w kontekście wydań gier CGB)
Yu-Gi-Oh! 4 KD	Yu-Gi-Oh! Duel Monsters 4: Battle of Great Duelists: Kaiba Deck

HUD na górze ekranu gry Super Mario Bros. Deluxe zasłania elementy których nie powinien, np. samego Mario.

Yu-Gi-Oh! 4: KD nie wyświetla poprawnie lewej strony menu ataku potwora, można jednak z niego normalnie korzystać.

Użycie jednych z pobocznych drzwi występujących na poziomie 3 Wario Land 3 wywołuje krytyczny błąd programu emulacji i jego przedwczesne zamknięcie.

Przy wybieraniu typu gry Tetris DX, to jest: czy gramy Maraton, Ultra Maraton, czy inny, menu wyboru typu muzyki jest w znacznej części czarne, poza pojedynczymi sprite'ami.

Metafight EX, inaczej Blaster Master: Enemy Below, posiada specyficzny błąd - jeśli gracz wciśnie jednocześnie lewy i prawy kierunek przy kierowaniu pojazdem, pojazd obraca się do góry nogami i zanurza pod ziemię. Oprócz tego, próba wciśnięcia klawisza START skutkuje naruszeniem pamięci.

GTA 2 zarówno u nas, jak i na oryginalnej konsoli posiada problemy z liczbą obliczeń, przez co wyczuwalne jest opóźnienie działania gry, w szczególności gdy na ekranie
jest wiele ludzi i samochodów.

5.2.2. Wnioski z testów

Aktualne wyniki naszych testów stawiają przed nami cele na przyszłość. O ile wydajność emulacji jest satysfakcjonująca zarówno w miarach subiektywnych, jak i obiektywnych, niestety nie możemy pochwalić się zgodnością z oryginałem, która by nas całkowicie zadowoliła.

Jednak dzięki zauważeniu, analizie i udokumentowaniu błędów, byliśmy w stanie w trakcie rozwoju systemu zwalczyć znaczącą większość z nich. Dowiodły nam one, że dotychczasowe dokumentacje CGB nie są idealne, a nawet mała niejednoznaczność wewnątrz nich może doprowadzić do znaczących różnic między emulatorem, a konsolą.

6. INSTRUKCJA DLA UŻYTKOWNIKA - MICHAŁ ZALEWSKI

Zgodnie z postawionymi celami projektu priorytetem było stworzenie systemu łatwego w użyciu dla średnio-zaawansowanego użytkownika komputera. W związku z tym udostępniany interfejs jest klarowny i prosty w użyciu.

Korzystanie z podstawowych funkcji zarządzania katalogiem gier jest intuicyjne, a bardziej zaawansowane opcje i ustawienia są dostępne w odpowiednich menu. Interfejs emulatora jest minimalny, ogranicza się do wyświetlania ekranu gry. Sterowanie odbywa się za pomocą przycisków zbliżonych do tych z oryginalnej konsoli zarówno przy użyciu kontrolera wbudowanego, jak i zewnętrznego.

Dodawanie nowych gier wymaga prostego skopiowania ich na urządzenie zewnętrzne na komputerze użytkownika, którego użytkowanie nie powinno mu sprawiać problemów. Po stronie wytworzonego systemu operacja ta ogranicza się do podłączenia urządzenia zewnętrznego. Nie jest wymagane użycia dodatkowych opcji z poziomu interfejsu użytkownika.

6.1. Opis interfejsu użytkownika

6.1.1. Uruchomienie urządzenia

Użytkownik uruchamia urządzenie przez podłączenie do niego źródła zasilania. Na wbudowanym ekranie widzi informacje dotyczące startu systemu operacyjnego. Po zakończeniu inicjalizacji systemu automatycznie uruchamiany jest graficzny interfejs zarządzania katalogiem gier.

Podczas pierwszego uruchomienia urządzenia wyświetlone zostanie menu konfiguracji przycisków do obsługi interfejsu zarządzania katalogiem gier. Jego użycie zostało opisane w sekcji *Konfiguracja przycisków* poniżej. Po zakończeniu konfiguracji, jak również w przypadku, gdy urządzenie zostało już wcześniej skonfigurowane, zostanie wyświetlone okno startowe interfejsu (Rys. 6.1).

