**ZACHODNIOPOMORSKI UNIWERSYTET**

**TECHNOLOGICZNY W SZCZECINIE**

WYDZIAŁ ELEKTRYCZNY



**Daniel Grzegorz Szot**

nr albumu: 16118

**Projekt i realizacja odtwarzacza audio formatów bezstratnych do współpracy z układem graficznym FPGA**

Praca dyplomowa inżynierska

kierunek: Automatyka i Robotyka

specjalność: Automatyzacja Procesów Przemysłowych

Opiekun pracy:

**dr inż. Krzysztof Penkala**  
Katedra Inżynierii Systemów, Sygnałów i Elektroniki  
Wydział Elektryczny

Szczecin 2013

**Oświadczenie autora**

Oświadczam, że przedkładaną pracę dyplomową pt. „Projekt i realizacja odtwarzacza audio formatów bezstratnych do współpracy z układem graficznym FPGA” napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zleciłem opracowania pracy lub jej części innym osobom oraz nie przypisałem sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego.

Załączona wersja elektroniczna pracy dyplomowej jest w pełni zgodna z wersją drukowaną.

Data:

Podpis autora

**Streszczenie pracy**

Streszczenie po Polsku.

**Słowa kluczowe**

RTOS, PIC32 itd.

**Streszczenie pracy**

Streszczenie po Polsku.

**Słowa kluczowe**

RTOS, PIC32 itd.

[Wprowadzenie 5](#_Toc347422911)

[1 Zastosowane technologie 5](#_Toc347422912)

[1.1 Hardware 5](#_Toc347422913)

[1.1.1 Procesor 5](#_Toc347422914)

[1.1.2 Płyta ewaluacyjna 5](#_Toc347422915)

[1.1.3 Układ graficzny 5](#_Toc347422916)

[1.2 Software 5](#_Toc347422917)

[1.2.1 Język programowania 5](#_Toc347422918)

[1.2.2 System operacyjny 5](#_Toc347422919)

[1.2.2.1 System operacyjny jako menadżer dostępu do zasobów 5](#_Toc347422920)

[1.2.2.2 System operacyjny jako maszyna wirtualna 5](#_Toc347422921)

[1.2.3 System plików 5](#_Toc347422922)

[1.2.4 Biblioteka graficzna 5](#_Toc347422923)

[2 Architektura systemu 5](#_Toc347422924)

[2.1 Struktura katalogów 5](#_Toc347422925)

[2.2 FreeRTOS 5](#_Toc347422926)

[2.3 Warstwa abstrakcji sprzętu 5](#_Toc347422927)

[2.3.1 Board init 5](#_Toc347422928)

[2.3.2 HLD 5](#_Toc347422929)

[2.3.3 LLD 5](#_Toc347422930)

[2.3.3.1 UART 5](#_Toc347422931)

[2.3.3.2 ADC 5](#_Toc347422932)

[2.3.3.3 IR 5](#_Toc347422933)

[2.3.3.4 AUDIO 5](#_Toc347422934)

[2.3.3.5 SDMMC 5](#_Toc347422935)

[2.3.3.6 LCD 5](#_Toc347422936)

[2.4 Biblioteki 5](#_Toc347422937)

[Bibliografia 5](#_Toc347422938)

# Wprowadzenie

Tematem pracy jest opracowanie projektu jak i realizacja odtwarzacza audio przez co rozumie się napisanie oprogramowania oraz dobór platformy sprzętowej w postaci procesora i płyty ewaluacyjnej posiadającej odpowiednie peryferia na której będzie można przeprowadzić pierwsze testy funkcjonalne.

Do przygotowania oprogramowania spełniającego założone zadanie można podejść w dwojaki sposób. Pierwszy ze sposobów jest znacznie prostszy. Ogranicza się on do napisania kodu źródłowego działającego tylko i wyłącznie na wybranej platformie sprzętowej, używającego jedynie przerwań jako namiastkę programowania wielowątkowego i spełniającego jedynie jedną założoną funkcję – w tym przypadku odtwarzanie plików audio. W ten sposób omijamy wiele pracy potrzebnej aby napisane oprogramowanie było uniwersalne oraz wygodne do rozszerzenia lub całkowitej zmiany funkcjonalności urządzenia. Rozwiązanie to nie jest złe w przypadku gdy projekt jest stosunkowo prosty a zapotrzebowanie na przyszły rozwój znikome. W przypadku gdy chcemy stworzyć uniwersalną platformę, na której odtwarzacz audio będzie jedynie aplikacją demonstracyjną a możliwości rozszerzania oraz zmiany funkcjonalności będą w zasadzie nieograniczone – będziemy musieli zastosować sposób drugi. Polega on na stworzeniu oprogramowania bazującego na jednym z wybranych, systemów operacyjnych czasu rzeczywistego, przygotowaniu warstwy abstrakcji sprzętu w celu umożliwienia łatwego przeniesienia oprogramowania na zupełnie inną platformę sprzętową, oraz napisaniu odpowiednich bibliotek ułatwiających przygotowywanie konkretnych aplikacji. Wymienione czynności prowadzą w efekcie do stworzenia prostego uniwersalnego systemu operacyjnego wyłączając z tego niskopoziomowe zarządzanie przestrzenią adresową, zarządzanie wątkami oraz komunikację międzyprocesową (IPC[[1]](#footnote-1)) – funkcje te zostaną dostarczone przez wybrany system operacyjny czasu rzeczywistego (RTOS[[2]](#footnote-2)).

Niniejsza praca traktować będzie o procesie przygotowywania takiego oprogramowania od stworzenia pierwszego sterownika urządzenia, przez napisanie biblioteki do obsługi odtwarzania plików audio, aż do przygotowania aplikacji widocznej dla użytkownika.

?????????/MOŻE COS O ANGIELSKICH KOMENTARZACH???????

?I ZE KOD JEST NA GOOGLE CODE WRZUCONY?

**Cel pracy**

Celem pracy było Napisac o FPGA koniecznie.

