НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ
КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ АПАРАТУРИ

«До захисту допущено»
Завідувач кафедри
_____ Лисенко О.М.
(підпис) (ініціали, прізвище)
“11” _червня_ 2021р.

Дипломний проект

на здобуття ступеня бакалавра

зі спеціальності 172 "Телекомунікації та радіотехніка"
(код та назва напряму підготовки або спеціальності)

на тему Інтерактивний пристрій для введення інформації на базі мікроконтролера

Виконав: студент IV курсу, групи ДК-71

Романенко Святослав Володимирович ____ ____
(прізвище, ім’я, по батькові) (підпис)

Керівник ____ст. викладач Лисенко О.І._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент заст. директора НДІ ЕМСТ_Богдан О.В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____ _____
(підпис)

Київ - 2021 року

Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет електроніки

Кафедра конструювання електронно-обчислювальної апаратури

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 172 "Телекомунікації та радіотехніка"

(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Лисенко О.М.
(підпис) (прізвище ініціали)

«02» березня 2021р.

ЗАВДАННЯ
на дипломний проект студенту
Романенку Святославу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту Інтерактивний пристрій для введення інформації на базі мікроконтролера
керівник проекту Лисенко Олександр Іванович, старший викладач
затверджені наказом по університету від 24.05.2021 року №1316-с

2. Термін подання студентом проекту 8 червня 2021 року

3. Вихідні дані до проекту Інтерактивний пристрій для введення інформації на базі мікроконтролера. Пристрій для житлових приміщень. Являє собою друковану плату з роз'ємом для підключення до комп'ютеру. Кліматичне виконання УХЛ 4.2 по ГОСТ 15150-69. Габаритні розміри для друкованої плати – не більше 115х25 мм, маса – не більше 100 г. Час напрацювання на відмову – не менше 5 років.

4. Зміст розрахунково-пояснювальної записки:
   - огляд існуючих пристроїв для введення інформації;

- аналіз технічного завдання;
- розробка структурної схеми приладу;
- розробка схеми електричної принципової;
- вибір і обґрунтування електронно-компонентної бази;
- розміщення електронних компонентів на друкованій платі;
- конструкторсько – технологічний розрахунок друкованої плати;
- електричний розрахунок друкованої плати;
- розрахунок вібростійкості друкованої плати;
- розрахунок основних показників надійності;
- проектування друкованої плати у середовищі Altium Designer;
- проєктування програмного забезпечення;
- висновки.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень, плакатів, презентацій тощо):

- схема електрична принципова – А2;
- креслення друкованої плати – А1;
- складальне креслення друкованої плати – А1;
- алгоритм програми – А2;
- креслення корпусу – А1.

6. Дата видачі завдання 17.02.21

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Аналіз технічного завдання та патентний пошук	17.02.21 - 26.03.21	виконано
2	Схемотехнічне проєктування та вибір елементої бази	26.03.21 - 02.04.21	виконано
3	Виконання креслень схеми електричної принципової	02.04.21 - 09.04.21	виконано
4	Конструкторсько-технологічні розрахунки	09.04.21 - 23.04.21	виконано
5	Проєктування друкованого вузлу у серидовищі Altium Designer	23.04.21 - 30.04.21	виконано
6	Електричний розрахунок друкованої плати	30.04.21 - 03.05.21	виконано
7	Розрахунок надійності друкованого вузлу	03.05.21 - 07.05.21	виконано
8	Розрахунок вібромічності друкованої плати	07.05.21 - 14.05.21	виконано
9	Виконання креслень друкованої плати та складального креслення друкованого вузлу	14.05.21 - 21.05.21	виконано
10	Оформлення пояснювальної записки	21.05.21 - 28.05.21	виконано

Студент_____ Романенко С.В.
(підпис) (прізвище та ініціали)

Керівник проекту_____ Лисенко О.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломний проект викладено на 63 сторінках, містить 4 розділи, 28 ілюстрацій, 10 таблиць та 30 бібліографічних джерела в списку посилань.

Метою даної проекту є розробка інтерактивного пристрою для введення інформації на базі мікроконтролера для передачі інформайії, яка активується натисканням кнопки з мінімальними затримками та не потребує від користувача додаткових налаштувань.

У дипломному проекті було проведкно патентий пошук існуючих рішень щодо пристрою для введення інформації та проаналізовано її проектування друкованих плат. Розроблено схему електричну структурну пристроя, схему електричну принципову, виконано відбір елементної бази, а також проведено розрахунки схеми, що підтверджують правильний вибір елементної бази та працездатність пристрою. Виконано проектування друкованого вузла, проведені розрахунки, що підтверджують вірність принятих рішень під час розробки. Проведено алгоритм роботи програмного забезпечення для пристроя, блока передачі даних.

Також було розроблено комплект конструкторської документації для виготовлення пристрою.

ABSTRACT

The diploma paper consists of 65 pages. It contains four sections, 28 illustrations, 10 tables and 30 sources in the list of references.

The purpose of the paper was to develop an interactive device for inputting information, based on a microcontroller, which transmits information, entering with a pressed button, with minimal delays and does not require extra settings from a user.

In the present project, we investigated both a patent search of input devices and design of printed circuit boards. We developed two schemes of an input device: electrical structural and electrical principal. According to the scheme, we chose all components and proved by making calculations of the circuit. These results illustrates that our choice of components is correct and the device is functional. We also designed a printed circuit board and calculated it, proved the correctness of our decision, made during the research. The algorithm of the software of the device is given, as well the tribute transmission unit.

A set of design documentation for manufacturing has been made.

Пояснювальна записка
до дипломного проекту

на тему: **Інтерактивний пристрій для введення інформації на базі мікроконтролера**

Київ – 2021

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

ДП – Друкована плата

ДВ – Друкований вузол

КМ – Контактний майданчик

МО – Монтажний отвір

ТЗ – Технічне завдання

УГП – Умовно-графічне позначення

ПЗ – Програмне забезпечення

USB – Universal Serial Bus

BLE – Bluetooth Low Energy

ВСТУП

Сучасний ринок інтерактивних пристроїв для введення інформації набирає обертів. Вони має широке використання у різних сферах економіки, тому що призначаються не тільки ігрових систем, а й для електронних систем, що використовують персональні комп'ютери та смартфони. Однією з тенденцій у галузі електроніки є конструювання та проектування зручних, компактних інтерактивних, з урахування відношення "ціна – якість – надійність".

Предметом даного проекту є розробка електронного пристрою для введення інформації, а ***об'єктом*** розробки ігрового пристрою, а саме геймпад. Геймпад, як інтерактивний пристрій для введення інформації, характеризується способом підключення, сумісністю з іншими приладами та підтримкою певної технології передачі. Тому проект стосується розробки пристрою для введення інформації, який має дротове підключення, сумісність з комп'ютером і смартфоном та технологією передачі інформації DInput.

Таким чином, ***метою*** проекту є розробка друкованої плати та корпусу для геймпаду на базі аналогічного пристрою. Виходячи з поставленої мети, можна виділити наступні ***завдання***:

- розробити пристрій для введення інформації;
- розробити схему друкованої плати та корпусу для геймпаду;
- розробити програмне забезпечення до даного геймпаду.

В проекті, передусім, розглядено пристрої для введення інформації та проводити пошук аналогів, на базі якого розроблено схему структурну пристрою. Далі, розроблена схема електрична принципова та її основі створити друкований вузол, що реалізує дану схему. Разом з тим провести вибір елементної бази, типу матеріалу друкованої плати та відного класу точності виконання друкованої плати.

Необхідно провести все необхідні розрахунки, що підтверджують, коректність конструкторських та схемотехнічних рішень і дають підставу

вважати, що розроблений пристрій задовольняє усім вимогам, що зазначені у ТЗ.

Врешті описати технології передачі інформації на електронний пристрій. Також навести блок-схему та надати опис принципу роботи програми, завдяки якої працює даний пристрій. Наприкінці передбачити шляхи для модернізації та доробки програми пристрою.

Актуальність даного проекту зумовлено тим, що використання однієї друкованої плати замість двох надає змогу зекономити процес виробництва, комлектуючі та витрати. Завдяки цьому, кінцевий продукт може бути здешевлено, що дозволить конкурувати на ринку ігрових пристроїв.

Розділ 1. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ ТА ОГЛЯД АНАЛОГІВ

В даному розділі розглянуто існуючі пристрої для введення інформації до електронного пристрою (комп'ютери, смартфони, тощо). Передусім перевагу надано тим, що найбільш задовольняють вимогам, що визначені у технічному завданні.

Пристрій для введення інформації в цілому є частиною повного продукту, однак задачею даного проекту є розробка продукту, який дозволить заносити дані до комп'ютера. Зважаючи на сучасні тренди, людині буде більш зручніше використовувати комп'ютер або смартфон. Для реалізації даного пристрою необхідно вирішити слідуючі задачі:

- обрати пристрій для введення інформації;
- створити схему пристрою, що буде працювати з комп'ютером або смартфоном, а також відображати внесену інформацію на екрані комп'ютера або смартфона;
- розробити програмну реалізацію даного пристрою

1.1 Вибір пристрою для введення інформації

Сучасний світ електронних товарів пропонує різноманітний та широкий вибір пристроїв для введення інформації. Для обрання пристрою необхідно проаналізувати їх класифікацию. Для цього скористаємося спрощенною класифікацією на 3 типа пристроів. Розглянємо класифікація пристроів для введення інформації, яка показана на Рис. 1.1. [1].

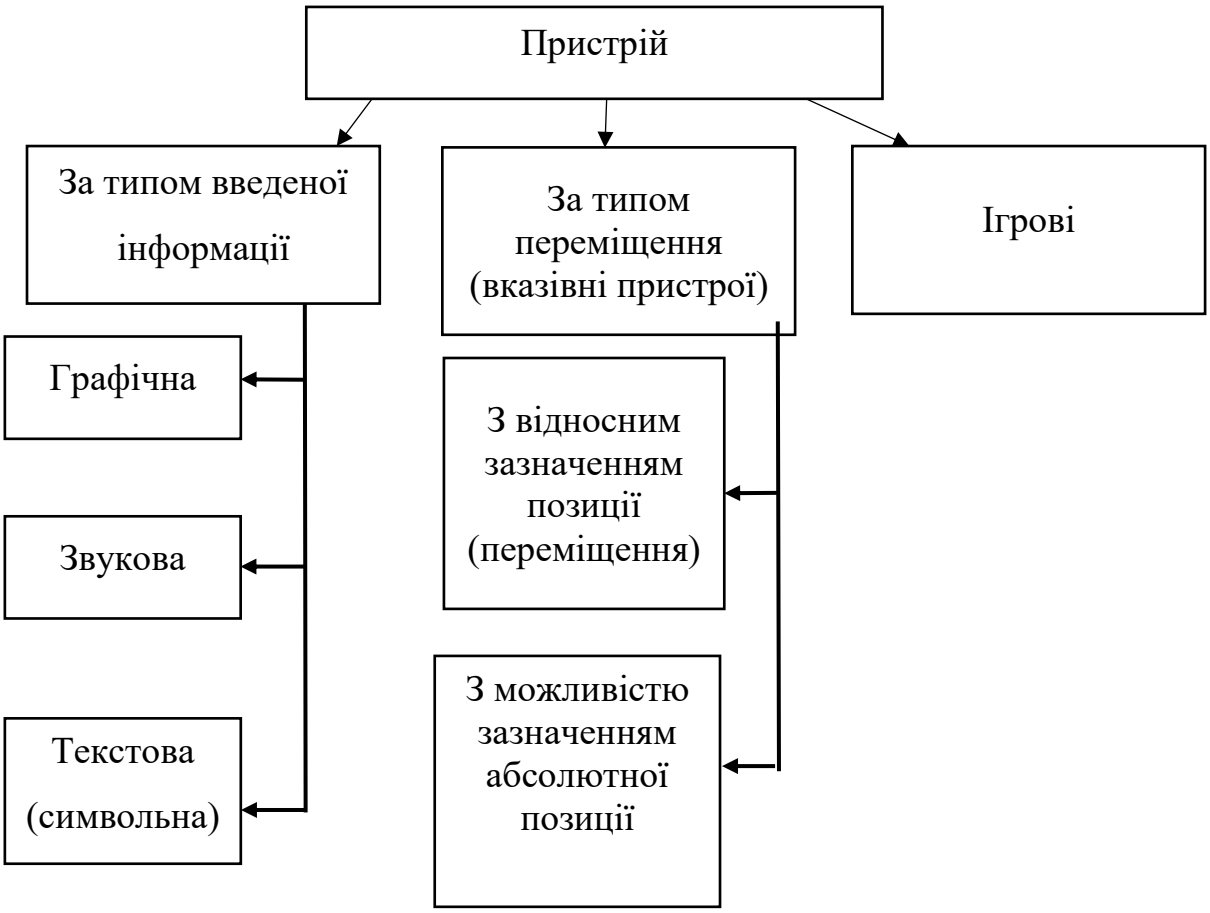


Рис. 1.1 - Класифікація пристроїв для введення інформації

За наведеною на Рис 1.1 класифікації, розглянемо пристрої детальніше.

Пристроями для введення графічної інформації називають, ті пристрої які після отримання інформації перетворюють її в цифровий код, що складається з ліній та символів. [2] До таких пристроїв належать сканер, цифровий фото апарат та відеокамера.

Аналогічно до пристроїв для введення графічної інформації працюють й пристрої для введення звукової та текстової інформації, а є й відмінність, яка полягає в тому, що при передачі звукової інформації буде перетворюватись аналоговий сигнал в цифровий, до цих пристроїв належать мікрофон та диктофон.

Пристрої для введення текстової інформації є одним з найважливіших пристроїв для введення інформації без якого важко уявити наше існування, адже людство завжди намагалось автоматизувати введення тексту. До появи

комп’ютерів єдиним пристроєм для введення інформації були друкувальні машини. Пристрої даного типу працюють за тим же принципом що й пристрої для введення графічної та звукової інформації. До таких пристроїв належить клавіатура. На рис. 1.2 зображено пристрої за типом введеної інформації [3].

Рис. 1.2 Приклад пристроїв для введення інформації

Пристрої за типом введення інформації мають як і свої плюси, так і мінуси, проте на ринку таких приладів дуже велика конкуренція, адже на цьому ринку пристроїв для введення текстової інформації дуже багато неякісних підробок, що ледве витримують гарантійний термін.

Пристрої за типом переміщення розрізняють на: пристрої з відносним визначенням позиції та на пристрої з можливістю визначення абсолютної позиції. Насамперед пристроями цього типу ми користуємось постійно. Розглянемо ж ці пристрої детальніше.

Пристрої з відносним визначенням позиції працюють за таким принципом: відносне визначення позиції задається переміщенням пристрою в області робочої площини дає певну інформацію, яка передається комп’ютеру. Також дані пристрої мають кнопки, які виступають в якості засобу керування курсором. До таких пристроїв належать комп’ютерна миш, тачпад (анг. Touchpad або ж сенсорна панель) та трекбол (анг. Trackball), під трекболом

розуміється пластмасова куля, що обертається в будь-якому напрямку. [3] На відміну від пристроїв з відносним визначенням позиції, пристрої з можливістю визначення абсолютної позиції не потрібно переміщати сам пристрій, щоб отримати інформацію, в якості об’єкта переміщення може слугувати такий пристрій як стилус (анг. Stylus), що являє собою пластмасовий олівець або ж ручку, який призначений до таких пристроїв як графічний планшет[4].

Пристрої за типом переміщення мають, як і пристрої за типом введення даний, великий попит. Основною перевагою пристроїв з відносним визначенням позиції є те, що вони більш-менш дешеві з погляду виготовлення даних пристроїв, проте варто пам’ятати, що для пристроїв з відносним визначенням позиції - це перевага, але для пристроїв з можливістю визначення абсолютної позиції - це один з основних недоліків.

Ігрові пристрої, насамперед, говорять самі за себе та являються компонентом ігрових систем. Даний тип пристроїв має на ринку розваг дуже великий попит в усьому світі. До таких пристроїв належать ігрові джойстики, геймпади та ін. Під “*геймпадом*” розуміють ігровий пульт, який складається з аналогових джойстиків та кнопок керування, що скомпоновані в герметизованому корпусі.[4] Даний пристрій зображено на Рис. 1.3

Рис. 1.3 Геймпад Xbox360 компанії Microsoft

В якості пристрою для введення даних будемо реалізовувати розробку геймпаду, тому що на сучасному ринку розваг з’явилась новітня течія, яка являє собою ностальгію за минулим та має великий попит серед багатьох людей. Ще однією з причин є простота розробки пристрою.

1.2 Огляд існуючих пристроїв для введення інформації.

В даному пункті були розглянуті аналоги схожих пристроїв для введення інформації. В цілому всі вони використовують один спосіб виготовлення друкованої плати, як наприклад в пристрої [5], тому така розробка є в якомусь плані унікальною.

Отже, розглянемо друковану плату для введення інформації. В даному пристрої до готової плати підключається пристрій, що показаний на Рис 1.4.

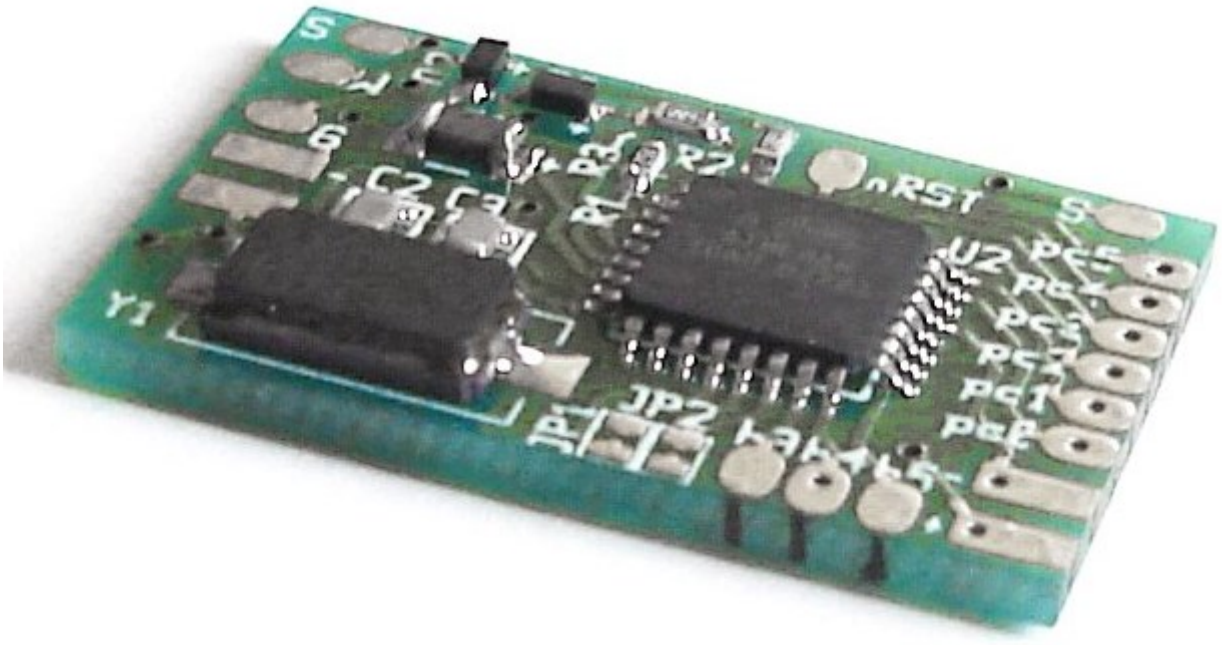


Рис 1.4 – Реалізований пристрій авторством *raphnet*

Перевірка на працездатність даного пристрою проводиться комп’ютером. Загалом, даний пристрій має схожість за ідеєю з даним проектом, за винятком друкованої плати, додаванням одного елементу керування та самого цільового приладу.

Обраний мікроконтролер *AtMega8A-AU* (підрозд. 2.2.1) має гарне співвідношення вартості, споживаної напруги та обчислювальних здатностей. Це дозволяє провести модифікації системи.

Висновок до розділу 1:

В даному розділі проведено аналіз існуючих електронних пристроїв для введення інформації та виконано пошук аналогів за обраною тематикою. При цьому було проаналізовано всі типи пристроїв для введення інформації, а також було встановлено, що серед усіх типів пристроїв для введення інформації. Отже, ***предметом*** даного проекту є аналіз існуючих електронних пристроїв для введення інформації, а ***об'єктом*** розробки буде ігровий пристрій, а саме геймпад.

Здійснивши аналіз технічного завдання, дає змогу дійти висновку, що пристрій має передавати дані на комп'ютер при натисканні щонайменше однієї кнопки. І тому даний пристрій буде працювати завдяки інтерфейсу USB.

Актуальність даного проекту зумовлено тим, що використання однієї друкованої плати замість двох надає змогу зекономити процес виробництва, комплектуючі та витрати. Завдяки цьому, кінцевий продукт може бути здешевлено, що дозволить конкурувати на ринку ігрових пристроїв.

Розділ 2. СХЕМОТЕХНІЧНЕ ПРОЕКТУВАННЯ ТА ПЕРЕВІРКА СХЕМОТЕХНІЧНИЙ РІШЕНЬ.

2.1 Розробка схеми електричної структурної

Спочатку розглянемо загальну структурну схему пристрою для введення інформації, яка проілюстрована на рис 2.1.

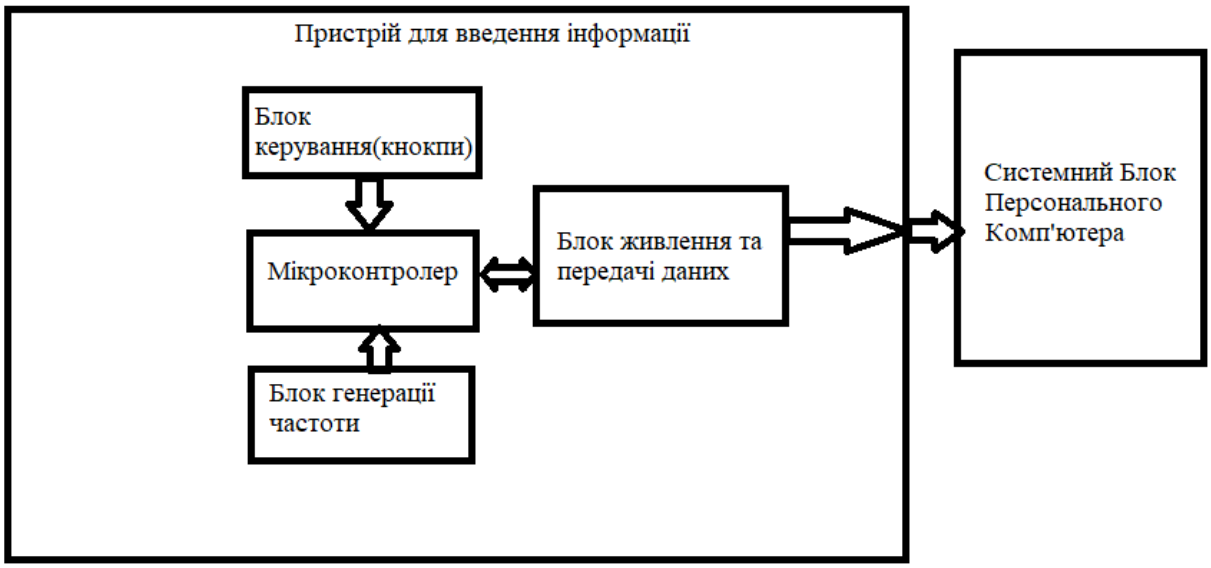


Рис 2.1 Схема електрична структурна

Перш за все, з наведеною у рис 2.1 структурою, розглянемо детальніше кожний з блоків та визначимо його головні частини та задачі, на які він розрахований.

Безпосередньо пристрій призначений для введення інформації, яка вводиться за допомогою кнопок, під'єднаних до мікроконтролера. З’єднання з пристроєм відбувається за допомогою інтерфейсу USB (Див підрозд. 4.1), завдяки мікроконтролеру. Мікроконтролер отримує дані з блоку керування й, на базі отриманих даних, передає блоку передачі даних. Сам же блок передачі даних при'єднаний до блоку живлення за допомогою інтерфейсу USB.

