СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ВОЛОДИМИРА ДАЛЯ

(м. Сєвєродонецьк)

Факультет Інформаційних технологій та електроніки

(повне найменування факультету)

Кафедра Електронних апаратів

(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

освітньо-кваліфікаційного рівня магістр

(бакалавр, спеціаліст, магістр)

спеціальності 171 – Електроніка

(шифр і назва спеціальності)

на тему Енергоефективні мікропроцесорні системи збору та обробки даних

Виконав: студент групи Ел-14м

Плескач Т.А.

(прізвище, та ініціали) (підпис)

|  |  |
| --- | --- |
| Керівник Захожай О.І. |  |
| (прізвище та ініціали) | (підпис) |
| Завідувач кафедри Паеранд Ю.Е. |  |
| (прізвище та ініціали) | (підпис) |
| Рецензент |  |
| (прізвище та ініціали) | (підпис) |

Сєвєродонецьк - 2020

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ВОЛОДИМИРА ДАЛЯ

(м. Сєвєродонецьк)

Факультет Інформаційних технологій та електроніки

Кафедра Електронних апаратів

Освітньо-кваліфікаційний рівень магістр

(бакалавр, спеціаліст, магістр)

Спеціальність 171 – Електроніка

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Ю.Е. Паеранд

“ ” 2020 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Плескач Тимур Олександрович

(прізвище, ім’я, по батькові)

1. Тема роботи Енергоефективні мікропроцесорні системи\_збору та обробки даних\_ керівник проекту (роботи) Захожай Олег Ігоревич к.т.н., доцент

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “06” вересня 2019 року № 121/15.14

1. Строк подання студентом роботи 15.01.2020
2. Вихідні дані до роботи Дослідити питання енергоефективності мікропроцесорних систем збору та обробки даних
3. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
4. Аналітичний огляд
5. Дослідження методів\_зниження\_енергоспоживання\_мікропроцесорних\_систем
6. Розробка, обґрунтування\_й\_опис\_схем
7. Розробка алгоритму роботи мікропроцесорної\_системи\_збору\_та\_обробки\_даних 5) Аналіз\_методів\_зниження\_енергоспоживання\_мікропроцесорних\_систем\_збору

\_та\_обробки\_даних\_за\_отриманими\_результатами

1. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
2. Дата видачі завдання 01.09.2019

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Примітка |
| 1 | Актуальність теми і постановка завдання | 02.09.19 - 09.09.19 |  |
| 2 | Аналітичний огляд | 10.09.19 - 29.09.19 |  |
| 3 | Дослідження методів зниження енергоспоживання мікропроцесорних систем | 30.09.19 - 31.10.19 |  |
| 4 | Розробка, обґрунтування й опис схем | 01.11.19 - 30.11.19 |  |
| 5 | Розробка алгоритму роботи мікропроцесорної системи збору та обробки даних | 01.12.19 - 08.12.19 |  |
| 6 | Аналіз методів зниження енергоспоживання мікропроцесорних систем збору та обробки даних за отриманими результатами | 09.12.19 - 17.12.19 |  |
| 7 | Узагальнення результатів. Висновки | 16.12.19 - 19.12.19 |  |
| 8 | Оформлення пояснювальної записки | 20.12.19 - 27.12.19 |  |

Студент Плескач Т.А.

(підпис) (прізвище та ініціали)

Керівник проекту (роботи) Захожай О.І.

(підпис) (прізвище та ініціали)

|  |  |
| --- | --- |
| ЗМІСТ |  |
| ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ…………………………………... | 6 |
| ВСТУП …………………………………………………………………….. | 7 |
| 1 ПЕРШИЙ РОЗДІЛ |  |
| ЕНЕРГОЕФЕКТИВНІСТЬ МІКРОПРОЦЕСОРНИХ СИСТЕМ ЗБОРУ І  ОБРОБКИ ДАНИХ ………………………………………………………. | 8 |
| 1.1 Фактори, що пливають на споживану потужність  мікропроцесорних систем…………………………………………………. | 8 |
| 1.2 Методи зниження енергоспоживання в строго самосинхронних  мікропроцесорних схемах ………………………………………………… | 11 |
| 1.2.1 Особливості строго самосинхронної схемотехніки ………… | 11 |
| 1.2.2 Методи зниження енергоспоживання в процесорних  системах…………………………………………………………………….. | 16 |
| 1.3 Мінімізація енергоспоживання в системах на основі  Мікроконтролера ………………………………………………………….. | 19 |
| 1.4 Технологія зниження енергоспоживання picoPower……………... | 21 |
| 1.5 Висновки за розділом ………………………………………………. | 34 |
| 2 ДРУГИЙ РОЗДІЛ |  |
| МЕТОДИ ЗНИЖЕННЯ ЕНЕРГОСПОЖИВАННЯ  МІКРОПРОЦЕСОРНИХ СИСТЕМ ЗБОРУ ТА ОБРОБКИ ДАНИХ ….. | 35 |
| 2.1 Підготовка до розробки мікропроцесорної системи …………….. | 35 |
| 2.1.1 Вибір апаратної частини ……..………………………………. | 35 |
| 2.1.2 Джерела живлення ……………………………………………. | 36 |
| 2.2 Живлення елементів мікропроцесорної системи ………………… | 37 |
| 2.3 Режими роботи мікропроцесора ………………………………….. | 40 |
| 2.4 Віддалена обробка та зберігання даних ………………………….. | 44 |
| 2.5 Висновки за розділом ………………………………………………. | 58 |

|  |  |
| --- | --- |
| 3 ТРЕТІЙ РОЗДІЛ |  |
| ІТОГОВИЙ АНАЛІЗ МЕТОДІВ ЗНИЖЕННЯ ЕНЕРГОСПОЖИВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ ЗБОРУ І ОБРОБКИ  ДАНИХ……………………………………………………………………… | 49 |
| 3.1 Стандартне рішення……..………………………………………….. | 49 |
| 3.2 Рішення з використанням режимів економії енергії  мікроконтролеру.……………………………………………………………. | 53 |
| 3.3 Рішення з використанням додаткових методів зниження  енергоспоживання.…………………………………………………………... | 56 |
| 3.4 Висновки за розділом …..…………………………………………… | 59 |
| ВИСНОВКИ…………………………………………………………………. | 61 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ………………………………….. | 63 |
| ДОДАТОК А КОД ПРОГРАМИ…………………………………………… | 65 |

# ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

КМОН (CMOS) - комплементарна структура метал-оксид- напівпровідник.

АСУ ТП - Автоматизована система управління технологічним процесом**.**

ССМ **-** строго самосинхронні мікропроцесори МК – мікроконтролер

ККД **–** Коефіцієнт корисної дії

# ВСТУП

Під енергоефективністю мікропроцесорних систем збору і обробки даних, мається на увазі технічний і технологічний комплекс методів і рішень, спрямованих на раціоналізацію споживання енергії при виконанні поставленого перед мікропроцесорною системою завдання.

Мікропроцесорні системи, самі по собі, споживають порівняно мало енергії, а проблема їх енергоефективності, в першу чергу виникає при відсутності можливості підключення до загальної енергомережі, або короткочасною відсутністю в ній напруги.

Наприклад, портативні пристрої, які повинні застосовуватися в польових умовах, які живляться від акумулятора, так як підключення до мережі не представляється можливим.

Так само і системи, розраховані на безперервну роботу, на випадок раптового відключення загального живлення, що працюють на резервних акумуляторах, наприклад: системи безпеки, АСУ ТП на підприємствах, системи розумного будинку.

У цих випадках важливим є те, як довго мікропроцесорна система здатна працювати в автономному режимі, що напряму залежить від її енергоефективності та ємності акумулятора.

Метою даної дипломної роботи є дослідження проблеми енергоефективності, а також методів підвищення енергоефективності мікропроцесорних систем збору та обробки даних їх огляд та аналіз.

# ПЕРШИЙ РОЗДІЛ ЕНЕРГОЕФЕКТИВНІСТЬ МІКРОПРОЦЕСОРНИХ СИСТЕМ ЗБОРУ І

**ОБРОБКИ ДАНИХ**

* 1. **Фактори, що впливають на споживану потужність мікропроцесорних систем**

Для сучасних цифрових мікросхем споживання розбивається на дві складові: статичну та динамічну. У технічній літературі та довідниках по дискретним мікросхем КМОН зазвичай зазначалося, що в статичному режимі вони не споживають струму. При цьому робилися уточнення, що при відсутності перемикань споживання мікросхем КМОН обумовлювалося в основному струмами витоку затворів - через оксид кремнію, що володіє дуже високим питомим опором. [1]

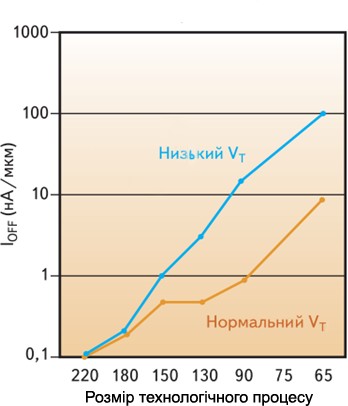


Рисунок 1.1 - Залежність струму витоку від характерного розміру технологічного процесу

Однак питомий опір - це тільки характеристика матеріалу, а власне опір пропорційний довжині провідника і обернено пропорційний площі його поперечного перерізу. Зі зменшенням норм технологічних процесів відбулося зменшення товщини оксиду кремнію (яка виступає як довжина провідника, через який відбувається витік), і в результаті абсолютне значення опору впало до величин, при яких говорити про нехтування малості струму витоку вже не можна. Графік залежності струму витоку від норми технологічного процесу показаний на рисунку 1.1. Необхідно звернути увагу, що по вертикальній шкалі масштаб логарифмічний.

Важливо, що при досягненні норм 90-65 нм статичне споживання досягло такого рівня, що для його зниження довелося вживати відповідних заходів. Зокрема, фірма Xilinx в 90-нм сімействі FPGA Virtex-4 використовувала технологію triple-oxide. Її суть полягає в тому, що тонка оксидна плівка використовується тільки там, де потрібна максимальна швидкодія транзисторів (чим менше товщина оксиду, тим більше вплив затвора на канал, тобто швидше відбувається перемикання транзистора). Для менш відповідальних елементів товщина плівки підвищується - до середньої і максимальної (отже, знижуються струм витоку). Таким чином, в залежності від необхідної для цього транзистора швидкості перемикання він може бути виконаний за одним з трьох варіантів товщини ізолятора між затвором і каналом. Транзистор із середньою товщиною оксиду кремнію в FPGA Virtex- 4 показаний на рисунку 1.2

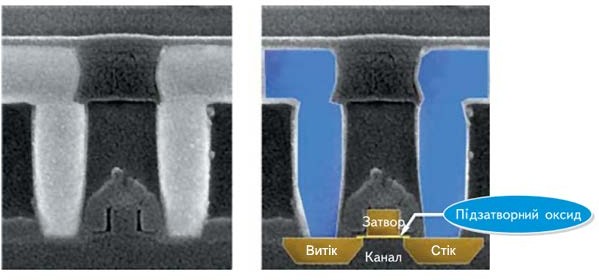


Рисунок. 1.2 - Транзистор з середньою товщиною оксиду кремнію в

FPGA Virtex + 4

На статичне споживання також впливають розкид параметрів транзисторів в процесі їх виготовлення, відхилення напруги живлення і температура (рис. 1.3).

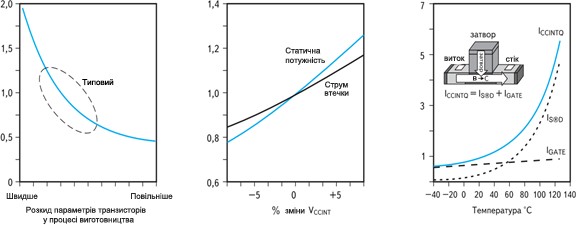


Рисунок. 1.3 - Залежність відносного рівня статичного споживання від

різних факторів

Динамічне споживання потужності визначається за формулою:

H:\Нужное\0000 2019.10.27\Screenshot_4.png

де n - число вузлів, що перемикаються; C - ємність; V - різницю рівнів напруги; f - частота.

Зі зміною технологічних процесів відбувається кількісна зміна параметрів, що визначають динаміку споживання. Крім того, нові архітектурні рішення, які закладаються в більш сучасні сімейства, також змінюють рівень динамічного споживання.

# Методи зниження енергоспоживання в строго самосинхронних мікропроцесорних схемах

Існує велика кількість засобів зниження енергоспоживання синхронних процесорних систем, пов'язаних з управлінням напругою живлення і частотою роботи, але їх можливості обмежені. Дана проблема особливо актуальна для бортових і портативних систем, для яких енергоспоживання стає одним з факторів, що стримують подальший розвиток таких систем. Одним з кардинальних шляхів зниження енергоспоживання є перехід до самосинхронної методики проектування. Через особливості даного підходу строго самосинхронні мікропроцесори (ССМ) мають менше енергоспоживання в порівнянні з широко вживаними синхронними обчислювальними пристроями при сумірній продуктивності. Це обумовлено, в першу чергу, відсутністю необхідності використання глобальних ланцюгів тактової синхронізації, котрі являються додатковими споживачами [2]. Крім того, строго самосинхронні схеми при відсутності зміни даних на вході знаходяться в статичному стані, що дозволяє автоматично знижувати споживану потужність у порівнянню з синхронними аналогами.

