

**PROJEKT INŻYNIERSKI**

Inteligentny rdzeń dalmierza ultradźwiękowego dla robota mobilnego

**Damian MACHURA**

**Nr albumu 196310 268930**

**Kierunek:** Elektronika i Telekomunikacja

**Specjalność:** Elektronika Cyfrowa

**PROWADZĄCY PRACĘ**

**Dr hab. Inż. Adam MILIK Prof. PŚ**

**KATEDRA Systemów cyfrowych**

**Wydział Automatyki, Elektroniki i Informatyki**

**GLIWICE Rok 2025**

**Tytuł pracy:**

Inteligentny rdzeń dalmierza ultradźwiękowego dla robota mobilnego

**Streszczenie:**

Celem projektu było opracowanie inteligentnego rdzenia dalmierza ultradźwiękowego przeznaczonego dla robotów mobilnych. System został zaprojektowany w oparciu o technologię ultradźwiękową, umożliwiając precyzyjne pomiary odległości do przeszkód w dynamicznym środowisku. Rdzeń dalmierza składa się z modułów sterowania emisją i odbiorem impulsów ultradźwiękowych, a także algorytmów obliczających odległość na podstawie czasu przelotu sygnału. Testy przeprowadzone na różnych rodzajach powierzchni wykazały wysoką skuteczność systemu, szczególnie w standardowych warunkach. Projekt wykazał potencjał do zastosowań w systemach nawigacyjnych, dronach, pojazdach autonomicznych oraz aplikacjach przemysłowych. Na podstawie doświadczeń uzyskanych w czasie konstruowania urzadzeia powstały również kierunki dalszego rozwoju, takie jak optymalizacja algorytmów i wykorzystanie sztucznej inteligencji.

**Słowa kluczowe:**

Dalmierz ultradźwiękowy, Serwomechanizm, Pomiar odległości, FPGA, VHDL

**Thesis title:**

Intelligent Core of an Ultrasonic Rangefinder for Mobile Robots

**Abstract:**

The project aimed to develop an intelligent core of an ultrasonic rangefinder designed for mobile robots. The system utilize ultrasonic technology, which enables precise distance measurements to obstacles in the field of view of the sensor. The rangefinder core consists of modules for controlling the emission and reception of ultrasonic bursts, as well as algorithms for calculating distances based on receiving the echo reflected from an obstacle. Tests conducted on various surface types demonstrated the correct distance detection in typical environment. The project is perfectly suited for navigation systems, drones, autonomous vehicles, and industrial applications. The development process reveals the possibility of further improvements such as algorithm optimization and the use of integration with other positioning sensors or artificial intelligence.

**Keywords:**

Ultrasonic rangefinder, servo, distance measurement, FPGA, VHDL

Spis treści

[1. Wstęp 5](#_Toc187605601)

[2. Analiza tematu 10](#_Toc187605602)

[2.1. Wybrane metody pomiaru odległości robota mobilnego od przeszkód. 10](#_Toc187605603)

[2.2. Przykładowe rodzaje czujników odległości 10](#_Toc187605604)

[2.3. Wybrane sposoby poruszania głowicą czujnika odległościowego. 12](#_Toc187605605)

[2.4. Teoretyczna analiza wpływu temperatury i wysokości na prędkość fali ultradźwiękowej 12](#_Toc187605606)

[2.5. Ultradźwiękowy czujnik odległości 14](#_Toc187605607)

[2.6. Serwomechanizm SG90 15](#_Toc187605608)

[3. Wymagania i narzędzia 17](#_Toc187605609)

[3.1. Założenia projektu 17](#_Toc187605610)

[3.2. Platforma 18](#_Toc187605611)

[3.3. Wybór metodologii projektowania układu 22](#_Toc187605612)

[3.4. Narzędzie implementacji układowej 24](#_Toc187605613)

[4. Implmentacja układu 25](#_Toc187605614)

[4.1. Układ sterowania pomiarem odległości 26](#_Toc187605615)

[4.2. Opis działania Distance Calculation 29](#_Toc187605616)

[4.3. Sterownik położenia wałka serwomechanizmu 31](#_Toc187605617)

[4.4. Kontrola pomiaru 34](#_Toc187605618)

[5. Proces uruchamiania projektu 37](#_Toc187605619)

[5.1. Symulacja funkcjonalna 37](#_Toc187605620)

[5.2. Testowanie w ukłądzie Cyclone IV 40](#_Toc187605621)

[6. Wnioski 46](#_Toc187605622)

[6.1. Realizacja celu pracy 46](#_Toc187605623)

[6.2. Możliwości rozwoju 47](#_Toc187605624)

[7. Bibliografia 48](#_Toc187605625)

[Spis skrótów i symboli 50](#_Toc187605626)

[Źródła 51](#_Toc187605627)

[Lista dodatkowych plików, uzupełniających tekst pracy 52](#_Toc187605628)

[Spis rysunków 53](#_Toc187605629)

[Spis tablic 54](#_Toc187605630)

# Wstęp

Współczesne systemy robotyki mobilnej wymagają precyzyjnych i wydajnych rozwiązań do nawigacji w złożonym środowisku. Przykładem tego typu systemów jest dalmierz ultradźwiękowy, który umożliwia detekcję odległości do przeszkód oraz odwzorowanie otoczenia. Pomiar odległości za pomocą echa odbitego od przeszkody wymaga precyzyjnego pomiaru czasu pomiędzy wysłaniem paczki impulsów pomiarowych i jej odebraniem. Wykorzystanie mikrokontrolera do pomiaru odcinka czasu, oferują łatwość implementacji i niski koszt, lecz ich ograniczona wydajność obliczeniowa i brak elastyczności w zaawansowanych operacjach przetwarzania danych mogą być istotnym ograniczeniem w aplikacjach czasu rzeczywistego.

Celem niniejszego projektu jest opracowanie inteligentnego rdzenia dalmierza ultradźwiękowego. Dalmierz ultradźwiękowy będzie sterowany za pomocą układu FPGA. Układ sprzętowy umożliwia realizację równoległą wielu zadań oraz precyzyjny pomiar czasu, który jest niezbędny w rozważanym zastosowaniu. Strukturę układu opisano za pomocą języka VHDL. Bezpośrednie odwzorowanie sprzętowe układu pomiarowego pozwala na realizację procesów pomiarowych i obliczeniowych w sposób równoległy, co przekłada się na większą szybkość działania i elastyczność. Układ realizuje zadania pomiarowe w sposób autonomiczny, nie obciążając układu mikroprocesorowego procesem pomiarowym.

|  |
| --- |
|  |

Rys. 1.1 Układ dalmierza ultradźwiękowego z serwomechanizmem przeznaczonego do obserwacji otoczenia.

W ramach pracy zaprojektowano oraz zbudowano model dalmierza ultradźwiękowego, zdolnego do pracy w czasie rzeczywistym, oraz jego integrację z systemem robota mobilnego. Zastosowanie ultradźwięków jako medium pomiarowego wynika z ich odporności na zmienne warunki oświetleniowe oraz możliwości dokładnego określenia odległości do obiektów o różnej teksturze i geometrii. Sprzętowy system pomiaru pozwala na precyzyjne określenie czasu, co jest istotne w przypadku pomiaru odległości metodą ultradźwiękową.

Prace te przyczynią się do rozwoju bardziej zaawansowanych i niezawodnych systemów nawigacyjnych, co ma istotne znaczenie zarówno w aplikacjach komercyjnych, jak i badawczych. W ramach projektu zostanie także porównana wydajność rozwiązania opartego na układzie FPGA z tradycyjnymi podejściami, co pozwoli na ocenę potencjału tej technologii w robotyce mobilnej.

Na rysunku przedstawiono schemat blokowy zaprojektowanego układu pomiarowego. Wykorzystany układ FPGA to EP4CE6E22 który jest na płytce CYCLONE IV:

|  |
| --- |
| Obraz zawierający tekst, zrzut ekranu, Karteczka samoprzylepna, Czcionka  Opis wygenerowany automatycznie |

Rys. 1.2 Schemat blokowy dalmierza ultradźwiękowego ze sprawdzaniem temperatury otoczenia oraz czujnikiem położeń skrajnych.

W programie do wyświetlania wyników użyto wyświetlacz 7 segmentowy – w przyszłości można użyć lcd, co wpłynie na poprawę liczby wyświetlanych komunikatów. Poniżej schemat ideowy proponowanego rozwiązania.

|  |
| --- |
| Obraz zawierający tekst, zrzut ekranu, elektronika, maszyna  Opis wygenerowany automatycznie |

Rys. 1.3 Widok poglądowy na płytkę rozwojową CycloneIV z dalmierzem ultradźwiękowym na serwomotorze.

Dalmierz ultradźwiękowy, w omawianym układzie, składa się z czujnika HC-SR04, zamontowanego na obracającym się serwomechanizmie SG90, co umożliwia pomiar odległości w zakresie 180 stopni. Serwo SG90 obraca sensor w zadanym zakresie kątowym, a czujnik HC-SR04 wysyła impulsy ultradźwiękowe i odbiera echo odbite od przeszkód w otoczeniu. Na podstawie czasu powrotu sygnału obliczana jest odległość do przeszkody.

|  |
| --- |
|  |

Rys. 1.4 Odbicie sygnału dźwiękowego od przeszkody, na podstawie czasu pomiędzy wysłaniem i odbiciem sygnału wyznaczony zostaje dystans pomiędzy nimi.

Podczas działania, serwomechanizm przemieszcza czujnik w kolejnych pozycjach kątowych, a w każdym z nich wykonywany jest pomiar odległości. Dane są następnie zapisywane i używane do tworzenia mapy otoczenia oraz przetwarzane w celu orientacji robota w przestrzeni.

|  |
| --- |
| Obraz zawierający diagram, linia, zrzut ekranu, tekst  Opis wygenerowany automatycznie |

Rys. 1.5 Zakres prowadzonych pomiarów przez dalmierz ultradźwiękowy

W celu precyzyjnej kontroli ruchu serwomechanizmu i uzyskania pełnego zakresu 180 stopni, konieczne jest monitorowanie faktycznego położenia serwa. Serwomechanizm taki jak SG90 wykorzystują sygnał PWM do określenia oczekiwanego położenia. Układ mnechaniczny złożony z silnika prądu stałego oraz przekładni wymaga pewnego czasu, aby osiągnąć zadaną pozycję. Na dokładność pozycjonowania wpływa również układ potencjometru służący do określenia położenia wału jak i własności układu sterowania a szczególnie błędy wnoszone przez układ porównania położenia zadanego i bieżącego.

Podczas ruchu mechanizmu, może dojść do błędów obliczeń związanych z niedokładnym ruchem mechanizmu. Opóźnienie jakie wynika z czasu wysyłania sygnału może doprowadzić do niepełnego lub niedokładnego ustawienia układu pomiarowego, co skutkuje trudnością w otrzymaniu odpowiedniego konta obserwacji. Szczególnie jest to widoczne w położeniach skrajnych (0 stopni oraz 180 stopni). Aby temu zapobiec konieczne jest sprawdzanie położenia serwomechanizmu w czasie rzeczywistym.