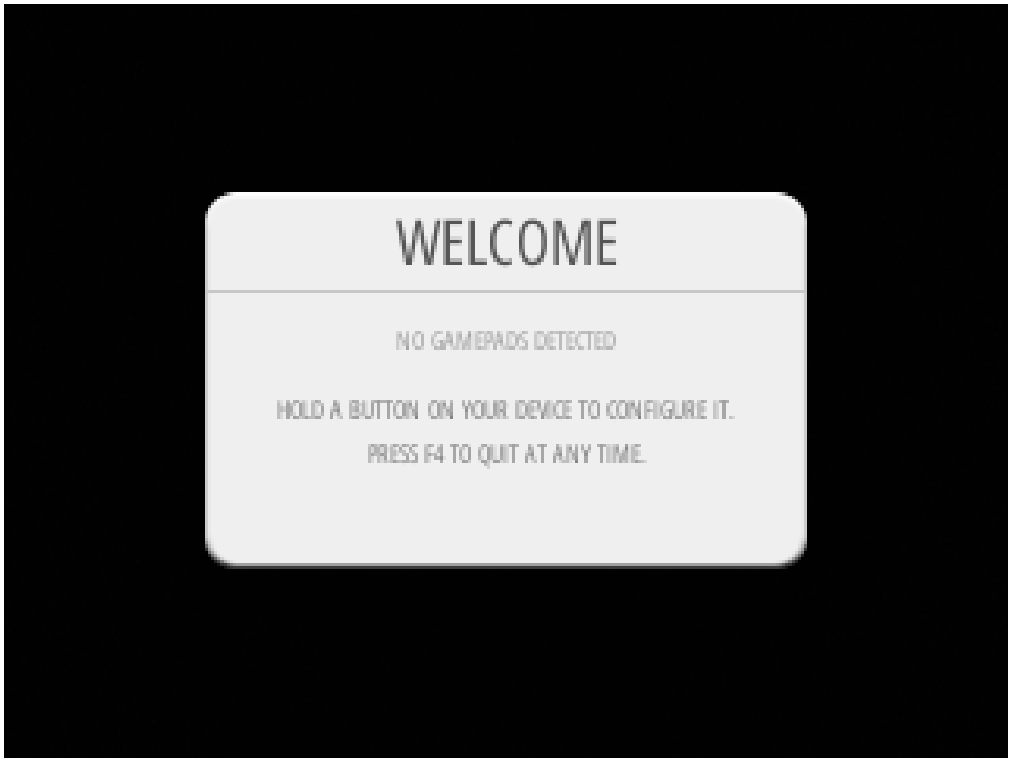
6.1.2. Konfiguracja przycisków

Opcja konfiguracji przycisków pozwala na skonfigurowanie przycisków używanych w interfejsie zarządzania katalogiem gier. Po jej włączeniu wyświetlany jest ekran przedstawiony na Rys. 6.2. Należy wtedy przytrzymać dowolny przycisk na kontrolerze, który ma zostać skonfigurowany.

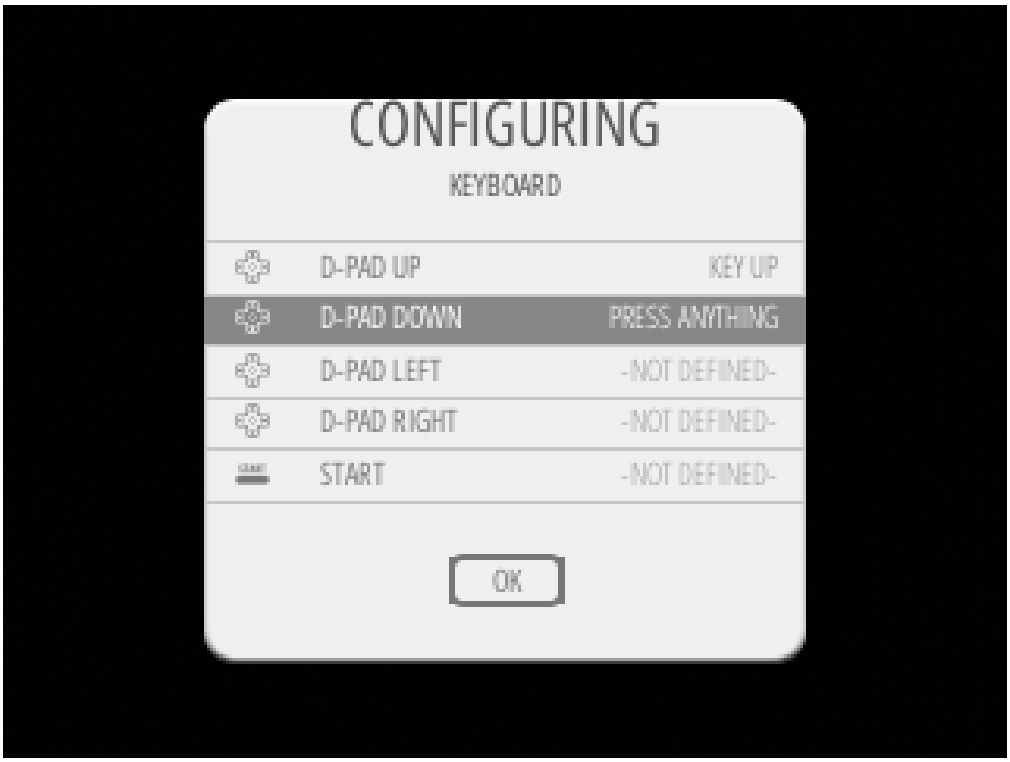
Po wybraniu kontrolera nastąpi automatyczne przejście do ekranu wyboru przycisków (Rys. 6.3) zawierającego listę przycisków. Należy wciskać kolejne przyciski na kontrolerze odpowiadające podświetlonym przyciskom na ekranie. W sytuacji, w której kontroler nie posiada odpowiadającego przycisku lub nie jest on konieczny można pominąć jego konfigurację przytrzymując dłużej dowolny przycisk. Do poprawnego działania interfejsu konieczne są przyciski: kierunkowe (D-Pad), START, SELECT, Button A (zatwierdzający), Button B (cofający).