# Особливості строго самосинхронної схемотехніки

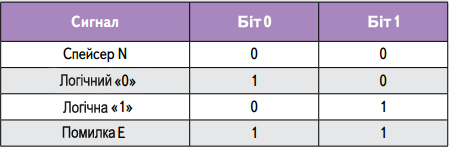
Особливістю самосинхронних схем є відсутність глобальних тактових сигналів, використовуваних в синхронній схемотехніці. При використанні

даного підходу кожен комбінаційний блок після завершення перехідних процесів повинен виробляти сигнал готовності прийому наступних даних, за яким здійснюється синхронізація попереднього блоку, тим самим забезпечуючи логічне упорядкування подій у схемі. Дані між логічними блоками передаються із застосуванням деякого протоколу передачі, який забезпечує синхронізацію інформації. У завдання протоколу входить супровід переданих даних парою «запит / підтвердження», яка забезпечує синхронізацію на локальній ділянці схеми. [3]

Один з варіантів побудови строго самосинхронних схем заснований на використанні парафазного представлення сигналів.

Для передачі одного біта інформації D використовується два біта - D0 і D1 (табл. 1.4). Таке представлення дозволяє, крім звичайних логічних станів -

«0» і «1», визначати спейсер (стан переходу, NULL).



Рисурок 1.4 - Надмірне представлення логічного сигналу

На рис. 1.5 показані часові діаграми представлення сигналу в парафазному коді. При передачі біти даних поділяються спейсером, що дозволяє відрізняти перехідні процеси від стаціонарних станів сигналів. Крім того, не потрібна окрема передача сигналу «запит», так як він безпосередньо вбудований в дані.

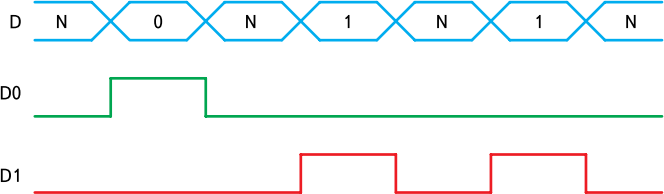


Рисунок 1.5 - Часові діаграми парафазного представлення сигналу

Стан помилки неприпустимий для справної схеми. В реальній схемі він не повинен зустрічатися ніколи, тому даний стан може, наприклад, використовуватися для контролю справності роботи схеми.

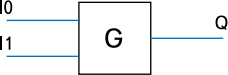
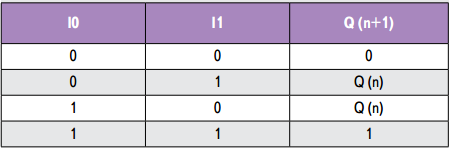
Парафазне кодування дозволяє визначати момент закінчення перехідного процесу, безпосередньо аналізуючи дані на виході блоку, без використання додаткових блоків індикації. Зараз це рішення найбільш часто використовується, воно дозволяє будувати строго самосинхронні схеми, використовуючи стандартні логічні елементи.

Рисунок 1.6 - Умовне графічне зображення G-тригера Основою строго самосинхронних схем є G-тригер (гістерезисний

тригер, С-елемент Мюллера). Вихід G-тригера є копією вхідних сигналів, якщо вони мають однакове значення, і зберігає свій попередній стан, якщо вони відрізняються один від одного. Тригер може мати довільну кількість входів. На рис. 1.6 показано умовне графічне зображення двухвхідного елемента, а в таблиці 1.7 наведено його таблиця істинності.



Таблця 1.7 *-* Таблиця істинності G-тригера

На рисунку 1.8 показаний приклад найпростішої строго самосинхронної схеми. Її основою є асинхронні засувки і індикатори закінчення перехідного процесу, які разом утворюють асинхронні регістри.

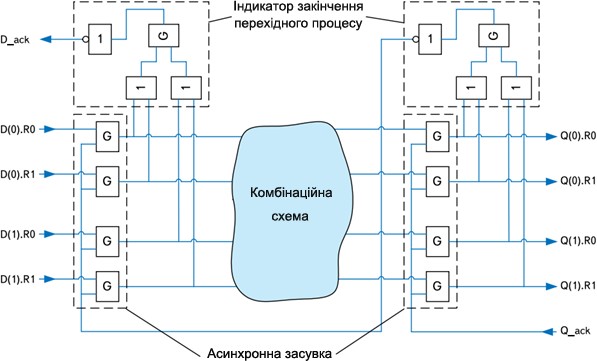


Рисунок 1.8 *-* Приклад строго самосинхронної схеми

Схема працює в такий спосіб. У початковому стані всі лінії даних знаходяться в стані спейсера N, а виходи блоків індикації - в стані логічної

«1», що говорить про те, що асинхронні засувки готові приймати дані. При надходженні на вхід схеми D даних вони проходять через G-тригери першої асинхронної засувки і далі надходять на вхід комбінаційної схеми та першого

індикатора. Наявність даних на вході індикатора викличе перемикання його виходу в стан логічного «0», що показує готовність першої асинхронної засувки прийняти наступний за даними спейсер. При його вступі на вхід схеми він не зможе пройти через першу асинхронну засувку, так як вихід другого індикатора знаходиться в стані логічної «1», і перша асинхронна засувка зберігає на виході свій попередній стан.

Через деякий час, що дорівнює затримці проходження сигналів через комбінаційну схему, дані з'являються на її виході. Для того щоб дані пройшли через другу асинхронну засувку, необхідно, щоб сигнал Q\_ack був

«1», показуючи, що наступна частина схеми готова прийняти дані. Якщо ця умова виконується, дані пройдуть другу асинхронну засувку і викличуть перемикання другого індикатора. Це в свою чергу дозволить першій засувці пропустити стан спейсера через себе, що призведе до перемикання першого індикатора. Коли стан спейсера пройде через комбінаційну схему, воно надійде на вхід другої засувки, яка, поки сигнал Q\_ack знаходиться в «1», буде зберігати на своєму виході прийшли раніше дані.

Розробникам синхронних схем не потрібно відстежувати порядок проходження даних через блоки: це робиться автоматично при використанні глобального тактового сигналу і перевірці дотримання обмежень проекту. Якщо в синхронних схемах розробник мав справу з окремими сигналами, зміни яких відбувалися синхронно у всій схемі, то при самосинхронному підході немає синхронізації між окремими подіями у схемі. Необхідно строго контролювати порядок проходження даних через модулі. Це вимагає додаткових апаратних витрат: введення додаткових маркерів даних, сигналів синхронізації роботи окремих блоків і принципово відрізняється підходу до проектування схеми. Завдання синхронізації обробки даних можуть вирішуватися як розробником при створенні опису схеми, так і за допомогою спеціальних програмних засобів на етапі синтезу схеми - з її високорівневого опису на спеціальній мові опису апаратури, призначеному для опису самосинхронних схем.

# Методи зниження енергоспоживання в мікропроцесорних системах

H:\Нужное\0000 2019.10.27\Screenshot_10.pngДля КМОН-технологій значну частку розсіяної потужності має динамічна складова, яка визначається паразитним струмом заряду ємності навантаження логічного елемента і наскрізним струмом від джерела живлення до загального проведення в момент зміни логічного рівня. Для синхронних мікропроцесорів, якщо частота перемикання сигналу дорівнює f, то потужність, що розсіяна паразитним струмом заряду вхідний ємності визначається виразом:

де С𝐿𝐿 - ємність навантаження,

U - напруга живлення.

Для ССМ ця формула виглядає інакше:

H:\Нужное\0000 2019.10.27\Screenshot_11.png

де *s* - миттєва швидкість процесора.

З цього виразу випливає, що зниження споживаної потужності ССМ можливо за рахунок зменшення ємності навантаження, напруги живлення і миттєвої швидкості *s*. [4]

Друга складова розсіяної потужності КМОН-схем - наскрізний струм. Величина наскрізного струму прямо пропорційна тривалості фронтів вхідного сигналу і обернено пропорційна ємності навантаження (С𝐿𝐿). Потужність, що розсіюється КМОН-схемою внаслідок протікання наскрізного струму, може становити від 10 до 60% загальної динамічної

потужності. Її величина залежить від параметрів технології виготовлення

мікросхеми, таких як вольт-амперні характеристики транзисторів, тривалість переднього і заднього фронтів вхідного сигналу, вхідний ємності і напруги живлення.

Крім динамічного, необхідно враховувати статичне енергоспоживання мікропроцесорної системи. Його причина – струми витоку КМОН- транзисторів і зворотні струми p-n-переходів.

Номінальні значення струмів витоку «стік / витік» лежать в діапазоні 1- 10 нА, причому величина струму експоненціально залежить від порогової величини напруги. Тому зі збільшенням розкиду порогових напруг середні значення також зростають експоненціально. Єдиним способом зменшення статичного енергоспоживання є зниження напруги живлення аж до відключення невикористовуваних блоків.

Таким чином, фактори, що впливають на енергоспоживання ССМ, можна розділити на дві групи - технологічні, залежні тільки від технології виготовлення мікросхеми, і зовнішні, якими можна управляти під час роботи. Аналіз динамічного енергоспоживання мікросхем, побудованих по КМОН- технології, показує, що питоме енергоспоживання (Вт / МГц) пропорційно квадрату напруги живлення, тобто при зменшенні напруги живлення знижується не тільки продуктивність ССМ, а й питоме енергоспоживання. Таким чином, з точки зору енергозбереження найбільш вигідною є робота ССМ на гранично низькій частоті. У реальному пристрої це неможливо, так як виконання обчислювальних задач необхідно проводити за обмежений час. На підставі вищесказаного можна виділити наступні способи управління енергоспоживанням ССМ:

* відключення не використовуваних функціональних блоків для зменшення статичного струму витоку;
* зниження напруги живлення під час роботи до мінімально можливого рівня, достатнього для вирішення необхідних обчислювальних задач з метою зменшення динамічної складової енергоспоживання.

Додатково варто відзначити, що в строго самосинхронних схемах перемикання логічних елементів, що викликають динамічну складову енергоспоживання, відбувається тільки при обробці даних, дід час простою схема знаходиться в статичному стані. У синхронних мікропроцесорах для зменшення динамічної складової енергоспоживання під час простою необхідно використовувати спеціальні технічні рішення, що здійснюють зниження або повне відключення тактової частоти у деяких блоків.

Для переходу в режим зниженого енергоспоживання і повернення з нього синхронним процесорам потрібен додатковий час, протягом якого процесор простоює. Цей час може бути значним, якщо необхідно перебудовувати синтезатор частоти (PLL). На відміну від цього, в самосинхронних процесорах перехід в такий режим відбувається автоматично. Споживання енергії відбувається тільки в процесі обробки даних. В інший час ССМ знаходиться в режимі очікування, поки не відбудеться запит або переривання. При цьому не потрібно додаткового часу на включення невикористовуваних блоків. Ця особливість строго самосинхронної схемотехніки дозволяє постійно використовувати режим зниженого енергоспоживання без застосування додаткових апаратних засобів, за винятком випадку керування напругою живлення.

Незважаючи на наявні у строго самосинхронних мікропроцесорів переваги, пов'язані з більш низьким енергоспоживанням в порівнянні з синхронними аналогами, проблема зниження енергоспоживання для них також актуальна. Розглянуті алгоритми дозволяють зменшити енергоспоживання ССМ за рахунок оптимального перерозподілу обчислювальної потужності в залежності від розв'язуваної задачі. Найбільш оптимальним є використання комбінації розглянутих алгоритмів. Вони можуть бути реалізовані як програмно, наприклад, засобами операційної системи, так і апаратно, за допомогою спеціальних блоків оцінки і прогнозування.

Описані алгоритми керування напругою живлення дозволяють підвищити пікову продуктивність обчислювальної системи, не допускаючи перегріву мікросхеми.

Результати моделювання алгоритмів управління показують можливість зниження споживаної потужності більш ніж на 80%. [3]

# Мінімізація енергоспоживання в системах на основі Мікроконтролера

Малий рівень енергоспоживання є часто визначальним фактором при виборі способу реалізації цифрової керуючої системи. Сучасні МК надають користувачеві великі можливості в плані економії енергоспоживання і мають, як правило, такі основні режими роботи [5]:

* + активний режим (Run mode) - основний режим роботи МК. В цьому режимі МК виконує робочу програму і все його ресурси доступні. Споживана потужність має максимальне значення Prun. Більшість сучасних МК виконано по КМОН - технології, тому потужність споживання в активному режимі сильно залежить від тактової частоти;
  + режим очікування (Wait mode, Idle mode або Halt mode). В цьому режимі припиняє роботу центральний процесор, але продовжують функціонувати периферійні модулі, які контролюють стан об'єкта управління. При необхідності сигнали від периферійних модулів переводять МК в активний режим, і робоча програма формує необхідні керуючі впливи. Переклад МК з режиму очікування в робочий режим здійснюється по перериваннях від зовнішніх джерел або периферійних модулів, або при скиданні МК. У режимі очікування потужність споживання МК Pwait знижується в порівнянні з активним режимом в 5 ... 10 разів;
  + режим зупинки (Stop mode, Sleep mode або Power Down mode). В цьому режимі припиняє роботу, як центральний процесор, так і більшість периферійних модулів. Перехід МК зі стану зупинки в робочий режим

можливий, як правило, тільки по перериваннях від зовнішніх джерел або після подачі сигналу скидання. В режимі зупинки потужність споживання МК Pstop знижується в порівнянні з активним режимом приблизно на три порядки і складає одиниці мікроват.

Два останніх режими називають режимами зниженого енергоспоживання. Мінімізація енергоспоживання системи на МК досягається за рахунок оптимізації потужності споживання МК в активному режимі, а також використання режимів зниженого енергоспоживання. При цьому необхідно мати на увазі, що режими очікування і зупинки істотно відрізняються часом переходу з режиму зниженого енергоспоживання в активний режим. Вихід з режиму очікування зазвичай відбувається протягом 3 ... 5 періодів синхронізації МК, в той час як затримка виходу з режиму зупинки становить кілька тисяч періодів синхронізації. Крім зниження динаміки роботи системи значний час переходу в активний режим є причиною додаткової витрати енергії.