Rozwiązaniem tego problemu jest zastosowanie czujnika położenia, który pozwala dokładnie odczytać rzeczywisty kąt położenia wałka serwomechanizmu. Dzięki temu możliwa jest synchronizacja pomiarów HC-SR04 z faktyczną pozycją sensora, co zapewnia:

* **pełny zakres obrotu** — umożliwiający pokrycie całego pola 180 stopni.
* **poprawę precyzji odwzorowania** — pomiary są wykonywane dokładnie w przewidzianych pozycjach kątowych.
* **redukcję błędów czasowych** — eliminuje problem nieodpowiedniego ustawienia sensora podczas wysyłania sygnałów ultradźwiękowych.

Czujnik położenia może być realizowany na kilka sposobów, jednym z nich jest zastosowanie magnetycznego czujnika zbliżeniowego w celu ustalenia pozycji skrajnych układu pomiarowego.

# Analiza tematu

## Wybrane metody pomiaru odległości robota mobilnego od przeszkód.

Autonomiczne roboty mobilne znajdują coraz większe zastosowanie w przemyśle jak i w naszych domach. W tym celu roboty muszą być zdolne do obserwacji otoczenia. Obserwacja umożliwia odnajdywanie i rozpoznawanie przeszkód, oraz określić odległość od wykrytych obiektów. Jeśli obszar, na którym robot ma pracować jest stały, wtedy można wstępnie ustalić ścieżkę i kontrolować np. wartości skrajne ustalenia robota. Jeśli natomiast środowisko, w którym pracować ma robot jest zmienne, na przykład znajdują się w nim inne poruszające się obiekty, lub sam robot porusza się po nieznanym obszarze, konieczne jest wykrywanie i określenie położenia w czasie rzeczywistym odległości od pojawiających się przeszkód. W tym celu mogą zostać wykorzystane czujniki odległościowe, osadzone na głowicy umożliwiającej „poszukiwanie” przeszkód i ocenienie odległości i konta położenia danej przeszkody.

## Przykładowe rodzaje czujników odległości

Zastosowanie czujników odległości jest niezbędne do pozyskiwania informacji o otoczeniu. W zależności od warunków pracy urządzenia, wykorzystać można kilka typów czujników odległości wymienionych poniżej.

**Czujniki indukcyjne**

Indukcyjny, cyfrowy czujnik zbliżeniowy działa jak wykrywacz metali ferromagnetycznych. Zakres detekcji to odległość od krańca czujnika do maksymalnej odległości, z jakiej czujnik może wykryć metalowy element (najczęściej znacznik). Odległość ta może różnić się w zależności od rodzaju materiału, lecz nie przekracza kilku milimetrów co wynika z dużego oporu magnetycznego powietrza. Wykrywalność jedynie przeszkód metalowych znajdujących się w niewielkiej odległości jest predestynuje ten czujnik do zastosowań, gdzie nie jest konieczny pomiar odległości a raczej wykrywanie znaczników. Przykładowo czujniki takie znajdują zastosowanie w silnikach spalinowych do określenia położenia wału korbowego na podstawie znaczników (wypustek) na kole zamachowym.

**Czujniki pojemnościowe**

**Kolejnym typem wykorzystywanym do wykrywania obiektów może być czujnik pojemnościowy. Działa on podobnie do czujników indukcyjnych natomiast materiałem, który wykrywa, jest materiał przewodzący prąd. Także odległość wykrywania obiektów jest niewielka co wyklucza jego wykorzystanie do zastosowań w robotyce mobilnej w celu obserwacji otoczenia.**

**Czujniki ultradźwiękowe**

**Wykorzystywanym w tym projekcie typem czujników jest czujnik ultradźwiękowy. Zasada działania tego typu czujników to wysyłanie krótkiej paczki impulsów o częstotliwości 40-250 kHz, a następnie nasłuchiwaniu echa odbitego od przeszkody. Na podstawie czasu potrzebnego na propagację fali oblicza się odległość do przeszkody. Zaletą tego typu czujników jest odporność na warunki oświetleniowe oraz duża precyzja wykrywania odległości do przeszkody. Korzystając z efektu Dopplera można też wykryć kierunek poruszających się przedmiotów. Podczas pomiarów należy uwzględnić też warunki temperatury czy ciśnienia otoczenia, Fala dźwiękowa porusza się z różną prędkością w zależności od ty czynników. Wadą natomiast może być duża zależność od kąta padania sygnału na przeszkodę, istotny jest też typ materiału z jakiego zrobiona jest przeszkoda.**

**Czujniki optyczne**

**Podobnie do ultradźwiękowego czujnika odległości, działają czujniki optyczne. Różnicą jest, że zamiast falami dźwiękowymi do pomiaru czasu odbicia wykorzystywane są fale elektromagnetyczne położone najczęściej w zakresie bliskiej podczerwienie co eliminuje zakłócenia spowodowane światłem widzialnym. Problemem jednak może być większy koszt tych czujników, zależność od warunków oświetlenia czy też promieniowania zakłócającego (podczerwonego). Wpływ na możliwość i zakres pomiaru ma materiał, z którego zbudowana jest przeszkoda szczególnie zdolność odbicia lub absorbcji promieniowania wykorzystywanego w pomiarach np. problem z wykrywaniem przezroczystych przeszkód. W odróżnieniu do metod ultradźwiękowych mogą działać w próżni.**

**Czujniki dotykowe**

**Czujniki te reagują dopiero po bezpośrednim fizycznym kontakcie z przeszkodą. Projekt wyklucza możliwość fizycznego kontaktu z przeszkodą, gdyż czujnika ma umożliwić omijanie przeszkód**

## Wybrane sposoby poruszania głowicą czujnika odległościowego.

**Silniki krokowe**

Silnik krokowy to silnik elektryczny, w którym impulsowe zasilanie powoduje, że jego wirnik nie obraca się ruchem ciągłym, lecz wykonuje za każdym razem obrót o ściśle określony kąt. Możliwości ustalania kąta przy pomocy tych silników sprawiają, że są idealnym rozwiązaniem dla układów wymagających precyzyjnego pozycjonowania. Dobre silniki krokowe mają dokładność rzędu 3-5% i błąd ten nie kumuluje się z kroku na krok. Zaletą też może być szeroki zakres prędkości obrotowych, możliwość szybkiego rozbiegu, hamowania i zmiany kierunku, czy proste sterowanie.

**Serwomechanizm**

Silniki serwo prądu przemiennego charakteryzują się stabilną pracą, dobrą sterownością szybką reakcją i wysoką czułością nie mniej niż 10 – 15 %. Zaletami silników serwo są dokładna kontrola prędkości prosta zasada sterowania oraz niska cena. Do terowania serwomechanizmem używany jest sygnał PWM który w zależności od długości trwania, ustawia określony kąt przesunięcia, w zakresie 0 do 180 stopni, wyróżnia się trzy metody sterowania serwomotorem: Kontrola momentu obrotowego, sterowanie pozycją lub tryb prędkości. W przypadku serwomechanizmu SG90 sterowanie odbywa się za pomocą zewnętrznego sygnału PWM który to na podstawie wypełnienia ustawia pozycję serwomechanizmu.

## Teoretyczna analiza wpływu temperatury i wysokości na prędkość fali ultradźwiękowej

Prędkość propagacji fali ultradźwiękowej w powietrzu jest ściśle zależna od parametrów środowiskowych, takich jak temperatura i ciśnienie atmosferyczne. Parametry te zmieniają się wraz z wysokością, co sprawia, że w różnych warunkach prędkość fali może przyjmować różne wartości. Poniższa analiza bazuje na podstawowych zależnościach akustycznych i danych dotyczących zmian prędkości dźwięku wraz z temperaturą i ciśnieniem, które znajdują się w takich źródłach, jak "Acoustics" Jaya Kadisa z Uniwersytetu Stanforda oraz materiały dostępne na stronach internetowych  [1] [2]

Prędkość fali dźwiękowej (w tym ultradźwięków) w powietrzu można obliczyć na podstawie równania:

gdzie:

* v = to prędkość dźwięku w powietrzu,
* γ - to stosunek ciepła właściwego powietrza przy stałym ciśnieniu do ciepła przy stałej objętości (dla powietrza wynosi on około 1,4),
* R - to uniwersalna stała gazowa dla powietrza (około 287 J/(kg·K)),
* T - to temperatura powietrza w skali Kelvina (K).

Z powyższego równania wynika, że prędkość dźwięku jest proporcjonalna do pierwiastka kwadratowego z temperatury. Wzrost temperatury powoduje wzrost energii kinetycznej cząsteczek, co prowadzi do szybszego przekazywania energii między nimi, zwiększając tym samym prędkość propagacji fali dźwiękowej. Praktycznie, przy wzroście temperatury o 1°C, prędkość dźwięku wzrasta o około 0,6 m/s.

Przykładowe obliczenie wpływu temperatury na prędkość dźwięku. Jeśli przyjmiemy temperaturę T = 293 K (20°C), możemy obliczyć prędkość dźwięku:

≈ 343m/s

Gdy temperatura wzrośnie do T=313 K(40°C):

≈ 354m/s

Zmiana temperatury o 20°C spowodowała wzrost prędkości fali dźwiękowej o około 11 m/s.

Wpływ wysokości (ciśnienia) na prędkość fali ultradźwiękowej w miarę wzrostu wysokości nad ziemią zmniejsza się gęstość i ciśnienie powietrza. W typowych warunkach atmosferycznych, przy niskich wysokościach, prędkość dźwięku zależy przede wszystkim od temperatury, a wpływ ciśnienia jest stosunkowo niewielki. Wynika to z faktu, że zmiana ciśnienia wpływa równocześnie na gęstość powietrza, co wzajemnie się kompensuje w równaniu prędkości dźwięku dla gazów idealnych.

W wyższych warstwach atmosfery, gdzie spada zarówno temperatura, jak i ciśnienie, prędkość fali ultradźwiękowej będzie mniejsza. Na przykład, w warunkach standardowych przy 0°C i ciśnieniu 1013 hPa prędkość dźwięku wynosi około 331 m/s. Na wysokości około 10 000 m, gdzie temperatura spada do około -50°C, a ciśnienie do około 250 hPa, prędkość dźwięku spada do około 295 m/s.

Pomimo tego, że prędkość dźwięku nie jest bezpośrednio zależna od ciśnienia przy stałej temperaturze, spadek ciśnienia na dużych wysokościach wpływa na rozchodzenie się fal ultradźwiękowych. Na wysokości 10 000 m fale ultradźwiękowe propagują się wolniej niż na poziomie morza, co należy uwzględniać w projektowaniu urządzeń i sensorów pracujących na różnych wysokościach.

Zarówno temperatura, jak i wysokość nad poziomem morza znacząco wpływają na prędkość rozchodzenia się fal ultradźwiękowych. Podczas gdy wzrost temperatury powoduje wzrost prędkości fali, wzrost wysokości nad ziemią — ze względu na spadek temperatury i ciśnienia — przyczynia się do jej zmniejszenia. Przy projektowaniu urządzeń ultradźwiękowych lub analizie propagacji fali należy zatem uwzględniać te czynniki, aby zapewnić dokładność i efektywność działania systemów opartych na falach ultradźwiękowych.

