



Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie



Wydział
Informatyki

Patryk Hachuła

nr albumu: 23433

kierunek studiów: Informatyka

specjalność: Sieci komputerowe

forma studiów: stacjonarne

**OPROGRAMOWANIE STERUJĄCE URZĄDZENIEM TYPU TRACKBALL POPRZEZ
INTERFEJS USB**

SOFTWARE FOR TRACKBALL DEVICE CONNECTED THROUGH USB INTERFACE

praca dyplomowa licencjacka

napisana pod kierunkiem:

dra inż. Krzysztofa Małeckiego

Katedra Systemów Multimedialnych

Data wydania tematu pracy: 10.12.2014

Data złożenia pracy: 20.04.2016

Szczecin, 2016

Streszczenie

Słowa kluczowe:

Abstract

Keywords:

OŚWIADCZENIE

Oświadczam, że przekładaną pracę dyplomową napisałem samodzielnie. Oznacza to, że przy pisaniu pracy poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania pracy lub jej części innym osobom oraz nie przypisałem sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego.

Spis treści

1 Wprowadzenie	6
2 Czym jest trackball?	7
2.1 Istniejące rozwiązania	10
3 Zarys historyczny	14
4 Projekt i wykonanie urządzenia	20
4.1 Proces wytwarzania podstawy sprzętowej	20
4.1.1 Opis rodziny wykorzystanego mikrokontrolera	21
4.2 Wymagania funkcjonalne	22
4.3 Elementy składowe urządzenia	34
4.4 Ułożenie podzespołów płytki drukowanej	35
5 Implementacja oprogramowania sterującego	36
5.1 Interfejs oraz sposób komunikacji	37
5.2 Wybór oraz opis działania biblioteki obsługującej interfejs USB	38
5.3 Analiza oprogramowania sterującego	41
6 Testowanie	52
6.1 Testy funkcjonalne	53
6.2 Testy zgodności	57
6.3 Testy wydajności	58
7 Zakończenie	60
Bibliografia	62
Spis rysunków	64
Spis tabel	64
A Pełny kod źródłowy	66

B Opis zawartości płyty CD	74
--------------------------------------	----

Rozdział 1

Wprowadzenie

Wraz z rozwojem urządzeń komputerowych ludzie mają dostęp do coraz większej liczby funkcji, w tym interfejsu graficznego. Użycie ich wiąże się z potrzebą posiadania sprzętu (który umożliwia obsługę nowoczesnych rozwiązań technologicznych) takiego jak urządzenia wskazujące. Jednym z takich urządzeń dostępnym na rynku oraz spełniającym te wymogi jest manipulator kulkowy, czyli potocznie trackball. Alternatywnym rozwiązaniem pozostaje nadal mysz komputerowa, jednak manipulator kulkowy o wiele lepiej sprawdza się w niecodziennych warunkach (Na przykład na statku podczas sztormu, w studiu nagraniowym jako część korektora akustycznego, czy jako manipulator kurSORA radaru).

Celem pracy jest skonstruowanie urządzenia typu trackball oraz implementacja oprogramowania sterującego tym urządzeniem przy wykorzystaniu interfejsu USB.

Teraz nastąpi krótkie omówienie zawartości niniejszej pracy. W kolejnym rozdziale zostało opisane urządzenie, jego budowa, zasada działania oraz przykłady zastosowania. Przedstawiono w nim także rozwiązania sprzętowe, aktualnie występujące na rynku urządzeń wskazujących. W kolejnym z rozdziałów opisano historię powstawania manipulatora kulkowego. Rozdział czwarty przedstawia proces projektowania oraz wytwarzania trackballa. Zostały w nim opisane szczegóły techniczne urządzenia oraz wymagania funkcjonalne. Rozdział piąty opisuje proces implementacji oprogramowania sterującego. Jest w nim uzasadniony wybór konkretnej biblioteki wykorzystanej w rozwiązaniu. W kolejnym rozdziale ukazany został proces testowania, jego techniki oraz wyniki testów. Rozdziałem kończącym pracę jest podsumowanie oraz wnioski, do których autor doszedł po wykonaniu projektu.

Rozdział 2

Czym jest trackball?

W rozdziale opisane zostanie urządzenie trackball. Przybliżona zostanie jego budowa oraz zastosowania.

Trackball jest to urządzenie wskazujące, wykorzystywane do obsługi specjalistycznych komputerów. Składa się z ruchomej kuli umieszczonej na statywie lub w obudowie. Kula znajduje się w jego górnej części w taki sposób, by użytkownik był w stanie w dowolny sposób ją obracać. Istnieją różne rozmiary kul, a co za tym idzie, różne typy trackballi. Różnice objawiają się w zależności od zastosowania. W przeciwieństwie do myszy komputerowych, gdzie średnica kuli wynosi około dwóch centymetrów, w trackballach rozmiar ten jest znacznie większy, ze względu na wysoką dokładność odczytu czujników. Dokładniejszy odczyt zmiany pozycji zapewniony jest dzięki temu, że elementy sensora mają więcej przestrzeni. Są one potrzebne do odczytu zmiany pozycji kuli. W większości przypadków dla każdej osi potrzebny jest jeden zestaw czujników, jednak istnieją precyzyjniejsze rozwiązania.

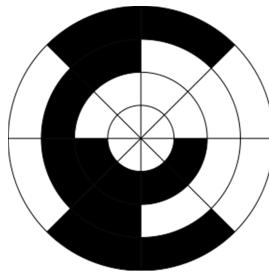


Rysunek 2.1: Wygląd przykładowego trackballa

Pierwsze trackballe używały dysków jako czujników. Poruszane one były poprzez ruch kuli w danej osi. Impulsy wytwarzane były poprzez przepuszczanie wiązki podczerwonej przez szczeliny dysków. Na podstawie tego mechanizmu powstały pierwsze myszki komputerowe wykorzystujące kule do wykrywania ruchu.

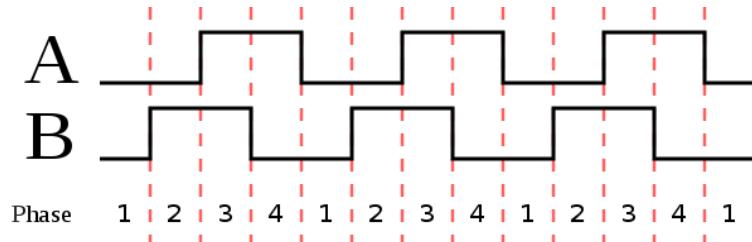
Istnieją dwa typy trackballi. Mechaniczno-optyczne i optyczne. Mechaniczno-optyczny działa na zasadzie poruszania rolek przez kulę, które przesłaniają fotodiody. Optyczny zaś posiada kulę w określonej fakturze, która w zależności od obrotu odbija światło w różnych kierunkach. Dzięki temu możemy stwierdzić, w którą stronę kula została obrócona. Opracowane rozwiązanie wykorzystuje technikę pierwszego typu.

Działanie urządzenia uzależnione jest od kliku czynników. Pierwszym z nich jest użycie sensorów umieszczonej na każdej z osi, po której obraca się kula. Czujniki wykrywają zmianę obrotu kuli w danej płaszczyźnie na podstawie stanu dwóch fotodiod. Stany te przy odczytywaniu tworzą kod Greya.



Rysunek 2.2: Schemat kodowania kodu Greya dla 3 bitów przedstawiony na kole

Poprzez jego odczyt możemy określić czy, i w jakim kierunku obróciła się kula. W zależności od obrotu kuli, przełączane są kolejno dwa bity odpowiedzialne za osią.



Rysunek 2.3: Diagram przebiegu wartości stanów dwóch sygnałów dla kolejnych wartości

Kolejnym czynnikiem jest przetwarzanie wyżej wymienionych sygnałów na dane zrozumiałe dla komputera oraz ich przesłanie za pomocą struktury.

Code for Clockwise Rotation		
Phase	A	B
1	1	0
2	1	1
3	0	1
4	0	0

Code for Counter-Clockwise Rotation		
Phase	A	B
1	1	0
2	1	1
3	0	1
4	0	0

Rysunek 2.4: Tabela przebiegu kolejnych faz przy obrocie koła w daną stronę w postaci danych dostarczanych do mikrokontrolera

Porównując trackball do powszechnie stosowanych myszek komputerowych można zauważyć znaczącą różnicę w budowie oraz dokładności poruszania kurSORA. Mysz komputerową poruszamy poprzez ruchy nadgarstkami, natomiast używając trackballa nie musimy poruszać całym nadgarstkiem, używamy wyłącznie palców. Jest to dość wygodne i nienadwyrężające nadgarstka rozwiązanie.

Jednym z popularnych zastosowań trackballa jest użycie go do tworzenia zaawansowanych projektów w rodzinie programów CAD, gdyż urządzenie ma bardzo wysoką precyzję. Używa się ich także na biurkach, na których jest niedostatecznie dużo miejsca do użycia myszki. Do innych zastosowań zaliczyć można również aparaturę medyczną np. urządzenie wykonujące badania USG, miksery akustyczne oraz pulpity sterownicze na statkach i samolotach.

2.1 Istniejące rozwiązania

Obecnie na rynku dostać można urządzenia typu trackball do wykorzystania domowego. Działają one jako mysz komputerowa. Liderami na rynku takich urządzeń są: Logitech, Kensington, Microsoft oraz Adesso. Urządzenia te cechuje:

- mały rozmiar
- kula umieszczona na urządzeniu
- wyższa cena w stosunku do zwykłych myszy komputerowych
- ergonomiczny kształt
- wygoda użytkowania

Tabela 2.1: Porównanie różnych modeli trackballi

Model	Bezprzewodowy	Liczba przycisków	Przewijanie	Oburęczny	Rozmiar kuli
Logitech TrackMan Marble	Nie	4	Przy pomocy za-programowanych przycisków	Tak	Średni
Logitech M570 Wireless Trackball	Tak	5	Pokrętło	Nie	Mały
Logitech Cordless Optical TrackMan	Tak	8	Pokrętło	Nie	Średni
Logitech TrackMan Wheel Optical	Nie	2	Pokrętło	Nie	Średni
Kensington Slimblade	Nie	4	Przewijanie dotykowe	Tak	Duży



Rysunek 2.5: Trackball Logitech TrackMan Marble

Logitech TrackMan Marble jest bardzo popularnym urządzeniem wskazującym. Można ją kupić bardzo tanio (jak na urządzenie trackball). Dostępna jest w sklepach za cenę około 80 złotych. Po bokach posiada dwa duże przyciski, które można oprogramować lub używać jako prawy i lewy przycisk myszy, jednak implementacja własnych rozwiązań jest niemożliwa z poziomu kodu źródłowego. Posiada on również dwa małe przyciski w górnej części obudowy. Trackball ten nie posiada funkcji przewijania za pomocą kuli. Ze względu na jego kształt urządzenie może być wykorzystywane zarówno przez osoby prawo- i lewo-ręczne. Ruchy kuli odbierane są przez czujnik optyczny. Jedyną wadą tego urządzenia jest brak funkcji przewijania.



Rysunek 2.6: Trackball Kensington Slimblade

Kensington Slimblade wykonany jest z materiałów wysokiej jakości. Urządzenie jest dość duże jak na trackball, jednak ma to swoje uzasadnienie. Jest tak, ponieważ wymaga tego budowa urządzenia. Otwór, w którym osadzona jest kula przechodzi na wylot przez urządzenie, co pozwala na łatwe czyszczenie, oraz umiejscowienie czujników bezpośrednio wokół kuli. Urządzenie posiada cztery przyciski, które mogą zostać dostosowywane do naszych potrzeb. Poprzez obracanie kuli możliwe jest przewijanie, którego prędkość również można zmieniać. Wadą może być wysoka cena. Około 360 złotych.



Rysunek 2.7: Trackball Logitech M570

Logitech M570 Wireless Trackball łączy w sobie funkcjonalność myszy komputerowej z funkcjonalnością manipulatora kulkowego. Jest to widoczne na zdjęciu powyżej. Budowa urządzenia sprawia, że jest ono bardzo wygodne w użytkowaniu jako trackball. Kula umieszczona jest po lewej stronie pod kciukiem. Urządzenie posiada również osobną rolkę do przewijania. Wadą jest mały rozmiar kuli, co skutkuje mniejszą dokładnością poruszania kursorem. Model przystosowany jest wyłącznie dla osób praworęcznych. Wyróżnia się pośród modeli dostępnych na rynku.

Rozdział 3

Zarys historyczny

Niniejszy rozdział opisuje początek historii trackballi.

W 1941 roku zaraz, niedługo po II wojnie światowej, w zastępach wojskowych wynalazców, znajdował się człowiek imieniem Ralph, którego do dziś uważa się za osobę, która zbudowała pierwszy trackball na świecie, jednak początkowo był on trzymany w tajemnicy ze względu na militarny charakter wynalazku. Pracował on nad projektem obliczania przyszłej trajektorii samolotów, do którego używały joysticków. Potrzebował on jednak dokładniejszego rozwiązania. Wtedy wpadł na pomysł pobierania pozycji na podstawie obracającej się kuli [1]. Pierwsze urządzenie wykorzystujące toczącą się sferę w połączeniu z analogowym komputerem pozwoliło na wskazywanie przyszłej pozycji obiektów latających. Urządzenie zostało opatentowane w 1947 roku.

Kolejną osobą która odegrała ważną rolę w rozwoju manipulatorów kulkowych był George M. Laman. Zgłosił on patent opisujący aparat sterujący pozycją oparty na elektromechanicznym oraz elektrooptycznym mechanizmie przesyłania informacji potrzebnych do kontroli pozycjonowania. Opisał on swój wynalazek w każdym szczególe, o czym mogą świadczyć cytaty zaczerpnięte z dokumentacji patentowej [2].

Na rysunku 3.1 znajduje się fragment dokumentacji patentowej.

W 1970 roku Edo corp. zgłasza patent na wykorzystanie mechanizmu kulkowego do poruszania się w przestrzeni układu współrzędnych biegunowych [3]. Ukażany jest on na rysunku 3.2.

