

Państwowa Wyższa Szkoła Zawodowa w Tarnowie



Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka stosowana

Rok akademicki 2019/2020

Maksymilian Hebda

Mateusz Kiełbasa

PRACA INŻYNIERSKA

System sterowania modelem inteligentnego miasta

Promotor pracy:

dr inż. Daniel Król

Tarnów 2020

Składamy serdeczne podziękowania promotorowi
dr inż. Danielowi Królowi za pomoc
okazywaną w trakcie tworzenia niniejszej pracy
oraz miłą atmosferę sprzyjającą efektywnemu działaniu

Spis treści

Wstęp	7
Cel i założenia projektu	8
Zakres pracy	9
1 Wykorzystane narzędzia i technologie	11
1.1 Środowisko programistyczne <i>MCUXpresso</i>	11
1.2 Biblioteka graficzna <i>emWin</i>	12
1.3 Dokumentacja kodów źródłowych	13
2 Realizacja sprzętowa	15
2.1 Mikrokontroler <i>LPC54628</i>	15
2.2 Magistrala I2C	17
2.2.1 Ekspandery wyprowadzeń	18
2.2.2 Multiplekser I2C	19
2.2.3 Realizacja inteligentnego parkingu	21
2.3 Moduł zasilający	24
2.4 Sprzętowa realizacja animacji	25
2.4.1 Kino Palace	25
2.4.2 Sygnalizacja świetlna	27
2.4.3 Turbina wiatrowa	28
2.4.4 Roller Coaster	30
2.5 Rzut poziomy makiety	31
3 Implementacja oprogramowania	33
3.1 Struktura projektu	33
3.2 Inicjalizacja systemu	34
3.3 System operacyjny czasu rzeczywistego	35
3.3.1 configUSE_TIME_SLICING	35
3.3.2 configUSE_PREEMPTION	36
3.3.3 configFRTOS_MEMORY_SCHEME	36

3.3.4	Zadania systemu sterowania	37
3.4	Obsługa kart SD	38
3.4.1	Format danych	39
3.4.2	FatFS	39
3.5	Sterowanie wyświetlaczem	40
3.6	Interfejs użytkownika	41
3.6.1	Okna dialogowe	42
3.6.2	Klawiatura ekranowa	43
3.6.3	Okno potwierdzenia	44
3.7	Serwer WWW	44
3.7.1	Żądania HTTP	44
3.7.2	Format wymiany danych JSON	45
3.7.3	Pobieranie danych (GET)	45
3.7.4	Wysyłanie danych (POST)	46
3.7.5	Pliki statyczne	48
3.8	Warstwa sterowania makietą	49
3.8.1	Inicjalizacja urządzeń	49
3.8.2	Struktura LED	49
3.8.3	Animacje systemu sterowania	50
3.8.4	Tryb automatyczny oraz ręczny	51
3.8.5	Grupowa komunikacja z urządzeniami	52
3.8.6	Inteligentny parking	52
3.8.7	Interfejs użytkownika	54
4	Testy działania systemu	55
4.1	Test spadku napięcia	55
4.2	Test błędnej inicjalizacji urządzeń I2C	55
4.3	Test zajętości pamięci operacyjnej	56
4.4	Test żądań HTTP	56
5	Podsumowanie	57
A	API serwera HTTP	59
B	Opis wyprowadzeń	67
B.1	Wyprowadzenia mikrokontrolera	67
B.2	Wyprowadzenia ekspanderów	67

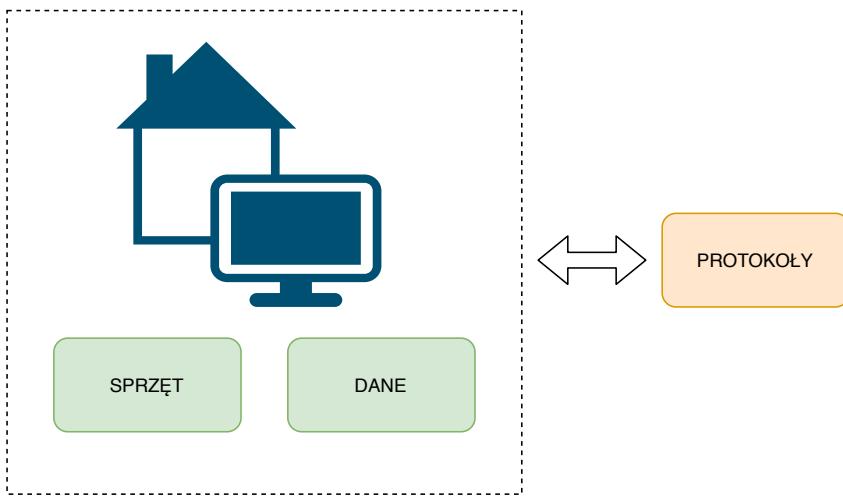
IoT	(ang. <i>Internet of Things</i>)
RTOS	(ang. <i>Real Time Operating System</i>)
API	(ang. <i>Application Programming Interface</i>)
ARM	(ang. <i>Advanced RISC Machine</i>)
SD	(ang. <i>Secure Digital</i>)
SCL	(ang. <i>Serial Clock Line</i>)
SDA	(ang. <i>Serial Data Line</i>)
LED	(ang. <i>Light Emitting Diode</i>)
GPIO	(ang. <i>General Purpose Input Output</i>)
OLED	(ang. <i>Organic Light Emitting Diode</i>)
IDE	(ang. <i>Integrated Development Environment</i>)
SDK	(ang. <i>Software Development Kit</i>)
JSON	(ang. <i>JavaScript Object Notation</i>)
SDRAM	(ang. <i>Synchronous Dynamic Random Access Memory</i>)
GUI	(ang. <i>Graphical User Interface</i>)
HTTP	(ang. <i>Hypertext Transfer Protocol</i>)
HTML	(ang. <i>HyperText Markup Language</i>)
TFT	(ang. <i>Thin Film Transistor</i>)
LCD	(ang. <i>Liquid Crystal Display</i>)

Wstęp

Szybki rozwój techniki, na przestrzeni ostatnich kilkunastu lat, na trwale zakorzenił koncepcję inteligentnych miast (ang. *smart city*). Wraz z dynamicznym powiększaniem się obszarów miejskich temat ten nabiera kluczowego znaczenia, gdyż stale postępująca komputeryzacja oraz informatyzacja muszą sprostać coraz to nowszym wymaganiom i oczekiwaniom społeczności. Z punktu widzenia informatyki, tworzenie i rozwój inteligentnych miast opiera się głównie na monitorowaniu i przetwarzaniu kluczowych informacji, efektywnym zarządzaniu energią, sprawnym zarządzeniu ruchem ulicznym oraz aktywnym uczestnictwie mieszkańców w życiu miasta i stałym jego ulepszaniu.

Pojęcie inteligentnych miast w powyższej formie nie byłoby możliwe bez istnienia koncepcji Internetu Rzeczy (ang. *Internet of Things*, IoT). Zgodnie z nią, wyróżnić możemy jednoznacznie identyfikowalne przedmioty, gromadzące, przetwarzające i przekazujące dane do systemów informatycznych za pomocą sieci komputerowych. Do tego typu urządzeń zaliczają się zarówno sprzęty gospodarstwa domowego, a także bardziej złożone struktury, takie jak inteligentne budynki, systemy energetyczne oraz nowoczesne systemy pomiarowe. Można zauważyć, że większość urządzeń IoT do poprawnej i wydajnej pracy wymaga nie tylko skutecznych infrastruktur teleinformatycznych, lecz stawia także wymóg sterowania w czasie rzeczywistym. Dało to początki do szybkiego rozwoju systemów wbudowanych.

Systemy wbudowane (ang. *embedded systems*), których podstawą są wyspecjalizowane mikrokontrolery, oferują wysoki stopień niezawodności oraz coraz większą integrację. Zapewniają mniejszy pobór energii przy jednoczesnym wyspecjalizowaniu w szybkim reagowaniu na zdarzenia krytyczne. Często spotyka się rozwiązania oparte na systemach operacyjnych czasu rzeczywistego (ang. *Real Time Operating Systems*, RTOS), które zapewniają odpowiedni przydział zasobów systemowych, priorytetyzowanie krytycznych zadań, obsługę przerwań systemowych i ułatwiających komunikację z urządzeniami wejścia-wyjścia.



Rysunek 1. Koncepcja Internetu Rzeczy.

Cel i założenia projektu

Celem niniejszej pracy było opracowanie systemu sterowania modelem inteligentnego miasta, spełniającego założenia koncepcji Internetu Rzeczy. Postawionymi celami projektu były:

1. budowa makietki miasta, z wykorzystaniem klocków LEGO Creator;
2. opracowanie i zrealizowanie niezbędnych modułów elektronicznych;
3. przegląd literatury dotyczącej implementacji oprogramowania, wykorzystującego możliwości systemów operacyjnych czasu rzeczywistego;
4. sterowanie makietą w czasie rzeczywistym;
5. implementacja i obsługa serwera HTTP w systemie wbudowanym;
6. optymalizacja czasu wykonania;
7. testy funkcjonalne, wydajnościowe oraz niezawodnościowe;
8. stosowanie sprawdzonych wzorców projektowych oraz dobrych praktyk programistycznych;

Zastosowanie technologii Ethernet, która jest jednym z protokołów wykorzystywanych przez koncepcję IoT umożliwiło sterowanie modelem miasta z dowolnego miejsca, za pośrednictwem sieci internetowej.

Zakres pracy

Prace nad systemem prowadzone były równolegle przez dwie osoby. W jej wyniku powstała część sprzętowa, na którą składają się:

- szyny zasilające oraz magistrale komunikacyjne;
- elementy oświetlenia inteligentnego miasta;
- moduł inteligentnego parkingu;

Część implementacyjna systemu obejmowała natomiast:

- implementację środowiska działającego w czasie rzeczywistym;
- komunikację z modułami systemu;
- implementację oraz obsługę wbudowanego serwera HTTP wraz z interfejsem programistycznym (ang. *Application Programming Interface, API*);
- komunikację z urządzeniami wejścia-wyjścia;



Rysunek 2. Makieta inteligentnego miasta.

Prace nad projektem prowadzone były zgodnie z podziałem:

Maksymilian Hebda:

- opracowanie układów animacji (kino, sygnalizacja świetlna, turbina wiatrowa);
- wykonanie modelu inteligentnego parkingu;
- obsługa wyświetlacza płyty prototypowej;
- opracowanie algorytmów sterowania;

Mateusz Kiełbasa:

- wykonanie infrastruktury komunikacyjnej, opartej na magistrali I2C;
- opracowanie programowej obsługi animacji (kolejka górska, tryb automatyczny);
- obsługa i konfiguracja systemu operacyjnego czasu rzeczywistego;
- implementacja serwera HTTP w systemie wbudowanym wraz z interfejsem programistycznym;

Efektem wspólnych działań było rozplanowanie i utworzenie instalacji elektrycznych w budynkach makiety.

Niniejszą pracę podzielono na siedem powiązanych ze sobą części.

Rozdział 1 dotyczy narzędzi oraz środowisk wykorzystanych w projekcie.

Rozdział 2 dotyczy realizacji sprzętowej pracy. Przedstawiono w nim niezbędne schematy elektryczne, urządzenia i wykorzystywaną magistralę komunikacyjną.

Rozdział 3 zawiera opis części programowej – w tym niezbędną konfigurację systemu, jego strukturę oraz działanie.

W rozdziale 4 opisano wykonane testy systemu wraz z ich wynikami.

Rozdział 5 zawiera podsumowanie niniejszej pracy oraz możliwości przyszłej rozbudowy makiety.

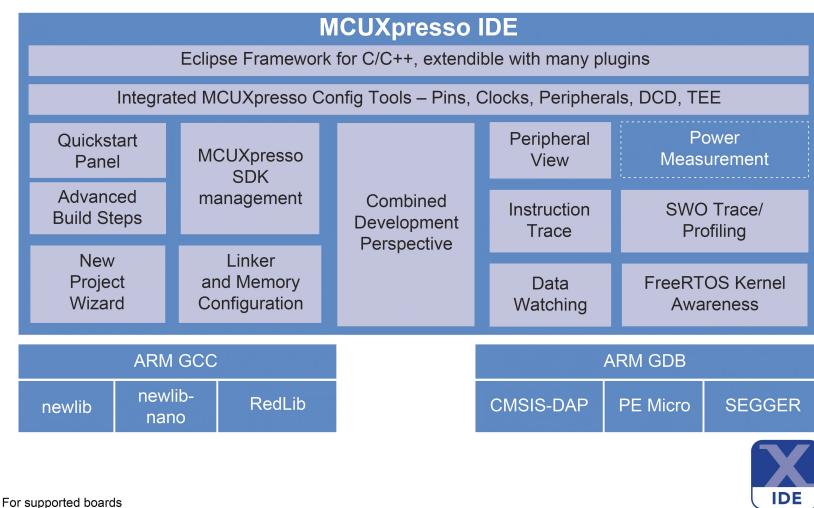
Dodatek A zawiera zaimplementowany interfejs programistyczny, który używany jest do dynamicznego sterowania elementami makiety.

Zebrane w dodatku B tabele przedstawiają schematy niezbędnych połączeń, upraszczając późniejszą rozbudowę makiety.

Wykorzystane narzędzia i technologie

1.1 Środowisko programistyczne *MCUXpresso*

Oprogramowanie sterujące makiety zaimplementowane zostało w środowisku *MCUXpresso IDE*. Jego dostępność na wszystkie popularne systemy operacyjne oraz wsparcie dla wielu bibliotek i dodatków zapewnia wysoką wygodę użytkowania. Środowisko dedykowane jest do pracy m.in. z płytami z serii LPC. Wbudowany debugger oraz łatwy proces programowania urządzeń to jedne z wielu zalet programu. Środowisko udostępnia również narzędzia przydatne podczas pracy z systemami czasu rzeczywistego.



Rysunek 1.1. Schemat blokowy środowiska *MCUXpresso* [1].

W projekcie wykorzystano zestaw narzędzi deweloperskich (SDK) w wersji *SDK v2.6.0*, który zawiera praktyczne przykłady programowania niezbędnych układów peryferyjnych mikrokontrolera. Projekt został skompilowany w systemie Linux wykorzystując kompilator gcc dla systemów wbudowanych wykorzystujących architekturę ARM oraz wykorzystuje darmowy system operacyjny czasu rzeczywistego *FreeRTOS*. Do obsługi serwera wykorzystano powszechnie stosowany, otwarty stos TCP/IP *LwIP* dedykowany dla systemów wbudowanych. Schemat wymiany danych opiera się na popularnym modelu JSON. Graficzny

interfejs użytkownika zaimplementowany został przy użyciu graficznej biblioteki *emWin*. W projekcie zastosowano wiele narzędzi i bibliotek usprawniających jego pracę:

- FatFS – obsługa kart SD;
- cJSON – interpreter formatu JSON w języku C¹;
- slre – parser wyrażeń regularnych w języku C²;
- Git – rozproszony system kontroli wersji;

Do stworzenia interfejsu użytkownika, dostępnego z przeglądarki internetowej wykorzystano frameworki *AngularJS* oraz *Bootstrap*.

1.2 Biblioteka graficzna *emWin*

Obsługę, wbudowanego w płytę prototypową wyświetlacza, umożliwia graficzna, dedykowana dla systemów wbudowanych biblioteka – *emWin*. Współpracuje ona zarówno z wyświetlaczami LCD (ang. *Liquid Crystal Display*) oraz TFT (ang. *Thin Film Transistor*) i umożliwia tworzenie zaawansowanych interfejsów graficznych użytkownika. Udostępnia obszerną dokumentację i praktyczne przykłady różnych aplikacji graficznych.



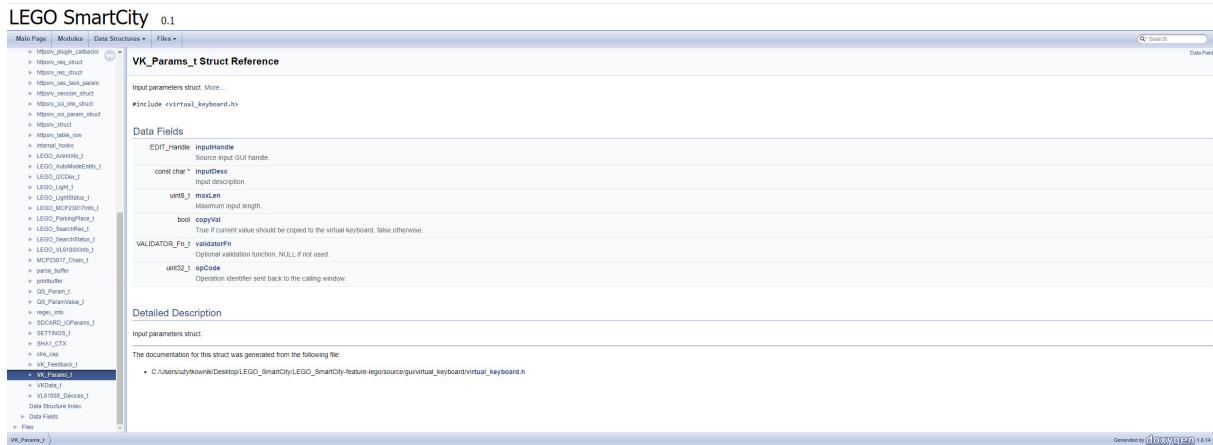
Rysunek 1.2. Przykładowy interfejs graficzny biblioteki *emWin* [2].