## Ultradźwiękowy czujnik odległości

Ultradźwiękowe czujniki odległości, takie jak HC-SR04, opierają się na zjawiskach fizycznych wykorzystujących fale dźwiękowe o wysokiej częstotliwości, które są niesłyszalne dla ludzkiego ucha. Oto szczegółowy opis ich działania i właściwości.

Ultradźwiękowy czujnik odległości mierzy odległość na podstawie czasu przelotu fali ultradźwiękowej. Generuje on falę dźwiękową, która przemieszcza się w powietrzu, odbija się od przeszkody, a następnie wraca do odbiornika. Czas pomiędzy wysłaniem a odebraniem fali pozwala wyliczyć odległość na podstawie prędkości rozchodzenia się fali dźwiękowej w danym ośrodku. Częstotliwość fal ultradźwiękowych waha się od 20 kHz do nawet 50 MHz. W zależności od zastosowania dobiera się odpowiednią częstotliwość sygnału co odpowiada również długości fali. W przypadku urządzeń USG w celu uzyskania precyzyjnego obrazu narządów stosuje się głowice pracujące na wysokich częstotliwościach. W zastosowaniu przeznaczonym do wykrywania przeszkód częstotliwość fali wykorzystywanej do pomiaru będzie się różnić w zależności od wielu czynników środowiskowych. Prędkość dźwięku w powietrzu wynosi około 340 m/s, ale zależy od takich czynników jak temperatura, wilgotność, ciśnienie atmosferyczne i gęstość ośrodka, w który rozchodzi się fala.

Czujniki ultradźwiękowe charakteryzują się dużą niezależnością od rodzaju materiału i koloru powierzchni, na którą pada fala. Nie są one również podatne na warunki optyczne (np. oświetlenie), co czyni je wszechstronnym rozwiązaniem w różnych środowiskach. Przeszkodami mogą okazać się **materiały absorbujące wibracje**: Przedmioty wykonane z materiałów pochłaniających fale dźwiękowe mogą powodować zakłócenia w pomiarach. Oraz **czynniki środowiskowe**: Parametry takie jak wilgotność czy temperatura wpływają na prędkość dźwięku, co może wymagać kalibracji czujnika.

Zastosowaniem Czujników ultradźwiękowych jest np. Robotyka, Przemysł czy Monitorowanie poziomu cieczy. Czujniki ultradźwiękowe są powszechnie używane w robotyce do wykrywania przeszkód i nawigacji bezkontaktowej, wykorzystywane są do kontroli jakości, wykrywania nieciągłości w materiałach, pomiaru grubości (np. w detekcji korozji) oraz pozycjonowania narzędzi i materiałów, lub detekcji poziomu wody i innych cieczy w zbiornikach.

Czujniki mogą mieć wyjście cyfrowe (przełączające) lub analogowe, zależnie od producenta i modelu. Zakres pomiaru wynosi zazwyczaj od 30 mm do 10 m, przy czym specyfika zależy od typu i konstrukcji urządzenia.

Ultradźwiękowe czujniki są uniwersalnym narzędziem do pomiarów odległości i detekcji obiektów. Dzięki ich niezależności od czynników optycznych i możliwości dostosowania do różnych warunków są stosowane w wielu dziedzinach, od prostych aplikacji konsumenckich po zaawansowane systemy przemysłowe i badawcze. Jednakże należy uwzględniać ograniczenia wynikające z absorpcji dźwięku przez niektóre materiały oraz wpływ zmiennych środowiskowych na dokładność pomiaru. [3]

## Serwomechanizm SG90

Serwomechanizmy takie jak użyty w tym projekcie SG90 są powszechnie używanymi, niewielkimi i precyzyjnymi silnikami używanymi w robotyce, modelarstwie i Automatyce. SG90 to popularny model mikro-serwomechanizmu, który działa na zasadzie precyzyjnego sterowania kątem obrotu za pomocą sygnału PWM (Pulse Width Modulation). Wewnątrz znajduje się mały silnik prądu stałego, który umożliwia ustawienie osi mechanizmu. Przekładnia redukcyjna zmniejsza prędkość obrotową silnika i zwiększa moment obrotowy, co umożliwia precyzyjne sterowanie pozycją. Potencjometr monitoruje aktualny kąt obrotu wału i przesyła tę informację do kontrolera. Układ elektroniczny porównuje sygnał sterujący PWM z pozycją zadaną, dostosowując napięcie zasilające silnik, aby osiągnąć żądany kąt. Mechanizm posiada też wał, który umożliwia wygodne podłączenie do serwomotoru naszego układu pomiarowego.

Serwomechanizm SG90 jest sterowany za pomocą sygnału prostokątnego o regulowanym współczynniku wypełnienia i częstotliwości około 50 Hz. Czas trwania stanu wysokiego określa żądany kąt ustawienia wału wyjściowego i zawiera się w granicach od 1ms do 2ms. Sygnał okresowy, w którym czas trwania stanu wysokiego wynosi 1 ms (1000 µs) odpowiada skrajnej pozycji wału 0° (minimalny kąt). Gdy w przebiegu czas trwania stanu wysokiego będzie wynosił 1,5 ms (1500 µs) nastąpi ustawienie wału w pozycji neutralnej (90° w SG90). Przebieg okresowy o czasie trwania stanu wysokiego 2 ms (2000 µs) ustawia wał serwomechanizmu w położeniu 180° (maksymalny kąt). Szerokość tego impulsu regulujemy zewnętrznie na przykład przy pomocy wyjścia układu FPGA. Układ sterujący w serwomechanizmie całkuje sygnał wejściowy uzyskując oczekiwane napięcie położenia wału i porównuje z sygnałem zwrotnym położenia bieżącego. Położenie bieżące jest określane za pomocą potencjometru, którego suwak jest połączony z wałem serwomechanizmu. Napięcie wyjściowe z potencjometru jest porównywane z wartością oczekiwaną. W przypadku różnicy, układ sterujący uruchamia silnik doprowadzając do zrównoważenia napięć pozycji żądanej i pozycji bieżącej.

Zakres ruchu: Typowo od 0° do 180°.

Zasilanie: Napięcie pracy 4,8–6,0 V.

Moment obrotowy: Około 1,8 kg·cm przy 4,8 V.

Rozmiar i waga: Kompaktowe wymiary i waga około 9 g.

Łatwość sterowania: Może być bezpośrednio sterowany z mikrokontrolera. [4]

# Wymagania i narzędzia

## Założenia projektu

Do wykonania projektu potrzeba odpowiednich narzędzi. Sprzęt: Czujnik ultradźwiękowy HC-SR04 działa na zasadzie wysyłania i odbierania fali ultradźwiękowej do pomiaru odległości. Czujnik posiada następujące wyprowadzenia: VCC (zasilanie, 5V), GND, TRIG żądanie wysłania paczki pomiarowej impulsów, ECHO wyjście sygnalizujące odebranie sygnału odbitego od przeszkody. Czas upływający pomiędzy impulsem na wejściu TRIG oraz ustawieniem wyjścia ECHO pozwala na określenie odległości przeszkody, od której nastąpiło odbicie sygnału pomiarowego.

|  |
| --- |
| Obraz zawierający głośnik, elektronika, pudło rezonansowe, Urządzenie elektroniczne  Opis wygenerowany automatycznie |

Rys. 3.1 Dalmierz ultradźwiękowy HC-SR04.

Serwomechanizm SG90 do ustawiania kąta pomiaru. Sterowany sygnałem PWM (0°–180°).

|  |
| --- |
|  |

Rys. 3.2 Rysunek ideowy serwomotoru SG90 z osadzonym dalmierzem.

Moduł uruchomieniowy z układem FPGA Cyclone IV wymaga narzędzi do przeprowadzenia procesu odwzorowania abstrakcyjnej reprezentacji układu logicznego w odpowiedni ciąg informacji konfigurującej zasoby programowalne układu tak aby realizowały funkcje określone przez użytkownika za pomocą odpowiedniego schematu ideowego czy też przedstawionego za pomocą języka opisu sprzętu. Najczęściej narzędzia dostarczane są przez producenta układu FPGA i tak w przypadku układu Cyclone IV narzędziem do prowadzenia odwzorowania technologicznego jest pakiet Quartus. Zawiera on wszystkie niezbędne narzędzia do prowadzenia efektywnego procesu implementacji układów cyfrowych w strukturach programowalnych FPGA firmy Altera. Stabilne źródło zasilania 5V dla HC-SR04 i SG90 (może być osobny moduł zasilania dla serwa, aby uniknąć przeciążeń). Kable połączeniowe do podłączenia czujnika i serwomechanizmu do FPGA.

Oprogramowanie: Intel Quartus Prime, do projektowania i syntezy układu w języku VHDL. ModelSim, do symulacji zachowania kodu. Sterownik USB-Blaster do programowania płytki FPGA. Oscyloskop lub analizator logiczny do debugowania sygnałów PWM i pomiarowych. [4] [5] [3]

## Platforma

**Układ programowalny FPGA**

Moduł uruchomieniowy do programowania układu FPGA za pomocą Altera blaster. Moduł uruchomieniowy oparty o układ Altera EL4CE6E22C8. Bogate wyposażenie oraz liczba interfejsów pozwalają na tworzenie zaawansowanych układów, oraz nauki sposobu obsługi oraz działania układów FPGA. Wbudowana pamięć SDRAM 64MB, pamięć Flash oraz EEPROM. Wszystkie porty wejściowe/wyjściowe zostały wyprowadzone dzięki czemu w łatwy sposób można rozbudować układ o dodatkowe moduły współpracujące z układem. Na płytce znajdują się 4 przyciski oraz 4 diody LED, które można wykorzystać programując układ.

|  |
| --- |
| Obraz zawierający obwód, elektronika, Inżynieria elektroniczna, Komponent elektroniczny  Opis wygenerowany automatycznie |

Rys. 3.3 ALTERA Cyclone IV EP4CE6 FPGA zestaw deweloperski.

Płytka zawiera:

* układ FPGA firmy ALTERA rodziny Cyclone IV EP4CE6E22C8N
* pamięć EEPROM do przechowywania konfiguracji układu FPGA EPCS16N
* programator ALTERA JTAG blaster do programowania układu FPGA oraz pamięci szeregowej EEPROM
* pamięć SDRAM o pojemności 64Mbit
* oscylator kwarcowy 50MHz dostarczający podstawowego sygnału zegarowego
* wyprowadzenia wejścia-wyjścia układu FPGA dostępne za pomocą złącz szpilkowych typu gold pin
* zestaw układów peryferyjnych takich jak wskaźniki diodowe, przyciski, wyświetlacze siedmiosegmentowe [5]

|  |
| --- |
| Obraz zawierający tekst, zrzut ekranu, Inżynieria elektroniczna, obwód  Opis wygenerowany automatycznie |

Rys. 3.4 Opis płyty Altera Cyclone IV.