Потужність споживання МК в активному режимі є однією з найважливіших характеристик контролера. Вона в значній мірі залежить від напруги живлення МК і частоти тактування. Залежно від діапазону живлячих напруг все МК можна розділити на три основні групи [6]:

* МК з напругою живлення 5,0В ± 10%. Ці МК призначені, як правило, для роботи в складі пристроїв з живленням від промислової або побутової мережі, мають розвинені функціональні можливості і високий рівень енергоспоживання.
* МК з розширеним діапазоном напруг живлення: від 2,0 ... 3,0В до 5,0- 7,0В. МК даної групи можуть працювати в складі пристроїв як з мережевим, так і з автономним живленням.
* МК із зниженою напругою живлення: від 1,8 до 3В. Ці МК призначені для роботи в пристроях з автономним живленням і забезпечують економну витрату енергії елементів живлення. Залежність струму споживання від напруги живлення МК майже прямо пропорційна. Тому зниження напруги

живлення суттєво знижує потужність споживання МК. Необхідно, однак, мати на увазі, що для багатьох типів МК зі зниженням напруги живлення зменшується максимально допустима частота тактування, тобто виграш в споживаної потужності супроводжується зниженням продуктивності системи.

В сучасних МК виконаних за технологією КМОН потужність споживання в активному режимі Prun практично прямо пропорційна тактовій частоті. Тому, вибираючи частоту тактового генератора, не слід прагнути до гранично високій швидкодії МК в задачах, які цього не вимагають. Часто визначальним фактором виявляється роздільна здатність вимірювачів або формувачів часових інтервалів на основі таймера або швидкість передачі даних по послідовному каналу. Вони здатні працювати при як завгодно низьких тактових частотах, аж до нульових.

# Технологія зниження енергоспоживання picoPower

У 2004 році компанією Atmel були вперше представлені пристрої, що працюють зі 100%-вим функціоналом при напрузі живлення 1,8 В. У наступному році технологічні напрацювання і дизайнерські прийоми Atmel оформилися в власну технологію, названу picoPower. Вона забезпечує мінімально можливе енергоспоживання кристала при збереженні інших параметрів вироби на високому рівні. Саме тоді був випущений перший мікроконтролер, що споживає менше 100нА при збереженні вмісту внутрішньої пам'яті SRAM. У 2009 році компанія Atmel представила кристал AТ32UC3L з струмом споживання 9нА в режимі Standby і енергоспоживанням в активному режимі 0,27 мВт / МГц.

Технологія Atmel picoPower базується на двох основних принципах:

* Завжди завершувати процес обробки інформації максимально швидко, витративши при цьому так мало енергії, наскільки можливо.
* Не застосовувати препарат або майже не використовувати енергію всякий раз, коли процес зупинений.

На додаток до технологічно вивіреним балансом підсилюючих характеристик транзисторів і мінімізації їх струмів витоку технологія picoPower включає наступні інженерні прийоми [7,8,10]:

# Зупинка або зниження тактової частоти для не використовуваних блоків мікроконтролера

Мікроконтролер може функціонувати при різних тактових частотах, при цьому його енергоспоживання зменшується зі зниженням тактової частоти. При проектуванні енергоефективного пристрою на базі мікроконтролера немає необхідності тактувати все його блоки з однаковою частотою. Мікроконтролери сімейств AVR і SMART ARM використовують тактові сигнали різної частоти для ядра, внутрішніх шин і для кожного незалежного периферійного блоку з метою зниження загального споживаного струму. Наприклад, якщо задані додатком функціональні характеристики периферійного блоку забезпечуються при його роботі на зниженій тактовій частоті, то енергоспоживання всього кристала буде знижено. Більш того, в сучасних мікроконтролерах AVR і SMART ARM не використовувані периферійні блоки можуть бути індивідуально повністю зупинені і знову активовані «на льоту», під час роботи пристрою, що також призводить до подальшого зниження середньої потужності без погіршення продуктивності.

# Динамічне живлення функціональних блоків мікроконтролера

Мікроконтролери Atmel містять внутрішні регулятори напруги з різними вихідними рівнями, які можуть бути налаштовані розробником або активовані автоматично в залежності від заданого рівня продуктивності і

обраного режиму роботи кристала. Існує безліч режимів, за допомогою яких можна керувати живленням периферійних блоків, а також включати і вимикати їх. Коли периферійний модуль відключається автоматично, його логічне стан є і прозоро для користувача програми, оскільки вміст регістрів зберігається в процесі включення і виключення.

Технологія picoPower також передбачає різні режими електроживлення статичної пам'яті SRAM на кристалі, в тому числі автоматичне перемикання в мало-яке споживає стан в режимі сну, вибіркове відключення окремих областей або повне відключення статичної пам'яті в режимах глибокого сну. У деяких кристалах доступна функція режиму сну з буферним живленням від гальванічного елемента. У цьому режимі всі периферійні блоки, статична пам'ять, тактові ланцюга і генератори знеструмлені. До батареї живлення залишаються підключеними джерело тактової частоти 32 кГц, годинник реального часу RTC і ланцюги пробудження контролера від зовнішніх сигналів.

# Контролер прямого доступу до пам'яті (DMA) і система подій

Ці апаратні блоки на кристалі мікроконтролера дозволяють значно розвантажити процесорне ядро і в активних, і в сплячих режимах роботи. Традиційно переміщення даних з периферійних блоків у область пам'яті і назад виконується під управлінням процесора. Чим більше даних необхідно перемістити, тим більше робочих циклів для цього буде потрібно. Але існує й інша можливість. Технологія picoPower передбачає активне використання контролера DMA, який виконує завдання по переміщенню даних значно ефективніше, дозволяючи при цьому центрального процесора залишатися в сплячому режимі і зберігати енергію або виконувати обчислювальні операції паралельно передачі даних. Наприклад, використання контролера DMA спільно з блоком апаратного обчислення контрольної суми (CRC) дозволяє здійснювати перевірку цілісності даних, що приймаються або готувати

передані дані без залучення в роботу центрального процесора. Це призводить до підвищення загальної продуктивності, надаючи можливість ядру частіше і на більш тривалий час йти в сплячий режим.

Застосовувана в родинах AVR і SMART ARM система подій дозволяє периферії на кристалі самостійно управляти пересиланням даних безпосередньо в інші периферійні блоки. При цьому використовуються спеціальні виділені зв'язку між блоками, повністю незалежні від центрального процесора. Такі зв'язки розвантажують ядро від рутинної передачі даних, і на даний час воно може бути переведено в сплячий режим. Загальна продуктивність системи при цьому зростає, а енергоспоживання знижується. Більш того, функціонування системи подій повністю передбачувано і відмінно відповідає вимогам до пристроїв, що працюють в реальному часі.

# Інтелектуальні периферійні пристрої

Складовою частиною технології picoPower є функція SleepWalking («сон на ходу»), яку корпорація Atmel додала деяким периферійним блокам. З її допомогою такої блок може самостійно вирішувати, чи потрібен центральний процесор для обробки даних, що надійшли чи ні. Функція названа SleepWalking, оскільки вона дозволяє ядру «мирно спати» до тих пір, поки не відбудеться важлива подія, що вимагає його активної дії. Вона дозволяє кардинально знизити загальну кількість надлишкових циклів пробудження процесора.

При традиційному способі моніторингу зовнішніх подій внутрішній таймер періодично переводить контролер в активний режим для перевірки зовнішніх умов, при цьому ядро і пам'ять споживають значну енергію. У деяких випадках, коли час реакції має бути дуже коротким, центральний процесор може навіть не встигати переходити в режим сну. У мікроконтролерах AVR і SMART ARM це вирішується інакше - наприклад,

поки ядро знаходиться в режимі сну, інтерфейс TWI чекає і самостійно визначає збіг адреси, або АЦП фіксує перевищення вхідного сигналу заданого порогу. Центральний процесор і пам'ять не будуть активовані, поки заданий умова не буде виконана. Функція SleepWalking в послідовних комунікаційних портах спільно з DMA дозволяють організувати передачу даних в ультрамалоспоживаючих режимах. Периферійний контролер торкання здатний будити контролер тільки під час наближення або дотику до сенсорних кнопок, забезпечуючи економічну роботу приладів з живленням від батареї. Більш того, за сигналами від периферійних блоків можуть змінюватися режими електроживлення інших апаратних вузлів мікроконтролера, дозволяючи подальшу оптимізацію енергоспоживання в таких «залежних» областях застосування, як носиться медична техніка, портативні сигналізатори газів і модулі бездротового зв'язку.

# Швидке пробудження кристала від декількох джерел сигналу з режимів найнижчого енергоспоживання

Для економії енергії переклад мікроконтролера в сплячий режим відключає живлення його деяких складових частин. Генератори і ланцюги тактування можуть споживати значну частку потужності в активному режимі, а в процесі пробудження з режимів сну вони повинні стабілізуватися до початку їх використання. Очікування стабілізації протягом тривалого часу призводить до втрат енергії. Мікроконтролери Atmel AVR і SMART ARM спроектовані для максимально швидкого пробудження. Не більше 8 тактів потрібно для стабілізації внутрішнього RC-генератора. Традиційний програмований генератор з фазовим автопідстроюванням частоти (PLL) замінений цифровим (DPLL), який значно швидше і точніше. Це також дозволило відмовитися від деяких зовнішніх компонентів і ще більше знизити споживану потужність. Коли синхронізація від тактового генератора відключена в режимі сну, мікроконтролер все ще може бути переведений в

активний режим за допомогою асинхронного події - зміна стану зовнішнього виведення, отримання даних або збіг адреси на шині TWI.

# Повна функціональність при роботі з низькою напругою живлення

Мікроконтролери Atmel, виконані по технології picoPower, включаючи їх аналогові блоки, повністю працездатні при зниженні напруги живлення аж до 1,62 В. Відомо, що простий спосіб зменшити енергоспоживання будь- якого приладу - знизити подається напруга живлення. Однак даний метод часто марний і не застосуємо через погіршення точності аналогових ланцюгів, нестабільності роботи окремих периферійних блоків і осциляторів або неможливості здійснити процес запису в енергонезалежну пам'ять. Визначальним в технології picoPower є саме те, що всі аналогові і цифрові вузли ретельно спроектовані для точної роботи при мінімальному напрузі 1,62 В, включаючи операції записи в Flash-пам'ять і EEPROM.

На практиці це означає, що функціональні блоки мікроконтролера не перестають правильно працювати при зниженні напруги живлення, і це гарантує надійне функціонування всього додатка. При батарейному живленні здатність до повноцінною роботі при екстремально низькій напрузі живлення дозволяє продовжити час життя батареї за рахунок найбільш повного використання міститься в ній енергії.

В даний час корпорація Atmel активно впроваджує технологію picoPower в усі свої мікроконтролери. У сімействі на основі ядра ARM Cortex-M4 випущена серія SAM4L, яка сьогодні налічує дев'ять виробів з об'ємом Flash-пам'яті 128, 256 і 512 кбайт і випускається в 48-, 64-, 100- вивідних корпусах. Мікроконтролери містять розвинену периферію, включаючи апаратний контролер торкання QTouch і контролер РКІ з можливістю управління 4 \* 40 сегментами. Кристали SAM4L споживають в активному режимі 90 мкA / MГц, забезпечуючи при цьому цифрову обробку

сигналів і роботу високошвидкісних комунікаційних пристроїв введення / виводу.

Сімейство мікроконтролерів Atmel SMART SAM G5 оптимізовано для малогабаритних високопродуктивних систем з наднизьким енергоспоживанням. Малий форм-фактор цих пристроїв забезпечується ядром ARM Cortex-M4 в поєднанні із блоком обробки операцій з плаваючою комою (FPU). Дані мікроконтролери випускаються в компактних корпусах WLCSP розміром 3 \* 3 мм.

Десять варіантів мікроконтролерів з робочою частотою до 48 МГц складають серію SAML21, що базується на ядрі ARM Cortex-M0 + і орієнтовану на застосування в пристроях з батарейним живленням. Мікроконтролер SMART SAM L21 - одна з останніх розробок Atmel, яка є прикладом вдалого використання всіх переваг технології picoPower. Він спеціально створювався для застосування в відособлених системах, віддалених від мережевого електроживлення, наприклад вузлах IoT (Internet of Things - «Інтернет речей»). Володіючи високою швидкістю пробудження, апаратної системою подій, інноваційної picoPower-периферією, SAM L21 відмінно підходить для переносної і портативної апаратури різного призначення.

Мікроконтролери серії SAM L21 включають розвинену периферію, до 256 кбайт вбудованої Flash-пам'яті і до 40 кбайт пам'яті SRAM, споживають струм менше 35 мкА / МГц в активному стані при роботі з Flash (може бути ще менше, якщо працювати з SRAM). Забезпечується легке міграція додатків, розроблених для базового кристала SAM D21. У 2015 році консорціум провідних виробників мікроконтролерів (Embedded Microprocessor Benchmark Consortium) визнав SAM L21 кращим за співвідношенням продуктивності і споживаної потужності.

Елементи технології picoPower активно застосовуються компанією Atmel і в мікроконтролерах AVR. Так, у всіх кристалах серії XMEGA реалізована система подій, їх повноцінне функціонування забезпечується при

напрузі живлення 1,62В, а таймер системи реального часу працює при струмі 100 нА.