**Zakres pracy**

asdasd

ROZDZIAŁ 1

# Zastosowane technologie

Proces budowania uniwersalnego oprogramowania pełniącego funkcję prostego systemu operacyjnego wymaga przygotowania zarówno od strony sprzętowej jak i programowej. Rozdział ten zostanie poświęcony opisowi procesu przygotowań do rozpoczęcia prac nad projektem. Zostaną w nim przedstawione dylematy na które natrafiłem zarówno przy wyborze pierwszej platformy sprzętowej na której oprogramowanie będzie rozwijane i testowane jak i przy podejmowaniu decyzji jaki język programowania wykorzystać oraz które biblioteki użyć gotowe, napisane przez ruch otwartego oprogramowania [1] a które przygotować samodzielnie.

## Hardware

Pomysł stworzenia niniejszej pracy dyplomowej powstał w momencie gdy kończyłem jeden z projektów w pracy zawodowej. Był to prosty układ oparty na procesorze Microchip PIC32MX460F512L, posiadający kolorowy wyświetlacz o przekątnej 1,8 cala i rozdzielczości 128x160 oraz prosty przetwornik cyfrowo analogowy wykorzystujący modulację szerokości impulsów (PWM[[3]](#footnote-3)). Układ ten miał zostać zastosowany w zabawce więc jakość odtwarzanego dźwięku nie musiała być wysoka. Urządzenie to było bazą dla kilku innych modeli podobnych zabawek. Oprogramowanie jednak wymagało modyfikacji pomiędzy każdą z nich a ponieważ zostało ono przygotowane pod konkretne rozwiązanie (czyli w sposób pierwszy opisany we wprowadzeniu) modyfikacje te były pracochłonne. Wtedy powstał pomysł stworzenia oprogramowania bazowego dzięki któremu będzie można bardzo szybko tworzyć projekty tego typu. Dodatkowym wymaganiem jakie zostało postawione dla nowego oprogramowania była wysoka wydajność w aplikacjach multimedialnych. Założenia te pozwoliły ustalić, iż platforma sprzętowa na której będzie rozwijane tworzone oprogramowanie musi posiadać:

1. Wydajny procesor
2. Wyświetlacz graficzny
3. Przetwornik cyfrowo analogowy
4. Wejście na kartę pamięci SD
5. Port RS232 (pomocny przy debugowaniu)
6. Odbiornik podczerwieni (sterowanie przy pomocy pilota)

Podrozdział ten przedstawiał będzie zastosowaną w pracy platformę oraz uzasadniał wykorzystanie wybranych elementów.

### Procesor

Wybór sprzętu na którym będziemy pracować warto rozpocząć od serca układu czyli procesora. W dzisiejszych czasach różnica między procesorem a mikrokontrolerem dąży do zatarcia ponieważ właściwie każdy układ scalony który nazywamy procesorem jest w istocie mikrokontrolerem. Popularny staje się termin SoC[[4]](#footnote-4) zarezerwowany dla procesorów dedykowanych do konkretnych rozwiązań np. odbiorników telewizji satelitarnej. Każdy z procesorów posiada przynajmniej kilka urządzeń peryferyjnych co pozwala sklasyfikować je jako mikrokontrolery. Dlatego w pracy terminy te będą używane zamiennie.

Wymagania które postawiliśmy sobie dla całej platformy sprzętowej implikują kryteria jakie powinniśmy wziąć pod uwagę przy wyborze procesora. Założyliśmy, że układ ma posiadać wysoką wydajność w aplikacjach multimedialnych. Nie chodzi tutaj oczywiście o stworzenie możliwości odtwarzania filmów w wysokiej rozdzielczości ani o możliwość uruchamiania trójwymiarowych gier. Pod pojęciem tym mam na myśli:

1. Przystosowanie układu do wyświetlania graficznego interfejsu użytkownika (GUI[[5]](#footnote-5)) oraz możliwie szybkiej reakcji tego interfejsu na polecenia ze strony użytkownika.
2. Możliwość odtwarzania plików audio co najmniej w tak zwanej jakości „CD”[[6]](#footnote-6)

Aby możliwe było spełnienie pierwszego założenia konieczne jest zapewnienie możliwości szybkiej komunikacji bezpośrednio z wyświetlaczem lub z układem graficznym. Obecnie na rynku możemy nabyć zarówno wyświetlacze bez wbudowanego układu graficznego, sterowane sygnałami RGB, oraz sygnałami synchronizacji pionowej i poziomej jak i wyświetlacze z wbudowanymi kontrolerami do których polecenia wysyłane są przez interfejs komunikacji równoległej. Jako że tematem pracy była realizacja odtwarzacza audio do współpracy z układem graficznym będziemy musieli zastosować procesor posiadający możliwość szybkiej komunikacji równoległej z innym układem scalonym. Oczywiście każdy z dostępnych na rynku mikrokontrolerów ma taką możliwość. Dla niektórych z nich będzie konieczna symulacja programowa sygnałów zapisu i odczytu oraz zegara a niektóre z nich mają wbudowane sprzętowe urządzenia peryferyjne które wspierają komunikację równoległą. Przykładem może być moduł transmisji równoległej (PMP[[7]](#footnote-7)) instalowany w procesorach firmy „Microchip”. Zastosowanie procesora posiadającego taki moduł wpłynie korzystnie zarówno na wydajność jak i prostotę implementacji protokołu komunikacji.

Spełnienie drugiego założenia będzie wymagało wybrania mikrokontrolera który posiada moduły komunikacji szeregowej I2C[[8]](#footnote-8) oraz SPI[[9]](#footnote-9) – są one wymagane do komunikacji z przetwornikiem cyfrowo analogowym. Podobnie jak poprzednio ich funkcjonalność można emulować programowo jednak wpływa to niekorzystnie na wydajność.

Istnieje wiele procesorów spełniających przedstawione założenia. Dzięki pracy zawodowej udało mi się poznać dość szczegółowo architekturę procesorów firmy „Microchip” i na wybór procesora tego producenta się zdecydowałem. Znajomość architektury nie była jedynym argumentem przemawiającym za tym wyborem. Procesory firmy „Microchip” posiadają bardzo dobrą, szczegółową i pozbawioną błędów dokumentację techniczną oraz są bardzo łatwo dostępne. Podczas realizacji projektu zabawki (patrz rozdział 1.1 na stronie 5) okazało się, że procesory 8-bitowe są zbyt wolne do zastosowań multimedialnych. Pierwszy prototyp tego urządzenia został zaprojektowany w oparciu o procesor PIC18. Jednak w trakcie tworzenia oprogramowania, po pierwszych testach uznaliśmy, że zastosowany został zbyt wolny procesor – wymusiło to zmianę na 32-bitową rodzinę procesorów PIC32. Został zastosowany model PIC32MX460F512L mimo tego, że okazał się wystarczający miał jeszcze pewne mankamenty. Kontroler DMA w nim zainstalowany pozwalał na zlecenie kopiowania jednorazowo maksymalnie 256 bajtów pamięci co przy rozmiarze jednostki alokacji dla karty SD o wielkości 512 bajtów wymaga zlecenia kopiowania dwa razy na jednostkę. Kolejnym mankamentem była znacznie ograniczona ilość pamięci operacyjnej, jest to 32 kB w przypadku tego modelu.