¹<https://github.com/DaveGamble/cJSON>, licencja MIT

²<https://github.com/cesanta/slre>, licencja GNU GPL v.2

1.3 Dokumentacja kodów źródłowych

Do utworzenia dokumentacji struktur danych oraz interfejsów wykorzystano narzędzie *Doxygen*. Wygenerowano pliki w formacie .html, które zawarte zostały na nośniku DVD. Uruchomienie dokumentacji polega na przejściu do katalogu *dokumentacja* na dołączonej płycie DVD, a następnie uruchomienie pliku *index.html* w przeglądarce internetowej.



Rysunek 1.3. Widok dokumentacji Doxygen.

Realizacja sprzętowa

Poniższy rozdział dotyczy sprzętowej realizacji systemu sterowania. Przedstawiono w nim mikrokontroler wykorzystywany w projekcie oraz wyszczególniono podstawowe schematy elektroniczne wraz z niezbędnymi połączeniami elektrycznymi i układami wspomagającymi. Opisano szynę komunikacyjną umożliwiającą niezbędną komunikację w systemie.

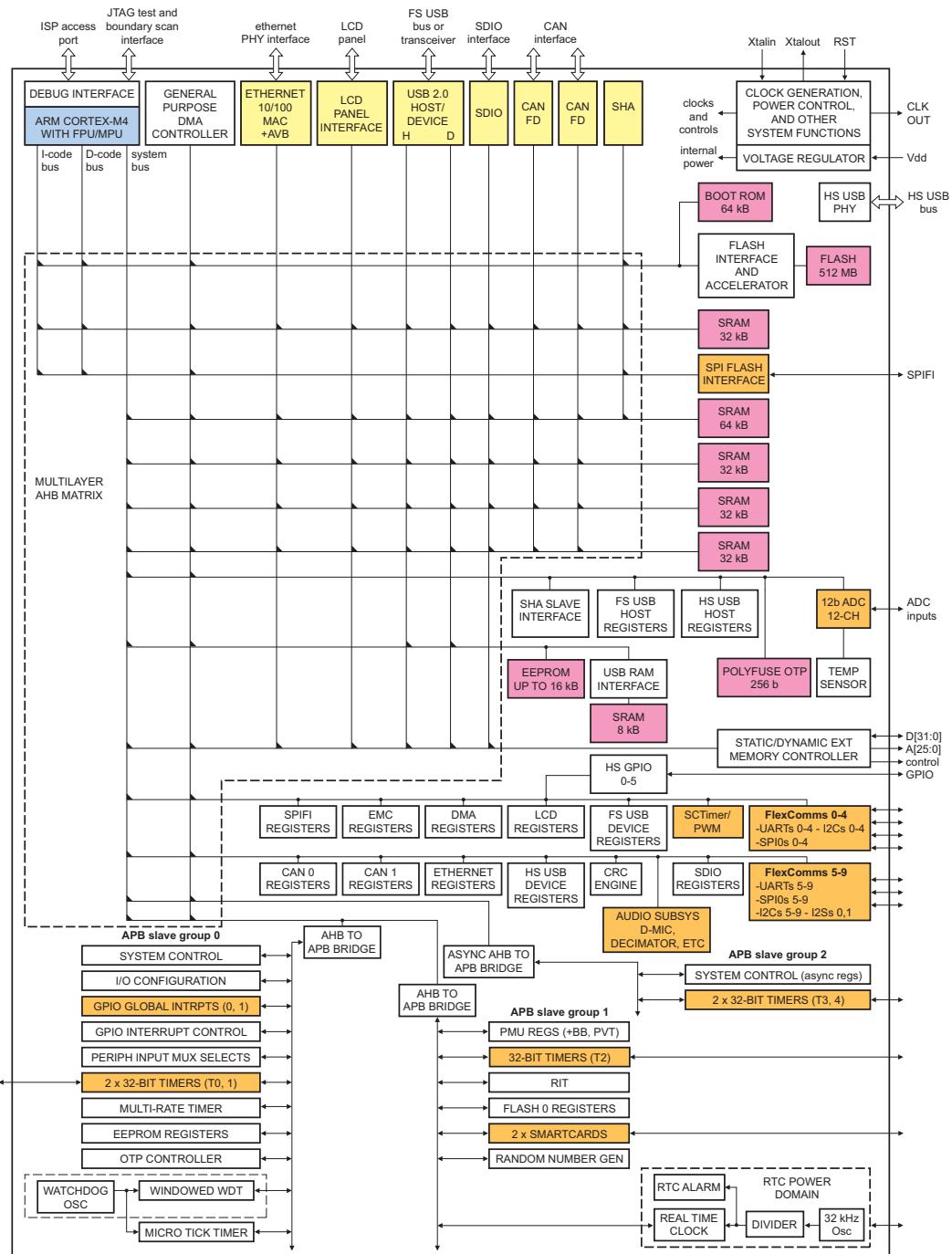
2.1 Mikrokontroler *LPC54628*

Wykorzystywana płyta ewaluacyjna, oparta jest na mikrokontrolerze z rdzeniem *ARM Cortex-M4* należy do rodziny *LPC546XX*, stanowiącej wydajne oraz bogate w interfejsy komunikacyjne procesory. Sprzętowy koprocesor (FPU) i szybkie magistrale obsługi pamięci zewnętrznych, znacznie upraszczają i przyśpieszają proces implementacji oprogramowania. Wbudowany, dotykowy wyświetlacz LCD umożliwia implementowanie zaawansowanych interfejsów użytkownika.



Rysunek 2.1. Płyta ewaluacyjna *LPCXpresso 54628* [3].

W projekcie wykorzystano zarówno pamięć wewnętrzna układu, jak również dedykowane moduły pamięci zewnętrznych FLASH (ang. *flash memory*) oraz SDRAM (ang. *Synchronous Dynamic Random Access Memory*). Sterowanie niskopoziomowymi modułami oparto na magistrali I2C. Dostęp do systemu realizowany jest poprzez wbudowany port Ethernet. Wykorzystano wbudowane złącze kart SD. Na poniższym schemacie przedstawiony został diagram blokowy mikrokontrolerów rodziny *LPC546XX*.

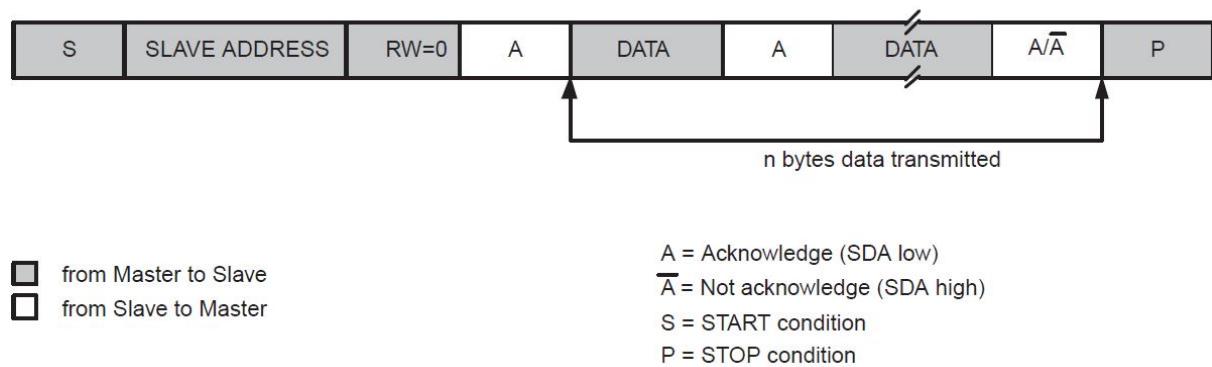


Rysunek 2.2. Diagram blokowy rodziny *LPC546XX* [4].

2.2 Magistrala I2C

Komunikacja z układami scalonymi sterującymi makietą realizowana są poprzez magistralę *I2C*.

Magistrala I2C jest szeregową magistralą dwukierunkową. Do transmisji danych używa zaledwie dwóch linii: *SDA* – linia danych (ang. *Serial Data Line*) oraz *SCL* - linia zegarowa (ang. *Serial Clock Line*). Schemat ramki komunikacyjnej zaprezentowano na poniższym rysunku.



Rysunek 2.3. Schemat wymiany danych I2C [4].

Magistrala I2C jest transmisją start-stopową, co oznacza, że każdy transfer zaczyna się od bitu *START*, następnie przesyłane są adres urządzenia oraz dane, których ilość nie jest przez standard ograniczona. Na końcu każdej transmisji wysyłany jest bit *STOP*. Każde urządzenie podłączone do magistrali jest skojarzone z unikalnym adresem jednoznacznie go identyfikującym. Zapewnia to dużą wygodę w procesie komunikacji. Istnieje wiele różnych standardów oraz szybkości transmisji. Tabela 2.1 prezentuje najpopularniejsze standardy magistrali I2C wraz z maksymalnymi prędkościami.

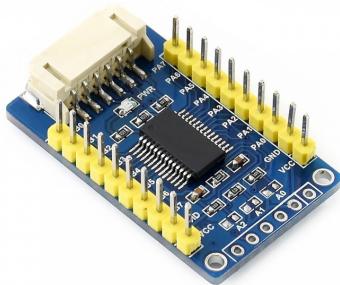
Tabela 2.1. Prędkości magistrali I2C.

Standard	Maksymalna prędkość
Standard mode	100 kbit/s
Fast mode	400 kbit/s
High speed (HS-Mode)	3,4 Mbit/s

W systemie użyty został standardowy tryb, którego maksymalna prędkość ograniczona jest do 100 kbit/s.

2.2.1 Ekspandery wyprowadzeń

Konieczność unikalnego adresowania LED stwarza problem braku wystarczającej liczby wyprowadzeń mikrokontrolera (ang. *General Purpose Input Output*, GPIO). Problem ten rozwiązano stosując układ MCP23017. Moduł ten, zwany ekspanderem wyprowadzeń umożliwia sterowanie 16 niezależnymi pinami. Warto zaznaczyć, że podobnie jak mikroprocesor nie ogranicza kierunku przesyłu informacji, lecz umożliwia obsługę zarówno wejść, jak i wyjść cyfrowych. Wyposażony jest w dwa niezależne porty (A/B), które umożliwiają sterowanie pojedynczymi wyprowadzeniami lub ich grupami.



Rysunek 2.4. Ekspander wyprowadzeń MCP23017 [5].

Niewątpliwą zaletą układu jest możliwość zmiany adresu urządzenia za pomocą umieszczonych na płycie zworek. Konfigurowany adres zawiera się w przedziale 0x20 – 0x27, a zatem na jednej linii I2C można umieścić do 8 urządzeń bez konfliktu adresów. Daje to 128 niezależnych pinów I/O. Poniższa tabela przedstawia najważniejsze informacje na temat układu.

Tabela 2.2. Układ MCP23017.

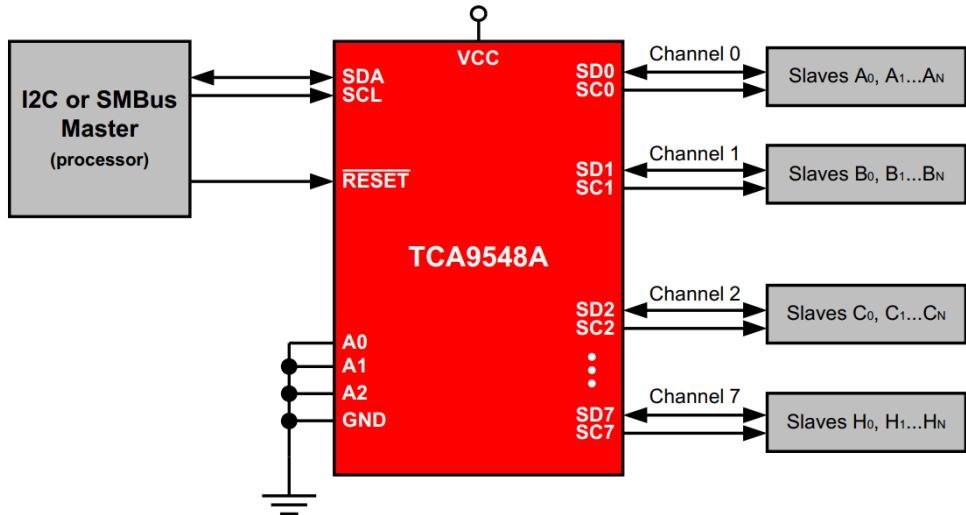
Napięcie zasilania	3.3 lub 5 V
Interfejs	I2C High-Speed
Ilość rozszerzeń	16 wyprowadzeń I/O
Dodatkowe informacje	Zewnętrzne przerwania INTA/INTB
Temperatura pracy	-40 – 80°C
Wymiary	38 x 23 mm

2.2.2 Multiplekser I2C

Zastosowanie ekspanderów wyprowadzeń znacznie uprościło proces implementacji systemu, lecz nie wyczerpało potrzeb projektu. Jak wspomniano wcześniej, do poprawnej pracy magistrali wymagane jest zastosowanie unikalnych adresów dla każdego urządzenia. Dzięki konfigurowalnym adresom układu MCP23017 można stosować maksymalnie 8 urządzeń na jednej linii I2C. Podobnie w przypadku stosowania dwóch lub więcej urządzeń, które nie mają konfigurowalnego adresu niezbędne staje się stosowanie układu, separującego kilka magistrali. Wykorzystany układ TCA9548A rozwiązuje wszystkie te problemy.

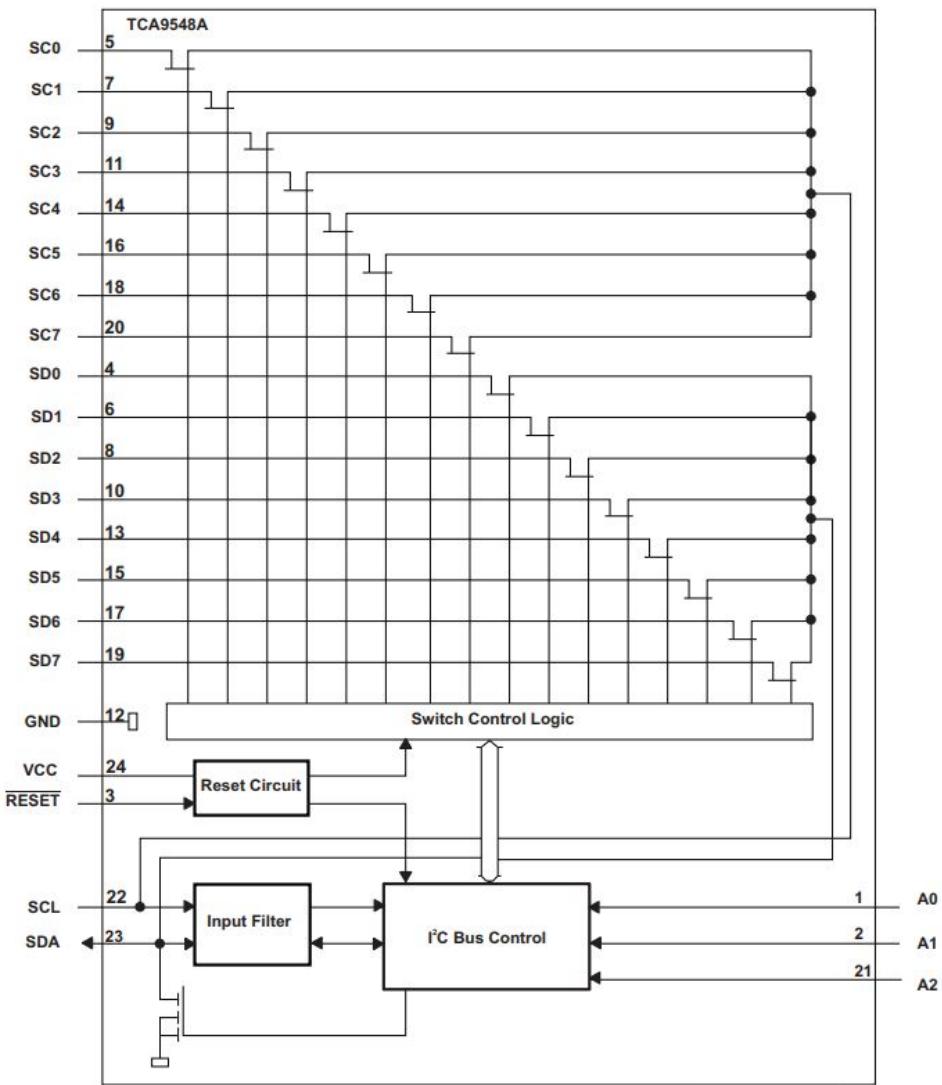


Rysunek 2.5. Multiplekser TCA9548A [6].



Rysunek 2.6. Schemat połączeń multipleksera [7].

Układ posiada 8 odseparowanych magistral cyfrowych I2C (kanałów). Komunikacja z wybraną magistralą odbywa się poprzez wysłanie komendy sterującej do multipleksera (wybór odpowiedniego kanału). Po tej operacji układ działa w trybie transparentnym, transferując dane z głównej magistrali do odpowiedniego kanału i odwrotnie. Możliwy jest wybór więcej niż jednego aktywnego kanału (dane wysłane zostaną na kilka kanałów jednocześnie). Przedstawiona poniżej grafika prezentuje schemat blokowy układu.



Rysunek 2.7. Schemat blokowy układu TCA9548A [7].

Tabela 2.3 zbiera najważniejsze informacje o module.

Tabela 2.3. Układ TCA9548A.