Для роботи стандартного мікроконтролера, зазвичай, потрібна напруга живлення не менше 1,8 В, в той час як напруга одного повністю зарядженого гальванічного елемента знаходиться в діапазоні 1,2-1,5 В. Це означає, що такому микроконтролеру необхідна батарея як мінімум з двох елементів. Після розряду в процесі роботи до напруги 1В гальванічний елемент містить ще значну величину енергії, яку бажано використовувати. Компанія Atmel запропонувала технічне рішення для такого завдання, включивши DC / DC- перетворювач до складу ATtiny43U, який підвищує вхідна напруга до потрібного рівня. Перетворювач забезпечує мікроконтролер фіксованою напругою 3 В від одного гальванічного елемента при його глибокому розряді до 0,7 В. Цей прийом дозволяє одноразовим джерел розряджатися до мінімуму і значно продовжує термін їх служби, а для перезаряджаються акумуляторів програмований поріг відсічення дозволяє запобігти їх руйнуванню. На рис. 1.9 представлена схема включення вбудованого DC / DC-перетворювача.

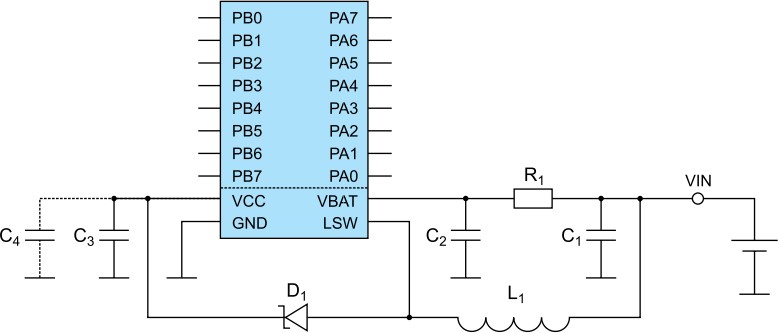


Рис 1.9 - Схема включення перетворювача напруги в мікроконтролері

ATtiny43U

В якості ілюстрації розглянемо реалізацію можливостей технології picoPower на прикладі SAM L21 [9]. Мікроконтролер містить до семи різних тактових доменів (тактових генераторів), що підключаються до будь-якого доступного джерела тактового сигналу. Будь-периферійний блок мікроконтролера має власний мультиплексор, що дозволяє підключити даний блок до будь-якого тактовою домену. На рисунку 1.10 представлена схема тактування блоків мікроконтролера SAM L21.

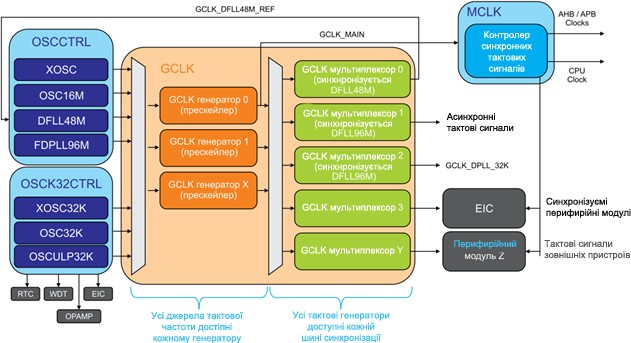


Рисунок 1.10 - Схема тактування мікроконтролера SAM L21

Число активованих джерел тактових сигналів істотно впливає на загальне енергоспоживання мікроконтролера. Залежно від режимів використання периферійних блоків, програмне включення / вимикання генераторів і настройка подільників частоти дозволяють тонко управляти їх споживаної потужністю, не впливаючи на інші блоки як в активному режимі роботи, так і в режимах зниженого енергоспоживання.

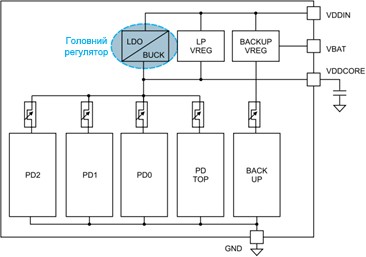


Рисунок 1.11 *-* Схема електроживлення мікроконтролера SAM L21

Для найбільшої ефективності управління споживаної потужністю в залежності від навантаження на мікроконтролер у кристалів серії SAM L21 є три різних регулятора напруги: головний (Main regulator), малої потужності (Low power regulator) і резервного живлення (Вackup regulator). Це проілюстровано на рис. 1.11. Головний регулятор може бути налаштований для роботи в двох режимах - лінійному (LDO) або імпульсному (BUCK). Для імпульсного режиму потрібне використання додаткових зовнішніх компонентів на платі. За замовчуванням система стартує в лінійному режимі, а потім, при наявності апаратної обв'язки, регулятор може бути програмно переключено в імпульсний режим для більшої енергоефективності (рисунок 1.12).

Напруга головного регулятора може динамічно перемикатися в одне з двох значень і визначається максимальною тактовою частотою в системі. Це властивість названо «рівень продуктивності» (PL - Performance Level). Знижена напруга може бути включено, якщо використовується системна тактова частота нижче 12 МГц, тобто в режимі PL = 0 (рис. 1.13).

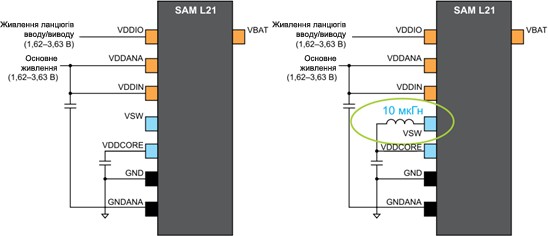


Рисунок 1.12 - Використання головного регулятора в лінійному або імпульсному режимі

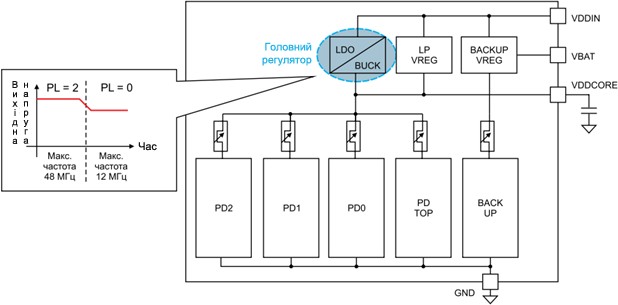


Рисунок 1.13 - Управління рівнями головного регулятора напруги у

SAM L21

У свою чергу, різні периферійні блоки мікроконтролера організовані в групи по живленню (power domains), їх управління здійснюється контролером електроживлення системи (рис. 1.14). При перемиканні в режим Standby мікроконтролер автоматично включає / вимикає напругу по групах,

якщо котрась із груп не потрібно в поточний момент роботи мікроконтролера.

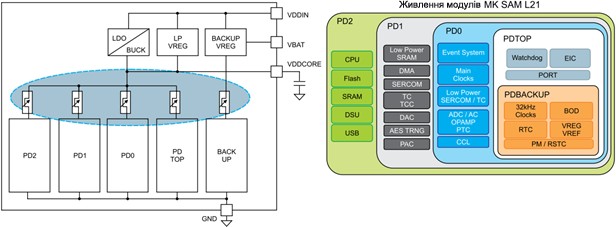


Рисунок 1.14 - Схема розподілу блоків SAM L21 на домени за живленням

Кожна група периферійних блоків може бути переключена в один з трьох станів з живлення:

* + Активне (Active): група повністю запитана і може бути повністю використана.
  + Збереження (Retention): група запитана частково, регістри зберігають інформацію, але периферійна логіка знеструмлена.
  + Виключено (Off): периферійні логічні ланцюги і регістри знеструмлені, стан регістрів не визначене.

Споживання струму у різних груп може відрізнятися, докладні дані наведені в технічній документації на мікроконтролер [9].

Мікроконтролери серії SAM L21 мають п'ять стандартних режимів електроспоживання, що дозволяють динамічно вибирати баланс між споживаної пристроєм потужністю і продуктивністю. Ці режими проілюстровані на рис. 1.15 [10]

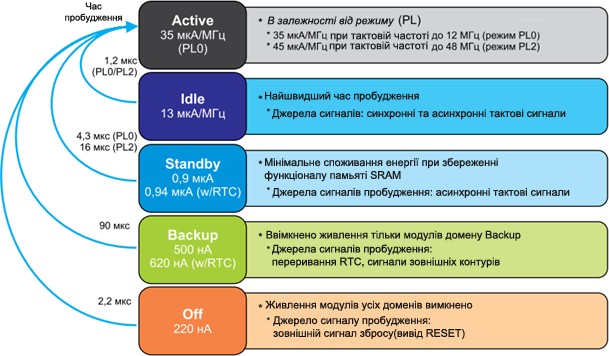
* ACTIVE: всі цифрові і аналогові блоки активні, центральний процесор може виконувати програмний код, все джерела тактового сигналу включені.
* IDLE: всі цифрові і аналогові блоки активні, центральний процесор зупинений, робота тактових генераторів визначається вибором трьох додаткових умов (IDLE0, IDLE1 і IDLE2).
* STANDBY: більшість цифрових і аналогових блоків переведено в статичний стан за винятком блоків, необхідних для функціонування конкретного додатка. Центральний процесор зупинений, тактові генератори вимкнені, вміст пам'яті зберігається. Якщо були сконфігуровані функції периферійних пристроїв SleepWalking, то їх ланцюга підтримуються в активному стані та сигнали можуть передаватися по внутрішнім шинам кристала.
* BACKUP: більшість цифрових і аналогових блоків знеструмлено, працездатний тільки кілька вузлів (система реального часу RTC, регістри BACKUP, ланцюги пробудження від зовнішніх сигналів)
* OFF: всі групи по живленню мікроконтролера знеструмлені, вихід з цього режиму можливий за сигналом на зовнішньому виведенні RESET.

Рисунок 1.15 - Стандартні режими і параметри електроживлення SAM L21

# Висновки за розділом

Виробники мікропроцесорів занепокоєні проблемою їх енергоефективності, тому постійно розробляють нові методи виробництва мікропроцесорів, а також змінюють їх конструкцію таким чином, щоб забезпечити зниження енергоспоживання з кожним новим поколінням. А також надають розробникам та користувачам мікропроцесорних систем можливість використовувати різні режими роботи мікропроцесора, які дають можливість розробляти енергоефективні мікропроцесорні системи. Проте, лише використання різних режимів роботи мікропроцесора не достатньо для того щоб говорити про дійсно енергоефективну систему, адже вона складається не тільки з безпосередньо мікропроцесора, а й з інших частин, які також споживають енергію, тому в наступному розділі буде розглянуто ці, а також інші методи зниження енергоспоживання.

# ДРУГИЙ РОЗДІЛ

**МЕТОДИ ЗНИЖЕННЯ ЕНЕРГОСПОЖИВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ ЗБОРУ ТА ОБРОБКИ ДАНИХ**

* 1. **Підготовка до розробки мікропроцесорної системи**

Під час підготовки до розробки мікропроцесорної системи збору та обробки даних, після того, як був визначений функціонал, та вимоги, формалізовано технічне завдання, ми переходимо до вибору апаратної частини. Вже на цьому етапі необхідно пригадати про енергоефективність системи і тому до вибору апаратної частини підходити з оглядом на енергоспоживання компонентів.

# Вибір апаратної частини

В першу чергу необхідно проаналізувати поставлену задачу і визначити вимоги до мікропроцесора, периферії, інтерфейсів, датчиків, та усіх інших компонентів будучої мікропроцесорної системи. Залежно від цих вимог слід вибрати апаратну частину, при цьому в першу чергу звертати увагу на енергоспоживання.

Чим вище продуктивність мікропроцесора, тим більше він буде споживати. У мікропроцесорі є багато транзисторів, які працюють в так званому ключовому режимі (по типу звичайної кнопки). Основне споживання потужності йде в момент перемикання ключа, тобто, чим менше перемикань - тим менше енергоспоживання. У звичайному режимі роботи сам чіп на 8 МГц споживає приблизно 10,2 мА.

Потужність споживання мікропроцесору в активному режимі є однією з найважливіших характеристик контролера. Вона в значній мірі залежить від напруги живлення мікропроцесора і частоти тактування. Залежно від

діапазону живлячих напруг все мікропроцесори можна розділити на три основні групи:

* мікропроцесори з напругою живлення 5,0 В ± 10%.
* мікропроцесори з розширеним діапазоном напруг живлення: від 2,0

... 3,0 В до 5,0-7,0 В.

* мікропроцесори із зниженою напругою живлення: від 1,8 до 3 В.

Залежність струму споживання від напруги живлення мікропроцесора майже прямо пропорційна. Тому зниження напруги суттєво знижує потужність споживання мікропроцесора. Необхідно, однак, мати на увазі, що для багатьох типів мікропроцесорів з пониженням напруги живлення зменшується максимально допустима частота тактування, тобто виграш в споживаної потужності супроводжується зниженням продуктивності системи.

Відповідно, необхідно підібрати мікропроцесор, продуктивності якого буде достатньо для поставленого завдання, але не буде значно її перевищувати, а також вибирати мікропроцесор із зниженою напругою живлення.

Таким же чином необхідно вибирати інші компоненти мікропроцесорної системи.

# Джерела живлення

Живлення мікропроцесорної системи може проводитися від промислової, або побутової мережі, а так само від елементів живлення, таких як акумулятори, або батарейки.

У тих випадках, коли необхідно розробити автономну мікропроцесорну систему, живиться від акумуляторів, необхідно враховувати, що найбільш ефективні - літій-іонні акумулятори, які найчастіше мають вихідну напругу 3.7В.

Якщо в такому випадку ми будемо використовувати мікропроцесор та іншу компоненти системи, що працюють не з 3.7В, виникне необхідність використовувати елементи, що підвищують, або знижують напругу, які не можуть мати 100% ККД, що негативно позначиться на енергоефективності. Тому варто використовувати мікропроцесор, з такою ж самою напругою, що і у акумулятора.

# Живлення елементів мікропроцесорної системи

У мікропроцесорних системах наявні такі компоненти, які використовуються не постійно, а з певними часовими проміжками, або з виконанням певних умов, такі як інтерфейси, таймери, датчики, різні периферійні пристрої. Під час роботи вони є енергоспоживачами і при цьому можуть споживати енергію в той час, поки мікропроцесорна система ніяк їх не використовує.

