

**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, jednocześnie wskazując 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 designed for mobile robots

**Abstract:**

The aim of the project was to develop an intelligent core of an ultrasonic rangefinder designed for mobile robots. The system was based on ultrasonic technology, enabling precise distance measurements to obstacles in dynamic environments. The rangefinder core consists of modules for controlling the emission and reception of ultrasonic pulses, as well as algorithms for calculating distances based on Time of Flight. Tests conducted on various surface types demonstrated the system's high efficiency, particularly under standard conditions. The project showcased the potential for applications in navigation systems, drones, autonomous vehicles, and industrial applications, while also identifying avenues for further development, such as algorithm optimization and the use of artificial intelligence.

**Keywords:**

Ultrasonic rangefinder, Servo, distance measurement, FPGA, VHDL

Spis treści

[1. Wstęp 4](#_Toc187089171)

[2. Analiza tematu 9](#_Toc187089172)

[2.1. Przykładowe sposoby pomiaru odległości robota mobilnego wprowadzenie. 9](#_Toc187089173)

[2.2. Przykładowe rodzaje czujników odległościowych 9](#_Toc187089174)

[2.3. Przykładowe sposoby poruszania głowicą czujnika odległościowego. 10](#_Toc187089175)

[2.4. Teoretyczna analiza wpływu temperatury i wysokości na prędkość fali ultradźwiękowej 11](#_Toc187089176)

[2.5. Ultradźwiękowy czujnik odległości 12](#_Toc187089177)

[2.6. Serwomotor SG90 13](#_Toc187089178)

[3. Wymagania i narzędzia 15](#_Toc187089179)

[3.1. Założenia projektu 15](#_Toc187089180)

[3.2. Hardware 16](#_Toc187089181)

[3.3. Software 20](#_Toc187089182)

[3.4. Narzędzie implementacji układowej 21](#_Toc187089183)

[4. Implmentacja układu 23](#_Toc187089184)

[4.1. Trigger generator 24](#_Toc187089185)

[4.2. Opis działania Distance Calculation 26](#_Toc187089186)

[4.3. Serwomotor i kontrola szerokości impulsu 29](#_Toc187089187)

[4.4. Kontrola pomiaru 33](#_Toc187089188)

[5. Weryfikacja i walidacja 36](#_Toc187089189)

[5.1. Testowanie w ModelSim 36](#_Toc187089190)

[5.2. Testowanie na Cyclone IV 39](#_Toc187089191)

[6. Wnioski 46](#_Toc187089192)

[6.1. Realizacja celu pracy 46](#_Toc187089193)

[6.2. Możliwości rozwoju 46](#_Toc187089194)

[Bibliografia 48](#_Toc187089195)

[Spis skrótów i symboli 53](#_Toc187089196)

[Źródła 54](#_Toc187089197)

[Lista dodatkowych plików, uzupełniających tekst pracy 55](#_Toc187089198)

[Spis rysunków 56](#_Toc187089199)

[Spis tablic 57](#_Toc187089200)

# 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 mapowanie otoczenia. Tradycyjne podejścia, takie jak wykorzystanie mikrokontrolerów, 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 wymagających czasu rzeczywistego.

Celem niniejszego projektu jest opracowanie inteligentnego rdzenia dalmierza ultradźwiękowego. Dalmierz ultradźwiękowy zostanie sterowany przy pomocy układu FPGA, ze względu na jego przewagi stosowane w cyfrowym przetwarzaniu sygnałów, oraz przetwarzaniu danych w czasie rzeczywistym. 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 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 wyświtlacz 7-dmio segmentowy – w przyszłości można użyć lcd, co wpłynie na poprawę ilości 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.

|  |
| --- |
| Obraz zawierający tekst, zrzut ekranu, Czcionka, Grafika  Opis wygenerowany automatycznie |
| 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. Serwa takie jak SG90 wykorzystują sygnał PWM do określenia pozycji, jednak czas reakcji mechanicznej mechanizmu oraz opóźnienia wynikające z przetwarzania danych pomiarowych mogą powodować niedokładności.