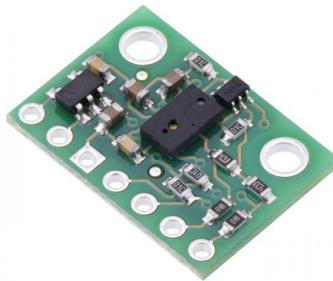
Napięcie zasilania	3.3 lub 5 V
Interfejs	I2C Full-Speed
Ilość kanałów	8
Temperatura pracy	-40 – 80°C
Wymiary	31 x 18 mm

Podobnie jak ekspander wyprowadzeń, układ ma możliwość zmiany adresu urządzenia. Dozwolony adres zawiera się w przedziale 0x70 – 0x77 (8 urządzeń), a co za tym idzie – istnieje możliwość stosowania 64 niezależnych kanałów I2C.

2.2.3 Realizacja inteligentnego parkingu

Ważną decyzją projektową był dobór czujników zajętości miejsc parkingowych. Na rynku istnieje wiele rozwiązań, na przykład czujniki magnetyczne, odbiciowe, ultradźwiękowe i optyczne. Istotną kwestią była konieczność pracy w różnych warunkach oświetleniowych oraz brak stosowania dodatkowych elementów (np. magnesów podczas użycia czujników magnetycznych). W projekcie postanowiono wykorzystać czujniki odległościowe bazujące na technice *time-of-flight*.

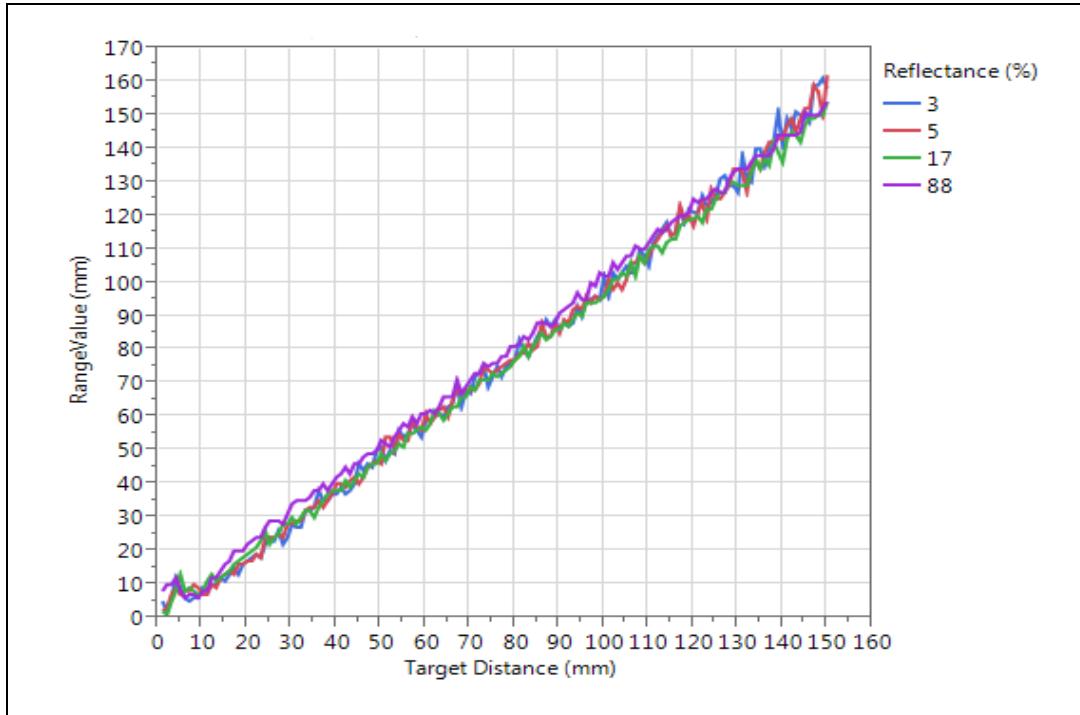
Zasada działania sensorów *time-of-flight* polega na wyemitowaniu wiązki lasera, a następnie pomiarze czasu, po którym sygnał powrócił do detektora. Rezultat jest następnie poddawany kompensacji, a następnie obliczana jest odległość od obiektu. Głównymi zaletami takiego rozwiązania są wysoka dokładność (rozdzielcość około 1 mm) oraz całkowita niezależność od warunków oświetleniowych. W projekcie wykorzystano moduł z układem VL6180x umożliwiający wiarygodny pomiar w zakresie 0 – 10 cm.



Rysunek 2.8. Moduł z układem VL6180x [8].

Wykorzystywane sensory mają możliwość pobrania aktualnej odległości w trybie sekwencyjnym, jak i ciągłym. W tym celu układ wyposażony jest w wewnętrzny układ zliczający, który aktualizuje odległość co ustalony czas (10 ms – 2.5 s), a następnie za pomocą przerwania systemowego informuje mikrokontroler o dostępności nowego odczytu. Możliwe jest zgłaszanie przerwań w zależności od przekroczonej granicznej odległości (minimalnej lub maksymalnej), co znacząco ułatwia pobranie krytycznych dla systemu odczytów. Układ wyposażony jest w moduł kalibracyjny. Dzięki niemu podczas inicjalizacji, jak również podczas pracy urządzenia możliwe jest ustalenie ilości odczytów, po których nastąpi samokalibracja układu, w celu zminimalizowania błędów i poprawienia rozdzielcości odczytów. Niewielkie znaczenie ma powierzchnia, od której odbija się laser. Zależność zmierzonej odległości od rodzaju powierzchni przedstawiona została poniżej.

Figure 6. Typical ranging performance



Rysunek 2.9. Zależność odległości od rodzaju powierzchni [8].

Jak widać na powyższym wykresie, niezależnie od rodzaju powierzchni istnieje praktycznie liniowa zależność między odczytem, a odległością faktyczną.

W przeciwieństwie do przedstawionych powyżej urządzeń I2C, układ VL6180x realizuje całkowicie programowe adresowanie urządzeń. Rozwiązuje to problem obecności wielu urządzeń na magistrali, lecz komplikuje proces inicjalizacji, gdyż po resecie układu wszystkie czujniki są skojarzone z tym samym adresem. Aby rozwiązać ten problem, producent przewidział dodatkowy pin, aktywujący układ po podaniu na niego stanu wysokiego. Dzięki temu proces inicjalizacji podzielony jest na kilka etapów:

1. Aktywowanie czujnika;
2. Oczekanie, aż układ będzie gotowy do użycia;
3. Inicjalizacja niezbędnych rejestrów;
4. Zmiana adresu urządzenia;

W projekcie wykorzystano monochromatyczny wyświetlacz OLED, który współpracując z inteligentnym parkingiem, umożliwia wyświetlanie informacji o jego stanie. Wykorzystany

wyświetlacz o przekątnej 1,3" i rozdzielcości 128 x 64 pikseli charakteryzuje się dobrą jasnością i niskim poborem prądu. Wykorzystuje sterownik SH1106.



Rysunek 2.10. Wyświetlacz OLED [9].

Tabela 2.4 prezentuje najważniejsze informacje na temat wyświetlacza.

Tabela 2.4. Sterownik OLED SH1106.

Napięcie zasilania	3.3 lub 5 V
Interfejs	I2C
Kąt widzenia	160°
Kolor znaków	Biały
Wymiary	35 x 33 mm



Rysunek 2.11. Budowa modelu parkingu.

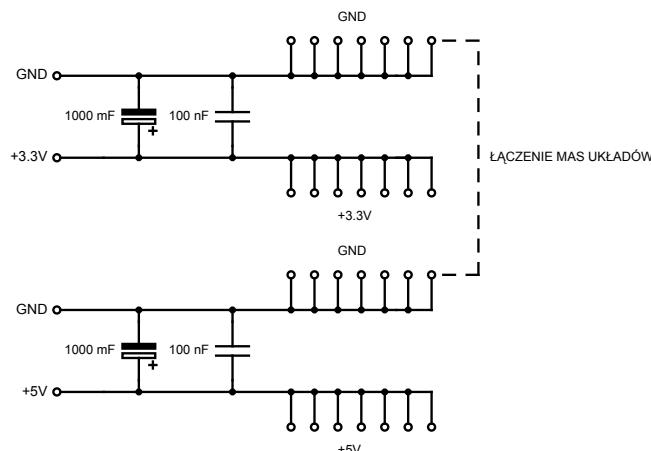
2.3 Moduł zasilający

Urządzenia składające się na warstwę sprzętową makiety wymagają różnego napięcia zasilania. Dotyczy to w szczególności czujników inteligentnego parkingu, które do poprawnej pracy wymagają napięcia nie wyższego, niż 3,3 V. W związku z tym, opracowano dwie sekcje zasilania, składające się z konektorów umożliwiających podłączenie zewnętrznych źródeł zasilania oraz wyprowadzeń i są wykorzystywane do doprowadzania zasilania do urządzeń. Używane poziomy napięć wraz z korzystających z nich urządzeń zaprezentowano w tabeli 2.5.

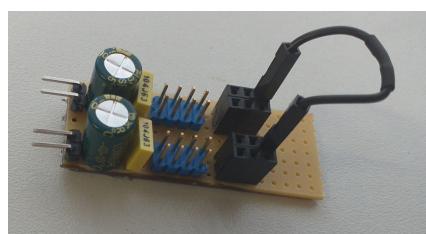
Tabela 2.5. Poziomy napięć wykorzystywane w projekcie.

Napięcie	Podłączone urządzenia
3,3 V	Czujniki inteligentnego parkingu, wyświetlacz OLED
5 V	Multiplekser I2C, ekspandery wyprowadzeń, zasilanie LED
18 V	Zasilanie animacji budynku kina

Na potrzeby zasilania urządzeń wykonany został moduł zasilający, którego dodatkową funkcją jest filtracja zakłóceń sieci poprzez zestaw kondensatorów odsprzęgających.



Rysunek 2.12. Schemat układu zasilającego.



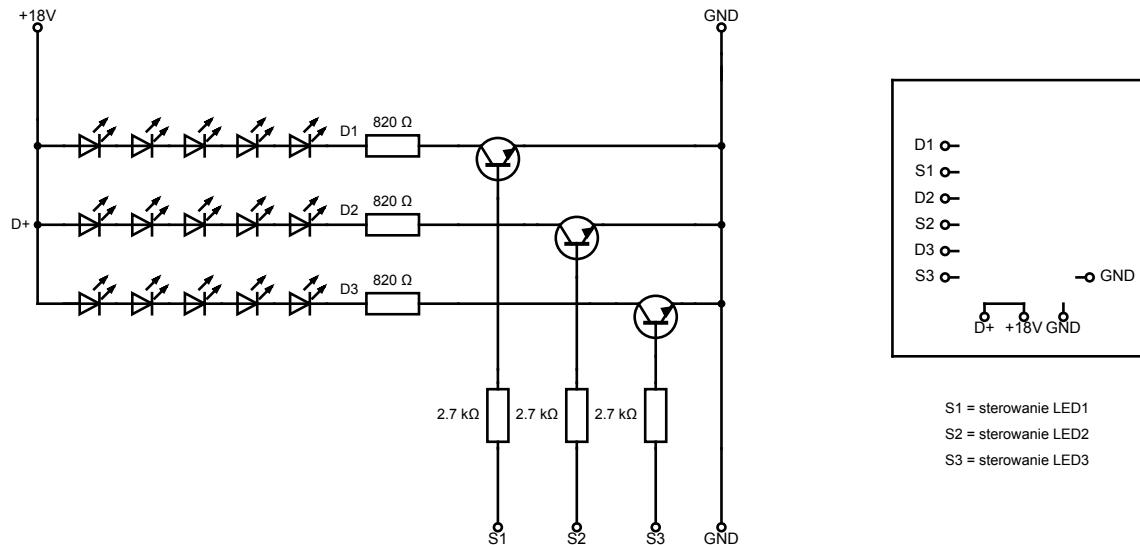
Rysunek 2.13. Układ zasilający (połączone masy).

2.4 Sprzętowa realizacja animacji

System sterowania korzysta z wielu animacji. Część z nich realizowana jest programowo, natomiast niektóre zrealizowano w oparciu o układy elektroniczne. Zaliczyć do nich można animacje neonu kina, czy oświetlenie turbiny wiatrowej.

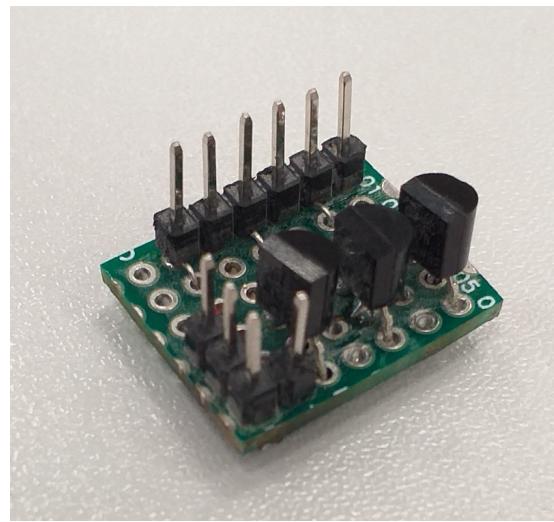
2.4.1 Kino Palace

Animowany neon budynku kina składa się trzech niezależnych obwodów, zawierających pięć połączonych szeregowo LED. Umożliwia to wyświetlenie trzech różnych klatek animacji. Zważywszy na fakt, że wymagane napięcie pracy pojedynczej diody wynosi około 2,8 V zastosowano zewnętrzne źródło zasilania 18 V. Wykorzystany układ tranzystorowy umożliwia bezpieczne sterowanie obwodami bezpośrednio z wyprowadzeń ekspandera.



Rysunek 2.14. Schemat elektryczny animacji Kina Palace.

Wykorzystane tranzystory NPN typu BC547 łączą obwód diod poprzez podanie na wyprowadzenie bazy sygnału wysokiego. Na rysunkach 2.15 oraz 2.16 zaprezentowano wykonany układ wraz z działaniem animacji (pojedynczą klatką).



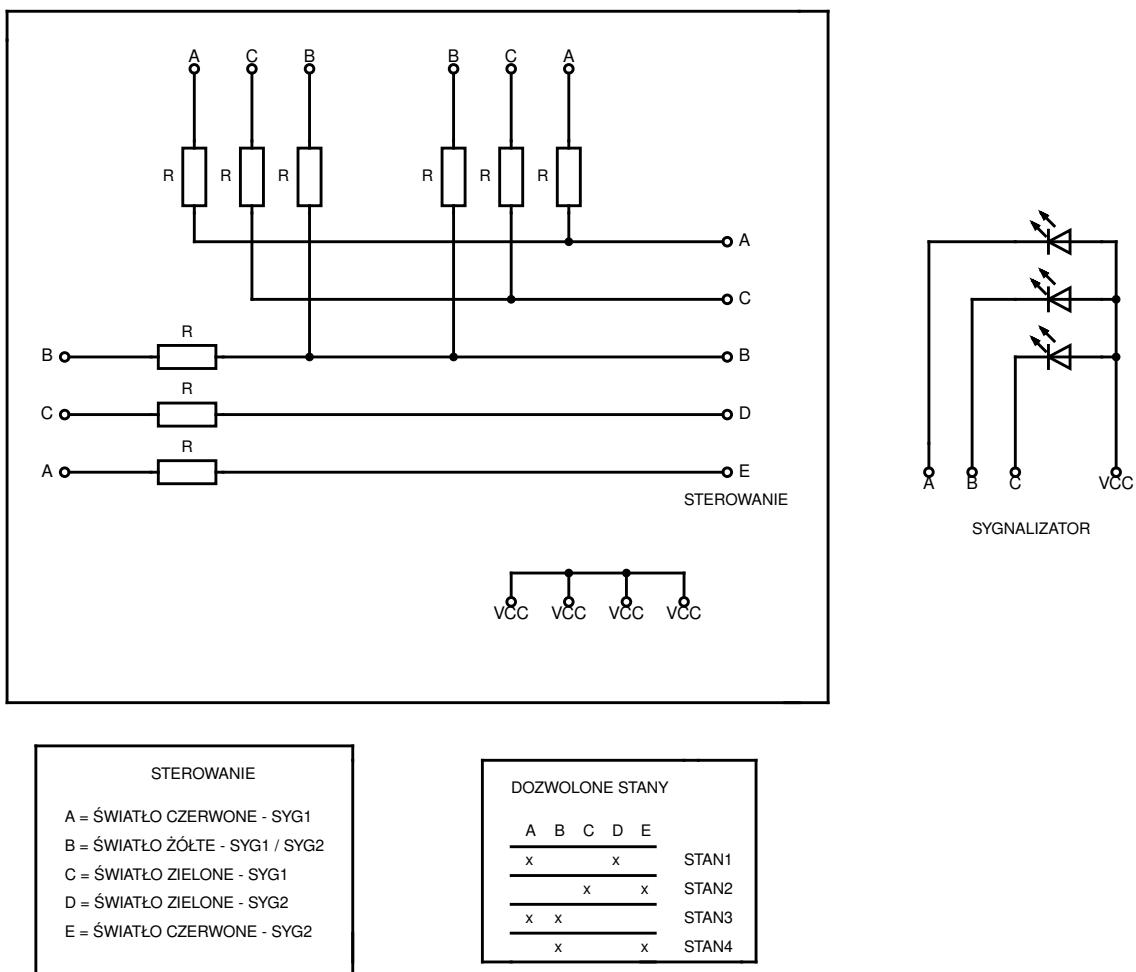
Rysunek 2.15. Układ animacji kina.



Rysunek 2.16. Działanie animacji kina.