Aby uniknąć wymienionych problemów zdecydowałem się na procesor **PIC32MX795F512L** posiadający 128 kB pamięci operacyjnej oraz możliwość kopiowania przez kontroler DMA maksymalnie 64 kB danych. Procesor ten posiada zarówno moduł PMP jak i moduły I2C i SPI.

### Płyta ewaluacyjna

Pierwszą czynnością po wyborze procesora było znalezienie odpowiedniej płyty ewaluacyjnej spełniającej założenia dotyczące zainstalowanych na niej peryferii. Oczywiście można było wykonać własny prototyp płyty drukowanej idealnie dopasowany do potrzeb pracy, jednak nie było to głównym jej celem. Dlatego zdecydowałem się użyć gotowego rozwiązania.

Mimo tego że firma „Microchip” udostępnia wiele narzędzi deweloperskich dla swoich procesorów to jednak nie udało mi się znaleźć jednej płyty spełniającej postawione wymagania. Korzystając z produktów producenta procesora konieczne byłoby zakupienie trzech płyt i połączenie ich ze sobą (posiadają kompatybilne złącza). Wszystkie trzy płyty kosztują razem $210.

Pomocna okazała się być oferta firmy „MikroElektronika” która jest polecanym przez „Microchip” dostawcą narzędzi deweloperskich. Zainteresowała mnie płyta o nazwie „multimedia for PIC32MX7” posiadająca wszystkie wymagane przeze mnie peryferia oprócz odbiornika podczerwieni. Po jego dołożeniu płyta spełniała założenia w stu procentach dlatego zdecydowałem się ją użyć. Cena płyty wynosi $149 i została ona zasponsorowana na potrzeby tej pracy inżynierskiej przez firmę „MB Turnkey Design” za co serdecznie dziękuję.



**Rysunek 1.1.** Zdjęcie płyty "multimedia for PIC32MX7" z obu stron

Źródło: Na podstawie [3]

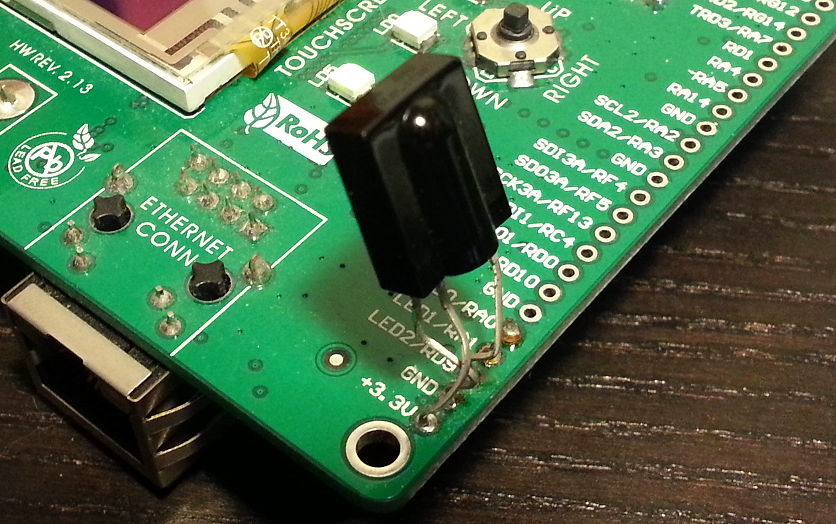
Na płycie zastosowany został kodek audio WM8731 wyprodukowany przez firmę „Wolfson”. Posiada on wbudowane dwa 24-bitowe przetworniki cyfrowo analogowe umożliwiające pracę z maksymalną częstotliwością próbkowania 96kHz [4]. To znacznie więcej niż założono. Układ ten posiada również wbudowany wzmacniacz słuchawkowy oraz również dwa przetworniki analogowo cyfrowe pozwalający na obsługę nie tylko wyjścia ale również wejścia stereo. Na wybranej płycie ewaluacyjnej wyprowadzone zostało stereofoniczne gniazdo słuchawkowe oraz monofoniczne gniazdo mikrofonowe. W pracy zostanie wykorzystane jedynie stereofoniczne wyjście audio.

Zastosowany na płycie wyświetlacz o rozdzielczości QVGA[[10]](#footnote-10) posiada wbudowany układ graficzny HX8347 obsługujący tryb RGB666 co daje możliwość wyświetlenia 256 tysięcy kolorów [5]. Docelowo projektowane urządzenie ma pracować z układem graficznym zrealizowanym na strukturze FPGA którego zaprojektowanie jest tematem innej pracy inżynierskiej (patrz rozdział Układ graficzny na stronie 5). Biorąc pod uwagę, iż prace nad oby dwoma projektami będą prowadzone równolegle, warto zapewnić sobie możliwość wcześniejszego uruchamiania i testowania aplikacji na rozwiązaniu tymczasowym w postaci gotowego układu graficznego.

Zgodnie z założeniami na płycie ewaluacyjnej znajdują się również takie peryferia jak gniazdo na kartę SD i port RS232 oraz dodatkowe, nie wymagane paryferia takie jak złącze Ethernet, czujnik temperatury, gniazdo USB (slave i host), cztery diody LED i joystick.

Wybrana płyta posiada możliwość zainstalowania dowolnych dodatkowych urządzeń dzięki wyprowadzeniu przez producenta płyty wszystkich portów zastosowanego mikrokontrolera na złącza znajdujące się na krawędziach laminatu. Tego rodzaju rozwiązanie pozwoli połączyć układ z osobą płytą ewaluacyjną z układem FPGA oraz umożliwi dołączenie jedynego brakującego urządzenia peryferyjnego którym jest odbiornik podczerwieni.