**Dane techniczne:**

* moduł uruchomieniowy - deweloperski Altera CYCLONE IV
* układ oparty o Altera EL4CE6E22C8
* złącze do podłączenia LCD wsparcie dla LCD 1602 lub 12864
* wyświetlacz 7- segmentowy
* regulacja kontrastu za pomocą potencjometru
* interfejs PS2
* interfejs RS232
* układ komunikacji szeregowej SP3232
* czujnik temperatury: LM75A
* sygnalizator dźwiękowy
* przełączniki DIP Switch
* interfejs VGA
* odbiornik IR
* 4 przyciski
* EEPROM AT24C08
* 4 diody LED
* SDRAM 64MB [5]
* I/O złącza kołowe
* przycisk RESET
* JTAG interfejs
* kwarc 50MHz
* pamięć flash M25P16
* interfejs AS
* wbudowane stabilizatory napięcia 1,2V 3,3V, 2,5V
* złączenia zasilania DC Jack 5,5/2,5 mm
* napięcie zasilania: 5V
* włącznik/wyłącznik modułu
* gniazdo zasilania USB

**Zestaw zawiera:**

* moduł uruchomieniowy z Altera CYCLONE IV
* przewód USB
* pilot IR
* interfejs zgodny z USB Blaster [5]

**HC-SR04**

Czujnik ultradźwiękowy działający w zakresie 2-200 cm. Zasilany napięciem 5 V. Wyjściem jest sygnał, którego czas trwania jest proporcjonalny do mierzonej odległości.

Aby rozpocząć pomiar należy podać na pin TRIG impuls napięciowy (stan wysoki 5V) przez 10 us. Moduł dokonuje pomiaru odległości przy pomocy fali dźwiękowej o częstotliwości 40 kHz. Do mikrokontrolera wysyłany jest sygnał, w którym odległość zależna jest od czasu trwania stanu wysokiego i można ją obliczyć ze wzoru:

gdzie:

*d* - odległość mierzona

- czas trwania stanu wysokiego

- prędkość rozchodzenia się fali dźwiękowej w powietrzu - 340 m/s

W skrócie, aby otrzymać wynik w cm można wykorzystać wzór:

Specyfikacja ultradźwiękowego czujnika HC-SR04 - justPi

Napięcie zasilania: 5 V

Średni pobór prądu: 15 mA

Zakres pomiarowy: od 2 cm do 200 cm

Wyjście: sygnał częstotliwościowy

Częstotliwość pracy: 40 kHz

Wymiary: 45 x 20 x 15 mm [3]

**SG90**

Serwomechanizm SG90 micro to popularny wybór wśród modelarzy i entuzjastów robotyki dzięki swojej niezawodności i wszechstronności. Oferuje kąt obrotu 180°, co pozwala na precyzyjne sterowanie w szerokim zakresie zastosowań. Kompaktowe wymiary oraz lekka konstrukcja sprawiają, że serwomechanizm jest idealne do małych projektów związanych z robotyką, modelarstwem czy pojazdami RC. Moment obrotowy wynosi 1,8 kg\*cm przy napięciu 4,8 V. W zestawie znajdują się śrubki montażowe oraz orczyki. Serwo TowerPro SG90 micro oferuje precyzyjne sterowanie, co sprawia, że jest idealne do szerokiego zakresu zastosowań. Modelarstwo - sterowanie ruchomymi elementami modeli samolotów, samochodów i innych Robotyka - napęd i kontrola ruchu ramion robotów, chwytaków oraz innych mechanizmów Projekty DIY - automatyzacja ruchomych elementów w projektach elektronicznych Systemy RC (zdalnie sterowane) - kontrola układów kierowniczych, przepustnic i innych mechanizmów. [4]

## Wybór metodologii projektowania układu

**VHDL czy Verilog? (z artykułu w nandland)**

Obecnie do dyspozycji projektanta układów sprzętowych dostępne są dwa języki opisu sprzętu. Wybór między VHDL (Very High Speed IC Hardware Description Language) a Verilog jest jednym z pierwszych kroków dla inżynierów FPGA. Oba języki są popularne, ale różnią się pod względem składni, przeznaczenia i przydatności w różnych zastosowaniach. Poniżej zamieszczono bardzo krótkie porównanie tych dwóch języków, które pozwoliło dokonać wyboru sposobu opisu projektu.

Język VHDL objęty jest normą IEEE1076 i posiada składnię podobną do języków ADA lub Pascal. Jest bardzo restrykcyjny pod kątem zgodności typów, co czyni go szczególnie odpowiednim dla projektów wymagających wysokiej niezawodności (np. w przemyśle lotniczym, medycznym). W założeniach projektantów miał charakter samodokumentujący, co powoduje, że charakteryzuje się małą zwięzłością (można też uznać to za pewną nadmiarowość) opisu w konsekwencji wymagając od projektanta wytworzenia relatywnie długiego opisu. Wykorzystaniu takiej koncepcji przyświecał cel ograniczenie błędów podczas tworzenia opisu wymuszającego bardzo restrykcyjne przekształcanie typów. Język jako taki definiuje niewielki zestaw typów podstawowych, które nie umożliwiają pełnego modelowania systemów cyfrowych. Typy wykorzystywane w modelowaniu układów cyfrowych należą do odrębnej biblioteki objętej oddzielnym standardem IEEE1164. W konsekwencji narzędzia syntezy logicznej dla języka VHDL domyślnie traktują standard języka IEEE1076 oraz standard biblioteki podstawowej IEEE1164 jako nierozdzielną całość umożliwiając przeprowadzenie procesu syntezy. Jako przykład można podać kwestie implementacji operatorów arytmetycznych, którego odwzorowania nie odnajdujemy w bibliotece IEEE1164, ale jest już kwestią odwzorowania technologicznego zaprojektowaną przez twórców narzędzia syntezy.

Język Verilog objęty jest normą IEEE1364, która później została zastąpiona normą IEEE1800 (SystemVerilog). Najprawdopodobniej język łączy wiele doświadczeń pochodzących z narzędzi tworzonych przez producentów układów scalonych na przestrzeni wielu lat. Składnia języka została zainspirowana językiem C, co czyni go bardziej intuicyjnym dla początkujących projektantów. W odróżnieniu od języka VHDL w pełni definiuje wartości logiczne oraz metodologię modelowania układów cyfrowych. Zwięzłość opisu wymaga utworzenia krótszego opisu. W składni języka zawarto wiele działań domyślnych i implikujących pewne konstrukcje, które potencjalnie mogą stanowić źródło błędów.

VHDL dominuje w Europie, Azji i sektorach związanych z wysokimi wymaganiami niezawodności, podczas gdy Verilog jest bardziej popularny w USA i stosowany w projektach konsumenckich i komercyjnych (np. sprzęt komputerowy).

Verilog sprzyja szybkiemu prototypowaniu, a VHDL wymusza bardziej precyzyjne podejście, co jest kluczowe w projektach krytycznych. Gdy priorytetem jest jednak niezawodność i czytelność kodu, a także konieczność ścisłego testowania, w projektach dla przemysłu wojskowego, medycznego lub lotniczego, gdzie standardy bezpieczeństwa są rygorystyczne, lepszym wyborem jest VHDL. Niektóre środowiska pracy lub preferowane narzędzia (np. Mentor Graphics, Xilinx Vivado) wspierają głównie VHDL.

Varilog może być odpowiednim wyborem dla projektów konsumenckich lub komercyjnych, gdzie szybki czas realizacji jest kluczowy, lub jeśli posiadasz doświadczenie w językach C/C++, oraz przypadku pracy nad projektami związanymi z ASIC (układy scalone aplikacji specjalnych), gdzie Verilog jest standardem.

Zarówno VHDL, jak i Verilog są wspierane przez większość narzędzi FPGA, takich jak Xilinx Vivado czy Intel Quartus. Wybór języka zależy często od wymagań projektu lub preferencji zespołu. [6]

## Narzędzie implementacji układowej

Quartus Prime to środowisko programistyczne firmy Intel (wcześniej Altera), które służy do projektowania układów cyfrowych implementowanych w programowalnych układach FPGA (Field-Programmable Gate Array) oraz CPLD (Complex Programmable Logic Device). Jest to kompleksowe środowisko umożliwiające zarówno tworzenie projektów sprzętowych, jak i ich symulację, implementację oraz analizę. Główne funkcje Quartus Prime to projektowanie układów logicznych.

Obsługuje języki opisu sprzętu, takie jak VHDL i Verilog. Posiada graficzny edytor schematów, który pozwala na projektowanie układów za pomocą symboli logicznych. Narzędzie syntezy logicznej dokonuje odwzorowania abstrakcyjnego opisu za pomocą elementów logicznych dostępnych w wybranym układzie programowalnym FPGA lub CPLD.

W pakiecie dostarczany jest również symulator ModelSim który pozwala na symulację projektów w celu sprawdzenia poprawności działania przed oraz po implementacji. Funkcje analizy czasowej pomagają ocenić wydajność i zgodność z założonymi parametrami projektu. Do narzędzia implementacji można przekazać wymagania, które będą wpływały na sposób optymalizacji układu pod kątem szybkości, zużycia zasobów czy zużycia energii. W wyniku przeprowadzenia syntezy logicznej i implementacji układowej uzyskuje się plik konfiguracyjny będący obrazem binarnym pamięci konfiguracji układu programowalnego. W pakiecie oprogramowania znajdują się również narzędzia umożliwiające wprowadzenie pliku konfiguracyjnego do układu fizycznego za pomocą odpowiedniego programatora ogólnie znanego pod nazwą Blaster z różnymi przyrostkami np. USB Blaster czy też starszej wersji Byte Blaster. [7]

# Implementacja układu

Na rys. 4.1 przedstawiono podstawowe moduły opracowanego modułu pomiarowego. W dalszej części zostaną przedstawione własności funkcjonalne oraz sposób konstrukcji poszczególnych bloków.

|  |
| --- |
|  |

Rys. 4.1 podstawowe moduły omawianego układu

Celem niniejszego rozdziału jest szczegółowe omówienie procesu implementacji zaprojektowanego układu, począwszy od analizy wymagań, poprzez wybór odpowiednich narzędzi, aż po testowanie i weryfikację działania, która będzie mieć miejsce w kolejnym rozdziale.

Do Analizy bierzemy kilka fragmentów całego programu z pominięciem metody zapisu danych oraz sposobu prezentowania danych co nie jest esencją działania dalmierza ultradźwiękowego, i może być stosowane wymiennie, poprzez użycie ekranów LCD, lub komunikacji UART tudzież bezprzewodowej.

Do analizy użyte zostają moduły kontroli pomiaru, obliczania pozycji, układ wyzwalacza sygnału układ PWM oraz sposób obliczania odległości.

## Układ sterowania pomiarem odległości

Najważniejszym elementem naszego programu, jest opisanie zasady działania wyzwalacza krótkiego sygnału dźwiękowego dla dalmierza ultradźwiękowego. Do syntezy impulsów sterujących oraz pomiaru czasu zostanie wykorzystany sygnał zegarowy o częstotliwości 50MHz. Zliczanie impulsów pozwala ukształtować impuls TRIG z dokładnością do 20ns, a następnie dokonać pomiaru czasu, który upływa od wysłania impulsu pomiarowego i powrotu odbitego od przeszkody do czujnika. W tym celu należy zbudować licznik zliczający impulsy sygnału zegarowego upływające od momentu wyzwolenia pomiaru. Sterowanie procesem pomiarowym odbywa się za pomocą licznika „trig\_cnt” – jest to liczba impulsów określająca czas trwania stanu wysokiego sygnału TRIGGER. Zakres zliczania określony jest za pomocą „cnt\_max” określający czas wykonania pomiaru przez czujnik. Na rys. 4.xx pokazano schemat blokowy sposobu działania układu sterowania pomiarem odległości

|  |
| --- |
| Obraz zawierający diagram, tekst, zrzut ekranu, Wielobarwność  Opis wygenerowany automatycznie |

Rys. 4.2 Proces blokowy wyzwalania sygnału Trigger.