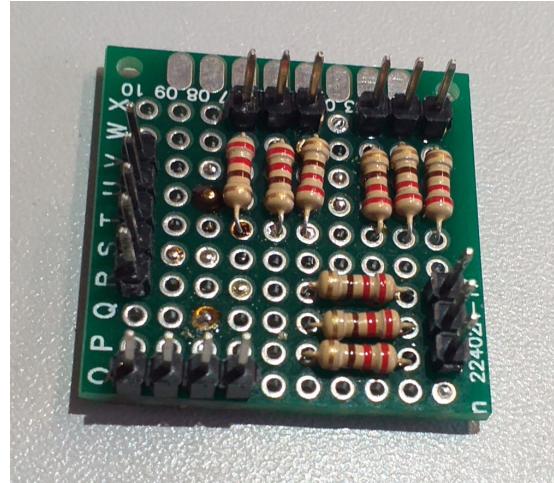
2.4.2 Sygnalizacja świetlna

Obsługa sygnalizacji świetlnej wymaga pięciu kombinacji stanów wyprowadzeń, które uruchamiają określoną sekwencję oświetlenia. W tym celu opracowano dwa układy rezystorowe obsługujące dwa niezależne skrzyżowania. Pojedynczy moduł składający się z odpowiednio połączonych rezystorów oraz wyprowadzeń umożliwia obsługę sygnalizatorów dla dwóch niezależnych kierunków ruchu.



Rysunek 2.17. Schemat elektryczny animacji sygnalizacji świetlnej.

Zamiana pinów sterujących ACBDE na EDBCA (odwrócenie kolejności) powoduje zmianę kolejności sekwencji świateł. W ten sposób, w danym momencie niezależne skrzyżowania mają różne stany sygnalizacji. Poniższe rysunki prezentują budowę układu wraz z działaniem animacji.



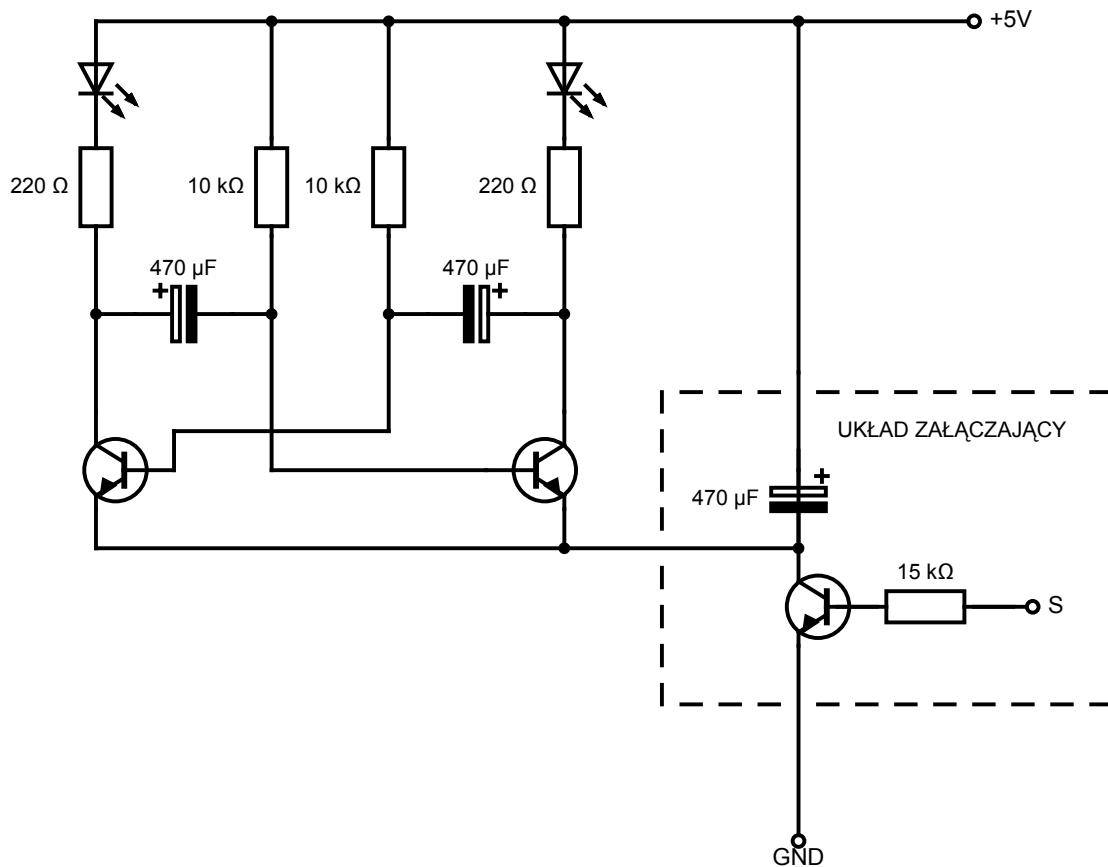
Rysunek 2.18. Moduł sygnalizacji świetlnej.



Rysunek 2.19. Działanie sygnalizacji świetlnej.

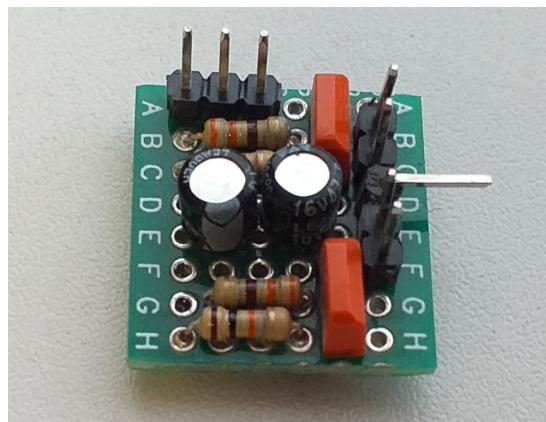
2.4.3 Turbina wiatrowa

Zastosowanie układu multiwibratora astabilnego umożliwia sprzętową realizację animacji naprzemiennie świecących LED. Wykorzystano tranzystory NPN – załączenie układu odbywa się poprzez podanie stanu wysokiego na bazę dodatkowego tranzystora sterującego (układ załączający).



Rysunek 2.20. Schemat elektryczny animacji turbiny wiatrowej.

Częstotliwość migania diod, która zależna jest od wielkości wykorzystanych kondensatorów oraz rezystorów wynosi około trzy sekundy. Rysunki 2.21 oraz 2.22 prezentują budowę układu migającego wraz z jego działaniem.



Rysunek 2.21. Układ multiwibratora turbiny wiatrowej.



Rysunek 2.22. Działanie turbiny wiatrowej.

2.4.4 Roller Coaster

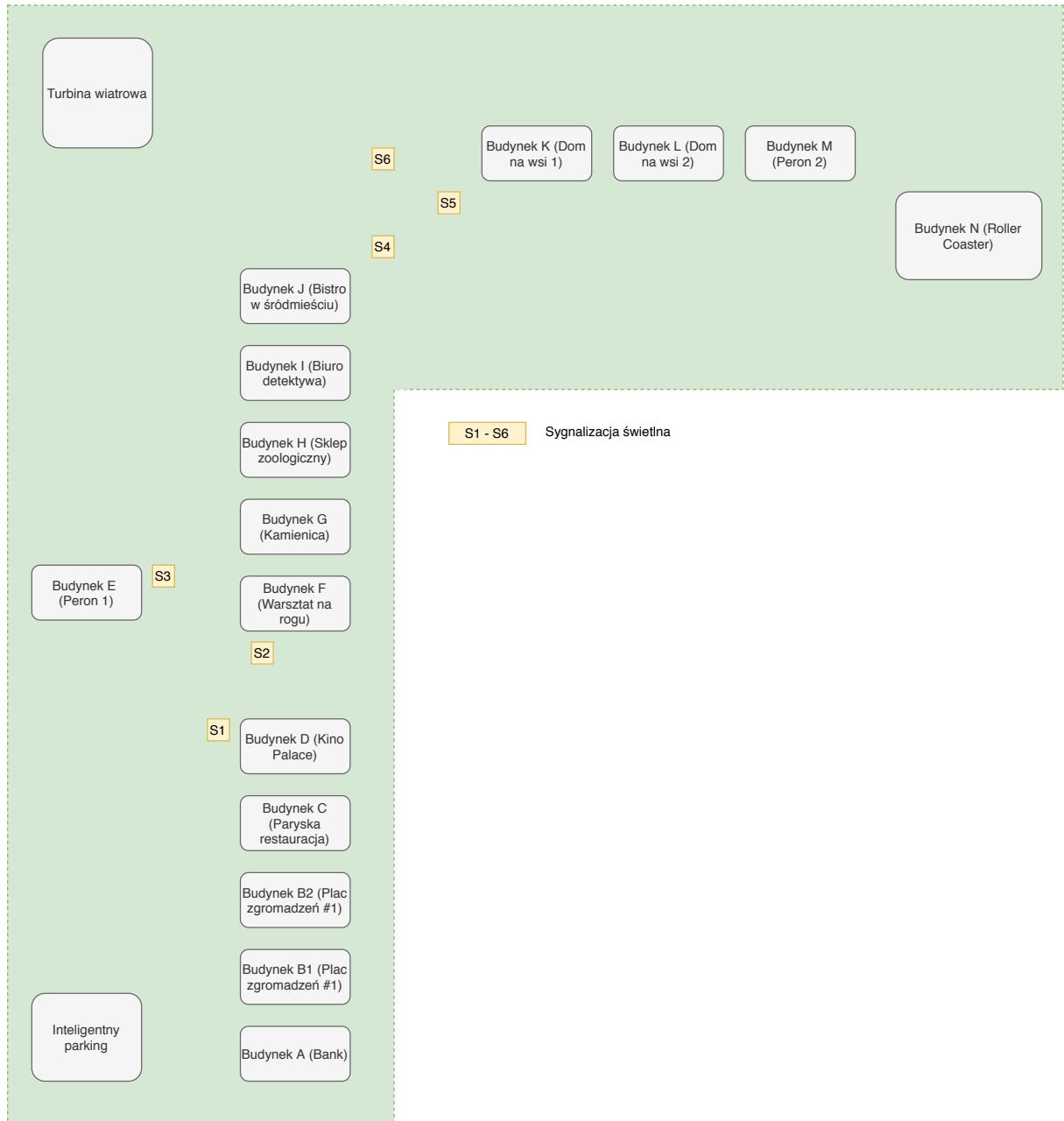
Obsługa animacji kolejki górskiej wymaga dziesięciu kombinacji stanów wyprowadzeń, które odpowiadają niezależnym kolumnom budynku. Równoległe połączenie diod w każdym filarze umożliwia bezpośrednie zasilanie obwodu z wyprowadzeń ekspandera oraz brak konieczności stosowania zewnętrznego układu sterującego.



Rysunek 2.23. Model kolejki górskiej.

2.5 Rzut poziomy makiety

Rysunek 2.24 przedstawia widok makiety z wyszczególnionymi elementami w niej występującymi.



Rysunek 2.24. Rzut poziomy makiety.

Implementacja oprogramowania

W rozdziale tym omówiono sposób implementacji poszczególnych modułów systemu, konfiguracji systemu czasu rzeczywistego oraz sposoby sterowania makietą. Ponadto podano w nim informacje o strukturach projektu i funkcjach bibliotecznych.

3.1 Struktura projektu

Struktura projektu wymuszona została przez środowisko *MCUXpresso IDE*. W związku z ciągłym jego rozwojem, a także ulepszaniem zestawu narzędzi i bibliotek całość plików nagłówkowych oraz źródłowych zgromadzona jest we wspólnym katalogu *source*. Zapewnia to całkowitą niezależność od wersji środowiska. Pliki projektu podzielone zostały na szereg podkatalogów, grupujących je według przeznaczenia:

Tabela 3.1. Struktura projektu.

Katalog	Opis
board	Inicjalizacja i obsługa płyty ewaluacyjnej
drivers	Niskopoziomowe sterowniki modułów I2C
gui	Graficzny interfejs użytkownika
httpsrv	Obsługa serwera WWW
lcd	Sterowanie wyświetlaczem
lego	System sterowania makietą
rtos	Ustawienia systemu <i>FreeRTOS</i>
sdcard	Obsługa kart SD
settings	Ustawienia systemu sterowania
utils	Zewnętrzne biblioteki oraz pliki pomocnicze

Ponadto starano się, aby wszelkie typy danych oraz nagłówki funkcji opatrzone były odpowiednim przedrostkiem, przyporządkowanego je do określonej kategorii. Format ten używa nazwy podkatalogu, złożonego z wielkich liter oraz właściwej nazwy elementu, np.

3.2 Inicjalizacja systemu

Niezbędnym procesem, koniecznym do poprawnej pracy systemu jest inicjalizacja wszelkich modułów oraz urządzeń. Proces ten podzielono na następujące po sobie etapy, których kolejność podyktywana jest zależnością jednych modułów od drugich. Cały proces przedstawiony został poniżej.

1. Sprawdzenie, czy w slocie znajduje się karta SD oraz jej inicjalizacja. Import ustawień systemu;
2. Inicjalizacja serwera HTTP;
3. Inicjalizacja urządzeń I2C;
4. Stworzenie zadań sterowania makietą (RTOS);
5. Utworzenie wątków interfejsu użytkownika oraz sterowania wyświetlaczem;
6. Przekazanie sterowania do poszczególnych zadań systemu;

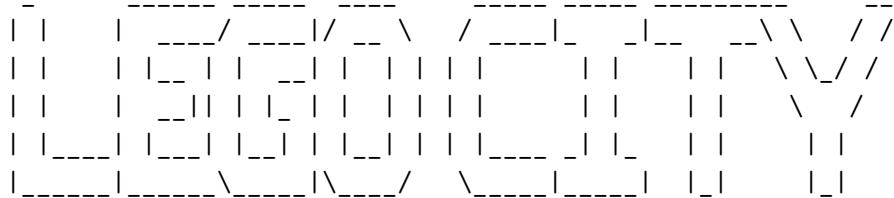
W trakcie procesu inicjalizacji odpowiednie etapy prezentowane są na ekranie LCD płyty prototypowej. Ponadto rozszerzone informacje, ostrzeżenia oraz błędy możliwe są do śledzenia w konsoli debuggera, wbudowanej w środowisko *MCUXpresso IDE*. Poniżej przedstawiono zrzut konsoli po poprawnej inicjalizacji wszystkich modułów systemu:

```
Initializing SD card
SD card not found
Initializing HTTP server
Initializing PHY...

HTTP Server configuration:
IPv4      : 192.168.0.105
Subnet mask : 255.255.255.0
Gateway    : 192.168.0.1
DNS        : lego

Performing LEGO startup. Initializing I2C devices
VL6180x device #0 OK
VL6180x device #1 OK
VL6180x device #2 OK
VL6180x device #3 OK
VL6180x device #4 OK
VL6180x device #5 OK
```

```
VL6180x device #6 OK
VL6180x device #7 OK
VL6180x device #8 OK
VL6180x device #9 OK
Initializing LEGO RTOS
```



3.3 System operacyjny czasu rzeczywistego

Zadaniem systemu czasu rzeczywistego jest spełnianie wymagań dotyczących narzuconego czasu wykonywania określonych operacji programu. Za pomocą wbudowanych mechanizmów zapewniona jest pełna synchronizacja zadań, wywłaszczenie mniej krytycznych wątków oraz asynchroniczne ich wykonywanie bazując na rozbudowanym mechanizmie zdarzeń. W projekcie wykorzystano odmianę popularnego systemu czasu rzeczywistego *Amazon FreeRTOS*. System ten nie modyfikuje bazowego jądra systemu *FreeRTOS*, a jedynie wspomaga implementowanie aplikacji IoT i pochodnych. Wsparcie dla architektur oraz wielu możliwości konfiguracyjnych zapewnione jest dzięki technice komplikacji warunkowej. Poniżej przedstawiono najważniejsze, z punktu widzenia problemu projektu ustawienia oraz ich konsekwencje.

3.3.1 configUSE_TIME_SLICING

Włączenie makra umożliwia dzielenie czasu procesora w zadaniach o takim samym priorytecie, a co za tym idzie równolegle ich wykonywanie. Dzięki temu mniej krytyczne zadania wykonywane są szybciej. Istotną wadą takiego rozwiązania jest konieczność stosowania dodatkowych mechanizmów, w sytuacji kiedy równolegle wykonywane zadanie powinno na pewnym etapie wykonać się bez zwłoki czasowej. W tym celu w programie wykorzystano *sekcje krytyczne*. Sekcja krytyczna jest wydzielonym fragmentem programu, który blokuje działanie algorytmu przełączania kontekstu, a co za tym idzie przekazuje cały czas procesora dla konkretnego zadania. Implementowana jest poprzez umieszczenie instrukcji programu pomiędzy wywołaniem jej rozpoczęcia i zakończenia.

Listing 3.1: Sekcja krytyczna systemu *FreeRTOS*

```
taskENTER_CRITICAL();  
/* Critical code */  
taskEXIT_CRITICAL();
```

3.3.2 configUSE_PREEMPTION

Istotą działania systemów operacyjnych czasu rzeczywistego jest przyznawanie pierwszeństwa krytycznym zadaniom. Ustawienie tego makra sprawia, że algorytm przełączania wątków zablokuje wykonywanie zadań o niższym priorytecie, jeśli którykolwiek z zadań o priorytecie wyższym będzie gotowe do działania. Proces ten określany jest mianem wywłaszczenia.

3.3.3 configFRTOS_MEMORY_SCHEME

FreeRTOS wspiera wiele schematów zarządzania pamięcią, aby maksymalnie dostosować się do używanej architektury. Są to m.in.: alokacja statyczna oraz dynamiczna, wirtualne łączenie niezależnych fragmentów pamięci. W projekcie wykorzystano wersję używającą standardowych mechanizmów alokacji dynamicznej pamięci na stercie. W związku z faktem, iż płyta ewaluacyjna dysponuje ograniczoną ilością wewnętrznej pamięci operacyjnej, a każde zadanie potrzebuje własnego stasu na dane, zdecydowano o zmianie standardowej mapy pamięci. W tym celu podzielono domyślny blok pamięci *BOARD_SDRAM* na dwa obszary. Mapę pamięci po modyfikacjach zaprezentowano w tabeli 3.2.