Jeżeli nie został wybrany przycisk odpowiadający Hotkey Button pojawi się stosowny komunikat. Można w nim wybrać opcję No, gdyż funkcje uruchamiane przez ten przycisk nie są wspierane przez emulator. Po zakończeniu powinno być widoczne menu, z którego została uruchomiona konfiguracja.

Rys. 6.1. Ekran startowy interfejsu zarządzania katalogiem gier. Taki ekran widoczny jest dla użytkownika konsoli. Efekt rozmazania wynika z faktu, że zrzut ekranu został zrobiony w natywnej rozdzielczości ekranu urządzenia i przeskalowany.



Rys. 6.2. Ekran wyboru kontrolera



Rys. 6.3. Ekran konfiguracji przycisków

6.1.3. Wybór gry

W celu wybrania gry użytkownik naciska przycisk ustawiony jako przycisk A przy konfiguracji. Pokazana zostaje lista dostępnych w urządzeniu gier (Rys. 6.4). Użytkownik podświetla pożądaną pozycję na liście używając przycisków kierunkowych góra/dół. Po wybraniu gry użytkownik naciska przycisk A i uruchomiony zostaje emulator (opis użycia emulatora znajduje się w sekcji *Sterowanie podczas emulacji* poniżej). Po zakończeniu emulacji następuje powrót do listy gier.

6.1.4. Sterowanie podczas emulacji

Po uruchomieniu emulatora jedynym wyświetlanym elementem interfejsu jest ekran gry. Sterowanie może odbywać się z użyciem wbudowanego kontrolera, zewnętrznego kontrolera lub podłączonej klawiatury.

W przypadku wykorzystania kontrolera, zarówno wbudowanego (Rys. 6.5), jak i zewnętrznego, używana jest domyślna konfiguracja standardowych przycisków. Używane przyciski zostały przedstawione w tabeli 6.1. W przypadku kontrolera zewnętrznego, jeżeli nie odpowiadają one przyciskom dostępnym w danym kontrolerze oznacza to, że prawdopodobnie do ich oznaczenia użyte zostały inne symbole. Użytkownik może we własnym zakresie stwierdzić, które z dostępnych przycisków odpowiadają wykonywanym akcjom.

Rys. 6.4. Ekran listy gier pozwalający na wybranie gry do uruchomienia, usunięcia lub zmiany metadanych