При цьому USB інтерфейс працює при напрузі 5 В, хоча сам блок має стабілізатори напруги через те, що обраний мікроконтролер (Див. підрозд. 2.2) не підтримує дану напругу живлення. Напруга подається з будь – якого електронного пристрою (комп’ютер, смартфон, тощо).

Для реалізації блоку генерації частоти дуже часто використовують RC-ланцюжок, задаючи частоту програмно, або ж з використанням кварцевого резонатору на певній частоті.

2.2 Вибір елементної бази

Найбільш важливі компоненти були обрані за допомогою матричного методу. Для найменш важливих компонентів вибір обгрунтовано методом порівняння компонентів.

Пристрій, що розроблюється має складатися з необхідних елементів, що подано нижче:

- Мікроконтролер
- Кварцовий резонатор
- Стабілітрони
- Резистори
- Конденсатори
- Кнопки

2.2.1 Вибір мікроконтролера

Мікроконтролер - є найбільш важливим з основних компонентів пристрою. До мікроконтролера виставленні наступні вимоги:

- підтримка (8-16) МГц частот, є одним з ключових факторів, адже якщо мікроконтролер буде підтримувати частоти, що є менше необхідних, то пристрій не буде працювати;
- споживання напруги є більш критичним фактором, бо при подачі більш напруги 5+ В мікроконтролер виходить з ладу;
- вартість та простота в отриманні мікросхем на території України;
- стандартизована документація на мікроконтролер, висока швидкість розробки ПЗ.

Насамперед, для початку варто визначитись з сімейством мікроконтролерів. Даний вибір є найпростішим. Варто лише обрати знайому розробнику архітектуру, завдяки якої, можливо досягти максимальну

швидкість у розробці. Тут лідерами є сімейства мікроконтролерів *STM32*, на базі архітектури *ARM Cortex* та *AtMega8*, на основі архітектури *RISC*, з якими в мене є необхідний досвід роботи для розробки.

Ще однією з переваг обох сімейств мікроконтролерів є різноманітна кількість навчальних матеріалів у вільному доступі, також їх низька вартість та доступність на території України.

Обидва сімейства поєднують в собі чималу кількість архітектур. При цьому, кожне сімейство поділяють на безліч серій, де основним критерієм є відповідність до їх призначення, а саме: з низький енергоспоживанням, з необхідною специфічною периферією, або просто мікроконтролери загального призначення.

Під всі вимоги до даної розробки найбільш всього підходить мікроконтролер *AtMega8A-AU*. [6] Даний мікроконтролер зображений на рис 2.2.

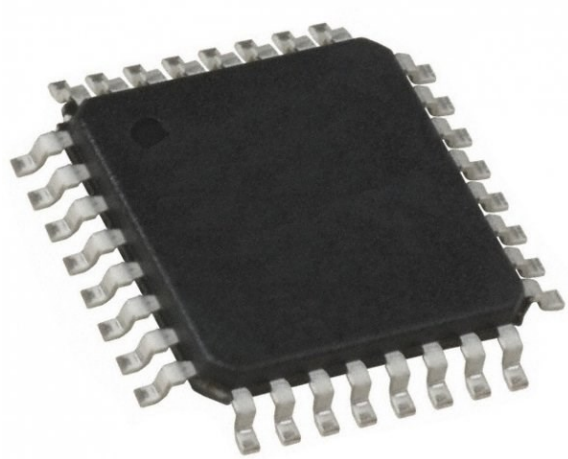


Рис 2.2 Мікроконтролер AtMega8A-AU у TQFP корпусі

Його частотний діапазон складає (1 – 16) МГц, чого достатньо для того, щоб, по-перше, забезпечити мінімальні затримки. По-друге, дозволить встановити необхідний для пристрою кварцевий резонатор. До того ж споживання напруги в діапазоні від 2,7 В до 5.5 В дозволить обрати стабілізатори напруги.

2.2.2 Вибір кварцового резонатору

Кварцовий резонатор – є одним з найважливіших компонентів. Він повинен задовольняти таким вимогам:

- частота повинна бути не більше максимальної, що підтримує мікроконтролер;
- малі масо-габаритні розміри;
- низька собівартість;
- доступність в Україні.

Оскільки однією з вимог при виборі мікроконтролера було певний частотний діапазон, а він з обраним мікроконтролером складає (1-16) МГц, то й частота кварцового резонатору повинна бути в цьому діапазоні.

Вибір резонатору проведено використовуючи матричний метод. Були розглянуті такі резонатори:

- SIWARD LP-3.5S [7]
- Raltron AS-10.000-18-EXT-SMD [8]
- RoHS ABLS-12000MHZ-B2-T [9]

Вище зазначені резонатори знаходяться в діапазоні від 8МГц до 12 МГц, тобто можна використовувати в блоці генерації частоти.

Резонатори порівнювались за такими параметрами як:

- частота
- ємність навантаження
- вартість.

Кожний параметр повинен мати свій ваговий коефіцієнт *bj*, в залежності від його важливості для роботи пристрою. Значення та вагові коефіцієнти відповідних параметрів до обраних кварцових резонаторів зведені у табл. 2.1.

Табл. 2.1 – Матриця параметрів X

Кварцові резонатори	Параметри		
	Частота, МГц	Ємність навантаження, пФ	Вартість, Грн
LP-3.5S	8	16	14,5
AS-10.000-18-EXT-SMD	10	18	14,5
ABLS-12.000MHZ-B2-T	12	18	7
Ваговий коеф, bj	0,35	0.1	0,55

На основі матриці *X*, створюємо матрицю приведених параметрів *Y*. Якщо більшому значенню параметра відповідає менша якість, то тоді його перераховують за формулою:

$$Y_{i_j} = \frac{1}{X_{i_j}} \qquad (2.1)$$

де $i = 1,n$ – кількість обраних компонентів;

$i = 1,n$ – кількість параметрів компонентів, що аналізуються.

Таким чином, було отримано матрицю приведених параметрів *Y*, табл. 2.2.

Табл. 2.2 – Матриця параметрів Y

LP-3.5S	8	16	0,07
AS-10.000-18-EXT-SMD	10	18	0,07
ABLS-12.000MHZ-B2-T	12	18	0,14
Ваговий коеф, bj	0,35	0,1	0,55

На основі матриці *Y*, створимо матрицю нормованих параметрів *A* за формулою:

$$A_{ij} = \frac{maxY_{ij} - Y_{ij}}{maxY_{ij}} \quad (2.2)$$

де $maxY_{ij}$ – максимальний елемент у стовпчику;

Y_{ij} – значення елементу, який нормується.

Отже, отримкємо матрицю, що показана у табл. 2.3

Табл. 2.3 – Матриця нормованих параметрів *A*

LP-3.5S	0,5	0,2	0,61
AS-10.000-18-EXT-SMD	0,6	0	0,61
ABLS-12.000MHZ-B2-T	0	0	0
Ваговий коеф, bj	0,35	0,1	0,55

Табл. 2.4 – Значення оціночної функції *Q*

Кварцовий резонатор	Q
LP-3.5S	0,53
AS-10.000-18-EXT-SMD	0,56
ABLS-12.000MHZ-B2-T	0

Як видно, ABLS-12.000MHZ-B2-T має краще значення, ніж у інших. Вибір даного резонатору обгрунтований тим, що цей кварцовий резонатор є найбільш доступним на території України та наявність приблизного значення ємностей конденсаторів, які слугуватимуть обвісткою для кварцу.

2.2.3 Вибір стабілітрона

Як було зазначено у підрозд. 2.1, в блоці живлення буде присутні стабілітрони, розраховані на $U_{стаб} = 3,6$, де $U_{стаб}$ – напруга стабілізації. Серед популярних стабілітронів є *1N4729A*, але він не підходить через те що він в *DIP* виконанні. Тому, доберемо схожий за характеристиками стабілітрон у SMD виконанні – *BZV55C3V6* від виробника *NXP*[10].

2.2.4 Вибір резисторів та конденсаторів

В даному приладі резистори потрібні тільки для підключення USB інтерфейсу до мікроконтролеру та для підтяжки сигнальних ліній до землі або до живлення для того, щоб запобігти короткому замиканню, тому потужність резисторів 0,5 Вт буде достатньою. Така потужність може розсіюватись при використанні корпусу *SMD 0805*. Для виконання проекту були обрані резистори компанії *Yageo* з точністю 5% з номіналом 67 Ом.

Конденсатори у приладі виконують захисні функції як від появи НЗЗ, так від паразитних параметрів. Конденсатори, які будуть працювати в зв'язці з кварцовим резонатором візьмемо у корпусі *SMD 0805* з точністю 5%, а ті, які будуть захищати від появи НЗЗ візьмемо у корпусі *SMD 0805* з точністю 10%.

2.2.5 Вибір кнопок.

Кнопки в даному пристрої відіграють ключову роль, а саме з них вводиться інформація. Тому кнопки будуть вибрані у двох реалізаціях.

Перша реалізація буде слугувати, як бічні кнопки (по одній з кожного боку), тому їх візьмемо у DIP виконанні кутових кнопок[11]. Друга реалізація буде основною, тому вона буде виконана у вигляді звичайного металізованого контакту, через те, що ці кнопки будуть передавати інформацію до мікроконтролера та до того ж дана реалізація прослугує на багато більше часу, ніж використовувати звичайні кнопки. Кутові кнопки зображені на рис 2. 3

Рис 2.3 Кутова кнопка у DIP виконанні

2.3 Розробка схеми електричного принципової

При розробці схеми електричної принципової приділимо необхідну увагу до таких блоків:

- Блок живлення та передачі даних
- Блок генерації частоти

- Блок керування

2.3.1 Принцип роботи блоку живлення та передачі даних

Блок живлення та передачі даних зображено на рис. 2.4

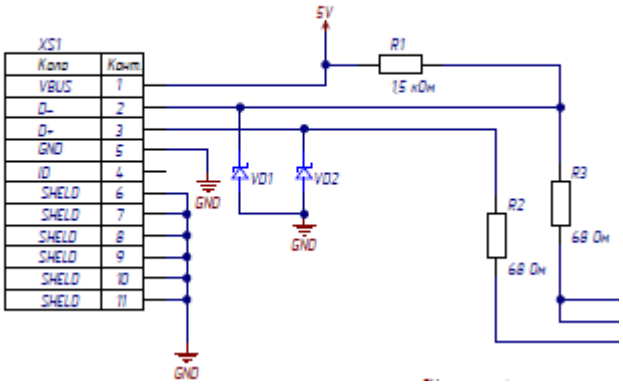


Рис. 2.4 Схема блоку живлення та передачі даних

Як вище було зазначено, блок живлення та передачі даних складається зі USB, стабілітронів, та резисторів, що виступають в ролі баласту для стабілітрона. Тому спочатку розглянемо принцип роботи даного блоку. На USB подається напруга живлення з електронного пристрою (комп'ютер, смартфон, тощо) та становить 5В, подається з входу VBUS. Входи D+ та D- слугують в якості передавача даних, що отримуються з мікроконтролера, до них вмикаємо резистори для того, щоб мікроконтролер не вийшов з ладу. Також на дані входи вмикаємо стабілітрони, що забезпечують свого роду безпеку, адже без нього може відбутись скачок напруги, який може вивести мікроконтролер з ладу, а з ним ми зменшуємо напругу живлення до 3,6 В. Так як зазначено в підрозд. 2.2.4 зазначено опір резистора, тому треба визначити прибрлизний струм, що тече через резистор на мікроконтролер. Тому необхідно знати приблизне значення струму. Опір резистора складає 67 Ом. Струм стабілізації розраховується за формулою[12]:

$$I_{ст} = \frac{U_{жив} - U_{ст}}{R} = \frac{5 - 3{,}6}{67} = 21\ (мА) \qquad (2.3)$$

Отже, очевидно, що при обраному стабілітроні та резисторі струм стабілізації становить 21 мА.

2.3.2 Принцип роботи блоків генерації частоти, керування та мікроконтролера.

Передусім, розглянемо блок генерації частоти, який зображено на рис. 2.5

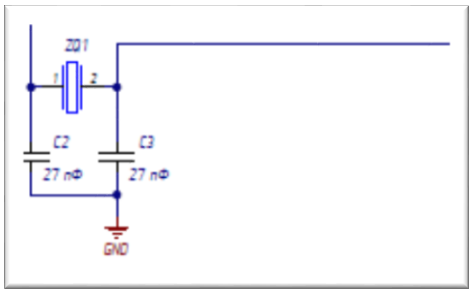


Рис. 2.5 Схема блоку генерації частоти

Даний блок підтримує частоту на мікроконтролері, завдяки кварцовому резонатору та його об’вязці. Насамперед, ємність конденсатора напряму залежить від таких факторів, як ємності навантаження резонатору та ємності друкованого монтажу. Ємність навантаження резонатора за даними даташиту (англ. DataSheet) [7] слугує 18 пФ, а ємність друкованого монтажу слід вибирати від 3 пФ до 5 пФ. Ємність конденсаторів при паралельному з'єднані розраховують за такою формулою[13]:

$$C_1 = C_2 = (C_н - C_д) * 2 = (18 - 4) * 2 = 28 \text{ пФ} \qquad (2.4)$$

де $C_н$ – ємність навантаження резонатора

$C_д$ – ємність друкованого монтажу

Тому, конденсатор при ємності 28 пФ може слугувати обв’язкою для кварцового резонатору, але при відсутності конденсатора з такою ємністю можна використовувати конденсатор, що має трохи меншу ємність.

Блок керування передає введену інформацію мікроконтролеру, порти якого було запрограмовано на введення. Далі, мікроконтролер при частоті, що підтримує блок генерації частоти, передає інформацію блоку живлення та передачі даних і, таким чином, вона передається на електронній пристрій. Даний блок зображено на рис. 2.6

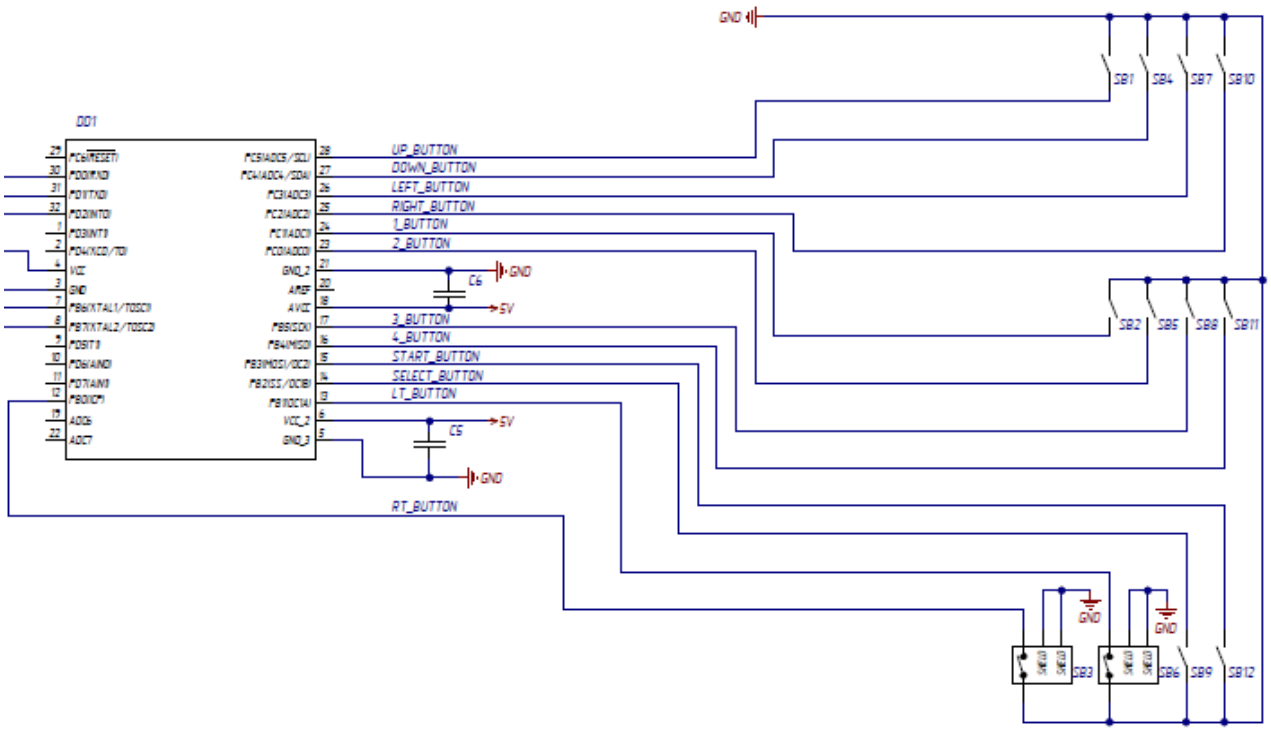


Рис. 2.6 Схема блоку управління та мікроконтролера

Висновки до розділу 2

В даному розділі було проаналізовано елементну базу, на основі цього були підібрані компоненти, що найкраще підходять для використання при розробці проекту, а саме:

- мікроконтролер *AtMega8A-AU*
- кварцовий резонатор ABLS-12.000MHZ-B2-T
- стабілітрон у SMD виконанні – *BZV55C3V6* від виробника *NXP*
- резистори у корпусі *SMD 0805* компанії *Yageo* з точністю 5%
- конденсатори у корпусі *SMD 0805* з точністю 5% та 10%
- два види кнопок: DIP виконанні кутові та у вигляді звичайного металізованого контакту.

Даний пристрій дозволяє вносити інформацію при напрузі живлення, що складає 5 В. Обраний мікроконтролер має переваги: з одного боку, за необхідністю він може заблокувати декілька кнопок, а з іншого – дозволяє додати аналоговий джойстик.

Також велику увагу було приділено до документації на компоненти, що дозволило застосувати їх для вирішення конкретних задач даного проекту.

Особливу увагу було приділено розробці схеми електричної принципової, яка була поділена на три блоки. Далі, кожен з них був проаналізован детально та визначени принципи їх роботи.

Розділ 3. КОНСТРУКТОРСЬКО – ТЕХНОЛОГІЧНИЙ РОЗРАХУНОК ДРУКОВАНОЇ ПЛАТИ

3.1 Вибір та обґрунтування типу друкованої плати

Друковані плати застосовують для реалізації електричних з'єднань між компонентами та має в основі діелектрик на одній стороні або на обох сторонах, якої розташовані конструктивні елементи та елементи друкованого монтажу, а саме: друковані провідники, контактні майданчики, металізовані монтажні отвори.

Друковані плати поділяють за конструкцією на наступні класи:

– одношарові друковані плати, надалі ОДП;

– двошарові друковані плати, надалі ДДП;

– багатошарові друковані плати, надалі БДП.

У випадку ОДП всі електронні компоненти знаходяться на шарі дієлектричної основи, а на іншій стороні розташовані елементи друкованого монтажу, які реалізують електричне з'єднання між компонентами. Даний тип плати є простішим у проєктуванні, проте їх монтажні і трасувальні можливості є найнижчими. Електрична надійність і механічна міцність кріплення компонентів у таких плат також є низькою. Тому ОДП, в основному, використовують лише у найпростіших пристроїв, до яких не має особливих вимог щодо їх масо-габаритних розмірів та електричних параметрів.

ДДП мають друковані провідники з обох сторін діелектричної основи. Електронні компоненти також можуть вмонтовані з обох сторін. Вони мають високу щільність монтажу та підвищену надійність з'єднань, що сприяє підвщенню щільності монтажу та великих можливостях при трасуванні з'єднань, які дозволяють значно зменшити масо-габаритні розміри друкованої плати. Вони мають високу електричну надійність та механічну міцність з'єднань на кріплення компонентів. Через сукупність цих параметрів вони набули широке розповсюдження на виробництвах, що зосереджене на створенні електорнної апаратури.

БДП складаються з парної кількості шарів діелектрика та провідного металевого шару, що чергуються між собою, утворюючи багатошарову структуру. З одного боку, даний тип плат є надто дорогим тому, що залежить від кількості шарів друкованої плати, яку проектувати дуже складно. З іншого боку, БДП дозволяють проєктувати та розробляти складні у виконанні електричні пристрої, що дає інженеру великі можливості у трасуванні з'єднань між провідниками.

Отже, для виготовлення пристрою, згідно технічного завдання, обрано ДДП. У зв'язку із кількістю компонентів, їх масо-габаритними розмірами та з огляду на складність зв'язків між компонентами, провести трасування ДВ використовуючи ОДП, при цьому забезпечивши компактні розміри пристрою та високу надійність електричних з'єднань не є можливим. Використання БДП також не є оптимальним, бо це призведе до значного збільшення складності трасування, виробництва та верифікації готового пристрою.

3.2 Вибір та обґрунтування матеріалу друкованої плати.

У якості матеріалу основи друкованої плати було обрано склотекстоліт типу FR4-2-35-1.5. Даний матеріал є більш поширеним з-поміж матеріалів, із яких виготовляються основи для друкованих плат. З фізичної точки зору, він являє собою декілька шарів скловолокна, просочених епоксидною смолою. Він має високі діелектричні показники та високу температурну стабільність електричних та габаритних характеристик. Це дозволить отримати низькі значення паразитних ємностей та струмів втрат, якими можна нехтувати при проектуванні електричних пристроїв. Великою з переваг цього матеріалу є його абсолютна негорючість. Склотекстоліт являє собою багато шарів склеєної тканини, він без проблем витримує вібрації та невеликі деформації, а ще має дуже високий рівень адгезії із клеєм, яким закріплюють шар мідної фольги.

Обраний матеріал має товщину 1.5 мм, а товщина шару фольги складає 35 мкм.

У табл. 3.1 приведені порівняльні характеристики матеріалів гетинакс (ГОСТ 2718 – 74), текстоліт (ГОСТ 2910-74) і склотекстоліт (ГОСТ 12652-74)
Табл. 3.1 – Основні характеристки матеріалів друкованої плати.

Властивості	Гетинакс	Текстоліт	Склотекстоліт
Густина, г/см^3	1,3…1,4	1,3…1,4	1,6…2,05
Теплостійкість по Мартенсу ^{o}C, не менше	150	125	180
Холодостійкість, ^{o}C	-60	-60	-60
Границя міцності, МПа: при розтягу на згин	80…100 130…150	50…65 90…120	120…180 200
Питома ударна в'язкість, кДж/м^2	15…20	20…27	75
Питомий об'ємний опір, Ом*м	10^8 …10^{11}	10^6…10^8	10^{11}
Діелектрична проникність	6…7	8	4…7
Тангенс кута діелектричних втрат при 10^6Гц	0,035…0,08	0,07	0,02
Електрична міцність (перпендикулярно шарам), МВ/м, не менше	33	4…8	18

3.3 Вибір методу виготовлення друкованої плати

Метод виготовлення друкованої плати визначає яким чином будуть видалятись зайві ділянки міді з поверхні діелектрику. Будуть розглянуті наступні методи: комбінований позитивний та негативний методи.

Насамперед негативний метод виконують на ранніх стадіях виготовлення, бо діелектричну основу піддають на тривалу обробку розчинниками, що погіршують зчеплення з міддю.

Комбінований позитивний метод є сучасним методом виготовлення друкованої плати та має декілька переваг, таких як: зменшення часу дії електролітів на діелектрик, що збільшує зчеплення з міддю, можливість виготовлення плат з великою точністю, простота виготовлення перехідних отворів, що має сенс для ДДП, що використовуємо.

Комбіновані методи виготовляються ОДП та ДДП з фольгованого діелектрика витравлюванням рисунку провідників хімічним методом з металізацією монтажних отворів КМ, провідників хімічним методом.

На основі описаних вище методів було вирішено використовувати комбінований позитивний метод для створення друкованої плати.

3.4 Вибір та обґрунтування класу точності друкованої плати

Однією з важливих вимог до розроблюваного пристрою є габарити друкованої плати, які дозволять розташувати її в корпусі. При цьому не можна допустити щоб зменшення габаритних розмірів відбувалося за рахунок зменшення якості трасування. Особливу увагу необхідно звернути на якість трасування ланцюгів живлення, контактних кнопок та землі.

Граничні значення основний параметрів друкованого монтажу наведені у табл. 3.2. Також необхідно враховувати допустимі похибки виконання елементів друкованого монтажу, що можуть бути допущені на виробництві, наведені у табл. 3.3. На ці параметри варто орієнтуватися при виробі класу точності поверхневого монтажу ДП.

