

**PROJEKT INŻYNIERSKI**

Inteligentny rdzeń dalmierza ultradźwiękowego dla robota mobilnego

**Damian MACHURA**

**Nr albumu <wpisać właściwy>**

**Kierunek:** <wpisać właściwy>

**Specjalność:** <wpisać właściwą>

**PROWADZĄCY PRACĘ**

**<Tytuł lub stopień naukowy oraz imię i nazwisko>**

**KATEDRA <wpisać właściwą>**

**Wydział Automatyki, Elektroniki i Informatyki**

**OPIEKUN, PROMOTOR POMOCNICZY (jeśli został powołany)**

**<stopień naukowy oraz imię i nazwisko>**

**GLIWICE Rok**

**Tytuł pracy:**

Inteligentny rdzeń dalmierza ultradźwiękowego dla robota mobilnego

**Streszczenie:**

(Streszczenie pracy –odpowiednie pole w systemie APD powinno zawierać kopię tego streszczenia. Streszczenie, wraz ze słowami kluczowymi.)

**Słowa kluczowe:**

(2-5 słów (fraz) kluczowych, oddzielonych przecinkami)

**Thesis title:**

(Thesis title in English)

**Abstract:**

(Thesis abstract – to be copied into an appropriate field during electronic submission, in English.)

**Keywords:**

(2-5 keywords, separated with commas, in English.)

Spis treści

[1. Wstęp 1](#_Toc183264826)

[2. Analiza tematu 3](#_Toc183264827)

[2.1.1. Teoretyczna analiza wpływu temperatury i wysokości na prędkość fali ultradźwiękowej 3](#_Toc183264828)

[2.2. Działanie sensora ultradźwiękowego 5](#_Toc183264829)

[2.3. 1. Zasada działania 5](#_Toc183264830)

[Rozdział 3 [Wymagania i narzędzia] 8](#_Toc183264831)

[Rozdział 4 [Właściwy dla kierunku – np. Specyfikacja zewnętrzna] 10](#_Toc183264832)

[Rozdział 5 [Właściwy dla kierunku – np. Specyfikacja wewnętrzna] 12](#_Toc183264833)

[Rozdział 6 Weryfikacja i walidacja 14](#_Toc183264834)

[Rozdział 7 Podsumowanie i wnioski 16](#_Toc183264835)

[Bibliografia 18](#_Toc183264836)

[Spis skrótów i symboli 22](#_Toc183264837)

[Źródła 23](#_Toc183264838)

[Lista dodatkowych plików, uzupełniających tekst pracy 24](#_Toc183264839)

[Spis rysunków 25](#_Toc183264840)

[Spis tablic 26](#_Toc183264841)

# Wstęp

* wprowadzenie w problem/zagadnienie
* Osadzenie problemu w dziedzinie
* Cel pracy
* Zakres pracy
* Zwięzła charakterystyka rozdziałów
* Jednoznaczne określenie wkładu autora, w przypadku prac wieloosobowych – tabela z autorstwem poszczególnych elementów pracy

Pierwsza linia akapitu z wcięciem. Czcionka Times New Roman lub Callibri 12pt. Obustronne wyrównanie. Interlinia 1.3

Współczesne systemy robotyki mobilnej wymagają precyzyjnych i wydajnych rozwiązań do nawigacji w złożonym środowisku. Kluczowym elementem takich systemów jest dalmierz ultradźwiękowy, który umożliwia detekcję odległości do przeszkód oraz mapowanie otoczenia. Tradycyjne podejścia, takie jak wykorzystanie mikrokontrolerów Arduino, 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 opartego na układzie FPGA (ang. *Field Programmable Gate Array*), programowanym w języku opisu sprzętu VHDL. FPGA zapewnia znaczącą przewagę w zakresie równoległego przetwarzania danych oraz możliwości integracji zaawansowanych algorytmów bezpośrednio w architekturze sprzętowej, co przekłada się na większą szybkość działania i elastyczność. W porównaniu do standardowych rozwiązań mikrokontrolerowych, zastosowanie FPGA pozwala na lepsze wykorzystanie zasobów obliczeniowych oraz implementację rozwiązań zoptymalizowanych pod kątem energooszczędności.