Dla ustalenia czasu trwania impulsu Trigger oraz szerokości jednego taktu szerokości impulsu w pliku Measurement control zostały zdefiniowane dwie wartości stałe trig\_cnt oraz cnt\_max. Stała trig\_cnt jest wykorzystywana do określenia czasu trwania impulsu TRIG natomiast cnt\_max określa czas trwania cyklu pomiarowego. W celu wygenerowania impulsu TRIG, którego czas trwania powinien wynosić 10us należy zliczyć 500 impulsów sygnału zegarowego. Całkowity czas pomiaru wynosi 25ms co wymaga odliczenia 1 250 000 impulsów sygnału zegarowego. Aby poprawnie zapisać obie wartości konieczne jest określenie niezbędnej liczby bitów. Przedstawienie większej wartości wymaga wykorzystania 21bitów co oznacza, że zarówno licznik jak i wartości określające wskazane momenty czasu zostaną przedstawione za pomocą liczb 21 bitowych. W przypadku języka VHDL liczby zostają przedstawione za pomocą wektora typu STD\_LOGIC\_VECTOR ograniczonego w zakresie 20 do 0.

:

|  |  |
| --- | --- |
| 51  52 | signal trig\_cnt: std\_logic\_vector(20 downto 0) := "000000000000111110100";  signal cnt\_max: std\_logic\_vector(20 downto 0) := "100110001001011010000"; |

Kod 4.1 Sygnały określające długość impulsu

Poniższy fragment przedstawia opis układu licznika, którego zakres zliczania jest określony przez stałą cnt\_max. Opis licznika jest opisany jako proces wyzwalany zdarzeniem występującym na sygnale clk. Oznacza to, że każdorazowa zmiana stanu sygnału clk spowoduje wyzwolenie procesu i jego przeliczenie. Wewnątrz procesu umieszczono wyrażenie warunkowe, którego warunkiem jest wykrycie wystąpienia narastającego zbocza na sygnale clk. Oznacza to, że procedura zliczania będzie odbywała się podczas wystąpienia narastającego zbocz sygnału clk. Wewnątrz warunku zliczania zapisano sposób zliczania. Kolejna instrukcja warunkowa porównuje zawartość licznika ze stała cnt\_max. W momencie, gdy licznik osiągnie wartość cnt\_max oraz wystąpi aktywne zbocze sygnału zegarowego do rejestru licznika zostanie wprowadzona wartość 0.

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 90  91  92  93  94  95  96  97  98  99  100  101  102  103 | Trigger\_proc: process(clk)  begin  if rising\_edge(clk) then  if count < cnt\_max then  count <= count + 1;  else  count <= (others => '0');  end if;  end if;  end process;  trigger <= '1' when (count < trig\_cnt) else '0';  end behavioral; | |  |

Kod 4.2 Poces wyzwalania impulsu

Opis działania:

1. **Główne komponenty i sygnały:**
   * Moduł Counter działa jako licznik impulsów, umożliwiający pomiar odcinka czasowego.
   * Sygnały resetCounter oraz outputCounter kontrolują stan licznika.
2. **Proces Trigger Generation:**
   * **Generowanie sygnału TRIGGER**: W momencie, gdy licznik Counter osiągnie wartość pomiędzy 0 a 10us, sygnał TRIGGER ustawia się na '1' następnie przez resztę trwania impulsu ustawia się na ‘0’.

## Opis działania Distance Calculation

Kolejną podstawową dla działania miernika ultradźwiękowego rzeczą jest obliczenia czasu jaki podróżuje sygnał, od wygenerowania przez „Trigger” aż po odbiór odpitego sygnału przez odbiornik „Echo”. Pod uwagę do obliczeń bierzemy w tym wypadku również temperaturę otoczenia, dla której w ROM zapisaliśmy tabelę z wartościami prędkości dźwięku dla określonych temperatur powietrze przy ciśnieniu atmosferycznym ok 1000hPa.

Dla uniknięcia nadmiaru obliczeń konieczne było policzenie dla każdej prędkości stosunku wyciągniętego z poniższego wzoru:*)*

Odległość jest obliczana w metrach, więc dla obliczenia odległości w centymetrach należy podzielić ją jeszcze na 100. Konieczne jest też obliczanie czasu podróży sygnału z liczby bitów licznika 50Mhz na jednostkę czasu w układzie Si, w tym wypadku sekundy.

*Czas\_podróży\_sygnału (bit) \*50Mhz = T(s)*

Czas podróży sygnału jest programowi znany, oblicza tę wartość licznik resetowany sygnałem „Trigger” i zatrzymywany/uruchamiany sygnałem „Echo”. Długość sygnału Echo w bitach musimy przeliczyć na wartości w sekundach. Stąd obliczamy stosunek x, posługując się przykładową prędkością dźwięku 340 m/s i wyłączamy ze wzoru czas trwania sygnału.

Aby uzyskać z powyższego wyniku konieczne jest przesunięcie bitowe. W tym wypadku, aby liczba była ośmiu bitową należy ją pomnożyć o mniej więcej milion. Najbliższa potęga dwójki od miliona to 2^20 = 1 048 576 i jest to liczba przez którą musimy przemnożyć wcześniejszy wynik. Aby otrzymać liczbę 8śmio bitową:

x = 0,00034 \* 2^20 = 356,51584

x = 101100100

obliczając stosunek do kolejnych wartości prędkości dźwięku mamy następującą tabelę ROM:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | type ROM\_Array is array (0 to 10) of std\_logic\_vector(8 downto 0);  constant ROM : ROM\_Array := (  "101011011", -- 331.3 m/s 0 C  "101011110", -- 334.3 m/s 5 C  "101100001", -- 337.3 m/s 10 C  "101100100", -- 340.3 m/s 15 C  "101100111", -- 343.3 m/s 20 C  "101101010", -- 346.3 m/s 25 C  "101101101", -- 349.3 m/s 30 C  "101110001", -- 352.3 m/s 35 C  "101110100", -- 355.3 m/s 40 C  "101110111", -- 358.3 m/s 45 C  "101111010" -- 361.3 m/s 50 C  ); |

Kod 4.3 tabela ROM dla kolejnych wartości prędkości dźwięku

Powyższy stosunek prędkości zostanie następnie przemnożony przez zmierzoną długość pulsu Echo. Przez konieczność poprzednich przesunięć musimy zmniejszyć wartość obliczonej całości o 2\*20, w tym wypadku zrobimy to w dwóch etapach. Najpierw zmniejszymy obliczoną długość impulsu o 12 najmłodszych bitów, a następnie zmniejszymy o 8 najmłodszych bitów wynik mnożenia, jak widać w poniższym procesie:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | Distance\_calculator : process(pulse)    -- Zmienne pomocnicze do obliczeń  variable Result : integer;--std\_logic\_vector(8 downto 0);  variable short\_width : std\_logic\_vector(8 downto 0);  variable Multiplier: std\_logic\_vector(17 downto 0);    begin    -- Gdy zakończy się impuls 'pulse', wykonywane są obliczenia  if (pulse = '0') then  -- Obliczanie odległośc w cm na podstawie czasu trwania impulsu  short\_width := Pulse\_width(20 downto 12);  Multiplier := short\_width \* Sound\_speed;  Result := to\_integer(unsigned(Multiplier(16 downto 8)));    -- Sprawdzenie zakresu odległości i ograniczenie maksymalnej wartości  if (Result > 400) then --"111001010"  Distance <= "000000000"; -- Przekroczenie zakresu  else  Distance <= STD\_LOGIC\_VECTOR(to\_unsigned(Result,9)); -- Wynik konwersji do 9-bitowej liczby unsigned  end if;  end if;  end proces |

Kod 4.4 Proces obliczający odległość od przeszkody

Opis działania:

1. **Główne komponenty i sygnały**:
   * **Counter**: komponent licznika odmierza czas trwania impulsu pulse, reprezentującego odbity sygnał ultradźwiękowy.
   * **Pulse\_width**: sygnał ten przechowuje szerokość impulsu, czyli czas trwania odbitego sygnału.
   * **Sound\_speed** – stosunek (prędkość dźwięku /100000)\*2^20 pobrany z ROM
2. **Proces Distance\_calculator**:
   * Gdy impuls pulse zakończy się (czyli pulse = '0'), następuje obliczenie odległości na podstawie zmierzonego czasu.
   * **Obliczenia odległości**: Wartość Pulse\_width jest skracana o 12 najmłodszych bitów, czyli dzielona przez 2^12, następnie mnożona razy stosunek pobrany z ROM dla określonej temperatury otoczenia. Potem wynik jest skracany o 8 najmłodszych bitów, czyli dzielony przez 2^8, łącznie wynik jest skrócony o 2^20 czyli o wartość o którą jest powiększona wartość z ROM.
   * **Ograniczenie zakresu**: Jeśli wynik przekracza wartość 400 cm, odległość jest przypisana do maksymalnej wartości 000000000 (9-bitowe unsigned).

## Sterownik położenia wałka serwomechanizmu

W tym projekcie dalmierz ultradźwiękowy osadzony jest na głowicy, która obraca dalmierzem od 0 do 180 stopni w celu poszukiwania przeszkód. Konieczne jest określenie kąta kierunku w jakim dalmierz jest skierowany w trakcie prowadzenia pomiaru. W tym celu w pliku sterującym PWM i Duty\_cycle umieściłem sygnał Step\_done, który wysyła wysoki sygnał w momencie zakończenia kroku. Pozostałe sygnały użyte to 8-śmio bitowy wektor pozycji dla ustalenia wypełnienia sygnału na tej podstawie ustalana jest długoś trwania sygnału PWM odpowiadającego za stopień wypełnienia. Entity pokazane jest w obrazie poniżej:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | entity pwm is  port (  clk : in std\_logic; -- Wejście zegara  rst : in std\_logic; -- Wejście resetu  position : in std\_logic\_vector(7 downto 0); -- Pozycja, od 0 do maksymalnej liczby kroków  step\_done: out std\_logic;  pwm : out std\_logic -- Wyjście sygnału PWM  );  end pwm; |

Kod 4.5 opis wejść I wyjść programu

Następnie ustalamy kilka stałych które mają ustalić czas trwania jednego cyklu (częstotliwość PWM) liczbę cykli na jeden krok, oraz minimalny czas trwania wysokiego impulsu.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | constant cnt\_min : std\_logic\_vector(16 downto 0) := "00100111000100000"; -- Liczba cykli dla minimalnego impulsu 20000 bo (400 \*50)/50e6 = 0.0004s  constant cycles\_per\_step : std\_logic\_vector(8 downto 0) := "110010000"; -- Liczba cykli zegara na krok pozycji ((max\_count - min\_count)/255)\*50 = 400  constant cnt\_max : std\_logic\_vector(20 downto 0) := "100110001001011010000"; -- 0,025 s 1 000 000 |