Zastosowany został odbiornik TSOP1138 [6] pracujący na częstotliwości nośnej 38kHz oraz kompatybilny z protokołem transmisji NEC [7]. Został on przylutowany do płyty wykorzystując złącza z wyprowadzeniami portów mikrokontrolera (patrz Rysunek 1.2 na stronie 5). Wybrany został NEC ponieważ jest to obecnie jeden z najczęściej stosowanych przez producentów sprzętu elektronicznego protokół.



**Rysunek 1.2.** Montaż odbiornika podczerwieni TSOP1138

Źródło: Opracowanie własne

### Układ graficzny

Chciałbym teraz powrócić do projektu opisanego we wstępie tego rozdziału. Na procesorze PIC32MX460L512 zastosowanym w tym układzie przy pomocy modułu PMP możliwe było wysłanie do sterownika wyświetlacza 160 ramek na sekundę. Wielkość ta jest imponująca jednak rozmywa się nieco z prawdą. Po pierwsze cała klatka obrazu musi być trzymana w pamięci operacyjnej procesora. W przypadku wyświetlacza o tak małej rozdzielczości oraz ograniczeniu formatu koloru piksela do RGB565 można sobie pozwolić na przetrzymywanie bufora ramki (FB[[11]](#footnote-11)) w pamięci operacyjnej. W przypadku większego wyświetlacza, zakładając dostępność jedynie 128 kB pamięci operacyjnej, staje się to niemożliwe. Po drugie obraz ten musi być najpierw w jakiś sposób wygenerowany – to właśnie ta operacja zajmuje najwięcej czasu procesora.

W obliczu tej sytuacji konieczne było zastosowanie dodatkowego układu graficznego odciążającego procesor. Możliwe było zastosowanie gotowego układu takiego jak na przykład „Solomon Systech SSD1926 Graphics Controller” lub stworzenie własnego układu graficznego opartego na programowalnym układzie logicznym (FPGA[[12]](#footnote-12)). Jako że Bartosz Zamolski zaproponował zaprojektowanie takiego układu graficznego, jako temat jego pracy inżynierskiej, stało się jasne, że moje urządzenie musi współpracować z tym właśnie układem.

## Software

Po wybraniu platformy sprzętowej dla projektowanego urządzenia, należało zastanowić się nad wyborem języka programowania oraz zdecydować, które części systemu zostaną oparte o gotowe rozwiązania. Podczas projektowania, stosunkowo rozbudowanej jak na system wbudowany aplikacji, zawsze należy zastanowić się, które fragmenty kodu będziemy tworzyć od zera a które wykorzystamy gotowe napisane przez kogoś innego. Tworzenie wszystkich elementów samodzielnie często prowadzi do niepowodzenia całego projektu. Podejście takie zajmuje znacznie więcej czasu a stworzone w ten sposób fragmenty oprogramowania często posiadają wiele błędów. Zawsze należy skupić się na fragmentach na których najbardziej nam zależy i dla których dostępność gotowych rozwiązań jest ograniczona.

W rozdziale tym został opisany proces podejmowania decyzji projektowych dotyczących programowej części pracy.

### Język programowania

W chwili rozpoczynania prac nad projektem, jedynym dostępnym językiem programowania dla procesorów z serii PIC32 był język „C”. Oczywiście znając architekturę procesora można zastosować dowolny kompilator wspierający daną architekturę. Jednak brak plików nagłówkowych określających rejestry procesora, środowiska programistycznego oraz możliwości debugowania aplikacji w sposób krokowy, skutecznie odstrasza od stosowania niewspieranych kompilatorów.

W trakcie trwania prac nad projektem firma „Microchip” wypuściła nową rodzinę kompilatorów dla swoich produktów. Kompilatory te wspierają zarówno język „C” jak i „C++”. W takiej sytuacji możliwe było zastosowanie języka „C” dla niskopoziomowych funkcji systemu, natomiast dla wysokopoziomowych „C++”. Zastosowanie języka obiektowego dla najwyższych warstw systemu spowodowałoby ułatwienie procesu tworzenia nowych aplikacji. Niestety stopień zaawansowania prac nad projektem wykluczył możliwość łatwej zmiany kompilatora.

### System operacyjny

Większość użytkowników komputerów oraz systemów wbudowanych posiada pewne doświadczenie związane z obsługą i zarządzaniem systemem operacyjnym, jednak trudno jest im określić czym właściwie jest system operacyjny. Dzieje się tak dlatego, że system operacyjny ma dwie niezwiązane ze sobą funkcje [8]:

1. Funkcja menadżera dostępu do zasobów
2. Funkcja maszyny wirtualnej

#### System operacyjny jako menadżer dostępu do zasobów

Obecnie komputery a nawet proste systemy wbudowane posiadają możliwość uruchamiania kilku aplikacji lub realizowania kilku zadań jednocześnie. Aby każde z uruchomionych zadań mogło korzystać z tych samych zasobów, np. sprzętowych, wymagane jest aby synchronizowały one między sobą kiedy które z nich będzie korzystało z danego zasobu. Zarówno za możliwość uruchamiania kilku wątków jednocześnie jak i komunikacji między nimi odpowiada moduł systemu operacyjnego zwany planistą (z ang. scheduler). Moduł ten jest absolutną podstawą działania każdego wielowątkowego systemu operacyjnego. Jeden błąd popełniony podczas jego pisania może implikować wiele trudnych do wychwycenia błędów występujących w aplikacjach. Dlatego właśnie tworzenie tego modułu systemu operacyjnego od podstaw jest ekstremalnie trudne oraz czasochłonne i mogłoby być tematem osobnej pracy inżynierskiej.

Jako że, każdy system operacyjny czasu rzeczywistego posiada planistę, postanowiłem skorzystać z jednego z nich. Ze względu na ilość wspieranych architektur procesorów, powszechność oraz licencję został wybrany FreeRTOS. Ciekawym aspektem tej pracy inżynierskiej będzie stworzenie w pełni funkcjonalnego systemu operacyjnego (z pewnymi wyjątkami) bazując na gotowym rozwiązaniu systemu czasu rzeczywistego.