Tabela 3.2. Mapa pamięci.

Nazwa	Opis	Wielkość
PROGRAM_FLASH	Program	Domyślna
BOARD_FLASH	Pamięć FLASH zewnętrzna	Domyślna
SRAM_UPPER	Pamięć RAM	Domyślna
SRAMX	Pamięć RAM	Domyślna
USB_RAM	Pamięć RAM dla USB	Domyślna
BOARD_SDRAM	Zewnętrzna pamięć RAM	8 MB
SDRAM_HEAP	Pamięć sterty	8 MB

Konieczna była zmiana sposobu rozmieszczenia elementów w pamięci. W tym celu ustalonio lokalizację sterty w zewnętrznej pamięci *SDRAM_HEAP* zamiast w wewnętrznej pamięci operacyjnej.

3.3.4 Zadania systemu sterowania

W procesie określania wymagań systemu wprowadzono szereg wykonywanych zadań. Wprowadzenie ich do systemu wymusiło zastosowanie hierarchicznej struktury związanej z ich priorytetami. Warto wspomnieć, że wyższa liczba priorytetu oznacza, że zadanie jest ważniejsze niż pozostałe. Tabela 3.3 przedstawia zadania systemu sterowania uporządkowane od najważniejszego (najbardziej krytycznego) do najmniej ważnego.

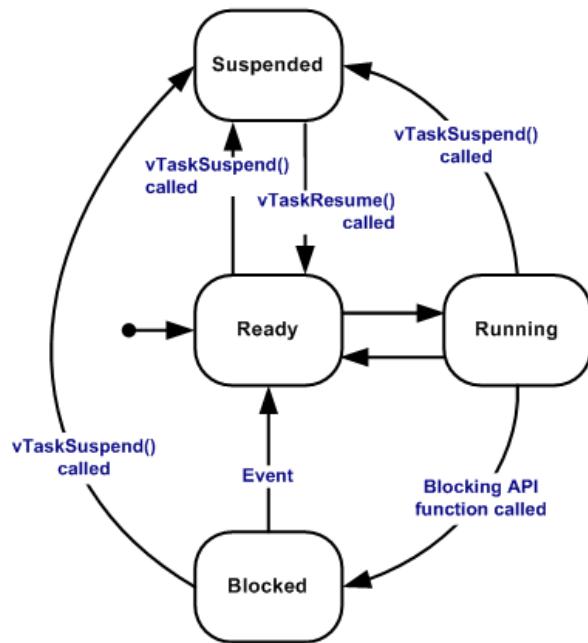
Tabela 3.3. Wątki RTOS.

Identyfikator	Opis	Priorytet
tcpip_thread	Obsługa portu Ethernet	8
HTTP server	Obsługa serwera HTTP	3
TASK-COASTER	Obsługa animacji modelu Kolejki górskiej	3
TASK-CINEMA	Obsługa animacji modelu Kina	3
TASK-TRAFFIC	Sterowanie sygnalizacją uliczną	3
TASK-LEGO-PARKING	Sterowanie inteligentnym parkingiem	2
TASK-LEGO-AUTO	Sterowanie automatyczne makietą	2
TASK-BACKLIGHT	Sterowanie podświetleniem ekranu	2
TASK-TOUCH	Obsługa ekranu dotykowego	1
TASK-GUI	Obsługa graficznego interfejsu użytkownika	1
IDLE	Wątek bezczynności	0

Zadania obsługi portu Ethernet wraz ze stosem TCP/IP mają najwyższe priorytety. Zapewnia to szybką komunikację klient-serwer. Ogół zadań sterowania makietą ma średni priorytet, a obsługa wyświetlacza najmniejszy z racji rzadkiego jego wykorzystywania (operacje serwisowe, zmiana ustawień). Ponadto wątki serwera oraz wyświetlacza są w pełni asynchroniczne, co oznacza, że są uruchamiane jedynie w momencie konieczności i nie zużywają niepotrzebnie czasu procesora. Jest to realizowane poprzez wprowadzenie trzech stanów, w których może znaleźć się zadanie:

1. aktywny (ang. *running*) - zadanie aktualnie wykonywane;
2. gotowy (ang. *ready*) - oczekiwanie na przydzielienie czasu procesora;
3. wstrzymany (ang. *suspended*) - zadanie wyłączone lub oczekujące na pojawienie się zdarzenia wybudzającego;

Proces *IDLE* jest tzw. procesem bezczynności. Algorytm zmiany kontekstu uruchamia go w momencie, kiedy żadne inne zadanie nie jest gotowe do wykonywania. Schemat przełączania zadań zaprezentowano na rysunku 3.1



Rysunek 3.1. Schemat przełączania zadań systemu FreeRTOS [10].

Szczegóły dotyczące poszczególnych zadań w systemie opisane zostaną w kolejnych podrozdziałach.

3.4 Obsługa kart SD

Istotnym problemem, wynikającym z używanej platformy sprzętowej był magazyn danych, który zapewniałby nieulotną pamięć na przechowywane dane oraz ustawienia. W tym celu wykorzystano wbudowany slot kart SD (ang. *Secure Digital*). Obsługuje on zarówno pełnowymiarowe wersje, jak również korzystając z adaptera, nowsze standardy kart, takie jak *mini SD* lub *micro SD*. Zaimplementowane mechanizmy dostępu do danych oraz ich modyfikacji umożliwiają:

1. szybki dostęp do danych;
2. importowanie oraz eksportowanie ustawień podczas działania programu;
3. resetowanie ustawień do ich domyślnych wartości (tzw. ustawienia fabryczne);

Warto zaznaczyć, że obecność karty SD nie jest wymagana do poprawnej pracy systemu (możliwość używania ustawień domyślnych), lecz do poprawnej pracy wymagane jest umieszczenie jej w slocie przed procesorem inicjalizacji programu.

3.4.1 Format danych

Wszelkie dostępne ustawienia systemu sterowania przechowywane są we wspólnej strukturze, której odpowiednie pola dotyczą pojedynczych wartości konfiguracyjnych. Stwarza to możliwość korzystania ze swoistej, uproszczonej bazy danych. Na etapie inicjalizacji, poszczególne pola otrzymują domyślną wartość, która może być odczytana lub zmieniona przez system lub na żądanie użytkownika.

Listing 3.2: Wartości domyślne ustawień

```
/* Default settings. They will never changed during runtime */
static const SETTINGS_t defaultSettings = {
    /* Http server */
    .httpsrvIp = CONV_IP4_TO_UINT32(192, 168, 0, 105),
    .httpsrvSm = CONV_IP4_TO_UINT32(255, 255, 255, 0),
    .httpsrvGw = CONV_IP4_TO_UINT32(192, 168, 0, 1),
    .httpsrvDnsName = "lego",
    /* Display */
    .lcdBrightness = 80,
    .lcdDimmingTime = LCD_DIMMING_10S
};
```

Możliwe jest szybkie zresetowanie ustawień do domyślnych wartości, używając dedykowanych funkcji bibliotecznych. Warto zaznaczyć, że sam proces zapisu ustawień nie opakowuje w żaden sposób surowych danych struktury, a kopiuje jedynie określone bajty struktury do zewnętrznej pamięci. Motywacją do tego było uproszczenie modelu zapisu i odczytu danych w celu szybszego wykonywania programu. Istotną konsekwencją tego wyboru jest brak możliwości edycji ustawień z poziomu innego, niż system sterowania.

3.4.2 FatFS

Dostęp do pamięci karty SD realizowany jest przez bibliotekę *FatFs*. Umożliwia ona sam proces montowania dysku, jak również wszystkie operacje na plikach i katalogach. Do poprawnego działania karta SD musi być sformatowana na system plików *FAT32*. Ustawienia systemu przechowywane są w pojedynczym pliku, którego nazwa może być dowolnie zmieniona podczas komplikacji programu. Podczas etapu inicjalizacji wykonywane są następujące kroki:

1. sprawdzenie, czy w slocie znajduje się karta SD;
2. zamontowanie systemu plików;
3. odczyt pliku konfiguracyjnego;

4. sprawdzenie poprawności odczytanych danych;
5. skopiowanie danych do pamięci operacyjnej;

Jeśli którykolwiek z etapów nie zakończy się poprawnie, dostęp do pamięci karty zostaje zablokowany oraz wczytywane są ustawienia domyślne. System plików pozostaje zamontowany na czas działania programu, umożliwiając tym samym szybki dostęp do danych. Proces eksportowania ustawień wykonywany jest analogicznie. Może być wykonany w dowolnym momencie działania programu. Wykonywane są następujące kroki:

1. sprawdzenie, czy w slocie umieszczona jest karta;
2. próba dostępu do zamontowanego systemu plików;
3. zapis ustawień do pliku konfiguracyjnego (a w przypadku jego braku stworzenie nowego);
4. sprawdzenie statusu operacji i zwrócenie jej wyniku;

Niepowodzenie którejkolwiek z powyższych operacji kończy proces zapisu do karty SD. Odpowiednie informacje diagnostyczne udostępnione są w konsoli debuggera.

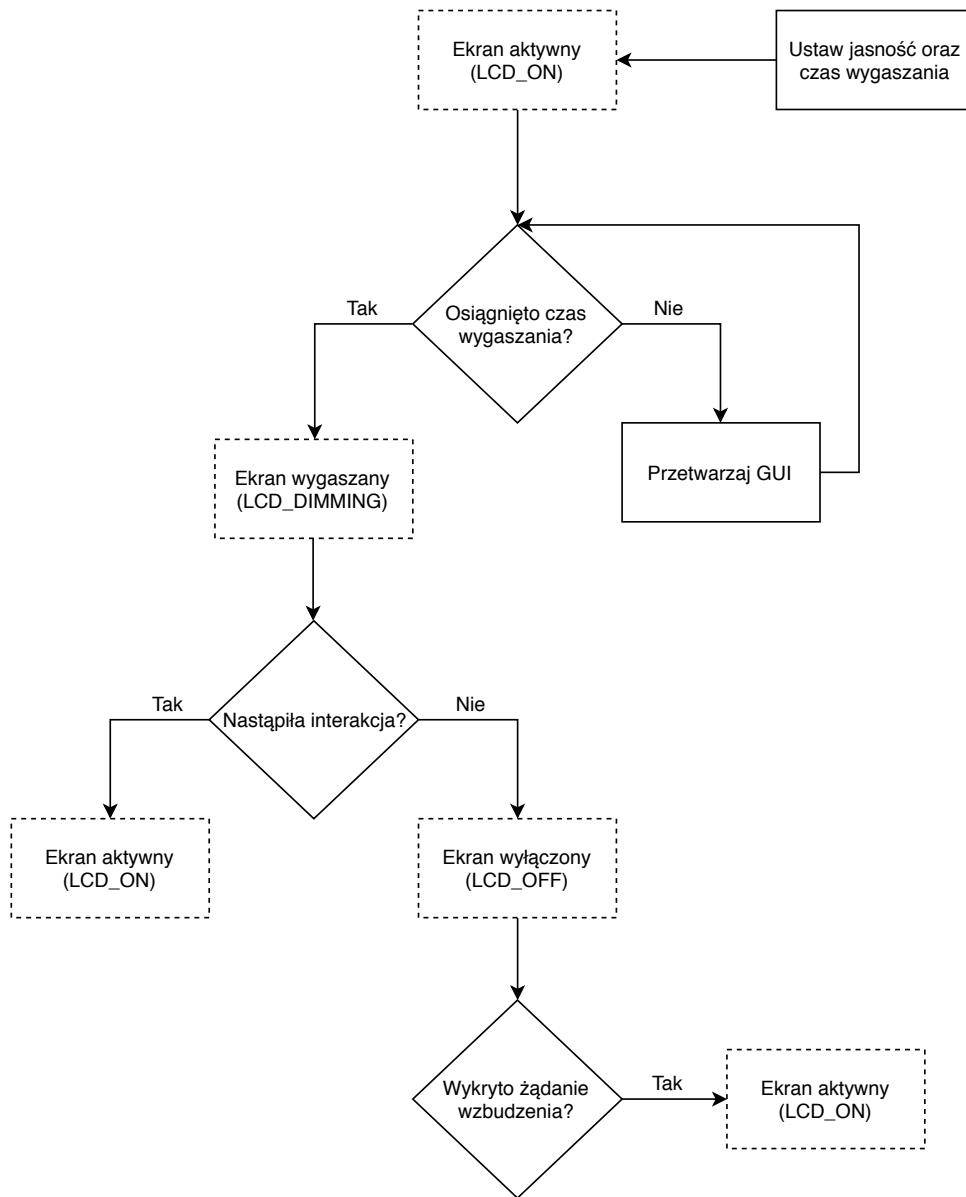
3.5 Sterowanie wyświetlaczem

Zadaniem wyświetlacza jest interakcja z użytkownikiem systemu. Jest to realizowane poprzez:

1. udostępnianie graficznego interfejsu (ang. *Graphical User Interface, GUI*);
2. wprowadzanie informacji (np. ustawień) do systemu za pomocą panelu dotykowego;

Warto wspomnieć, że sam wyświetlacz nie jest konieczny do poprawnej pracy systemu, a co za tym idzie może być aktywowany na żądanie użytkownika. Brak przetwarzania graficznego interfejsu skraca czas wykonania programu oraz zapewnia mniejszy pobór energii układu. Możliwy jest dobór jego jasności, aby jak najlepiej dostosować go do panujących warunków otoczenia. W tym celu wprowadzono trzy stany jednoznacznie identyfikujące aktualny stan działania ekranu. Stan *LCD_ON* oznacza, że wyświetlacz jest włączony oraz gotowy na interakcję z użytkownikiem poprzez wprowadzanie koordynat ekranu dotykowego. W tym trybie możliwa do wyboru jest jego jasność oraz czas oczekiwania, po którym, jeśli nie zanotowano aktywności system otrzyma żądanie jego wygaszenia. Podczas procesu wygaszania wyświetlacz przechodzi w stan *LCD_DIMMING*, w którym panel dotykowy nie przekazuje informacji do systemu, lecz umożliwia jego szybkie wzbudzenie poprzez jednokrotne dotknięcie ekranu. Stan ten wyróżnia ograniczenie jasności do 50% ustalonej

wartości. Po procesie wygaszenia wyświetlacz przechodzi w stan *LCD_OFF*, w którym przetwarzanie interfejsu jest zablokowane, a podświetlenie ekranu zostaje całkowicie wyłączone. Jednokrotna interakcja (przyciśnięcie lub przytrzymanie ekranu dotykowego) wzbudza ekran, co jest jednoznaczne z jego aktywowaniem i gotowością na interakcję z użytkownikiem. Uproszczony schemat tego procesu zaprezentowany został na poniższym schemacie.



Rysunek 3.2. Algorytm sterowania wyświetlaczem LCD

3.6 Interfejs użytkownika

Głównym zadaniem interfejsu użytkownika jest prezentowanie informacji o stanie systemu oraz umożliwienie jego modyfikacji. W projekcie wykorzystano bibliotekę dedykowaną dla

systemów wbudowanych – *emWin*. Jej głównymi zaletami są zadowalająca wydajność oraz mnogość mechanizmów usprawniających implementowanie zaawansowanych interfejsów graficznych. Podobnie jak w przypadku popularnych bibliotek podstawowym obiektem graficznym jest *okno* (widget). Biblioteka działa w sposób asynchroniczny, reagując wtedy, kiedy pojawi się komunikat wymagający zmianę stanu lub odświeżenia okna roboczego. Realizowane jest to poprzez mechanizm wywołań zwrotnych (ang. *callback*). Na przykład:

Listing 3.3: Mechanizm wywołania zwrotnego okna roboczego

```
static void VK_Callback(WM_MESSAGE *pMsg)
{
    switch (pMsg->MsgId) {
    case WM_PAINT:
        /* Window repaint */
        break;
    case WM_NOTIFY_PARENT:
        /* Child window status changed */
        break;
    }
}
```

3.6.1 Okna dialogowe

Często zachodzi sytuacja, w której przetwarzanie programu powinno zostać wstrzymane, z powodu wymuszenia zaistnienia określonej akcji w interfejsie. W tym celu wykorzystano mechanizm *okien dialogowych*. Z punktu widzenia użytkownika okno dialogowe dziedziczy wszelkie właściwości zwykłego okna, lecz przetwarzanie GUI zostaje całkowicie wstrzymane, aż do momentu wprowadzenia danych lub potwierdzenia wykonania akcji. Wykorzystane w systemie okna dialogowe są również modalne – blokują wysyłanie zdarzeń do innych widocznych okien interfejsu.

Istotnym wymogiem stawianym przez wykorzystywane środowisko graficzne jest konieczność uruchamiania okien dialogowych na tym samym poziomie programowym, co wywoływanie instancji menadżera okien [2]. W projekcie wykorzystano do tego wątek menadżera okien, który oprócz przetwarzania GUI, na podstawie odczytanych notyfikacji realizuje niezbędne wywołania okien.