Табл. 3.2 – Граничні значення основних параметрів друкованого монтажу

Параметр	Позначення	Клас точності			
		2	3	4	5
Ширина друкованого провідника, мм	$b_{пр}$	0,45	0,25	0,15	0,10
Відстань між елементами друкованого монтажу, мм	I	0,45	0,25	0,15	0,10
Гарантований поясок, мм	$b_{по}$	0,20	0,10	0,05	0,03
Відношення номінального діаметру найменшого з металізованих отворів до товщини друкованої плати, мм	$K_{дт}$	0,40	0,33	0,25	0,20

Табл. 3.3 – Допустимі похибки виконання елементів друкованого монтажу за 4 класом точності

Похибка	Позначення	Максимальне значення, мм
Зміщення провідників відносно ліній КС	$\delta_{сп}$	0,05
Розташування отворів (всіх) відносно вузлу КС	$\delta_{о}$	0,07
Розташування КМ відносно вузлу КС	$\delta_{КМ}$	0,015(0,05)
Фотокопії та фотошаблону	$\delta_{фф}$	0,06
Розташування КМ відносно вузлу КС на фотошаблоні	$\delta_{сш}$	0,05

Необхідну точність виконання елементів друкованого монтажу можна реалізувати виключно із застосуванням 4 і 5 класів точності. При цьому, виготовлення друкованої плати із використанням 5 класу точності є дорожчим.

Отже, було вирішено використати 4 клас точності при проектуванні друкованої плати

3.5. Проектування друкованого вузлу у середовищі Altium Designer

Altium Designer – САПР професійного рівня, який надає розробнику, можливості по реалізації електронних приладів високої складності. Програма являє собою комплекс, що дозволяє реалізувати увесь цикл виробництва електричних пристроїв всередині однієї системи, без використання допоміжних програм. Інженеру надаються можливості у розробці електричної схеми принципової, комп'ютерного моделювання її роботи, розробку певних електричних компонентів, що будуть використовуватись у пристрої, проектування друкованої плати (ОДП, ДДП та БДП) та отримання конструкторської документації на проєктований пристрій.

Для розробки пристрою були використані такі можливості САПР:

- Самостійне створення бібліотеки умовно-графічних позначень та посадкових майданчиків для електроних компонентів;
- Створення схеми електричної принципової;
- Трасування друкованої плати;
- Генерація необхідної конструкторської документації.

3.5.1 Створення проєкту друкованого вузлу

Щоб уся документація, яка необхідна для реалізації пристрою знаходилася в одному місці та мала внутрішні зв'язки між собою (УГП можна було прив'язувати до посадкових майданчиків, 3D STEP моделей і тд.), необхідним є створення проєктку друкованого вузлу.

PCB Project поєднує всередині себе усі необхідні документи, таким чином надає інженеру користувацький інтерфейс для одночасного оперування ними.

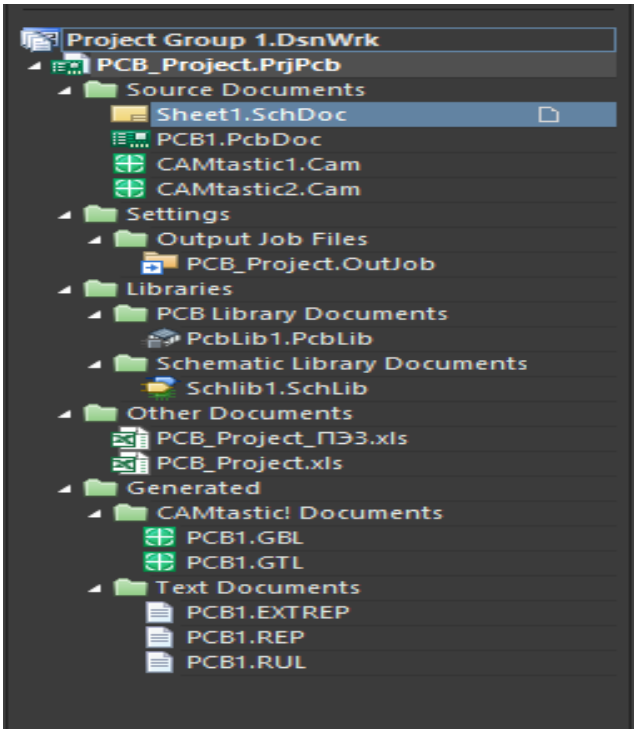


Рис. 3.1 Структура проекту в Altium Designer розроблюваного ДВ

3.5.2. Створення бібліотеки умовно-графічних позначень

Altium Designer – містить бібліотеки готових компонентів. Проте не завжди необхідний компонент є серед тих, що доступні в стандартних бібліотеках. Також його УГП може не відповідати вимогам ЄСКД. Тому були створенні УГП для усіх електронних компонентів, що використовуються у розробленому приладі, що будуть відповідати вимогам ЄСКД.

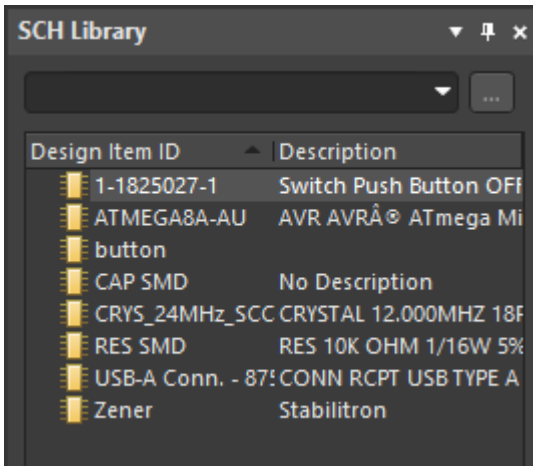


Рис. 3.2 Розроблена бібліотека УГП

3.5.3 Створення бібліотеки посадкових місць

Altium Designer містить бібліотеки посадкових майданчиків для найрозповсюдженіших типів корпусів. Проте не завжди необхідний компонент наявний серед стандартних. Тому була розроблена бібліотека посадкових місць для усіх електричних компонентів, використаних у проєкті.

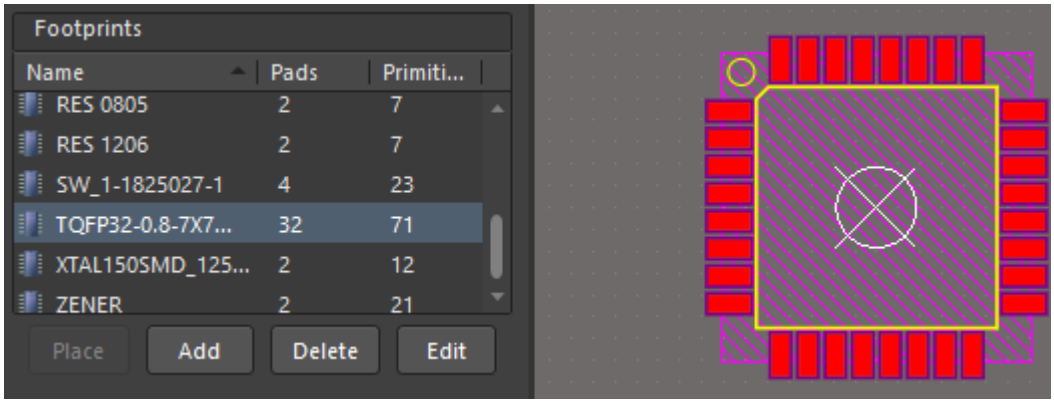


Рис. 3.3 Розроблена бібліотека посадкових місць

3.5.4 Створення схеми електричної принципової

За допомогою вбудованого у AD редактора принципових схем, з використанням самостійно створеної бібліотеки УГП, було накреслено схему електричну принципову.

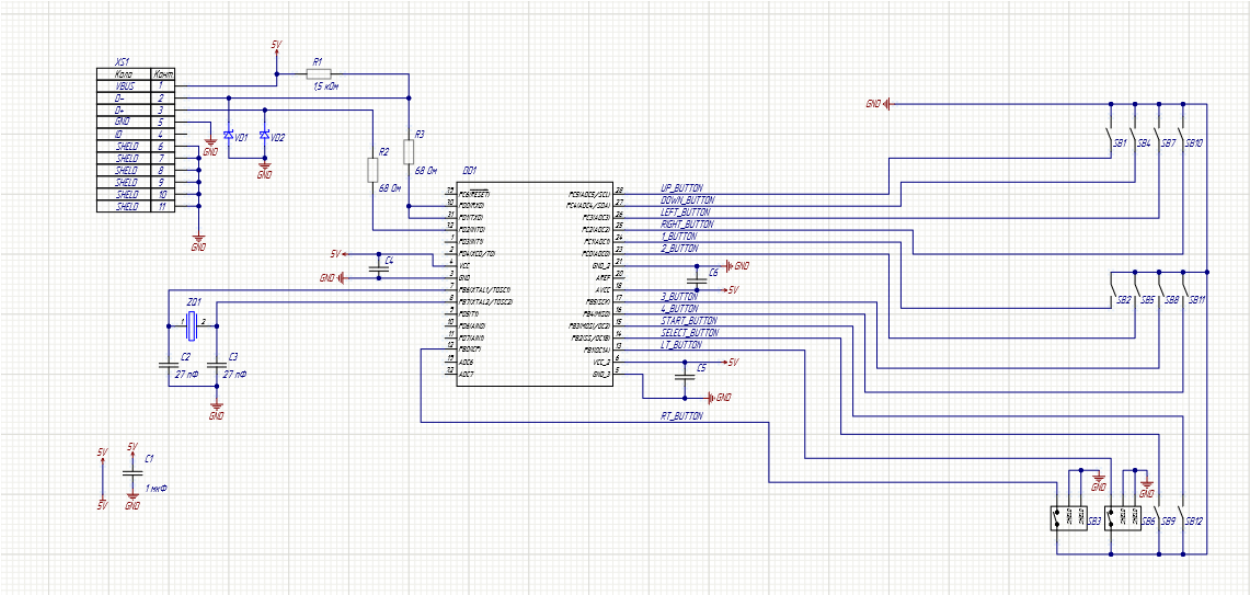
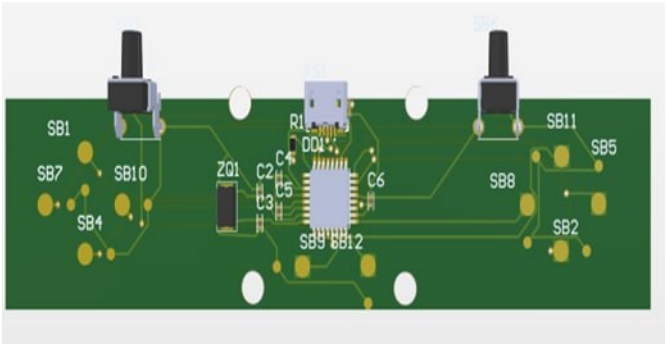


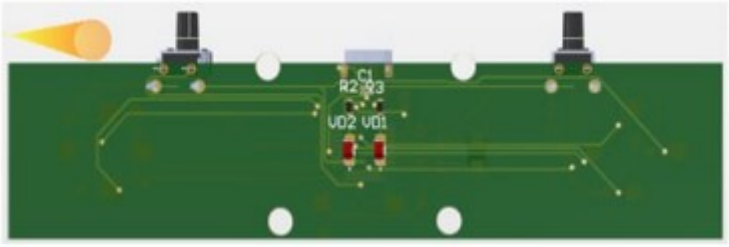
Рис. 3.4 Розроблена схема електрична принципова

3.5.5 Трасування друкованої плати

Трасування ДП може виконуватися двома способами: вручну та автоматично. Для того щоб отримати адекватне трасування автоматичним способом, необхідно детально налаштувати відповідні коефіцієнти у програмі і все-одно отриманий результат буде мати багато недоліків і потребуватиме доопрацювання (у випадку проектування складного пристрою). Так як проекткуэться не складний пристрій тому принято рішення використати автоматичне трасування. Результат зображено на рис. 3.5.



а)



б)

Рис. 3.5. Зовнішній вигляд ДП у 3D перспективі а) вид зверху, б) вид зниз

3.6 Розрахунок елементів друкованого монтажу

3.6.1 Визначення мінімальної ширини друкованого провідника на постійному струмі для лінії GND

Мінімальна ширина друкованого провідника на постійному струмі b_{min} (мм) для ланцюгів живлення та землі визначається формулою 3.1

$$b_{min} = \frac{I_{max}}{j_{доп} * t_{пров}} \qquad (3.1)$$

де I_{max} – пікове значення струму через провідник, А;

$j_{доп}$ – допустима щільність струму друкованого провідника, що виготовлений комбінованим позитивним методом,

$j_{доп}$ = 48 А/мм^2 [ОСТ 4.010.022];

$t_{пров}$ – товщина друкованого провідника, мм;

$$t_{пров} = h_ф + h_{гм} + h_{хм}, \quad (3.2)$$

де $h_ф$ – товщина фольги матеріалу, $h_ф$ = 0,035 мм;

$h_{гм}$ – товщина гальванічно осадженої міді $h_{гм}$ = 0,055 мм;

$h_{хм}$ – товщина шару хімічно осадженої міді $h_{хм}$ = 0,0065 мм.

$$t_{пров} = 0,035 + 0,055 + 0,0065 = 0,0965 \text{ (мм)}$$

Для визначення максимального струму I_{max} важливо розглянути ситуацію, при якій споживання струму пристроєм буде максимальним. Це можливо у ситуації натисненій кнопці при подальшій передачі інформації через USB. Максимальний струм I_{max} розраховується за формулою 3.3.

$$I_{max} = I_1 + I_2 + I_3 \quad (3.3)$$

Де I_1 – максимальний струм, що подається на мікроконтролер;

I_2 – максимальний струм, що подається при натисканні кнопки.

I_3 – максимальний струм, що споживається з USB.

В сумі маємо 200 + 50 + 70 = 320 (мА).

Можна розрахувати мінімальну ширину друкованого провідника на постійному струмі для ліній живлення і землі:

$$b_{min} = \frac{320*10^{-3}}{48*0,0965} = 0,069 \text{ (мм)}.$$

Отримано мінімальну ширину друкованих провідників лінії GND. Для підвищення надійності та покращення якості живлення компонентів приладу, вирішено збільшити ширину друкованих провідників ліній живлення та землі до 0.3 мм. Ця змінна не погіршила якість трасування.

3.6.2 Визначення мінімальної ширини провідника з урахуванням допустимого падіння напруги на ньому

Необхідно перевірити максимальне падіння напруги, яке відбувається при протіканні найбільшого струму по-найдовшому друкованому провіднику:

$$b_{\min U} = \frac{p \cdot I_{max} \cdot L_{пров}}{U_{доп} \cdot t_{пров}}, \quad (3.4)$$

де ρ–питомий опір провідника, виготовленого комбінованим позитивним методом, ρ=0,0175 $\frac{\text{Ом}\cdot\text{мм}^2}{\text{м}}$

$L_{\text{пров}}$ – довжина найдовшого друкованого провідника ДП, $L_{\text{пров}}$=307мм

$U_{\text{доп}}$ – допустиме падіння напруги на друкованому провіднику,

$U_{\text{доп}}$ = 0,05×$U_{\text{п}}$ =0,05×5=0,25 (В)

Мінімально допустима ширина провідника при заданих параметрах задовольняє вимогу по максимальному допустимому падінні напруги на друкованому провіднику:

$$b_{\min U} = \frac{p \cdot I_{max} \cdot L_{\text{пров}}}{U_{\text{доп}} \cdot t_{\text{пров}}} = \frac{0{,}0175 \cdot 0{,}37 \cdot 0{,}307}{0{,}0965 \cdot 0{,}25} = 0{,}082\ (\text{В}) < 0{,}25\ (\text{В})$$

3.6.3. Розрахунок номінального діаметру монтажного отвору

Деякі компоненти встановлюються за допомогою DIP – монтажу, в отвори друкованої плати. Їх діаметр може бути розрахований за формулою:

$$d \geq d_{\text{вэ}} + \Delta d_{\text{мо}} + r, \qquad (3.5)$$

де $d_{\text{вэ}}$ – діаметр виводу елементів, для якого визначається діаметр монтажного отвору,

Для комнонентів SB3,6 діаметр виводу елементів складає 0,5 мм.

Δd – нижнє граничне відхилення від номінального діаметру МО, $\Delta d_{\text{мо}}$=0,1 мм

r– різниця між мінімальним діаметром МО та максимальним діаметром виводу елемента, r=0,1…0,2 мм

$$d_{SB} \geq d_{\text{вэ}} + \Delta d_{\text{мо}} + r = 0{,}5 + 0{,}1 + 0{,}2 = 0{,}8\ (\text{мм})$$

3.6.4. Розрахунок діаметра контактного майданчика

Мінімальний діаметр контактного майданчика може бути розрахований за формулою:

$$D_{min} = D_{min1} + 1{,}5 \cdot h_{\text{ф}} + 0{,}03, \qquad (3.6)$$

де D_{min1} – мінімальний ефективний діаметр КМ, мм,

$h_ф$ – товщина фольги, $h_ф$ = 0,035 мм. Коефіцієнт 1,5$h_ф$ враховує підтравлювання фольги друкованого провідника у ширину,

$$D_{\min I} = 2 \cdot (b_{по} + \frac{d_{max}}{2} + \delta_o + \delta_{км}), \qquad (3.7)$$

де d_{max}– максимальний діаметр отвору в ДП, мм,

$b_{по}$ - ширина пояска КМ, $b_{по}$ =0,05 мм;

δ_o- похибка розташування центру отвору відносно вузла КС, δ_o=0,07 мм,

$\delta_{км}$- похибка розташування центру КМ відносно вузла КС, $\delta_{км}$=0,05.

Максимальний діаметр просвердленого отвору у друкованій платі розраховується за формулою:

$$d_{max}=d+\Delta d+(0,1…0,15), \qquad (3.8)$$

де d– номінальний діаметр МО, мм,

Δd- допуск на діаметр отвору, Δd=0,05 мм

Максимальний діаметр КМ:

$$D_{max} = D_{min}+0,02, \qquad (3.9)$$

Для SB3, SB6:

За формулою 3.8 розраховуємо максимальний діаметр просвердленого отвору:

$d_{max} = d + \Delta d + (0,1…0,15) = 0,8 + 0,05 + 0,1 = 0,95$ (мм)

За формулою 3.7 розраховуємо мінімальний ефективний діаметр КМ:

$$D_{\min I} = 2 \cdot \left(b_{по} + \frac{d_{max}}{2} + \delta_o + \delta_{км}\right) = 2 \cdot \left(0,05 + \frac{0,95}{2} + 0,07 + 0,05\right) = 1,29 \text{ (мм)}$$

За формулою 3.6 розраховуємо мінімальний діаметр КМ:

$$D_{min} = D_{min1} + 1,5 \cdot h_ф + 0,03 = 1,29 + 1,5 \cdot 0,035 + 0,03 = 1,3725 \text{ (мм)}$$

За формулою 3.9 розраховуємо максимальний діаметр просвердленого отвору:

D_{max} = 1,3725 + 0,02=1,3925 (мм)

3.6.5. Розрахунок мінімальної ширини друкованого провідника

Для обрахування цього параметру, необхідно застосувати формулу:

$$b_{min}=b_{пр}^{г} + 1{,}5 \cdot h_{ф} + 0{,}03, \qquad (3.10)$$

де $b_{пр}^{г}$–мінімальна ширина провідника. Визначаємо з таблиці класів точності (табл.3.2). Для 4-го класу точності $b_{пр}^{г} = 0{,}15$ мм.

Отже:

$$b_{min}=b_{пр}^{г} + 1{,}5 \cdot h_{ф} + 0{,}03 = 0{,}15 + 1{,}5 \cdot 0{,}035 + 0{,}03 = 0{,}2325 \text{ (мм)}$$

Максимальна ширина провідника:

$$b_{max} = b_{min} + 0{,}02, \qquad (3.11)$$

$$b_{max} = 0{,}2325 + 0{,}02 = 0{,}2525 \text{ (мм)}$$

3.6.6 Розрахунок мінімальної відстані між провідником та контактним майданчиком

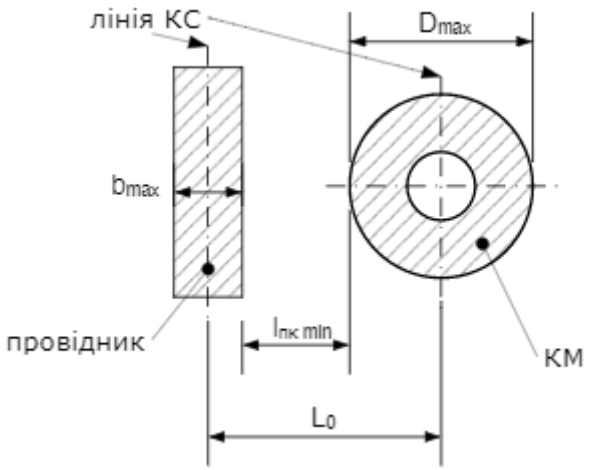


Рис. 3.6 – Схематичне зображення відстані між провідником і контактним майданчиком

Відстань між провідником і контактним майданчиком може бути розрахована за формулою 3.12

$$l_{ПКМ\,min} = L_0 - \left(\frac{D_{max}}{2} + \delta_{км} + \frac{b_{max}}{2} + \delta_{сп}\right), \qquad (3.12)$$

де L_0– відстань між центрами отворів та друкованим провідником, які кратні кроку КС, L_0=1,25 мм (найгірший випадок).

D_{max} - максимальний діаметр КП,

b_{max} - максимальна ширина провідника,

$\delta_{км}$ - похибка розташування центра КП відносно вузла КС, $\delta_{км}$=0,05 (табл.3.3),

$\delta_{сп}$ - похибка, яка враховує зміщення провідника, $\delta_{сп}$=0,05 мм

$$l_{\text{ПКМ}\,min} = L_0 - \left(\frac{D_{max}}{2} + \delta_{\text{км}} + \frac{b_{max}}{2} + \delta_{\text{сп}}\right)$$

$$= 1{,}25 - \left(\frac{1{,}9}{2} + 0{,}05 + \frac{0{,}2526}{2} + 0{,}05\right) = 0{,}073\ (\text{мм})$$

Отримане значення може бути реалізоване за допомогою обраного класу точності.

3.6.7 Розрахунок мінімальної відстані між двома сусідніми провідниками

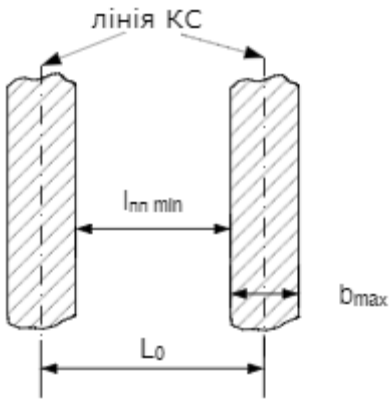


Рис. 3.7 – Схематичне зображення відстані між двома сусідніми провідниками

Мінімальна відстань між двома сусідніми провідниками може бути розрахована за формулою:

$$l_{\text{ПП}\,min} = L_0 - (b_{max} + 2 \cdot \delta_{\text{сп}}), \qquad (3.13)$$

Так як для трасування друкованої плати необхідно проводити друковані провідники між виводами мікросхем, застосовано координатну сітку із кроком 0.5 мм, тобто L_0= 0.5 мм.

$l_{\text{ПП}\,min} = L_0 - (b_{max} + 2 \cdot \delta_{\text{сп}}) = 0{,}5 - (0{,}2525 + 2 \cdot 0{,}05) =$ $0{,}1475\ (\text{мм})$

Отже, мінімальна відстань між двома друкованими провідниками має складати 0,1475 мм.