Kolejną firmą, która opatentowała własny model była firma Wico corporation. Ich urządzenie zostało wypuszczone na rynek w 1983 roku. Patent zaprezentowany jest na rysunku 3.3

Aug. 30, 1966

G. M. LAMAN

3,269,190

POSITION CONTROL BALL ASSEMBLY

Filed Aug. 9, 1965

Fig. 1

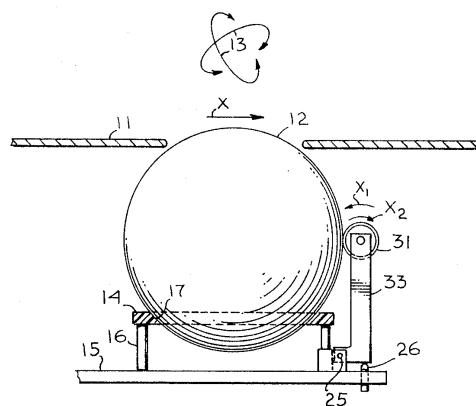
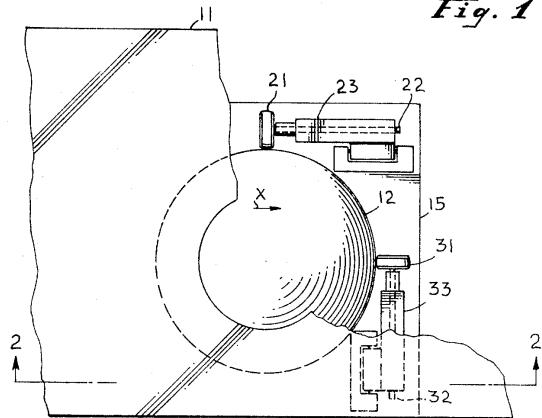


Fig. 2

INVENTOR
GEORGE M. LAMAN
BY Arthur F. Neibich

ATTORNEY

Rysunek 3.1: Fragment dokumentacji patentowej US3269190

PATENTED FEB 15 1972

3,643,148

SHEET 1 OF 2

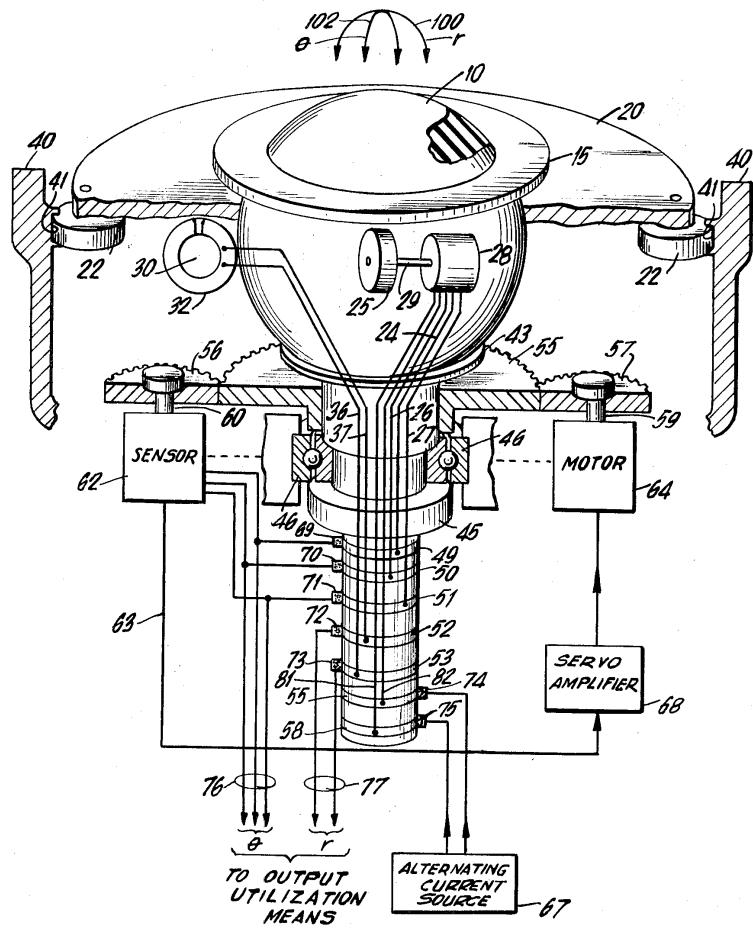


FIG. I

INVENTORS
ARTHUR BROWN
THOMAS W. WONG
BY

Davis, Horie, Faithfull & Hapgood
ATTORNEYS

Rysunek 3.2: Fragment dokumentacji patentowej US3643148

Hewlett Packard company w 1989 roku patentuje manipulator kulkowy, który jako pierwszy wykorzystuje diody emitujące światło i fotokomórki, co pozwala zrezygnować z mechanicznego przenoszenia sygnału do medium.

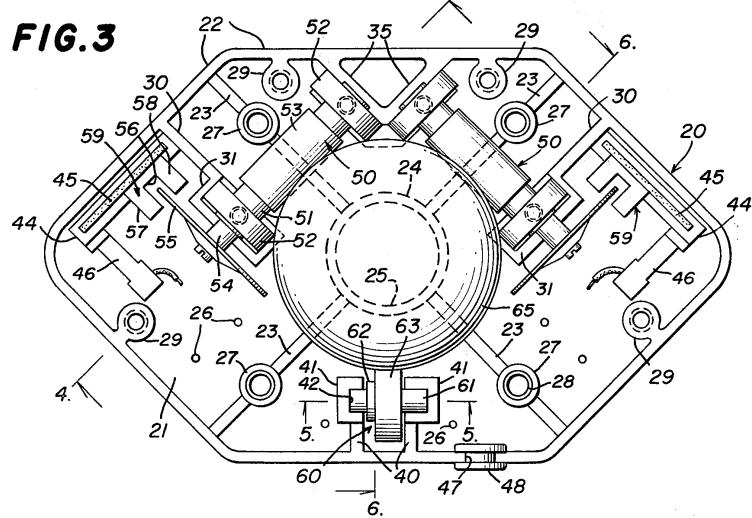
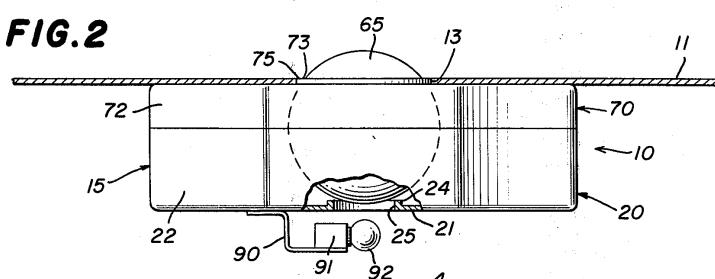
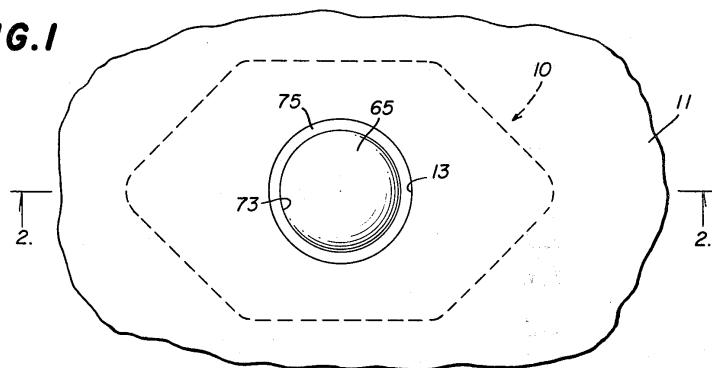
Jako ciekawostkę można przytoczyć wynalazek firmy Matsushita Electric Industrial co., ltd., było to urządzenie wskazujące technologię śledzenia kuli zamknięte w obudowę podobnej do długopisu.

Pierwszym ergonomicznym trackballem był manipulator stworzony przez Dawida Baronowskiego, który opatentował swoje urządzenie w 1993 roku.

Przed nim kolejne urządzenie opracowało dwóch studentów akademii naukowej z Berlina w 1989 roku. Należeli oni do Katedry Fizyki Wysokich Napięć. Ich wynalazkiem było kulkowe urządzenie manipulujące pozycją kurSORA na wyświetlaczu, w szczególności sterowanie ekranem urządzenia komputera. Z ich pomysłu czerpała firma Telefunken budując pierwszą na świecie mysz komputerową.

W lutym 1983 roku firma Wico corporation ponownie zgłosiła patent tym razem w europejskim urzędzie patentowym. 6 lat później firma Tektronix inc. zgłasza patent opracowany przez Nippoldta E. Reubena. Było to już zaawansowane urządzenie co uwidacznia rysunek 3.4 [4].

Urządzenie opatentowane przez Reubena daje początek manipulatorom kulkowym w formie, jaką znamy dziś. Zostaną one opisane w dalszej części pracy.



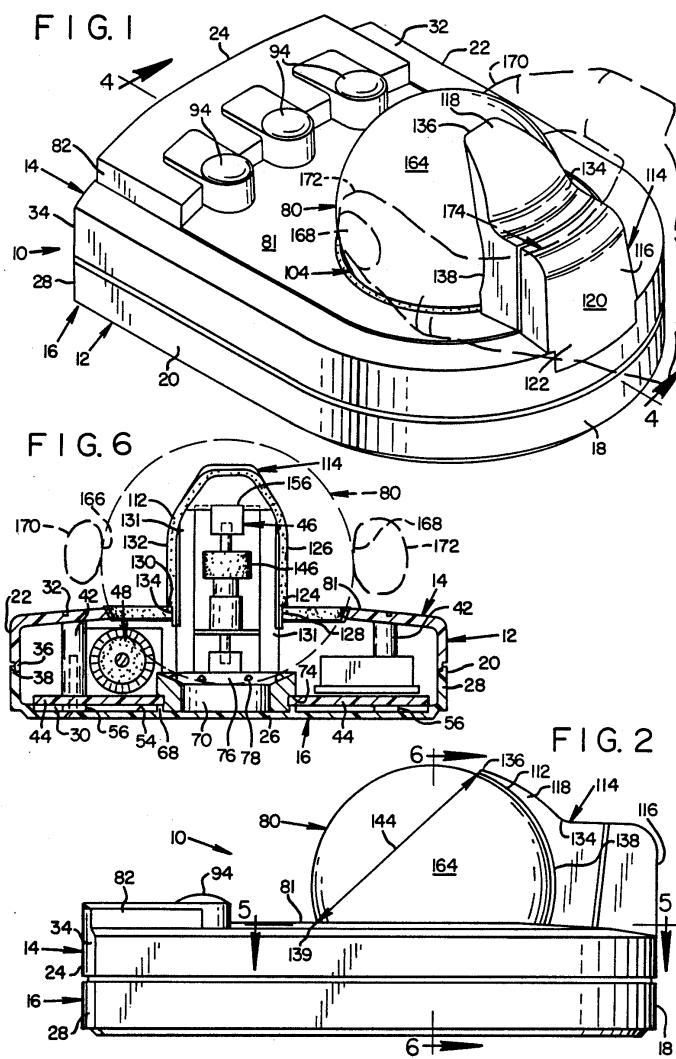
Rysunek 3.3: Fragment dokumentacji patentowej US4404865

U.S. Patent

Aug. 28, 1990

Sheet 1 of 3

4,952,919



Rysunek 3.4: Fragment dokumentacji patentowej US4952919

Rozdział 4

Projekt i wykonanie urządzenia

Słowo trackball należy interpretować jako urządzenie o którym mowa była w rozdziale drugim, a także oprogramowanie sterujące tym urządzeniem należy interpretować jako implementację kodu autora.

4.1 Proces wytwarzania podstawy sprzętowej

W tym rozdziale zostanie opisany proces wytwarzania podstawy sprzętowej urządzenia typu trackball. Będzie to krótki przegląd technologii projektowania wraz z przebiegiem wytwarzyczym. Oprogramowanie sterujące zostanie opisane w rozdziale piątym.

Niezbędnym elementem realizacji projektu było samo urządzenie, jednak modele dostępne na rynku nie pozwalały na implementację własnego rozwiązania. Dlatego autor skonstruował własne urządzenie z podzespołów dostępnych na rynku. Aby skonstruować działające urządzenie należało wykorzystać dostępny model. Płytkę pierwotnie dołączona do urządzenia nie spełniała wymagań autora, ponieważ w projekcie, do urządzenia miał być podłączony mikrokontroler. W tych okolicznościach łatwiej było zaprojektować własny układ elektroniczny, a niżeli wykorzystywać dostępny z urządzeniem. W tym celu należało opracować własny układ elektroniczny, który byłby odpowiedzialny za obsługę komunikacji pomiędzy urządzeniem a komputerem. W celu sprawdzenia poprawność działania urządzenia autor skonstruował zestaw uruchomieniowy.

Przy użyciu tego zestawu urządzenie zostało przetestowane w celu wykrycia potencjalnych wad w działaniu. Podczas tej czynności nie zostały ujawnione żadne wady sprzętu. Działał on w oczekiwany sposób. Rozwiązanie było wzbogacone poprzez nowy, autorski układ elektroniczny.

4.1.1 Opis rodziny wykorzystanego mikrokontrolera

Wykorzystanym mikrokontrolerem jest Atmel ATmega8 [5].

Rodziny mikrokontrolerów Atmel AVR (Alf Egil Bogen and Vegard Wollan's RISC, pierwsze dwa członu pochodzą od imion twórców. Następny zaś pochodzą od metodologii). RISC (Reduced instruction set computing) jest to metodologia projektowania procesorów polegająca na zaimplementowaniu zredukowanego zestawu rozkazów procesora. W odróżnieniu od metodologii CISC (Complex instruction set computing), w której zestaw rozkazów jest liczniejszy. Firma Atmel produkuje ośmiorozkazowe mikroprocesory z rodziną ATmega (megaAVR), w których realizowany jest zmodyfikowany typ architektury Harvardzkiej. Cechuje ją wbudowana, programowalna pamięć flash o rozmiarach od 4 do 256 kilobajtów, od 28 do 100 wyprowadzeń (z których większość jest programowalna), poszerzony zestaw rozkazów spowodowany większą ilością pamięci do zaadresowania, wiele funkcji danego wyprowadzenia.

4.2 Wymagania funkcjonalne

Powstałe urządzenie spełniało następujące wymagania funkcjonalne:

- Urządzenie wymaga podłączenia do komputera poprzez port USB
- Urządzenie musi być typu wskazującego
- Obsługiwane jest przez każdy system operacyjny
- Do obsługi urządzenia wymagany jest operator
- Urządzenie może być obsługiwane niezależnie od powierzchni
- Urządzenie musi posiadać przyciski funkcyjne oraz kule

Aby spełnić powyższe wymagania zaprojektowano nowe rozwiązanie elektroniczne, do obsługi urządzenia. Do projektowania płytka użyto demonstracyjnej wersji programu Eagle [6]. Jest to program do specjalistycznego projektowania rozwiązań elektronicznych. Początkowo w programie tworzy się schemat ideowy. Po połączeniu elementów w schemacie przechodzi się do trybu ich rozmieszczania. W trakcie tej czynności sprawdzana jest poprawność ułożenia ścieżek oraz elementów (czy nie są za blisko lub czy na siebie nie zachodzą) na płytce. Program posiada również proces podglądu gotowej płytki.