Listing 3.4: Algorytm otwierania okien dialogowych

```
static void GUI_MainTask(void *pvParameters)
{
    /* Blocking dialog details */
    GUI_BlockingDialogInfo_t *blockingDialogInfo;
```

```

while (1) {
    /* Dialog window required */
    if (xQueueReceive(blockingDialogQueue, &blockingDialogInfo, 0) == pdPASS) {
        switch (blockingDialogInfo->dialog) {
            case DIALOG_VK: { /* Virtual keyboard should be opened */
                /* Get dialog details */
                const VK_Parms_t *params = (const VK_Parms_t *)blockingDialogInfo->data;
                /* Create dialog window */
                VK_InputStatus_t res = VK_GetInput(params);
                /* Feedback message instance */
                WM_MESSAGE feedback;
                /* Dialog window closed - notify last window */
                WM_SendMessage(blockingDialogInfo->srcWin, &feedback);
                break;
            }
        }
    }

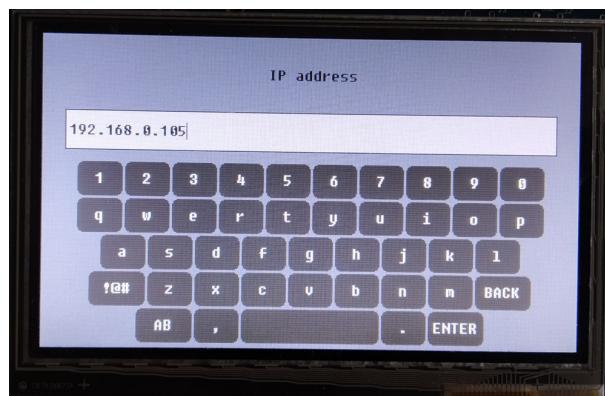
    /* Execute window manager instance */
    WM_Exec();
}

```

W następnych podrozdziałach szczegółowo opisane zostaną okna dialogowe zaimplementowane na potrzeby systemu sterowania.

3.6.2 Klawiatura ekranowa

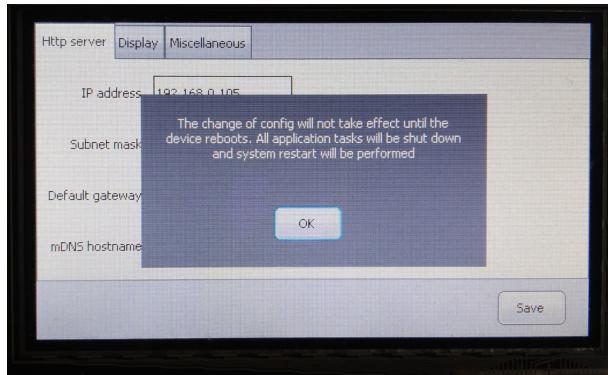
Umożliwienie wprowadzania informacji do systemu realizowane jest poprzez klawiaturę ekranową. Dostępne jest zarówno wprowadzanie liter, jak i cyfr oraz znaków specjalnych. Podczas uruchamiania okna przekazywany jest uchwyt do edytowanego okna (widgetu). Wprowadzanie danych może być oparte o istniejącą wartość lub rozpoczęte od nowa. Możliwe jest przekazanie funkcji walidującej wprowadzone dane, która zostanie uruchomiona podczas zamknięcia okna – zmiana aktualnej wartości zostanie zablokowana, gdy nowa nie spełnia oczekiwanych kryteriów.



Rysunek 3.3. Widok klawiatury ekranowej.

3.6.3 Okno potwierdzenia

Obsługa krytycznych operacji często wymaga potwierdzenia przez użytkownika. Wymóg ten realizowany jest poprzez okno potwierdzenia. Pierwszą jego odmianą jest okno zwracające do systemu wynik wyboru. W tym celu wraz z komunikatem wyświetlane są dwa przyciski realizujące odpowiedzi “potwierdź” lub “zrezygnuj”. Wynik wyboru zwracany jest do systemu, gdzie na jego podstawie wykonywane są odpowiednie instrukcje. Drugą odmianą jest wyświetlenie komunikatu, który pełni jedynie rolę informacyjną i może zostać zamknięty przyciskiem odblokowującym działanie systemu.



Rysunek 3.4. Widok okna potwierdzenia.

3.7 Serwer WWW

Wbudowany serwer WWW zapewnia warstwę komunikacyjną pomiędzy systemem sterowania makiety, a jej użytkownikiem. Dzięki zastosowaniu protokołu komunikacyjnego HTTP zaimplementowano uniwersalne i intuicyjne rozwiązanie z którego korzystać mogą różne warstwy programowe, w tym głównie przeglądarki internetowe.

3.7.1 Żądania HTTP

Protokół HTTP działa w trybie żądanie-odpowiedź (ang. *request-response*). Żądania generowane przez klienta są nasłuchiwanie oraz przetwarzane przez wyspecjalizowane części oprogramowania serwera, następnie zwracana jest odpowiedź określonego typu. Klientem może być zarówno przeglądarka internetowa, jak i zewnętrzna aplikacja lub jej część [11]. W celu rozróżnienia intencji klienta wprowadzono szereg metod, identyfikujących typ żądania, m.in.:

- GET – pobierz dane z serwera;
- POST – wyślij dane do serwera (tworzenie, aktualizacja zasobów);

- PUT – zamień zasoby na serwerze;
- DELETE – usuń zasoby na serwerze;

W projekcie użyte zostały dwie pierwsze metody żądań. Motywacją do tego była ich uniwersalność oraz implementacja we wszystkich popularnych przeglądarkach internetowych.

3.7.2 Format wymiany danych JSON

Komunikacja klient-serwer zaimplementowana została przy pomocy popularnego modelu JSON (ang. *JavaScript Object Notation*). Sam standard jest całkowicie niezależny od konkretnego języka programowania, a jego prostota i łatwość organizacji czynią go idealnym formatem wymiany danych. Wiadomość zawiera zbiór par *klucz:wartość*. Możliwe jest tworzenie list wartości. Przykładowy komunikat wymiany informacji zaprezentowano poniżej:

```
{
  "status": 200,
  "animStatus": 1
}
```

3.7.3 Pobieranie danych (GET)

Metoda pobierania danych zaimplementowana została w celu możliwości odpytania serwera o stan działania części programu. Umożliwia pobranie informacji na temat określonych zasobów (na przykład stanu parkingu, aktywności oświetlenia, itp.). Została opracowana tak, aby spełnić wymagania specyfikacji *RFC7231* [11], tj.

- nie zawiera ciała, a wszelkie dane przekazywane są w adresie url;
- jest idempotentna (wielokrotne żądanie nie zmieni w żaden sposób stanu serwera);
- jest bezpieczna (za jej pomocą można jedynie odczytać dane);

Dane przekazywane do serwera tą metodą dotyczą jedynie wyboru określonego podzbioru danych i są przekazywane poprzez mechanizm *query-string*. Jest to zbiór par *klucz=wartosc* oddzielonych znakiem &. Przykładowy adres zawierający dwa klucze z przydzielonymi wartościami zaprezentowano poniżej.

```
/destinationAddress/foo.api?key1=30&key2=true
```

Dane te przekazywane są do serwera jako surowy ciąg znaków (ang. *raw data*), a następnie parsowane. Po tym etapie są gotowe do przetworzenia. W związku z tym przyjęto następujące obsługiwane typy danych:

1. stałoprzecinkowy (QS_PARAM_INT);
2. zmiennoprzecinkowy (QS_PARAM_DOUBLE);
3. logiczny (QS_PARAM_BOOL) – literały *true* oraz *false*;
4. znakowy (QS_PARAM_STRING);

Podczas procesu parsowania odpowiednie fragmentu surowego ciągu danych otrzymuję swój identyfikator oraz odpowiedni typ. Należy pamiętać, że w sytuacji, w której nie udało się przyporządkować wartości do żadnego typu otrzymuje ona typ znakowy. Po udanym procesie przetwarzania żądania następuje wykonanie właściwej części programu – pobrania żądanej informacji, a następnie zbudowanie wyniku w formacie JSON i zwróceniu go klientowi. Wymaganą składnię pakietu GET zaprezentowano poniżej:

```
GET /destination.api?publicKey=visible&otherKey=true HTTP/1.1
HOST 192.168.0.105
```

W prezentowanym powyżej komunikacie zmienna *publicKey* przyjmie wartość znakową, a *otherKey* – logiczną.

3.7.4 Wysyłanie danych (POST)

W przeciwieństwie do metody GET, metoda POST służy do modyfikowania działania systemu sterowania. Zgodnie ze specyfikacją *RFC7231* [11] metoda ta:

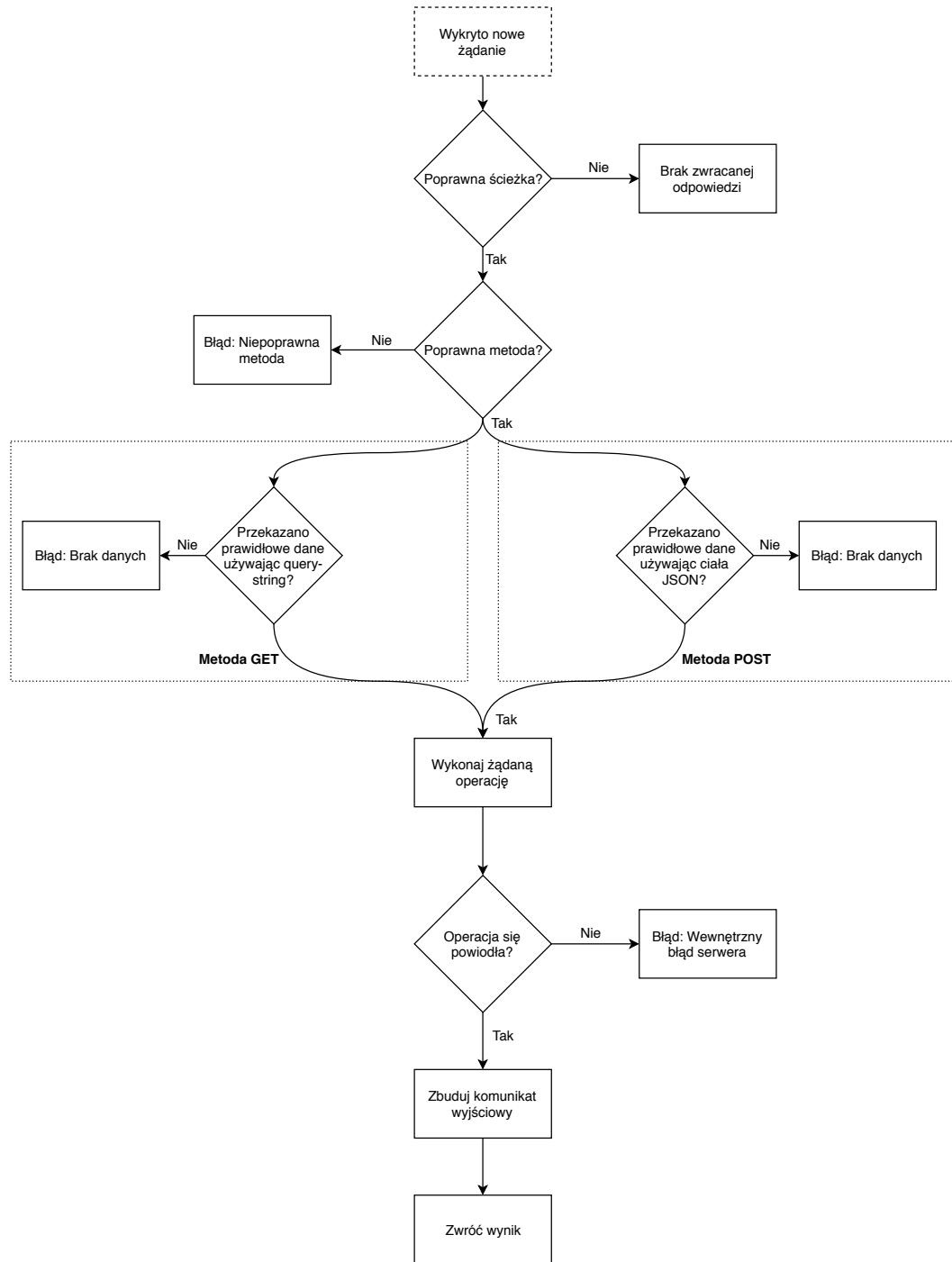
- zawiera ciało;
- nie jest idempotentna (przetworzenie żądania może spowodować zmianę stanu serwera);
- może spowodować efekty uboczne (ang. *side effects*)

W związku z przyjęciem sposobu kodowania ciała – jako obiekt JSON, do poprawnego działania nagłówków określający typ danych musi zostać ustawiony, jako *application-json*. Podobnie, jak w poprzedniej metodzie po udanym procesie parsowania ciała żądania wykonywana zostaje właściwa operacja, a następnie zwracany jest wynik. Przykład pakietu POST zaprezentowany został poniżej:

```
POST /destination.api HTTP/1.1
HOST 192.168.0.105
Content-Length: 18
Content-Type: application/json

hiddenField=secret
```

W przeciwieństwie do metody GET, gdzie wszelkie informacje przekazywane są w adresie URL, pakiet POST ukrywa sekcję wysyłanych danych (ang. *payload*). Bezpieczne jest zatem wysyłanie wrażliwych z punktu widzenia oprogramowania komunikatów (na przykład haseł). Rysunek 3.5 przedstawia sposób obsługi żądania przez serwer.



Rysunek 3.5. Algorytm przetwarzania żądania HTTP.

Możliwe wywołania API, wraz z niezbędnymi parametrami oraz zwracanymi wartościami zebrane zostały w Dodatku A.

3.7.5 Pliki statyczne

Wbudowany w projekt serwer WWW umożliwia udostępnianie plików statycznych. Dzięki temu możliwe jest implementowanie interfejsów użytkownika opartych o protokół HTTP. Udostępniać można zarówno dokumenty hipertekstowe (HTML), jak również arkusze stylów (CSS), skrypty oraz grafiki. Ważną kwestią staje się zatem rozróżnianie żądania HTTP pliku statycznego oraz wywołania API. W projekcie wykorzystano metodę, w której w celu użycia API dodajemy przyrostek *.api*:

```
API  
http://192.168.0.105/lego-parking-status.api  
  
Plik statyczny  
http://192.168.0.105/index.html
```

Pliki statyczne przechowywane są w pamięci nieulotnej programu. Tworzone są za pomocą dedykowanego narzędzia konwertującego faktyczny plik do tablicy bajtów możliwej do przetworzenia przez serwer. Konwertowanie możliwe jest dzięki skryptowi języka Perl, który dostępny jest dla wszystkich popularnych języków operacyjnych. W tym celu należy uruchomić skrypt *mkfs.pl* wraz z podaniem katalogu źródłowego, zawierającego konwertowane pliki. W środowisku Unixowym, które zazwyczaj wyposażone jest w interpreter języka Perl należy nadać prawa do uruchamiania skryptu oraz wykonać konwersję:

```
cd httpsrv/static  
chmod +x mkfs.pl  
.mkfs.pl webpage
```

W środowisku Windows istnieje potrzeba zainstalowania odpowiednich narzędzi. Można w tym celu użyć otwartego oprogramowania *Padre, the Perl IDE*, które zawiera interpreter języka wraz edytorem¹. Podczas konwersji odpowiadające plikom tablice, za pomocą makr sekcji przekierowane są do pamięci nieulotnej *BOARD_FLASH*. Wynikiem działania skryptu jest pojedynczy plik języka C komplilowany wraz z projektem. Podczas inicjalizacji programu odpowiednie tablice przekazywane są do oprogramowania serwera, skąd dostępne są z poziomu przeglądarki.

¹<http://padre.perlide.org/>

3.8 Warstwa sterowania makietą

Warstwa sterowania umożliwia niskopoziomową wymianę danych z urządzeniami realizującymi niezbędne założenia systemu sterowania. Za jej pośrednictwem możliwa jest komunikacja z niezależnymi modelami (budynkami) intelligentnego miasta. Jest najbardziej krytyczną częścią całego systemu, gdyż z jej implementacji wynika opóźnienie czasu sterowania oraz niezawodność jego pracy. W sekcji tej opisano niezbędne typy danych wraz z ich wykorzystaniem. Wyszczególniono newralgiczne sekcje oprogramowania oraz proces inicjalizacji urządzeń.

3.8.1 Inicjalizacja urządzeń

Każdorazowe uruchomienie systemu wymaga skonfigurowania istniejących urządzeń I2C do ich poprawnej pracy. Ważna jest kolejność operacji, gdyż istnieją zależności pomiędzy niektórymi elementami systemu. Pełen proces inicjalizacji zaprezentowany został poniżej.

1. Przekazanie uchwytów do niskopoziomowych funkcji odczytu i zapisu I2C do sterowników urządzeń (biblioteki TCA9548A oraz MCP23017);
2. Inicjalizacja ekspanderów wyprowadzeń. Ustawienie pinów w tryb wyjścia oraz zresetowanie portów do wartości domyślnych;
3. Inicjalizacja czujników odległości. Ustawienie trybu pracy ciągłej (automatyczne odświeżanie danych). Rozpoczęcie transmisji danych;
4. Zainicjalizowanie wyświetlacza OLED. Narysowanie bitmapy (tła) parkingu;
5. Zresetowanie animacji oraz wyłączenie wszystkich światel w makiecie;

Po pomyślnym zakończeniu wszystkich etapów system jest gotowy do działania w pełnym zakresie.

3.8.2 Struktura LED

Podstawowymi elementami makiety są diody emitujące światło, które są wykorzystywane zarówno do sterowania oświetleniem ulicznym, zewnętrznym oraz wewnętrznym, jak i tworzą niezbędne struktury animacji.