3.6.8 Розрахунок мінімальної відстані між двома контактними майданчиками

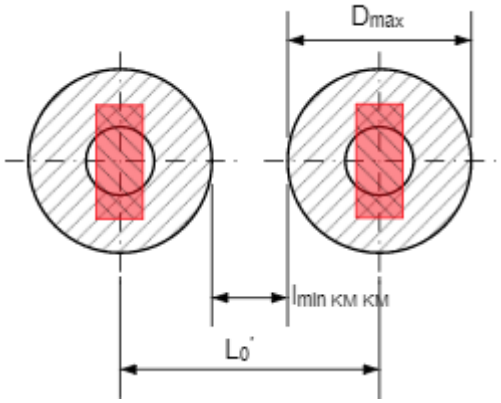


Рис. 3.8 – Схематичне зображення відстані між двома контактними майданчиками

Мінімальна відстань між двома контактними майданчиками може бути розрахована за формулою

$$l_{\text{min КМ КМ}} = L_{01} - (D_{max} + 2 \cdot \delta_{\text{КМ}}), \qquad (3.14)$$

де L_{01}- відстань між центрами сусідніх КП, L_{01}=2,5 мм.

Підставивши числа, маємо:

$$l_{\text{min КМ КМ}} = L_{01} - (D_{max} + 2 \cdot \delta_{\text{КМ}}) = 2{,}5 - (1{,}3 + 2 \cdot 0{,}05) = 1{,}10 \text{ (мм)}.$$

Отримане значення задовольняє 4-ий клас точності.

3.7 Електричний розрахунок друкованої плати

У друкованих платах неминуче виникають паразитні електричні величини. Особливо це явище проявляється у випадку зменшення геометричних розмірів (через посилення ємнісних та індуктивних зв’язків між сигнальними провідниками) та у випадку використання ДДП та БДП. При зміні логічного рівня сигналу на цифрових компонентах, у ланцюгах виникають високочастотні імпульсні струми з крутими фронтами, які внаслідок наявності паразитних зв’язків продукують завади на сусідніх провідниках. У деяких випадках вони здатні викликати хибні спрацювання цифрових компонентів

Паразитні реактивності – це приховані паразитні ємності та індуктивності. Вони утворюються за рахунок паразитних зв'язків між виводами компонентів, довгими друкованими провідниками, ємності між друкованими контактними майданчиками та полігоном землі, паразитною взаємодією між перехідними отворами. При проектуванні друкованого вузлу необхідно, щоб величини паразитних параметрів не перевищували допустимі значення.

Для аналізу впливу паразитних параметрів на роботу друкованого вузла, проведено електричних розрахунок друкованої плати.

3.7.1. Розрахунок падіння напруги на найдовшому друкованому провіднику

Необхідний розрахунок проведений у п. 3.6.2. Отримане значення падіння напруги на найдовшому провіднику склало 0,082 В, що є менше 5% від напруги живлення. Отже ширина друкованих провідників вибрана вірно.

3.7.2 Найбільша ємність між двома сусідніми провідниками

Паразитна ємність на друкованій платі розраховується за формулою:

$$C = 0{,}12 \cdot \varepsilon \cdot l_{\text{пр}} \cdot \left[lg \frac{2 \cdot S}{b_{\text{пр}} + t_{\text{пр}}} \right]^{-1}, \qquad (3.15)$$

де S – відстань між двома паралельними провідниками, S=0,35 мм

$b_{\text{пр}}$ - ширина друкованого провідника, мм

$t_{\text{пр}}$ - товщина друкованого провідника, мм

$l_{\text{пр}}$ - довжина взаємного перекриття двох паралельних провідників, мм

ε – діелектрична проникність друкованої плати із урахуванням лаку

Для даного розрахунку ε може бути розрахована за формулою:

$$\varepsilon = \frac{\varepsilon_{\text{лаку}} * \varepsilon_{\text{мат.ДП}} * (h_{\text{мат.ДП}} + 2 * h_{\text{шар.лаку}})}{2 * \varepsilon_{\text{лаку}} * h_{\text{мат.ДП}} + \varepsilon_{\text{мат.ДП}} * h_{\text{шар.лаку}}} \qquad (3.16)$$

де $\varepsilon_{лаку}$ – діелектрична проникність лаку УР-231 ТУ 6-21-14-90, $\varepsilon_{лаку} = 4{,}5$;

$h_{\text{мат.ДП}}$ – товщина друкованої плати, $h_{\text{мат.ДП}} = 1{,}5$ мм;

$\varepsilon_{\text{мат.ДП}}$ – діелектрична проникність склотекстоліту FR4, $\varepsilon_{\text{мат.ДП}} = 4{,}5$;

$h_{\text{шар.лаку}}$ – товщина шару лаку, $h_{\text{шар.лаку}} = 10 * 10^{-6}$ м;

Маємо:

$$\varepsilon = \frac{\varepsilon_{\text{лаку}} * \varepsilon_{\text{мат.ДП}} * (h_{\text{мат.ДП}} + 2 * h_{\text{шар.лаку}})}{2 * \varepsilon_{\text{лаку}} * h_{\text{шар.лак}} + \varepsilon_{\text{мат.ДП}} * h_{\text{мат.ДП}}}$$

$$= \frac{4{,}5 * 4{,}5 * (1{,}5 * 10^{-3} + 2 * 10 * 10^{-6})}{2 * 4{,}5 * 10 * 10^{-6} + 4{,}5 * 1{,}5 * 10^{-3}} = 4.55$$

Тоді:

$$\text{C} = 0{,}12 \cdot \varepsilon \cdot l_{\text{пр}} \cdot \left[lg \frac{2 \cdot S}{b_{\text{пр}} + t_{\text{пр}}} \right]^{-1} = 0{,}12 \cdot 4{,}55 \cdot 0.307 \cdot \left[lg \frac{2 \cdot 0{,}35}{0{,}25 + 0{,}0965} \right]^{-1} = 0{,}549 \text{ (пФ)}$$

3.7.3 Найбільша взаємна індуктивність двох паралельних друкованих провідників

Паразитна індуктивність на друкованій платі розраховується за формулою:

$$\text{M} = 0{,}02 \left(l_{\text{пр}} lg \frac{\sqrt{l_{\text{пр}}^2 - L_0^2} + l_{\text{пр}}}{L_0} - \sqrt{l_{\text{пр}}^2 - L_0^2} + l_{\text{пр}} \right), \qquad (3.17)$$

де $l_{\text{пр}}$ – довжина перекриття паралельних провідників, $l_{\text{пр}}$= 307 мм

L_o-відстань між осьовими лініями двох паралельних провідників, L_o=0,15см

$$\text{M} = 0{,}02 \left(l_{\text{пр}} lg \frac{\sqrt{l_{\text{пр}}^2 - L_0^2} + l_{\text{пр}}}{L_0} - \sqrt{l_{\text{пр}}^2 - L_0^2} + l_{\text{пр}} \right) =$$

$$= 0{,}02 \left(0{,}307 \cdot lg \frac{\sqrt{0{,}09 - 0{,}0225} + 0{,}307}{0{,}15} - \sqrt{0{,}09 - 0{,}0225} + 0{,}15 \right) = 0{,}0673 \text{ (нГн)}$$

3.7.4 Потужність втрат двосторонньої друкованої плати

Потужність втрат визначається:

$$\text{P}_{\text{пот}} = 2 \cdot \pi \cdot f \cdot C \cdot E_n^2 \cdot tg\sigma, \qquad (3.18)$$

де f =12 МГц, тому що розрахунок виконується на робочій частоті приладу

*tg*σ – тангенс кута діелектричних втрат для матеріала друкованої плати

С – ємність друкованої плати, Ф:

Е – напруга живлення, Е = 5В.

Ємність друкованої плати можна розрахувати за формулою:

$$C = \frac{0{,}009 \cdot \varepsilon \cdot S_m}{h}, \qquad (3.19)$$

де ε – діелектрична проникність, ε = 4,5 для FR4

S_m - площа металізації, S_m =0,000456 м^2

h - товщина ДП, мм

Тангенс кута діелектричних втрат можна розрахувати за формулою:

$$tg\sigma = \frac{\varepsilon_{\text{лаку}} * h_{\text{мат.ДП}} * tg\sigma_{\text{мат.ДП}} + 2 * \varepsilon_{\text{мат.ДП}} * h_{\text{шар.лак}} * tg\sigma_{\text{лаку}}}{2 * \varepsilon_{\text{лаку}} * h_{\text{шар.лак}} * \varepsilon_{\text{мат.ДП}} * h_{\text{мат.ДП}}} \qquad (3.20)$$

де $\varepsilon_{лаку}$ – діелектрична проникність лаку УР-231 ТУ 6-21-14-90, $\varepsilon_{лаку}$ = 4,5;

$h_{\text{мат.ДП}}$ – товщина друкованої плати, $h_{\text{мат.ДП}}$ = 1,5 мм;

$\varepsilon_{\text{мат.ДП}}$ – діелектрична проникність склотекстоліту FR4, $\varepsilon_{\text{мат.ДП}}$ = 4,5;

$h_{\text{шар.лаку}}$ – товщина шару лаку, $h_{\text{шар.лаку}}$ = 10 *10^{-6} м;

tgσ$_{\text{мат.ДП}}$ – тангенс кута діелектричних втрат матеріалу ДП,

tgσ$_{\text{мат.ДП}}$ = 0,002;

tgσ$_{\text{мат.ДП}}$ – тангенс кута діелектричних втрат лаку УР-231 ТУ 6-21-14-90,

tgσ$_{\text{мат.ДП}}$ = 0,03;

Маємо:

$$tg\sigma = \frac{\varepsilon_{\text{лаку}} * h_{\text{мат.ДП}} * tg\sigma_{\text{мат.ДП}} + 2 * \varepsilon_{\text{мат.ДП}} * h_{\text{шар.лак}} * tg\sigma_{\text{лаку}}}{2 * \varepsilon_{\text{лаку}} * h_{\text{шар.лак}} * \varepsilon_{\text{мат.ДП}} * h_{\text{мат.ДП}}}$$

$$= \frac{4.5 * 1.5 * 10^{-3} * 0.002 + 2 * 4.5 * 10 * 10^{-6} * 0{,}03}{2 * 4.5 * 10 * 10^{-6} * 4.5 * 1.5 * 10^{-3}} = 0{,}0024$$

$$C = \frac{0{,}009 * \varepsilon * S_m}{h} = \frac{0{,}009 * 4{,}5 * 0{,}000456}{1{,}5 * 10^{-3}} = 0{,}012 \text{ (нФ)}$$

$$P_{\text{пот}} = 2 \cdot \pi \cdot f \cdot C \cdot E_n^2 \cdot tg\sigma = 2 \cdot 3{,}14 \cdot 12 * 10^6 \cdot 0{,}000012 \cdot 25 \cdot 0{,}0024 = 54{,}2592 \text{ (нВт)}$$

3.8 Розрахунок вібромічності друкованої плати

Табл. 3.4 – Маси електричних компонентів, використаних у ДП.

Компонент	Кількість	Маса, г	Заг. Маса, г
Резистор 0805	3	0.07	0.21
Конденсатор 0805	6	0.05	0.3
Стабілітрон	2	0.1	0.2
IC AtMega8A-AU	1	0.15	0.15
Контактні кнопки	10	0.01	0.1
DIP кнопки	2	0.2	0.4
USB	1	2.00	2.0
SMD кварцовий резонатор	1	1	1
Всього			4.36

Маса електричних компонентів друкованої плати занесені до табл. 3.4

Розмір друкованої плати: a * b * δ = 115 * 25 * 1.5 мм.

Маса друкованої плати може бути розрахована за формулою:

$m = \rho * V = \rho * a * b * \delta = 2050 * 0{,}115 * 0{,}025 * 0{,}0015 = 0{,}0088$ (кг)

Параметри склотекстоліту FR4:

модуль Юнга $E = 3.02 * 10^{10}$ Па;

коефіцієнт Пуассона $\mu = 0.22$;

показник затухання $\varepsilon = 0.06$;

питома щільність $\nu = 2.05 * 10^4$ Н/м^3;

коефіцієнт запасу міцності $n_1 = 2$.

Обрано метод закріплення друкованої плати – спирання на 4 сторони. Схематично це зображено на рис. 3.9

Рис. 3.9 Схематичне зображення методу закріплення друкованої плати спиранням на 4 сторони

Далі наведено розрахунки методу закріплення на 4 сторони:

$$K_B = \frac{1}{\sqrt{1+\frac{m_g}{m_{ДП}}}} \quad (3.21)$$

$$K_B = \frac{1}{\sqrt{1+\frac{m_g}{m_{ДП}}}} = \frac{1}{\sqrt{1+\frac{0.00436}{0.0088}}} = 0.82$$

У (3.22) наведено формулу розрахунку коефіцієнта α, що враховує обраний тип закріплення друкованої плати:

$$\alpha = \pi^2 * \left(1 + \frac{a^2}{b^2}\right) \quad (3.22)$$

$$\alpha = \pi^2 * \left(1 + \frac{a^2}{b^2}\right) = 3.14^2 * \left(1 + \frac{0.115^2}{0.030^2}\right) = 154.74$$

У (3.23) наведено формулу розрахунку циліндричної жорсткості D:

$$D = E * \frac{h^3}{12-(1-\mu^2)} \quad (3.23)$$

$$D = E * \frac{h^3}{12-(1-\mu^2)} = 3.02 * 10^{10} * \frac{0.0015^2}{12 * (1-0.22^2)} = 8.92$$

У (3.24) наведено формулу власної частоти коливань друкованої плати $f_{власн}$:

$$f_{власн} = \frac{K_B * \alpha}{2 * \pi * \alpha^2} * \sqrt{\frac{D * g}{v * h}} \quad (3.24)$$

$$f_{власн} = \frac{K_B * \alpha}{2 * \pi * a^2} * \sqrt{\frac{D * g}{v * h}} = \frac{0.82 * 154.74}{2 * 3.14 * 0.115^2} * \sqrt{\frac{8.92 * 9.81}{2.05 * 10^4 * 0.0015}}$$

$$= 1527.7810 * \sqrt{2.84} = 2574.7 \text{ (Гц)}$$

Так як $f_{власн} > 250$ Гц – конструкція жорстка. Тому немає потреби у проведенні розрахунків амплітуди вібрацій на власній частоті та динамічного прогину.

3.9 Розрахунок надійності друкованої плати

Надійність є однією з основних характеристик електричного приладу, що характеризується властивістю виконувати заявлені функції протягом певного часу в заданих умовах експлуатації та з проведенням необхідного обслуговування. Для отримання інформації про надійність прилада, необхідно проаналізувати надійність кожного його компонента. Надійність всієї системи визначається надійністю найменш надійного з її компонентів.

Надійність складається із таких параметрів:

- P(t) – ймовірність безвідмовної роботи протягом часу t;
- λ(t) – інтенсивність відмов;
- T – середній час напрацювання на відмову;
- Q(t) – ймовірність відмови.

Середній час напрацювання на відмову визначається за формулою 3.25

$$T_{\text{сер}} = 1/\lambda \qquad (3.25)$$

Інтенсивність відмов визначається за формулою 3.8.2

$$\lambda_{\text{e}} = \lambda_{\text{oe}} * \ \text{K}_1 * \ \text{K}_2 * \ldots * \ \text{K}_{\text{n}} \qquad (3.26)$$

де λ_{oe} – інтенсивність відмов компоненту за нормальних умов роботи (н.у. – темп. навколишнього середовища $25 \pm 15\%$);

K_n – коефіцієнт електричного навантаження, що враховує умови експлуатації компонента і може бути розрахований за формулою 3.27

$$K_{\text{нав}} = \frac{H_{\text{робоч}}}{H_{\text{номінал}}} \qquad (3.27)$$

Розрахунки проводяться на основі базової інтенсивності відмов для н.у., бо згідно ТЗ передбачена робота пристрою лише всередині приміщення н.у. Ця величина взята[14].

Окрім базової інтенсивності відмов також будуть враховані:

– $K_н$ – коефіцієнт навантаження;

– α_t – коефіцієнт температурного режиму;

– α_e – коефіцієнт зовнішніх впливів;

3.9.1 Розрахунок коефіцієнта навантаження для резисторів

На резисторах падає певна напруга, яка перетворюється в струм. Найбільш навантаженим у пристрої є резистори R2-R3. Вони виступають у якості захисту від різької зміни напруги. При розрахунку коефіцієнта навантаження порівняємо номінальну та реальну розсіювану потужності:

$$K_{\text{нав}} = \frac{P_{\text{робоч}}}{P_{\text{номінал}}} = \frac{U^2}{R * P_{\text{номінал}}} = \frac{3.6^2}{67 * 0.5} = 0.387$$

де $P_{\text{номінал}}$ – номінальна потужність резистора, $P_{\text{номінал}}$ = 0,5 Вт;

U – напруга стабілізації, U = 3,6 В;

R – опір резистора, R = 67 Ом.

3.9.2 Розрахунок коефіцієнта навантаження для конденсаторів

$K_{\text{нав}}$ розраховується на основі номінальної та робочої напруги конденсатора. В якості робочої напруги обрано 3.6 В – напруга що подається на мікроконтролер

$$K_{\text{нав } c\text{кв.}} = \frac{U_{\text{робоч}}}{U_{\text{номінал}}} = \frac{3.6}{100} = 0.036$$

$$K_{\text{нав } c\text{жив.}} = \frac{U_{\text{робоч}}}{U_{\text{номінал}}} = \frac{3.6}{16} = 0.225$$

Решта елементів

Всі компоненти підбиралися таким чином, щоб їх робочі навантаження не перевищували номінальні. Тому для всіх інших компонентів приймаємо $K_{\text{нав}}$ як 1 – найгірший випадок.

3.9.3 Розрахунок результуючої інтенсивності відмов

Передбачається, що приклад буде експлуатуватися у капітальних, критих, опалювальних приміщеннях із штучною вентиляцією. Цим умовам відповідає кліматичне виконання приладу УХЛ 4.2 за винятком того температурний режим складає [+1;+35]°С. Враховуючи це, із

довідника обираємо поправочний коефіцієнт a_t [14]. Прилад належить до класу пристроїв, що експлуатуються у приміщеннях з регульованою температурою та вологістю – тому значення поправочного коефіцієнта становить 1.1 [15].

Маючи необхідні дані, створено табл. 3.5. Значення необхідних коефіцієнтів взяті з [15].

Табл. 3.5 – Надійність електричних компонентів пристрою

Компонент	N	$\lambda_{0e} * 10^{-6}$, год$^{-1}$	$K_н$	a_t	a_e	$\lambda_{0e} * N * K_н * a_t * a_e$
Резистори	3	0.044	0.387	0.2	1.1	$0.011238 * 10^{-6}$
Конденсатори керамічні для резонатору	2	0.022	0.036	0.4	1.1	$0.000697 * 10^{-6}$
Конденсатори керамічні для живлення	4	0.022	0.225	0.4	1.1	$0.008712 * 10^{-6}$
Мікросхеми	1	0.025	1	1.2	1.1	$0.02475 * 10^{-6}$
Стабілітрон	2	0.004	1	0.9	1.1	$0.00792 * 10^{-6}$
SMD кварцовий резонатор	1	0.03	1	1	1.1	$0.033 * 10^{-6}$
Контактна кнопка	10	0.16	1	1	1.1	$1.76 * 10^{-6}$
Кнопка	2	0.16	1	1	1.1	$0.352 * 10^{-6}$
Друкована плата	1	0.002	1	1	1.1	$0.0022 * 10^{-6}$

Друкована плата (мет отвори)	20	0.000017	1	1	1.1	0.0008976 * 10^{-6}
Контакти роз'ємів	4	0.015	1	1	1.1	0.066 * 10^{-6}
Пайка виводу	84	0.000069	1	1	1.1	0.0063756 * 10^{-6}
			Всього			2.2737906 * 10^{-6}

де a_e – поправочний коефіцієнт зовнішніх впливів, для приміщень з регульованою вологістю та температурою, $a_e = 1.1$ [15];

a_t – поправочний температурний коефіцієнт.

Остаточна інтенсивність відмов дорівнює сумі інтенсивностей відмов компонентів:

$$\lambda_p = \sum_{i=1}^{n} \lambda_{pi} \quad (3.28)$$

Де $\lambda_{pi} = \lambda_{0e} * N * K_н * a_t * a_e$;

N – кількість компонентів даного класу;

λ_{0e} – інтенсивність відмов;

a_t – поправочний температурний коефіцієнт;

a_e – поправочний коефіцієнт зовнішніх впливів, залежить від умов експлуатації пристрою.

$$\lambda_p = \sum_{i=1}^{n} \lambda_{pi} = 2.2737906 * 10^{-6} \text{ год}^{-1}$$

Середній час напрацювання на відмову може бути розрахований за формулою:

$$T_{\text{ср}} = \frac{1}{\lambda_p} \quad (3.29)$$

$$T_{\text{ср}} = \frac{1}{\lambda_p} = \frac{1}{2.2737906\ 10^{-6}} = 439794 \text{ (год)} \approx 50 \text{ років}$$

Ймовірність безвідмовної роботи P(t) та ймовірність відмов Q(t) протягом часу t може бути розрахована за формулою:

$$P(t) = e^{-\lambda_p * t} \quad (3.30)$$

$$Q(t) = 1 - P(t) \quad (3.31)$$

Необхідно обчислити ймовірність безвідмовної роботи та ймовірність відмови протягом року з початку роботи приладу. У одному році 8760 годин. Тоді, маємо

$$P(t) = e^{2.2737906\,10^{-6} * 8760} = 0.98$$

$$Q(t) = 1 - P(t) = 0.02$$

На рис. 3.10. наведено графік залежностей P(t) та Q(t) на проміжку часу [1; 500000] від годин роботи, в таблиці 3.9.3.2 наведено розрахунки P(t) та Q(t) в даному проміжку часу. З графіку видно, що ймовірності стануть рівними після проходження 30 років. Причиною такої високої надійності є майже лабораторні умови експлуатації приладу. Також отриманий час роботи на відмову не враховує раптових відмов та низьких температур.

Табл. 3.6 – Розрахунок залежностей P(t) та Q(t)

T, год	P(t)	Q(t)
100	0,999772647	0,000227
1000	0,997728792	0,002271
10000	0,977518652	0,022481
100000	0,796618756	0,203381
200000	0,634601442	0,365399
300000	0,505535411	0,494465

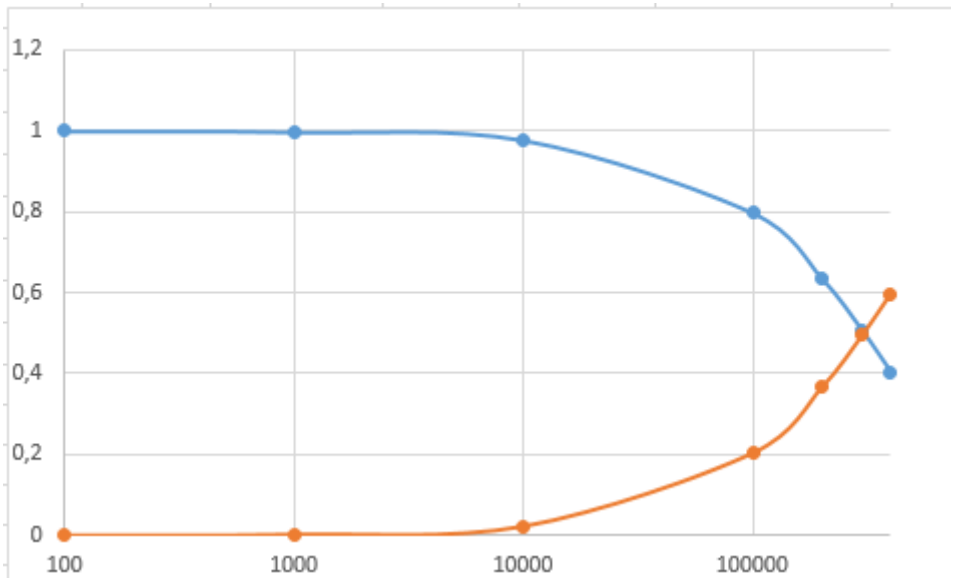


Рис. 3.10 Графік залежностей P(t) та Q(t) від часу

Загалом, отримані результати надійності повністю задовольняють вимогам ТЗ.

Висновки до розділу 3

У розділі 3 було визначено параметри друкованої плати пристрою. Було обрано двосторонню друковану плату, виготовлену зі склотекстоліту FR4-2-35-1.5, комбінованим позитивним методом, із застосуванням 4-го класу друкованого монтажу. На основі проведеного вибору було розроблено проект в Altium Designer, а також було проведено певні розрахунки з яких слідує:

По-перше розрахунки в п. 3.5 підтверджують можливість виготовлення друкованої плати обраним методом із обраним класом точності.