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, w jakim znajduje się serwo. 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 mapowania** — 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

## Przykładowe sposoby pomiaru odległości robota mobilnego wprowadzenie.

Autonomiczne roboty mobilne znajdują coraz większe zastosowanie w przemyśle jak i w naszych domach. W tym celu roboty muszą potrafić odnajdywać i rozpoznawać przeszkody, oraz potrafić 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 środowisku 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 odległościowych

Wykorzystanie czujników powszechnym i stosunkowo tanim sposobem pozyskiwania informacji o otoczeniu. W zależności od potrzeb i warunków środowiskowych w jakich chcemy badać otoczenie, wykorzystać można kilka możliwych 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 cel. Odległość ta może różnić się w zależności od metalu, lecz nie przekracza kilku milimetrów. Wykrywalność jedynie przeszkód metalowych jest jednak wadą tego typu rozwiązania.

**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**

**Czujniki ultradźwiękowe**

**Wykorzystywanym w tym projekcie typem czujników jest czujnik ultradźwiękowy. Zasada działania tego typu czujników to wysyłanie sygnału dźwiękowego o częstotliwości 40-250 KHz, a następnie odebraniu tego sygnału po odbiciu od przeszkody, na podstawie czasu potrzebnego na powrót sygnału oblicza się odległość. Zaletą tego typu czujników jest odporność na warunki oświetleniowe, oraz duża precyzja wykrywania odległości do przeszkody. Poprzez działanie efektu Dopplera można też wykryć kierunek poruszających się przedmiotów. Pod uwagę należy wziąć 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 świetlne. Problemem jednak może być większy koszt tych czujników, zależność od warunków świetlnych. Czy typ materiału, problem z wykrywaniem przezroczystych przeszkód. W odróżnieniu do Ultradźwiękowych mogą jednak działać w próżni.**

**Czujniki dotykowe**

**Czujniki te reagują dopiero po bezpośrednim fizycznym kontakcie z przeszkodą.**

## Przykładowe 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 Stanfordaść prędkości fali dźwiękowej od temperatury

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 czestotliowośc sygnało co odpowiada również długości fali. W przypadku urządzeń USG w celu uzyskania precyzyjnego obrazu narządów stosuje się głowice pracujące na wysokich częstotliwościach. W zastosowaniu przeznaczynym do wykrywania przeszkód czestotliwosć 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.

Sensory 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.

## Serwomotor SG90

Serwomotory 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 serwomotoru znajduje się mały silnik prądu stałego (DC), który jest źródłem ruchu. Przekładnia 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.

Zasada działania. Serwomotor SG90 odbiera sygnał sterujący PWM o określonej częstotliwości (zwykle 50 Hz). Czas trwania impulsu w sygnale PWM określa żądany kąt ustawienia wału wyjściowego. Impuls 1 ms (1000 µs): Pozycja wału na 0° (minimalny kąt). Impuls 1,5 ms (1500 µs): Pozycja neutralna wału (90° w SG90). Impuls 2 ms (2000 µs): Pozycja wału na 180° (maksymalny kąt). Szerokość tego impulsu regulujemy zewnętrznie na przykład przy pomocy wyjścia układu FPGA. Układ sterujący w serwomechanizmie mierzy szerokość impulsu PWM i porównuje ją z sygnałem zwrotnym z potencjometru, który wskazuje bieżący kąt wału. Jeśli wał nie jest w żądanej pozycji, układ sterujący wysyła odpowiedni sygnał do silnika DC, aby obracać wał w kierunku żądanej pozycji. Silnik obraca wał wyjściowy, a potencjometr na bieżąco koryguje różnicę między kątem rzeczywistym i żądanym. Gdy kąt wału osiągnie żądaną wartość, silnik przestaje się obracać, a wał pozostaje w miejscu. Gdy wał osiągnie żądany kąt, serwo nadal otrzymuje sygnał PWM, co pozwala na utrzymanie wału w tej pozycji, nawet przy działających siłach zewnętrznych.