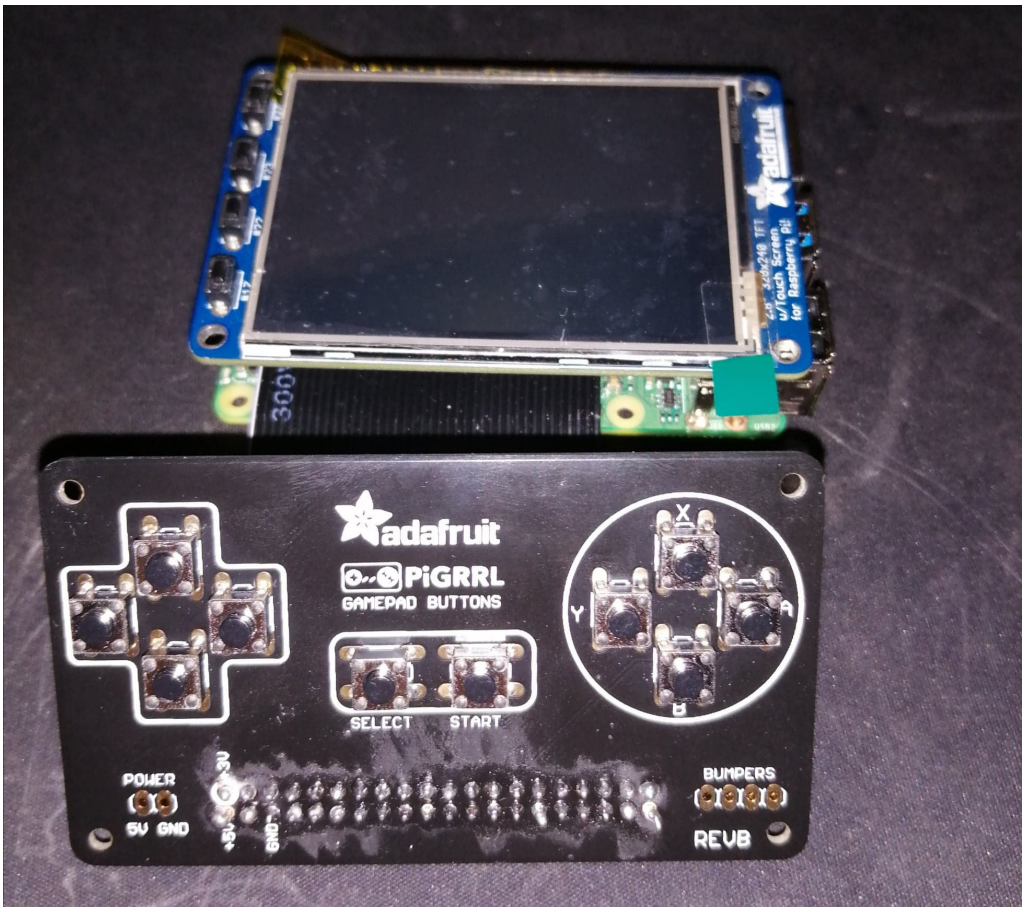
Tabela 6.1. Lista domyślnie używanych przycisków. W przypadku kontrolera zewnętrznego podane zostały przyciski dostępne w typowym kontrolerze.

Game Boy Color	Kontroler wbudowany	Kontroler zewnętrzny	Klawiatura
↑ (D-Pad)	↑ (D-Pad)	↑ (D-Pad)	↑ (przycisk kierunkowy)
↓ (D-Pad)	↓ (D-Pad)	↓ (D-Pad)	↓ (przycisk kierunkowy)
← (D-Pad)	← (D-Pad)	← (D-Pad)	← (przycisk kierunkowy)
→ (D-Pad)	→ (D-Pad)	→ (D-Pad)	→ (przycisk kierunkowy)
A	A lub ×	A lub ×	Z
B	B lub ◯	B lub ◯	X
START	START	START	ENTER
SELECT	SELECT	SELECT	Spacja
wyłączenie konsoli	Przycisk 1 przy ekranie	X lub □	Q

6.1.5. Dodawanie gier

Dodawanie gier odbywa się bez użycia interfejsu graficznego. W celu dodania nowych pozycji do kolekcji gier użytkownik powinien umieścić powiązane pliki w katalogu nadrzędnym pamięci USB. Istotne jest, żeby pamięć używała systemu plików FAT32 lub exFAT oraz aby pliki miały jedno z rozszerzeń: *.gb, .gbc, .GB, .GBC*. W przeciwnym wypadku
nie będzie możliwe dodanie nowych gier.

Interfejs zarządzania katalogiem gier nie odświeża automatycznie listy gier. Aby załadować do niego nowe pozycje, należy go zrestartować. W tym celu trzeba wcisnąć ustawiony wcześniej przycisk START, z widocznego menu wybrać za pomocą przycisków kierunkowych góra/dół wybrać pozycję *QUIT* i zatwierdzić ustawionym przyciskiem A. Z kolejnego menu należy wybrać opcję *RESTART EMULATIONSTATION* i ponownie zatwierdzić. Po tej operacji nowo dodane gry powinny być dostępne na liście.



Rys. 6.5. Wygląd prototypu wbudowanego kontrolera. Widoczne są oznaczenia przycisków.

6.1.6. Usuwanie gry

W celu usunięcia pozycji z listy dostępnych gier należy podświetlić tę pozycję używając przycisków kierunkowych góra/dół. Następnie wymagane jest wciśnięcie przycisku SELECT. W wyświetlonym menu trzeba wybrać opcję *EDIT THIS GAME'S METADATA*, zatwierdzić ją przyciskiem A i w kolejnym menu za pomocą przycisków kierunkowych dojść na sam koniec listy opcji. Po wciśnięciu przycisku dół powinien zostać podświetlony jeden z przycisków w dolnej części menu. Za pomocą przycisków kierunkowych należy wybrać opcję *DELETE* (Rys. 6.6) i zatwierdzić ją używając przycisku A. Wyświetlony zostanie dodatkowy komunikat, w którym użytkownik potwierdza wolę usunięcia wybierając opcję *YES*. Po wyjściu z menu z użyciem przycisku B użytkownik wraca do listy gier, na której nie ma już usuniętej pozycji.