Отже, частина енергії витрачатиметься на роботу елементів, котрі не виконують ніякої корисної дії. Для того що б уникнути цього, необхідно потурбуватися можливістю тимчасово відключати від джерела живлення елементи, коли немає необхідності в їх використанні.

Для цього, живлення елементів можна здійснювати не на пряму від джерела живлення, а підключати до цифрових виходів мікропроцесора, в тих випадках, якщо вихідні характеристики струму цифрового виходу мікропроцесору задовольняють умов енергоспоживача, а якщо ні, то подавати живлення на цей елемент через реле, або транзистор, якай буде керуватися мікропроцессором. Таким чином можна програмно керувати живленням елементів, котрі не використовуються постійно.

Розглянемо на прикладі, як буде виглядати таке підключення. Припустимо, в мікропроцесорній системі, яка живиться від батарейки на 3В, до мікроконтроллеру Arduino Nano(версії на 3В) підключається аналоговий

датчик температури, при звичайному підключенні схема виглядає так, як показано на рисунку 2.1.

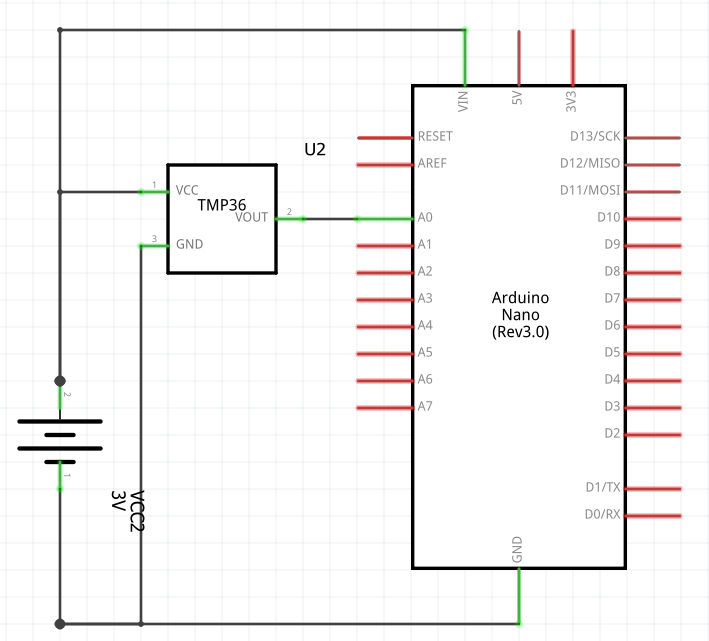


Рисунок 2. 1 - Схема підключення датчику безпосередньо до джерела

струму

При такому підключенні датчик буде працювати завжди, незалежно від того, чи необхідні у певний час данні о температурі, чи ні. Якщо немає потреби стежити за температурою постійно, а лише періодично, або за якихось умов, правильніше буде підключити його до живлення не до джерела струму, а до цифрового виходу(рисунок 2.2), для того щоб програмно, посилаючи на цифровий вихід D10 сигнал HIGH, вмикати датчик лише за необхідності. Таким же чином можна керувати цілою групою датчиків (рисунок 2.3), або інших елементів мікропроцесорної системи.

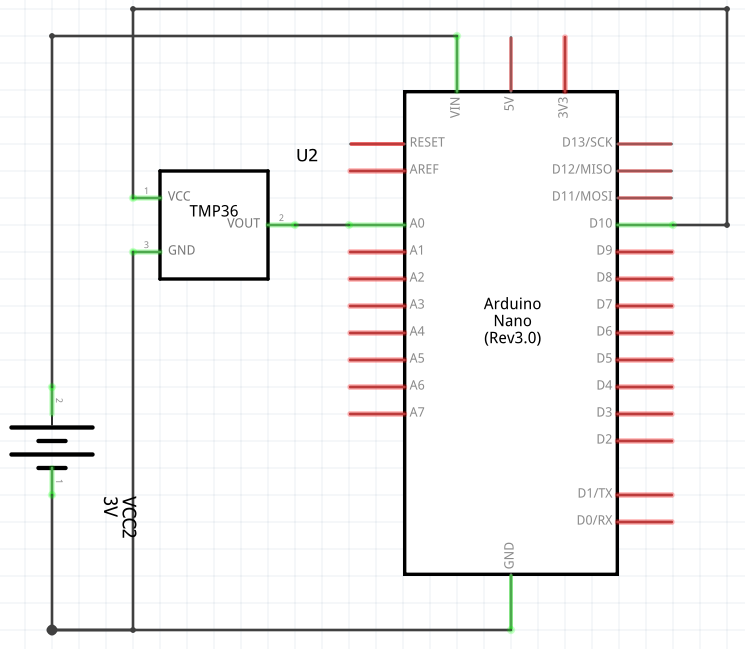


Рисунок 2.2 Схема підключення датчику від цифрового виходу

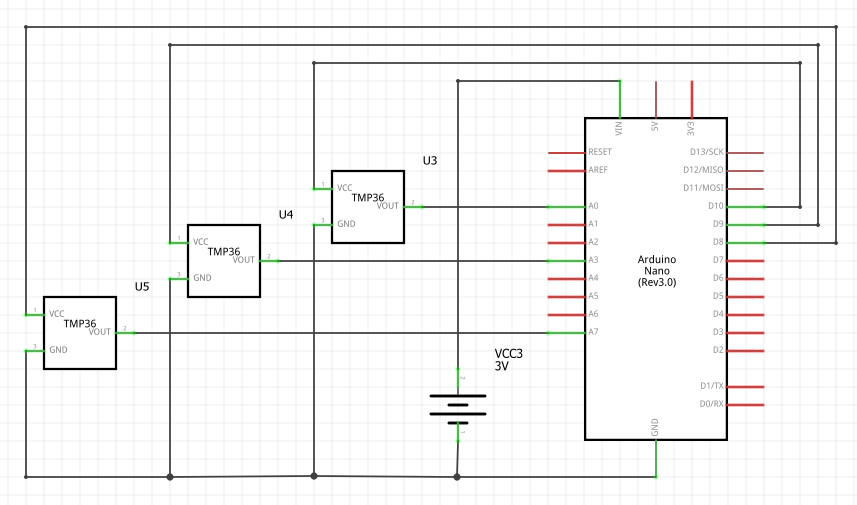


Рисунок 2.3 - Схема підключення групи датчиків

# Режими роботи мікропроцесора

У попередньому розділі розглядався варіант роботи мікропроцесорної системи збору та обробки даних, при котрому деякі елементи системи не використовуються постійно і тому їх можна вимикати на той час, поки в них немає необхідності. Бувають випадки, коли від усієї мікропроцесорної системи не очікується ніяких дій. Такі випадки можуть виникати, коли система повинна обробляти чи збирати інформацію через певні проміжки часу, або лише після спрацьовування певного тригера, наприклад переривання від певного датчику. У таких випадках, під час «простою», в якому від мікроконтролера не потрібні ніякі дії (а часто такі «простої» займають набагато більше часу, ніж час, необхідний для обробки даних), він все одно продовжує споживати енергію. Для того що б уникнути втрат енергії в цьому випадку, необхідно переводити мікроконтролер в режим економії енергії на той час, поки від нього не потрібні ніякі дії.

Сучасні мікроконтролери надають користувачеві великі можливості в плані економії енергоспоживання і мають, як правило, такі основні режими роботи:

* активний режим (Run mode) - основний режим роботи мікроконтролера. В цьому режимі мікроконтролер виконує робочу програму і все його ресурси доступні. Споживана потужність має максимальне значення Prun. Більшість сучасних мікроконтролерів виконано по КМОН - технології, тому потужність споживання в активному режимі сильно залежить від тактової частоти;
* режим очікування (Wait mode, Idle mode або Halt mode). В цьому режимі припиняє роботу центральний процесор, але продовжують функціонувати периферійні модулі, які контролюють стан об'єкта управління. При необхідності сигнали від периферійних модулів переводять мікроконтролер в активний режим, і робоча програма формує необхідні керуючі впливи. Перехід мікроконтролера з режиму очікування в робочий

режим здійснюється по перериванню від зовнішніх джерел або периферійних модулів, або при скиданні мікроконтролера. У режимі очікування потужність споживання мікроконтролера Pwait знижується в порівнянні з активним режимом в 5 - 10 разів;

* режим зупинка (Stop mode, Sleep mode або Power Down mode). В цьому режимі припиняє роботу, як центральний процесор, так і більшість периферійних модулів. Перехід мікроконтролера зі стану зупинки в робочий режим можливий, як правило, тільки по перериваннях від зовнішніх джерел або після подачі сигналу скидання. В режимі зупинки потужність споживання мікроконтролера Pstop знижується в порівнянні з активним режимом приблизно на три порядки і складає одиниці мікроват.

Два останніх режими називають режимами зниженого енергоспоживання. При цьому необхідно мати на увазі, що режими очікування і зупинки істотно відрізняються часом переходу з режиму зниженого енергоспоживання в активний режим. Вихід з режиму очікування зазвичай відбувається протягом 3 - 5 періодів синхронізації мікроконтролера, в той час як затримка виходу з режиму зупинки становить кілька тисяч періодів синхронізації. Крім зниження динаміки роботи системи значний час переходу в активний режим є причиною додаткової витрати енергії.

Розглянемо варіант, в якому необхідно розробити мікропроцесорну систему, яка з певною періодичністю, наприклад раз на хвилину, повинна за допомогою датчиків фіксувати зміну температури у навколишньому просторі.

Можна побачити з умов поставленої задачі, ні сам датчик, ні мікроконтролер не використовуються на протязі періоду очікування в одну хвилину. Тому виникає можливість заощадити енергію в момент простою системи. Для цього необхідно подавати живлення на датчик, лише тоді, коли необхідно зчитати інформацію о температурі, для цього необхідно підключити живлення на датчик не від джерела струму, а від цифрового виходу мікроконтролера (рис 2.4), і програмно керувати поданням струму на

цей цифровий вихід, а також програмно переводити мікропроцесор у режим зупинки.

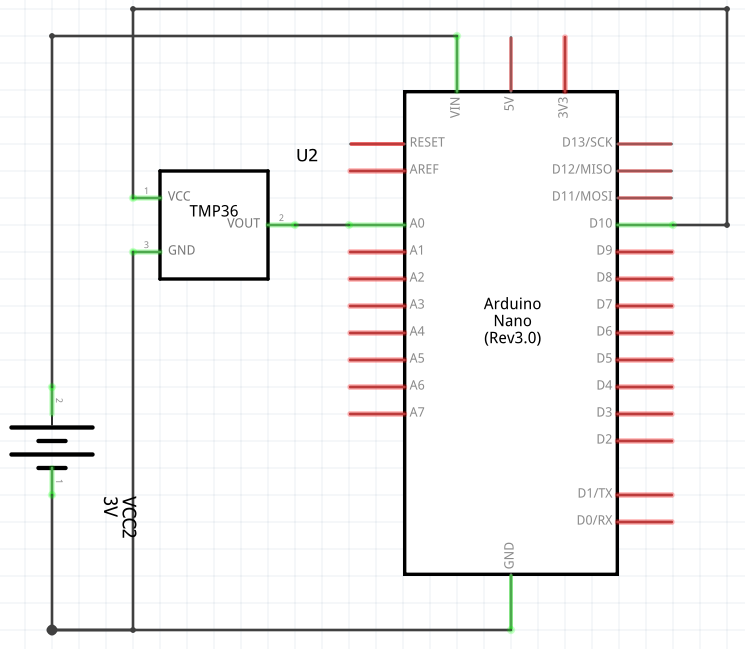


Рисунок 2.4 - Схема підключення датчику від цифрового виходу Блок-схема алгоритму роботи програми керування мікроконтролером у

цьому випадку буде виглядати як показано на рисунку 2.5.

Як витікає з цієї блок-схемі, на початку роботи програми ми вимикаємо датчик, це означає, що він не буде споживати енергію на момент простою. Після цього задається відлік таймеру в одну хвилину і переводимо мікроконтролер у режим зупинки, із якого він вийде лише за перериванням по таймеру, це також означає, що під час очікування він буде споживати значно менше енергії. Після того, як таймер надсилає переривання до мікроконтролеру, він переходить з режиму зупинки в робочій режим, подає на цифровий вихід D10 сигнал HIGH, що вмикає датчик, після цього отримує з датчика показники на аналоговому вході A0, обробляє дані і знову переходить до початку, тобто припиняє роботу на одну хвилину.

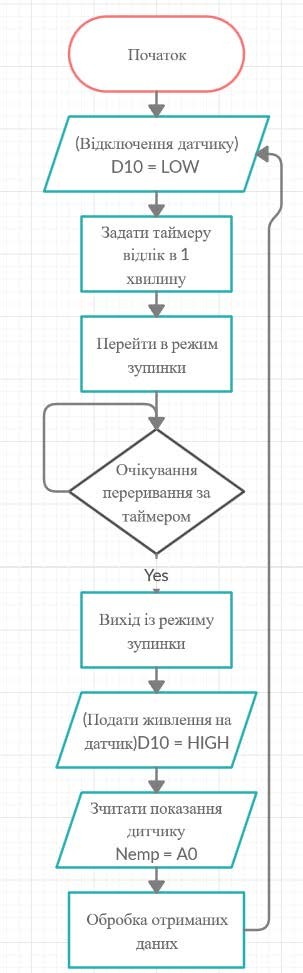


Рисунок 2.5 - Блок-схема алгоритму роботи мікропроцесорної системи

Розглядаючи цей алгоритм, можна визначити що в режимі зупинки мікропроцесорна система перебуває значно більший проміжок часу, ніж у робочому режимі, з цього можна зробити висновок, що ми добилися значної економії енергії, в порівнянні з варіантом, при якому датчик і мікроконтролер знаходились би у робочому режимі постійно.