По-другє електричний розрахунок друкованої плати показав, що отримана напруга втрат значно нижча допустимої в 5%.

По-третє розрахунок вібромічності показав, що при обраному спирані на 4 сторони конструкція має високу жорсткість.

По-четверте надійність приладу склала приблизно 50 років, це пов’язано з тим що надійність була розрахована для раптових відмов.

Розділ 4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вибір інтерфейсу передачі даних для мікроконтролера.

Інтерфейс передачі даних може бути реалізован так і за допомогою бездротових засобів так й дротовими. При цьому варто пам’ятати, що прилад до якого під’єднують розроблювальний пристрій повинен підтримувати дану технологію передачі.

Серед бездротових засобів передачі даних самим відомими є Wi-Fi та Bluetooth. Розглянемо дані засоби передачі даних детальніше.

Bluetooth є одним з найстаріших засобів для передачі даних на сьогоднішній день. Дана технологія дозволяє передавати дані на малих відстанях, вона застосовувалась лише на телефонах та планшетах. Насамперед Bluetooth має багато реалізацій, серед відомих Bluetooth Low Energy, який завдяки меньшому використанні енергії акамулятора телефону, зміг перевершити стандартний Bluetooth. Дана технологія була представлена компанією Nokia, і була однією з частин стандартного Bluetooth. На даний момент BLE підтримується великою кількістю виробників смартфонів та комп’ютерів. Bluetooth для передачі даних застосовує радіохвилі високочастотного електромагнітного поля. Технологія Bluetooth має суттєвий недолік, а саме передача даних можлива на малих відстанях. Даний недолік було вирішено у технології Wi-Fi. Головним недоліком є залежність швидкості передачі даних від відстані на якій знаходиться об’єкт від центру точки доступу. [16]

Дані технологі хоч й дозволяють передавати дані безперешкод, але вони мають суттєвий недолік, у випадку пристроїв з підтримкою Wi-Fi може статись, що прилад може не запрацювати при ситуації, коли точка доступу має велику завантаженість, а у випадку Bluetooth може бути ситуація коли прилад з даною технологією не підтримується комп’ютером. Тому розробники використовують в якості запасного варіанту для передачі даних інтерфейс USB.

Інтерфейс USB (Universal Serial Bus) є одним з розповсюджених засобів для передачі даних. Даний інтерфейс має 3 швидкості для передачі даних[17]:

- LS – низька швидкість, 1,5 Мбіт/с
- FS – повна швидкість, 12 Мбіт/с
- HS – висока швидкість, 480 Мбіт/С

Інтерфейс USB 2.0 використовується в якості засобу передачі даних через його розповсюдженість та низьку ціну на ринку. Слід зауважити, що будь-який пристрій з підтримкою інтерфейсу USB має ієрархію дескрипторів, які описують інформацію для пристроя до якого було здійснено підключення. Опис інформації здійснюється таким чином, наприклад, інформація, виробник, ПЗ що підтримує пристрій, спосіб конфігурації, кількість кінцевих точок тощо. [18]

Серед загальних дескрипторів USB виділяють[18]:

- Дескриптор пристрою;
- Дескриптор конфігурації;
- Дескриптор інтерфейсу;
- Дескриптор кінцевої точки
- Дескриптор строк.

Розглянемо чотири типи дескрипторів детальніше, а саме дескриптор пристрою, дескриптор конфігурації, дескриптор інтерфейсу та дескриптор кінцевої точки.

По-перше, пристрої USB мають лише один дескриптор пристрою, який включає в себе інформацію таку що підтримує ревізію USB, що використовують для завантаження відповідного драйверу, та можливу кількість налаштувань пристрою. Число конфігурацій вказує на кількість розгалужень по дескрипторам конфігурації. [18]

По-другє дескриптор конфігурації, вказує на: величину потужності, що споживається від шини, як саме живиться пристрій, від власного джерела живлення, або ж з шини USB, та на кількість інтерфейсів, котрі є в

налаштуванні. Даний дескриптор використовують, коли пристрій до якого підключились зчитує дескриптор пристрою та приймає рішення, яку з конфігурацій запустити, пристрій до якого підключились може дозволити лише одну конфігурацію. [18]

По-третє, дескриптор інтерфейсу являє собою заголовок, або групування кінцевих точок в функціональну групу, що виконує єдину особливість пристрою. Даний дескриптор не має обмежень на кількість одночасних підключених інтерфейсів. [18]

По-четверте, дескриптор кінцевої точки вказує на тип передачі, направлення, інтервал опитування й максимальний розмір пакета, для кожної кінцевої точки. Під кінцевою точкою 0 завжди, за замовченням, приймають, як точку для керування і тому вона не має дескриптора. [18]

При використанні мікроконтролера AtMega8A-AU було вирішено використати програмну реалізацію USB драйверу, під час написання програмної реалізації пристрою.

4.2 Налаштування портів вводу-виводу мікроконтролера

Перед тим як, почати розробку приладу, потрібно налаштувати порти вводу-виводу мікроконтролера. Дані порти будуть як приймати дані, при натисненій кнопці, так й передавати дані по інтерфейсу USB. Крім цього для мікроконтролера використаємо зовнішній генератор тактової частоти. Тому, для того щоб пристрій працював коректно необхідно налаштувати порти вводу-виводу. [19]

В мікроконтролері AtMega8A-AU є портами вводу-виводу: порт B, порт C, та порт D. Кожний порт вводу-виводу має три регістри: PORTx, PINx, DDRx, при цьому кожний пін кожного з портів відповідає за певний біт регістру. [19]

Регістр PINx – це регістр зчитування, завдяки якому маємо інформацію про реальний логічний рівень на виводах порту, тобто коли відбувається

перехід з логічного 0 в логічну 1, можливий лише при певній напрузі і навпаки. [19]

Регістр DDRx дозволяє визначити, як саме буде працювати порт на вхід чи на вихід, тобто коли в регістрі знаходиться в лог. 0, то тоді порт працює на вхід, а коли – 1, тоді порт працює на вихід. [19]

Регістр POTRx виконує роль перемикача, тобто при налаштуванні виводу на вхід(вихід), від даного регістру залежить режим входу, або ж високим імпедансом(Hi-Z), або режим підтяжки. [19]

Режим з високим імпедансом входу, ввімкнений за замовчуванням. Даний режим дозволяє залишати один з виводів непідключеним, що на роботу не вливає. [19]

Режим підтяжки проявляється коли DDRx = 0 та PORTx = 1 замикає ключ підтяжки та до лінії підключають резистор 100кОм, що переводить не підключену лінію в стан логічної 1. Метою підтяжки є запобігти хаотичним змінам стану на вході. На рис 4.1 зображено загальний принцип роботи портів. [19]

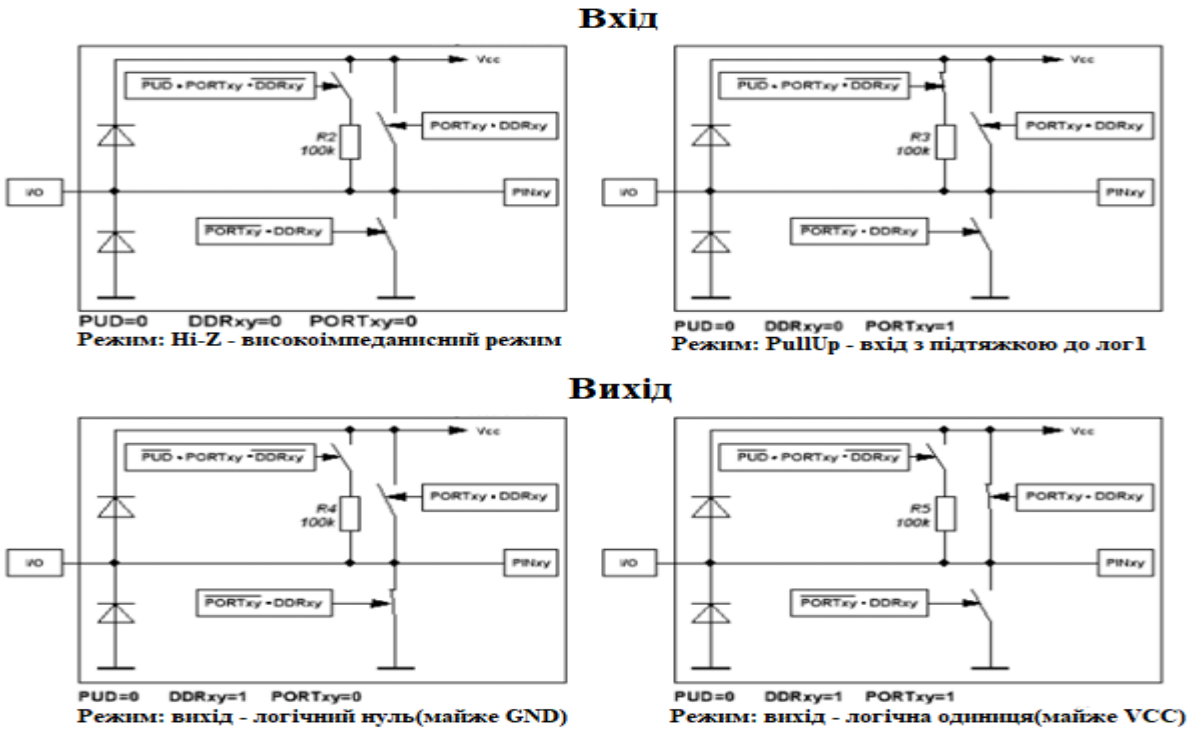


Рис. 4.1 Принцип робити портів на вхід(вихід)

В табл. 4.1 наведено налаштування портів вводу-виводу. Для полегшення розуміння вказано до якого блоку структурної схеми відноситься порт.

Табл. 4.1 – Найменування портів вводу-виводу

Порт	Мітка порту	Призначення	Налаштування вхід/вихід(режим)	Блок
PD0	RXD	Ініціалізація портів для передачі даних	Вихід	Блок передачі даних
PD1	TXD		Вихід	
PD2	INT0		Вихід	
PB6	XTAL1/TOSC1	Ініціалізація портів для підключення резонатору	Вхід(Hi-Z)	Блок генерації частоти
PB7	XTAL2/TOSC2		Вхід(Hi-Z)	
PB0	ICP	Ініініціаліція бічних кнопок	Вхід(підтяжка)	Блок керування
PB1	OC1A		Вхід(підтяжка)	
PB2	SS/OC1B	Ініціалізація кнопки Select	Вхід(підтяжка)	
PB3	MOSI/OC2	Ініціалізація кнопки Start	Вхід(підтяжка)	
PB4	MISO	Ініціалізація кнопки 4		
PB5	SCK	Ініціалізація кнопки 3		
PC0	ADC0	Ініціалізація кнопки 2	Вхід(підтяжка)	
PC1	ADC1	Ініціалізація кнопки 1	Вхід(підтяжка)	

PC2	ADC2	Ініціалізація кнопки Right	Вхід(підтяжка)	
PC3	ADC3	Ініціалізація кнопки Left	Вхід(підтяжка)	
PC4	ADC4/SDA	Ініціалізація кнопки Down	Вхід(підтяжка)	
PC5	ADC5/SCL	Ініціалізація кнопки Up	Вхід(підтяжка)	

4.3 Алгоритм роботи пристрою

Після того коли вперше увімкнули пристрій, він ще не пройшов ініціалізацію та має пройти налаштування на програмному рівні.

Для цього при розробці пристрою, в якому є інтерфейс USB, застосовують дескриптори, які при ввімкнені пристрою проводять ініціалізацію приладу та перевіряють сумісність. На рис. 4.2 зображено порядок проходження ініціалізації приладу.

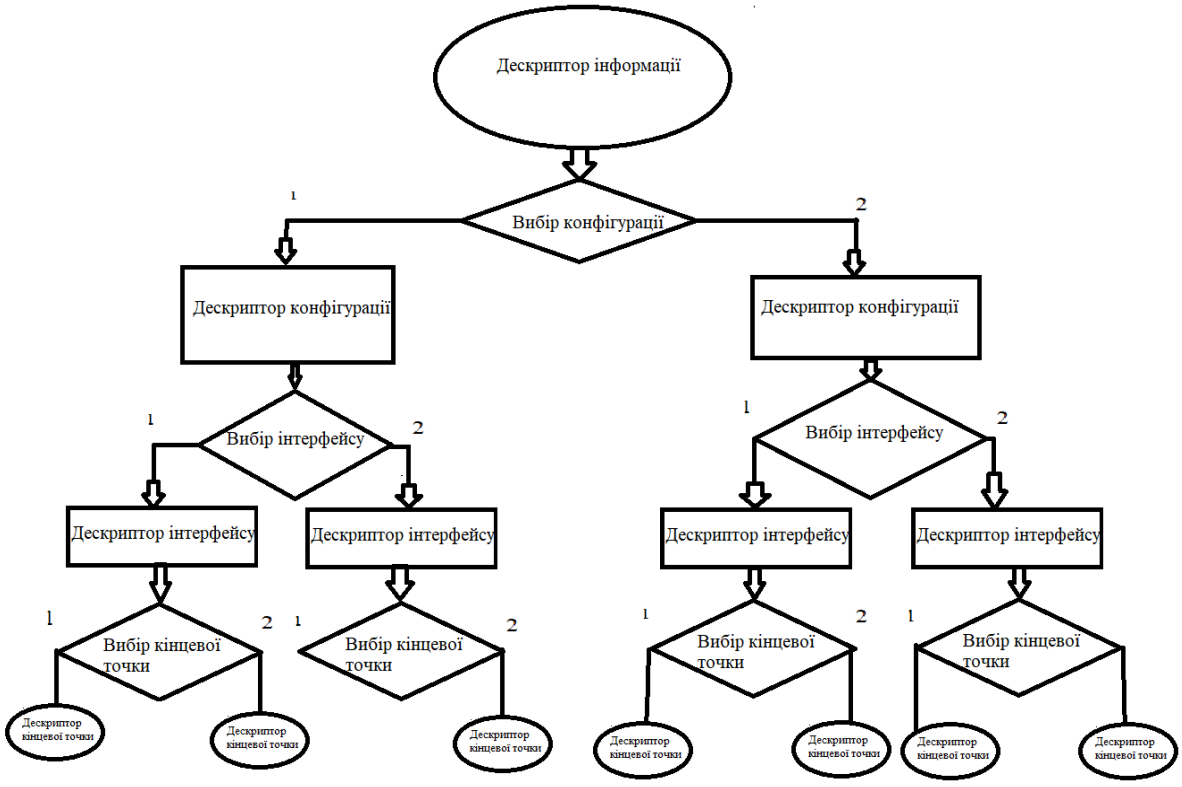


Рис. 4.2. Алгоритм ініціалізації пристрою з інтерфейсом USB

Після першої ініціалізації пристрою, комп’ютер запам’ятовує даний пристрій та зберігає інформацію у своєму регістрі. Користувачу після ініціалізації пристрою може перевірити пристрій на працездатність введенням інформації через натискання кнопки на пристрої. Наприклад, можна даний пристрій застосувати в якості засобу керування у грі, тоді гра буде відтворювати певну дію при натисненій кнопці. На рис. 4.3.2 зображено алгоритм роботи програмного забезпечення.

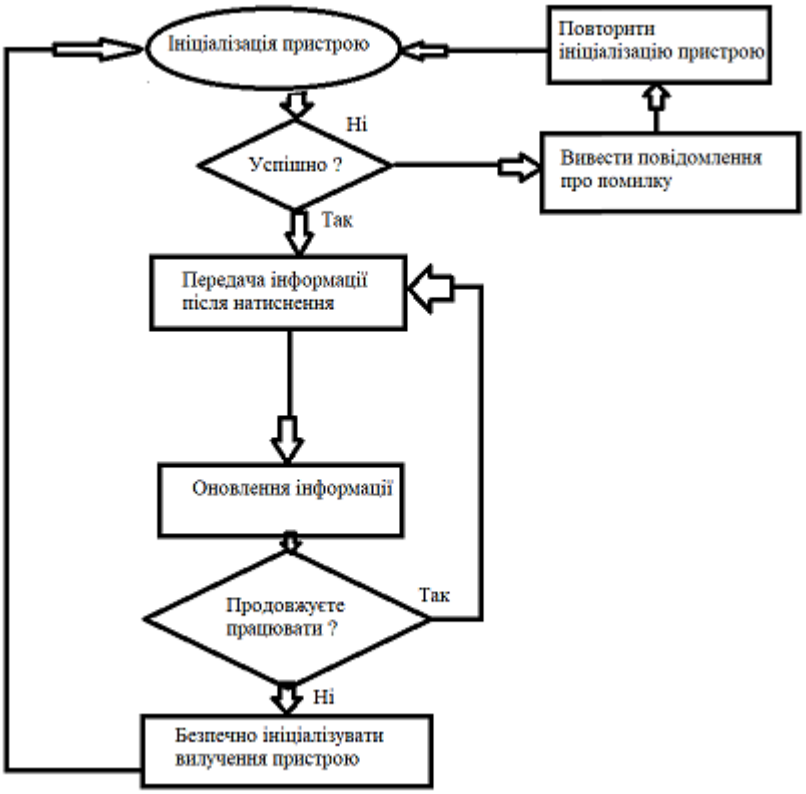


Рис. 4.3 Алгоритм роботи програмного забезпечення

4.4 Перевірка роботи пристрою

Спершу було створено схему електричну принципову інтерактивного пристрою для введення інформації в Proteus, яка зображена на Рисунок 4.4.

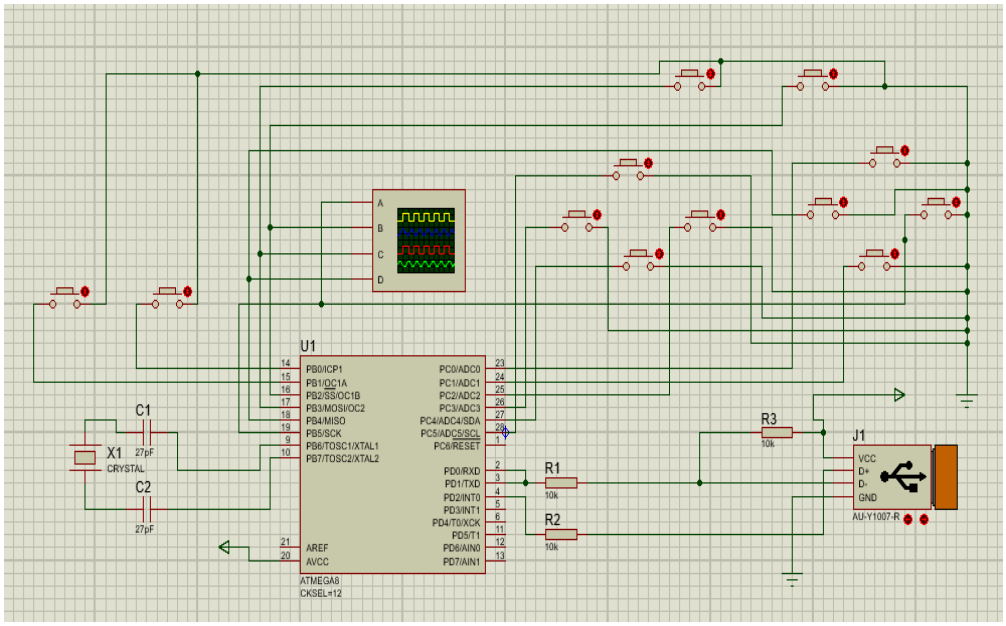


Рис. 4.4. Схема інтерактивного пристрою для введення інформації в Proteus

Після чого було перевірено роботу пристрою. Під час моделювання перевірялась при якому логійному рівні (0 чи 1) подається інфромація, яка проходить після натискання однієї з кнопок. Графік можна побачити на Рис.4.5. За отриманим результатом можемо зробити висновок, що пристрій працює коректно так як інформація подається при лог. 0.

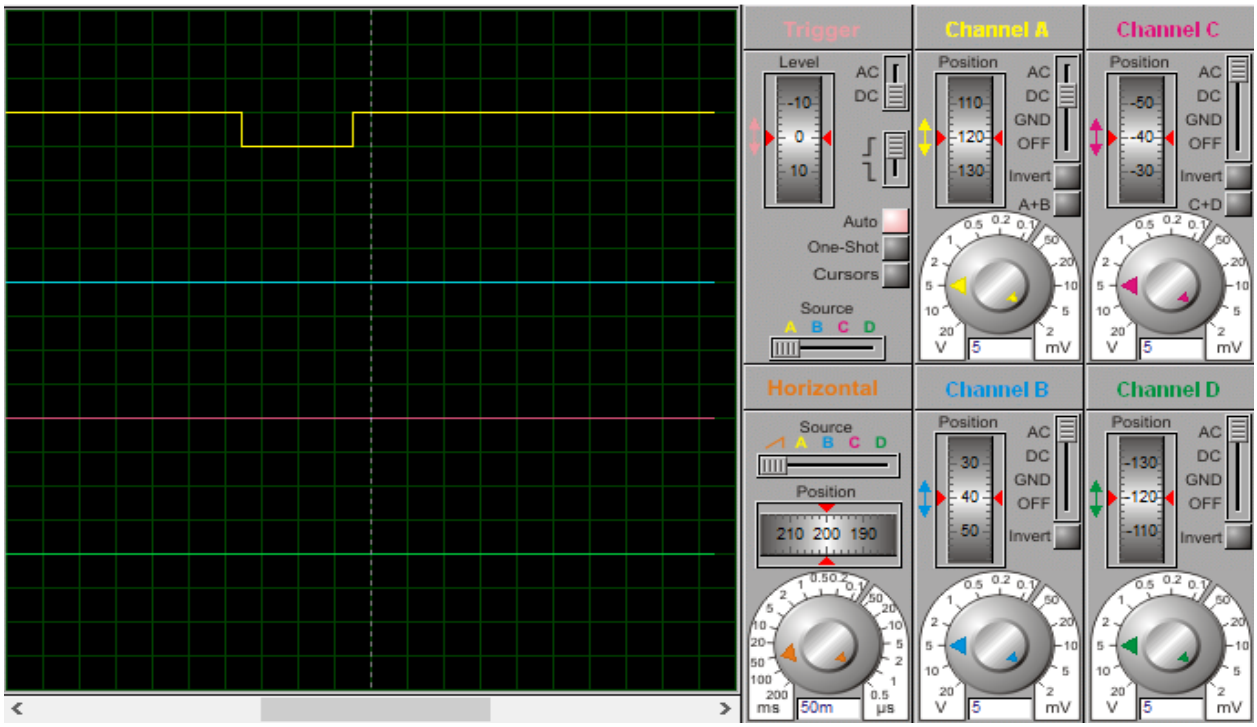


Рис. 4.5 Графік, що показує роботу пристрою

Висновок до розділу 4

В даному розділі описано за яким алгоритмом може користувач взаємодіяти з системою. Обґрунтовано вибір інтерфейсу передачі даних, що буде отримувати дані від мікроконтролера. Описано порти вводу-виводу, та їх режими роботи та на основі даної інформації було проведено налаштування портів. Також описано принцип, за яким працює пристрій. Наведена блок схема ілюструє роботу даного пристрою. Наведена симуляція пристрою, з якої робимо висновок, що пристрій працює коректно.

ВИСНОВКИ

В даному проекті було розроблено інтерактивний пристрій для введення інформації на базі мікроконтролера. Пристрій можемо використовувати у житловому приміщенні.

Було проведено аналіз ТЗ й здійснили пошук аналогів інтерактивних пристроїв для введення інформації. Аналіз ТЗ показав, що пристроїв для введення інформації є багата кількість тому наведено класифікацію та надано опис пристроїв для введення інформації. Обрано в якості об’єкту розробки – геймпад. Пошук аналогів показав, що існуючі прилади, або являють собою модифікацію існуючих пристроїв, або мають високу цінові показники.