6.1.7. Sytuacje szczególne

Podczas usuwania gier lub przy początkowym wdrożeniu systemu na platformę może zdarzyć się sytuacja, w której nie będzie dostępna żadna gra. Wtedy interfejs zarządzania katalogiem gier wyświetli komunikat informujący o tej sytuacji i nie będzie możliwe włączenie go. Użytkownik powinien dodać gry do katalogu zgodnie z opisem z sekcji *Dodawanie gier* i uruchomić ponownie urządzenie odłączając od niego zasilanie i powtórnie podłączając je. Po uruchomieniu systemu użytkownik będzie mógł normalnie korzystać z interfejsu.

Rys. 6.6. Menu edycji metadanych gry w katalogu. Z jego poziomu możliwe jest usunięcie gry z listy z użyciem zaznaczonej opcji *DELETE*

6.2. Wdrożenie systemu

Poniższa sekcja przedstawia sposób, w jaki użytkownik może wdrożyć system na własnej platformie z użyciem przygotowanych konfiguracji i narzędzi.

6.2.1. Instrukcja wdrożenia

Aby wdrożyć komponenty programowe systemu na platformie Raspberry Pi 4 przede wszystkim należy wgrać odpowiedni system operacyjny na kartę SD, z której uruchamiane jest oprogramowanie komputera. W ramach projektu używany jest system Raspberry Pi OS Lite w wersji z 20.08.2020 r. Instalacja systemu jest opisana na stronie producenta. [9]

Po zainstalowaniu systemu operacyjnego należy umieścić w komputerze przygotowaną kartę SD oraz podłączyć klawiaturę, monitor i kabel sieciowy. Następnie użytkownik powinien podłączyć zasilanie, co spowoduje uruchomienie systemu operacyjnego. Po niedługim czasie wyświetlony zostanie formularz logowania. Należy zalogować się korzystając z loginu *pi* i hasła *raspberry*.

Po zalogowaniu użytkownik powinien uruchomić narzędzie konfiguracyjne platformy poleceniem:

```
sudo raspi-config
```

Używając wyświetlonego menu można dostosować ustawienia klawiatury do używanego układu (*Localisation* → *Keyboard*). Konieczne jest ustawienie automatycznego logowania przez wybranie w menu *Boot* → *Desktop/CLI* opcji *Console Autologin*. Następnie można zakończyć działanie narzędzia używając opcji *<Finish>*. Nie jest konieczne restartowanie systemu.

Po skonfigurowaniu podstawowych opcji wymagane jest włączenie serwera SSH, aby umożliwić wgranie wymaganych plików na platformę. W tym celu używane jest polecenie:

```
sudo systemctl start ssh
```

Następnie można sprawdzić adres IP, pod którym dostępna jest platforma w lokalnej sieci (polecenie *ip address*). Znając adres urządzenia można przesłać pliki systemu w formie archiwum ZIP z innego komputera na platformę docelową korzystając z narzędzia SCP. Z komputera roboczego należy wykonać polecenie:

```
scp gbc.zip pi@<adres IP platformy>:/home/pi
```

gdzie *gbc.zip* to archiwum z plikami systemu. Wymagane będzie podanie hasła użytkownika *pi*: *raspberry*.

Następnie na platformie wdrożeniowej można uruchomić przygotowany skrypt wdrożeniowy, który zainstaluje wymagane zależności, umieści odpowiednie pliki konfiguracyjne i skompiluje elementy systemu. Dokładniejsze omówienie działania skryptu znajduje się w sekcji *Kroki wdrożenia* poniżej. W celu wykonania skryptu należy wpisać następujące polecenia:

```
unzip gbc.zip
cd gbc/deploy
sudo ./deploy.sh
```

Jeżeli w trakcie wykonania skryptu nie pojawiły się informacje o błędach, oznacza to, że przygotowanie systemu przebiegło pomyślnie. Można zresetować urządzenie i po ponownym uruchomieniu widoczny będzie standardowy interfejs użytkownika. W celu zresetowania urządzenia używana jest instrukcja:

```
reboot
```