#### System operacyjny jako maszyna wirtualna

Jak wiadomo każdy procesor jak i każde urządzenie wejścia/wyjścia posiada pewien zbiór instrukcji na podstawie których wykonuje operacje. Należałoby zastanowić się, czy programista piszący aplikację, powinien zastanawiać się jakich rozkazów procesora użyć oraz jakie polecenia wysłać do urządzeń peryferyjnych aby wykonać zadaną czynność. Oczywiście jeżeli istniałaby taka konieczność, to przez stopień skomplikowania tworzenia oprogramowania, większość obecnie dostępnych programów komputerowych nigdy by nie powstało. Aby lepiej zobrazować problem wykorzystajmy przykład rezystancyjnego panelu dotykowego. Gdyby system operacyjny nie pełnił opisywanej funkcji, programista piszący prostą aplikację okienkową (np. kalkulator) aby odczytać pozycję rysika na panelu musiałby wystawić stan wysoki na wejście X+ panelu oraz stan niski na wejście X- po czym odczytać stan z wyjścia Y+, operację powtórzyć, odfiltrować najbardziej odstające próbki po czym uśrednić pozostałe wyniki. Zamiast niego, może to zrobić system i udostępnić programiście jedną funkcję za pomocą której pobierze dokładną odfiltrowaną pozycję rysika. Idąc dalej programista nie musi nawet znać pozycji rysika – wystarczy mu informacja, że użytkownik wcisnął przycisk „+”. Te informacje przez różne warstwy abstrakcji udostępnia programiście system operacyjny.

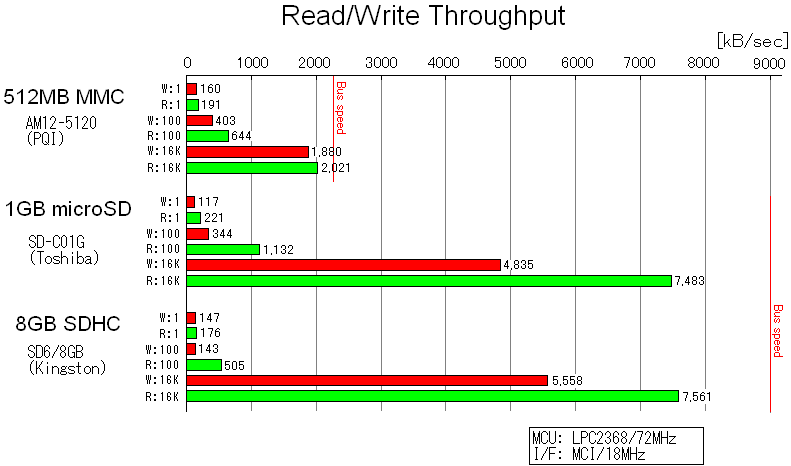
To właśnie nad tą funkcją systemu operacyjnego chciałbym skupić się w pracy najbardziej. Samodzielne przejście przez wszystkie etapy tworzenia tej funkcji pozwoliło mi dogłębnie zrozumieć zasady działania dzisiejszych systemów operacyjnych oraz zgłębić wiedzę dotyczącą projektowania tego typu systemów. Głównym źródłem informacji którym kierowałem się podczas tworzenia poszczególnych elementów projektu były kody źródłowe innych systemów operacyjnych tj. Linux, Haiku, ReactOS oraz ChibiOS.

### System plików

Głównym założeniem wynikającym z tematu pracy ma być możliwość odtwarzania plików audio. Warunkuje to konieczność implementacji wybranego systemu plików. Jako jeden z najczęściej stosowanych na pamięciach przenośnych systemów plików został wybrany system FAT32. Jest on powszechnie stosowany przez twórców różnego rodzaju systemów wbudowanych dzięki czemu istnieje wiele gotowych implementacji. Proces tworzenia własnej implementacji systemu plików, podobnie jak w przypadku planisty, jest procesem skomplikowanym oraz wymagającym specjalistycznej wiedzy i doświadczenia. Dlatego też zdecydowałem się użyć jednej z gotowych implementacji.

Projekt zabawki (patrz rozdział 1.1 na stronie 5) był oparty o implementację przygotowaną przez firmę „Microchip” do współpracy z ich procesorami. Okazało się, że jest ona bardzo wolna i w oryginalnej formie nie pozwoli na spełnienie założonych prędkości odczytu[[13]](#footnote-13). Konieczna była modyfikacja polegająca na zastosowaniu kontrolera DMA[[14]](#footnote-14) do kopiowania bloków pamięci z karty SD do pamięci procesora.

Po zapoznaniu się z wynikami testów wydajności (patrz rysunek 1.3 na stronie 17) oraz aby uniknąć problemów z koniecznością modyfikacji istniejącego rozwiązania zdecydowałem się na użycie implementacji o nazwie FatFS [9]. Testy zostały przeprowadzone na procesorze o podobnej częstotliwości taktowania więc spodziewamy się zbliżonych wyników.



**Rysunek 1.3.** Wyniki testów biblioteki FatFS

Źródło: Na podstawie [9]

### Biblioteka graficzna

Z tematu niniejszej pracy inżynierskiej wynika, iż projektowane urządzenie ma współpracować z układem graficznym zbudowanym na układzie FPGA. Biblioteka graficzna powinna być zaprojektowana tak, aby jak najlepiej pasowała do możliwości układu graficznego. Jest to główny argument dla którego zdecydowałem się na napisanie własnej biblioteki graficznej. Kolejnym argumentem „za”, była chęć stworzenia takiego graficznego interfejsu użytkownika jaki sobie wyobrażałem. Posiadanie możliwości wpływu na to jakie efekty graficzne są wspierane sprzętowo pobudza wyobraźnię.

Niestety, z powodu opóźnionego dostępu do płyt ewaluacyjnych, koledze nie udało się skończyć projektu układu graficznego na czas. Prace są nadal prowadzone, na bieżąco dodawane są nowe funkcje do sterownika układu graficznego.

W zastępstwie wykorzystany został układ graficzny wbudowany w wyświetlacz znajdujący się na wybranej w rozdziale 1.1.2 na stronie 5 płycie ewaluacyjnej. Kod źródłowy przystosowany jest do łatwego przełączania między tymi dwoma układami. Jeżeli tylko uda się zakończyć prace nad układem graficznym FPGA do czasu obrony – zaprezentowana zostanie wersja z tym właśnie układem.

Więcej szczegółów technicznych dotyczących biblioteki graficznej zostanie omówione w dalszej części pracy (patrz XXX na stronie XXX).