Projekt ten obejmuje projektowanie i implementację 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. Realizacja systemu na FPGA umożliwia również implementację zaawansowanych algorytmów filtrowania sygnałów i detekcji przeszkód, co poprawia jakość danych pomiarowych.

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.

Poniższy schemat blokowy określa jak wygląda struktura programu w VHDL do programowania układu FPGA. Wykorzystany okłąd FPGA to EP4CE6E22 ktry jest na płytce CYCLONE IV:

Obraz zawierający tekst, zrzut ekranu, diagram, design

Opis wygenerowany automatycznie

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

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 (TOF, *Time of Flight*) obliczana jest odległość do przeszkody.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Grafika

Opis wygenerowany automatycznie

Podczas działania, serwo 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 lub do omijania przeszkód w robotyce mobilnej.

Obraz zawierający linia, zrzut ekranu, diagram, Równolegle

Opis wygenerowany automatycznie

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 serwa oraz opóźnienia wynikające z przetwarzania danych pomiarowych mogą powodować niedokładności.

Podczas ruchu serwa, szczególnie w skrajnych pozycjach (0 i 180 stopni), czas trwania sygnałów PWM może być niedostosowany do opóźnienia wynikającego z pomiaru odległości przez HC-SR04. Ponieważ HC-SR04 wymaga krótkiego odstępu czasu między kolejnymi pomiarami (minimalny czas oczekiwania po odbiorze sygnału to około 10 ms), serwo w trakcie tego opóźnienia może nie osiągnąć zaprogramowanego skrajnego kąta, co skutkuje niepełnym pokryciem pola widzenia.

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:

1. **Pełny zakres obrotu** — umożliwiający pokrycie całego pola 180 stopni.
2. **Poprawę precyzji mapowania** — pomiary są wykonywane dokładnie w przewidzianych pozycjach kątowych.
3. **Redukcję błędów czasowych** — eliminuje problem nieodpowiedniego ustawienia sensora podczas wysyłania sygnałów ultradźwiękowych.

Czujnik położenia może być realizowany na kilka sposobów, np. przez wykorzystanie potencjometru wbudowanego w SG90 lub zewnętrznego enkodera. Informacja o położeniu jest integrowana z logiką sterowania w układzie FPGA, co pozwala na dynamiczne dostosowanie sygnałów sterujących do aktualnego kąta serwa.

# Analiza tematu

* sformułowanie problemu
* osadzenie tematu w kontekście aktualnego stanu wiedzy (state of the art.) o poruszanym problemie
* studia literaturowe [2,3,4,1] – opis znanych rozwiązań (także opisanych naukowo, jeżeli problem jest poruszany w publikacjach naukowych), algorytmów

(1)

## 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.

#### **2. 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.

#### **Wpływ ciśnienia na wyższych wysokościach**

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.

## Działanie sensora ultradźwiękowego

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, wzbogacony o dodatkowe informacje z cytowanego tekstu:

### 1. **Zasada działania**

Sensor ultradźwiękowy mierzy odległość na podstawie czasu przelotu sygnału ultradźwiękowego (*time-of-flight principle*). 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 dźwięku w danym medium.

#### **2. Charakterystyka fal ultradźwiękowych**

* Częstotliwość fal waha się od 20 kHz do nawet 50 MHz, w zależności od zastosowania.
* Prędkość dźwięku w powietrzu wynosi około 340 m/s, ale zależy od takich czynników jak temperatura, wilgotność, ciśnienie atmosferyczne i gęstość medium.

#### **3. Rodzaje sensorów ultradźwiękowych**

* **Oddzielny nadajnik i odbiornik**: Są to układy, w których jeden element odpowiada za generowanie fal, a drugi za ich odbiór.
* **Pojedynczy przetwornik**: Często wykorzystywany w urządzeniach takich jak HC-SR04. Jeden przetwornik działa zarówno jako nadajnik, jak i odbiornik, co zwiększa kompaktowość systemu.

#### **4. Niezależność od materiałów i koloru powierzchni**

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.