# Віддалена обробка та зберігання даних

Найчастіше мікропроцесорні системи збору та обробки даних, збирають та обробляють інформацію для подальшого використання вже іншим пристроєм, мікропроцесорною системою, комп’ютером, сервером, користувачем. Тому вони підключені до отримувача інформації за допомогою певного інтерфейсу, Bluetooth, Wifi, USB, Ethernet, та інших.

З цього з’являється ще одна можливість заощадити енергію для роботи мікропроцесорної системи. Адже якщо використовувати інформацію буде не сама мікропроцесорна система, а інший користувач, то можна зберігати та обробляти інформацію вже на стороні отримувача, а не за рахунок мікропроцесорної системи, для якої питання збереження енергії біль критичне.

Якщо необхідно зберігати велику кількість інформації на протягом великого проміжку часу і цю інформацію ми зберігаємо у самій мікропроцесорній системі, нам знадобляться об’ємні носії інформації, а крім того, на зчитування та надсилання до отримувача усієї інформації буде затрачено багато часу, на протязі котрого мікропроцесорна система буде працювати у звичайному робочому режимі і витрачати велику кількість енергії.

Якщо ж відмовитись зберігати інформацію на стороні мікропроцесорної системі, а лише надсилати нові данні при необхідності, не прийдеться зберігати велику кількість даних, а також нові дані, котрі мі будемо надсилати до отримувача будуть не такими об’ємними, що значно

знизить час на їх відправку, а значить і системі прийдеться менше працювати у робочому режимі.

Те саме стосується й обробки даних, якщо необхідно не просто перетворити дані, отримані з датчиків до необхідного вигляду, а проаналізувати їх, наприклад построїти графіки, при можливості цей процес обробки інформації краще виконувати не за рахунок мікропроцесорної системи, а на стороні отримувача інформації.

Усе це в першу чергу підходить тоді, коли мікропроцесорна система має постійне підключення до серверу, який зможе в любий момент часу отримувати та обробляти дані з мікропроцесорної системи. Але бувають випадки, коли мікропроцесорна система та отримувач не можуть мати постійного підключення. Наприклад система розумного будинку. У ситуації, коли система знаходиться в режимі зупинки, а користувач розумного будинку за допомогою свого смартфону хоче отримати актуальну інформацію, або навпаки, неможливе підключення до смартфону, котрий являється отримувачем інформації відсутній. В такому випадку мікропроцесорна система не може надсилати інформацію до користувача, а користувач не може отримати актуальну інформацію, із-за чого частина даних буде втрачена.

Для того щоб вирішити цю проблему, не повертаючись до збереження та обробки усіх даних на стороні мікропроцесорної системи, необхідно ввести посередника між мікропроцесорною системою та користувачем, який завжди буде в робочому режимі та зможе приймати усі дані надіслані мікропроцесорною системою.

Виникає складність у пошуку такого посередника для звичайного користувача. Оптимальним рішенням буде використання для надсилання данних протоколу MQTT, а в якості посередника, відповідних віддалених брокерів.

Протокол MQTT - Message Queuing Telemetry Transport - протокол для передачі послідовності повідомлень з телеметричними даними, тобто інформації від датчиків температури, вологості, освітленості і ін. [20]

Основні риси протоколу MQTT:

* обмін повідомленнями відбувається за принципом "відправник -

отримувач" (Pub-Sub);

* розмір заголовка повідомлення становить 2 байта, а корисне навантаження може варіюватися від 1 байта до 260 Мбайт;
* в протоколі закладена можливість вибору одного з трьох рівнів обслуговування.

Принцип "відправник - отримувач ". Відмінною особливістю принципу "відправник - отримувач " від клієнт-серверного підходу є те, що клієнти, що посилають повідомлення (відправники, Publisher), і клієнти, які беруть повідомлення (отримувач, Subscriber), як правило, розділені. Поділ може бути організовано в трьох площинах:

* простір - відправник і отримувач не зобов'язані знати один одного;
* час - відправник і отримувач не повинні бути включені в один і той же час;
* синхронізація - операції на обох сторонах не повинні припинятися протягом публікації або отримання інформації.

Відправник і отримувач не передають один одному повідомлення безпосередньо, не встановлюють прямий контакт, можуть не знати про існування один одного. Координує і керує передачею повідомлень від відправника до отримувача і від отримувача до відправника брокер (Broker).

Розпаралелювання операцій на брокера є другою важливою особливістю принципу взаємодії "відправник-отримувач".

MQTT-клієнт - це пристрій, оснащений мікроконтролером, що підтримує стек TCP / IP. Клієнтські бібліотеки MQTT доступні для великого

числа мов програмування, наприклад Android, Arduino, C, C ++, C #, Go, iOS,

Java, JavaScript, NET.

Брокер є основним елементом системи "відправник-отримувач". Він відповідає за прийом всіх повідомлень, їх фільтрацію, прийняття рішення про те, кому цікаві ці повідомлення, і, в кінцевому підсумку, за пересилку повідомлень всім клієнтам-отримувачам.

Серед серверних реалізацій брокера можна виділити IBM WebSphere MQ; відкрите ПЗ Mosquitto; рішення, засноване на хмарному сервісі Eurotech Everywhere Device Cloud; легко масштабується і високопродуктивний відкритий сервер emqttd, дозволяє обслуговувати 1,3 мільйона з'єднань; брокер HiveMQ, що забезпечує корпоративну безпеку і максимальну масштабованість.

Спрощений процес обміну інформацією можна описати таким чином:

* Відправник передає повідомлення з даними (наприклад, інформація з датчиків температури) на брокер, вказуючи при цьому тему (Topic), до якої ці дані відносяться (наприклад, "Temp").
* Брокер аналізує, які з отримувачів мають підписку на певні теми, в даному випадку - на тему "Temp".
* Отримувачам, які підписані на тему "Temp", брокером буде відправлено повідомлення з інформацією від датчиків температури.

Таким чином, для того щоб розв’язати проблему зі збереженням та обробкою інформації не на стороні мікропроцесорної системи, для якої питання енергоефективності дуже важливе, ми можемо підключити систему до мережі Інтернет і надсилати усі дані за допомогою протоколу MQTT на один з багатьох онлайн брокерів, який і буде зберігати необхідну інформацію.

З цього виходить, що мікропроцесорна система, може не зберігати і не обробляти велику кількість інформації, а в ті моменти, коли виходить із режиму зупинки в робочий режим і отримує та обробляє тільки нову актуальну інформацію, відразу надсилає її до брокера. А користувач в любий

момент часу, навіть коли система знаходить в режимі зупинки і не доступня для зчитування інформації, отримати усю необхідну інформацію від брокера.

# Висновки за розділом

У розділі розглянуто основні методи зниження енергоспоживання мікропроцесорною систему збору та обробки даних, такі як:

* Вибір апаратної частини та джерела живлення - де визначили, що важливо турбуватися проблемою енергозбереження вже на початку підготовки до розробки мікропроцесорної системи. При виборі апаратної частини необхідно вибирати елементи з оглядом на їх енергоспоживання.
* Живлення елементів мікропроцесорної системи – ми побачили, що деякі елементи, які споживають енергію не завжди використовуються системою і в таких випадках, при їх тимчасовому вимиканню, можна заощадити значну кількість енергії.
* Режими роботи мікроконтролера – визначили, що далеко не завжди нам необхідний робочий мікроконтролер, а значить його теж можно тимчасово перемикати в режими зниженого енергоспоживання.
* Віддалене зберігання та обробка даних – встановили, що для зниження енергоспоживання мікропроцесорної системи, зберігання та обробку інформації можна перенести на посередника, або отримувача.

# 3 ТРЕТІЙ РОЗДІЛ

**ІТОГОВИЙ АНАЛІЗ МЕТОДІВ ЗНИЖЕННЯ ЕНЕРГОСПОЖИВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ ЗБОРУ І ОБРОБКИ ДАНИХ**

У попередніх розділах було розглянуто основні методи зниження енергоспоживання мікропроцесорних систем збору та обробки даних. Для підтвердження їх ефективності та аналізу, необхідно розглянути їх використання на прикладі. При цьому, для достовірності результатів, необхідно одну й ту саму задачу реалізувати спочатку без використання методів зниження енергоспоживання, а потім використовуючи різні методи, таким чином, розглянувши результати замірів енергоспоживання можна буде порівняти і підтвердити ефективність використаних методів.

У якості приклада буде розглянута така задача: необхідно розробити мікропроцесорну систему, яка за допомогою фоторезистору, раз на хвилину збирає інформацію о рівні освітлення навколишнього середовища, та надсилає дані на сервер.

Буде розглянуто три варіанти реалізації поставленої задачі:

* стандартне рішення;
* рішення з використанням режимів економії енергії мікроконтролеру;
* рішення з використанням додаткових методів зниження енергоспоживання.

# Стандартне рішення

Для виконання поставленої задачі буде використана така апаратна частина:

* Arduino Uno;
* WiFi Shield CC3000 для Arduino UNO;
* фотоелемент GL5528;
* резистор 0.25Вт, 10 кОм;
* акумулятор Eneloop AAА Ni-MH, 750mAh;
* датчик струму та напруги INA219.

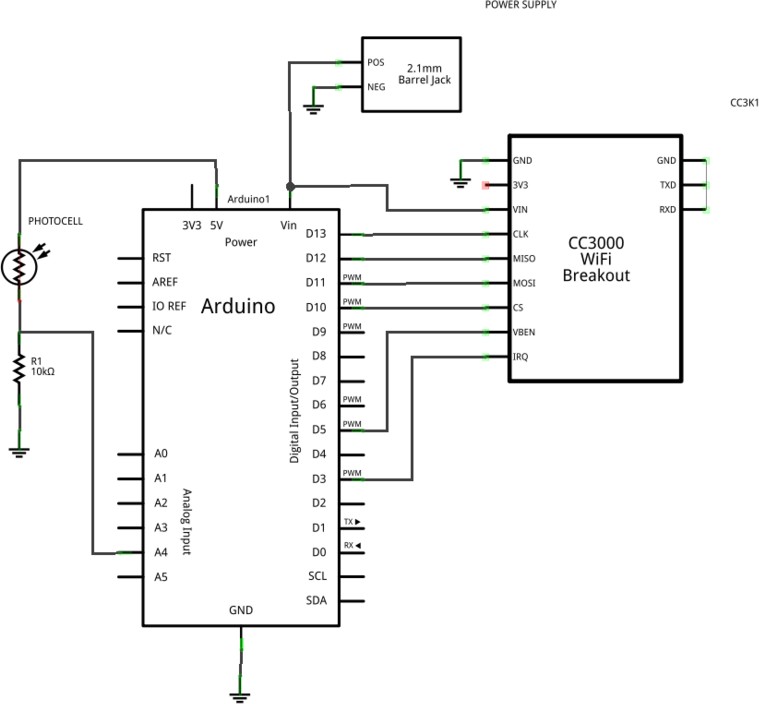
На рисунку 3.1 зображена принципова схема даної мікропроцесорної системи збору та обробки даних. Необхідно звернути увагу на те, що фоторезистор підключений не до цифрового виходу Arduino, а до виходу живлення, тобто він буде працювати завжди, під час роботи мікроконтролеру.

Рисунок 3.1 – Принципова схема мікропроцесорної системи

Після збирання схеми, на Arduino було завантажено скетч Example\_1\_No\_Optimizations (Лістинг А). Згідно з кодом програми, спочатку виконується функція setup() в якій відбувається підключення до мережі WiFi, основна функція loop() викликає затримку на одну хвилину, після чого відправляє нові дані на сервер за допомогою плати CC3000, після чого цикл починається спочатку.

Поточне споживання цієї мікропроцесорної системи буде високим, оскільки і Arduino, і CC3000, і фоторезистор постійно працюють на повній потужності, навіть під час очікування між вимірюваннями. Для підтвердження цього припущення до мікропроцесорної системи був підключений датчик INA219, після чого були проведені заміри споживання струму на протязі 5 хвилин. Результати замірів можна побачити на рисунку 3.2.

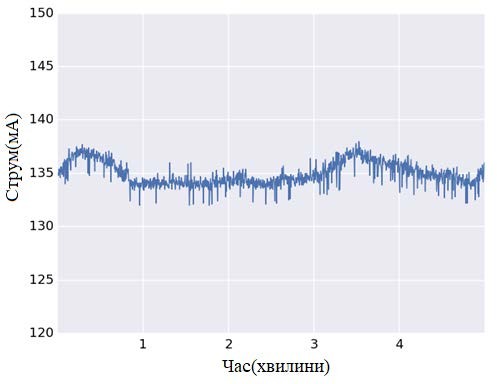


Рисунок 3.2 – Графік споживання струму

Згідно з графіком, нинішнє споживання енергії досить стабільне та становить приблизно 135 мА. Це вимірювання підтверджує, що під час очікування та в момент збору та надсиланням даних на сервер, мікропроцесорна система споживає однакову кількість енергії.

За допомогою показника середнього споживання струму можна оцінити і перевірити, як довго мікропроцесорна система буде працювати від акумуляторів.

Для того, щоб розрахувати очікуваний ресурс акумулятору, необхідно поділити ємність акумулятору (750 мА\*г) на середнє споживання струму. Таким чином очікуваний ресурс акумулятора становить:

750 мА\*г / 135 мА = 5,6 годин

Для вимірювання фактичного ресурсу акумулятора був застосований датчик INA219 для вимірювання напруги акумулятору протягом часу.