6.2.2. Kroki wdrożenia

W czasie wykonania skryptu wdrożeniowego *deploy.sh* wszystkie komponenty systemu i ich zależności są przygotowywane do działania na platformie. Można wyróżnić kolejne kroki przygotowania:

1. aktualizacja systemu operacyjnego – za pomocą menedżera pakietów *apt* aktualizowane są wszystkie pakiety programowe w systemie.
2. instalacja wymaganych pakietów – za pomocą menedżera pakietów *apt* instalowane są pakiety potrzebne do kompilacji interfejsu EmulationStation, biblioteka SDL2 wraz z plikami nagłówkowymi wymagana do kompilacji emulatora, menedżer okien Openbox wraz z serwerem wyświetlania X11, system kontroli wersji *git* potrzebny do pobrania EmulationStation oraz program *usbmount* odpowiadający za automatyczne montowanie podłączanych nośników pamięci.
3. kopiowanie konfiguracji – z dostarczonych zasobów kopiowane są pliki konfiguracyjne do serwera X11 i powłoki systemowej uruchamiające automatycznie menedżer okien i EmulationStation, plik konfiguracyjny i motyw graficzny do EmulationStation oraz skrypty do automatycznego kopiowania plików po podłączeniu nośnika pamięci. Dodatkowo kopiowana jest zasada dla menedżera urządzeń *udev*, która uruchamia skrypty do kopiowania po podłączeniu nośnika.
4. kompilacja emulatora – program emulatora jest automatycznie kompilowany z użyciem narzędzia *GNU Make*.
5. kompilacja EmulationStation – program używany w roli interfejsu graficznego jest pobierany z repozytorium z użyciem systemu *git* i kompilowany.

7. PODSUMOWANIE

7.1. Napotkane problemy - Łukasz Mrugała

Głównymi problemami, z jakimi mierzyliśmy się w trakcie tworzenia naszego emulatora były błędy w module GPU oraz niewystarczająca wydajność systemu. Towarzyszyły nam przez praktycznie cały proces jego pisania.

Rys. 7.1. Przykładowy błąd GPU - niepoprawne wartości początkowe rejestru STAT.

Wiedzieliśmy, że wydajność będzie problemem już w trakcie analizy, gdyż chcieliśmy umieścić emulator na małej platformie komputerowej. Z tego względu musieliśmy optymalizować działające poprawnie sekcje kodu w celu uzyskania tych samych wyników przy mniejszych nakładach czasowych. W praktyce trzy zmiany przyniosły nam największe korzyści.

Pierwszą z nich było wydzielanie modułów, które w trakcie testów zostały zidentyfikowane jako zajmujące dużo cykli do osobnych wątków. Drugą - zmiana oryginalnej biblioteki odpowiadającej za wejście programowe, dźwięk i grafikę, Allegro 5, na SDL 2. Ostatnią zaś - implementacja leniwego ładowania danych wewnątrz GPU.

Dopiero w trakcie implementacji zorientowaliśmy się, że GPU jest modułem sprawiającym najwięcej problemów. Nie bez powodu wcześniej w pracy komponent emulacji grafiki został nazwany najbardziej skomplikowanym. Częściowo wynikało to z dokumentacji, która nie była tak dokładna, jak w wypadku części niezmienionych od DMG.

Dużo stosunkowo małych różnic względem oryginału w tym module, wynikających z kwestii nie wspomnianych w dokumentacji bądź wtrąconych mimochodem mogło skutkować doświadczeniem użytkownika analogicznym do emulatora, który nie miał zaimplementowanego procesora graficznego wcale. Przykład błędu GPU został przedstawiony na Rys. 7.1, gdzie niepoprawna początkowa wartość rejestrów poskutkowała obrazem wyjściowym nieprzydatnym dla użytkownika. Jednocześnie wiele z błędów GPU było stosunkowo łatwe do zauważenia i testowania, dzięki czemu byliśmy w stanie szybko im zaradzić.

Oprócz tych dwóch głównych nurtów problemów w trakcie tworzenia naszego systemu, należałoby wspomnieć też o trudności w pozyskaniu fizycznych zewnętrznych komponentów systemu, takich jak np. kontroler dla guzików. Zamawianie wielu z istniejących produktów wiązałoby się z długim czasem transportu i małymi liczbami zasobów w magazynach. Brak łatwo dostępnych komponentów utrudniał końcowe fazy testowania, gdy emulator winien już być umieszczony na sprzęcie docelowym.

Ostatnim, wciąż nierozwiązanym problemem, jest synchronizacja programu na procesorach architektury arm64. Jest ona powiązana z poprzednim - trudny dostęp do sprzętu docelowego ograniczył nam możliwości dopracowania rozwiązania.