Було запропоновано структуру проекту, що дало нам підставу на розробку схеми електричної структурної. Проведено вибір елементної бази, при цьому для важливих компонентів використано метод матриці параметрів. Структурна схема дозволила створити схему електричну принципову та на її основі провести певні розрахунки. Проведені розрахунки струму стабілізації для блоку передачі даних, він склав 21 мА. Також було проведено розрахунок обв'язки блоку генерації тактової частоти, обв’язкою слугують конденсатори, їх номінал склав 28 пФ.

Проведено конструкторський розрахунок друкованої плати. Обрано двошарову друковану плату. Матеріалом буде слугувати FR4-35-1,5. В якості методу виготовлення друкованої плати обрано позитивний комбінований метод. Для ДМ обрано 4 клас точності. На основі проведеного вибору розроблено проект в Altium Designer. Проведені розрахунки дають нам підстави вважати:

По-перше конструкторсько-технонологічний розрахунок підтвердив можливість виготовлення друкованої плати, з обраним методом із обраним класом точності. Даний розрахунок відповідає вимогм ТЗ.

По-друге електричний розрахунок друкованої плати показав, що отримана напруга втрат складає 0.0097 В, що значно нижче допустимої межі в

5 % від напруги живлення. Паразитна ємність (0,549 пФ) та індуктивність (0,0673 нГн) не впливає на роботу друкованого вузлу. Потужність втрат склала 54,25 нВт, що э не значною втратою. Отримані розрахунки відповідають вимогам ТЗ.

По-третє при розрахунку вібромічності плати визначено власну частоту коливань друкованої плати, яка склала 2575 Гц, що є більше чим 250 Гц та дає підстави вважати, що конструкція при спиранні на 4 сторони є жорсткою, що задовольняє ТЗ.

По-четверте розраховано надійність приладу. Вона склала 50 років, що набаго більше ніж зазначено у ТЗ, це пов’язано з тим що надійність розраховувалась у випадку раптових відмов.

Описано розробку програмного забезпечення. Розглянуто й подалі обґрунтовано вибір технології для передачі інформації. Наведено опис портів вводу-виводу та подальше їх налаштування. Описано принцип ініціалізації пристрою та наведено блок-схему, що наглядно показує, як проходить ініціалізація. На основі описаного принципу ініціалізації пристрою описано алгоритм роботи пристрою, для цього також створено блок-схему, що ілюструє роботу даного пристрою.

Отриманий пристрій відповідає усім поставленим вимогам ТЗ.
Наведені розрахунки та матеріали, що були наведені у проекті, задовольняють усім вимогам зазначеним у ТЗ. Також пристрій може бути покращеним, основою для цього може слугувати, додавання ще одного елемента керування, підтримка Bluetooth або Wi-Fi техонології

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Класифікація пристроїв для введення інформації [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D1%82%D1%80%D1%96%D0%B9_%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%BD%D1%8F.
2. Принцип роботи пристроїв для введення графічної інформації [Електроний ресурс] – Режим доступу: http://koi.tspu.ru/koi_books/skachkova/raz5%201.htm
3. Пристрої для введення інформації [Електроний ресурс] – Режим доступу:http://infoplaneta.ucoz.net/index/urok_18_ustrojstva_vvoda_prakticheskaja_rabota_8_rabotaem_s_graficheskimi_fragmentam/0-103
4. Тлумачний словник з інформатики / Півняк Г.Г., Бусигін Б.С. та ін. – Дніпропетровськ: Національний гірничий університет, 2010. - 607 с.
5. Circuit for USB game controller with 12 inputs (8 buttons + 4 directions)[Електронний ресурс] – Режим доступу: https://www.raphnet.net/electronique/usb_game12/index_en.php
6 Документація на AtMega8A-AU [Електронний ресурс] Режим доступу: https://pdf1.alldatasheet.com/datasheet-pdf/view/313647/ATMEL/ATmega8A-AU.html
7. Документація на LP-3.5S [Електронний ресурс] Режим доступу: https://www.rcscomponents.kiev.ua/datasheets/lp-35s-siward.pdf
8. Документація на AS-10.000-18-EXT-SMD [Електронний ресурс] Режим доступу: https://datasheet.octopart.com/AS-10.000-18-EXT-SMD-TR-Raltron-Electronics-datasheet-12509842.pdf
9. Документація на ABLS [Електронний ресурс] Режим доступу: https://www.tme.eu/Document/8138fc95be563aa64fed590b18efbe60/ABLS.pdf
10. Документація на BZV55 [Електронний ресурс] Режим доступу: http://www.kosmodrom.com.ua/pdf/BZV55.pdf

11. Документація на кнопку 1-1825027-1 [Електронний ресурс] Режим доступу: https://www.snapeda.com/parts/1-1825027-1/TE%20Connectivity/datasheet/

12. Розрахунок параметричного стабілізатора [Електронний ресурс] Режим доступу: https://radiohlam.ru/paramstab/

13. Розрахунок обв‘язки кварцового резонатору [Елекронний ресурс] Режим доступу: http://www.techstages.ru/setons-899-1.html

14. Губар В.Г. Курс лекцій по ФТОК [Електронний ресурс] Режим доступу: https://onedrive.live.com/?authkey=%21AIHNjiziB4gvyj0&id=D1785C298F1B017B%21244&cid=D1785C298F1B017B

15. С.М. Боровиков. Расчет показателей надежности радиоэлектронный средств: учеб.-метод. Пособие. – Минск: БГУИР,2010. – 68 с.

16. Розгляд бездротових технологій Bluetooth та Wi-Fi [Електронний ресурс] Режим доступу: https://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:Wi-Fi_(Wireless_Fidelity)_-_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82_%D0%B1%D0%B5%D1%81%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%BD%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D0%B8_802.11

17. Розгляд USB технології [Електронний ресурс] Режим доступу: https://www.beyondlogic.org/usbnutshell/usb1.shtml

18. Дескриптори USB пристроїв [Електронний ресурс] Режим доступу: https://www.beyondlogic.org/usbnutshell/usb5.shtml

19. Порти вводу-виводу та їх принцип роботи [Електронний ресурс] Режим доступу: http://easyelectronics.ru/avr-uchebnyj-kurs-ustrojstvo-i-rabota-portov-vvoda-vyvoda.html

ГОСТ 15150-69. Машины, приборы и другие технические изделия. Исполнения для различных климатических районов.

ГОСТ 23665-79. Платы печатные. Обработка контура. Требования к типовым технологическим процессам.

ГОСТ 23751-86 ПЛАТЫ ПЕЧАТНЫЕ Основные параметры конструкции.

ГОСТ 23770-86. Платы печатные. Типовые технологические процессы химической и гальванической метализации.

ГОСТ 2.417-9. Платы печатные. Правила выполнения чертежей.

ГОСТ 2.701-84 – ЕСКД. Правила выполнения схем.

ГОСТ 2.701-84 – ЕСКД. Правила выполнения электрических схем.

ГОСТ 2.701-84 – ЕСКД. Правила оформления схем.

ГОСТ 2.743-91 – ЕСКД. Элементы цифровой техники.

ГОСТ 2.759-82 – ЕСКД. Элементы аналоговой техники.

ОСТ 4.010.022-85. Платы печатные. Методы конструирования и расчета.

Додаток А

Технічне завдання

1. Найменування та галузь використання

Інтерактивний пристрій для введення інформації на базі мікроконтролера. Пристрій належить до галузі розваг. Може використовуватись для введення даний при натисненні однієї з керуючих кнопок.

2. Підстава для розробки

Підставою для розробки дипломного проекту є завдання, що було дано за наказом по КПІ ім. Ігоря Сікорського №1316-с від 24.05.2021.

3. Мета і призначення розробки

Проектування зручного інтерактивного пристрою для введення інформації. Основна задача – введення даний при натисненій кнопці.

4. Технічні вимоги

4.1. Склад виробу й вимоги до пристрою, що розробляється

Пристрій являє собою моноблочну конструкцію, у якості вхідної напруги береться напруга в ділянці кола. Не доцільно підключати прилад послідовно

4.2. Вимоги до конструкції

Габаритні розміри готового виробу повинні дозолити вмонтувати у готовий для нього корпус не більше: 130x35x25 мм

Вага пристрою повинна бути не більше 0.4 кг.

4.3. Вимоги до надійності

Середній час напрацювання на відмову повинен бути не менше 80000 год.

4.4. Вимоги до вібростійкості

Конструкція повинна бути жорсткою.

4.5. Вимоги безпеки обслуговування

Керуватися загальними вимогами безпеки до апаратури низької напруги ГОСТ 12.2.007-75.

4.6. Вимоги до складових частин виробу, сировини, вихідних й експлуатаційних матеріалів

Повинні використовуватись електронні компоненти, що знаходяться у легкому доступі.

Отримана конструкція повинна бути ремонтопридатною й забезпечувати заміннність компонентів.

4.7. Умови експлуатації

Кліматичне виконання УХЛ.4.2 по ГОСТ 15150-69.

4.8. Вимоги до транспортування і зберігання

Група умов зберігання Л1 згідно ГОСТ 15150-69. Зберігати у зачинених опалювальних вентильованих приміщеннях, при температурі повітря +5…+40°С, відносній вологості повітря 60% при 20°С (середньорічне значення) та атмосферному тиску 84,0…106,7кПа.

Транспортувати автомобільним, залізничним або авіаційним видами транспорту.

5. Вимоги до документації

Необхідна документація наводиться в пояснювальній записці до роботи. Додаткова документація не потрібна.

6. Робота повинна містити в собі документи

• Пояснювальну записку (формату А4, до 70 аркушів)

• Схему електричну принципову та перелік елементів (формату А2, А4 відповідно)

• Складальне креслення та специфікацію (формату А1, А4 відповідно)

• Креслення друкованої плати (формату А1)

• Алгоритм программи (формату А2)

• Креслення корпусу (формату А2)

7. Економічні показники

В умовах даного проекту не розглядаються.

Додаток Б

Акт впровадження результатів проектування

ЗАТВЕРДЖУЮ
ДП «Автоматор»
(посада керівника підприємства)
_____ П.С.
(підпис)
"09" червня 2021р.

АКТ

впровадження результатів дипломного проекту

виконаного студентом групи ДК-71

кафедри КЕОА, ФЕЛ, НТУУ «КПІ ім. Ігоря Сікорського»

Романенком Святославом Володимировичем

на тему «Інтерактивний пристрій для введення інформації на базі мікроконтролера»

Даний акт складено про те, що результати дипломного проекту «Інтерактивний пристрій для введення інформації на базі мікроконтролера» студента групи ДК-71, кафедри КОЕА, ФЕЛ, НТУУ «КПІ ім. Ігоря Сікорського» Романенка Святослава Володимировича використані для покращення існуючих компонентів та результатів.

ДП «Автоматор»

(підпис)

Дірінго П.С.
(ініціали, прізвище)

Лістинг Програми

Main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <string.h>

#include "usbdrv.h"
#include "oddebug.h"
#include "gamepad.h"

#include "twelve.h"

#include "devdesc.h"

static uchar *rt_usbHidReportDescriptor=NULL;
static uchar rt_usbHidReportDescriptorSize=0;
static uchar *rt_usbDeviceDescriptor=NULL;
static uchar rt_usbDeviceDescriptorSize=0;

PROGMEM int usbDescriptorStringSerialNumber[]  = {
        USB_STRING_DESCRIPTOR_HEADER(4),
        '1','0','0','0'
};

char usbDescriptorConfiguration[] = { 0 }; // dummy
```

```
uchar my_usbDescriptorConfiguration[] = {    /* USB configuration descriptor */
    9,          /* sizeof(usbDescriptorConfiguration): length of descriptor in bytes */
    USBDESCR_CONFIG,    /* descriptor type */
    18 + 7 * USB_CFG_HAVE_INTRIN_ENDPOINT + 9, 0,
                /* total length of data returned (including inlined descriptors) */
    1,          /* number of interfaces in this configuration */
    1,          /* index of this configuration */
    0,          /* configuration name string index */
#if USB_CFG_IS_SELF_POWERED
    USBATTR_SELFPOWER,  /* attributes */
#else
    USBATTR_BUSPOWER,   /* attributes */
#endif
    USB_CFG_MAX_BUS_POWER/2,            /* max USB current in 2mA units */
/* interface descriptor follows inline: */
    9,          /* sizeof(usbDescrInterface): length of descriptor in bytes */
    USBDESCR_INTERFACE, /* descriptor type */
    0,          /* index of this interface */
    0,          /* alternate setting for this interface */
    USB_CFG_HAVE_INTRIN_ENDPOINT,   /* endpoints excl 0: number of
endpoint descriptors to follow */
    USB_CFG_INTERFACE_CLASS,
    USB_CFG_INTERFACE_SUBCLASS,
    USB_CFG_INTERFACE_PROTOCOL,
    0,          /* string index for interface */
//#if (USB_CFG_DESCR_PROPS_HID & 0xff)    /* HID descriptor */
    9,          /* sizeof(usbDescrHID): length of descriptor in bytes */
    USBDESCR_HID,   /* descriptor type: HID */
```

```
    0x01, 0x01, /* BCD representation of HID version */
    0x00,       /* target country code */
    0x01,       /* number of HID Report (or other HID class) Descriptor infos to
follow */
    0x22,       /* descriptor type: report */
    USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH, 0,  /* total length of
report descriptor */
//#endif
#if USB_CFG_HAVE_INTRIN_ENDPOINT    /* endpoint descriptor for endpoint
1 */
    7,          /* sizeof(usbDescrEndpoint) */
    USBDESCR_ENDPOINT,  /* descriptor type = endpoint */
    0x81,       /* IN endpoint number 1 */
    0x03,       /* attrib: Interrupt endpoint */
    8, 0,       /* maximum packet size */
    USB_CFG_INTR_POLL_INTERVAL, /* in ms */
#endif
};

static Gamepad *curGamepad;



/* ------------------------ hardware I/O abstraction ------------------------ */

static void hardwareInit(void)
{
        uchar i, j;

        // init port C as input with pullup
```

```
DDRC = 0x00;
PORTC = 0xff;

/* 1101 1000 bin: activate pull-ups except on USB lines
 *
 * USB signals are on bit 0 and 2.
 *
 * Bit 1 is connected with bit 0 (rev.C pcb error), so the pullup
 * is not enabled.
 * */
PORTD = 0xf8;

/* Usb pin are init as outputs */
DDRD = 0x01 | 0x04;



j = 0;
while(--j){     /* USB Reset by device only required on Watchdog Reset */
        i = 0;
        while(--i); /* delay >10ms for USB reset */
}
DDRD = 0x00;    /* 0000 0000 bin: remove USB reset condition */
                /* configure timer 0 for a rate of 12M/(1024 * 256) = 45.78 Hz
(~22ms) */
TCCR0 = 5;      /* timer 0 prescaler: 1024 */

TCCR2 = (1<<WGM21)|(1<<CS22)|(1<<CS21)|(1<<CS20);
OCR2 = 196; // for 60 hz
```

```
}

static uchar    reportBuffer[6];    /* buffer for HID reports */



/* ------------------------------------------------------------------------- */
/* ---------------------------- USB interface ---------------------------- */
/* ------------------------------------------------------------------------- */

static uchar    idleRate;           /* in 4 ms units */

uchar usbFunctionDescriptor(struct usbRequest *rq)
{
        if ((rq->bmRequestType & USBRQ_TYPE_MASK) !=
USBRQ_TYPE_STANDARD)
                return 0;

        if (rq->bRequest == USBRQ_GET_DESCRIPTOR)
        {
                // USB spec 9.4.3, high byte is descriptor type
                switch (rq->wValue.bytes[1])
                {
                        case USBDESCR_DEVICE:
                                usbMsgPtr = rt_usbDeviceDescriptor;
                                return rt_usbDeviceDescriptorSize;
                        case USBDESCR_HID_REPORT:
                                usbMsgPtr = rt_usbHidReportDescriptor;
                                return rt_usbHidReportDescriptorSize;
```

```
                case USBDESCR_CONFIG:
                    usbMsgPtr = my_usbDescriptorConfiguration;
                    return sizeof(my_usbDescriptorConfiguration);
            }
        }

        return 0;
}

uchar usbFunctionSetup(uchar data[8])
{
        usbRequest_t    *rq = (void *)data;

        usbMsgPtr = reportBuffer;
        if((rq->bmRequestType & USBRQ_TYPE_MASK) ==
USBRQ_TYPE_CLASS){    /* class request type */
            if(rq->bRequest == USBRQ_HID_GET_REPORT){  /* wValue:
ReportType (highbyte), ReportID (lowbyte) */
                /* we only have one report type, so don't look at wValue */
                curGamepad->buildReport(reportBuffer);
                return curGamepad->report_size;
            }else if(rq->bRequest == USBRQ_HID_GET_IDLE){
                usbMsgPtr = &idleRate;
                return 1;
            }else if(rq->bRequest == USBRQ_HID_SET_IDLE){
                idleRate = rq->wValue.bytes[1];
            }
        }else{
        /* no vendor specific requests implemented */
```

```
        }
        return 0;
}

/* ---------------------------------------------------------------------- */


int main(void)
{
        char must_report = 0, first_run = 1;
        uchar   idleCounter = 0;
        int run_mode;

        // led pin as output
//      DDRD |= 0x20;

        /* Dip switch common: DB0, outputs: DB1 and DB2 */
        DDRB |= 0x01;
        DDRB &= ~0x06;

        PORTB |= 0x06; /* enable pull up on DB1 and DB2 */
        PORTB &= ~0x01; /* Set DB0 to low */

        _delay_ms(10); /* let pins settle */

        run_mode = (PINB & 0x06)>>1;

        curGamepad = twelveGetGamepad();
```

```
// configure report descriptor according to
// the current gamepad
rt_usbHidReportDescriptor = curGamepad->reportDescriptor;
rt_usbHidReportDescriptorSize = curGamepad->reportDescriptorSize;

if (curGamepad->deviceDescriptor != 0)
{
        rt_usbDeviceDescriptor = (void*)curGamepad->deviceDescriptor;
        rt_usbDeviceDescriptorSize = curGamepad->deviceDescriptorSize;
}
else
{
        // use descriptor from devdesc.c
        //
        rt_usbDeviceDescriptor = (void*)usbDescrDevice;
        rt_usbDeviceDescriptorSize = getUsbDescrDevice_size();
}

// patch the config descriptor with the HID report descriptor size
my_usbDescriptorConfiguration[25] = rt_usbHidReportDescriptorSize;

//wdt_enable(WDTO_2S);
hardwareInit();
curGamepad->init();
odDebugInit();
usbInit();
sei();
DBG1(0x00, 0, 0);
```

```
for(;;){        /* main event loop */
        wdt_reset();

        // this must be called at each 50 ms or less
        usbPoll();

        if (first_run) {
                curGamepad->update();
                first_run = 0;
        }

        if(TIFR & (1<<TOV0)){   /* 22 ms timer */
                TIFR = 1<<TOV0;
                if(idleRate != 0){
                        if(idleCounter > 4){
                                idleCounter -= 5;   /* 22 ms in units of 4 ms */
                        }else{
                                idleCounter = idleRate;
                                must_report = 1;
                        }
                }
        }

        if (TIFR & (1<<OCF2))
        {
                TIFR = 1<<OCF2;
                if (!must_report)
```

```
                {
                        curGamepad->update();
                        if (curGamepad->changed()) {
                                must_report = 1;
                        }
                }

        }


        if(must_report)
        {
                if (usbInterruptIsReady())
                {

                must_report = 0;

                curGamepad->buildReport(reportBuffer);
                usbSetInterrupt(reportBuffer, curGamepad->report_size);
                }
        }
    }
    return 0;
}

/* ------------------------------------------------------------------------- */
```

Twelve.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <string.h>
#include "gamepad.h"
#include "twelve.h"

#define REPORT_SIZE          3
#define GAMEPAD_BYTES   2

/* for chaning IO easily */



/********** prototypes *************/
static void twelveInit(void);
static void twelveUpdate(void);
static char twelveChanged(void);
static void twelveBuildReport(unsigned char *reportBuffer);



// report matching the most recent bytes from the controller
static unsigned char last_read_controller_bytes[REPORT_SIZE];
// the most recently reported bytes
static unsigned char last_reported_controller_bytes[REPORT_SIZE];

static void readController(unsigned char bits[2])
{
        bits[0] = PINC;
```

```
	bits[1] = PINB;
}

static void twelveInit(void)
{
	unsigned char sreg;
	sreg = SREG;
	cli();


	/*
	 * --- 10 button on multiuse2 pinout ---
	 * PC5: Up
	 * PC4: Down
	 * PC3: Left
	 * PC2: Right
	 *
	 * PC1: Button 0
	 * PC0: Button 1
	 * PB5: Button 2
	 * PB4: Button 3
	 * PB3: Button 4
	 * PB2: Button 5 (JP2)
	 * PB1: Button 6 (JP1)
	 * PB0: Button 7
	 */

	DDRB = 0; // all inputs
	PORTB |= 0xff; // all pullups enabled
```

```
	// all of portC input with pullups
	DDRC &= ~0x3F;
	PORTC |= 0x3F;

	twelveUpdate();

	SREG = sreg;
}



static void twelveUpdate(void)
{
	unsigned char data[2];
	int x=128,y=128;

	readController(data);

	/* Buttons are active low. Invert values. */
	data[0] = data[0] ^ 0xff;
	data[1] = data[1] ^ 0xff;

	if (data[0] & 0x20) { y = 0; } // up
	if (data[0] & 0x10) { y = 255; } //down
	if (data[0] & 0x08) { x = 0; }  // left
	if (data[0] & 0x04) { x = 255; } // right

	last_read_controller_bytes[0]=x;
```

```
        last_read_controller_bytes[1]=y;
        last_read_controller_bytes[2]=0;

        if (data[0] & 0x02) // btn 0
                last_read_controller_bytes[2] |= 0x01;
        if (data[0] & 0x01) // btn 1
                last_read_controller_bytes[2] |= 0x02;
        if (data[1] & 0x20) // btn 2
                last_read_controller_bytes[2] |= 0x04;
        if (data[1] & 0x10) // btn 3
                last_read_controller_bytes[2] |= 0x08;
        if (data[1] & 0x08) // btn 4
                last_read_controller_bytes[2] |= 0x10;
        if (data[1] & 0x04) // btn 5
                last_read_controller_bytes[2] |= 0x20;
        if (data[1] & 0x02) // btn 6
                last_read_controller_bytes[2] |= 0x40;
        if (data[1] & 0x01) // btn 7
                last_read_controller_bytes[2] |= 0x80;


}

static char twelveChanged(void)
{
        static int first = 1;
        if (first) { first = 0;  return 1; }

        return memcmp(last_read_controller_bytes,
```

```
                                last_reported_controller_bytes, REPORT_SIZE);
}

static void twelveBuildReport(unsigned char *reportBuffer)
{
        if (reportBuffer != NULL)
        {
                memcpy(reportBuffer, last_read_controller_bytes, REPORT_SIZE);
        }
        memcpy(last_reported_controller_bytes,
                        last_read_controller_bytes,
                        REPORT_SIZE);
}

#include "snes_descriptor.c"

Gamepad twelveGamepad = {
        report_size:          REPORT_SIZE,
        reportDescriptorSize:     sizeof(snes_usbHidReportDescriptor),
        init:                 twelveInit,
        update:               twelveUpdate,
        changed:              twelveChanged,
        buildReport:          twelveBuildReport
};

Gamepad *twelveGetGamepad(void)
{
        twelveGamepad.reportDescriptor = (void*)snes_usbHidReportDescriptor;
```

```
        return &twelveGamepad;
}
```

Gamepad.h

```
#ifndef _gamepad_h__
#define _gamepad_h__

typedef struct {
        // size of reports built by buildReport
        int report_size;

        int reportDescriptorSize;
        void *reportDescriptor; // must be in flash

        int deviceDescriptorSize; // if 0, use default
        void *deviceDescriptor; // must be in flash

        void (*init)(void);
        void (*update)(void);
        char (*changed)(void);
        void (*buildReport)(unsigned char *buf);
} Gamepad;

#endif // _gamepad_h__
```

