odczyt danych z myszki w protokole komunikacyjnym ps2

Nikodem Szafran

Spis treści

[1. Cel projektu 1](#_Toc189355453)

[2. Wykorzystane zasoby 1](#_Toc189355454)

[3. Teoretyczny wstęp do projektu 2](#_Toc189355455)

[a. Budowa ramki danych 2](#_Toc189355456)

[b. Transmisja device-to-host 3](#_Toc189355457)

[c. Transmisja host-to-device 3](#_Toc189355458)

[4. Proces realizacji projektu 4](#_Toc189355459)

[5. Kod oraz pliki 5](#_Toc189355460)

[a. Header file „mousePacket.h” 5](#_Toc189355461)

[b. „main.c” oraz „main.h” 5](#_Toc189355462)

[c. Opis funkcji w „main.c” 5](#_Toc189355463)

[i. delay\_us(uint16\_t us) 5](#_Toc189355464)

[ii. send\_command(uint8\_t commandToSend) 6](#_Toc189355465)

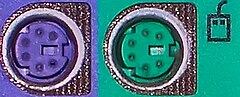
[iii. HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin) 6](#_Toc189355466)

[d. Program w języku Python „USART\_decypher” 6](#_Toc189355467)

[6. Pomocne źródła 6](#_Toc189355468)