#### **5. Ograniczenia i szczególne przypadki**

* **Materiały absorbujące wibracje**: Przedmioty wykonane z materiałów pochłaniających fale dźwiękowe mogą powodować zakłócenia w pomiarach.
* **Czynniki środowiskowe**: Parametry takie jak wilgotność czy temperatura wpływają na prędkość dźwięku, co może wymagać kalibracji czujnika.

#### **6. Zastosowania**

* **Robotyka**: Czujniki ultradźwiękowe są powszechnie używane w robotyce do wykrywania przeszkód i nawigacji bezkontaktowej.
* **Przemysł**: Wykorzystywane do kontroli jakości, wykrywania nieciągłości w materiałach, pomiaru grubości (np. w detekcji korozji) oraz pozycjonowania narzędzi i materiałów.
* **Monitorowanie poziomu cieczy**: Możliwość detekcji poziomu wody i innych cieczy w zbiornikach.

#### **7. Typy wyjść i zakres pomiarowy**

* 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.

#### **8. Podsumowanie technologiczne**

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.

# Rozdział 3 [Wymagania i narzędzia]

* wymagania funkcjonalne i niefunkcjonalne
* przypadki użycia (diagramy UML) – dla prac, w których mają zastosowanie
* opis narzędzi, metod eksperymentalnych, metod modelowania itp.
* metodyka pracy nad projektowaniem i implementacja – dla prac, w których ma to zastosowanie

# Rozdział 4 [Właściwy dla kierunku – np. Specyfikacja zewnętrzna]

Jeśli to specyfikacja zewnętrzna:

* wymagania sprzętowe i programowe
* sposób instalacji
* sposób aktywacji
* kategorie użytkowników
* sposób obsługi
* administracja systemu
* kwestie bezpieczeństwa
* przykład działania
* scenariusze z systemu (ilustrowane zrzutami z ekranu lub generowanymi dokumentami)

|  |
| --- |
|  |
| Rys.4.1. Podpis rysunku jest pod rysunkiem |

# Rozdział 5 [Właściwy dla kierunku – np. Specyfikacja wewnętrzna]

Jeśli to specyfikacja wewnętrzna:

* przedstawienie idei
* architektura systemu
* opis struktur danych (i organizacja baz danych)
* komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
* przegląd ważniejszych algorytmów (jeśli występują)
* szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
* diagramy UML

krótka wstawka kodu w linii tekstu jest możliwa, np. **descriptor**, a nawet **descriptor\_gaussian**. Dłuższe fragmenty lepiej jest umieszczać jako rysunek, np. kod na rysunku 5.1, a naprawdę długie fragmenty – w załączniku.