Kod 4.6 stałe ustalający czas przebiegów poszczególnych procesów

Następnie stwórzmy proces duty\_cycle który obliczy wypełnienie sygnału PWM. W sytuacji, gdy wciskamy reset, sygnał duty\_cycle ustawiany jest na wartość minimalnej pozycji. Natomiast w innym przypadku obliczany jest stosunek pobranej z rom pozycji, pomnożonej przez ilość cykli na krok i powiększonej o minimalną możliwą wartość ustawienia serwomotoru. Opisuje to poniższy rysunek:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | DUTY\_CYCLE\_PROC : process(clk)  variable Multiplier : std\_logic\_vector(16 downto 0);  begin    if rising\_edge(clk) then  if rst = '0' then  duty\_cycle <= cnt\_min; -- Ustawienie szerokości impulsu na minimalną pozycję  else  Multiplier := position \* cycles\_per\_step + cnt\_min;  duty\_cycle <= Multiplier; -- Obliczenie szerokości impulsu na podstawie pozycji  end if;  end if;  end process; |

Kod 4.7 proces duty\_cycle obliczający szerokość impulsu

Następnie ustalimy proces liczący, gdzie maksymalna wartość licznika będzie też częstotliwością działania układu. A jego kolejne wartości będą sprawdzane w procesie, który ustala czas trwania impulsu PWM. Jak widzimy w poniższym kodzie maksymalna wartość licznika jest ustalona przez zmienną cnt\_max, liczba binarna jednego miliona w przypadku zegara 50Mhz odpowiada za czas trwania jednego cyklu 0,025 sekundy, czyli nasz serwomotor działa w tym układzie z częstotliwością 40hz, co jest także wartością na jakiej działa dalmierz ultradźwiękowy. Ustawiany jest także sygnał step\_done po każdym cyklu licznika.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | COUNTER\_PROC: process(clk)  begin  if rising\_edge(clk) then  if rst = '0' then  count <= (others =>'0');  else  if count < cnt\_max then  count <= count + 1;  step\_done <= '0';  else  step\_done <= '1';  count <= (others => '0');  end if;  end if;  end if;  end process; |

Kod 4.8 proces obliczający częstotliwość

Ostatnim proces w tym pliku jest proces, który ma za zadanie ustawić wyjście PWM, które jest wysyłane do serwomotoru. Mając ustaloną wartość czasu trwania sygnału w taktach musimy ją jedynie porównać do aktualnej wartości licznika, I w wypadku gdy ta wartość jest mniejsza ustawiamy 1 na wyjściu w innym wypadku jest to 0.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | PWM\_PROC : process(clk)  begin  if rising\_edge(clk) then  if rst = '0' then  pwm <= '0'; -- Reset sygnału PWM  else  pwm <= '0'; -- Domyślnie ustawienie PWM na niski stan  if count < duty\_cycle then  pwm <= '1'; -- Sygnał wysoki, jeśli licznik jest mniejszy  end if;  end if;  end if;  end process; |

Kod 4.9 roces określający szerokość impulsu

Opis działania:

1. **Funkcja**cycles\_per\_us: Oblicza liczbę cykli zegara potrzebnych do uzyskania określonego czasu trwania impulsu w mikrosekundach.
2. **Stałe czasowe**:
   * min\_count**i**max\_count: odpowiadają liczbie cykli zegara dla minimalnej i maksymalnej szerokości impulsu PWM.
   * step\_us: krok czasowy w mikrosekundach dla pozycji między minimalną a maksymalną.
   * counter\_max: maksymalna liczba cykli dla generowania sygnału PWM z częstotliwością określoną.
3. **Proces**COUNTER\_PROC: Zlicza cykle zegara i resetuje licznik po osiągnięciu counter\_max.
4. **Proces**PWM\_PROC: Ustawia stan sygnału PWM na podstawie porównania licznika i wartości duty\_cycle.
5. **Proces**DUTY\_CYCLE\_PROC: Oblicza szerokość impulsu PWM dla danej pozycji.

## Kontrola pomiaru

W celu łatwiejszej kontroli pomiaru umieściłem procesy PWM i Trigger\_generator w jednym pliku kontrolnym, dzięki czemu łatwiej będzie kontrolować częstotliwość działania układu jak i sprawdzać sygnał końca kroku. Sygnał końca kroku używany jest do uruchamiania licznika, który wybiera kolejną wartość pobieraną z ROM, która następnie przesyłana jest do procesu PWM:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | signal step\_done : std\_logic; -- sprawdza czy krok jest skończony  signal cnt : std\_logic\_vector(5 downto 0); -- Sygnalizuje obecny stan licznika  signal rst : std\_logic;  signal count : std\_logic\_vector(20 downto 0);  signal position : std\_logic\_vector(7 downto 0); -- Pozycja w zakresie od 0 do 255  signal rom\_addr : std\_logic\_vector(7 downto 0); -- Adres dla pamięci ROM  signal rom\_data : std\_logic\_vector(7 downto 0); -- Dane z pamięci ROM  signal trig\_cnt : std\_logic\_vector(20 downto 0) := "000000000000111110100";  signal cnt\_max : std\_logic\_vector(20 downto 0) := "100110001001011010000";  begin  -- Przypisanie pozycji na podstawie wartości `rom\_data`  position <= rom\_data;  rom\_addr <= cnt\*"11"; -- Wyliczanie adresu do ROM t= 2^20 / 50e6 = 0,02s jeden krok  -- Moduł PWM odpowiedzialny za generowanie sygnału  SERVO : pwm  port map (  clk => clk,  rst => reset,  position => position,  step\_done => step\_done,  pwm => servo\_pwm  );  -- Moduł licznika używany do inkrementacji `count` z określonym zakresem  COUNTING: Counter  generic map (  6 -- Ustawia maksymalną wartość licznika  )  port map (  clk => clk,  reset => reset,  enable => step\_done, -- Licznik zawsze włączony  count => cnt -- Wartość licznika wyjściowego  );  -- Pamięć ROM przechowująca wartości sinusoidy, do generowania danych  SINE : sine\_rom  port map (  clk => clk,  addr => rom\_addr,  data => rom\_data  );  Trigger\_proc : process(clk)  begin  if rising\_edge(clk) then  if count < cnt\_max then  count <= count + 1;  else  count <= (others => '0');  end if;  end if;  end process;  trigger <= '1' when (count < trig\_cnt) else '0';  end behavioral; |

Kod 4.10 Moduł scalający i kontrolujący proces pomiaru

# Proces uruchamiania projektu

Współczesny rozwój technologii wymaga tworzenia coraz bardziej zaawansowanych układów elektronicznych, które znajdują zastosowanie w różnych dziedzinach przemysłu. Proces implementacji układowej mimo wspomagania przez zaawansowane narzędzia jest zadaniem wymagającym. Wiele elementów z pozoru drobnych może mieć istotny wpływ na jakość powstałego układu. Bardzo istotna jest znajomość architektury układów programowalnych pozwalająca na odpowiedni dobór elementów czy też sposób odwzorowania obliczeń arytmetycznych.

Celem niniejszego rozdziału jest szczegółowe omówienie procesu implementacji zaprojektowanego układu, począwszy od analizy wymagań, poprzez wybór odpowiednich narzędzi, aż po testowanie i weryfikację działania.

W pierwszej części przedstawiono specyfikację techniczną układu oraz uzasadniono wybór zastosowanych metod i narzędzi programistycznych. Następnie opisano kolejne etapy implementacji, zwracając uwagę na napotkane wyzwania i sposoby ich rozwiązania. W ostatniej sekcji omówiono wyniki testów oraz oceniono spełnienie założonych kryteriów projektowych.

Do realizacji projektu wykorzystano język VHDL oraz środowisko Xilinx ISE, które umożliwiły efektywne modelowanie i symulację układu. Wybór tych narzędzi podyktowany był ich szerokim zastosowaniem w przemyśle oraz dostępnością zasobów edukacyjnych.

Przedstawiona struktura rozdziału pozwala na kompleksowe zrozumienie procesu implementacji układu, podkreślając kluczowe aspekty techniczne oraz decyzje projektowe, które wpłynęły na ostateczny kształt rozwiązania.

## Symulacja funkcjonalna

Testowanie opisu układu cyfrowego w języku VHDL przeprowadzono za pomocą symulatora ModelSim. Umożliwia on symulację zachowania układu przy wykorzystaniu środowiska testowego generującego pobudzenia oraz weryfikującego odpowiedzi układu. Układ może zostać opisany w sposób funkcjonalny. Opis taki skupia się na funkcjonalności pomijając fizyczne własności elementów logicznych jakimi jest czas propagacji sygnału. W przypadku opisu funkcjonalnego opisanie czasu propagacji w wielu przypadkach może być trudne ze względu na abstrakcyjny opis złożonych funkcji. Poddając opis syntezie logicznej oraz odwzorowaniu technologicznemu można uzyskać model fizyczny w którym zostają uwzględnione rzeczywiste czasy propagacji poszczególnych elementów.

Dzięki temu możliwe jest wczesne wykrywanie błędów w działaniu układu, co znacznie obniża koszty i czas potrzebny na poprawki. Środowisko oferuje zaawansowane narzędzia debugowania, takie jak analiza przebiegów sygnałów, przegląd zmiennych i sygnałów w czasie rzeczywistym, punkty kontrolne oraz monitorowanie symulacji, co ułatwia diagnostykę problemów.

Pisząc testbanch należ zastanowić się które z danych wejściowych potrzebują inicjacji przez warunki zewnętrzne i spróbować je zainicjować. Pierwszym potrzebnym sygnałem zewnętrznym w naszym programie jest zegar: clock <= not clock after 10 ns;

Zmiana zegara co 10ns na wartość odwrotną, oznacza że cały okres wynosi 20 ns, czyli testowy zegar ma częstotliwość 50Mhz, jak w przypadku zegara wykorzystywanego w układzie.