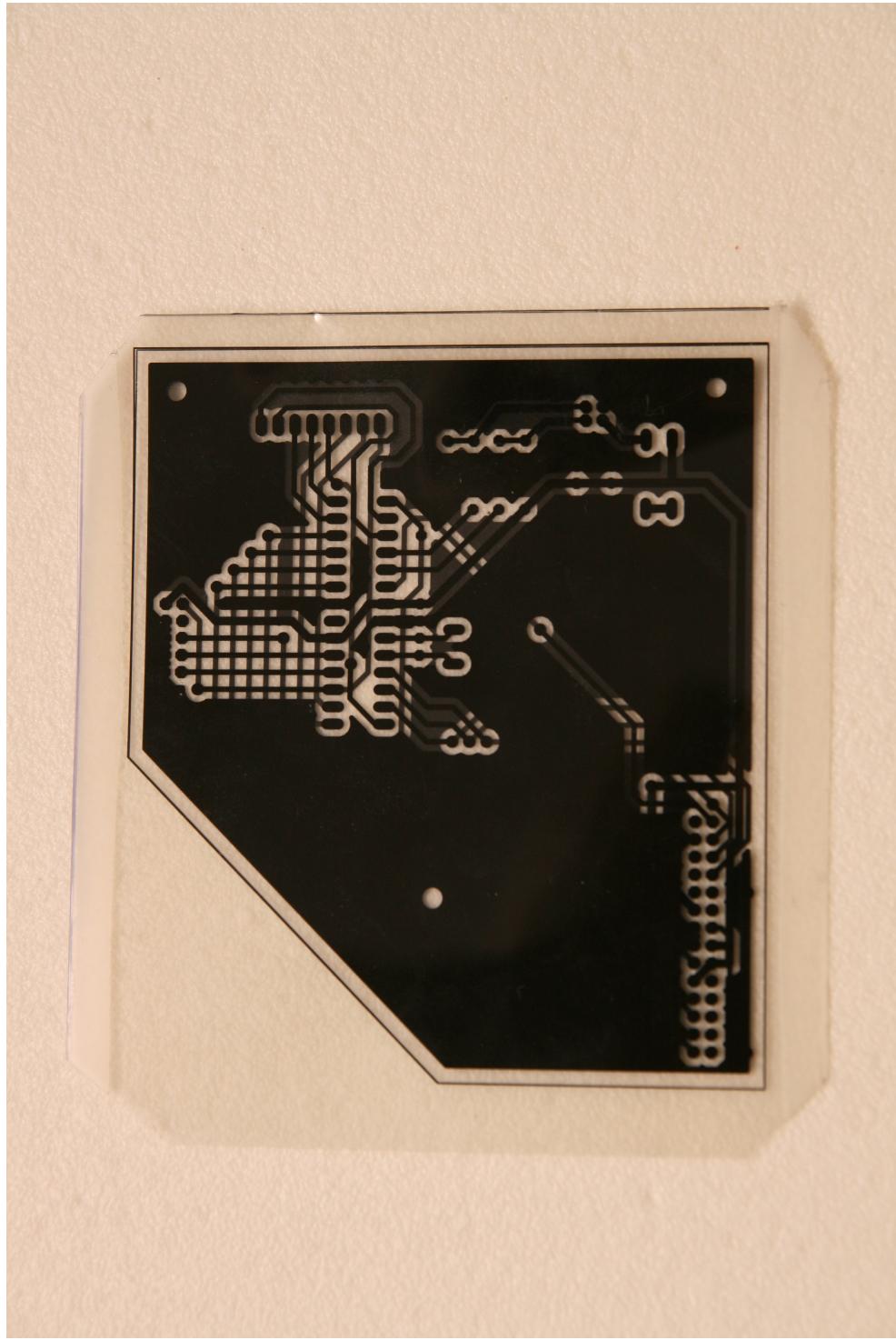
Z programu Eagle eksportuje się plik w formacie postscript, który następnie należy przenieść do dokumentu w formacie pdf. Powinien on zawierać dwie strony, na których znajdują się odbitki stykowe, gdzie pierwsza z nich jest odbita względem osi y. Oba pozytywy muszą być umieszczone w odpowiednim miejscu, by po wywołaniu lakieru światłoczułego pola lutownicze znajdowały się dokładnie nad sobą. Tak przygotowany dokument przekazuje się do firmy zajmującej się naświetlaniem kliszy.

Otrzymany film (rysunek 4.1) łączy się zostawiając miejsce na laminat, a następnie umieszcza go wewnętrz. Następnie naświetla się to promieniowaniem ultrafioletowym z zakresu UV-A. Naświetloną płytke wywołuje się w roztworze sody kaustycznej, a następnie wytrawia. Po tej czynności lakier zmywany jest acetolem. W dalszej kolejności płytka zabezpieczana jest roztworem kalafonii. Po tym jest ona gotowa do wywiercenia otworów lutowniczych oraz montażu elementów.

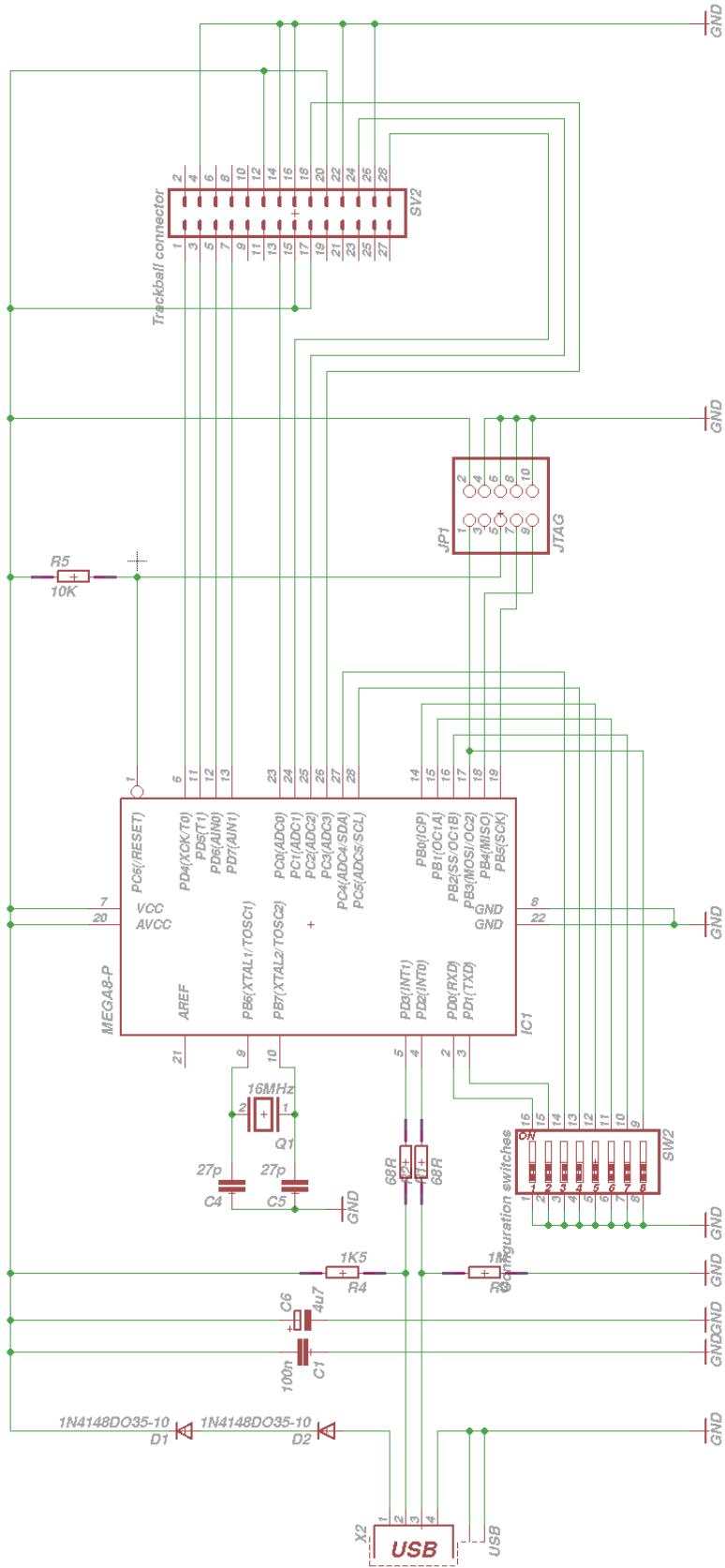
Przed montażem elementów przymontowano zworki łączące obie warstwy płytki. Obie warstwy zostały złączone również poprzez nóżki elementów. Po przymontowaniu elementów sprawdzono poprawność działania układu.

Schemat oraz wygląd projektu płytki przedstawiono na rysunkach 4.2 oraz 4.3.

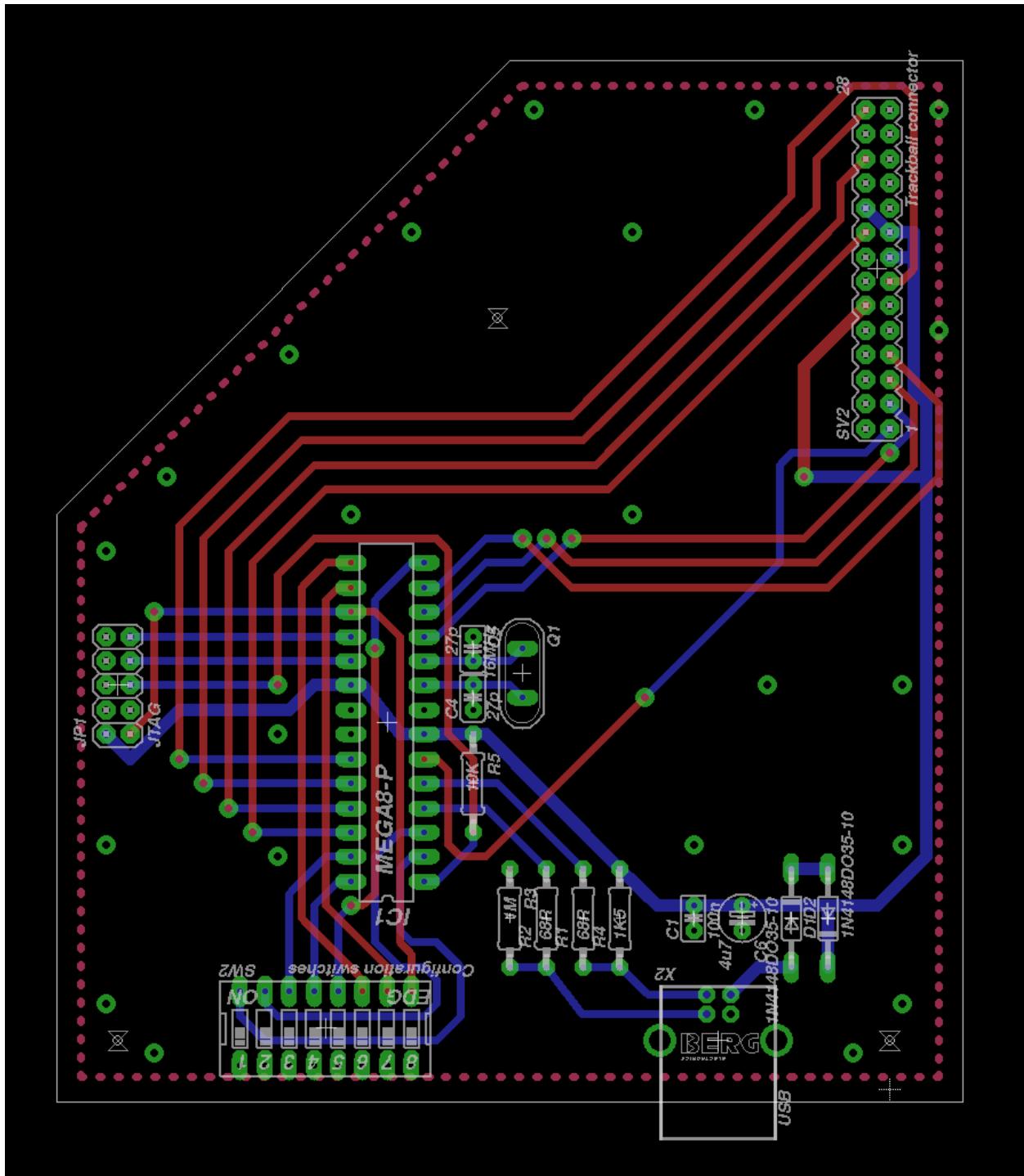
Natomiast fizyczny wygląd po montażu przedstawiony znajduje się na zdjęciach ponumerowanych od 4.4 do 4.11.