|  |  |
| --- | --- |
| 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  71  72  73  74  75  76  77  78  79 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **use** **IEEE.NUMERIC\_STD.ALL**;  *-- Definicja modułu głównego Range\_Sensor, który służy do pomiaru odległości*  **Entity** **Range\_Sensor** **is**  **Port** (  fpgaclk : **in** STD\_LOGIC; *-- Wejście zegara FPGA*  pulse : **in** STD\_LOGIC; *-- Wejście impulsu ECHO od czujnika ultradźwiękowego*  trigger\_out : **out** STD\_LOGIC; *-- Wyjście sygnału TRIGGER do czujnika*  reset: **in** STD\_LOGIC; *-- Wejście sygnału reset*  distance\_out: **out** Unsigned(8 **downto** 0) *-- Wyjście wynikowe z odległością*  );  **end** **Range\_Sensor**;  **architecture** **Behavioral** **of** **Range\_Sensor** **is**  *-- Definicja komponentu Distance\_calculator, który oblicza odległość*  **COMPONENT** **Distance\_calculator**  **port**(  clk: **in** std\_logic; *-- Wejście zegara*  Calculation\_Reset: **in** std\_logic; *-- Sygnał resetu obliczeń*  pulse :**in** std\_logic; *-- Wejście sygnału ECHO z czujnika*  Distance: **out** Unsigned(8 **downto** 0) *-- Wyjście z obliczoną odległością*  );  **END** **COMPONENT**;  *-- Definicja komponentu Trigger\_Generator, który generuje sygnał TRIGGER*  **COMPONENT** **Trigger\_Generator**  **PORT**(  clk : **IN** std\_logic; *-- Wejście zegara*  echo : **IN** std\_logic; *-- Wejście sygnału ECHO do detekcji impulsu*  trigger : **OUT** std\_logic *-- Wyjście sygnału TRIGGER*  );  **END** **COMPONENT**;  *-- Definicja sygnału do połączenia wewnętrznego*  **signal** trig\_out : std\_logic;  **begin**  *-- Instancja komponentu Trigger\_Generator*  *-- Ten komponent generuje sygnał TRIGGER do czujnika ultradźwiękowego*  trig\_generator: Trigger\_Generator **PORT** **MAP**(  clk => fpgaclk, *-- Podłączenie zegara FPGA*  echo => pulse, *-- Podłączenie sygnału ECHO z czujnika*  trigger => trig\_out *-- Wyjście sygnału TRIGGER do czujnika*  );  *-- Instancja komponentu Distance\_calculator*  *-- Ten komponent mierzy szerokość impulsu ECHO, aby obliczyć odległość*  Pulse\_width: Distance\_calculator **PORT** **MAP**(  clk => fpgaclk, *-- Podłączenie zegara FPGA*  calculation\_reset => trig\_out, *-- Sygnał resetu oparty na sygnale TRIGGER*  pulse => pulse, *-- Podłączenie sygnału ECHO z czujnika*  Distance => distance\_out *-- Wyjście z obliczoną odległością*  );  *-- Przekazanie sygnału TRIGGER na wyjście głównego modułu*  trigger\_out <= trig\_out;  **end** **Behavioral**; |

**Opis działania:**

1. **Moduł Trigger\_Generator** generuje sygnał TRIGGER, który jest wysyłany do czujnika ultradźwiękowego, aby rozpocząć pomiar. Sygnał ten jest synchronizowany z sygnałem fpgaclk.
2. **Moduł Distance\_calculator** mierzy czas trwania impulsu ECHO zwróconego przez czujnik i na tej podstawie oblicza odległość. Wynik jest podany na wyjście distance\_out.
3. **Główna architektura** (Behavioral) zarządza połączeniami między komponentami.

Trigger generator:

|  |  |
| --- | --- |
| 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 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **use** **ieee.numeric\_std.all**;  *-- Definicja modułu Trigger\_Generator*  **entity** **Trigger\_Generator** **is**  **Port** (  clk : **in** std\_logic; *-- Wejście zegara*  echo : **in** std\_logic; *-- Wejście sygnału ECHO*  trigger : **out** std\_logic *-- Wyjście sygnału TRIGGER*  );  **end** **Trigger\_Generator**;  **architecture** **Behavioral** **of** **Trigger\_Generator** **is**  *-- Definicja komponentu Counter*  **component** **Counter** **is**  **generic**(n: POSITIVE := 10); *-- Parametr do ustawienia zakresu licznika*  **Port** (  clk : **in** std\_logic; *-- Wejście zegara*  enable : **in** STD\_LOGIC; *-- Wejście sygnału umożliwiającego pracę licznika*  reset : **in** std\_logic; *-- Wejście sygnału resetu licznika*  count : **out** integer **range** 0 **to** n *-- Wyjście z aktualną wartością licznika*  );  **end** **component**;  *-- Sygnały wewnętrzne*  **signal** resetCounter : std\_logic; *-- Sygnał resetu dla licznika*  **signal** outputCounter : integer; *-- Wartość licznika*  **begin**  *-- Instancja komponentu Counter*  *-- Licznik odmierza czas 250 ms oraz dodatkowe 100 μs (250.1 ms)*  trigg : Counter  **generic** **map**(2\*\*24) *-- Zakres licznika ustawiony na 2^24*  **port** **map**(clk, '1', resetCounter, outputCounter); *-- Podłączenia sygnałów*  **process**(clk)  *-- Definicje stałych czasowych*  **constant** ms250 : integer := 12500000; *-- 250 ms przy zegarze 50 MHz*  **constant** ms250And100us : integer := 12505000; *-- 250.1 ms przy zegarze 50 MHz*  **begin**  *-- Proces generujący sygnał TRIGGER w określonym przedziale czasowym*  **if**(outputCounter > ms250 **and** outputCounter < ms250And100us) **then**  trigger <= '1'; *-- Aktywacja sygnału TRIGGER przez 100 μs*  **else**  trigger <= '0'; *-- Sygnał TRIGGER w stanie niskim poza wyznaczonym czasem*  **end** **if**;  *-- Proces resetowania licznika*  **if**(outputCounter = ms250And100us **or** outputCounter > ms250And100us **or** outputCounter < 0) **then**  resetCounter <= '0'; *-- Zatrzymanie licznika, gdy przekroczy ustawiony czas*  **else**  resetCounter <= '1'; *-- Włączony licznik, dopóki nie osiągnie limitu czasowego*  **end** **if**;  **end** **process**;  **end** **Behavioral**; |