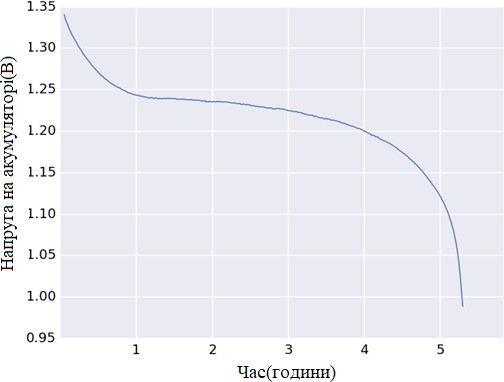
На рисунку 3.3 зображено графік зміни напруги акумулятору у часі. Напруга одного акумулятору була розрахована діленням загальної напруги в акумуляторній батареї на кількість акумуляторів, в цьому випадку 6.

Рисунок 3.3 - Графік напруги акумулятора у часі

При повному заряді напруга в акумуляторі починається з близько 1,35 вольт. З часом напруга падає за кривою, що представлена на графіку.

Як тільки напруга в акумуляторі опуститься нижче 1 вольта, акумулятор вичерпано, і обладнання перестане працювати.

Фактичний термін служби акумулятора виходить досить близьким до очікуваного терміну служби акумулятора. Такі фактори, як вік батарей, температура навколишнього середовища та кількість циклів заряду / розряду можуть впливати на ємність і можуть пояснити певну різницю у очікуваній та фактичній ємності.

Фактичний ресурс акумулятора - 5,3 години.

# Рішення з використанням режимів економії енергії мікроконтролеру.

У цьому варіанті реалізації буде використаний режим економії енергії power-down – найбільш економічний режим, доступний на Arduino. У цьому режимі майже всі компоненти мікроконтролеру відключені, і лише зовнішній сигнал переривання, TWI сигнал або сторожовий таймер watchdog можуть перевести мікроконтроллер до робочого режиму. Оскільки, згідно з поставленим завданням пробудження із сну відбувається через певний проміжок часу, буде використаний таймер watchdog, щоб повертати мікроконтролер до робочого режиму.

Сторожовий таймер – це компонент AVR, який періодично скидається системою управління. Якщо скидання не відбулося протягом деякого інтервалу часу, відбувається примусове перезавантаження системи. Однак, змінивши реєстри контролю сторожового таймеру, можна використовувати його як таймер, який

переводить мікроконтроллер із режиму power-down в звичайний робочий режим замість того, щоб перезавантажувати його.

Принципова схема не буде відрізнятися від схеми стандартного рішення. Зміни відбулись тільки у програмному коді скетчу (лістинг Б).

Сторожовий таймер налаштовується як таймер переривання. Всередині функції setup() змінюється реєстр контролю сторожового таймеру, тому він функціонує як таймер, який спрацьовує як переривання. Сторожовий таймер має обмежену конфігурацію періодів, тому для цього прикладу використовується максимальний період часу приблизно 8 секунд.

Arduino вводиться в режим power-down, в період очікування між вимірюваннями. Коли сторожовий таймер спрацьовує кожні 8 секунд, Arduino переходить в робочий режим. Так як вимірювання необхідно проводити раз на хвилину, а не раз на 8 секунд, використовується лічильник, який підраховує ітерації переходів в робочий режим. Доки не настане 7 ітерація Arduino одразу повертається в режим power-down. Після досягнення

7 ітерацій (приблизно 56 секунд) реєструється вимірювання датчика. Нарешті, Arduino повертається в power-down, і чекає, коли сторожовий таймер знову спрацює.

CC3000 відключений, поки Arduino знаходиться в режимі power-down. У функції setup() зв'язок з CC3000 ініціюється, і тоді викликається функція wlan\_stop(), щоб перевести CC3000 у стан економії енергії. Коли Arduino на 7 ітерації переходить в робочий режим, здійснити вимірювання, CC3000 вмикається за допомогою виклику wlan\_start(), підключається до бездротової мережі для надсилання вимірювань і, нарешті, вимикається, знову викликом функції wlan\_stop().

В результаті було отримано новий графік споживання струму у часі

(рисунок 3.4). Як можна встановити з графіку, існують періодичні піки

~ 135мА під час реєстрації вимірювань, а в той час, коли мікроконтролер перебуває в режимі економії енергії ~ 19мА. Відразу видно, що використання режиму економії енергії мікроконтролеру має високий вплив на

енергоефективність, оскільки середнє споживання струму впало з 135 мА до приблизно 45 мА.

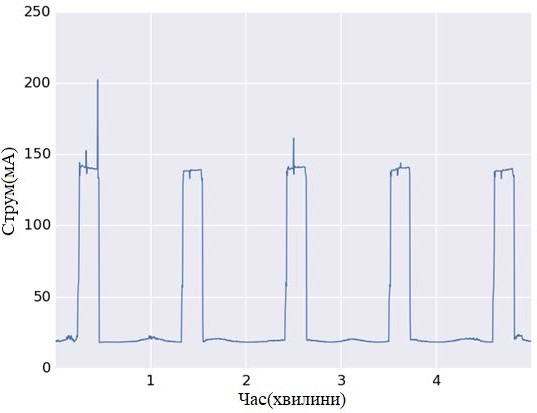


Рисунок 3.4 - Графік споживання струму

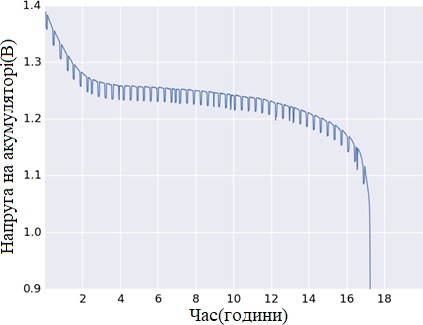


Рисунок 3.5 - Графік напруги в акумуляторі з плином часу

Очікуваний термін служби акумулятора розраховується тим же методом, що і в попередньому випадку і становить 775 мА\*г / 45 мА = 17,2 години.

Ці самі дані підтверджуються й отриманим графіком напруги в акумуляторі з плином часу (рисунок 3.5), отже ресурс акумулятору виріс з 5,3 годин, до 17,2 годин.

# Рішення з використанням додаткових методів зниження енергоспоживання

В мікропроцесорній системі, що була розглянута у попередньому варіанті, присутні елементи, котрі споживають значну кількість енергії, доки мікроконтролер знаходиться в режимі економії енергії, для того, щоб досягнути більшої енергоефективності, необхідно або програмно вимикати їх, або замінити на більш енергоефективні аналоги, або зовсім усунути, при можливості. Такими елементами являються:

Індикатори живлення. Arduino, і CC3000 мають невеликі світлодіоди, які являються споживачами енергії. Ці світлодіоди є корисними індикаторами, коли мікропроцесорна система не розраховується на енергоефективну роботу, але в іншому випадку, це зайві енергоспоживачі, кожний світлодіод споживає приблизно 8-10 мА струму, при цьому для виконання поставленої перед мікропроцесорною системою задачі не використовується.

Регулятори напруги. Лінійні регулятори напруги встановлені на Arduino і CC3000 не особливо енергоефективні, їх струм спокою сягає приблизно 5-10 мА для кожного.Фотоелементи та резистори, які використовуються в мікропроцесорній системі. Фотоелемент та підключений послідовно з ним резистор, як і світлодіоди, є ще одним джерелом постійного енергоспоживання. Навіть підтягуючий резистор на Arduino споживає близько 5 мА.

Для того щоб зменшити енергоспоживання необхідно виконані наступні дії:

* Видалити з Arduino світлодіодні індикатори, а також замінити старий підтягуючий резистор на 1-мегаомний, щоб зменшити споживання струму.
* Оновити регулятори напруги. Замість старих, будуть використані регулятори LT1529, щоб забезпечити 5 вольт і 3,3 вольт, необхідні Arduino і CC3000 відповідно. Ці регулятори мають дуже низький струм спокою в 0,05 мА. Регулятори також можна повністю вимкнути. Це означає, що Arduino може відключити живлення до CC3000, поки він не використовується і забезпечити відсутність зайвого споживання струму від CC3000.
* Живлення на датчик лише під час вимірювань. Для цього необхідно підключити живлення датчику до цифрового виходу Arduino замість 5В виходу, для того, щоб з’явилась можливість програмно керувати живленням датчику.

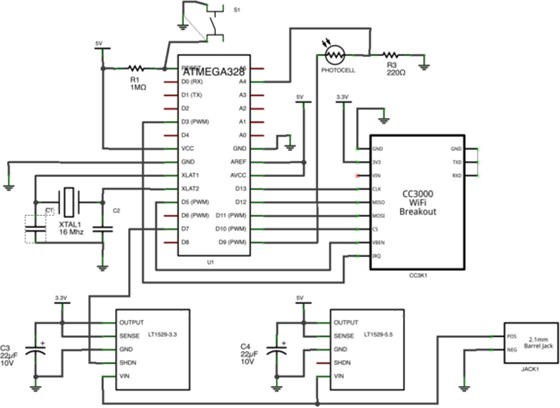
Таким чином, принципова схема мікропроцесорної системи змінилася і тепер виглядає так, як показано на рисунку 3.6

Рисунок 3.6 - Принципова схема мікропроцесорної системи

Програмний код скетчу (лістинг В) майже не відрізняється від попереднього варіанту, крім декількох поправок, зроблених для керування живлення елементів системи:

* В залежності від режиму роботи CC3000, вмикається і вимикається 3,3-вольтовий регулятор, завдяки посиланням на цифровий вихід D7 сигналів HIGH, або LOW (HIGH - ввімкнення, LOW - вимкнення).
* На цифровий вихід D9 значення HIGH встановлюється лише перед тим, як починати вимірювання, а потім знову встановлюється LOW. Це забезпечує роботу фотоелементу лише тоді, коли це необхідно для вимірювань.

Після усіх зазначених змін, було отримано новий графік споживання струму мікропроцесорною системою (рисунок 3.7), згідно з яким, споживання струму під час очікування складає лише 0.9мА, тож економія енергії в цей час майже 20мА, у порівнянні з попереднім варіантом. Під час процесу вимірювання, споживання струму також трохи зменшилося і становить 132мА. У цілому, значення середнього споживаного струму становить приблизно 23.3мА.

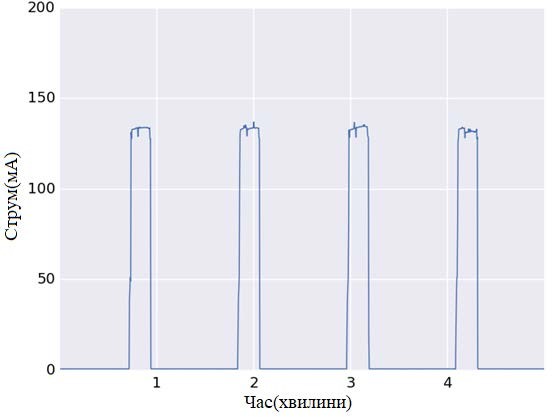


Рисунок 3.7 - Графік споживання струму

Таким чином, за допомогою отриманих даних можна розрахувати очікуваний ресурс роботи акумулятора, у цьому випадку він буде складати 775 мА\*г / 23,3 мА = 33,3 години

Дочекавшись розряджання акумуляторів, отримаємо графік напруги в акумуляторі з плином часу (рисунок 3.8), з якого можемо побачити, що фактичний термін служби склав 28 годин, значно більше, ніж у попередніх варіантах.

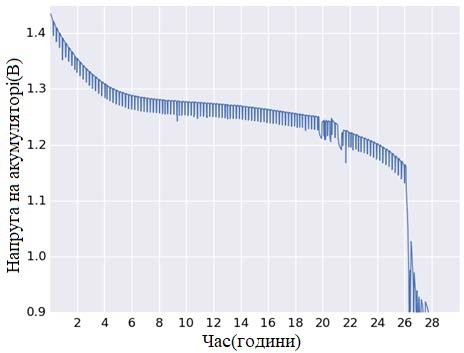


Рисунок 3.8 - Графік напруги в акумуляторі з плином часу

# Висновки за розділом

Було розглянуто 3 варіанти рішення поставленої перед мікропроцесорною системою задачі.

При стандартному рішенні, без використання методів зниження енергоспоживання, середнє споживання струму мікропроцесорною системою становить 135мА, час роботи використаних акумуляторів у цьому випадку становить 5.3 години.

При використанні режиму економії енергії мікроконтроллера power- down, середнє споживання струму становить 45мА, що в 3 рази менше, за середнє споживання стандартного варіанту, а час роботи акумуляторів становить 17.2 години, що в 3.2 рази більше, за час роботи акумуляторів стандартного варіанту. Із цього можна зробити висновок, о великій ефективності використання режимів економії енергії мікроконтролеру.

У третьому варіанті, окрім режиму power-down, були використані додаткові методи зниження енергоспоживання мікропроцесорної системи, такі як: заміна деяких елементів на більш енергоефективні, видалення із мікропроцесорної системи елементів, які не використовуються при поставленій задачі, але споживають енергію, а також програмне керування живленням елементів, котрі тимчасово не використовуються. У цьому випадку, середнє споживання струму становить 23.3мА, що в 5.8 разів менше, за споживання у стандартному варіанті, а час роботи на акумуляторах становить 28 годин, що в 5.3 рази довше, ніж у стандартному варіанті. Що доказує ефективність використаних методів зниження енергоспоживання мікропроцесорної системи збору та обробки даних.

# ВИСНОВКИ

Аналітичний огляд проблеми енергоефективності мікропроцесорної системи збору та обробки даних показав, що питання енергоєфективності в першу чергу актуальне для систем, котрі повинні працювати автономно, без можливості підключення до загальної енергомережі, або систем, котрі повинні продовжувати працювати при тимчасовій відсутності напруги у загальній мережі.