ROZDZIAŁ 2

# Architektura systemu

Rozdział ten będzie prezentował architekturę oprogramowania które powstało na potrzeby pracy. Została przyjęta kolejność omawiania modułów od tych najniżej poziomowych do coraz wyższej warstwy abstrakcji. Na początku przedstawiona zostanie struktura katalogów w projekcie aby w razie ewentualnego czytania kodu równolegle z dokumentacją ułatwić czytelnikowi nawigację. Następnie przejdziemy do sposobu przystosowywania źródła do konkretnej płyty drukowanej. Kolejnym etapem będzie prezentacja warstwy abstrakcji sprzętu (HAL[[15]](#footnote-15)) oraz omówienie najważniejszych z obecnie dostępnych sterowników. Po omówieniu sterowników przejdziemy do prezentacji bibliotek.

## Struktura katalogów

Aktualnie oprogramowanie urządzenia składa się z kilkudziesięciu plików. Aby łatwo nimi zarządzać konieczne było przygotowanie drzewa katalogów podzielonego w taki sposób, aby każdy programista który po raz pierwszy widzi projekt, był w stanie bez trudu znaleźć poszukiwany sterownik czy bibliotekę. Kiedy projekt był rozpoczynany, taka struktura katalogów wydawała się być zbyt rozbudowana. Wraz z rozwojem projektu okazało się, iż pozwoliła ona w łatwy sposób zapanować nad porządkiem. Tabela 2.1 przedstawia obecne drzewo najważniejszych katalogów (niektóre, nieznaczące zostały pominięte).

**Tabela 2.1.** Struktura katalogów projektu

|  |  |
| --- | --- |
| Ścieżka katalogu | Zawartość |
| ./app | Katalog zarezerwowany dla plików dodatkowych aplikacji. Aktualnie znajduje się w nim jedynie kod źródłowy odtwarzacza audio. |
| ./boards | Katalog przeznaczony na pliki konfiguracyjne projektu dla konkretnych płyt PCB. Dokładny opis znajduje się w rozdziale 2.3.1 na stronie 5. |
| ./hal | Katalog zarezerwowany na pliki warstwy abstrakcji sprzętu |
| ./hal/hld | W katalogu tym znajdują się pliki nagłówkowe definiujące modele struktur opisujących sterowniki różnych typów (patrz rozdział 2.3.2 na stronie 5). |
| ./hal/lld | Pliki źródłowe oraz nagłówkowe konkretnych sterowników sprzętu. |
| ./hal/lld/platforms | Katalog na sterowniki zależne od wybranej platformy sprzętowej czyli procesora lub projektu płyty PCB. |
| ./lib | Katalog zawiera źródła wszystkich bibliotek systemowych. Każda z nich posiada swój podkatalog. Pliki bibliotek wspólnych nie posiadają podkatalogu. |
| ./os | Pliki źródłowe systemu FreeRTOS. |

## FreeRTOS

Jak zostało wcześniej wspomniane w projekcie za wielowątkowość oraz zarządzanie dostępem do zasobów odpowiedzialny będzie system operacyjny czasu rzeczywistego FreeRTOS. Podrozdział ten nie ma na celu prezentacji możliwości tego systemu ani opisywania jego poszczególnych funkcji. Zostanie tutaj przedstawiona jedynie konfiguracja tego systemu do pracy w naszym środowisku. Plik konfiguracyjny znajduje się pod ścieżką: ./os/FreeRTOSConfig.h.

Jednym z parametrów konfiguracyjnych systemu FreeRTOS jest stała configUSE\_PREEMPTION, odpowiedzialna jest ona za włączenie wywłaszczania. Oczywiście w naszym przypadku została ustawiona na wartość „1” co oznacza włączenie danej funkcji. Włączenie wywłaszczanie oznacza, że wykonywany aktualnie wątek zostać w dowolnej chwili przerwany a czas procesora zostanie przekazany do innego wątku który aktualnie ma najwyższy priorytet a jego flaga odpowiadająca za gotowość jest ustawiona. W systemach bez wywłaszczania wykonywanie wątku zostanie przerwane jedynie w sytuacji gdy ten wyrazi na to zgodę (wykona tak zwaną procedurę „yeld” – nazwa zależna od systemu).

Kolejnym interesującym nas parametrem konfiguracyjnym jest configTICK\_RATE\_HZ. Stała ta definiuje z jaką częstotliwością mają być wykonywane kroki planisty. Dokładniej oznacza to czas na jaki zostanie ustawiony zegar który po przepełnieniu zgłasza przerwanie w którym wykonywany jest jeden krok planisty. Planista podczas wykonywania kroku może zmienić kontekst procesora, czyli zmienić wykonywany wątek. W naszym przypadku wartość to została ustawiona na 1000 Hz. Jest to stosunkowo wysoka wartość. Nie zostały jeszcze przeprowadzone testy wpływu tej wartości na ogólną wydajność systemu. Można uznać, że wartość ta jest tymczasowa.

Następne dwa parametry configCPU\_CLOCK\_HZ i configPERIPHERAL\_CLOCK\_HZ określają częstotliwości zegarów dla procesora oraz urządzeń peryferyjnych. W naszym przypadku dla obu wartości jest to 80Mhz i jest to maksymalna wartość z jaką może pracować zastosowany procesor.

Nieco niżej w pliku pojawiają się parametry odpowiedzialne za wielkość stosu przerwania (configISR\_STACK\_SIZE) oraz wielkość sterty (configTOTAL\_HEAP\_SIZE). Wielkość stosu dla przerwania została ustawiona na 1024. Jest to wysoka wartość, niestety przerwanie odpowiedzialne za pobieranie z pamięci oraz wysyłanie do przetwornika kolejnej próbki pliku audio wymaga tak dużego stosu. Sterta jest to miejsce w pamięci operacyjnej na które trafiają wszystkie zmienne zadeklarowane dynamicznie. Wielkość ta została ustawiona na 64kB czyli dokładnie połowę dostępnej pamięci.