1. **Opis działania:**
2. **Główne komponenty i sygnały:**
   * Moduł Counter działa jako licznik impulsów, umożliwiający pomiar odcinka czasowego.
   * Sygnały resetCounter oraz outputCounter kontrolują stan licznika.
3. **Proces Trigger Generation:**
   * **Generowanie sygnału TRIGGER**: W momencie, gdy licznik outputCounter osiągnie wartość pomiędzy ms250 a ms250And100us, sygnał TRIGGER ustawia się na '1' przez 100 µs.
   * **Resetowanie licznika**: Gdy outputCounter osiągnie wartość ms250And100us lub przekroczy ją, resetCounter ustawia się na '0', co zatrzymuje licznik.

Opis działania Distance Calculation

|  |  |
| --- | --- |
| 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 | **library** **ieee**;  **use** **ieee.std\_logic\_1164.all**;  **use** **ieee.numeric\_std.all**;  **use** **ieee.std\_logic\_unsigned.all**;  *-- Definicja modułu Distance\_calculator*  **entity** **Distance\_calculator** **is**  **port**(  clk : **in** std\_logic; *-- Wejście zegara*  Calculation\_Reset : **in** std\_logic; *-- Sygnał resetu obliczeń*  pulse : **in** std\_logic; *-- Wejście impulsu z czujnika*  Distance : **out** Unsigned(8 **downto** 0) *-- Wyjście z wartością odległości*  );  **end** **Distance\_calculator**;  **architecture** **Behavioral** **of** **Distance\_calculator** **is**  *-- Definicja komponentu Counter*  **component** **Counter** **is**  **generic**(  n : POSITIVE := 10 *-- Parametr dla zakresu licznika*  );  **port**(  clk : **in** std\_logic; *-- Wejście zegara*  enable : **in** std\_logic; *-- Sygnał włączający pracę licznika*  reset : **in** std\_logic; *-- Sygnał resetu licznika*  count : **out** integer **range** 0 **to** n *-- Wyjście z aktualną wartością licznika*  );  **end** **component**;  *-- Sygnał wewnętrzny do przechowywania szerokości impulsu*  **signal** Pulse\_width : integer **range** 0 **to** 2\*\*22;  **begin**  *-- Instancja komponentu Counter*  *-- Licznik mierzy czas trwania impulsu 'pulse' otrzymanego z czujnika*  Counter\_pulse : Counter  **generic** **map**(2\*\*22) *-- Zakres licznika ustawiony na 2^22*  **port** **map**(clk, pulse, **not** Calculation\_Reset, Pulse\_width);  *-- Proces obliczający odległość*  Distance\_calculator : **process**(pulse)    *-- Zmienne pomocnicze do obliczeń*  **variable** Result : integer **range** 0 **to** 1000;  **variable** Multiplier : integer;  **begin**  *-- Gdy zakończy się impuls 'pulse', wykonywane są obliczenia*  **if** (pulse = '0') **then**  *-- Obliczanie odległości w cm na podstawie czasu trwania impulsu*  Result := (Pulse\_width / 50) / 58;    *-- Sprawdzenie zakresu odległości i ograniczenie maksymalnej wartości*  **if** (Result > 458) **then**  Distance <= "111111111"; *-- Przekroczenie zakresu*  **else**  Distance <= to\_unsigned(Result, 9); *-- Wynik konwersji do 9-bitowej liczby unsigned*  **end** **if**;  **end** **if**;  **end** **process**;  **end** **Behavioral**; |