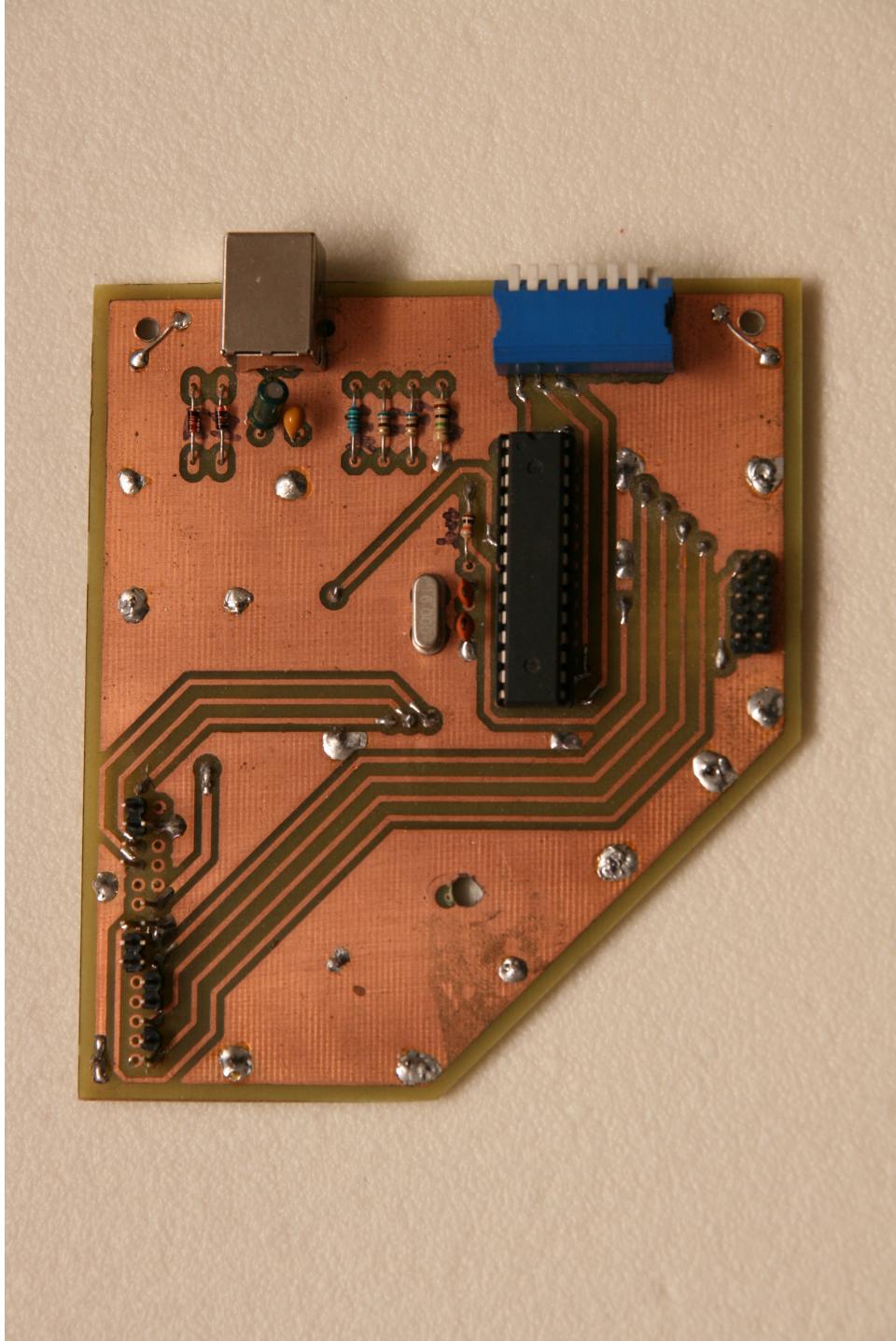
Rysunek 4.1: Zdjęcie kliszy



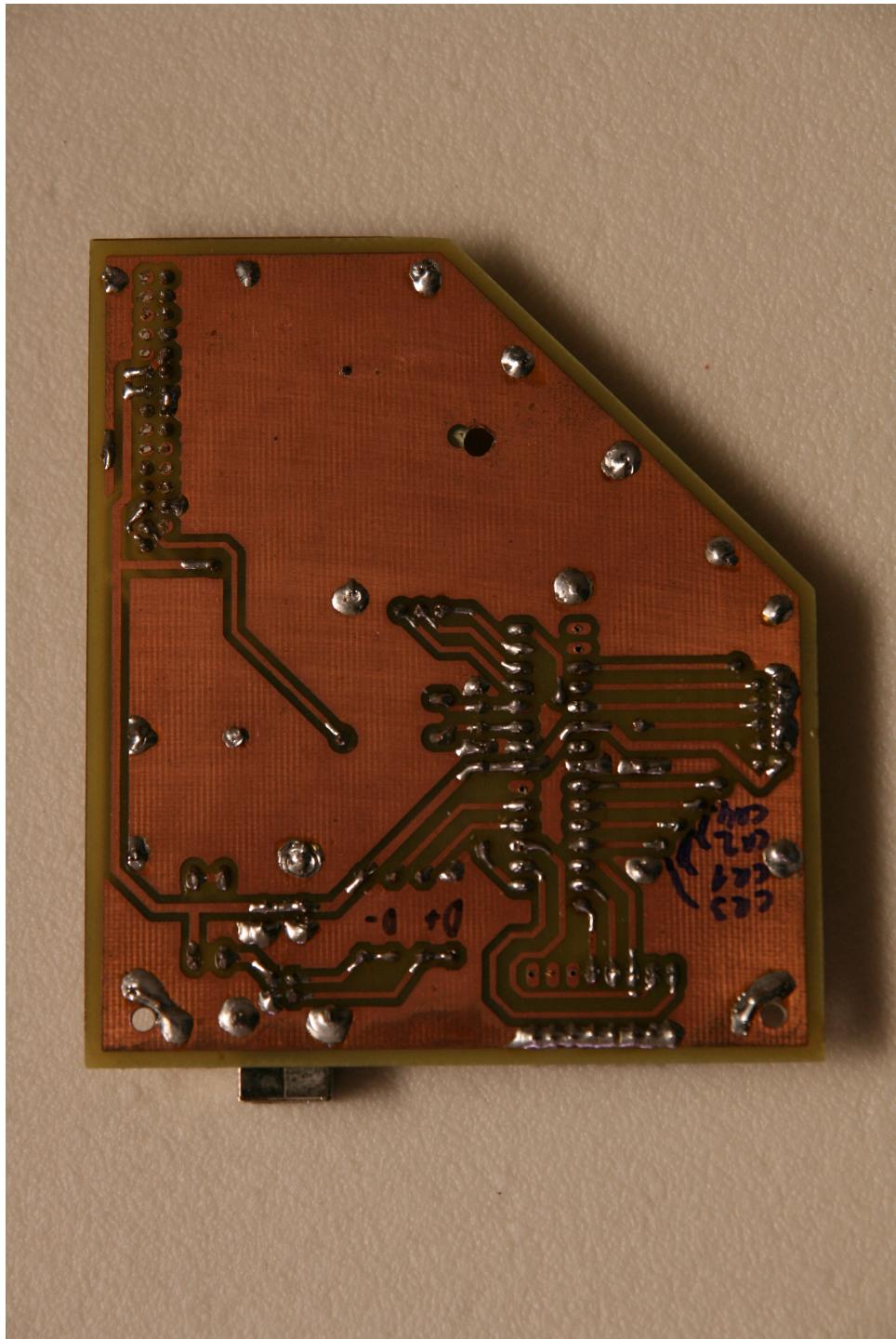
Rysunek 4.2: Schemat ideowy



Rysunek 4.3: Ostateczny wygląd projektu płytki drukowanej



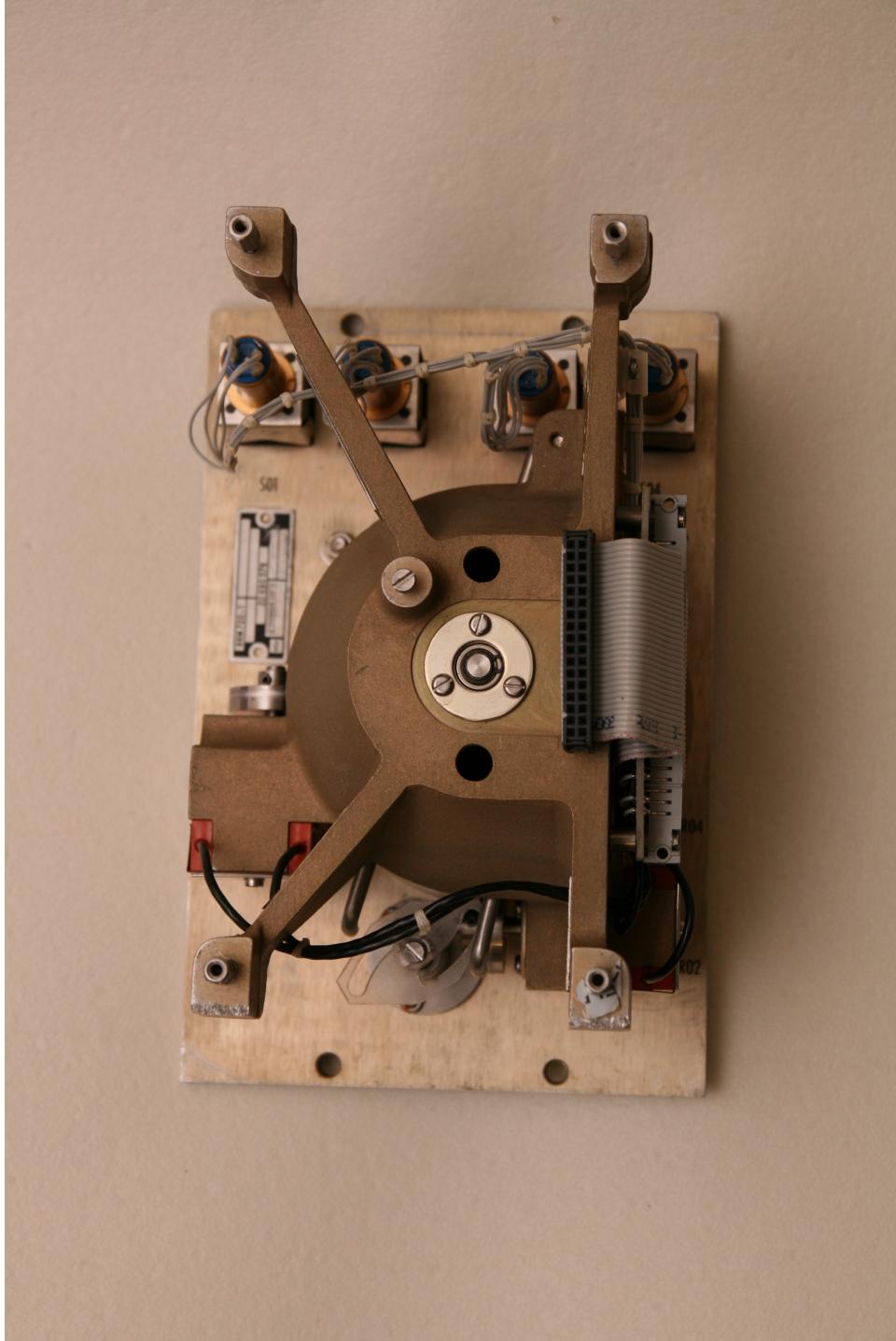
Rysunek 4.4: Zdjęcie płytki drukowanej z zamontowanymi komponentami



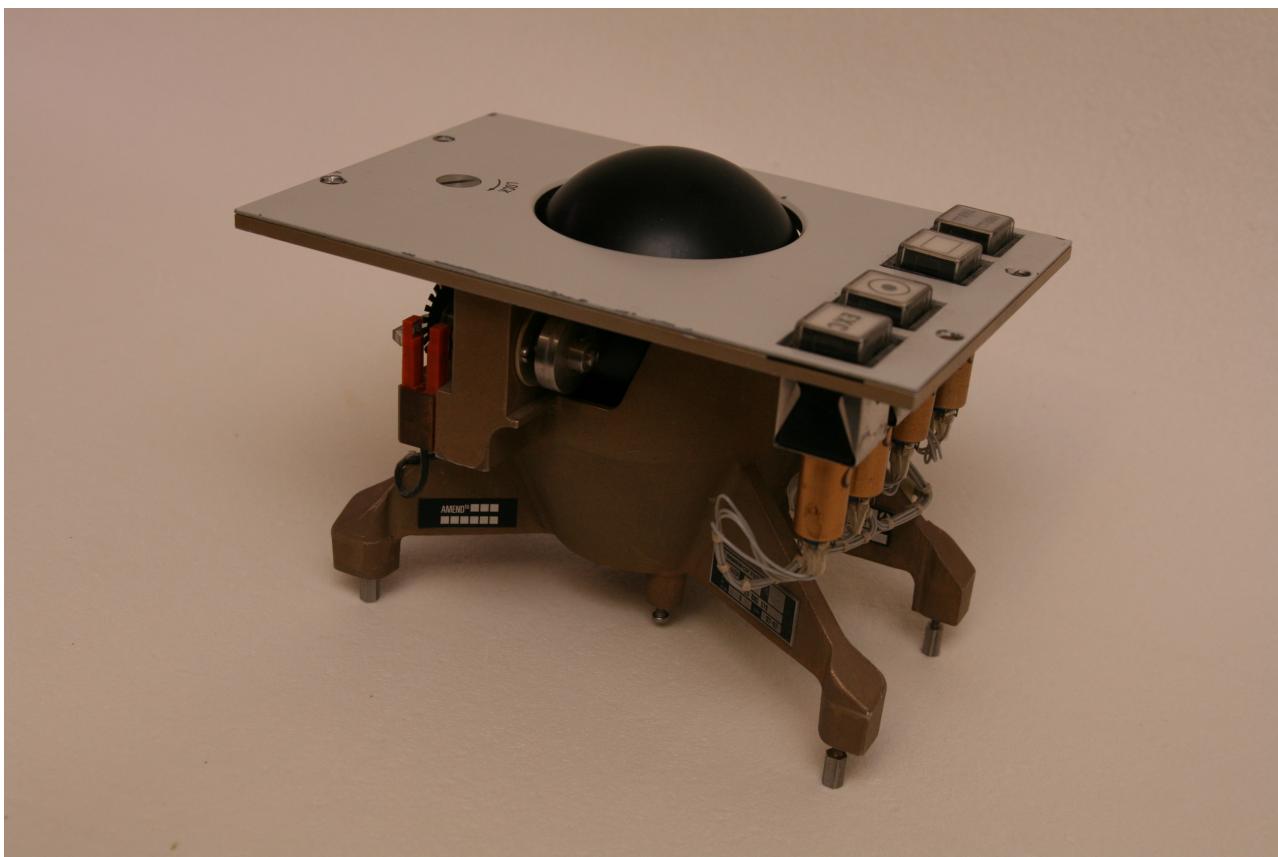
Rysunek 4.5: Zdjęcie spodu płytki drukowanej z zamontowanymi komponentami