Przy okazji omawiania wielkości sterty warto wspomnieć o wybranym modeli pamięci. FreeRTOS oferuje aż cztery modele do wyboru. Model pierwszy pozwala jedynie alokować pamięć. Raz zadeklarowanej pamięci nie można już zwolnić. Model drugi pozwala zarówno alokować jak i zwalniać pamięć, jednak po zwolnieniu pamięci nie grupuje zwolnionych bloków w jeden ciągły obszar pamięci. Trzeci model jest jedynie zapewnieniem synchronizacji między wątkami dla poleceń malloc i free używanego kompilatora. Model ten nie jest zalecany ponieważ powoduje wzrost objętości kodu. Czwarty model działa podobnie do modelu drugiego z tą różnicą, że powoduje grupowanie zwolnionych bloków pamięci. Jako że model czwarty jest zalecany jedynie w przypadku alokowania bloków pamięci o losowych rozmiarach, a my tego nie robimy, w projekcie zastosowany został model drugi.

Następne istotne parametry definiują jedynie włączenie lub wyłączenie poszczególnych funkcji systemu. W omawianym projekcie wszystkie funkcje poza współprogramami (ang. co-routines) są włączone.

FreeRTOS udostępnia dwa bardzo pomocne podczas tworzenia oprogramowania wywołania zwrotne. Jedno z nich odpowiada za wystąpienie wyjątku procesora (np. dzielenie przez zero) drugie natomiast wywoływane jest w przypadku wystąpienia przepełnienia stosu.

W przypadku wystąpienia wyjątku procesora odczytanie wartości kilku rejestrów może być niezwykle pomocne podczas debugowania aplikacji. Dla architektury MIPS są to rejestry EPC, STATUS oraz CAUSE. Rejestr EPC[[16]](#footnote-16) przechowuje licznik programu w chwili wystąpienia wyjątku. Na podstawie tej wartości możemy dokładnie określić linię kodu w której wystąpił problem. Rejestry STATUS oraz CASUE odpowiadają kolejno za status procesora oraz powód wystąpienia wyjątku. Na podstawie wartości zapisanych w tych trzech rejestrach możemy szczegółowo określić problem jaki wystąpił. Aby ułatwić uzyskiwanie wartości tych rejestrów w wywołaniu zwrotnym została zapisana procedura która drukuje je na ekranie. Jest to typowe zachowanie systemu operacyjnego znane z systemu „Windows” pod niechlubną nazwą „Blue Screen Of Death”. Rysunek 2.1 przedstawia efekt przykładowego, wywołanego sztucznie, wyjątku procesora. Dokładne informacje na temat rejestrów procesorów MIPS oraz ich szczegółowe opisy znajdują się w pozycji [10] bibliografii.

!!!!RYSUNEK!!!!!

**Rysunek 2.1.** Wyjątek procesora. Wartości rejestrów EPC, STATUS oraz CAUSE wydrukowane na ekran

Źródło: Opracowanie własne

Podobnie w przypadku wystąpienia przepełnienia stosu, pomocne podczas debugowania jest wyciągnięcie z systemu informacji o tym, który wątek spowodował przepełnienie oraz o jaką ilość danych został on przepełniony. Ta sytuacja również została obsłużona wyświetleniem pomocnych informacji na ekranie. W celu ułatwienia identyfikacji czy wystąpił wyjątek czy przepełnienie, przypadek ten został wyświetlony na czerwonym tle. Zdjęcie ekranu po wystąpieniu zasymulowanego przepełnienia stosu przedstawia Rysunek 2**.2**.

!!!!!!!!!RYSUNEK!!!!!!!

**Rysunek 2.2.** Przepełnienie stosu. Zwizualizowana nazwa wątku oraz adresy początku i końca stosu

Źródło: Opracowanie własne

Szczegółowe informacje na temat obsługi systemu FreeRTOS dostępne są w: [11]. Wewnętrzna budowa systemu opisana została w [12].

## Warstwa abstrakcji sprzętu

W ujęciu systemu operacyjnego warstwa abstrakcji sprzętu jest zaimplementowaną w oprogramowaniu warstwą pomiędzy platformą sprzętową a oprogramowaniem. Jej funkcją jest ukrycie różnic w sprzęcie tak aby programista piszący aplikację, na dany system operacyjny nie musiał używać poleceń specyficznych dla konkretnego urządzenia a zamiast tego stosował uniwersalne odwołania specyficzne dla tego typu urządzeń. Kolejną funkcją jaką zawdzięczamy warstwie abstrakcji sprzętu jest możliwość uruchamianie systemu operacyjnego na różnym sprzęcie bez konieczności ponownej kompilacji. Funkcja ta jest zarezerwowana dla rozbudowanych systemów operacyjnych uruchamianych na różnych, aczkolwiek podobnych pod względem architektury platformach sprzętowych. W ujęciu systemów wbudowanych dynamiczne wykrywanie podłączonego sprzętu ma sens, ale tylko w ściśle zdefiniowanym zakresie np. wykrycie typu głowicy w odbiorniku telewizyjnym.

Nawet jeżeli zrezygnujemy z dynamicznej detekcji platformy sprzętowej dzięki warstwie abstrakcji sprzętu znacznie ułatwimy sobie przenoszenie na inną platformę. Wystarczy przygotować odpowiedni zestaw sterowników, aby bez modyfikacji wysokopoziomowej części oprogramowania uruchomić system na zupełnie innym sprzęcie. Oczywiście będzie to wymagało kompilacji, ale jak już zostało powiedziane w przypadku systemów wbudowanych nie jest to przeszkodą.

Projekt realizowany na potrzeby tej pracy miał się wykazywać uniwersalnością oraz łatwością zastosowania do przyszłych rozwiązań – właśnie dlatego została w nim zaimplementowana prosta, aczkolwiek w pełni funkcjonalna warstwa abstrakcji sprzętu. Pierwszym momentem w którym okazała się pomocna była wymiana wyświetlacza z zainstalowanego na płycie ewaluacyjnej na zewnętrzy wyświetlacz z układem graficznym FPGA. Jeżeli chcemy skompilować program do współpracy z układem graficznym FPGA dodajemy do pliku konfiguracyjnego platformy sprzętowej definicję LCD\_FPGA. Dzięki temu kompilator wie który sterownik wyświetlacza wybrać (patrz **Kod źródłowy 2.1**).

**Kod źródłowy 2.1.** Wybór sterownika wyświetlacza w czasie kompilacji

#ifdef LCD\_FPGA

ret = lldFpgaGpuAttach(); // Select FPGA based GPU

#else

ret = lldHx8347Attach(); // Select HX8347 graphics controller

#endif

Znaczenie funkcji zostanie dokładnie lldXXXAttach() zostanie opisane w dalszej części pracy (patrz 2.3.3 na stronie 5).