1. **Opis działania:**
2. **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.
3. **Proces Distance\_calculator**:
   * Gdy impuls pulse zakończy się (czyli pulse = '0'), następuje obliczenie odległości na podstawie zmierzonego czasu.
   * **Obliczenia odległości**: Wartość Pulse\_width jest dzielona przez 50 (dostosowanie do częstotliwości zegara 50 MHz), a następnie przez 58 (stała przeliczeniowa dla przeliczenia impulsu na odległość w centymetrach).
   * **Ograniczenie zakresu**: Jeśli wynik przekracza wartość 458 cm, odległość jest przypisana do maksymalnej wartości 111111111 (9-bitowe unsigned).

Opis top design Serva:

|  |  |
| --- | --- |
| 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  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  10 | **library** **ieee**;  **use** **ieee.std\_logic\_1164.all**;  **use** **ieee.numeric\_std.all**;  *-- Definicja komponentu Servo*  **entity** **Servo** **is**  **port** (  clk : **in** std\_logic; *-- Wejście zegara*  reset : **in** std\_logic; *-- Wejście sygnału resetu*  servo\_pwm : **out** std\_logic *-- Wyjście sygnału PWM*  );  **end** **Servo**;  **architecture** **behavioral** **of** **Servo** **is**  **COMPONENT** **pwm** **is**  **generic** (  clk\_hz : real *-- Częstotliwość zegara wejściowego (w Hz)*  );  **port** (  clk : **in** std\_logic; *-- Wejście zegara*  rst : **in** std\_logic; *-- Wejście resetu*  position : **in** integer **range** 0 **to** 256; *-- Pozycja, od 0 do maksymalnej liczby kroków*  pwm : **out** std\_logic *-- Wyjście sygnału PWM*  );  **end** **COMPONENT**;  **COMPONENT** **Counter** **is**  **generic**(n: POSITIVE := 10);  **Port** (  clk : **in** std\_logic; *-- Clock input*  enable : **in** STD\_LOGIC;  reset : **in** std\_logic; *-- Reset input*  count : **out** integer **range** 0 **to** n  );  **end** **COMPONENT**;  **COMPONENT** **sine\_rom** **is**  **port** (  clk : **in** std\_logic; *-- Wejście zegara*  addr : **in** unsigned(7 **downto** 0); *-- Adres odczytu w pamięci ROM*  data : **out** unsigned(7 **downto** 0) *-- Wartość sinusoidy wyjściowej*  );  **end** **COMPONENT**;  **constant** clk\_hz : real := 50.0e6; *-- Częstotliwość zegara (50 MHz)*  **signal** cnt : unsigned(26 **downto** 0); *-- Sygnalizuje obecny stan licznika*  **signal** count : integer; *-- Wartość licznika do użycia*  **signal** rst : std\_logic;  **signal** position : integer **range** 0 **to** 255; *-- Pozycja w zakresie od 0 do 255*  **signal** rom\_addr : unsigned(7 **downto** 0); *-- Adres dla pamięci ROM*  **signal** rom\_data : unsigned(7 **downto** 0); *-- Dane z pamięci ROM*  **begin**  *-- Przypisanie pozycji na podstawie wartości `rom\_data`*  position <= to\_integer(rom\_data);  cnt <= to\_unsigned(count, 27); *-- Konwersja liczby całkowitej na `unsigned`*  rom\_addr <= cnt(cnt'left **downto** cnt'left - 7); *-- Wyliczanie adresu do ROM*  *-- Moduł PWM odpowiedzialny za generowanie sygnału*  SERVO : pwm  **generic** **map** (  clk\_hz => clk\_hz  )  **port** **map** (  clk => clk,  rst => **not** reset, *-- Odwrócony sygnał resetu*  position => position,  pwm => servo\_pwm  );  *-- Moduł licznika używany do inkrementacji `count` z określonym zakresem*  COUNTING : Counter  **generic** **map** (  2\*\*27 *-- Ustawia maksymalną wartość licznika*  )  **port** **map** (  clk => clk,  reset => reset,  enable => '1', *-- Licznik zawsze włączony*  count => count *-- Wartość licznika wyjściowego*  );  *-- Pamięć ROM przechowująca wartości sinusoidy, do generowania danych*  SINE : sine\_rom  **port** **map** (  clk => clk,  addr => rom\_addr,  data => rom\_data  );  **end** **behavioral**; |