Kolejnym zewnętrznym sygnałem wymaganym do poprawnego działania programu, który należy zaprogramować jest sygnał ECHO, którego długość pozwala nam mierzyć odległość do obiektu. Poniższy sygnał echo symuluje odległość do obiektu 2m, czyli czas podróży sygnału dźwiękowego od czujnika do obiektu i z powrotem powinien wynosić około 11,6 ms. Następnie sygnał Echo jest ustawiany na tę długość w stan wysoki, który to czas będzie mierzony przez moduł obliczania odległości.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | entity measurement\_tb is  end entity measurement\_tb;  architecture test of measurement\_tb is  component Measurement\_control is  port (  clk : in std\_logic;  reset : in std\_logic;  trigger : out std\_logic;  servo\_pwm : out std\_logic  );  end component;  component Distance\_calculator is  port(  clk : in std\_logic;  Calculation\_Reset : in std\_logic;  pulse : in std\_logic;  Distance : out STD\_LOGIC\_VECTOR(8 downto 0)  );  end component;  component BlackBoxRAM is  Port (  clk : in std\_logic; -- System clock  reset : in std\_logic; -- Synchronous reset  trigger : in std\_logic; -- Trigger signal  data\_in : in std\_logic\_vector(8 downto 0); -- Data to write  addr\_out : out std\_logic\_vector(15 downto 0); -- RAM address  we\_out : out std\_logic; -- Write enable  ram\_data : inout std\_logic\_vector(8 downto 0) -- Data bus to RAM  );  end component;  signal clock : std\_logic := '0';  signal reset : std\_logic := '0';  signal out\_q : std\_logic;  signal echo : std\_logic:= '0';  signal Dist : STD\_LOGIC\_VECTOR(8 downto 0);  signal position: std\_logic;  signal addr : STD\_LOGIC\_VECTOR(15 downto 0);  signal we : std\_logic;  signal data : STD\_LOGIC\_VECTOR(8 downto 0);  begin  -- Reset and clock  clock <= not clock after 10 ns;  reset <= '1', '0' after 1000 ms;  --echo <= not echo after 9 ms;  echo\_proc: process  begin  if(out\_q'event) then  wait for 11600 us;  echo <= '1';  wait for 11600 us;  echo <= '0';  end if;  wait on out\_q;  end process;  uut: Measurement\_control Port MAP (clock, reset, out\_q, position);  uut0: Distance\_calculator port map (clock, out\_q, echo, dist);  uut1: BlackBoxRAM Port map(clock, reset, out\_q, Dist, addr, we, data);  end architecture test; |

Kod 5.1 opis testu dla programu modelsim

W naszym teście ustawiamy również sygnał Reset, który na początku przez sekundę jest w stanie wysokim, po sekundzie zmienia się w stan niski i zatrzymuje symulacje. Sygnał Trigger opisany na poniższym obrazie jako out\_q wyzwala wysoki stan co 25ms który trwa 10ns. Wraz z tym sygnałem ustawiana jest pozycja serwomotoru, opisana jako sygnał position. Na poniższym teście widać również odczyt odległości po zakończeniu sygnału Echo. Pomiar wynosi w systemie binarnym 11000100, czyli 196.w dziesiętnym. Dodatkowo wraz z sygnałem wyzwalacza dokonany zostaje zapis do kolejnej komórki pamięci. Odczyt jest sterowany przez sygnał we(*write enable*) który wyzwalany jest przez krótką chwilę w trakcie stanu wysokiego out\_q.

|  |
| --- |
|  |

Rys. 5.1 przebieg symulacji

Poniżej przedstawiony jest opis wykorzystywany dla symulacji zapisu do pamięci RAM.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | architecture Behavioral of BlackBoxRAM is  -- Internal signals  signal write\_enable : std\_logic := '0';  signal write\_addr : std\_logic\_vector(15 downto 0):=(others=>'0'); -- licznik adresu  signal ram\_data\_reg :std\_logic\_vector(8 downto 0):=(others=>'0'); -- rejestr danych  signal cnt\_max : std\_logic\_vector(8 downto 0) := "111110100";  signal cnt : std\_logic\_vector(8 downto 0):= "000000000";  begin  -- Assign outputs  addr\_out <= write\_addr;  we\_out <= write\_enable;  -- Process wpisujacy dane na trigger  WriteToRAM : process(clk, reset)  begin  if rising\_edge(clk) then  if reset = '0' then  write\_enable <= '0';  write\_addr <= (others => '0');      else  if trigger = '1' then  cnt <= cnt + 1;  else  cnt <= (others => '0');  end if;  if cnt(8) = '1' then  -- Load data into RAM  write\_enable <= '1';  ram\_data\_reg <= data\_in;  write\_addr <= std\_logic\_vector(unsigned(write\_addr) + 1); -- zwi?ksza adres  cnt <= (others => '0');  else  write\_enable <= '0';  end if;  end if;  end if;  end process;  -- RAM data bus assignment  ram\_data <= ram\_data\_reg when write\_enable = '1' else (others => 'Z'); -- Z gdy nie czyta  end Behavioral; |

Kod 5.2 opis symulacji pamięci RAM

## Testowanie w układzie Cyclone IV

Kluczowym aspektem weryfikacji projektu jest jego przetestowanie na rzeczywistym fizycznym układzie. W tym celu wykorzystamy płytkę rozwojową Cyclone IV, czujnik odległości oraz serwomechanizm. Ze względu na trudność w prostym sprawdzeniu poprawności działania serwomechanizmu, na tym etapie pozostanie on odłączony. Do złożenia układu potrzebne będą również płytka stykowa oraz kilka przewodów z odpowiednimi końcówkami. W środowisku programistycznym wykorzystamy narzędzie do planowania połączeń, aby przypisać odpowiednie piny. Wejścia oraz wyjścia odpowiadające za sygnał zegara i reset wyświetlacza zostaną skonfigurowane zgodnie ze schematem połączeń właściwym dla naszej płytki rozwojowej. Pozostałe dostępne piny zostaną wykorzystane do podłączenia urządzeń zewnętrznych, zgodnie z poniższą listą:

|  |
| --- |
|  |

Rys. 5.2 Przypisanie sygnałów do wyprowadzeń fizycznych układu FPGA.

Następnie przystępujemy do kompilacji naszego programu, pierwsza kompilacja powinna odbyć się jeszcze przed planowanie podłączeń w programie PIN Planner, w innym wypadku program ten nie rozpozna zaprojektowanych podmiotów wejść i wyjść naszego programu. Następnie należy skompilować ponownie aby plik wysyłany do układu miał odpowiednio skonfigurowane wejścia. Jak wynika z raportu, wykorzystujemy zaledwie 7% elementów logicznych i mniej niż jeden 1% pamięci. Nie wykorzystujemy też żadnego wbudowanego mnożnika 9-cio bitowego. Jest to zasługą nie wykonywania w programie mnożenia z wykorzystaniem długich słów bitowych. Wykorzystujemy dzięki temu zaledwie niewielki ułamek możliwości jakie daje nam FPGA, zarówno pod względem ilości elementów logicznych, pamięci jak i podłączeń. W tym wypadku wykorzystano liczne podłączenia do obsługi wyświetlacza 7-dmio segmentowego, jednak wykorzystując magistralę I2C i wyświetlacz LCD, można by znacznie zmniejszyć ilość potrzebnych połączeń.

|  |
| --- |
|  |

Rys. 5.3 Wynik kompilacji

Następnie po uruchomieniu układu podłączamy oscyloskop by sprawdzić poprawność działania. Niestety prosty oscyloskop połączony z multimetrem który został użyty, ma problemy z rozpoznaniem sygnałów o wysokiej częstotliwości używanych w do nadawania sygnału wyzwalania lub odbiory sygnału. Jak widać na poniższych obrazkach:

|  |
| --- |
|  |

Po lewej sygnał echo, po prawej sygnał Trig

Możemy również sprawdzić sygnał PWM wysyłany do serwomechanizmu:

|  |
| --- |
|  |

Rys. 5.4 PWM, z lewej w minimalnej pozycji z prawej maksymalnej

Jeśli wszystko działa poprawnie, przystępujemy do pomiaru, wykorzystamy do tego taśmę mierniczą firmy sparta, oraz trzy rodzaje przeszkód, książkę, kubek oraz pogniecioną kartkę papieru. Przykład prowadzenia pomiaru pokazuje poniższy obrazek

|  |
| --- |
|  |
| Rys 5.6. Pomiar odległości |

Rys. 5.5 Pomiar odległości

Tabela z wynikami (w cm)

|  |  |  |  |
| --- | --- | --- | --- |
|  | książka | kubek | zgnieciona kartka |
| **zadane** | **pomiar** | **pomiar2** | **pomiar3** |
| 5 | 4 | 3 | 3 |
| 10 | 10 | 8 | 16 |
| 15 | 15 | 14 | 20 |
| 20 | 19 | 19 | 35 |
| 25 | 25 | 25 | 40 |
| 30 | 30 | 29 | 34 |
| 35 | 35 | 34 | 50 |
| 40 | 40 | 39 | 60 |
| 45 | 45 | 44 | 37 |
| 50 | 49 | 59 | 80 |
| 55 | 55 | 55 | 60 |
| 60 | 60 | 59 | 74 |
| 65 | 64 | 65 | 65 |
| 70 | 70 | 68 | 75 |
| 75 | 75 | 75 | 84 |
| 80 | 79 | 79 | 88 |
| 85 | 84 | 84 | 90 |
| 90 | 90 | 88 | 99 |
| 95 | 94 | 94 | 120 |
| 100 | 99 | 98 | 98 |
| 110 | 109 |  |  |
| 120 | 120 |  |  |
| 130 | 130 |  |  |
| 140 | 140 |  |  |
| 150 | 149 |  |  |
| 160 | 161 |  |  |
| 170 | 170 |  |  |
| 180 | 180 |  |  |
| 190 | 186 |  |  |
| 200 | 199 |  |  |

Rys. 5.6 Tabela przedstawiająca wyniki pomiarów

Rys. 5.7 Wykres prezentujące wyniki pomiarów

Na podstawie analizy przedstawionego wykresu i tabeli można stwierdzić, że dla wyznaczonych punktów pomiarowych, takich jak książka i kubek, zmierzona odległość w dużym stopniu odpowiada rzeczywistej wartości. Przy założeniu, że błędy mogą wynikać z niedokładności narzędzi pomiarowych lub ich niewłaściwego ustawienia, można uznać uzyskane wyniki za bardzo zbliżone do idealnych. Jednakże przeprowadzone obliczenia wskazują, że zaobserwowany błąd pomiarowy wynika przede wszystkim z niedokładności zastosowanych przekształceń matematycznych oraz wprowadzenia istotnych uproszczeń w modelu. Użycie długich słów bitowych (powyżej 30 bitów) umożliwiłoby znacznie precyzyjniejsze obliczenia odległości, zbliżając się do wartości idealnych. Niemniej jednak, takie podejście wymagałoby znacznych zasobów sprzętowych, w szczególności dużej liczby wbudowanych 9-bitowych mnożników, co mogłoby znacząco obciążyć układ i wpłynąć na jego efektywność operacyjną.

Stosując następującą metodę szacowania błędu:

**Średni wynik pomiaru (ꭓ)**: Średnia wszystkich odczytów w danej kategorii.

**Wartość rzeczywista (xrzeczywista)**: Odległość rzeczywista, która była mierzona.

**Błąd bezwzględny (∣E∣)**: Średnia wartość różnicy między wynikami pomiarów a wartością rzeczywistą (**ꭓ - xrzeczywista**).

**Błąd względny (%)**: Wyrażony procentowo błąd obliczany jako:

Otrzymujemy następujące błędy pomiarowe:

Książka:

Błąd bezwzględny: 0,53 cm

Błąd względny: 0,62%

Kubek:

Błąd bezwzględny: 1,45 cm

Błąd względny: 2,76%

Zgnieciona kartka:

Błąd bezwzględny: 10,1 cm

Błąd względny: 19,24%

# Wnioski

Opracowanie inteligentnego rdzenia dalmierza ultradźwiękowego to krok w kierunku zwiększenia funkcjonalności i autonomii robotów mobilnych. Projekt udowodnił skuteczność technologii ultradźwiękowej w aplikacjach robotów mobilnych, jednocześnie wskazując na wyzwania związane z jej ograniczeniami w specyficznych warunkach. Integracja dalmierza z systemami wieloczujnikowymi stanowi obiecujący kierunek dalszych badań i rozwoju. Technologia ultradźwiękowa sprawdziła się jako rozwiązanie o dużej niezawodności w standardowych warunkach. Jednak jej ograniczenia w wykrywaniu przeszkód z materiałów pochłaniających dźwięk sugerują potrzebę zastosowania systemów wspomagających, takich jak lidar lub czujniki optyczne, w szczególnie wymagających środowiskach.