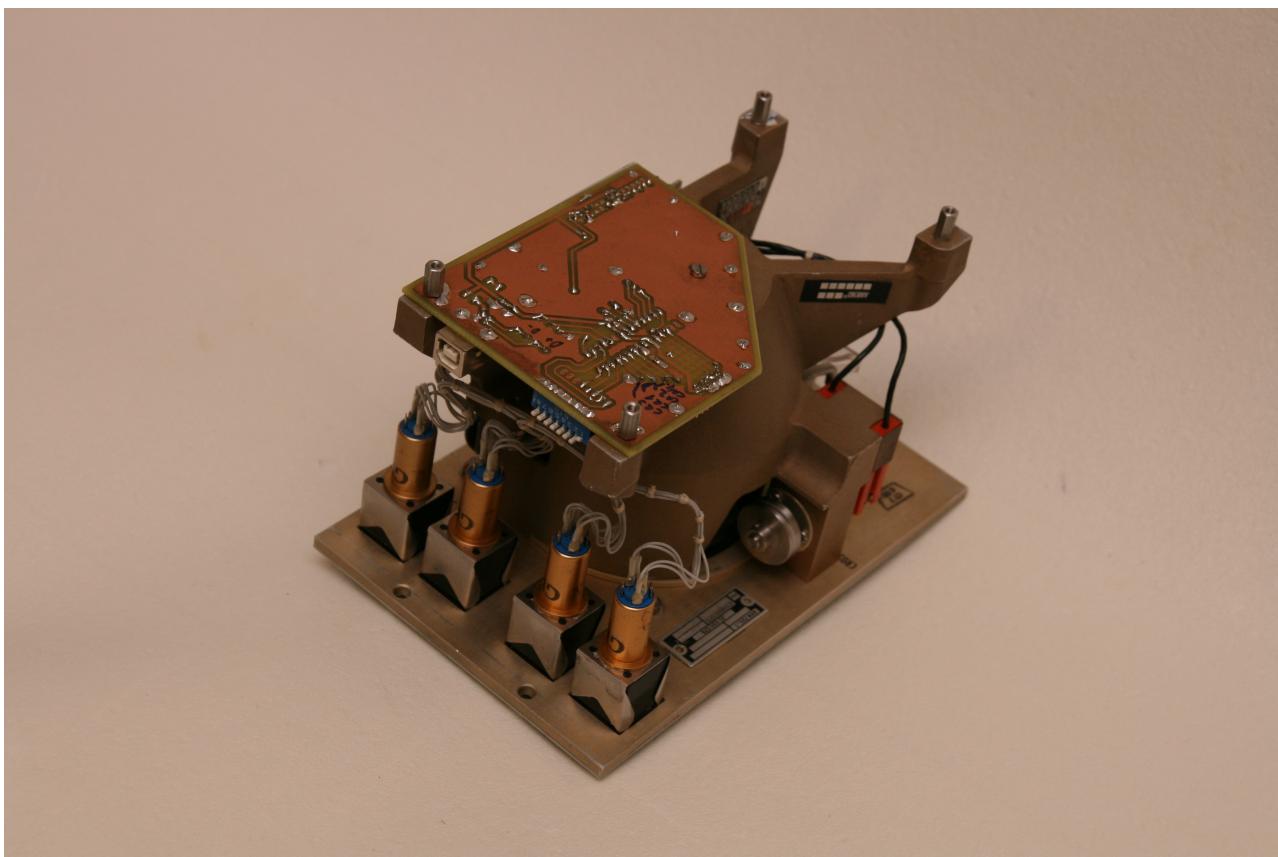
Rysunek 4.6: Zdjęcie trackballa od góry



Rysunek 4.7: Zdjęcie trackballa od spodu



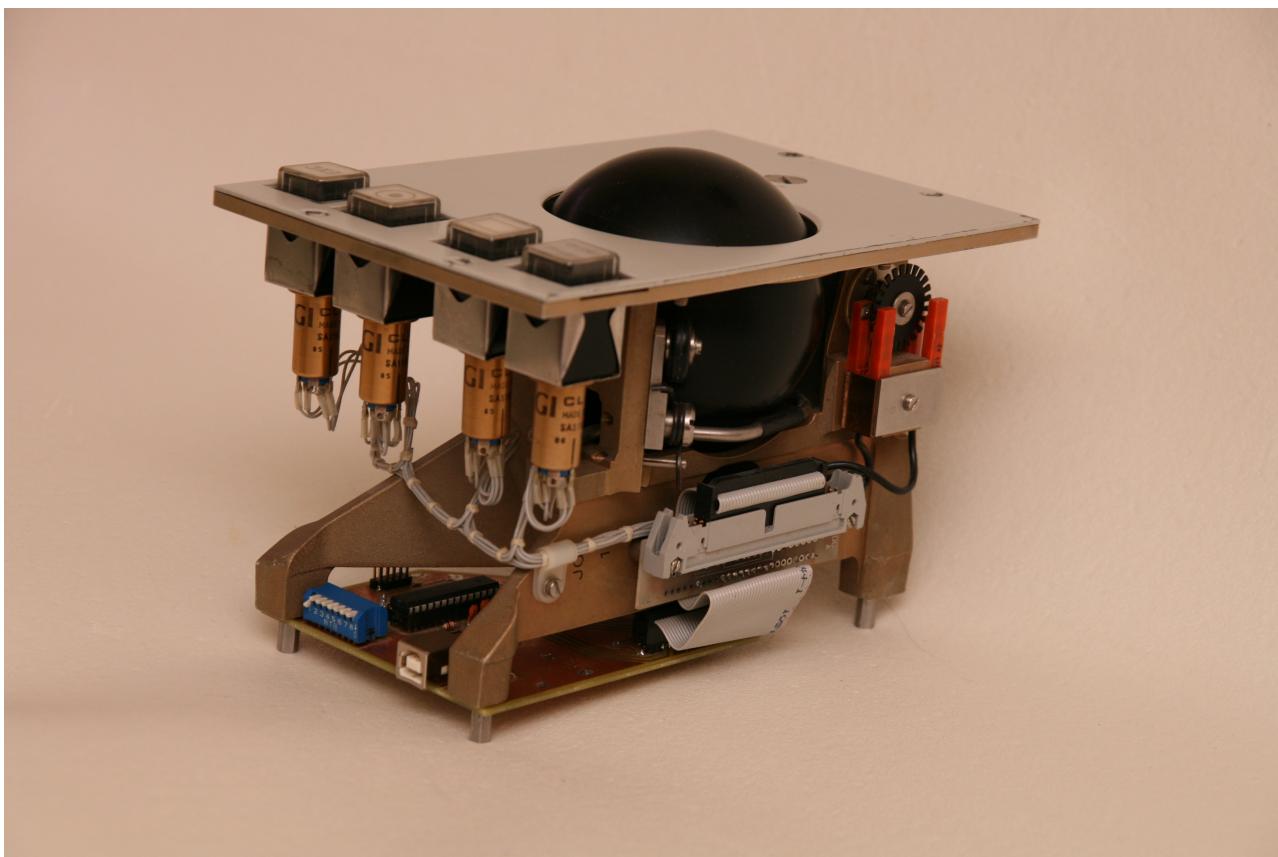
Rysunek 4.8: Zdjęcie trackballa



Rysunek 4.9: Zdjęcie gotowego urządzenia od spodu



Rysunek 4.10: Zdjęcie gotowego urządzenia od góry



Rysunek 4.11: Zdjęcie gotowego urządzenia od boku

4.3 Elementy składowe urządzenia

Urządzenie składa się ze:

- Szkieletu
- Panelu wierzchniego
- Kuli
- Płytki drukowanej
- Mechanizmu przenoszenia zmiany ruchu osi na łopatki przerywające promień podczerwieni
- Fotodiod
- Przycisków
- Złącza trackballa
- Przewodu łączącego trackball z płytą drukowaną

Spis podzespołów na płytce drukowanej:

- IC1 - AVR Atmel ATmega8
- X2 - gniazdo USB
- JP1 - złącze JTAG
- SV2 - złącze Trackballa
- SW2 - przełączniki typu DIP
- Q1 - rezonator kwarcowy 16 MHz
- C1 - kondensator ceramiczny 100 nF
- C4, C5 - kondensatory ceramiczne 27 pF
- C6 - kondensator elektrolityczny 4.7 µF
- R1, R2 - rezystory 68 Ω
- R3 - rezistor 1 MΩ
- R4 - rezistor 1.5 kΩ
- R5 - rezistor 10 kΩ
- D1, D2 - diody 1N4148

4.4 Ułożenie podzespołów płytki drukowanej

Ułożenie podzespołów na płytce drukowanej nie jest przypadkowe. Diody służą jako zmniejszenie ilości napięcia z 5V na 3.3V (a dokładnie 3.6V). Kondensatory znajdują się blisko portu USB, by wytlumić zakłócenia tuż przy samym źródle zasilania. Odległość między portem USB, a mikrokontrolerem jest mała, by wyeliminować interferencje między innymi ścieżkami płytki. Rezonator kwarcowy 16MHz wraz z dwoma kondensatorami znajduje się blisko mikrokontrolera, by wyeliminować zakłócenia oraz by stabilność działania była najwyższa. 8 przełączników konfiguracyjnych podłączonych jest do 8 portów mikrokontrolera w trybie wejścia, by umożliwić podpięcie ich pod zaprogramowane funkcje. Należy uważać na ostatni przełącznik, ponieważ jest on podłączony również do portu JTAG i podczas programowania powinieneń być w pozycji wyłączonej. Zwiera on sygnał do masy, co może utrudnić zaprogramowanie układu.

Na płytce drukowanej dostępny jest port JTAG do ISP (In System Programming), który umożliwia programowanie mikrokontrolera bez wyjmowania go z płytki.

Płytki jest dopasowana do urządzenia. Ścieżki znajdują się z obu stron. Jest ona zamontowana na trzech otworach. W nich znajdują się śrubki, które stykają masę płytki z masą obudowy. Większa część płytki, po obu stronach, pokryta jest masą, która w wielu miejscach zwarta jest przelotkami. Niektóre z elementów należy lutować obustronnie, gdyż część z nich łączy obie warstwy. Ścieżkę zasilaną łatwo zauważać, jest pogrubiona, gdyż płynie nią więcej prądu niż przez pozostałe. Elementy znajdują się tylko po jednej stronie płytki.

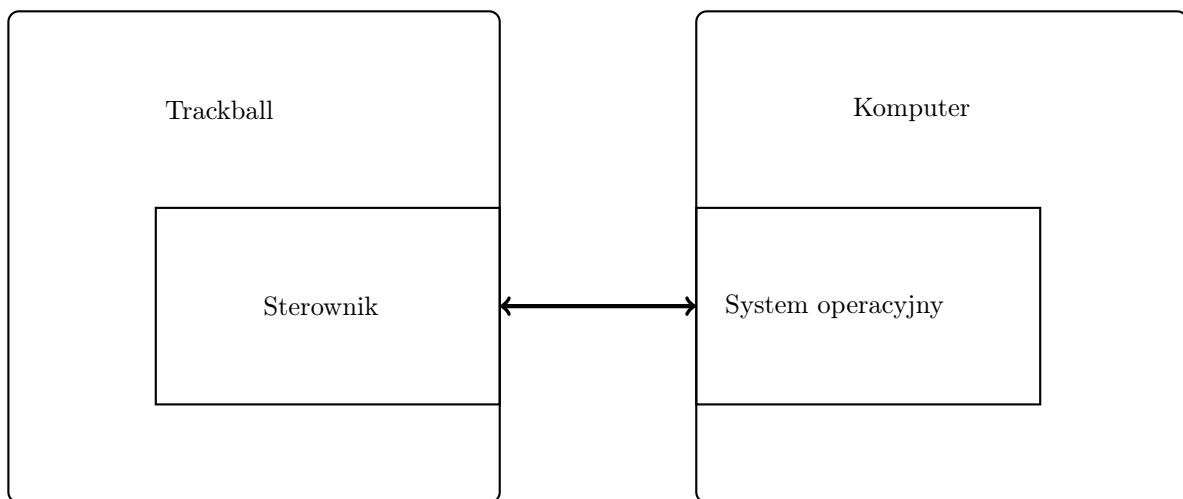
Grubości ścieżek, otworów, rozmieszczenie elementów oraz dokładne parametry techniczne znaleźć można w dołączonym pliku projektu płytki w formacie obsługiwany przez program Eagle.

W kolejnym rozdziale zostanie omówiona implementacja oprogramowania sterującego urządzeniem.

Rozdział 5

Implementacja oprogramowania sterującego

W rozdziale zostanie zaprezentowana implementacja kodu źródłowego, pozwalającego na obsługę urządzenia.



Rysunek 5.1: Diagram komunikacji urządzenia z komputerem

Zaprezentowany diagram pokazuje zależność przepływu danych. Wyraźnie z niego wynika, że niezaprogramowany sprzęt nie jest w stanie spełnić założonych funkcji.

Rozwiązań pod względem elektronicznym nie było wystarczające do obsługi trackballa. Podczas używania urządzenia autor zauważył, że nie spełnia ono podstawowych funkcji. Mianowicie poruszanie kulą

nie przekładało się na zmianę pozycji kurSORA. Aby to zmienić należało umożliwić obustronną komunikację, poprzez zaimplementowanie oprogramowania sterującego.

Oprogramowanie wykorzystane w pierwszej fazie projektu jest przykładowym kodem poruszającym kurSorem zaczerpniętym z biblioteki V-USB [7]. Niestety do działania trackballa to nie wystarczało. Należało uzupełnić kod o funkcje pozwalającą na przekształcenie sygnałów wejściowych w odpowiednią zmianę pozycji kurSORA. Funkcja ta przekształca sygnały w postaci kodu Greya na zmianę pozycji kurSORA w danej osi. Kolejną zaimplementowaną funkcjonalnością jest możliwość kliknięcia, zrealizowana przy użyciu jednego z przycisków znajdujących się na trackballu. Rozwiążanie opracowane przez autora nie pozwalało jednak na używanie rolki. W tym celu użyto jednego z przycisków do przełączania trybu urządzenia. W pierwszym z trybów poruszanie kuli zmienia pozycję kurSORA. W drugim zaś kula pełni rolę rolki jak w tradycyjnej myszce komputerowej.

5.1 Interfejs oraz sposób komunikacji

Do komunikacji po między trackballem, a komputerem użyto interfejsu USB. Interfejs USB (Universal Serial Bus) jest to popularny port komunikacyjny stosowany w komputerach, do podłączania szerokiej gamy urządzeń. Złącze USB pozwala na instalację dodatkowych peryferiów takich jak drukarka, kamera, mysz komputerowa, klawiatura, skaner, czytnik kodów kreskowych, joystick oraz pamięci przenośne różnego rodzaju. Duża część z nich nie wymaga instalacji dodatkowych sterowników, są gotowe do działania zaraz po podłączeniu.

Tabela 5.1: Charakterystyka kolejnych wersji interfejsu. Źródło: wikipedia

Wersja	Ilość styków	Szybkość transferu
1.1	4	1.5Mbit/s
2.0	4	480Mbit/s
3.0	9	4Gbit/s
3.1	9	10Gbit/s

Więcej szczegółów technicznych na temat interfejsu można znaleźć w opracowaniu [8].

5.2 Wybór oraz opis działania biblioteki obsługującej interfejs USB

Dostępnymi bibliotekami spełniającymi założenia projektu są [9]:

Sprzętowe:

- Atmel USB AVR Stack (Atmel Inc.)
- Dr. Stefan Salewski's AT90USB1287 Stack (Dr. Stefan Salewski)
- V-USB
- FreakUSB Stack (FreakLabs)
- PJRC Teensy Stack (Paul Stoffregen)

Programowe:

- AVR309: Software USB (Atmel)
- V-USB (Objective Development)

Do implementacji obsługi interfejsu USB została wykorzystana biblioteka V-USB. Duża popularność biblioteki oraz liczne przykłady użycia przesądziły o jej wyborze. Umożliwia ona komunikację pomiędzy mikrokontrolerem z rodziną AVR, a komputerem. Zaimplementowana została wyłącznie po stronie oprogramowania. Pozwala to na zredukowanie liczby urządzeń potrzebnych do komunikacji. Aby móc skorzystać z biblioteki należy określić deskryptor.

Deskryptor - zbiór wartości opisujących konstrukcję pakietu danych wysyłanych przez urządzenie implementujące bibliotekę. Deskryptor pakietu komunikacyjnego zostaje ustalony w celu konfiguracji biblioteki, aby umożliwić wykorzystanie jej w konkretnym celu.