1. **Opis działania:**
2. **Parametry wejściowe**:
   * clk to główny sygnał zegara, reset służy do resetowania modułu, a pwm jest wyjściowym sygnałem PWM generowanym w oparciu o ustawienia pozycji serwomechanizmu.
3. **Definicje parametrów**:
   * clk\_hz, pulse\_hz, min\_pulse\_us, max\_pulse\_us określają parametry zegara i PWM.
   * step\_bits i step\_count definiują liczbę kroków i adresów w pamięci, co jest pomocne przy ustawianiu pozycji.
4. **Moduły**:
   * **SERVO**: odpowiada za generowanie sygnału PWM dla serwomechanizmu.
   * **COUNTER**: zlicza cykle zegara, generując adresy pamięci ROM i pozwalając na pozycjonowanie serwomechanizmu.
   * **SINE\_ROM**: Pamięć ROM, z której pobierane są dane sinusoidalne, w oparciu o adresy wygenerowane przez licznik.
5. **Procesy pomocnicze**:
   * position jest wyliczany na podstawie rom\_data.
   * Adres rom\_addr jest wyprowadzany z cnt, co pomaga przetworzyć dane dla ustawienia pozycji serwomechanizmu.

Dzięki takiej strukturze kodu można generować precyzyjny sygnał PWM dla serwomechanizmu z sygnałem sinusoidalnym, a także płynnie regulować pozycję.

|  |  |
| --- | --- |
| 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  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115 | **library** **ieee**;  **use** **ieee.std\_logic\_1164.all**;  **use** **ieee.numeric\_std.all**;  **use** **ieee.math\_real.round**;  *-- Definicja modułu PWM*  **entity** **pwm** **is**  **generic** (  clk\_hz : real; *-- Częstotliwość zegara wejściowego (w Hz)*  pulse\_hz : real; *-- Częstotliwość generowanego sygnału PWM*  min\_pulse\_us : real; *-- Minimalna szerokość impulsu w mikrosekundach*  max\_pulse\_us : real; *-- Maksymalna szerokość impulsu w mikrosekundach*  step\_count : positive *-- Liczba kroków od pozycji minimalnej do maksymalnej*  );  **port** (  clk : **in** std\_logic; *-- Wejście zegara*  rst : **in** std\_logic; *-- Wejście resetu*  position : **in** integer **range** 0 **to** step\_count - 1; *-- Pozycja, od 0 do maksymalnej liczby kroków*  pwm : **out** std\_logic *-- Wyjście sygnału PWM*  );  **end** **pwm**;  **architecture** **rtl** **of** **pwm** **is**  *-- Funkcja obliczająca liczbę cykli zegara dla określonego czasu w mikrosekundach*  **function** cycles\_per\_us (us\_count : real) **return** integer **is**  **begin**  **return** integer(round(clk\_hz / 1.0e6 \* us\_count));  **end** **function**;  *-- Stałe dla minimalnej i maksymalnej liczby cykli zegara w zależności od szerokości impulsu*  **constant** min\_count : integer := cycles\_per\_us(min\_pulse\_us); *-- Liczba cykli dla minimalnego impulsu*  **constant** max\_count : integer := cycles\_per\_us(max\_pulse\_us); *-- Liczba cykli dla maksymalnego impulsu*  **constant** min\_max\_range\_us : real := max\_pulse\_us - min\_pulse\_us; *-- Zakres czasowy impulsu*  **constant** step\_us : real := min\_max\_range\_us / real(step\_count - 1); *-- Krok w mikrosekundach między pozycjami*  **constant** cycles\_per\_step : positive := cycles\_per\_us(step\_us); *-- Liczba cykli zegara na krok pozycji*  *-- Maksymalna wartość licznika dla określonej częstotliwości PWM*  **constant** counter\_max : integer := integer(round(clk\_hz / pulse\_hz)) - 1;  **signal** counter : integer **range** 0 **to** counter\_max; *-- Licznik sygnału PWM*  **signal** duty\_cycle : integer **range** 0 **to** max\_count; *-- Liczba cykli zegara odpowiadająca wybranej szerokości impulsu*  **begin**  *-- Proces licznika PWM*  COUNTER\_PROC : **process**(clk)  **begin**  **if** rising\_edge(clk) **then**  **if** rst = '1' **then**  counter <= 0; *-- Reset licznika*  **else**  **if** counter < counter\_max **then**  counter <= counter + 1; *-- Inkrementacja licznika*  **else**  counter <= 0; *-- Reset licznika, gdy osiągnie maksimum*  **end** **if**;  **end** **if**;  **end** **if**;  **end** **process**;  *-- Proces generujący sygnał PWM*  PWM\_PROC : **process**(clk)  **begin**  **if** rising\_edge(clk) **then**  **if** rst = '1' **then**  pwm <= '0'; *-- Reset sygnału PWM*  **else**  pwm <= '0'; *-- Domyślnie ustawienie PWM na niski stan*  **if** counter < duty\_cycle **then**  pwm <= '1'; *-- Sygnał wysoki, jeśli licznik jest mniejszy od wartości duty\_cycle*  **end** **if**;  **end** **if**;  **end** **if**;  **end** **process**;  *-- Proces ustawiający wartość szerokości impulsu na podstawie pozycji*  DUTY\_CYCLE\_PROC : **process**(clk)  **begin**  **if** rising\_edge(clk) **then**  **if** rst = '1' **then**  duty\_cycle <= min\_count; *-- Ustawienie szerokości impulsu na minimalną pozycję*  **else**  duty\_cycle <= position \* cycles\_per\_step + min\_count; *-- Obliczenie szerokości impulsu na podstawie pozycji*  **end** **if**;  **end** **if**;  **end** **process**;  **end** **architecture**; |