7.2. Sukcesy i porażki - Adrian Misiak

Ostatnie poprawki do systemu powstawały jeszcze w trakcie pisania tej pracy. Teraz jednak dobiegły końca, więc jest to odpowiedni moment na podsumowanie projektu i opisanie kwestii, które się powiodły oraz tych, które zostały pominięte lub ostatecznie okazały się niedziałające.

7.2.1. Sukcesy

Zaczynając od pozytywnych aspektów: udało się napisać emulator, który działa dla większej części próbowanych gier, z czego gry dobierane do testów należą do najbardziej popularnych. Emulator, poza komputerem stacjonarnym, udało się też umieścić na układzie jednopłytkowym z małym wyświetlaczem i kontrolerem, przez co całość jest rozmiarowo podobna do oryginalnej konsoli. Udało się też napisać instrukcję wdrożeniową zawierającą wszystko, co jest potrzebne do przygotowania systemu emulacyjnego. Przez cały okres prac rozwojowych nad emulatorem udało nam się trzymać reguł ustalonych przed rozpoczęciem prac, a każdy fragment kodu przechodził przegląd przez pozostałych członków zespołu, dzięki czemu nie zdarzyła się sytuacja, aby na głównej gałęzi repozytorium znalazła się duża regresja.

7.2.2. Porażki

Projektu nie można uznać za skończony z kilku powodów. Brakuje w nim dźwięku, zarówno po stronie programowej, jak i sprzętowej. Jak wcześniej zostało wspomniane emulator działa dla wielu gier. Istnieje jednak grupa gier, które nadal nie działają. Wiele z nich zatrzymuje się tuż po włączeniu. W małym stopniu jest to spowodowane brakiem implementacji wszystkich MBC, ponieważ istnieje stosunkowo mała ilość gier wykorzystujące MBC niewspierane przez nasz emulator. Głównym powodem niepoprawnego działania są błędy implementacyjne w module GPU. Sprzętowa część projektu posiada też spore braki: brak własnego zasilania, brak obudowy i nakładek na przyciski. Interfejs wyboru gier na małym wyświetlaczu traci na czytelności.

7.3. Plany rozwojowe - Adrian Misiak

Najbliższy czas powinien być poświęcony na próbę naprawienia błędów powodujących wadliwe działanie niektórych gier. Z dotychczasowej analizy doszliśmy do wniosku, że przyczyną najprawdopodobniej jest niewłaściwy stan, w jakim znajduje się GPU w pierwszych fazach ładowania gry. Implementacja brakujących kontrolerów banków pamięci jest sprawą drugorzędną, ponieważ, jak już zostało wcześniej wspomniane, istnieje mała ilość gier wykorzystujących te kontrolery.

Posiadając już stabilny emulator po poprawie znalezionych błędów, następną rzeczą w planach jest implementacja dźwięku. Już na dość wczesnym etapie projektu doszliśmy do wniosku, że prawdopodobnie dźwięk nie zostanie zaimplementowany w ramach tej pracy inżynierskiej. Tworzenie dźwięku na konsoli odbywa się na bardzo niskim poziomie i wymaga dodatkowego przestudiowania tematu przed próbą implementacji.

Po implementacji wszystkich funkcjonalności otwierają się dwie drogi rozwojowe. Pierwsza polega na zoptymalizowania działania emulatora w jak największy sposób, aby zapewnić działanie na słabszych układach. Druga to próba dojścia do jak największej precyzji emulacji. Wiązałaby się z pracami głównie nad GPU i CPU aby przejść na dokładność co do cyklu, próbując przy tym utrzymać obecną wydajność. Obie ścieżki oferują ciekawe problemy do rozważenia w przyszłości.

Część sprzętowa też wymaga pewnych prac. Główną jej rolą w tym projekcie było pokazanie działania naszego emulatora. Początkowo planowaliśmy, aby podział czasu prac nad częścią programową i sprzętową trwał tyle samo. Pracę nad samym programem się przedłużyły, co skutkowało skróceniem czasu, jaki byliśmy w stanie poświęcić na drugą część. Najważniejszymi aspektami, którymi należy się zająć jest zapewnienie własnego zasilania do układu, co zapewniłoby większą jego przenośność, oraz umieszczenie go we własnej obudowie.