Na początku należy stworzyć deskryptor zgodny ze specyfikacją protokołu USB. Zawiera on wzór, według którego należy przesyłać dane do hosta, do którego podłączone jest urządzenie.

```
1 PROGMEM const char usbHidReportDescriptor [61] = {  
2     0x05, 0x01, // USAGE_PAGE (Generic Desktop)  
3     0x09, 0x02, // USAGE (Mouse)  
4     0xa1, 0x01, // COLLECTION (Application)  
5  
6     0x09, 0x01, //     USAGE (Pointer)  
7     0xa1, 0x00, //     COLLECTION (Physical)  
8  
9     0x05, 0x09, //     USAGE_PAGE (Button)  
10    0x19, 0x01, //     USAGE_MINIMUM (Button 1)  
11    0x29, 0x03, //     USAGE_MAXIMUM (Button 3)  
12    0x15, 0x00, //     LOGICAL_MINIMUM (0)  
13    0x25, 0x01, //     LOGICAL_MAXIMUM (1)  
14    0x75, 0x01, //     REPORT_SIZE (1)  
15    0x95, 0x03, //     REPORT_COUNT (3)  
16    0x81, 0x02, //     INPUT (Data, Var, Abs)  
17    0x75, 0x05, //     REPORT_SIZE (5)  
18    0x95, 0x01, //     REPORT_COUNT (1)  
19    0x81, 0x03, //     INPUT (Const, Var, Abs)  
20  
21    0x05, 0x01, //     USAGE_PAGE (Generic Desktop)  
22    0x09, 0x30, //     USAGE (X)  
23    0x09, 0x31, //     USAGE (Y)  
24    0x09, 0x38, //     USAGE (Wheel)  
25    0x15, 0x81, //     LOGICAL_MINIMUM (-127)
```

```

26      0x25 , 0x7f , //      LOGICAL_MAXIMUM (127)
27      0x75 , 0x08 , //      REPORT_SIZE (8)
28      0x95 , 0x03 , //      REPORT_COUNT (3)
29      0x81 , 0x06 , //      INPUT (Data, Var, Rel)
30
31      0x05 , 0x0c , //      USAGE_PAGE (Counsumer Devices)
32      0x0a , 0x38 , 0x02 , // USAGE (AC Pan)
33      0x95 , 0x01 , //      REPORT_COUNT (1)
34      0x81 , 0x06 , //      INPUT (Data, Var, Rel)
35
36      0xc0 ,           // END_COLLECTION
37
38      0xc0 ,           // END_COLLECTION
39 };
```

Na potrzeby przesyłania danych autor stworzył własną strukturę odpowiadającą deskryptorowi, którą następnie uzupełnił wartościami. Tak skonstruowany pakiet danych biblioteka przesyła do komputera w celu manipulacji pozycją kurSORA.

Biblioteka jest w pełni zgodna ze standardem USB w wersji 1.1. Wspiera wiele kanałów komunikacyjnych. Domyślnie umożliwia transfer rozmiaru do 254 bajtów. Można tę wartość zwiększyć po odpowiedniej konfiguracji. Biblioteka udostępnia darmowe identyfikatory określające producenta i produkt, które można wykorzystać do własnych zastosowań. Są one potrzebne do zarejestrowania urządzenia w systemie operacyjnym. V-USB jest kompatybilne z mikroprocesorami z rodziny AVR posiadającymi co najmniej 2 kilobajty pamięci flash, 128 bajtów pamięci RAM oraz zegarem taktującym mikrokontroler z częstotliwością nie mniejszą niż 12 MHz. Zegar może być taktowany rezonatorem kwarcowym o częstotliwościach: 12 MHz, 15 MHz, 16 MHz 18 MHz lub 20 MHz. Biblioteka jest napisana w języku C, posiada liczne komentarze ułatwiające jej dalsze użytkowanie oraz jej rozmiar jest niewielki. [10]

5.3 Analiza oprogramowania sterującego

Wszystkie informacje wymagane do obsługi biblioteki są niezbędne w przypadku implementacji własnego oprogramowania, jednak należy także dołączyć inne biblioteki, aby oprogramowanie funkcjonowało. W projekcie użyto następujących bibliotek:

```
1 #include <avr/io.h>
2 #include <avr/wdt.h>
3 #include <avr/interrupt.h>
4 #include <util/delay.h>
5
6 #include <avr/pgmspace.h>
7 #include "usbdrv.h"
```

- io.h - Biblioteka ma za zadanie przygotować pliki nagłówkowe dla danego mikrokontrolera, których nie należy dołączać ręcznie. Znajdują się w nich definicje portów wejścia oraz wyjścia mikrokontrolera. Proces ten sterowany jest ustawniem flagi mmcu przy komplikacji.
- wdt.h - Definiuje interfejs do obsługi makr używanych przez watchdoga [11] występującego w wielu urządzeniach AVR.
- interrupt.h - Biblioteka jest odpowiedzialna za obsługę przerwań oraz ich definiowanie. Zawiera funkcje pozwalające na ich włączanie oraz wyłączanie.
- delay.h - Definiuje funkcje związanie z opóźnianiem wykonywania kodu po uprzednim skonfigurowaniu makra F_CPU do częstotliwości taktowania mikrokontrolera.
- pgmspace.h - Biblioteka odpowiedzialna za dostęp do pamięci flash urządzenia. Jest ona wymagana przez następną bibliotekę.
- usbdrv.h - Definiuje potrzebne makra oraz struktury (usbTxStatus, usbWord, usbRequest).

```

1 #define BUTTONS ( (PIND & 0xe0) >> 5 )
2 #define TOGGLED ( (PIND & 0x10) >> 4 )
3
4 #define CR4      ( (PINC & 0x01)      )
5 #define CR2      ( (PINC & 0x02) >> 1 )
6 #define CR1      ( (PINC & 0x04) >> 2 )
7 #define CR3      ( (PINC & 0x08) >> 3 )
8
9 #define CONF0    ( (PIND & 0x01)      )
10 #define CONF1   ( (PIND & 0x02) >> 1 )
11 #define CONF2   ( (PINC & 0x10) >> 4 )
12 #define CONF3   ( (PINC & 0x20) >> 5 )
13
14 #define SCROLL_STEP_DIVIDER ( 2 << ( 1 << (1 << CONF3) | CONF2) )

```

- BUTTONS - Zawiera aktualne stany wciśniętych przycisków urządzenia.
- TOGGLED - Mówi o tym czy przełącznik główny (przełączający mniędzy trybami myszki i rolki) jest wciśnięty.
- CR1 - CR4 - Przechowują wartości stanów fotodiod odpowiedzialnych za zczytywanie zmiany położenia kuli.
- CONF0 - CONF7 - Określa stan przełączników konfiguracyjnych znajdujących się z tyłu urządzenia. Obecnie tylko cztery pierwsze są w użytku. Reszta z nich przeznaczona jest do przyszłego wykorzystania.
- SCROLL_STEP_DIVIDER - Wartość wykorzystywana w trybie rolki do zmniejszenia ilości kroków, a tym samym spowolnienia przewijania. Ustalana jest na podstawie przełączników konfiguracyjnych o numerach 3 oraz 4.

```

1 #define abs(a) \
2         ((a > 0)? a: -a)
3
4 #define clamp(a, min, max) \
5         ((a > max)? max: (a < min)? min: a)

```

- abs - Wyznacza wartość absolutną z podanej liczby.
- clamp - Normalizuje wartość do podanego przedziału.

```

1 #define LINEARITY ( (1 << CONF3) | CONF2 )
2 #define INTERSECTION 0.5
3
4 #define accelerate(x, y) \
5     x = (1/(INTERSECTION+LINEARITY)) * x * (abs(x) + LINEARITY); \
6     y = (1/(INTERSECTION+LINEARITY)) * y * (abs(y) + LINEARITY)
7
8 #define direction(a, b, pa) \
9     ((a ^ b)? -1: 1) * ((a ^ pa)? 1: -1)
10
11
12 #define move(a, b, pa, pb, report_d, report_wheel, w_operation, \
13         counter, enable_button) \
14 \
15     if((a ^ pa) || (b ^ pb)) { \
16         if (TOGGLED) { \
17             if (scrolling_mode & enable_button \
18                 && abs(counter) == SCROLL_STEP_DIVIDER) \
19             { \
20                 report_wheel += \
21                     direction(a, b, pa); \
22                 change = 1; \
23                 counter = 0; \
24             } else { \
25                 counter += direction(a, b, pa); \
26             } \
27         } else { \
28             report_d w_operation direction(a, b, pa); \
29             change = 1; \
30         } \
31     }

```

- LINEARITY - Przechowuje wartość odpowiedzialną za nagięcie krzywej przyspieszenia. Wartość ta ustalana jest za pomocą przełączników konfiguracyjnych numer 3 oraz 4.
- INTERSECTION - Stała wartość mówiąca o miejscu, w którym krzywa przyspieszenia przecina się z prostą wyznaczającą ruch w przypadku braku przyspieszenia.

- accelerate - Funkcja wyznacza pozycję kurSORA uwzględniając przyspieszenie [12].
- direction - Funkcja zwraca kierunek poruszania się kurSORA w danej osi.
- move - Funkcja uzupełniająca pola struktury odpowiedzialnej za przesunięcie kurSORA bądź pokrętła w zależności od trybu w danej osi.

```

1 PROGMEM const char usbHidReportDescriptor [61] = {
2     0x05, 0x01, // USAGE_PAGE ( Generic Desktop )
3     0x09, 0x02, // USAGE ( Mouse )
4     0xa1, 0x01, // COLLECTION ( Application )
5
6     0x09, 0x01, //     USAGE ( Pointer )
7     0xa1, 0x00, //     COLLECTION ( Physical )
8
9     0x05, 0x09, //     USAGE_PAGE ( Button )
10    0x19, 0x01, //     USAGE_MINIMUM ( Button 1 )
11    0x29, 0x03, //     USAGE_MAXIMUM ( Button 3 )
12    0x15, 0x00, //     LOGICAL_MINIMUM ( 0 )
13    0x25, 0x01, //     LOGICAL_MAXIMUM ( 1 )
14    0x75, 0x01, //     REPORT_SIZE ( 1 )
15    0x95, 0x03, //     REPORT_COUNT ( 3 )
16    0x81, 0x02, //     INPUT ( Data, Var, Abs )
17    0x75, 0x05, //     REPORT_SIZE ( 5 )
18    0x95, 0x01, //     REPORT_COUNT ( 1 )
19    0x81, 0x03, //     INPUT ( Const, Var, Abs )
20
21    0x05, 0x01, //     USAGE_PAGE ( Generic Desktop )
22    0x09, 0x30, //     USAGE ( X )
23    0x09, 0x31, //     USAGE ( Y )
24    0x09, 0x38, //     USAGE ( Wheel )
25    0x15, 0x81, //     LOGICAL_MINIMUM ( -127 )
26    0x25, 0x7f, //     LOGICAL_MAXIMUM ( 127 )
27    0x75, 0x08, //     REPORT_SIZE ( 8 )
28    0x95, 0x03, //     REPORT_COUNT ( 3 )
29    0x81, 0x06, //     INPUT ( Data, Var, Rel )
30

```

```

31      0x05 , 0x0c , //      USAGE_PAGE (Consumer Devices)
32      0x0a , 0x38 , 0x02 , // USAGE (AC Pan)
33      0x95 , 0x01 , //      REPORT_COUNT (1)
34      0x81 , 0x06 , //      INPUT (Data, Var, Rel)
35
36      0xc0 , //      END_COLLECTION
37
38      0xc0 , // END_COLLECTION
39  };

```

Jest to deskryptor raportu. Opisuje on model pakietu danych przesyłanych do hosta. Pierwsze pole mówi o tym, że urządzenie jest urządzeniem wskazującym. Dalej zdefiniowane są przyciski (ich ilość oraz zakres zwracanych wartości). Kolejne pola opisują osie x, y, rolkę oraz zwracane przez nie wartości. Dodatkowo podane jest również nietypowe pole rolki przewijania poziomego. Deskryptor ten jest zbudowany na polu kolekcji.

O większości funkcji przedstawionych w nim pól można przeczytać na stronie ze specyfikacją [13].

```

1 typedef struct {
2     uchar buttons;
3     char dx;
4     char dy;
5     char dvwheel;
6     char dhwheel;
7 } report_t;

```

Struktura report_t zawiera w sobie pola przechowujące wartości opisane uprzednio zdefiniowanym deskryptorem raportu. Kolejno:

- buttons - Wartości wciskniętych przycisków.
- dx - Przesunięcie kurSORA w osi horyzontalnej.
- dy - Przesunięcie kurSORA w osi wertykalnej.
- dvwheel - Przesunięcie pokrętła w osi wertykalnej.
- dhwheel - Przesunięcie pokrętła w osi horyzontalnej.

Struktura ta używana jest do przechowywania danych generowanych przez stan urządzenia gotowych do wysłania w danym oknie czasowym.

```
1 static report_t report;
2 static uchar idleRate;
```

- report instancjonuje uprzednio zdefiniowaną strukturę report_t.
- Zmienna idleRate wykorzystywana w przykładowym projekcie jest do ponawiania sygnału wciśnięcia klawisza.

```
1 usbMsgLen_t usbFunctionSetup(uchar data[8])
2 {
3     usbRequest_t *rq = (void *)data;
4
5     if ((rq->bmRequestType & USBRQ_TYPE_MASK) == USBRQ_TYPE_CLASS) {
6         if (rq->bRequest == USBRQ_HID_GET_REPORT) {
7             usbMsgPtr = (void *)&report;
8             return sizeof(report);
9         } else if (rq->bRequest == USBRQ_HID_GET_IDLE) {
10            usbMsgPtr = &idleRate;
11            return 1;
12        } else if (rq->bRequest == USBRQ_HID_SET_IDLE) {
13            idleRate = rq->wValue.bytes[1];
14        }
15    } else {
16    }
17    return 0;
18 }
```

Funkcja niezbędna do poprawnego zainicjowania portu USB. Dostępna w przykładowym projekcie z początkowej fazą implementacji.

```

1 static volatile uchar key_press;
2
3 ISR(TIMER1_COMPA_vect)
4 {
5     static uchar key_state;
6     static uchar ct0, ct1;
7
8     uchar i = key_state ^ ~BUTTONS;
9
10    ct0 = ~(ct0 & i);
11    ct1 = (ct0 ^ ct1) & i;
12    i &= ct0 & ct1;
13    key_state ^= i;
14
15    key_press |= key_state & i;
16 }
17
18 #define DEBOUNCE 200L
19
20 void timer_init()
21 {
22     TIMSK = 1 << OCIE1A;
23     TCCR1B = (1 << CS10) | (1 << WGM12);
24     OCR1A = F_CPU / DEBOUNCE;
25 }
```

Funkcja odpowiadająca za niwelowanie drgań na stykach przycisków. Innymi słowy naciśnięcie przycisku nie zostaje odebrane jako wielokrotne wcisnięcie tego samego przycisku. Jest to funkcja wywoływana przerwaniem, które następuje wraz z przepełnieniem licznika numer 1. Skonfigurowany jest on w funkcji timer_init przy pomocy wartości z makra DEBOUNCE.

```

1 static void hardware_init(void)
2 {
3     watchdog_enable(WDTO_1S);
4
5     DDRD = 0x00;
6     PORTD = 0xf3; // with pull-ups
7
8     DDRC = 0x00; // First 4 pins - sensors; conf sw
9     PORTC = 0xff; // with pull-ups
10
11    DDRB = 0x30; // debug, conf sw
12    PORTB = 0xcf;
13
14    usbInit();
15    usbDeviceDisconnect();
16
17    uchar i = 0;
18    while (--i) {
19        watchdog_reset();
20        _delay_ms(1);
21    }
22
23    usbDeviceConnect();
24
25    sei();
26 }