Snes_descriptor.c

```
static const char snes_usbHidReportDescriptor[] PROGMEM = {
   0x05, 0x01,                  // USAGE_PAGE (Generic Desktop)
   0x09, 0x05,                  // USAGE (Game Pad)
   0xa1, 0x01,                  // COLLECTION (Application)
```

```
    0x09, 0x01,             //   USAGE (Pointer)
    0xa1, 0x00,             //   COLLECTION (Physical)
    0x09, 0x30,             //     USAGE (X)
    0x09, 0x31,             //     USAGE (Y)
    0x15, 0x00,             //   LOGICAL_MINIMUM (0)
    0x26, 0xff, 0x00,         //     LOGICAL_MAXIMUM (255)
    0x75, 0x08,             //   REPORT_SIZE (8)
    0x95, 0x02,             //   REPORT_COUNT (2)
    0x81, 0x02,             //   INPUT (Data,Var,Abs)
    0xc0,                 // END_COLLECTION
    0x05, 0x09,             // USAGE_PAGE (Button)
    0x19, 0x01,             //   USAGE_MINIMUM (Button 1)
    0x29, 0x08,             //   USAGE_MAXIMUM (Button 8)
    0x15, 0x00,             //   LOGICAL_MINIMUM (0)
    0x25, 0x01,             //   LOGICAL_MAXIMUM (1)
    0x75, 0x01,             // REPORT_SIZE (1)
    0x95, 0x08,             // REPORT_COUNT (8)
    0x81, 0x02,             // INPUT (Data,Var,Abs)
    0xc0                  // END_COLLECTION
};
```

Devdesc.c

```
#include "devdesc.h"
#include "usbconfig.h"
#include "usbdrv.h"

#define USBDESCR_DEVICE         1

const char usbDescrDevice[] PROGMEM = {    /* USB device descriptor */
```

```
    18,         /* sizeof(usbDescrDevice): length of descriptor in bytes */
    USBDESCR_DEVICE,    /* descriptor type */
    0x01, 0x01, /* USB version supported */
    USB_CFG_DEVICE_CLASS,
    USB_CFG_DEVICE_SUBCLASS,
    0,          /* protocol */
    8,          /* max packet size */
    USB_CFG_VENDOR_ID,  /* 2 bytes */
    USB_CFG_DEVICE_ID,  /* 2 bytes */
    USB_CFG_DEVICE_VERSION, /* 2 bytes */
        1,
        2,
        3,
//      USB_CFG_DESCR_PROPS_STRING_VENDOR != 0 ? 1 : 0,         /*
manufacturer string index */
//      USB_CFG_DESCR_PROPS_STRING_PRODUCT != 0 ? 2 : 0,        /*
product string index */
//      USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER != 0 ? 3 : 0,  /*
serial number string index */
    1,          /* number of configurations */
};

int getUsbDescrDevice_size(void) { return sizeof(usbDescrDevice); }
```

devdesc.h

```
#ifndef _devdesc_h__
#define _devdesc_h__

#include <avr/pgmspace.h>
```

```
extern const char usbDescrDevice[] PROGMEM;
int getUsbDescrDevice_size(void);

#endif // _devdesc_h__
```

Oddebug.h

```
#ifndef __oddebug_h_included__
#define __oddebug_h_included__

/*
General Description:
This module implements a function for debug logs on the serial line of the
AVR microcontroller. Debugging can be configured with the define
'DEBUG_LEVEL'. If this macro is not defined or defined to 0, all debugging
calls are no-ops. If it is 1, DBG1 logs will appear, but not DBG2. If it is
2, DBG1 and DBG2 logs will be printed.

A debug log consists of a label ('prefix') to indicate which debug log created
the output and a memory block to dump in hex ('data' and 'len').
*/


#ifndef F_CPU
#   define  F_CPU   12000000    /* 12 MHz */
#endif

/* make sure we have the UART defines: */
#include "iarcompat.h"
```

```
#ifndef __IAR_SYSTEMS_ICC__
#   include <avr/io.h>
#endif

#ifndef uchar
#   define  uchar   unsigned char
#endif

#if DEBUG_LEVEL > 0 && !(defined TXEN || defined TXEN0) /* no UART in
device */
#   warning "Debugging disabled because device has no UART"
#   undef   DEBUG_LEVEL
#endif

#ifndef DEBUG_LEVEL
#   define  DEBUG_LEVEL 0
#endif

/* ---------------------------------------------------------------------- */

#if DEBUG_LEVEL > 0
#   define  DBG1(prefix, data, len) odDebug(prefix, data, len)
#else
#   define  DBG1(prefix, data, len)
#endif

#if DEBUG_LEVEL > 1
#   define  DBG2(prefix, data, len) odDebug(prefix, data, len)
#else
```

```
#   define  DBG2(prefix, data, len)
#endif

/* ---------------------------------------------------------------------- */

#if DEBUG_LEVEL > 0
extern void odDebug(uchar prefix, uchar *data, uchar len);

/* Try to find our control registers; ATMEL likes to rename these */

#if defined UBRR
#   define  ODDBG_UBRR  UBRR
#elif defined UBRRL
#   define  ODDBG_UBRR  UBRRL
#elif defined UBRR0
#   define  ODDBG_UBRR  UBRR0
#elif defined UBRR0L
#   define  ODDBG_UBRR  UBRR0L
#endif

#if defined UCR
#   define  ODDBG_UCR   UCR
#elif defined UCSRB
#   define  ODDBG_UCR   UCSRB
#elif defined UCSR0B
#   define  ODDBG_UCR   UCSR0B
#endif

#if defined TXEN
```

```
#   define  ODDBG_TXEN  TXEN
#else
#   define  ODDBG_TXEN  TXEN0
#endif

#if defined USR
#   define  ODDBG_USR   USR
#elif defined UCSRA
#   define  ODDBG_USR   UCSRA
#elif defined UCSR0A
#   define  ODDBG_USR   UCSR0A
#endif

#if defined UDRE
#   define  ODDBG_UDRE  UDRE
#else
#   define  ODDBG_UDRE  UDRE0
#endif

#if defined UDR
#   define  ODDBG_UDR   UDR
#elif defined UDR0
#   define  ODDBG_UDR   UDR0
#endif

static inline void  odDebugInit(void)
{
   ODDBG_UCR |= (1<<ODDBG_TXEN);
   ODDBG_UBRR = F_CPU / (19200 * 16L) - 1;
```

```
}
#else
#   define odDebugInit()
#endif

/* ------------------------------------------------------------------------- */

#endif /* __oddebug_h_included__ */
```

Usbconfig.h(стандартна бібліотека)

```
#ifndef __usbconfig_h_included__
#define __usbconfig_h_included__

/*
General Description:
This file is an example configuration (with inline documentation) for the USB
driver. It configures AVR-USB for an ATMega8 with USB D+ connected to Port
D
bit 2 (which is also hardware interrupt 0) and USB D- to Port D bit 0. You may
wire the lines to any other port, as long as D+ is also wired to INT0.
To create your own usbconfig.h file, copy this file to the directory
containing "usbdrv" (that is your project firmware source directory) and
rename it to "usbconfig.h". Then edit it accordingly.
*/

/* ---------------------------- Hardware Config ---------------------------- */

#define USB_CFG_IOPORTNAME      D
/* This is the port where the USB bus is connected. When you configure it to
```

```
 * "B", the registers PORTB, PINB and DDRB will be used.
 */
#define USB_CFG_DMINUS_BIT      0
/* This is the bit number in USB_CFG_IOPORT where the USB D- line is
connected.
 * This may be any bit in the port.
 */
#define USB_CFG_DPLUS_BIT       2
/* This is the bit number in USB_CFG_IOPORT where the USB D+ line is
connected.
 * This may be any bit in the port. Please note that D+ must also be connected
 * to interrupt pin INT0!
 */
#define USB_CFG_CLOCK_KHZ       (F_CPU/1000)
/* Clock rate of the AVR in MHz. Legal values are 12000, 15000, 16000 or 16500.
 * The 16.5 MHz version of the code requires no crystal, it tolerates +/- 1%
 * deviation from the nominal frequency. All other rates require a precision
 * of 2000 ppm and thus a crystal!
 * Default if not specified: 12 MHz
 */

/* ---------------------- Optional Hardware Config ---------------------- */

/* #define USB_CFG_PULLUP_IOPORTNAME   D */
/* If you connect the 1.5k pullup resistor from D- to a port pin instead of
 * V+, you can connect and disconnect the device from firmware by calling
 * the macros usbDeviceConnect() and usbDeviceDisconnect() (see usbdrv.h).
 * This constant defines the port on which the pullup resistor is connected.
 */
```

```
/* #define USB_CFG_PULLUP_BIT         4 */
/* This constant defines the bit number in USB_CFG_PULLUP_IOPORT (defined
 * above) where the 1.5k pullup resistor is connected. See description
 * above for details.
 */

/* --------------------------- Functional Range ---------------------------- */

#define USB_CFG_HAVE_INTRIN_ENDPOINT    1
/* Define this to 1 if you want to compile a version with two endpoints: The
 * default control endpoint 0 and an interrupt-in endpoint 1.
 */
#define USB_CFG_HAVE_INTRIN_ENDPOINT3   0
/* Define this to 1 if you want to compile a version with three endpoints: The
 * default control endpoint 0, an interrupt-in endpoint 1 and an interrupt-in
 * endpoint 3. You must also enable endpoint 1 above.
 */
/* #define USB_INITIAL_DATATOKEN          USBPID_DATA0 */
/* The above macro defines the startup condition for data toggling on the
 * interrupt/bulk endpoints 1 and 3. Defaults to USBPID_DATA0.
 */
#define USB_CFG_IMPLEMENT_HALT         0
/* Define this to 1 if you also want to implement the ENDPOINT_HALT feature
 * for endpoint 1 (interrupt endpoint). Although you may not need this feature,
 * it is required by the standard. We have made it a config option because it
 * bloats the code considerably.
 */
#define USB_CFG_INTR_POLL_INTERVAL      10
/* If you compile a version with endpoint 1 (interrupt-in), this is the poll
```

```
 * interval. The value is in milliseconds and must not be less than 10 ms for
 * low speed devices.
 */
#define USB_CFG_IS_SELF_POWERED         0
/* Define this to 1 if the device has its own power supply. Set it to 0 if the
 * device is powered from the USB bus.
 */
#define USB_CFG_MAX_BUS_POWER           100
/* Set this variable to the maximum USB bus power consumption of your device.
 * The value is in milliamperes. [It will be divided by two since USB
 * communicates power requirements in units of 2 mA.]
 */
#define USB_CFG_IMPLEMENT_FN_WRITE      0
/* Set this to 1 if you want usbFunctionWrite() to be called for control-out
 * transfers. Set it to 0 if you don't need it and want to save a couple of
 * bytes.
 */
#define USB_CFG_IMPLEMENT_FN_READ       0
/* Set this to 1 if you need to send control replies which are generated
 * "on the fly" when usbFunctionRead() is called. If you only want to send
 * data from a static buffer, set it to 0 and return the data from
 * usbFunctionSetup(). This saves a couple of bytes.
 */
#define USB_CFG_IMPLEMENT_FN_WRITEOUT   0
/* Define this to 1 if you want to use interrupt-out (or bulk out) endpoint 1.
 * You must implement the function usbFunctionWriteOut() which receives all
 * interrupt/bulk data sent to endpoint 1.
 */
#define USB_CFG_HAVE_FLOWCONTROL        0
```

```
/* Define this to 1 if you want flowcontrol over USB data. See the definition
 * of the macros usbDisableAllRequests() and usbEnableAllRequests() in
 * usbdrv.h.
 */
/* #define USB_RX_USER_HOOK(data, len)     if(usbRxToken ==
(uchar)USBPID_SETUP) blinkLED(); */
/* This macro is a hook if you want to do unconventional things. If it is
 * defined, it's inserted at the beginning of received message processing.
 * If you eat the received message and don't want default processing to
 * proceed, do a return after doing your things. One possible application
 * (besides debugging) is to flash a status LED on each packet.
 */
#define USB_COUNT_SOF                   0
/* define this macro to 1 if you need the global variable "usbSofCount" which
 * counts SOF packets.
 */

/* ------------------------- Device Description -------------------------- */

#define  USB_CFG_VENDOR_ID       0x81, 0x17
/* USB vendor ID for the device, low byte first. If you have registered your
 * own Vendor ID, define it here. Otherwise you use obdev's free shared
 * VID/PID pair. Be sure to read USBID-License.txt for rules!
 * This template uses obdev's shared VID/PID pair for HIDs: 0x16c0/0x5df.
 * Use this VID/PID pair ONLY if you understand the implications!
 */
#define  USB_CFG_DEVICE_ID       0x96, 0x0a
/* This is the ID of the product, low byte first. It is interpreted in the
 * scope of the vendor ID. If you have registered your own VID with usb.org
```

```
 * or if you have licensed a PID from somebody else, define it here. Otherwise
 * you use obdev's free shared VID/PID pair. Be sure to read the rules in
 * USBID-License.txt!
 * This template uses obdev's shared VID/PID pair for HIDs: 0x16c0/0x5df.
 * Use this VID/PID pair ONLY if you understand the implications!
 */
#define USB_CFG_DEVICE_VERSION  0x00, 0x01
/* Version number of the device: Minor number first, then major number.
 */
#define USB_CFG_VENDOR_NAME     'r', 'a', 'p', 'h', 'n', 'e', 't', '.', 'n', 'e', 't'
#define USB_CFG_VENDOR_NAME_LEN 11
/* These two values define the vendor name returned by the USB device. The
name
 * must be given as a list of characters under single quotes. The characters
 * are interpreted as Unicode (UTF-16) entities.
 * If you don't want a vendor name string, undefine these macros.
 * ALWAYS define a vendor name containing your Internet domain name if you
use
 * obdev's free shared VID/PID pair. See the file USBID-License.txt for
 * details.
 */
#define USB_CFG_DEVICE_NAME     'U', 'S', 'B', '_', 'G', 'a', 'm', 'e', '1', '2'
#define USB_CFG_DEVICE_NAME_LEN 10
/* Same as above for the device name. If you don't want a device name, undefine
 * the macros. See the file USBID-License.txt before you assign a name if you
 * use a shared VID/PID.
 */
//#define USB_CFG_SERIAL_NUMBER   'N', 'o', 'n', 'e'
#define USB_CFG_SERIAL_NUMBER_LEN   4
```

```
/* Same as above for the serial number. If you don't want a serial number,
 * undefine the macros.
 * It may be useful to provide the serial number through other means than at
 * compile time. See the section about descriptor properties below for how
 * to fine tune control over USB descriptors such as the string descriptor
 * for the serial number.
 */
#define USB_CFG_DEVICE_CLASS        0
#define USB_CFG_DEVICE_SUBCLASS     0
/* See USB specification if you want to conform to an existing device class.
 */
#define USB_CFG_INTERFACE_CLASS     3   /* HID */
#define USB_CFG_INTERFACE_SUBCLASS  0
#define USB_CFG_INTERFACE_PROTOCOL  0
/* See USB specification if you want to conform to an existing device class or
 * protocol.
 * This template defines a HID class device. If you implement a vendor class
 * device, set USB_CFG_INTERFACE_CLASS to 0 and
USB_CFG_DEVICE_CLASS to 0xff.
 */
#define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH    0  /* total length
of report descriptor */
/* Define this to the length of the HID report descriptor, if you implement
 * an HID device. Otherwise don't define it or define it to 0.
 * Since this template defines a HID device, it must also specify a HID
 * report descriptor length. You must add a PROGMEM character array named
 * "usbHidReportDescriptor" to your code which contains the report descriptor.
 * Don't forget to keep the array and this define in sync!
 */
```

```
/* #define USB_PUBLIC static */
/* Use the define above if you #include usbdrv.c instead of linking against it.
 * This technique saves a couple of bytes in flash memory.
 */

/* ------------------- Fine Control over USB Descriptors ------------------- */
/* If you don't want to use the driver's default USB descriptors, you can
 * provide our own. These can be provided as (1) fixed length static data in
 * flash memory, (2) fixed length static data in RAM or (3) dynamically at
 * runtime in the function usbFunctionDescriptor(). See usbdrv.h for more
 * information about this function.
 * Descriptor handling is configured through the descriptor's properties. If
 * no properties are defined or if they are 0, the default descriptor is used.
 * Possible properties are:
 *   + USB_PROP_IS_DYNAMIC: The data for the descriptor should be fetched
 *     at runtime via usbFunctionDescriptor().
 *   + USB_PROP_IS_RAM: The data returned by usbFunctionDescriptor() or
found
 *     in static memory is in RAM, not in flash memory.
 *   + USB_PROP_LENGTH(len): If the data is in static memory (RAM or flash),
 *     the driver must know the descriptor's length. The descriptor itself is
 *     found at the address of a well known identifier (see below).
 * List of static descriptor names (must be declared PROGMEM if in flash):
 *   char usbDescriptorDevice[];
 *   char usbDescriptorConfiguration[];
 *   char usbDescriptorHidReport[];
 *   char usbDescriptorString0[];
 *   int usbDescriptorStringVendor[];
```

```
 *   int usbDescriptorStringDevice[];
 *   int usbDescriptorStringSerialNumber[];
 * Other descriptors can't be provided statically, they must be provided
 * dynamically at runtime.
 *
 * Descriptor properties are or-ed or added together, e.g.:
 * #define USB_CFG_DESCR_PROPS_DEVICE   (USB_PROP_IS_RAM |
USB_PROP_LENGTH(18))
 *
 * The following descriptors are defined:
 *   USB_CFG_DESCR_PROPS_DEVICE
 *   USB_CFG_DESCR_PROPS_CONFIGURATION
 *   USB_CFG_DESCR_PROPS_STRINGS
 *   USB_CFG_DESCR_PROPS_STRING_0
 *   USB_CFG_DESCR_PROPS_STRING_VENDOR
 *   USB_CFG_DESCR_PROPS_STRING_PRODUCT
 *   USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER
 *   USB_CFG_DESCR_PROPS_HID
 *   USB_CFG_DESCR_PROPS_HID_REPORT
 *   USB_CFG_DESCR_PROPS_UNKNOWN (for all descriptors not handled by
the driver)
 *
 */

#define USB_CFG_DESCR_PROPS_DEVICE
USB_PROP_IS_DYNAMIC
#define USB_CFG_DESCR_PROPS_CONFIGURATION
(USB_PROP_IS_DYNAMIC | USB_PROP_IS_RAM)
#define USB_CFG_DESCR_PROPS_STRINGS              0
```

```
#define USB_CFG_DESCR_PROPS_STRING_0            0
#define USB_CFG_DESCR_PROPS_STRING_VENDOR       0
#define USB_CFG_DESCR_PROPS_STRING_PRODUCT      0
#define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER
USB_PROP_LENGTH((6*2))
#define USB_CFG_DESCR_PROPS_HID                 0
#define USB_CFG_DESCR_PROPS_HID_REPORT
USB_PROP_IS_DYNAMIC
#define USB_CFG_DESCR_PROPS_UNKNOWN             0

/* ---------------------- Optional MCU Description ------------------------ */

/* The following configurations have working defaults in usbdrv.h. You
 * usually don't need to set them explicitly. Only if you want to run
 * the driver on a device which is not yet supported or with a compiler
 * which is not fully supported (such as IAR C) or if you use a differnt
 * interrupt than INT0, you may have to define some of these.
 */
/* #define USB_INTR_CFG            MCUCR */
/* #define USB_INTR_CFG_SET        ((1 << ISC00) | (1 << ISC01)) */
/* #define USB_INTR_CFG_CLR        0 */
/* #define USB_INTR_ENABLE         GIMSK */
/* #define USB_INTR_ENABLE_BIT     INT0 */
/* #define USB_INTR_PENDING        GIFR */
/* #define USB_INTR_PENDING_BIT    INTF0 */
/* #define USB_INTR_VECTOR         SIG_INTERRUPT0 */

#endif /* __usbconfig_h_included__ */
```

Usbdrv.h(стандартна бібліотека для USB драйвера)

```
#ifndef __usbdrv_h_included__
#define __usbdrv_h_included__
#include "usbconfig.h"
#include "iarcompat.h"

/*
Hardware Prerequisites:
=======================
USB lines D+ and D- MUST be wired to the same I/O port. We recommend that
D+
triggers the interrupt (best achieved by using INT0 for D+), but it is also
possible to trigger the interrupt from D-. If D- is used, interrupts are also
triggered by SOF packets. D- requires a pullup of 1.5k to +3.5V (and the device
must be powered at 3.5V) to identify as low-speed USB device. A pullup of
1M SHOULD be connected from D+ to +3.5V to prevent interference when no
USB
master is connected. We use D+ as interrupt source and not D- because it
does not trigger on keep-alive and RESET states.

As a compile time option, the 1.5k pullup resistor on D- can be made
switchable to allow the device to disconnect at will. See the definition of
usbDeviceConnect() and usbDeviceDisconnect() further down in this file.

Please adapt the values in usbconfig.h according to your hardware!

The device MUST be clocked at exactly 12 MHz, 15 MHz or 16 MHz
or at 16.5 MHz +/- 1%. See usbconfig-prototype.h for details.
```

Limitations:

==========

Robustness with respect to communication errors:

The driver assumes error-free communication. It DOES check for errors in the PID, but does NOT check bit stuffing errors, SE0 in middle of a byte, token CRC (5 bit) and data CRC (16 bit). CRC checks can not be performed due to timing constraints: We must start sending a reply within 7 bit times.

Bit stuffing and misplaced SE0 would have to be checked in real-time, but CPU performance does not permit that. The driver does not check Data0/Data1 toggling, but application software can implement the check.

Input characteristics:

Since no differential receiver circuit is used, electrical interference robustness may suffer. The driver samples only one of the data lines with an ordinary I/O pin's input characteristics. However, since this is only a low speed USB implementation and the specification allows for 8 times the bit rate over the same hardware, we should be on the safe side. Even the spec requires detection of asymmetric states at high bit rate for SE0 detection.

Number of endpoints:

The driver supports up to four endpoints: One control endpoint (endpoint 0), two interrupt-in (or bulk-in) endpoints (endpoint 1 and 3) and one interrupt-out (or bulk-out) endpoint (endpoint 1). Please note that the USB standard forbids bulk endpoints for low speed devices! Most operating systems allow them anyway, but the AVR will spend 90% of the CPU time in the USB interrupt polling for bulk data.

By default, only the control endpoint 0 is enabled. To get the other endpoints,

define USB_CFG_HAVE_INTRIN_ENDPOINT,
USB_CFG_HAVE_INTRIN_ENDPOINT3 and/or
USB_CFG_IMPLEMENT_FN_WRITEOUT respectively (see usbconfig-prototype.h for
details).

Maximum data payload:
Data payload of control in and out transfers may be up to 254 bytes. In order to accept payload data of out transfers, you need to implement 'usbFunctionWrite()'.

USB Suspend Mode supply current:
The USB standard limits power consumption to 500uA when the bus is in suspend mode. This is not a problem for self-powered devices since they don't need bus power anyway. Bus-powered devices can achieve this only by putting the CPU in sleep mode. The driver does not implement suspend handling by itself. However, the application may implement activity monitoring and wakeup from sleep. The host sends regular SE0 states on the bus to keep it active. These SE0 states can be detected by wiring the INT1 pin to D-. It is not necessary to enable the interrupt, checking the interrupt pending flag should suffice. Before entering sleep mode, the application should enable INT1 for a wakeup on the next bus activity.

Operation without an USB master:
The driver behaves neutral without connection to an USB master if D- reads as 1. To avoid spurious interrupts, we recommend a high impedance (e.g. 1M) pullup resistor on D+ (interrupt). If D- becomes statically 0, the driver may block in the interrupt routine.

Interrupt latency:
The application must ensure that the USB interrupt is not disabled for more than 25 cycles (this is for 12 MHz, faster clocks allow longer latency).
This implies that all interrupt routines must either be declared as "INTERRUPT" instead of "SIGNAL" (see "avr/signal.h") or that they are written in assembler with "sei" as the first instruction.

Maximum interrupt duration / CPU cycle consumption:
The driver handles all USB communication during the interrupt service routine. The routine will not return before an entire USB message is received and the reply is sent. This may be up to ca. 1200 cycles @ 12 MHz (= 100us) if the host conforms to the standard. The driver will consume CPU cycles for all USB messages, even if they address another (low-speed) device on the same bus.