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.

# 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. Główne piny: VCC (zasilanie, 5V), GND, TRIG (sygnał inicjujący pomiar), ECHO (czas odbicia sygnału w mikrosekundach).

|  |
| --- |
| 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. |

Płytka FPGA Obsługuje Quartus i język VHDL np. Cyclone IV. 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.

## Hardware

**Altera Cyclone IV[12]**

Moduł deweloperski do programowania układu FPGA za pomocą Altera blaster. Moduł uruchomieniowy oparty o układ Altera EL4CE6E22C8. Bogate wyposażenie oraz liczba interfejsów pozwala 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. 3.1. ALTERA Cyclone IV EP4CE6 FPGA zestaw deweloperski Altera EP4CE NIOSII FPGA płyta i USB Blaster downloader kontroler podczerwieni. |

Płytka zawiera:

* główny układ nowej serii FPGA Cyclone IV firmy ALTERA EP4CE6E22C8N
* wbudowany układ szeregowy EPCS16N o dużej pojemności, obsługuje dwa tryby pobierania JTAG / AS
* wbudowany 64Mbit SDRAM, wsparcie dla rozwoju SOPC, NIOSII
* wbudowany aktywny kryształ 50MHz, stabilność systemu zegara głównego  
  użycie układu regulatora napięcia 1117-3,3 V, aby zapewnić napięcie wyjściowe 3,3 V.
* użycie układu regulatora napięcia 1117-1,2 V, aby zapewnić napięcie rdzenia FPGA
* użycie układu regulatora napięcia 1117-2,5 V, aby zapewnić napięcie wyjściowe PLL
* zastosowanie dużej liczby kondensatorów odsprzęgających, konstrukcja odsprzęgająca
* gniazdo zasilania 5 V DC i gniazdo zasilania interfejsu USB, dwa rodzaje trybu zasilania
* 1 przycisk resetowania, może być również używany jako przycisk wprowadzania danych przez użytkownika
* 1 przycisk samoblokujący wyłącznik zasilania
* 1 wskaźnik zasilania LED
* spośród wszystkich interfejsów IO, uniwersalny rozstaw 2,54mm, można łatwo rozszerzyć
* interfejs pobierania JTAG do pobierania pliku to .SOF, szybkość jest duża, zalecane jest zwykłe używanie tego interfejsu
* 1 interfejs pobierania AS do pobierania pliku to .POF, prędkość jest niska, należy użyć tego interfejsu podczas procesu utwardzania

|  |
| --- |
| Obraz zawierający tekst, zrzut ekranu, Inżynieria elektroniczna, obwód  Opis wygenerowany automatycznie |
| Rys. 3.2. Opis płyty Altera Cyclone IV. |

**Dane techniczne:**

* moduł uruchomienowy - deweloperski Altera CYCLONE IV
* układ oparty o Altera EL4CE6E22C8
* złącze do podłączenia LCD wsprawcie dla LCD 1602 lub 12864
* wyświetlacz 7- segmentowy
* regulacja kontrastu za pomocą potencjometru
* interfejs PS2
* interfejs RS232
* układ komunikacji szeregowej SP3232
* czujnik temperatury: LM75A
* buzzer
* przełączniki DIP Switch
* interfejs VGA
* odbiornik IR
* 4 przyciski
* EEPROM AT24C08
* 4 diody LED
* SDRAM 64MB
* 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 uruchomienowy z Altera CYCLONE IV
* przewód USB
* pilot IR
* interfejs zgodny z USB Blaster

**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:

distance = (high level time × velocity of sound / 2

gdzie:

test distance - odległość mierzona

high level time - czas trwania stanu wysokiego

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

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

distance [cm] = ( high level time [us] \* Sound\_Speed ) / 100 / 2

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

**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 serwo 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.

## Software

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

Wybór między VHDL (VHSIC Hardware Description Language) a Verilogiem 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 znajdziesz porównanie tych dwóch języków, które pomoże Ci dokonać wyboru.