### Board init

Pierwszym bardzo prostym elementem warstwy abstrakcji sprzętu jest moduł inicjalizacji systemu do współpracy z płytą PCB, na potrzeby której kompilowane jest oprogramowanie.

W strukturze katalogów zawarty został katalog „boards”. W jego wnętrzu znajdują się podkatalogi przypisane do konkretnych płyt. Poprzez ustawienia projektu wybieramy który z tych katalogów ma zostać dołączony do kompilacji. W ten sposób wybieramy płytę do współpracy z którą będzie kompilowane oprogramowanie. Każdy z tych katalogów posiada dwa pliki board.c oraz board.h.

W pliku nagłówkowym konieczne jest zdefiniowanie pewnych stałych konfiguracyjnych, wymaganych przez system bądź niektóre sterowniki, oraz zdefiniowanie porów procesora do których podłączone są pewne urządzenia. Na przykład sterownik wyświetlacza, wymusza konieczność zdefiniowania orientacji ekranu, oraz portu, pod który podłączone jest sterowanie podświetleniem (patrz Kod źródłowy 2.1). Natomiast sterownik panelu dotykowego, fabrycznych wartości kalibracji, oraz kanałów przetwornika analogowo cyfrowego, do których jest on podłączony. Mogą się tutaj znaleźć również definicje określone i wykorzystywane później przez użytkownika. Taką definicją jest na przykład opisywana wcześniej LCD\_FPGA.

**Kod źródłowy 2.2.** Konfiguracja sprzętowa sterownika wyświetlacza

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// BOARD Configuration

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define LCD\_ORIENTATION 90

#define LCD\_FPGA

#define LCD\_FPGA\_TEST

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GPIO Definitions

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef LCD\_FPGA

#define LCD\_BACKLIGHT\_GPIO LATDbits.LATD2 // not used

#define LCD\_RESET\_GPIO LATCbits.LATC1 // not used

#define LCD\_RS\_GPIO LATBbits.LATB5

#define LCD\_CS\_GPIO LATAbits.LATA10

#else

#define LCD\_BACKLIGHT\_GPIO LATDbits.LATD2

#define LCD\_RESET\_GPIO LATCbits.LATC1

#define LCD\_RS\_GPIO LATBbits.LATB15

#define LCD\_CS\_GPIO LATBbits.LATB14

#endif

### HLD

Asdasdasdasd

### LLD

Adasdasdasda

#### UART

Asdas

#### ADC

Asd

#### IR

Asdasd

#### AUDIO

Asdasdasd

#### SDMMC

Asdasdsa

#### LCD

Asdasdas

## Biblioteki

# Bibliografia

[1] Wikipedia, Otwarte oprogramowanie: <http://pl.wikipedia.org/wiki/Open_source> [dostęp: 2013-01-29]

[2] Wikipedia, SoC: <http://pl.wikipedia.org/wiki/System-on-a-chip> [dostęp: 2013-01-29]

[3] MikroElektronika: <http://www.mikroe.com/multimedia/pic32mx7> [dostęp: 2013-01-29]

[4] Wolfson Microelectronics plc: WM8731 Datasheet <http://www.wolfsonmicro.com/documents/uploads/data_sheets/en/WM8731.pdf>, 2012

[5] Himax Technologies, Inc: HX8347 Datasheet <http://www.displayfuture.com/Display/datasheet/controller/HX8347.pdf>, 2007

[6] Vishay Semiconductors: IR Receiver Modules for Remote Control Systems, <http://www.vishay.com/docs/82006/tsop11xx.pdf>, 2012

[7] Jason Howie: NEC Infrared Transmission Protocol, <http://wiki.altium.com/display/ADOH/NEC+Infrared+Transmission+Protocol>, 2008

[8] Andrew S. Tanenbaum, Albert S. Woodhull: *Operating Systems Design and Implementation, Third Edition*, Prentice Hall, 2006

[9] FatFS, <http://elm-chan.org/fsw/ff/00index_e.html> [dostęp: 2013-01-31]

[10] Dominic Sweetman: *See MIPS® Run - Second Edition*, Elsevier, San Francisco, 2007

[11] Nicolas Melot: *Study of an operating system: FreeRTOS*, <http://stiff.univ-brest.fr/~boukhobza/images/stories/Documents/Teachings/OSM/expo/FreeRTOS_Melot.pdf>, 2009

[12] Rich Goyette: *An Analysis and Description of the Inner Workings*

*of the FreeRTOS Kernel*,<http://www.mikrocontroller.net/attachment/95930/FreeRTOSPaper.pdf>, 2007

1. (ang. Inter-Process Communication – IPC) [↑](#footnote-ref-1)
2. (ang. Real-Time Operating System – RTOS) [↑](#footnote-ref-2)
3. (ang. Pulse Width Modulation – PWM) [↑](#footnote-ref-3)
4. (ang. System on Chip – SOC) - układ scalony zawierający kompletny system elektroniczny, w tym układy cyfrowe, analogowe (także radiowe) oraz cyfrowo-analogowe [2] [↑](#footnote-ref-4)
5. (ang. Graphical User Interface – GUI) [↑](#footnote-ref-5)
6. 44,1kHz/16bitów [↑](#footnote-ref-6)
7. (ang. Parallel Master Port – PMP) [↑](#footnote-ref-7)
8. (ang. Inter-Integrated Circuit – I2C/IIC) [↑](#footnote-ref-8)
9. (ang. Serial Peripheral Interface) [↑](#footnote-ref-9)
10. 320x240 pikseli [↑](#footnote-ref-10)
11. (ang. Frame Buffer – FB) [↑](#footnote-ref-11)
12. (ang. Field Programmable Gate Array – FPGA) [↑](#footnote-ref-12)
13. Odczyt animacji: 15 fps, 40 kB na jedną klatkę obrazu. Wymagana prędkość odczytu 600 kB/s [↑](#footnote-ref-13)
14. (ang. Direct Memory Access – DMA) [↑](#footnote-ref-14)
15. (ang. Hadrware Abstraction Layer – HAL) [↑](#footnote-ref-15)
16. (ang. Exception Program Counter – EPC) [↑](#footnote-ref-16)