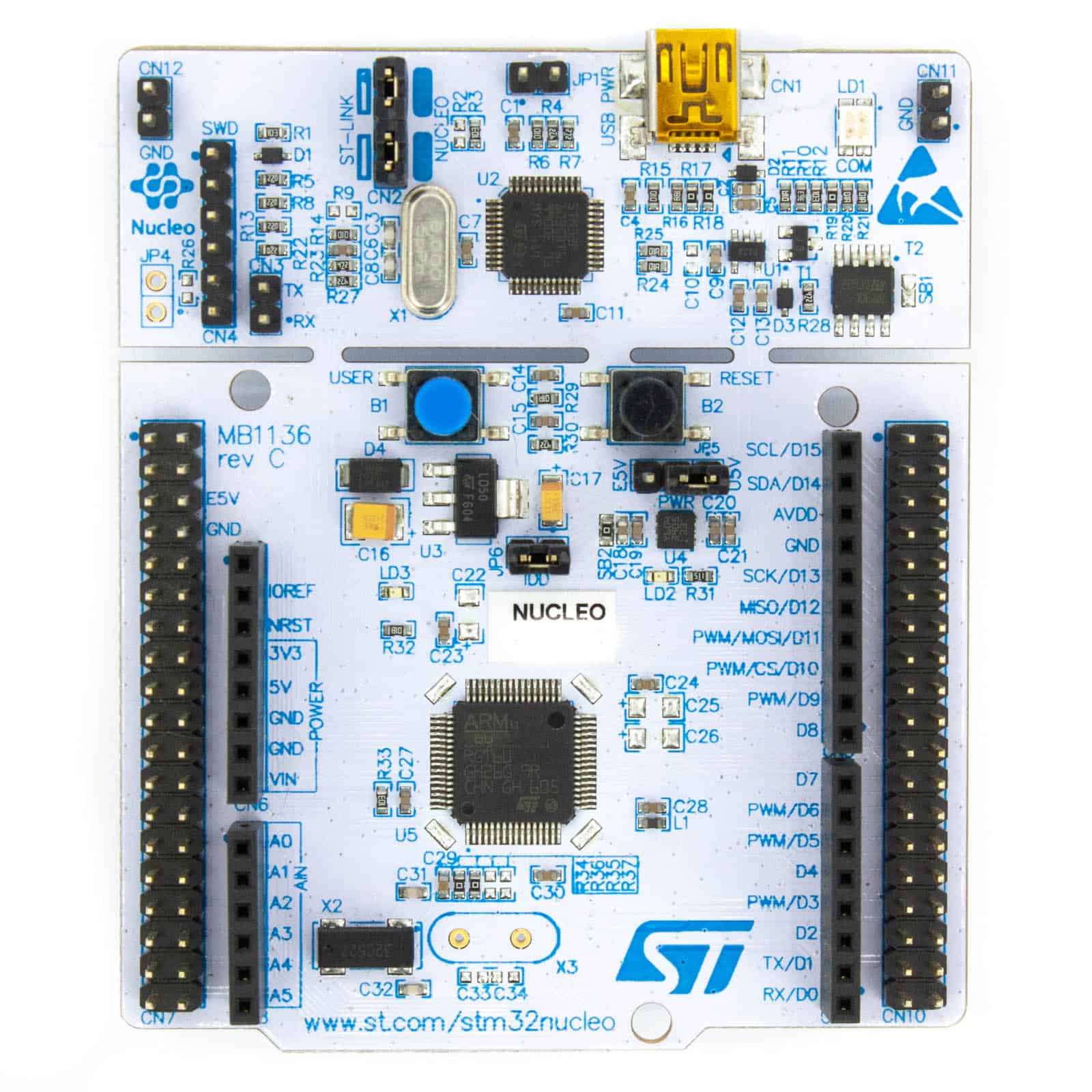
# Cel projektu

Celem projektu było odbieranie danych przesyłanych przez myszkę, z pomocą protokołu komunikacyjnego SP2.

Port komunikacyjny PS2 to złącze, które służy do podłączania urządzeń peryferyjnych do komputerów, w szczególności klawiatury i myszy. Jest to port okrągły, o średnicy 6 pinów, który był powszechnie stosowany w komputerach stacjonarnych do lat 2000-2010.

*Wygląd portu PS2, klawiatura (lewe wejście) i mysz (prawe wejście)*

# Wykorzystane zasoby

Do realizacji wykorzystano płytkę STM32 NUCLEO-L476RG. Wykorzystano także adapter od prowadzącego zajęcia, co umożliwiło łatwe i szybkie łączenie myszki z płytką. Konieczna oczywiście była także myszka wykorzystująca protokół PS2  


*Płytka L476RG*

Obraz zawierający elektronika, Komponent elektroniczny, Element obwodu, Inżynieria elektroniczna

Opis wygenerowany automatycznie

*Adapter PS2*

# Teoretyczny wstęp do projektu

Urządzenia w tym protokole poza VCC i GND, wykorzystują także dwie linie: DATA oraz CLOCK. Obie są fizycznie podłączone jako „otwarty kolektor”, zatem ważnym jest, by wejścia do których będą podłączone będą typu PULL-UP. Linią CLOCK przesyłany jest do hosta, generowany przez urządzenie, sygnał zegarowy. Drugą linią jak łatwo się domyślić – przenoszone są dane wysyłane przez hosta do urządzenia, lub dużo częściej – przez urządzenie do hosta.

## Budowa ramki danych

Każda ramka w komunikacji PS2 składa się z 11 bitów: pierwszy to start bit (1 bit), który jest zawsze ustawiony na 0 i informuje urządzenia, kiedy rozpoczyna się transmisja; następnie przesyłane są dane (8 bitów), zawierające rzeczywistą informację, np. kody klawiszy w przypadku klawiatury lub ruchy i przyciski myszy; potem mamy bit parzystości (1 bit), który sprawdza, czy liczba "1" w danych jest parzysta lub nieparzysta, co pozwala na wykrycie błędów w transmisji; na końcu znajduje się stop bit (1 bit), który jest zawsze ustawiony na 1 i oznacza zakończenie transmisji tej ramki.

Obraz zawierający tekst, Czcionka, zrzut ekranu, biały

Opis wygenerowany automatycznie

*Graficzne ukazanie transmisji (w tym przypadku dla hosta)*

## Transmisja device-to-host

Jest to przypadek dużo częstszy, bo i dużo częściej stosowany. Przechodząc od ogółu, do myszki, to wysyła ona 3 bajty danych (czasami 4, dla kółka na środku). Urządzenie przesyła do hosta sygnał zegarowy, a dane z linii DATA są odczytywane przez niego na opadających zboczach zegarowych z urządzenia.

Pierwszy bajt informuje o naciśniętych przyciskach, kierunku ruchu myszki w osi X (poziomej) oraz Y (pionowej), a także czy nastąpiło przepełnienie (do którego wrócimy zaraz).

Drugi bajt przesyła nam informację o wartości ruchu w osi X, jest on zapisywany jako signed int, czyli z rozróżnieniem znaku. Wartości są zatem w zakresie [-128, 127], gdzie minusowe to ruch w lewo, a dodatnie – w prawo. Bit przepełnienia w pierwszym bajcie jest ustawiony na wysoki, kiedy wartość ruchu przekroczy ten zakres.