VHDL jest bardziej formalny i przypomina języki ADA lub Pascal, co czyni go szczególnie odpowiednim dla projektów wymagających wysokiej niezawodności (np. w przemyśle lotniczym, medycznym). Wymaga większej ilości kodu i ma bardziej restrykcyjną typologię.

Verilog natomiast jest mniej formalny i bliższy językom C lub C++, co czyni go bardziej intuicyjnym dla początkujących programistów. Dzięki mniejszej ilości koniecznego kodu jest szybszy w pisaniu, ale mniej odporny na błędy wynikające z luźniejszej typologii.

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 zależy często od wymagań projektu lub preferencji zespołu.

## 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. Narzedzie syntezy logicznej dokonuje odwzrowania 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 implmentacji 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 konfiguracvji układu programolnego. 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.

# Implmentacja układu

W układzie wyróżniono następujące bloki które w dalszej części zostaną scharakteryzowane:

|  |
| --- |
|  |
| Rys. 4.1. schemat blokowy opisywanego fragmentu 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.

## Trigger generator

Najważniejszym elementem naszego programu, jest opisanie zasady działania wyzwalacza krótkiego sygnału dźwiękowego dla dalmierza ultradźwiękowego. Najpierw zastanówmy się nad jego zasadą działania. Zegar 50mhz zlicza kolejne impulsy, które należy policzyć aby określić dokładny czas wyzwolenia impulsu dźwiękowego, a następnie czasu podroży tego impulsu w przestrzeni do momentu wykrycia przez odbiornik czujnika. W tym celu należy stworzyć zegar który ma na celu liczyć impulsy i zapamiętywać je w buforze, nazwijmy go „counter” nastepnie przyrównamy go do dwóch zmiennych, „trig\_cnt” – jest to liczba impulsów określająca czas trwania stanu wysokiego sygnału TRIGGER. Oraz zmienna „cnt\_max” jest to zmienna o wartości całego okresu wyzwolenia impulsu oraz podróży impulsu w przestrzeni od wyzwalacza do odbiornika dla założonej maksymalnej odległości. Poniższy schemat blokowy określa działanie tego programu:

|  |
| --- |
| Obraz zawierający diagram, tekst, zrzut ekranu, Wielobarwność  Opis wygenerowany automatycznie |
| Rys. 4.2. Proces blokowy wyzwalania triggera. |

Dla ustalenia szerokości impulsu Trigger oraz szerokości jednego taktu szerokości impuslu w pliku Measurement control ustalamy dwa sygnały o odpowiednich wielkościach. Trig cnt jest wielkości 500 zapisany binarnie w słowie o długości 21 bitów dla wygody późniejszych przekształceń. 500 taktów zegara daje nam długość trwania impuslu 0,00001s czyli 10us. Kolejny sygnał pomocniczy to cnt\_max o wartości binarnej 1 250 000 taktów zegara. Co daje czas jednego takiego taktu 0,025s czyli 25ms, czyli ustalamy częstotliwość działania dlamierza ultradźwiękowego 40hz :

|  |  |
| --- | --- |
| 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 kod opisuje opisany wcześniej proces w programie quartus prime:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 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 pdrużuje 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 = (czas podróży sygnału × prędkość dźwięku / 2)*

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.

= 0,00034

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).