7.4. Wnioski - Michał Zalewski

Przy implementacji emulatora bardzo istotne jest dokładne odtwarzanie zachowań obiektu emulowanego. Dokładne zbadanie konsoli GBC pod kątem wszystkich zachowań wymagających odtworzenia byłoby wymagającym przedsięwzięciem. Konieczny nakład pracy zdecydowanie wykraczałby poza oczekiwania od projektu inżynierskiego. Aktywnie działająca społeczność entuzjastów i wytworzone przez nią dokumenty przeznaczone dla twórców gier i emulatorów pozwoliły na skorzystanie z istniejącej wiedzy i doświadczenia. Bez tego nie byłoby możliwe ukończenie projektu w wymaganym terminie lub wcale, ze względu na brak wyposażenia i umiejętności koniecznych do tak szczegółowej inżynierii odwrotnej.

Przy korzystaniu z nieoficjalnej dokumentacji istnieje jednak ryzyko, że nie będzie ona dokładna, lub zawierać będzie pewne braki. Niektóre informacje, w szczególności te na temat funkcji dodanych w GBC względem poprzedniej konsoli GB, w używanej dokumentacji były wybrakowane. Ze względu na to musiały być czynione teoretyczne założenia co do działania konsoli, które były później weryfikowane przy testowaniu z użyciem oryginalnych gier. W ten sposób wprowadzone zostały błędy, które, mimo narzędzi wspomagających ich detekcję, dalej wymagały pracy do znalezienia i naprawienia.

Język C został wybrany ze względu na brak nadmiarowych abstrakcyjnych konstrukcji i duże możliwości optymalizacji działania programu na niskim poziomie. Z uwagi na to, że jest językiem w paradygmacie proceduralnym nie posiada mechanizmów wymuszających porządkowanie struktury implementacji. Dzięki wczesnemu przyjęciu określonych standardów użycia języka i podziału funkcjonalności na moduły, udało się jednak stworzyć dość przejrzysty kod, który nie sprawia dużych trudności w rozszerzaniu i analizowaniu w poszukiwaniu błędów.

WYKAZ LITERATURY

[1] Nintendo Co., Ltd., *Consolidated Sales Transition by Region* , https://web.archive.org/web/20160427084600if_/https://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1603.pdf
(dostęp 29.11.2020)

[2] Kuchera B., *Accuracy takes power: one man's 3GHz quest to build a perfect SNES emulator* , https://arstechnica.com/gaming/2011/08/accuracy-takes-power-one-mans-3ghz-quest-to-build-a-perfect-snes-emulator/ (dostęp 26.11.2020)

[3] Emulation General wiki, *Game Boy/Game Boy Color emulators* https://emulation.gametechwiki.com/index.php/Game_Boy/Game_Boy_Color_emulators (dostęp 22.11.2020)

[4] Kristoffer Almroth Anton le Clercq, *Comparison of Rendering Performance Between Multimedia Libraries Allegro, SDL and SFML*, Stockholm, Sweden 2019

[5] Shay Green, *Blargg's Gameboy hardware test ROMs* https://github.com/retrio/gb-test-roms (dostęp 24.06.2020)

[6] *Linux kernel coding style*, https://www.kernel.org/doc/html/v4.10/process/coding-style.html (dostęp 24.11.2020)

[7] Adafruit Industries, LLC, *Raspberry-Pi-Installer-Scripts*, https://github.com/adafruit/Raspberry-Pi-Installer-Scripts (dostęp 24.11.2020)

[8] Adafruit Industries, LLC, *Adafruit PiTFT 2.8" Touchscreen*, https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/backlight-control, (dostęp 26.11.2020)

[9] Raspberry Pi Foundation, *Installing operating system images*, https://www.raspberrypi.org/documentation/installation/installing-images/ (dostęp 27.11.2020)

[10] Díaz Antonio Niño (AntonioND), *The Cycle-Accurate Game Boy Docs*, https://raw.githubusercontent.com/AntonioND/giibiiadvance/master/docs/TCAGBD.pdf 2015 (dostęp 13.06.2020)

[11] Pan of Anthrox et al., *Pan Docs*, https://github.com/gbdev/pandocs 1995-2020 (dostęp 29.02.2020)

[12] Pan of Anthrox, GABY, Marat Fayzullin, Pascal Felber, Paul Robson, Martin Korth, kOOPa, Bowser, *Game Boy(TM) CPU Manual*, http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf 1999 (dostęp 29.02.2020)

[13] Autor nieznany, *GameBoy Opcode Summary*, http://www.devrs.com/gb/files/opcodes.html 1995-2020 (dostęp 29.02.2020)

[14] Autor nieznany, *Gameboy CPU (LR35902) instruction set*, https://www.pastraiser.com/cpu/gameboy/gameboy_opcodes.html 1995-2020 (dostęp 29.02.2020)

[15] Bryan Maddock, *Handheld Game Consoles*, https://www.dimensions.com/collection/handheld-game-consoles (dostęp 08.03.2020)

WYKAZ RYSUNKÓW

WYKAZ TABEL