Listing 3.5: Struktura LED

```
typedef struct {  
    uint32_t lightId;
```

```

const LEGO_MCP23017Info_t *mcp23017Info;
uint8_t mcp23017Pin;
uint32_t groupsId[LEGO_LIGHT_MAX_GROUPS];
uint8_t autoModePercentage;
} LEGO_Light_t;

```

Każda LED posiada unikalny identyfikator, umożliwiający bezpośredni i szybki dostęp. Następnie przechowywane są parametry przypisujące ją do odpowiedniego urządzenia na magistrali I2C. Są to kanał multipleksera, adres ekspandera wyprowadzeń wraz z używanym portem oraz określone wyprowadzenie danego portu. Kombinacja powyższych trzech wartości jest unikalna w skali całego systemu. Wprowadzono przynależność do określonych grup oświetlenia. W ten sposób możliwe jest sterowanie zbiorem powiązanych ze sobą elementów, na przykład budynkami lub oświetleniem ulicznym. Typ wyliczeniowy *LEGO_LightGroup_t* zawiera grupy używane w systemie. Szczególną z nich jest *LEGO_GROUP_BROADCAST*, która łączy wszystkie statyczne elementy oświetlenia (nie biorących udziału w animacjach), umożliwiająca szybką zmianę stanu oświetlenia całej makiety. Biorąc pod uwagę możliwość sterowania pojedynczego oraz grupowego oraz fakt, że możliwe jest włączanie, wyłączanie oraz zmiana stanu oświetlenia na przeciwny istnieją praktycznie nieograniczone możliwości konfiguracji według niezbędnych w danej chwili potrzeb sterowania. Ostatnią informacją, którą zawiera struktura jest stosunek czasu włączenia do czasu wyłączenia diody w trybie automatycznym sterowania.

3.8.3 Animacje systemu sterowania

Kolejnym elementem sterowania są animacje. Część z nich realizowana jest programowo, a niektóre zawierają wsparcie sprzętowe niezbędne do poprawnej ich pracy. W rozdziale 2 zaprezentowano niezbędne schematy elektryczne. Poniższa tabela zbiera najważniejsze informacje o wszystkich animacjach działających w systemie.

Tabela 3.4. Animacje systemu sterowania.

Nazwa	Wsparcie sprzętowe
Tryb automatyczny	Nie
Kino Palace	Tak
Roller Coaster	Nie
Sygnalizacja uliczna	Tak
Turbina wiatrowa	Tak

W celu zapewnienia maksymalnej wydajności każda animacja przydzielona została do niezależnego wątku (zadania) systemu. Umożliwia to równoległe działanie dwóch lub więcej

z nich – brak zależności jednej animacji od drugiej (na przykład oczekiwania na aktualizację innej animacji). Wyjątkiem jest animacja turbiny wiatrowej, która dzięki zastosowaniu układu multiwibratora pozwala na całkowite odseparowanie od części programowej. Zminimalizowano niezbędne instrukcje wymagane do zmiany klatek animacji. Niezbędne dane pobierane są ze wcześniej przygotowanych tablic zmniejszając ryzyko opóźnienia w procesie sterowania. Zastosowanie niezależnych wątków zrodziło problem dostępu do wymaganych urządzeń na linii I2C, gdyż możliwe jest przydzielenie czasu procesora w momencie aktualizacji (wysyłania danych) animacji innemu zadaniu w systemie. Spowodowałoby to przekazanie danych do niepoprawnego urządzenia, a w konsekwencji doprowadziło do niespójności działania programu. Wspomniane we wcześniejszym podrozdziale sekcje krytyczne skutecznie uniemożliwiają wystąpienie takiej sytuacji. W systemie przyjęto, że każdorazowa komunikacja z urządzeniem, tj. wybór odpowiedniego kanału multipleksera oraz docelowa komunikacja z urządzeniem jest operacją atomową. Na przykład:

Listing 3.6: Atomowa operacja komunikacji z urządzeniem I2C

```
taskENTER_CRITICAL();
/* Choose correct I2C channel and update MCP23017 port data */
if (TCA9548A_SelectChannels(TCA9548A_DEFAULT_ADDR, TCA9548A_CHANNEL0)) {
    MCP23017_PortWriteMasked(&chain, devNum, MCP23017_PORT_A, 0xFF, 0xFF);
}
taskEXIT_CRITICAL();
```

3.8.4 Tryb automatyczny oraz ręczny

Sterowanie makietą może odbywać się w dwóch, wybieranych podczas działania systemu trybach. Pierwszym z nich jest tryb ręczny. Umożliwia on zadawanie określonych stanów oświetlenia przez użytkownika. Tryb ten gwarantuje niezmienność stanu makiety bez zamierzenia osoby obsługującej (nie dotyczy to aktualnie działających animacji w systemie). Drugim trybem jest szczególny rodzaj animacji – tryb automatyczny, który zależnie od przekazanych parametrów realizuje odświeżanie stanu oświetlenia. Bazuje on na wspomnianym wcześniej procentowym czasie aktywności pojedynczej diody. Ustalona wartość, mieszcząca się w zakresie 0-100 porównywana jest z wartością bazową (losowaną na początku każdego obiegu). Wynik tej operacji decyduje, czy dioda powinna zostać zgaszona lub zaświecona. Zastosowanie różnych czasów aktywności dla elementów makiety tworzy symulację życia w mieście. Zebrane w typie wyliczeniowym *LEGO_AutoModePercentage_t* wartości szeregują różne rodzaje oświetlenia, jak na przykład oświetlenie uliczne, oświetlenie wewnętrzne, czy małe obiekty w mieście. Możliwe jest zablokowanie aktywności pojedynczej

diody – poprzez ustalenie czasu aktywności na 0.

3.8.5 Grupowa komunikacja z urządzeniami

Zmiana stanu makiety często zależna jest od zmiany stanu kilku pinów w jednym lub więcej ekspanderach wyprowadzeń. Wysyłanie komendy dla każdej diody osobno, choć poprawne z punktu działania systemu byłoby znaczącym obniżeniem jego wydajności. W tym celu zaimplementowano mechanizm, który zapobiega nadmiernej transmisji łącząc dane adresowane do tego samego urządzenia. Jego działanie opiera się na pobraniu zmienianych identyfikatorów oświetlenia wraz z żądanymi stanami, a następnie zbudowaniu ramek, które łączą pojedyncze wyprowadzenia ekspandera w jedną grupę. W przypadku ekstremalnym, gdzie zmianie ulegają wszystkie wyprowadzenia portu ekspandera pozwala to skrócić czas wysyłania danych osiemokrotnie, gdyż każdy z pinów aktualizowany jest zbiorczo. Do zobrazowania tej akcji opisano sytuację, w której zmiana dotyczy niższego półbajtu portu (ang. *low nibble*) oraz bitu najbardziej znaczącego na jedynki logiczne. Zamiast stosować pięć niezależnych operacji wysłania danych odczytywany jest obecny stan portu, który za pomocą maski blokuje zmianę stanu innych pinów. Wyznaczona wartość jest następnie wysyłana jako stan nowego portu.

Listing 3.7: Algorytm grupowej komunikacji

```
/* Bits 0:3 = 1, MSB = 1 */
uint8_t newPortStatus = 0b10001111;
/* Current port status */
uint8_t currentPortStatus = MCP23017_PortRead(...);
/* Mask */
uint8_t mask = 0b10001111;
currentPortStatus &= ~mask;
newPortStatus &= mask;
/* Calculate new port status */
uint8_t sendPortStatus = currentPortStatus | newPortStatus;
/* Send new port status */
MCP2017_PortWrite(..., sendPortStatus, ...);
```

Na potrzeby systemu sterowania zaimplementowano funkcję *LEGO_SearchLights*, która umożliwia tworzenie zbiorczych ramek zarówno w trybie aktualizacji pojedynczych wyprowadzeń oraz ich grup.

3.8.6 Inteligentny parking

Model inteligentnego parkingu, w który wyposażona jest makieta miasta jest elementem, dynamicznie reagującym na zmieniające się warunki. Użyte rozwiązanie, gdzie każde miejsce

parkingowe jest osobno adresowane umożliwia określenie stanu jego zajętości, wraz z dokładnym określeniem, które miejsca są już zajmowane. Wbudowany wyświetlacz monochromatyczny służy do prezentowania zmieniających się stanów oraz informowaniu ile miejsc parkingowych jest możliwych do zajęcia.



Rysunek 3.6. Działanie modelu parkingu.

Struktura, identyfikująca pojedyncze miejsce parkingowe zaprezentowana została poniżej.

Listing 3.8: Struktura miejsca parkingowego

```
typedef struct {
    uint8_t placeId;
    const LEGO_VL6180XInfo_t *vl6180xInfo;
    uint8_t vl6180xDevNum;
    const LEGO_MCP23017Info_t *mcp23017CEInfo;
    uint8_t mcp23017CEPin;
    bool prevState;
    bool occupied;
} LEGO_ParkingPlace_t;
```

Podobnie jak w przypadku LED, każdy czujnik zawiera unikalny identyfikator, następnie przechowywane są dane urządzenia. Jak wspomniano w rozdziale 2, urządzenie wymaga dedykowanego pinu aktywującego układ w procesie inicjalizacji – stąd konieczność przechowywania wykorzystywanego ekspandera wraz z portem i wyprowadzeniem. Przechowywane są ostatnie dwa odczyty. Motywacją do tego był fakt, że tylko zmiana stanu któregokolwiek z czujników (zmiana zajętości miejsca parkingowego) wymusza aktualizację wyświetlacza, co w znacznym stopniu przyczynia się do zwiększenia wydajności systemu. Odświeżanie informacji o parkingu oraz aktualizacja wyświetlacza zostały przyporządkowane do niezależnego wątku, który z założonym interwałem czasowym odczytuje nowe wartości oraz w zależności od nich realizuje określone akcje.

3.8.7 Interfejs użytkownika

Zaimplementowany interfejs użytkownika, dostępny za pomocą przeglądarki internetowej umożliwia sterowanie makietą w czasie rzeczywistym. Z jego poziomu możliwe jest sterowanie zarówno oświetleniem w mieście, jak i włączanie oraz działających wyłączanie animacji.



Rysunek 3.7. Interfejs użytkownika systemu sterowania.

Interfejs składa się z opisanych poniżej elementów.

- Pasek nawigacyjny umieszczony na górze strony umożliwiający szybkie włączenie oraz wyłączenie oświetlenia całej makiety oraz przyciski sterujące aktywnymi animacjami w systemie;
- Grafika przedstawiająca aktualnie wybrany model makiety;
- Przyciski umożliwiające szybkie włączenie oraz wyłączenie oświetlenia w obecnie wybranym budynku;
- Zbiór przycisków, które pozwalają na sterowanie wybranymi diodami oświetlenia;

Zmiana aktywnego budynku (modelu) odbywa się poprzez wcisnięcie odpowiedniej kontrolki nawigacyjnej (poprzedni / następny).

Testy działania systemu

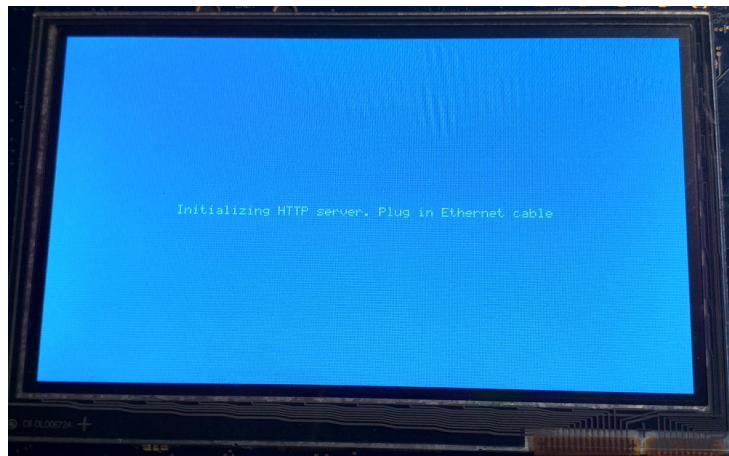
Rozdział ten przeznaczono na opis testów wykonanych podczas implementacji pracy. Dotyczyły one najbardziej krytycznych części warstw sprzętowej oraz programowej.

4.1 Test spadku napięcia

1. **Cel:** Ustalenie maksymalnego spadku napięcia przy całkowitym obciążeniu systemu;
2. **Sposób:** Jednoczesne zapalenie wszystkich świateł oraz uruchomienie dostępnych animacji;
3. **Wynik:** Z uwagi na zastosowane zewnętrzne źródło zasilania spadek napięcia na końcu magistrali zasilającej wynosił około 0,5 V. To wartość, która nie wpływa w żaden sposób na działające urządzenia I2C. Jasność świecenia LED nie uległa znacznemu pogorszeniu;

4.2 Test błędnej inicjalizacji urządzeń I2C

1. **Cel:** Ustalenie zachowania systemu podczas niekompletnej inicjalizacji;
2. **Sposób:** Odłączenie wybranych urządzeń lub kabla sieciowego podczas uruchamiania programu;
3. **Wynik:** Zgodnie z założeniami wyświetlacz LCD wskazuje na aktualnie wykonywany etap inicjalizacji, a dostępna wbudowana konsola debugera wyświetla szczegółowe instrukcje, których nie udało się poprawnie wykonać;



Rysunek 4.1. Błąd inicjalizacji systemu – oczekiwanie na podłączenie kabla Ethernet.

4.3 Test zajętości pamięci operacyjnej

1. **Cel:** Ustalenie zajętości stosów dla przydzielonych zadań RTOS;
2. **Sposób:** Uruchomienie wszystkich zadań działających w systemie. Obserwacja zużycia pamięci przez wbudowany w środowisko programistyczne widok wątków FreeRTOS;
3. **Wynik:** Biorąc pod uwagę duży rozmiar sterty (8 MB), która wykorzystywana jest przez działające zadania, przydzielone stosy dysponują dużym zapasem pamięci. Podczas testów nie stwierdzono zużycia większego, niż 50% dostępnej pamięci. Najbardziej wymagającym w stosunku co do wykorzystania zasobów zadaniem systemu jest obsługa menadżera okien (GUI);

4.4 Test żądań HTTP

1. **Cel:** Ustalenie zachowania serwera podczas braku przekazanych, lub błędnych typów parametrów;
2. **Sposób:** Wysłanie niekompletnych lub wadliwych żądań HTTP poprzez tester wywołań API;
3. **Wynik:** Reakcja serwera jest zależna od występującej sytuacji. Zwracane są odpowiednie komunikaty o błędach parsowania parametrów lub kody odpowiedzi HTTP wskazujące na popełniony błąd;

Podsumowanie

Celem niniejszej pracy była implementacja wbudowanego systemu sterowania makietą inteligentnego miasta. W jej ramach powstały zarówno warstwa sprzętowa, jak i programowa. Makieta może być wykorzystywana podczas różnych wydarzeń oraz obrazować koncepcję Internetu Rzeczy. Warto nadmienić, że projekt stwarza ogromne możliwości rozbudowy. Możliwe jest połączenie nowych urządzeń, pełniących nowe role w systemie. Ciekawymi pomysłami, które mogłyby zostać wykorzystane są:

- wyposażenie budynków w czujniki temperatury;
- pomiar wilgotności w mieście;
- aktywne sterowanie latarniami ulicznymi;
- aplikacja mobilna współpracująca z modelem parkingu;
- wykorzystanie graficznych matryc – wyświetlanie informacji pogodowych, banery reklamowe;

Zrealizowane zostały wszystkie wytyczone cele oraz funkcjonalności. Projekt umożliwił zgłębienie wiedzy na temat systemów operacyjnych czasu rzeczywistego, a także sposobów wymiany danych na różnych poziomach programowych. Autorzy niniejszej pracy są pewni, że wrażenia użytkownika końcowego aplikacji będą pozytywne, a sama makieta będzie sukcesywnie rozwijana w przyszłości.

API serwera HTTP

GET	/lego-light.api Pobierz status LED
------------	------------------------------------

Wymagane parametry:

Nazwa	Typ	Opis
mode	string	Sposób przeszukiwania. Dopuszczalne wartości to "group" dla przeszukiwania grupowego oraz "single" dla przeszukiwania pojedynczego. Typ wyliczeniowy <i>LEGO_LightGroup_t</i> zawiera dopuszczalne grupy światel
id	integer	Przeszukiwany identyfikator

Zwracane wartości:

Status	Opis
400	Brak wymaganych parametrów
404	Przekazany identyfikator nie istnieje w systemie
500	Błąd komunikacji z urządzeniem na magistrali I2C lub wewnętrzny błąd systemu
200	Pobrany status LED

Przykładowe żądanie wraz z odpowiedzią:

```
/lego-light.api?mode=single&id=10
{
    "status": 200,
    "states": [
        {
            "id": 10,
            "state": 0
        }
    ]
}
```

POST /lego-light.api Ustaw status LED

Wymagane parametry:

Nazwa	Typ	Opis
mode	string	Sposób ustawiania. Dopuszczalne wartości to “group” dla ustawiania grupowego oraz “single” dla ustawiania pojedynczego. Typ wyliczeniowy <i>LEGO_LightGroup_t</i> zawiera dopuszczalne grupy światel
id	integer	Ustawiany identyfikator
operation	string	Wybrana operacja. Dopuszczalne wartości to “on” dla włączania, “off” dla wyłączania oraz “toggle” dla zmiany stanu na przeciwny

Zwracane wartości:

Status	Opis
415	Wartość nagłówka Content-Type różna od <i>application/json</i>
400	Brak wymaganych parametrów
404	Przekazany identyfikator nie istnieje w systemie
500	Błąd komunikacji z urządzeniem na magistrali I2C lub wewnętrzny błąd systemu
200	Powodzenie operacji

Przykładowe żądanie wraz z ciałem (payload)

```
/lego-light.api
{
    "mode": "group",
    "operation": "toggle",
    "id": 15
}
```

GET

/lego-anim-status.api Pobierz status animacji

Wymagane parametry:

Nazwa Typ Opis

Nazwa	Typ	Opis
id	integer	Identyfikator animacji. Typ wyliczeniowy <i>LEGO_Anim_t</i> zawiera dopuszczalne identyfikatory

Zwracane wartości:

Status Opis

400	Brak wymaganych parametrów
404	Przekazany identyfikator nie istnieje w systemie
500	Wewnętrzny błąd systemu
200	Status animacji

Przykładowe żądanie wraz z odpowiedzią:

```
/lego-anim-status.api?id=0
{
    "status": 200,
    "animStatus": 0
}
```

POST	/lego-anim-status.api	Ustaw status animacji
-------------	-----------------------	-----------------------

Wymagane parametry:

Nazwa	Typ	Opis
id	integer	Identyfikator animacji. Typ wyliczeniowy <i>LEGO_Anim_t</i> zawiera dopuszczalne identyfikatory
animStatus	integer	Status animacji

Zwracane wartości:

Status	Opis
415	Wartość nagłówka Content-Type różna od <i>application/json</i>
400	Brak wymaganych parametrów
404	Przekazany identyfikator nie istnieje w systemie
500	Błąd komunikacji z urządzeniem na magistrali I2C lub wewnętrzny błąd systemu
200	Powodzenie operacji

Przykładowe żądanie wraz z ciałem (payload)

/lego-anim-status.api
{
"id": 4,
"animStatus": 0
}

GET

/lego-anim-delay.api Pobierz czas odświeżania animacji

Wymagane parametry:

Nazwa	Typ	Opis
id	integer	Identyfikator animacji. Typ wyliczeniowy <i>LEGO_Anim_t</i> zawiera dopuszczalne identyfikatory

Zwracane wartości:

Status	Opis
400	Brak wymaganych parametrów
404	Przekazany identyfikator nie istnieje w systemie
500	Wewnętrzny błąd systemu
200	Czas odświeżania animacji

Przykładowe żądanie wraz z odpowiedzią:

```
/lego-anim-delay.api?id=2
{
    "status": 200,
    "animDelayMs": 150
}
```

Uwagi

Animacja turbiny wiatrowej realizowana jest sprzętowo. W tym przypadku wartość zwracana jest zawsze równa 0.