## Serwomotor i kontrola szerokości impulsu

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  15  16 | 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 jednago 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 ! 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. Proces 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  62  63  64  65  66  67  68  69  70 | 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 | |

# Weryfikacja i walidacja

Współczesny rozwój technologii wymaga tworzenia coraz bardziej zaawansowanych układów elektronicznych, które znajdują zastosowanie w różnych dziedzinach przemysłu. Implementacja takich układów stanowi kluczowy etap w procesie projektowania, wpływając bezpośrednio na ich funkcjonalność i niezawodność.

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.

## Testowanie w ModelSim

Testowanie programów w języku VHDL za pomocą narzędzia ModelSim oferuje szereg zalet, które czynią je jednym z najlepszych wyborów do symulacji układów cyfrowych. ModelSim umożliwia precyzyjną symulację zachowania układu na różnych poziomach abstrakcji, takich jak opis funkcjonalny, symulacja czasowa czy poziom bramek, pozwalając na dokładne odwzorowanie działania jeszcze przed implementacją na sprzęcie. Dzięki temu możliwe jest wczesne wykrywanie błędów w logice 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 | 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;  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;  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);  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. Syngał 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 11000100 czyli 196.

|  |
| --- |
|  |
| Rys. 5.1. przebieg symulacji |

## Testowanie na Cyclone IV

Najważniejszym spośród wszystkich jest testowanie naszego układu na rzeczywistym fizycznym układzie. W tym celu będzie nam potrzebna płytka testowa Cyclone IV czujnik odległości, oraz serwomechanizm. Serwomechanizm ze względu na trudność w sprawdzeniu czy działa odpowiednio w prosty sposób, na razie pozostanie odpięty. Do złożenia układu potrzebne będzie jeszcze płytka stykowa i kilka kabelków z zarobionymi odpowiednimi końcówkami. W programie włączamy planowanie pinów, i podłączamy piny wyświetlacza zegara i resetu według schematu podłączeń odpowiedniego dla naszej płytki, oraz wolne piny wykorzystujemy do podłączenia zewnętrznych urządzeń, jak na liście poniżej:

|  |
| --- |
|  |
| Rys. 5.2. Opis połączeń |

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 raporty, 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 podłączam oscyloskop by sprawdzić poprawność działania. Niestety prosty oscyloskop połączony z multimetrem którego używam, ma problemy z rozpoznaniem sygnałów o wysokiej częstotliwości używanych w do nadawania sygnału wyzwalania lub odbiory sygnału. Jak widać na poniższych obrazkach:

|  |
| --- |
|  |
| Rys. 5.4. Po lewej sygnał echo, po prawej sygnał Trig |

Możemy również sprawdzić sygnał PWM wysyłany do serwomechanizmu:

|  |
| --- |
|  |
| Rys. 5.5. PWM, z lewej w minimalnej pozycji z prawej maksymalnej |

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

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

Tabela z wynikami

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 |  |  |

Jak widać na powyższym wykresie, oraz w tabeli , dla wyznaczonych punktów pomiarowych, w przypadku książki oraz kubka pomiar odległości oddaje całkiem nieźle zadaną wartość, przy założeniu że błąd może wynikać z niedokładności narzędzi pomiarowych, lub nieodpowiedniego ich ustawienia można nawet zakładać że wynik jest idealny. Jednak z przeprowadzonych obliczeń wynika że błąd pomiarowy jest spowodowany niedokładnością przekształceń i dużymi uproszczeniami. Wykorzystując długie słowa bitowe, powyżej 30 bitów można dość do prawie idealnego obliczenia odległości. Konsekwencją tego była by duże ilość wykorzystywanych wbudowanych 9-cio bitowych mnożników.