```

Przedstawiona funkcja inicjuje urządzenie. Na początku na sekundę wyłączany jest watchdog w celu zainicjowania portu USB, następnie ustawiane są kolejne kierunki komunikacji portów oraz zaczyna się włączanie rezystorów pull-up (odpowiedzialnych za początkowy stan wyjścia, czyli domyślnym stanem jest stan wysoki). W dalszej części jest inicjowany port USB, poprzez tymczasowe rozłączenie, oczekanie kwantu czasu oraz ponowne podłączenie urządzenia. W ostatniej fazie jest włączana globalna obsługa przerwań.

```

1 static uchar v1, v2, v3, v4, p1, p2, p3, p4;
2 static uchar buttons_now, buttons_prev, scrolling_mode;
3 static uchar change;
4

```

```

5 static char v_counter = 0;
6 static char h_counter = 0;
```

Zmienne z pierwszej linii przechowują stan fotodiod: v - aktualny, p - poprzedni. Kolejna linia zawiera zmienne przechowujące stan przycisków: aktualny oraz poprzedni. Ostatnia z nich przechowuje tryb przewijania rolki. Zmienna change mówi o tym, czy urządzenie powinno wysłać pakiet po zmianie stanu. Następne dwie zmienne są licznikami wyznaczającymi moment zmiany pozycji rolki w danej osi.

```

1 int __attribute__((noreturn)) main(void)
2 {
3     timer_init();
4     hardware_init();
5
6     change = 0;
7     buttons_prev = 0;
8     scrolling_mode = 0x04;
9
10    p1 = CR1;
11    p3 = CR3;
12    p2 = CR2;
13    p4 = CR4;
14
15    for (;;) {
16        wdt_reset();
17        usbPoll();
18
19        buttons_now = key_press;
20
21        v1 = CR1;
22        v3 = CR3;
23        v2 = CR2;
24        v4 = CR4;
25
26        if (buttons_prev != buttons_now) {
27            key_press = 0;
28            if(TOGGLED) {
29                if(buttons_now & 0x07) {
```

```

30                     scrolling_mode = buttons_now;
31                     v_counter = h_counter = SCROLL_STEP_DIVIDER;
32                 }
33             } else {
34                     report.buttons = buttons_now;
35                     change = 1;
36             }
37         }
38
39         move(v1, v3, p1, p3, report.dx, report.dhwheel, +=, h_counter, 0x06);
40         move(v2, v4, p2, p4, report.dy, report.dvwheel, -=, v_counter, 0x05);
41
42         buttons_prev = buttons_now;
43
44         p1 = v1;
45         p2 = v2;
46         p3 = v3;
47         p4 = v4;
48
49     if (change && usbInterruptIsReady()) {
50         if (!CONF0) {
51             accelerate(report.dx, report.dy);
52         }
53
54         usbSetInterrupt((void *)&report, sizeof(report));
55
56         report.dx      = 0;
57         report.dy      = 0;
58         report.dvwheel = 0;
59         report.dhwheel = 0;
60
61         change = 0;
62     }
63 }
64 }
```

W dalszej części zostanie opisana główna funkcja programu. Linia 3 oraz 4 są wywołaniem funkcji inicjujących urządzenie. Funkcje te zostały już opisane na początku rozdziału. Następnie inicjowane są zmienne odpowiadające za zmianę stanu, wartości przycisków oraz trybu przewijania rolki do wartości domyślnych. Kolejnym krokiem jest ustawienie wartości p1 do p4 odpowiadających za stan fotodiod. Dalsza część sterowania przekazywana jest do nieskończonej pętli. W niej watchdog zostaje zresetowany w celu zapewnienia poprawnego działania urządzenia. Następnie wywoływane są funkcje usbPoll odpowiedzialna za utrzymywanie komunikacji z hostem. buttons_now ustawiane jest na aktualny stan wcisniętych przycisków, zapisywany jest również, do odpowiednich zmiennych, aktualny stan sygnałów fotodiod. Sterowanie przenoszone jest do wnętrza bloku warunkowego jeśli stan przycisków jest różny od poprzedniego. Następuje wtedy szereg testów mających na celu ustalenie dokąd przesłany zostanie stan wartości przycisków (odpowiednio do trybu przewijania w wypadku trybu rolki lub do stanu przycisków myszki w trybie przesuwania kurSORA). Następnie jest wykonywany szereg testów dotyczących zmiany pozycji kurSORA (w trybie poruszania kurSORA), bądź zmiany pozycji rolki (w trybie poruszania rolki) dla danej osi. Przedostatnią rzeczą jest ustawienie wartości buttons_prev oraz zmiennych stanów fotodiod dla kolejnego przebiegu pętli. Na samym końcu, gdy nastąpiła zmiana stanu urządzenia, oraz gdy host jest gotowy, aby przyjąć dane (sprawdzane jest to przy pomocy funkcji usbInterruptIsReady), wyliczana jest pozycja kurSORA po przyspieszeniu, jeśli przełącznik konfiguracyjny numer jeden jest włączony, ustawiane jest przerwanie sygnalizujące chęć przesłania danych raportu, a następnie, zaraz po przesłaniu danych, pola raportu ustawiane są do wartości początkowych zarówno jak i wartość zmienne eliminować stanu urządzenia.

Odpowiednim przygotowaniem zestawu uruchomieniowego było zaprogramowanie mikrokontrolera. Do wykonania tej czynności niezbędny jest programator dedykowany do programowania mikrokontrolerów z rodziną AVR podłączany przez złącze JTAG. Kolejnym niezbędnym elementem było kompilator avr-gcc. Kompiluje on kod do postaci pliku elf, który następnie trzeba przekazać do programu avr-objcopy. Kolejnym krokiem było sprawdzenie czy rozmiar napisanego programu po skompilowaniu nie przekracza rozmiaru wewnętrznej pamięci mikrokontrolera. Aby to stwierdzić należało uruchomić program avr-size z podanym plikiem wynikowym. Rozmiar pliku był mniejszy niż wielkość pamięci wbudowanej w mikrokontroler, więc można było przystąpić do zaprogramowania mikrokontrolera.

Zaprogramowany mikrokontroler oraz urządzenie połączone z komputerem stwarza możliwość przetestowania rozwiązania. Dlatego w kolejnym rozdziale opisane zostanie proces testowania.

Rozdział 6

Testowanie

Rozdział opisuje proces testowania gotowego urządzenia typu trackball. Jego zachowanie po podłączeniu do komputera oraz opis czynności mający na celu poprawne użytkowanie.

Po skompletowaniu urządzenia rozpoczęła się faza testów.

6.1 Testy funkcjonalne

Tabela 6.1: Test funkcjonalny: Sprawdzenie napięcia

Nazwa	Sprawdzenie napięcia
Krótki opis	Sprawdzenie czy płytka urządzenia jest zasilana poprzez port USB.
Warunki wstępne	<ul style="list-style-type: none">• Włączony komputer z zainstalowanym systemem operacyjnym• Urządzenie trackball podłączone do komputera za pomocą przewodu USB• Multimetr lub dowolne urządzenie mierzące spadek napięcia
Kroki	<ol style="list-style-type: none">1. Włączenie multimetru2. Zmierz spadek napięcia na kondensatorze elektrolitycznym (C6)
Oczekiwany rezultat	Napięcie rzędu 3,3 do 3,7 V.
Warunek końcowy	Zmierzone napięcie mieści się w zakresie oczekiwanej.
Wynik testu	POZYTYWNY

Tabela 6.2: Test funkcjonalny: Wykrywanie urządzenia

Nazwa	Wykrywanie urządzenia
Krótki opis	Sprawdzenie czy urządzenie jest wykrywane w Systemie Operacyjnym.
Warunki wstępne	<ul style="list-style-type: none"> • Włączony komputer z zainstalowanym systemem operacyjnym • Urządzenie trackball podłączone do komputera za pomocą przewodu USB • Przewód USB
Kroki	1. Podłączenie urządzenie do komputera za pomocą przewodu USB
Oczekiwany rezultat	Urządzenie zostanie rozpoznane przez system operacyjny.
Warunek końcowy	Urządzenie zostało wykryte i rozpoznane jako urządzenie wskazujące typu HID.
Wynik testu	NEGATYWNY

Po złożeniu zestawu (podłączenia płytka do urządzenia), trackball został podłączony do komputera przewodem USB. Jednak urządzenie nie zadziałało w oczekiwany sposób. Stało się tak ponieważ w procesie sprawdzania poprawności schematu wystąpił błąd. Mianowicie rezystancja oporników była za wysoka. Można było to stwierdzić, po zauważeniu zbyt niskiej amplitudy badanego sygnału, poprzez pomiar oscyloskopem. Wymiana nieodpowiednich rezystorów poskutkowała poprawnym działaniem układu. Można było przejść zatem do dalszego testowania urządzenia.

Tabela 6.3: Test funkcjonalny: Sprawdzenie działania przycisków

Nazwa	Sprawdzenie działania przycisków
Krótki opis	Sprawdzenie poprawnego działania przycisków na trackballu.
Warunki wstępne	<ul style="list-style-type: none"> • Włączony komputer z zainstalowanym systemem operacyjnym • Urządzenie trackball podłączone do komputera za pomocą przewodu USB
Kroki	<ol style="list-style-type: none"> 1. Nacisnąć dany przycisk 2. Obserwować efekt 3. Porównać działanie przycisku z oczekiwany
Oczekiwany rezultat	<p>W trybie myszki:</p> <ul style="list-style-type: none"> • Przycisk drugi działa jak lewy przycisk myszy • Przycisk trzeci działa prawy przycisk myszy • Przycisk czwarty działa jak środkowego przycisku myszy <p>W trybie przewijania:</p> <ul style="list-style-type: none"> • Przycisk drugi odpowiada za przejście w tryb przewijania w pionie • Przycisk trzeci odpowiada za przejście w tryb przewijania w poziomie • Przycisk czwarty odpowiada za przejście w tryb jednoczesnego przewijania obu osi
Warunek końcowy	Wszystkie z przycisków spełniają zakładane funkcje.
Wynik testu	POZYTYWNY

Tabela 6.4: Test funkcjonalny: Tryb przewijania

Nazwa	Tryb przewijania
Krótki opis	Sprawdzenie czy jest możliwe przewijanie w trybie rolki.
Warunki wstępne	<ul style="list-style-type: none"> • Włączony komputer z zainstalowanym systemem operacyjnym oraz przeglądarką internetową • Urządzenie Trackball podłączone do komputera za pomocą przewodu USB • Włączony tryb przewijania
Kroki	<ol style="list-style-type: none"> 1. Odwiedzić dowolną witrynę internetową dłuższą niż wysokość ekranu 2. Obrócić kulę w dół 3. Obserwować czy strona internetowa została przewinięta
Oczekiwany rezultat	Storna internetowa zostaje przewijana w trakcie poruszania kula.
Warunek końcowy	Ruch kuli wywołuje efekt, witryna przesuwa się w sposób, w jaki powinna się przesuwać podczas manipulacji rolką myszki.
Wynik testu	POZYTYWNY

6.2 Testy zgodności

Tabela 6.5: Test zgodności: Kompatybilność w systemach operacyjnych

Nazwa	Kompatybilność w systemach operacyjnych
Krótki opis	Sprawdzenie działania trackballa w różnych systemach operacyjnych
Warunki wstępne	<ul style="list-style-type: none">• Komputery z systemami operacyjnymi: Microsoft Windows, Linux• Urządzenie trackball z przewodem USB
Kroki	<ol style="list-style-type: none">1. Podłączyć trackball do komputera z systemem Microsoft Windows2. Poruszyć kulą3. Podłączyć trackball do komputera z systemem Linux4. Poruszyć kulą
Oczekiwany rezultat	Urządzenie zostaje wykryte w obu systemach operacyjnych oraz działa prawnie
Warunek końcowy	Urządzenie zostaje rozpoznane w obu systemach operacyjnych.
Wynik testu	POZYTYWNY

6.3 Testy wydajności

Tabela 6.6: Test wydajności: Test wytrzymałości

Nazwa	Test wytrzymałości
Krótki opis	Sprawdzenie działania urządzenia przy jego intensywnym używaniu
Warunki wstępne	<ul style="list-style-type: none">• Włączony komputer z zainstalowanym systemem operacyjnym• Urządzenie Trackball podłączone do komputera za pomocą przewodu USB
Kroki	<ol style="list-style-type: none">1. Energicznie poruszać kulą2. Intensywnie naciskać przyciski
Oczekiwany rezultat	Urządzenie kontynuuje działać poprawnie.
Warunek końcowy	Urządzenie działa poprawnie.
Wynik testu	POZYTYWNY

W trakcie działania urządzenia zabrakło dwóch znaczących funkcji. Mianowicie takiej, która eliminuje drgania styków podczas przyciskania klawiszy oraz przyspieszenia poruszania kurSORA. Funkcje te zostały zaimplementowane na końcu fazy testów. Są one punktem wyjścia do dalszego rozwoju oprogramowania sterującego urządzeniem typu trackball poprzez interfejs USB.

Rozdział 7

Zakończenie

Przedstawiony projekt zachęca do zastanowienia się nad alternatywnym rozwiązaniem dla urządzenia wskazującego, którego najpopularniejszą opcją nadal pozostaje mysz komputerowa. Trackball jest jednak w kwestii precyzji niezastąpionym urządzeniem, które stosowane jest do dziś tam gdzie zwykła myszka to za mało. Używanie trackballa wymaga o wiele mniejszej ilości miejsca przy stanowisku roboczym, oraz nadgarstek nie męczy się tak szybko jak przy zwykłej myszy. Powodem powstania projektu była chęć wypróbowania alternatywy dla urządzenia wskazującego jakim jest myszka. Niewątpliwą zaletą była również mała ilość miejsca potrzebnego do użytkowania trackballa oraz bardzo wysoka precyzja wykonywanych operacji.

Cel projektu został osiągnięty, w wyniku czego powstało działające urządzenie, spełniające oczekiwane funkcje. Praca nad projektem pozwoliła wykorzystać umiejętności, które posiadał autor. Przyczyniły się one do powstania gotowego rozwiązania. Najważniejszą częścią budowy urządzenia było napisanie oprogramowania sterującego, ponieważ bez tej części urządzenie było by niezdolne do działania. Kolejną równie istotną czynnością było wykonanie podzespołu elektronicznego (płytki), który przekształcał ruch kuli na dane gotowe do wysłania poprzez interfejs USB do hosta. Projekt ten może być w pełni zrealizowany na podstawie niniejszej pracy, jednak wymaga pewnych umiejętności specjalistycznych, w przypadku pojawiения się błędów.