Trzeci bajt informuje nas o wartości ruchu w osi Y i działa analogicznie do bajtu 2., ujemne wartości reprezentują ruch w dół, a dodatnie – w górę.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

*Opis bitowy bajtów danych*

## Transmisja host-to-device

W tym przypadku transmisja wygląda trochę inaczej – host najpierw musi ustawić stan linii CLK na stan niski, przynajmniej na 100 us. W momencie gdy ją „puści”, tzn. przestanie ustalać na stan niski, urządzenie zacznie generować znowu sygnał zegarowy i będzie odczytywało stan linii DATA, w momencie rosnącego zbocza zegarowego.

Tym samym jesteśmy w stanie do niego przesłać własną ramkę danych. Musimy jednak pamiętać, by przesłać ją razem z bitem startu, parzystości i stopu – tak samo jak urządzenie wysyła do hosta w poprzednim przypadku.

Obraz zawierający tekst, zrzut ekranu, linia, Czcionka

Opis wygenerowany automatycznie

*Możliwe komendy do wysłania do myszki*

# Proces realizacji projektu

Zgodnie z pierwotnym założeniem projektu, czyli odbieraniem danych, został napisany kod, który miał na celu realizację komunikacji device-to-host. Miał on odbierać paczki danych wysyłane przez myszki, procesować je w odpowiedni sposób, a następnie wysyłać do komputera przez USART.

Jednakże po napisaniu i sprawdzeniu poprawności kodu – komunikacja nie była realizowana. Myszka nie przesyłała żadnych danych. Po próbie komunikacji z inną myszką (również nie działała), a następnym podłączeniu ich do komputera i sprawdzenia czy działają (obie działały) – koniecznym stało się znalezienie rozwiązania tego problemu.

Odpowiedzią okazała się właśnie transmisja host-to-device. Po napisaniu kodu realizującego również ten typ komunikacji wysłano na myszkę komendy „0xFF – Reset” oraz „0xF4 – Data Enable” nasze urządzenie ożyło i zaczęło wysyłać dane, które zczytywano i przetwarzano na mikrokontrolerze, po czym wysyłano jako komunikat na USART.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*Wysyłana na USART ramka danych*

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

*Odbierane na porcie szeregowym przez komputer dane*

W kolejnym kroku napisano program w języku Python, który zajmował się odczytem tychże danych i graficzną reprezentacją w formie kursora ruszającego się po ekranie oraz kwadracikami symbolizującymi naciśnięcie przycisków, jak i pojawienie się przepełnienia w którejś z osi ruchu.  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

# Kod oraz pliki

## Header file „mousePacket.h”

Zawiera on strukturę danych MousePacket i jest niezbędny do działania programu, znajduje się w nim także bitowy opis bajtów danych przesyłany przez myszkę.

## „main.c” oraz „main.h”

Cały kod wraz z funkcjami znajduje się w tych dwóch plikach. Występują tam też zdefiniowane makra, które pozwalają użytkownikowi na lekką modyfikację działania kodu

## Opis funkcji w „main.c”

### delay\_us(uint16\_t us)

Funkcja generująca opóźnienie w mikrosekundach. Używa timera (htim17), by poczekać przez zadany czas (w mikrosekundach). Jest wykorzystywana do generowania wymaganych opóźnień, np. przy wysyłaniu danych do urządzenia PS2.

### send\_command(uint8\_t commandToSend)

Funkcja, która wysyła komendę do urządzenia PS2. Wysyłanie komend jest realizowane przez manipulację stanem linii danych i zegara. Najpierw ustawia stan niskiego zegara, następnie ustawia flagę informującą o tym, że host chce przesyłać komendy, wysyła bit startu. Reszta jest zrealizowana w funkcji poniżej, gdzie potem przesyła dane na linii danych, bit po bicie. Obsługuje również obliczanie bitu parzystości.

### HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

Funkcja wywoływana, gdy wystąpi przerwanie na pinie GPIO (w tym przypadku na pinie zegara PS2). Jej zadaniem jest odbieranie danych z myszki PS2. Na podstawie przerwania zegara, dane są odczytywane bit po bicie, a gdy pełny pakiet zostanie odebrany, ustawiany jest flag packet\_ready.

## Program w języku Python „USART\_decypher”

Jest to program realizujący graficzną reprezentację pokazaną wcześniej w dokumentacji.

# Pomocne źródła

<https://digilent.com/reference/pmod/pmodps2/reference-manual>  
<https://digilent.com/reference/_media/reference/pmod/pmodps2/pmodps2_sch.pdf>

<https://www.eecg.utoronto.ca/~jayar/ece241_08F/AudioVideoCores/ps2/ps2.html#mousedata>

https://www.burtonsys.com/ps2\_chapweske.htm