## Realizacja celu pracy

Testy wykazały, że precyzja pomiaru dalmierza ultradźwiękowego jest silnie uzależniona od warunków środowiskowych, takich jak obecność przeszkód o różnych właściwościach powierzchni. W szczególności obiekty o nieregularnych kształtach (np. zgniecione kartki) oraz materiały pochłaniające dźwięk (np. tkaniny) mogą wprowadzać istotne zakłócenia. Istotnym czynnikiem jest też konieczność przekształceń wyników pomiarów, która istotnie wpływa na ostateczny zapisany wynik, lecz nie sam pomiar. Należałoby również sprawdzić wpływ temperatury i ciśnienia na wyniki pomiaru, jednak w przypadku naszego prostego układu pomiarowego było to utrudnione.

## Możliwości rozwoju

Opracowany inteligentny rdzeń dalmierza ma potencjał zastosowania nie tylko w robotach mobilnych, ale także w dronach, autonomicznych pojazdach lądowych oraz systemach bezpieczeństwa, takich jak automatyczne bramy czy czujniki parkowania.

Projekt wykazał możliwości dalszego rozwoju, w tym implementację zaawansowanych algorytmów przetwarzania sygnału (np. filtry adaptacyjne) oraz wykorzystanie uczenia maszynowego do klasyfikacji przeszkód na podstawie ich odpowiedzi ultradźwiękowej. Dalsze prace mogą również obejmować optymalizację energetyczną systemu dla zastosowań w urządzeniach zasilanych bateryjnie

# Bibliografia

|  |  |
| --- | --- |
| [1] | A. K. R. Pratap, Application of Ultrasonic Waves for the Evaluation of Mechanical Properties., Procedia Engineering Conference, 2016, pp. 153-159. |
| [2] | D. A. Russel, Absorption of Sound in Air. Demos and Simulations, Pensylvania State University, 2010. |
| [3] | S. Electronics, HC-SR04 Ultrasonic Sensor Datasheet., 2015. |
| [4] | FriendlyWire, SG90 9 g Micro Servo, 2015. |
| [5] | Gotronik, „Altera Cyclone IV EP4CE6,” [Online]. Available: https://www.gotronik.pl/altera-cyclone-iv-ep4ce6-fpga-zestaw-uruchomieniowy-p-8377.html. |
| [6] | Nandland, VHDL vs Verilog: Which Language Should You Learn First?, 2020. |
| [7] | I. Corporation, Intel Quartus Prime Documentation. Oprogramowanie Quartus Prime, 2021. |
| [8] | J. Kadis, Introduction to Acustics. Wykład CCRMA, Stanford, 2006. |
| [9] | N. Instruments, Storing Data on an FPGA Target., NI Documentation, 2021. |

Dodatki

* Źródło kodu

# Spis skrótów i symboli

Technologie i standardy

* ***VHDL*** *- VHSIC Hardware Description Language, język opisu sprzętu do projektowania układów cyfrowych.*
* ***Verilog*** *- Język opisu sprzętu do modelowania układów cyfrowych.*
* ***C/C++*** *- Języki programowania stosowane w systemach wbudowanych i ogólnym programowaniu.*
* ***ASIC*** *- Application-Specific Integrated Circuit, układ scalony dedykowany konkretnemu zastosowaniu.*
* ***CPLD*** *- Complex Programmable Logic Device, programowalny układ logiczny o średniej złożoności.*
* ***FPGA*** *- Field Programmable Gate Array, programowalny układ logiczny o dużej złożoności.*

Urządzenia i podzespoły

* ***EP4CE6E22*** *- Układ FPGA z rodziny Cyclone IV.*
* ***CYCLONE IV*** *- Rodzina układów FPGA firmy Intel (dawniej Altera).*
* ***lcd*** *- Wyświetlacz ciekłokrystaliczny (Liquid Crystal Display).*
* ***HC-SR04*** *- Ultradźwiękowy czujnik odległości.*
* ***SG90*** *- Mikroserwomechanizm.*
* ***PWM*** *- Pulse Width Modulation, modulacja szerokości impulsów.*
* ***Vcc*** *- Napięcie zasilania.*
* ***GND*** *- Masa (połączenie z potencjałem ziemi).*
* ***TRIG*** *- Sygnał wyzwalający.*
* ***ECHO*** *- Sygnał powrotny z czujnika odległości.*
* ***SDRAM*** *- Synchronous Dynamic Random-Access Memory.*
* ***EEPROM*** *- Electrically Erasable Programmable Read-Only Memory.*
* ***JTAG*** *- Joint Test Action Group, standard testowania i programowania układów scalonych.*
* ***PS2*** *- Interfejs komunikacyjny stosowany w klawiaturach i myszach.*
* ***RS232*** *- Standard komunikacji szeregowej.*
* ***SP3232*** *- Układ konwertujący napięcia interfejsu RS232.*
* ***LM75A*** *- Czujnik temperatury.*
* ***DIP Switch*** *- Przełącznik typu DIP.*
* ***VGA*** *- Standard wyświetlania obrazu (Video Graphics Array).*
* ***IR*** *- Podczerwień (Infrared).*
* ***RESET*** *- Sygnał resetujący.*
* ***USB*** *- Universal Serial Bus, standard komunikacji i zasilania.*
* ***DC*** *- Prąd stały (Direct Current).*
* ***Flash*** *- Pamięć typu flash do przechowywania danych.*

Jednostki i stałe

* ***Hz, kHz, MHz*** *- Jednostki częstotliwości: herce, kiloherce, megaherce.*
* ***hPa*** *- Hektopaskale, jednostka ciśnienia.*
* ***m/s*** *- Metry na sekundę, jednostka prędkości.*
* ***s, ns, us*** *- Sekunda, nanosekunda, mikrosekunda.*
* ***°C*** *- Stopnie Celsjusza, jednostka temperatury.*
* ***m, V, g, kg*** *- Jednostki: metr, wolt, gram, kilogram.*

Wielkości fizyczne i wzory

* ***v*** *- Prędkość dźwięku w powietrzu.*
* ***γ*** *- Stosunek ciepła właściwego powietrza przy stałym ciśnieniu do tego przy stałej objętości (ok. 1,4).*
* ***R*** *- Uniwersalna stała gazowa dla powietrza (ok. 287 J/(kg·K)).*
* ***T*** *- Temperatura powietrza w skali Kelvina (K).*
* ***d*** *- Odległość mierzona.*
* ***t\_H*** *- Czas trwania stanu wysokiego.*
* ***V\_s*** *- Prędkość rozchodzenia się fali dźwiękowej w powietrzu (ok. 340 m/s).*

Oprogramowanie i funkcje

* ***IEEE1364, IEEE1800*** *- Standardy języków opisu sprzętu: Verilog (1364) i SystemVerilog (1800).*
* ***USB Blaster, Byte Blaster*** *- Programatory do układów FPGA.*
* ***UART*** *- Universal Asynchronous Receiver-Transmitter, interfejs komunikacyjny.*
* ***STD\_LOGIC\_VECTOR, STD\_Logic*** *- Typy danych używane w języku VHDL.*
* ***clc*** *- Nazwa sygnału/zmiennej.*
* ***cnt\_max, cnt\_min*** *- Maksymalne i minimalne wartości licznika.*
* ***Measurement control*** *- Kontrola pomiaru.*
* ***ROM*** *- Read-Only Memory.*
* ***duty\_cycle*** *- Wypełnienie sygnału PWM.*
* ***ModelSim*** *- Oprogramowanie do symulacji projektów w językach VHDL i Verilog.*

# Źródła

[Kod 4.1 Sygnały określające długość impulse 27](#_Toc187660624)

[Kod 4.2 Poces wyzwalania impulsu 27](#_Toc187660625)

[Kod 4.3 tabela ROM dla kolejnych wartości prędkości dźwięku 29](#_Toc187660626)

[Kod 4.4 Proces obliczający odległość od przeszkody 30](#_Toc187660627)

[Kod 4.5 opis wejść I wyjść programu 31](#_Toc187660628)

[Kod 4.6 stałe ustalający czas przebiegów poszczególnych procesów 31](#_Toc187660629)

[Kod 4.7 proces duty\_cycle obliczający szerokość impulsu 31](#_Toc187660630)

[Kod 4.8 proces obliczający częstotliwość 32](#_Toc187660631)

[Kod 4.9 roces określający szerokość impulsu 33](#_Toc187660632)

[Kod 4.10 Moduł scalający i kontrolujący proces pomiaru 35](#_Toc187660633)

[Kod 5.1 opis testu dla programu modelsim 38](#_Toc187660634)

Kod 5.2 opis symulacji pamięci RAM……………………………………………………40

# Spis rysunków

[Rys. 1.1 Układ dalmierza ultradźwiękowego z serwomechanizmem przeznaczonego do obserwacji otoczenia. 5](#_Toc187593785)

[Rys. 1.2 Schemat blokowy dalmierza ultradźwiękowego ze sprawdzaniem temperatury otoczenia oraz czujnikiem położeń skrajnych. 6](#_Toc187593786)

[Rys. 1.3 Widok poglądowy na płytkę rozwojową CycloneIV z dalmierzem ultradźwiękowym na serwomotorze. 7](#_Toc187593787)

[Rys. 1.4 Odbicie sygnału dźwiękowego od przeszkody, na podstawie czasu pomiędzy wysłaniem i odbiciem sygnału wyznaczony zostaje dystans pomiędzy nimi. 7](#_Toc187593788)

[Rys. 1.5 Zakres prowadzonych pomiarów przez dalmierz ultradźwiękowy 8](#_Toc187593789)

[Rys. 3.1 Dalmierz ultradźwiękowy HC-SR04. 18](#_Toc187593790)

[Rys. 3.2 Rysunek ideowy serwomotoru SG90 z osadzonym dalmierzem. 19](#_Toc187593791)

[Rys. 3.3 ALTERA Cyclone IV EP4CE6 FPGA zestaw deweloperski. 20](#_Toc187593792)

[Rys. 3.4 Opis płyty Altera Cyclone IV. 21](#_Toc187593793)

[Rys. 4.1 podstawowe moduły omawianego układu 26](#_Toc187593794)

[Rys. 4.2 Proces blokowy wyzwalania sygnału Trigger. 28](#_Toc187593795)

[Rys. 5.1 przebieg symulacji 41](#_Toc187593796)

[Rys. 5.2 Przypisanie sygnałów do wyprowadzeń fizycznych układu FPGA. 42](#_Toc187593797)

[Rys. 5.3 Wynik kompilacji 43](#_Toc187593798)

[Rys. 5.4 PWM, z lewej w minimalnej pozycji z prawej maksymalnej 44](#_Toc187593799)

[Rys. 5.5 Pomiar odległości 44](#_Toc187593800)

[Rys. 5.6 Tabela przedstawiająca wyniki pomiarów 45](#_Toc187593801)

[Rys. 5.7 Wykres prezentujące wyniki pomiarów 46](#_Toc187593802)