1. **Opis działania:**
2. **Funkcja**cycles\_per\_us: Oblicza liczbę cykli zegara potrzebnych do uzyskania określonego czasu trwania impulsu w mikrosekundach.
3. **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 w pulse\_hz.
4. **Proces**COUNTER\_PROC: Zlicza cykle zegara i resetuje licznik po osiągnięciu counter\_max.
5. **Proces**PWM\_PROC: Ustawia stan sygnału PWM na podstawie porównania licznika i wartości duty\_cycle.
6. **Proces**DUTY\_CYCLE\_PROC: Oblicza szerokość impulsu PWM dla danej pozycji.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **package** polsl.iinf.lab;  **import** java.util.Random;  **public** **class** Main {  **public** **static** **void** main(String[] args) {  Random r = **new** Random();  // drawing a number from the range 1..10  **int** a = r.nextInt(10 + 1);  System.*out*.println(a);  // drawing a number from the range -5..15  System.*out*.println(r.nextInt(21) - 5);  }  } |

Rysunek 5.1: Pseudokod

# Rozdział 6 Weryfikacja i walidacja

* sposób testowania w ramach pracy (np. odniesienie do modelu V)
* organizacja eksperymentów
* przypadki testowe, zakres testowania (pełny/niepełny)
* wykryte i usunięte błędy
* opcjonalnie wyniki badań eksperymentalnych

|  |  |
| --- | --- |
| Tabela 4.2. Nagłówek tabeli jest nad tabelą. | |
| Poziom 1 | 24 pt |
| Poziom 2 | 20 pt |
| Poziom 3 | 16 pt |
|  | |

# Rozdział 7 Podsumowanie i wnioski

* uzyskane wyniki w świecie postawionych celów i zdefiniowanych wyżej wymagań
* kierunki ewentualnych danych prac (rozbudowa funkcjonalna …)
* problemy napotkane w trakcie pracy

# 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)

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

# Spis tablic

6.1 Opis tabeli nad nią 16