W trakcie trwania projektu można było nabyć wielu przydatnych umiejętności (zwłaszcza w dziedzinie systemów wbudowanych) niezbędnych do późniejszej pracy w dziedzinach dotyczących informatyki. Autor zdobył także wiedzę na temat programowania oraz eksploatacji mikrokontrolerów. Kolejną rzeczą było zintegrowanie biblioteki V-USB z kodem własnego programu (poznanie jej interfejsu) oraz innych jej zastosowań [14]. Nastecną ważną umiejętnością, która została zdobыта w czasie wykonywania projektu, było projektowanie płytki drukowanej, jak i jej montaż. Dzięki doświadczeniu autora w oprogramowywaniu urządzeń powstała funkcja odpowiedzialna za eliminację drgania styków, które powstają w związku z przyciśnięciem klawisza. Kolejnym usprawnieniem było prawidłowe rozmieszczenie elementów na płytce drukowanej, w taki sposób,

aby wyeliminować interferencje oraz zakłócenia. Następnym usprawnieniem było zaimplementowanie funkcji odpowiedzialnej za przyspieszanie ruchu kurSORA, a także zdolność urządzenia do zmiany trybu w tryb rolki (znany z tradycyjnej myszki).

Realizowany projekt pozwolił autorowi w pełni wykorzystać potencjał, a także stworzyć działające urządzenie od podstaw.

Bibliografia

- [1] Opis urządzenia typu trackball
<https://pl.wikipedia.org/wiki/Trackball>
Dostęp: 2016.04.10
- [2] <https://www.google.com/patents/US3269190>
Dostęp: 2016.04.03
- [3] <https://www.google.com/patents/US3643148>
Dostęp: 2016.04.03
- [4] <https://www.google.com/patents/US4952919>
Dostęp: 2016.04.03
- [5] Specyfikacja mikrokontrolera Atmel ATmega8
http://www.atmel.com/images/atmel-2486-8-bit-avr-microcontroller-atmega8_1_datasheet.pdf
Dostęp: 2016.04.23
- [6] Program Eagle do projektowania rozwiązań elektronicznych
<http://www.cadsoftusa.com/>
Dostęp: 2016.04.10
- [7] Strona domowa biblioteki V-USB
<https://www.obdev.at/products/vusb/index.html>
Dostęp: 2016.04.03
- [8] Specyfikacja interfejsu USB
<https://pl.wikipedia.org/wiki/USB>
Dostęp: 2016.04.03
- [9] Spis dostępnych bibliotek dla mikrokontrolerów implementujących obsługę interfejsu USB
http://www.fourwalledcubicle.com/files/LUFA/Doc/120219/html/_page__alternative_stacks.html
Dostęp: 2016.04.03

[10] Strona biblioteki V-USB

<https://www.obdev.at/products/vusb/index.html>

Dostęp: 2016.04.03

[11] Wyjaśnienie mechanizmu Watchdoga

<https://pl.wikipedia.org/wiki/Watchdog>

Dostęp: 2016.04.10

[12] Opis algorytmu przyspieszenia ruchu kurSORA

<http://forum.unity3d.com/threads/mouse-acceleration-curves.11405/>

Dostęp: 2016.04.10

[13] Dokumentacja deskryptora USB

http://www.usb.org/developers/hidpage/Hut1_12v2.pdf

Dostęp: 2016.04.10

[14] Przykłady projektów wykorzystujących bibliotekę V-USB

<https://www.obdev.at/products/vusb/prjall.html>

Dostęp: 2016.04.10

Spis rysunków

2.1	Wygląd przykładowego trackballa	7
2.2	Schemat kodowania kodu Greya dla 3 bitów przedstawiony na kole	8
2.3	Diagram przebiegu wartości stanów dwóch sygnałów dla kolejnych wartości	8
2.4	Tabela przebiegu kolejnych faz przy obrocie koła w daną stronę w postaci danych dostarczanych do mikrokontrolera	9
2.5	Trackball Logitech TrackMan Marble	11
2.6	Trackball Kensington Slimblade	12
2.7	Trackball Logitech M570	13
3.1	Fragment dokumentacji patentowej US3269190	15
3.2	Fragment dokumentacji patentowej US3643148	16
3.3	Fragment dokumentacji patentowej US4404865	18
3.4	Fragment dokumentacji patentowej US4952919	19
4.1	Zdjęcie kliszy	23
4.2	Schemat ideowy	24
4.3	Ostateczny wygląd projektu płytki drukowanej	25
4.4	Zdjęcie płytki drukowanej z zamontowanymi komponentami	26
4.5	Zdjęcie spodu płytki drukowanej z zamontowanymi komponentami	27
4.6	Zdjęcie trackballa od góry	28
4.7	Zdjęcie trackballa od spodu	29
4.8	Zdjęcie trackballa	30
4.9	Zdjęcie gotowego urządzenia od spodu	31
4.10	Zdjęcie gotowego urządzenia od góry	32
4.11	Zdjęcie gotowego urządzenia od boku	33
5.1	Diagram komunikacji urządzenia z komputerem	36

Spis tabel

2.1	Porównanie różnych modeli trackballi	10
5.1	Charakterystka kolejnych wersji interfejsu. Źródło: wikipedia	37
6.1	Test funkcjonalny: Sprawdzenie napięcia	53
6.2	Test funkcjonalny: Wykrywanie urządzenia	54
6.3	Test funkcjonalny: Sprawdzenie działania przycisków	55
6.4	Test funkcjonalny: Tryb przewijania	56
6.5	Test zgodności: Kompatybilność w systemach operacyjnych	57
6.6	Test wydajności: Test wytrzymałości	58

Dodatek A

Pełny kod źródłowy

```
1 #include <avr/io.h>
2 #include <avr/wdt.h>
3 #include <avr/interrupt.h>
4 #include <util/delay.h>
5
6 #include <avr/pgmspace.h>
7 #include "usbdrv.h"
8
9
10 #define BUTTONS ( (PIND & 0xe0) >> 5 )
11 #define TOGGLED ( (PIND & 0x10) >> 4 )
12
13 #define CR4      ( (PINC & 0x01)      )
14 #define CR2      ( (PINC & 0x02) >> 1 )
15 #define CR1      ( (PINC & 0x04) >> 2 )
16 #define CR3      ( (PINC & 0x08) >> 3 )
17
18 #define CONF0   ( (PIND & 0x01)      )
19 #define CONF1   ( (PIND & 0x02) >> 1 )
20 #define CONF2   ( (PINC & 0x10) >> 4 )
21 #define CONF3   ( (PINC & 0x20) >> 5 )
22 #define CONF4   ( (PINB & 0x01)      )
23 #define CONF5   ( (PINB & 0x02) >> 1 )
```

```

24 #define CONF6    ( (PINB & 0x04) >> 2 )
25 #define CONF7    ( (PINB & 0x08) >> 3 )
26
27
28 #define SCROLL_STEP_DIVIDER ( 2 << ( 1 << (1 << CONF3) | CONF2) )
29
30
31 #define abs(a) \
32     ((a > 0)? a: -a)
33
34 #define clamp(a, min, max) \
35     ((a > max)? max: (a < min)? min: a)
36
37 #define LINEARITY ( (1 << CONF3) | CONF2 )
38 #define INTERSECTION 0.5
39
40 #define accelerate(x, y) \
41     x = (1/(INTERSECTION+LINEARITY)) * x * (abs(x) + LINEARITY); \
42     y = (1/(INTERSECTION+LINEARITY)) * y * (abs(y) + LINEARITY)
43
44 #define direction(a, b, pa) \
45     ((a ^ b)? -1: 1) * ((a ^ pa)? 1: -1)
46
47
48 #define move(a, b, pa, pb, report_d, report_wheel, w_operation, \
49             counter, enable_button) \
50 \
51     if((a ^ pa) || (b ^ pb)) { \
52         if (TOGGLED) { \
53             if (scrolling_mode & enable_button \
54                 && abs(counter) == SCROLL_STEP_DIVIDER) \
55             { \
56                 report_wheel += \
57                     direction(a, b, pa); \
58                 change = 1; \
59                 counter = 0; \

```

```

60             } else { \
61                 counter += direction(a, b, pa); \
62             } \
63         } else { \
64             report_d_w_operation_direction(a, b, pa); \
65             change = 1; \
66         } \
67     }
68
69
70 PROGMEM const char usbHidReportDescriptor[61] = {
71     0x05, 0x01, // USAGE_PAGE (Generic Desktop)
72     0x09, 0x02, // USAGE (Mouse)
73     0xa1, 0x01, // COLLECTION (Application)
74
75     0x09, 0x01, //    USAGE (Pointer)
76     0xa1, 0x00, //    COLLECTION (Physical)
77
78     0x05, 0x09, //    USAGE_PAGE (Button)
79     0x19, 0x01, //    USAGE_MINIMUM (Button 1)
80     0x29, 0x03, //    USAGE_MAXIMUM (Button 3)
81     0x15, 0x00, //    LOGICAL_MINIMUM (0)
82     0x25, 0x01, //    LOGICAL_MAXIMUM (1)
83     0x75, 0x01, //    REPORT_SIZE (1)
84     0x95, 0x03, //    REPORT_COUNT (3)
85     0x81, 0x02, //    INPUT (Data, Var, Abs)
86     0x75, 0x05, //    REPORT_SIZE (5)
87     0x95, 0x01, //    REPORT_COUNT (1)
88     0x81, 0x03, //    INPUT (Const, Var, Abs)
89
90     0x05, 0x01, //    USAGE_PAGE (Generic Desktop)
91     0x09, 0x30, //    USAGE (X)
92     0x09, 0x31, //    USAGE (Y)
93     0x09, 0x38, //    USAGE (Wheel)
94     0x15, 0x81, //    LOGICAL_MINIMUM (-127)
95     0x25, 0x7f, //    LOGICAL_MAXIMUM (127)

```

```

96         0x75 , 0x08 , //      REPORT_SIZE (8)
97         0x95 , 0x03 , //      REPORT_COUNT (3)
98         0x81 , 0x06 , //      INPUT (Data, Var, Rel)
99
100        0x05 , 0x0c , //      USAGE_PAGE (Counsumer Devices)
101        0x0a , 0x38 , 0x02 , //  USAGE (AC Pan)
102        0x95 , 0x01 , //      REPORT_COUNT (1)
103        0x81 , 0x06 , //      INPUT (Data, Var, Rel)
104
105        0xc0 , //      END_COLLECTION
106
107        0xc0 , //      END_COLLECTION
108    };
109
110
111 typedef struct {
112     uchar buttons;
113     char dx;
114     char dy;
115     char dwheel;
116     char dhwheel;
117 } report_t;
118
119
120 static report_t report;
121 static uchar idleRate;
122
123 usbMsgLen_t usbFunctionSetup(uchar data[8])
124 {
125     usbRequest_t *rq = (void *)data;
126
127     if ((rq->bmRequestType & USBRQ_TYPE_MASK) == USBRQ_TYPE_CLASS) {
128         if (rq->bRequest == USBRQ_HID_GET_REPORT) {
129             usbMsgPtr = (void *)&report;
130             return sizeof(report);
131         } else if (rq->bRequest == USBRQ_HID_GET_IDLE) {

```

```

132                     usbMsgPtr = &idleRate;
133                     return 1;
134     } else if (rq->bRequest == USBRQ_HID_SET_IDLE) {
135             idleRate = rq->wValue.bytes[1];
136     }
137 } else {
138 }
139 return 0;
140 }
141
142
143 static volatile uchar key_press;
144
145 ISR(TIMER1_COMPA_vect)
146 {
147     static uchar key_state;
148     static uchar ct0, ct1;
149
150     uchar i = key_state ^ ~BUTTONS;
151
152     ct0 = ~(ct0 & i);
153     ct1 = (ct0 ^ ct1) & i;
154     i &= ct0 & ct1;
155     key_state ^= i;
156
157     key_press |= key_state & i;
158 }
159
160 #define DEBOUNCE 200L
161
162 void timer_init()
163 {
164     TIMSK = 1 << OCIE1A;
165     TCCR1B = (1 << CS10) | (1 << WGM12);
166     OCR1A = F_CPU / DEBOUNCE;
167 }
```

```

168
169 static void hardware_init(void)
170 {
171     wdt_enable(WDTO_1S);
172
173     DDRD = 0x00;
174     PORTD = 0xf3;
175
176     DDRC = 0x00;
177     PORTC = 0xff;
178
179     DDRB = 0x30;
180     PORTB = 0xcf;
181
182     usbInit();
183     usbDeviceDisconnect();
184
185     uchar i = 0;
186     while (--i) {
187         wdt_reset();
188         _delay_ms(1);
189     }
190
191     usbDeviceConnect();
192
193     sei();
194 }
195
196
197
198 static uchar v1, v2, v3, v4, p1, p2, p3, p4;
199 static uchar buttons_now, buttons_prev, scrolling_mode;
200 static uchar change;
201
202 static char v_counter = 0;
203 static char h_counter = 0;

```

204

```
205 int __attribute__(( noreturn )) main(void)
206 {
207     timer_init();
208     hardware_init();
209
210     change = 0;
211     buttons_prev = 0;
212     scrolling_mode = 0x04;
213
214     p1 = CR1;
215     p3 = CR3;
216     p2 = CR2;
217     p4 = CR4;
218
219     for (;;) {
220         wdt_reset();
221         usbPoll();
222
223         buttons_now = key_press;
224
225         v1 = CR1;
226         v3 = CR3;
227         v2 = CR2;
228         v4 = CR4;
229
230         if (buttons_prev != buttons_now) {
231             key_press = 0;
232             if(TOGGLED) {
233                 if(buttons_now & 0x07) {
234                     scrolling_mode = buttons_now;
235                     v_counter = h_counter = SCROLL_STEP_DIVIDER;
236                 }
237             } else {
238                 report.buttons = buttons_now;
239                 change = 1;
```

```

240         }
241     }
242
243     move(v1, v3, p1, p3, report.dx, report.dhwheel, +=, h_counter, 0x06);
244     move(v2, v4, p2, p4, report.dy, report.dvwheel, ==, v_counter, 0x05);
245
246     buttons_prev = buttons_now;
247
248     p1 = v1;
249     p2 = v2;
250     p3 = v3;
251     p4 = v4;
252
253     if (change && usbInterruptIsReady()) {
254         if (!CONF0) {
255             accelerate(report.dx, report.dy);
256         }
257
258         usbSetInterrupt((void *)&report, sizeof(report));
259
260         report.dx      = 0;
261         report.dy      = 0;
262         report.dvwheel = 0;
263         report.dhwheel = 0;
264
265         change = 0;
266     }
267 }
268 }
```

Dodatek B

Opis zawartości płyty CD

Do pracy powinna być dołączona płyta CD zawierająca pracę w formie pliku PDF, kod źródłowy trackballa oraz projekt płytka stworzony w programie Eagle.