POST	/lego-anim-delay.api Ustaw czas odświeżania animacji
-------------	---

Wymagane parametry:

Nazwa	Typ	Opis
id	integer	Identyfikator animacji. Typ wyliczeniowy <i>LEGO_Anim_t</i> zawiera dopuszczalne identyfikatory
animDelayMs	integer	Czas odświeżania animacji

Zwracane wartości:

Status	Opis
415	Wartość nagłówka Content-Type różna od <i>application/json</i>
400	Brak wymaganych parametrów
404	Przekazany identyfikator nie istnieje w systemie
500	Błąd komunikacji z urządzeniem na magistrali I2C lub wewnętrzny błąd systemu
200	Powodzenie operacji

Przykładowe żądanie wraz z ciałem (payload)

```
/lego-anim-delay.api
{
    "id": 1,
    "animDelayMs": 300
}
```

Uwagi

Animacja turbiny wiatrowej realizowana jest sprzętowo. W tym przypadku nie jest możliwe ustawienie czasu odświeżania, a serwer zwraca status błędu 500.

GET

/lego-parking-status.api Pobierz zajętość parkingu

Wymagane parametry: BRAK

Zwracane wartości:

Status Opis

500	Wewnętrzny błąd systemu
200	Zajętość parkingu

Przykładowe żądanie wraz z odpowiedzią:

```
/lego-parking-status.api
{
    "status": 200,
    "parkingStatus": [
        {
            "placeId": 0,
            "occupied": 0
        },
        {
            "placeId": 1,
            "occupied": 1
        },
        ...
        {
            "placeId": 9,
            "occupied": 1
        }
    ]
}
```


Opis wyprowadzeń

B.1 Wyprowadzenia mikrokontrolera

Podłączenie urządzeń I2C z mikrokontrolerem wymaga dwóch linii tworzących szynę komunikacyjną. W tym celu wykorzystano interfejs FLEXCOMM1 znajdujący się w sekcji PMOD (interfejs hosta). Podczas używania zewnętrznego źródła zasilania konieczne staje się także połączenie mas płyty ewaluacyjnej i urządzeń. Wykorzystane piny zebrano w poniższej tabeli.

Tabela B.1. Używane piny mikrokontrolera.

Wyprowadzenie	Opis
P0_13	FLEXCOMM1 – linia SCL szyny I2C
P0_14	FLEXCOMM1 – linia SDA szyny I2C
GND	Dowolne wyprowadzenie masowe dostępne na płycie

B.2 Wyprowadzenia ekspanderów

Budynek A (Bank) - Kanał multipleksera: 0 | Adres ekspandera 0x20

Identyfikator	Wyprowadzenie	Opis
1	PB2	Lampa uliczna
2	PB1	Pralnia
3	PB0	Lampy zewnętrzne
4	PB6	Biurko 1
5	PB5	Żyrandol
6	PB4	Biurko 2

Identyfikator	Wyprowadzenie	Opis
1	PB0	Lampa uliczna 1
2	PB1	Lampa uliczna 2
3	PB2	Fontanna
4	PB3	Pokój – 2 piętro
5	PB4	Toaleta – 2 piętro
6	PB5	Gablota 1
7	PB6	Lampa zewnętrzna – tył
8	PB7	Gablota 2
9	PA2	Lampy zewnętrzne – przód
10	PA3	Lampa zewnętrzna – taras
11	PA4	Lampa zewnętrzna – bok
12	PA5	Piekarnia
13	PA6	Dentysta – lampa UV
14	PA7	Kwiaciarnia
15	PA0	Fotograf
16	PA1	Dentysta

Budynek B2 (Plac zgromadzeń 2) - Kanał multipleksera: 0 | Adres ekspandera 0x22

Identyfikator	Wyprowadzenie	Opis
1	PB0	Studio tańca
2	PB1	Kafejka – stolik
3	PB2	Lampy zewnętrzne 1
4	PB4	Schody – parter
5	PB5	Schody – 2 piętro
6	PB6	Schody – 1 piętro
7	PA0	Sklep muzyczny
8	PA1	Lampy zewnętrzne 2
9	PA2	Kafejka

Budynek C (Paryska restauracja) - Kanał multipleksera: 0 | Adres ekspandera 0x23

Identyfikator	Wyprowadzenie	Opis
1	PB0	Restauracja – lampa stołowa 1
2	PB1	Lampy zewnętrzne
3	PB2	Salon
4	PB4	Balkon – lampa 1
5	PB5	Balkon – lampa 2
6	PB6	Studio malarskie
7	PA0	Balkon – lampa 3
8	PA1	Balkon – lampa 4
9	PA3	Restauracja – lampa stołowa 2
10	PA4	Lampa uliczna

Budynek D (Kino Palace) - Kanał multipleksera: 0 | Adres ekspandera 0x24

Identyfikator	Wyprowadzenie	Opis
1	PB0	Zablokuj działanie animacji kina
2	PA0	Animacja – klatka 1
3	PA1	Animacja – klatka 2
4	PA2	Animacja – klatka 3
5	PB5	Lampy zewnętrzne
6	PB6	Lampa uliczna
7	PB7	Rzutnik sali kinowej
8	PB2	Sala kinowa
9	PB3	Kino – kasa

Budynek E (Peron 1) - Kanał multipleksera: 0 | Adres ekspandera 0x26

Identyfikator	Wyprowadzenie	Opis
1	PA6	Peron – lewa strona
2	PA5	Peron – prawa strona
3	PA4	Bankomat
4	PA3	Zajezdnia

Budynek F (Warsztat na rogu) - Kanał multipleksera: 0 | Adres ekspandera 0x25

Identyfikator	Wyprowadzenie	Opis
1	PB0	Lampa zewnętrzna – przód
2	PA0	Taras
3	PA2	Lecznica
4	PA4	Stacja paliw – dystrybutor
5	PB5	Lampa zewnętrzna – tył
6	PB6	Mechanik – lampa zewnętrzna
7	PA0	Kuchnia
8	PA1	Mechanik – warsztat
9	PA2	Mechanik – biuro
10	PA4	Biurko
11	PA5	Lampa uliczna
12	PA6	Jadalnia

Budynek G (Kamienica) - Kanał multipleksera: 0 | Adres ekspandera 0x26

Identyfikator	Wyprowadzenie	Opis
1	PB6	Remontowany pokój
2	PB7	Salon
3	PA0	Strych
4	PA1	Lampa zewnętrzna – tył

Budynek H (Sklep zoologiczny) - Kanał multipleksera: 0 | Adres ekspandera 0x26

Identyfikator	Wyprowadzenie	Opis
1	PB0	Sklep zoologiczny
2	PB1	Lampa – strych
3	PB2	Lampa uliczna
4	PB4	Schody
5	PB6	Kuchnia

Budynek I (Biuro detektywa) - Kanał multipleksera: 0 | Adres ekspandera 0x27

Identyfikator	Wyprowadzenie	Opis
1	PA0	Kuchnia
2	PA1	Lampa zewnętrzna 1
3	PA2	Toaleta
4	PB0	Fryzjer – lampa 1
5	PB1	Biuro detektywa
6	PB2	Fryzjer – lampa 2
7	PA4	Lampa zewnętrzna 2
8	PA5	Lampa zewnętrzna 3
9	PB4	Biuro detektywa – lampka biurkowa
10	PB5	Pokój gier
11	PB6	Lampa uliczna

Budynek J (Bistro w śródmieściu) - Kanał multipleksera: 1 | Adres ekspandera 0x20

Identyfikator	Wyprowadzenie	Opis
1	PB0	Studio nagraniowe 1
2	PB1	Lampa uliczna
3	PB2	Knajpa 2
4	PB4	Siłownia 1
5	PB5	Lampa zewnętrzna 1
6	PB6	Knajpa 1
7	PA0	Siłownia 2
8	PA1	Lampa zewnętrzna 2
9	PA3	Salon
10	PA4	Studio nagraniowe 2
11	PA7	Włącz/wyłącz animację turbiny wiatrowej

Budynek K (Dom na wsi 1) - Kanał multipleksera: 1 | Adres ekspandera 0x21

Identyfikator	Wyprowadzenie	Opis
1	PB4	Pomieszczenie
2	PB6	Altana – lampa 1
3	PB5	Altana – lampa 2

Budynek L (Dom na wsi 2) - Kanał multipleksera: 1 | Adres ekspandera 0x21

Identyfikator	Wyprowadzenie	Opis
1	PA2	Lampa zewnętrzna
2	PA1	Lampa uliczna
3	PA0	Pomieszczenie

Budynek M (Peron 2) - Kanał multipleksera: 1 | Adres ekspandera 0x21

Identyfikator	Wyprowadzenie	Opis
1	PA2	Pomieszczenie 1
2	PA1	Pomieszczenie – 1 piętro
3	PA0	Tablica informacyjna
4	PA2	Pomieszczenie 2
5	PA1	Lampa zewnętrzna 1
6	PA0	Lampa zewnętrzna 2

Budynek N (Roller Coaster) - Kanał multipleksera: 1 | Adres ekspandera 0x22

Identyfikator	Wyprowadzenie	Opis
1	PA0	Bramki
2	PA1	Bilety
3	PA2	Lodziarnia
4	PA3	Animacja – klatka 1
5	PA4	Animacja – klatka 2
6	PA5	Animacja – klatka 3
7	PA6	Animacja – klatka 4
8	PA7	Animacja – klatka 5
9	PB2	Animacja – klatka 6
10	PB1	Animacja – klatka 7
11	PB0	Animacja – klatka 8
12	PB5	Animacja – klatka 9
13	PB4	Animacja – klatka 10
14	PB3	Budka z watą cukrową
15	PB7	Reflektory

Sygnalizacja drogowa - Kanał multipleksera: 0 | Adres ekspandera 0x24

Identyfikator	Wyprowadzenie	Opis
1	PA3	Sygnał sterujący 0
2	PA4	Sygnał sterujący 1
3	PA5	Sygnał sterujący 2
4	PA6	Sygnał sterujący 3

Parking - Kanał multipleksera: 2 | Adres ekspandera 0x21

Identyfikator	Wyprowadzenie	Opis
1	PA6	Aktywowanie czujnika parkingowego 1
2	PA7	Aktywowanie czujnika parkingowego 2
3	PA3	Aktywowanie czujnika parkingowego 3
4	PA5	Aktywowanie czujnika parkingowego 4
5	PA4	Aktywowanie czujnika parkingowego 5
6	PB4	Aktywowanie czujnika parkingowego 6
7	PB3	Aktywowanie czujnika parkingowego 7
8	PB5	Aktywowanie czujnika parkingowego 8
9	PB7	Aktywowanie czujnika parkingowego 9
10	PB6	Aktywowanie czujnika parkingowego 10

Bibliografia

- [1] *MCUXpresso User Guide*. Grud. 2019. URL: https://www.nxp.com/docs/en/user-guide/MCUXpresso_IDE_User_Guide.pdf.
- [2] *EmWin User Guide & Reference Manual*. Grud. 2019. URL: <https://www.segger.com/downloads/emwin/UM03001>.
- [3] *LPCXpresso54628*. URL: <https://os.mbed.com/platforms/LPCXpresso54628/>.
- [4] *LPC546xx User manual*. Lip. 2017. URL: <https://studio.segger.com/packages/LPC54000/CMSIS/Documents/UM10912.pdf>.
- [5] *Waveshare MCP23017*. URL: <https://www.waveshare.com/mcp23017-io-expansion-board.htm>.
- [6] *Układ TCA9548A*. URL: <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout.pdf>.
- [7] *TCA9548A - user manual*. URL: <http://www.ti.com/lit/ds/symlink/tca9548a.pdf>.
- [8] *VL6180X proximity, gesture and ambient light sensing*. URL: <https://www.st.com/resource/en/datasheet/vl6180x.pdf>.
- [9] *1.3Inch OLED Display Module*. URL: <http://www.openplatform.cc/index.php/home/index/products/apiinfor/tf051>.
- [10] *The FreeRTOS™ Reference Manual*. Version 10.0.0. 2017.
- [11] *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. URL: <https://tools.ietf.org/html/rfc7231>.
- [12] Joseph Yiu. *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors*. 3rd edition. 2013.
- [13] Richard Barry. *Mastering the FreeRTOS™ Real Time Kernel*. Pre-release 161204 Edition. 2016.
- [14] Brian W. Kernighan i Dennis M. Ritchie. *Język ANSI C. Programowanie*. Wydanie II. Maj 2010.

- [15] *FatFS - przewodnik użytkowania*. URL: http://elm-chan.org/fsw/ff/00index_e.html (term. wiz. 11/2019).
- [16] *MCP23017 - user manual*. URL: <http://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf>.

Spis tablic

2.1	Prędkości magistrali I2C	17
2.2	Układ MCP23017	18
2.3	Układ TCA9548A	20
2.4	Sterownik OLED SH1106	23
2.5	Poziomy napięć wykorzystywane w projekcie	24
3.1	Struktura projektu	33
3.2	Mapa pamięci	36
3.3	Wątki RTOS	37
3.4	Animacje systemu sterowania	50
B.1	Używane piny mikrokontrolera.	67

Spis rysunków

1	Koncepcja Internetu Rzeczy	8
2	Makieta inteligentnego miasta	9
1.1	Schemat blokowy środowiska MCUXpresso	11
1.2	Przykładowy interfejs graficzny biblioteki <i>emWin</i>	12
1.3	Widok dokumentacji Doxygen	13
2.1	Płyta ewaluacyjna LPCXpresso 54628	15
2.2	Diagram blokowy rodziny LPC546XX	16
2.3	Schemat wymiany danych I2C	17
2.4	Ekspander wyprowadzeń MCP23017	18
2.5	Multiplekser TCA9548A	19
2.6	Schemat połączeń multipleksera	19
2.7	Schemat blokowy układu TCA9548A	20
2.8	Moduł z układem VL6180x	21
2.9	Zależność odległości od rodzaju powierzchni	22
2.10	Wyświetlacz OLED	23
2.11	Budowa modelu parkingu	23
2.12	Schemat układu zasilającego	24
2.13	Układ zasilający (połączone masy)	24
2.14	Schemat elektryczny animacji Kina Palace	25
2.15	Układ animacji kina	26
2.16	Działanie animacji kina	26
2.17	Schemat elektryczny animacji sygnalizacji świetlnej	27
2.18	Moduł sygnalizacji świetlnej	28
2.19	Działanie sygnalizacji świetlnej	28
2.20	Schemat elektryczny animacji turbiny wiatrowej	29
2.21	Układ multiwibratora turbiny wiatrowej	29
2.22	Działanie turbiny wiatrowej	30
2.23	Model kolejki górskiej	30
2.24	Rzut poziomy makiety	31

3.1	Schemat przełączania zadań systemu FreeRTOS	38
3.2	Algorytm sterowania wyświetlaczem LCD	41
3.3	Widok klawiatury ekranowej	43
3.4	Widok okna potwierdzenia	44
3.5	Algorytm przetwarzania żądania HTTP	47
3.6	Działanie modelu parkingu	53
3.7	Interfejs użytkownika systemu sterowania	54
4.1	Błąd inicjalizacji systemu – oczekiwanie na połaczenie kabla Ethernet	56

Listings

3.1	Sekcja krytyczna systemu <i>FreeRTOS</i>	36
3.2	Wartości domyślne ustawień	39
3.3	Mechanizm wywołania zwrotnego okna roboczego	42
3.4	Algorytm otwierania okien dialogowych	42
3.5	Struktura LED	49
3.6	Atomowa operacja komunikacji z urządzeniem I2C	51
3.7	Algorytm grupowej komunikacji	52
3.8	Struktura miejsca parkingowego	53