Stosując następującą metodę pomiaru 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:

Błąd względny =()×100

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 temperatura, czy 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. Do

## 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. Imię Nazwisko, Imię Nazwisko. *Tytuł książki*. Wydawnictwo, Warszawa, 2017.
2. Imię Nazwisko, Imię Nazwisko. Tytuł artykułu w czasopiśmie. *Tytuł czasopisma*, 157(8):1092–1113, 2016.
3. Imię Nazwisko, Imię Nazwisko, Imię Nazwisko. Tytuł artykułu konferencyjnego. *Nazwa konferencji*, str. 5346–5349, 2006.
4. Autor, jeśli znany. https: [www.adres.strony](http://www.adres.strony) (dostęp:dzień.miesiąc.rok)
5. **HC-SR04 Ultrasonic Sensor Datasheet**  
   SparkFun Electronics. Retrieved from: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>
6. **Application of ultrasonic waves for the evaluation of mechanical properties**  
   Authors: A. Kumar, R. Pratap.  
   Published in: Procedia Engineering, Volume 144, 2016.  
   Retrieved from: <https://www.sciencedirect.com/science/article/pii/S2405896316326623>
7. **Absorption of Sound in Air**  
   Dr. Daniel A. Russell, Penn State University.  
   Retrieved from: <https://www.acs.psu.edu/drussell/Demos/Absorption/Absorption.html>
8. **Acoustics: Lecture 1 - Introduction**  
   Jay Kadis, Stanford University.  
   Retrieved from: <https://ccrma.stanford.edu/~jay/subpages/Lectures/Lecture1-Acoustics.pdf>
9. **Storing Data on an FPGA Target**  
   National Instruments.  
   Retrieved from: <https://www.ni.com/docs/en-US/bundle/labview-fpga-module/page/storing-data-on-an-fpga-target-fpga-module.html>
10. **Intel Quartus Prime Documentation**  
    Intel Corporation.  
    Retrieved from: <https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime/docs.html>
11. **VHDL vs Verilog: Which Language Should You Learn First?**  
    Nandland.  
    Retrieved from: <https://nandland.com/lesson-16-vhdl-vs-verilog-which-language-should-you-learn-first/>
12. **Altera Cyclone IV EP4CE6 FPGA Development Kit**  
    Gotronik.  
    Retrieved from: <https://www.gotronik.pl/altera-cyclone-iv-ep4ce6-fpga-zestaw-uruchomieniowy-p-8377.html>

Dodatki

# Spis skrótów i symboli

*DNA* kwas deoksyrybonukleinowy (ang. *deoxyribonucleic acid*)

*MVC* model – widok – kontroler (ang. *model–view–controller*)

*N* liczebność zbioru danych

µ stopień przynależności do zbioru

# Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego,

należy je przenieść do tego miejsca.

# Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

* źródła programu,
* dane testowe
* film pokazujący działanie opracowanego oprogramowania lub zaprojektowanego i wykonanego urządzenia,
* itp.

# Spis rysunków

4.1 Podpis rysunku jest pod rysunkiem 12

5.1 Pseudokod w listings 14

5.2 Pseudokod w minted 14

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

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

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

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

Rys. 1.5 Zakres prowadzonych pomiarów przez dalmierz ultradźwiękowy.

Rys. 3.1. Dalmierz ultradźwiękowy HC-SR04.

Rys. 3.2. Rysunek ideowy serwomotoru SG90 z osadzonym dalmierzem.

Rys. 3.1. ALTERA Cyclone IV EP4CE6 FPGA zestaw deweloperski Altera EP4CE NIOSII FPGA płyta i USB Blaster downloader kontroler podczerwieni.

Rys. 3.2. Opis płyty Altera Cyclone IV.

Rys. 4.1. schemat blokowy opisywanego fragmentu układu.

Rys. 4.2. Proces blokowy wyzwalania triggera.

Rys. 5.1. przebieg symulacji

Rys. 5.2. Opis połączeń

Rys. 5.3. Wynik kompilacji

Rys. 5.4. Po lewej sygnał echo, po prawej sygnał Trig

Rys. 5.5. PWM, z lewej w minimalnej pozycji z prawej maksymalnej

Rys 5.6. Pomiar odległości

# Spis tablic

6.1 Opis tabeli nad nią 16