*/

/* -- */
/* ------------------------- Module Interface ---------------------------- */
/* -- */

#define USBDRV_VERSION 20070919
/* This define uniquely identifies a driver version. It is a decimal number
 * constructed from the driver's release date in the form YYYYMMDD. If the
 * driver's behavior or interface changes, you can use this constant to
 * distinguish versions. If it is not defined, the driver's release date is
 * older than 2006-01-25.
 */

```
#ifndef USB_PUBLIC
#define USB_PUBLIC
#endif
/* USB_PUBLIC is used as declaration attribute for all functions exported by
 * the USB driver. The default is no attribute (see above). You may define it
 * to static either in usbconfig.h or from the command line if you include
 * usbdrv.c instead of linking against it. Including the C module of the driver
 * directly in your code saves a couple of bytes in flash memory.
 */

#ifndef __ASSEMBLER__
#ifndef uchar
#define uchar   unsigned char
#endif
#ifndef schar
#define schar   signed char
#endif
/* shortcuts for well defined 8 bit integer types */

struct usbRequest;  /* forward declaration */

USB_PUBLIC void usbInit(void);
/* This function must be called before interrupts are enabled and the main
 * loop is entered.
 */
USB_PUBLIC void usbPoll(void);
/* This function must be called at regular intervals from the main loop.
 * Maximum delay between calls is somewhat less than 50ms (USB timeout for
 * accepting a Setup message). Otherwise the device will not be recognized.
```

```
 * Please note that debug outputs through the UART take ~ 0.5ms per byte
 * at 19200 bps.
 */
extern uchar *usbMsgPtr;
/* This variable may be used to pass transmit data to the driver from the
 * implementation of usbFunctionWrite(). It is also used internally by the
 * driver for standard control requests.
 */
USB_PUBLIC uchar usbFunctionSetup(uchar data[8]);
/* This function is called when the driver receives a SETUP transaction from
 * the host which is not answered by the driver itself (in practice: class and
 * vendor requests). All control transfers start with a SETUP transaction where
 * the host communicates the parameters of the following (optional) data
 * transfer. The SETUP data is available in the 'data' parameter which can
 * (and should) be casted to 'usbRequest_t *' for a more user-friendly access
 * to parameters.
 *
 * If the SETUP indicates a control-in transfer, you should provide the
 * requested data to the driver. There are two ways to transfer this data:
 * (1) Set the global pointer 'usbMsgPtr' to the base of the static RAM data
 * block and return the length of the data in 'usbFunctionSetup()'. The driver
 * will handle the rest. Or (2) return 0xff in 'usbFunctionSetup()'. The driver
 * will then call 'usbFunctionRead()' when data is needed. See the
 * documentation for usbFunctionRead() for details.
 *
 * If the SETUP indicates a control-out transfer, the only way to receive the
 * data from the host is through the 'usbFunctionWrite()' call. If you
 * implement this function, you must return 0xff in 'usbFunctionSetup()' to
 * indicate that 'usbFunctionWrite()' should be used. See the documentation of
```

```
 * this function for more information. If you just want to ignore the data sent
 * by the host, return 0 in 'usbFunctionSetup()'.
 *
 * Note that calls to the functions usbFunctionRead() and usbFunctionWrite()
 * are only done if enabled by the configuration in usbconfig.h.
 */
USB_PUBLIC uchar usbFunctionDescriptor(struct usbRequest *rq);
/* You need to implement this function ONLY if you provide USB descriptors at
 * runtime (which is an expert feature). It is very similar to
 * usbFunctionSetup() above, but it is called only to request USB descriptor
 * data. See the documentation of usbFunctionSetup() above for more info.
 */
#if USB_CFG_HAVE_INTRIN_ENDPOINT
USB_PUBLIC void usbSetInterrupt(uchar *data, uchar len);
/* This function sets the message which will be sent during the next interrupt
 * IN transfer. The message is copied to an internal buffer and must not exceed
 * a length of 8 bytes. The message may be 0 bytes long just to indicate the
 * interrupt status to the host.
 * If you need to transfer more bytes, use a control read after the interrupt.
 */
extern volatile uchar usbTxLen1;
#define usbInterruptIsReady()   (usbTxLen1 & 0x10)
/* This macro indicates whether the last interrupt message has already been
 * sent. If you set a new interrupt message before the old was sent, the
 * message already buffered will be lost.
 */
#if USB_CFG_HAVE_INTRIN_ENDPOINT3
USB_PUBLIC void usbSetInterrupt3(uchar *data, uchar len);
extern volatile uchar usbTxLen3;
```

```
#define usbInterruptIsReady3()   (usbTxLen3 & 0x10)
/* Same as above for endpoint 3 */
#endif
#endif /* USB_CFG_HAVE_INTRIN_ENDPOINT */
#if USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH    /* simplified interface for backward compatibility */
#define usbHidReportDescriptor  usbDescriptorHidReport
/* should be declared as: PROGMEM char usbHidReportDescriptor[]; */
/* If you implement an HID device, you need to provide a report descriptor.
 * The HID report descriptor syntax is a bit complex. If you understand how
 * report descriptors are constructed, we recommend that you use the HID
 * Descriptor Tool from usb.org, see http://www.usb.org/developers/hidpage/.
 * Otherwise you should probably start with a working example.
 */
#endif  /* USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH */
#if USB_CFG_IMPLEMENT_FN_WRITE
USB_PUBLIC uchar usbFunctionWrite(uchar *data, uchar len);
/* This function is called by the driver to provide a control transfer's
 * payload data (control-out). It is called in chunks of up to 8 bytes. The
 * total count provided in the current control transfer can be obtained from
 * the 'length' property in the setup data. If an error occurred during
 * processing, return 0xff (== -1). The driver will answer the entire transfer
 * with a STALL token in this case. If you have received the entire payload
 * successfully, return 1. If you expect more data, return 0. If you don't
 * know whether the host will send more data (you should know, the total is
 * provided in the usbFunctionSetup() call!), return 1.
 * NOTE: If you return 0xff for STALL, 'usbFunctionWrite()' may still be called
 * for the remaining data. You must continue to return 0xff for STALL in these
 * calls.
```

```
 * In order to get usbFunctionWrite() called, define
USB_CFG_IMPLEMENT_FN_WRITE
 * to 1 in usbconfig.h and return 0xff in usbFunctionSetup()..
 */
#endif /* USB_CFG_IMPLEMENT_FN_WRITE */
#if USB_CFG_IMPLEMENT_FN_READ
USB_PUBLIC uchar usbFunctionRead(uchar *data, uchar len);
/* This function is called by the driver to ask the application for a control
 * transfer's payload data (control-in). It is called in chunks of up to 8
 * bytes each. You should copy the data to the location given by 'data' and
 * return the actual number of bytes copied. If you return less than requested,
 * the control-in transfer is terminated. If you return 0xff, the driver aborts
 * the transfer with a STALL token.
 * In order to get usbFunctionRead() called, define
USB_CFG_IMPLEMENT_FN_READ
 * to 1 in usbconfig.h and return 0xff in usbFunctionSetup()..
 */
#endif /* USB_CFG_IMPLEMENT_FN_READ */
#if USB_CFG_IMPLEMENT_FN_WRITEOUT
USB_PUBLIC void usbFunctionWriteOut(uchar *data, uchar len);
/* This function is called by the driver when data on interrupt-out or bulk-
 * out endpoint 1 is received. You must define
USB_CFG_IMPLEMENT_FN_WRITEOUT
 * to 1 in usbconfig.h to get this function called.
 */
#endif /* USB_CFG_IMPLEMENT_FN_WRITEOUT */
#ifdef USB_CFG_PULLUP_IOPORTNAME
#define usbDeviceConnect()      ((USB_PULLUP_DDR |=
(1<<USB_CFG_PULLUP_BIT)), \
```

```
                    (USB_PULLUP_OUT |= (1<<USB_CFG_PULLUP_BIT)))
/* This macro (intended to look like a function) connects the device to the
 * USB bus. It is only available if you have defined the constants
 * USB_CFG_PULLUP_IOPORT and USB_CFG_PULLUP_BIT in usbconfig.h.
 */
#define usbDeviceDisconnect()   ((USB_PULLUP_DDR &=
~(1<<USB_CFG_PULLUP_BIT)), \
                    (USB_PULLUP_OUT &=
~(1<<USB_CFG_PULLUP_BIT)))
/* This macro (intended to look like a function) disconnects the device from
 * the USB bus. It is only available if you have defined the constants
 * USB_CFG_PULLUP_IOPORT and USB_CFG_PULLUP_BIT in usbconfig.h.
 */
#endif /* USB_CFG_PULLUP_IOPORT */
extern unsigned usbCrc16(unsigned data, uchar len);
#define usbCrc16(data, len) usbCrc16((unsigned)(data), len)
/* This function calculates the binary complement of the data CRC used in
 * USB data packets. The value is used to build raw transmit packets.
 * You may want to use this function for data checksums or to verify received
 * data. We enforce 16 bit calling conventions for compatibility with IAR's
 * tiny memory model.
 */
extern unsigned usbCrc16Append(unsigned data, uchar len);
#define usbCrc16Append(data, len)    usbCrc16Append((unsigned)(data), len)
/* This function is equivalent to usbCrc16() above, except that it appends
 * the 2 bytes CRC (lowbyte first) in the 'data' buffer after reading 'len'
 * bytes.
 */
extern uchar    usbConfiguration;
```

```
/* This value contains the current configuration set by the host. The driver
 * allows setting and querying of this variable with the USB
SET_CONFIGURATION
 * and GET_CONFIGURATION requests, but does not use it otherwise.
 * You may want to reflect the "configured" status with a LED on the device or
 * switch on high power parts of the circuit only if the device is configured.
 */
#if USB_COUNT_SOF
extern volatile uchar   usbSofCount;
/* This variable is incremented on every SOF packet. It is only available if
 * the macro USB_COUNT_SOF is defined to a value != 0.
 */
#endif

#define USB_STRING_DESCRIPTOR_HEADER(stringLength)
((2*(stringLength)+2) | (3<<8))
/* This macro builds a descriptor header for a string descriptor given the
 * string's length. See usbdrv.c for an example how to use it.
 */
#if USB_CFG_HAVE_FLOWCONTROL
extern volatile schar   usbRxLen;
#define usbDisableAllRequests()     usbRxLen = -1
/* Must be called from usbFunctionWrite(). This macro disables all data input
 * from the USB interface. Requests from the host are answered with a NAK
 * while they are disabled.
 */
#define usbEnableAllRequests()      usbRxLen = 0
/* May only be called if requests are disabled. This macro enables input from
 * the USB interface after it has been disabled with usbDisableAllRequests().
```

```
 */
#define usbAllRequestsAreDisabled() (usbRxLen < 0)
/* Use this macro to find out whether requests are disabled. It may be needed
 * to ensure that usbEnableAllRequests() is never called when requests are
 * enabled.
 */
#endif

#define USB_SET_DATATOKEN1(token)   usbTxBuf1[0] = token
#define USB_SET_DATATOKEN3(token)   usbTxBuf3[0] = token
/* These two macros can be used by application software to reset data toggling
 * for interrupt-in endpoints 1 and 3.
 */

#endif  /* __ASSEMBLER__ */


/* ------------------------------------------------------------------------- */
/* ----------------- Definitions for Descriptor Properties ----------------- */
/* ------------------------------------------------------------------------- */
/* This is advanced stuff. See usbconfig-prototype.h for more information
 * about the various methods to define USB descriptors. If you do nothing,
 * the default descriptors will be used.
 */
#define USB_PROP_IS_DYNAMIC     (1 << 8)
/* If this property is set for a descriptor, usbFunctionDescriptor() will be
 * used to obtain the particular descriptor.
 */
#define USB_PROP_IS_RAM         (1 << 9)
```

```
/* If this property is set for a descriptor, the data is read from RAM
 * memory instead of Flash. The property is used for all methods to provide
 * external descriptors.
 */
#define USB_PROP_LENGTH(len)    ((len) & 0xff)
/* If a static external descriptor is used, this is the total length of the
 * descriptor in bytes.
 */

/* all descriptors which may have properties: */
#ifndef USB_CFG_DESCR_PROPS_DEVICE
#define USB_CFG_DESCR_PROPS_DEVICE              0
#endif
#ifndef USB_CFG_DESCR_PROPS_CONFIGURATION
#define USB_CFG_DESCR_PROPS_CONFIGURATION       0
#endif
#ifndef USB_CFG_DESCR_PROPS_STRINGS
#define USB_CFG_DESCR_PROPS_STRINGS             0
#endif
#ifndef USB_CFG_DESCR_PROPS_STRING_0
#define USB_CFG_DESCR_PROPS_STRING_0            0
#endif
#ifndef USB_CFG_DESCR_PROPS_STRING_VENDOR
#define USB_CFG_DESCR_PROPS_STRING_VENDOR       0
#endif
#ifndef USB_CFG_DESCR_PROPS_STRING_PRODUCT
#define USB_CFG_DESCR_PROPS_STRING_PRODUCT      0
#endif
#ifndef USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER
```

```
#define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER    0
#endif
#ifndef USB_CFG_DESCR_PROPS_HID
#define USB_CFG_DESCR_PROPS_HID                     0
#endif
#if !(USB_CFG_DESCR_PROPS_HID_REPORT)
#   undef USB_CFG_DESCR_PROPS_HID_REPORT
#   if USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH /* do some backward
compatibility tricks */
#       define USB_CFG_DESCR_PROPS_HID_REPORT
USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH
#   else
#       define USB_CFG_DESCR_PROPS_HID_REPORT       0
#   endif
#endif
#ifndef USB_CFG_DESCR_PROPS_UNKNOWN
#define USB_CFG_DESCR_PROPS_UNKNOWN                 0
#endif

/* ------------------- forward declaration of descriptors -------------------- */
/* If you use external static descriptors, they must be stored in global
 * arrays as declared below:
 */
#ifndef __ASSEMBLER__
extern
#if !(USB_CFG_DESCR_PROPS_DEVICE & USB_PROP_IS_RAM)
PROGMEM
#endif
char usbDescriptorDevice[];
```

```
extern
#if !(USB_CFG_DESCR_PROPS_CONFIGURATION & USB_PROP_IS_RAM)
PROGMEM
#endif
char usbDescriptorConfiguration[];

extern
#if !(USB_CFG_DESCR_PROPS_HID_REPORT & USB_PROP_IS_RAM)
PROGMEM
#endif
char usbDescriptorHidReport[];

extern
#if !(USB_CFG_DESCR_PROPS_STRING_0 & USB_PROP_IS_RAM)
PROGMEM
#endif
char usbDescriptorString0[];

extern
#if !(USB_CFG_DESCR_PROPS_STRING_VENDOR & USB_PROP_IS_RAM)
PROGMEM
#endif
int usbDescriptorStringVendor[];

extern
#if !(USB_CFG_DESCR_PROPS_STRING_PRODUCT &
USB_PROP_IS_RAM)
PROGMEM
```

```
#endif
int usbDescriptorStringDevice[];

extern
#if !(USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER &
USB_PROP_IS_RAM)
PROGMEM
#endif
int usbDescriptorStringSerialNumber[];

#endif /* __ASSEMBLER__ */

/* ------------------------------------------------------------------------- */
/* ------------------------ General Purpose Macros ------------------------- */
/* ------------------------------------------------------------------------- */

#define USB_CONCAT(a, b)            a ## b
#define USB_CONCAT_EXPANDED(a, b)   USB_CONCAT(a, b)

#define USB_OUTPORT(name)           USB_CONCAT(PORT, name)
#define USB_INPORT(name)            USB_CONCAT(PIN, name)
#define USB_DDRPORT(name)           USB_CONCAT(DDR, name)
/* The double-define trick above lets us concatenate strings which are
 * defined by macros.
 */

/* ------------------------------------------------------------------------- */
/* ------------------------ Constant definitions ---------------------------- */
/* ------------------------------------------------------------------------- */
```

```
#if !defined __ASSEMBLER__ && (!defined USB_CFG_VENDOR_ID ||
!defined USB_CFG_DEVICE_ID)
#warning "You should define USB_CFG_VENDOR_ID and
USB_CFG_DEVICE_ID in usbconfig.h"
/* If the user has not defined IDs, we default to obdev's free IDs.
 * See USBID-License.txt for details.
 */
#endif

/* make sure we have a VID and PID defined, byte order is lowbyte, highbyte */
#ifndef USB_CFG_VENDOR_ID
#   define  USB_CFG_VENDOR_ID   0xc0, 0x16  /* 5824 in dec, stands for VOTI
*/
#endif

#ifndef USB_CFG_DEVICE_ID
#   if USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH
#       define USB_CFG_DEVICE_ID    0xdf, 0x05  /* 1503 in dec, shared PID for
HIDs */
#   elif USB_CFG_INTERFACE_CLASS == 2
#       define USB_CFG_DEVICE_ID    0xe1, 0x05  /* 1505 in dec, shared PID for
CDC Modems */
#   else
#       define USB_CFG_DEVICE_ID    0xdc, 0x05  /* 1500 in dec, obdev's free
PID */
#   endif
#endif
```

```
/* Derive Output, Input and DataDirection ports from port names */
#ifndef USB_CFG_IOPORTNAME
#error "You must define USB_CFG_IOPORTNAME in usbconfig.h, see
usbconfig-prototype.h"
#endif

#define USBOUT          USB_OUTPORT(USB_CFG_IOPORTNAME)
#define USB_PULLUP_OUT
USB_OUTPORT(USB_CFG_PULLUP_IOPORTNAME)
#define USBIN           USB_INPORT(USB_CFG_IOPORTNAME)
#define USBDDR          USB_DDRPORT(USB_CFG_IOPORTNAME)
#define USB_PULLUP_DDR
USB_DDRPORT(USB_CFG_PULLUP_IOPORTNAME)

#define USBMINUS    USB_CFG_DMINUS_BIT
#define USBPLUS     USB_CFG_DPLUS_BIT
#define USBIDLE     (1<<USB_CFG_DMINUS_BIT) /* value representing J state
*/
#define USBMASK     ((1<<USB_CFG_DPLUS_BIT) |
(1<<USB_CFG_DMINUS_BIT))  /* mask for USB I/O bits */

/* defines for backward compatibility with older driver versions: */
#define USB_CFG_IOPORT
USB_OUTPORT(USB_CFG_IOPORTNAME)
#ifdef USB_CFG_PULLUP_IOPORTNAME
#define USB_CFG_PULLUP_IOPORT
USB_OUTPORT(USB_CFG_PULLUP_IOPORTNAME)
#endif
```

```
#define USB_BUFSIZE     11  /* PID, 8 bytes data, 2 bytes CRC */

/* ----- Try to find registers and bits responsible for ext interrupt 0 ----- */

#ifndef USB_INTR_CFG    /* allow user to override our default */
#   if defined  EICRA
#       define USB_INTR_CFG EICRA
#   else
#       define USB_INTR_CFG MCUCR
#   endif
#endif
#ifndef USB_INTR_CFG_SET    /* allow user to override our default */
#   define USB_INTR_CFG_SET ((1 << ISC00) | (1 << ISC01))    /* cfg for rising
edge */
#endif
#ifndef USB_INTR_CFG_CLR    /* allow user to override our default */
#   define USB_INTR_CFG_CLR 0    /* no bits to clear */
#endif

#ifndef USB_INTR_ENABLE     /* allow user to override our default */
#   if defined GIMSK
#       define USB_INTR_ENABLE  GIMSK
#   elif defined EIMSK
#       define USB_INTR_ENABLE  EIMSK
#   else
#       define USB_INTR_ENABLE  GICR
#   endif
#endif
```

```
#ifndef USB_INTR_ENABLE_BIT /* allow user to override our default */
#   define USB_INTR_ENABLE_BIT  INT0
#endif

#ifndef USB_INTR_PENDING    /* allow user to override our default */
#   if defined  EIFR
#       define USB_INTR_PENDING EIFR
#   else
#       define USB_INTR_PENDING GIFR
#   endif
#endif
#ifndef USB_INTR_PENDING_BIT    /* allow user to override our default */
#   define USB_INTR_PENDING_BIT INTF0
#endif

/*
The defines above don't work for the following chips
at90c8534: no ISC0?, no PORTB, can't find a data sheet
at86rf401: no PORTB, no MCUCR etc, low clock rate
atmega103: no ISC0? (maybe omission in header, can't find data sheet)
atmega603: not defined in avr-libc
at43usb320, at43usb355, at76c711: have USB anyway
at94k: is different...

at90s1200, attiny11, attiny12, attiny15, attiny28: these have no RAM
*/

/* ------------------------------------------------------------------------- */
/* ----------------- USB Specification Constants and Types ----------------- */
```

```
/* ------------------------------------------------------------------------- */

/* USB Token values */
#define USBPID_SETUP    0x2d
#define USBPID_OUT      0xe1
#define USBPID_IN       0x69
#define USBPID_DATA0    0xc3
#define USBPID_DATA1    0x4b

#define USBPID_ACK      0xd2
#define USBPID_NAK      0x5a
#define USBPID_STALL    0x1e

#ifndef USB_INITIAL_DATATOKEN
#define USB_INITIAL_DATATOKEN   USBPID_DATA0
#endif

#ifndef __ASSEMBLER__

extern uchar    usbTxBuf1[USB_BUFSIZE], usbTxBuf3[USB_BUFSIZE];

typedef union usbWord{
    unsigned    word;
    uchar       bytes[2];
}usbWord_t;

typedef struct usbRequest{
    uchar       bmRequestType;
    uchar       bRequest;
```

```
    usbWord_t   wValue;
    usbWord_t   wIndex;
    usbWord_t   wLength;
}usbRequest_t;
/* This structure matches the 8 byte setup request */
#endif

/* bmRequestType field in USB setup:
 * d t t r r r r r, where
 * d ..... direction: 0=host->device, 1=device->host
 * t ..... type: 0=standard, 1=class, 2=vendor, 3=reserved
 * r ..... recipient: 0=device, 1=interface, 2=endpoint, 3=other
 */

/* USB setup recipient values */
#define USBRQ_RCPT_MASK         0x1f
#define USBRQ_RCPT_DEVICE       0
#define USBRQ_RCPT_INTERFACE    1
#define USBRQ_RCPT_ENDPOINT     2

/* USB request type values */
#define USBRQ_TYPE_MASK         0x60
#define USBRQ_TYPE_STANDARD     (0<<5)
#define USBRQ_TYPE_CLASS        (1<<5)
#define USBRQ_TYPE_VENDOR       (2<<5)

/* USB direction values: */
#define USBRQ_DIR_MASK              0x80
#define USBRQ_DIR_HOST_TO_DEVICE    (0<<7)
```

```
#define USBRQ_DIR_DEVICE_TO_HOST    (1<<7)

/* USB Standard Requests */
#define USBRQ_GET_STATUS        0
#define USBRQ_CLEAR_FEATURE     1
#define USBRQ_SET_FEATURE       3
#define USBRQ_SET_ADDRESS       5
#define USBRQ_GET_DESCRIPTOR    6
#define USBRQ_SET_DESCRIPTOR    7
#define USBRQ_GET_CONFIGURATION 8
#define USBRQ_SET_CONFIGURATION 9
#define USBRQ_GET_INTERFACE     10
#define USBRQ_SET_INTERFACE     11
#define USBRQ_SYNCH_FRAME       12

/* USB descriptor constants */
#define USBDESCR_DEVICE         1
#define USBDESCR_CONFIG         2
#define USBDESCR_STRING         3
#define USBDESCR_INTERFACE      4
#define USBDESCR_ENDPOINT       5
#define USBDESCR_HID            0x21
#define USBDESCR_HID_REPORT     0x22
#define USBDESCR_HID_PHYS       0x23

#define USBATTR_BUSPOWER        0x80
#define USBATTR_SELFPOWER       0x40
#define USBATTR_REMOTEWAKE      0x20
```

```
/* USB HID Requests */
#define USBRQ_HID_GET_REPORT    0x01
#define USBRQ_HID_GET_IDLE      0x02
#define USBRQ_HID_GET_PROTOCOL  0x03
#define USBRQ_HID_SET_REPORT    0x09
#define USBRQ_HID_SET_IDLE      0x0a
#define USBRQ_HID_SET_PROTOCOL  0x0b

/* ------------------------------------------------------------------------- */

#endif /* __usbdrv_h_included__ */
```