Розглянута проблема енергоспоживання мікропроцесорних систем, та безпосередньо мікропроцесорів, у результаті було встановлено, як саме виробники мікропроцесорів працюють над розробкою нових методів підвищення енергоефективності, у тому числі надаючи розробникам та користувачам мікропроцесорних систем використовувати режими економії енергії мікропроцесора.

Були розглянуті основні методи зниження енергоспоживання мікропроцесорною системою, такі як: використання режимів економії енергії мікроконтролера, підхід до вибору апаратної частини, керування живленням елементів мікропроцесорної системи, віддалене зберігання та обробка інформації.

Проведені експериментальні дослідження методів зниження енергоспоживання мікропроцесорною системою збору та обробки даних. Під час дослідження було розглянуто 3 варіанти рішення однієї задачі з використанням різних методів зниження енергоспоживання. Для підтвердження ефективності обраних методів на прикладі, в першу чергу була розглянута мікропроцесорна система, виконана без використання методів зниження енергоспоживання. У другому варіанті була розглянута мікропроцесорна система виконана з використанням режиму економії енергії мікроконтролера power-down. У третьому варіанті була розглянута мікропроцесорна система, в котрій окрім режиму економії енергії

мікроконтролера power-down, були використані додаткові методи зниження енергоспоживання.

У результаті проведення експериментів було доказано високу ефективність використання режимів економії енергії мікроконтролера, а також було доказано ефективність використання додаткових методів зниження енергоспоживання мікропроцесорної системи, таких як: підхід до вибору апаратної частини (заміна не енергоефективних елементів елементів більш енергоефективними) та керування живленням елементів мікропроцесорної системи (видалення із мікропроцесорної системи елементів, які не використовуються при поставленій задачі, але споживають енергію, а також програмне керування живленням елементів, котрі тимчасово не використовуються).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Илья Тарасов Оценка потребляемой мощности и выбор системы питания: Компоненты и Технологии № 2 2009. – 162 с.
2. Рабаи Ж. М., Чандракасан А., Николич Б. Цифровые интегральные схемы. Методология проектирования / Пер. с англ. М.: ИД Вильямс, 2007.
3. А. БумАгин, Е. Гладкова, А. Гондарь, М. Куляс, А. Руткевич, В. Стешенко, М Тайлеб, Г. Шишкин. Методы снижения энергопотребления в строго самосинхронных микропроцессорных схемах: Компоненты и Технологии № 9 2009. – С. 114.
4. Белоус А. И., Мурашко И. А., Сякерский В. С. Методы минимизации энергопотребления при проектировании КМОП БИС: Технология и конструирование в электронной аппаратуре. 2008. № 2.
5. Шаляпин В. В. Основы микропроцессорной техники: учебное пособие, Шаляпин В. В., 2011. — 214 с.
6. Микропроцессорные системы: учеб. Пособие, Александров Е. К. и др.. Под общ. ред. Пузанкова Д.В. - СПб.: Политехника, 2002. - 935с.
7. Eieland A. Turn Power-Reducing Features into Low-Power Systems, [www.atmel.com](http://www.atmel.com/)
8. AVR188: Design Guidelines for ATtiny43U, [www.atmel.com](http://www.atmel.com/)
9. SAM L21\_Datasheet\_Preliminary, [www.atmel.com](http://www.atmel.com/)
10. Александр Рыжаков, Игорь Кривченко. Технология снижения энергопотребления picoPower: Компоненты и Технологии № 9 2015. –

11с.

1. Гусс С.В. «Микропроцессорные системы». ОМСК-2013
2. Микропроцессорные системы: Учебное пособие для вузов. Е. К.

Александров, Р. И. Грушвицкий, М. С. Куприянов, О. Е. Мартынов, Д. И. Панфилов, Т. В. Ремизевич, Ю. С. Татаринов, Е. П. Угрюмов, И. И. Шагурин; Под общ. ред. Д. В. Пузанкова. — СПб.: Политехника, 2002. — 935 c.

1. Завражина Т.Г. Датчики: Учеб пособие - Екатеринбург: Изд-во Рос. гос. проф. - пед. ун-та, 2002. - 87 с.

# ДОДАТОК А КОД ПРОГРАМИ

Лістинг А – Код скетчу Example\_1\_No\_Optimizations

#include <Adafruit\_CC3000.h> #include <SPI.h>

// CC3000 configuration.

#define ADAFRUIT\_CC3000\_IRQ 3

#define ADAFRUIT\_CC3000\_VBAT 5

#define ADAFRUIT\_CC3000\_CS 10

// Wifi network configuration.

#define WLAN\_SSID "network"

#define WLAN\_PASS "password"

#define WLAN\_SECURITY WLAN\_SEC\_WPA2

// Data logging configuration.

#define LOGGING\_FREQ\_SECONDS 60

#define SENSOR\_PIN 4

#define SERVER\_IP 192, 168, 1, 105

#define SERVER\_PORT 8000

Adafruit\_CC3000 cc3000 = Adafruit\_CC3000(ADAFRUIT\_CC3000\_CS, ADAFRUIT\_CC3000\_IRQ, ADAFRUIT\_CC3000\_VBAT);

uint32\_t ip;

unsigned long lastSend = 0;

void logSensorReading() {

int reading = analogRead(SENSOR\_PIN);

Serial.print(F("Sending measurement: ")); Serial.println(reading, DEC); Adafruit\_CC3000\_Client server = cc3000.connectTCP(ip, SERVER\_PORT); if (server.connected()) {

server.println(reading);

}

else {

Serial.println(F("Error sending measurement!"));

}

delay(100);

server.close();

}

void setup(void)

{

Serial.begin(115200); Serial.println(F("\nInitializing CC3000...")); if (!cc3000.begin())

{

Serial.println(F("Couldn't begin()! Check your wiring?")); while(1);

}

if (!cc3000.connectToAP(WLAN\_SSID, WLAN\_PASS, WLAN\_SECURITY)) {

Serial.println(F("Failed!")); while(1);

}

Serial.println(F("Connected!")); Serial.println(F("Request DHCP")); while (!cc3000.checkDHCP())

{

delay(100);

}

ip = cc3000.IP2U32(SERVER\_IP);

Serial.println(F("Setup complete."));

}

void loop(void)

{

unsigned long time = millis();

if (time - lastSend >= (1000 \* (unsigned long)LOGGING\_FREQ\_SECONDS)) { logSensorReading();

lastSend = time;

}

}

Лістинг Б – Код скетчу Example\_2\_Power\_Down\_Sleep

#include <Adafruit\_CC3000.h> #include <SPI.h>

#include <avr/sleep.h> #include <avr/power.h> #include <avr/wdt.h>

#define ADAFRUIT\_CC3000\_IRQ 3

#define ADAFRUIT\_CC3000\_VBAT 5

#define ADAFRUIT\_CC3000\_CS 10

#define WLAN\_SSID "network"

#define WLAN\_PASS "password"

#define WLAN\_SECURITY WLAN\_SEC\_WPA2

#define LOGGING\_FREQ\_SECONDS 60

#define SENSOR\_PIN 4

#define SERVER\_IP 192, 168, 1, 105

#define SERVER\_PORT 8000

#define MAX\_SLEEP\_ITERATIONS LOGGING\_FREQ\_SECONDS / 8

Adafruit\_CC3000 cc3000 = Adafruit\_CC3000(ADAFRUIT\_CC3000\_CS, ADAFRUIT\_CC3000\_IRQ, ADAFRUIT\_CC3000\_VBAT);

int sleepIterations = 0; uint32\_t ip;

volatile bool watchdogActivated = false;

ISR(WDT\_vect)

{

watchdogActivated = true;

}

void sleep()

{

set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN); power\_adc\_disable();

sleep\_mode(); sleep\_disable(); power\_all\_enable();

}

boolean enableWiFi() { Serial.println(F("Turning on CC3000.")); wlan\_start(0);

if (!cc3000.connectToAP(WLAN\_SSID, WLAN\_PASS, WLAN\_SECURITY)) {

Serial.println(F("Failed!")); return false;

}

Serial.println(F("Connected!")); Serial.println(F("Request DHCP")); int attempts = 0;

while (!cc3000.checkDHCP())

{

if (attempts > 5) { Serial.println(F("DHCP didn't finish!")); return false;

}

attempts += 1; delay(100);

}

return true;

}

void shutdownWiFi() {

if (cc3000.checkConnected()) { cc3000.disconnect();

}

while (cc3000.checkConnected()) { delay(100);

}

wlan\_stop();

Serial.println(F("CC3000 shut down."));

}

void logSensorReading() {

int reading = analogRead(SENSOR\_PIN);

Serial.print(F("Sending measurement: ")); Serial.println(reading, DEC); Adafruit\_CC3000\_Client server = cc3000.connectTCP(ip, SERVER\_PORT); if (server.connected()) {

server.println(reading);

}

else {

Serial.println(F("Error sending measurement!"));

}

delay(100); server.close();

}

void setup(void)

{

Serial.begin(115200); Serial.println(F("\nInitializing CC3000...")); if (!cc3000.begin())

{

Serial.println(F("Couldn't begin()! Check your wiring?")); while(1);

}

wlan\_stop();

ip = cc3000.IP2U32(SERVER\_IP);

noInterrupts();

MCUSR &= ~(1<<WDRF);

WDTCSR |= (1<<WDCE) | (1<<WDE); WDTCSR = (1<<WDP0) | (1<<WDP3); WDTCSR |= (1<<WDIE);

interrupts();

Serial.println(F("Setup complete."));

}

void loop(void)

{

if (watchdogActivated)

{

watchdogActivated = false; sleepIterations += 1;

if (sleepIterations >= MAX\_SLEEP\_ITERATIONS) { sleepIterations = 0;

if (enableWiFi()) { logSensorReading();

}

shutdownWiFi();

}

}

sleep();

}

Лістинг В – Код скетчу Example\_3\_Upgraded\_Hardware

#include <Adafruit\_CC3000.h> #include <SPI.h>

#include <avr/sleep.h> #include <avr/power.h> #include <avr/wdt.h> #include <avr/interrupt.h>

#define ADAFRUIT\_CC3000\_IRQ 3

#define ADAFRUIT\_CC3000\_VBAT 5

#define ADAFRUIT\_CC3000\_CS 10

#define WLAN\_SSID "network"

#define WLAN\_PASS "password"

#define WLAN\_SECURITY WLAN\_SEC\_WPA2 #define LOGGING\_FREQ\_SECONDS 60

#define SENSOR\_PIN 4

#define SERVER\_IP 192, 168, 1, 105

#define SERVER\_PORT 8000

#define MAX\_SLEEP\_ITERATIONS LOGGING\_FREQ\_SECONDS / 8

#define REG\_SHUTDOWN\_PIN 7

#define SENSOR\_POWER\_PIN 9

Adafruit\_CC3000 cc3000 = Adafruit\_CC3000(ADAFRUIT\_CC3000\_CS, ADAFRUIT\_CC3000\_IRQ, ADAFRUIT\_CC3000\_VBAT);

int sleepIterations = 0; uint32\_t ip;

volatile bool watchdogActivated = false;

ISR(WDT\_vect)

{

watchdogActivated = true;

}

void sleep()

{

set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN); power\_adc\_disable();

sleep\_enable();

MCUCR |= (1<<BODS) | (1<<BODSE); MCUCR &= ~(1<<BODSE);

sleep\_cpu(); sleep\_disable(); power\_all\_enable();

}

boolean enableWiFi() { Serial.println(F("Turning on CC3000.")); digitalWrite(REG\_SHUTDOWN\_PIN, HIGH); delay(100);

wlan\_start(0);

if (!cc3000.connectToAP(WLAN\_SSID, WLAN\_PASS, WLAN\_SECURITY)) {

Serial.println(F("Failed!")); return false;

}

Serial.println(F("Connected!")); Serial.println(F("Request DHCP")); int attempts = 0;

while (!cc3000.checkDHCP())

{

if (attempts > 5) {

Serial.println(F("DHCP didn't finish!")); return false;

}

attempts += 1; delay(100);

}

return true;

}

void shutdownWiFi() {

if (cc3000.checkConnected()) { cc3000.disconnect();

}

while (cc3000.checkConnected()) { delay(100);

}

wlan\_stop();

digitalWrite(REG\_SHUTDOWN\_PIN, LOW); delay(100);

Serial.println(F("CC3000 shut down."));

}

void logSensorReading() { digitalWrite(SENSOR\_POWER\_PIN, HIGH); int reading = analogRead(SENSOR\_PIN); digitalWrite(SENSOR\_POWER\_PIN, LOW);

Serial.print(F("Sending measurement: ")); Serial.println(reading, DEC); Adafruit\_CC3000\_Client server = cc3000.connectTCP(ip, SERVER\_PORT); if (server.connected()) {

server.println(reading);

}

else {

Serial.println(F("Error sending measurement!"));

}

delay(100); server.close();

}

void setup(void)

{

Serial.begin(115200); pinMode(REG\_SHUTDOWN\_PIN, OUTPUT); pinMode(SENSOR\_POWER\_PIN, OUTPUT);

digitalWrite(REG\_SHUTDOWN\_PIN, HIGH); delay(100); digitalWrite(SENSOR\_POWER\_PIN, LOW); Serial.println(F("\nInitializing CC3000..."));

if (!cc3000.begin())

{

Serial.println(F("Couldn't begin()! Check your wiring?")); while(1);

}

shutdownWiFi();

ip = cc3000.IP2U32(SERVER\_IP);

noInterrupts();

MCUSR &= ~(1<<WDRF);

WDTCSR |= (1<<WDCE) | (1<<WDE); WDTCSR = (1<<WDP0) | (1<<WDP3); WDTCSR |= (1<<WDIE);

interrupts();

Serial.println(F("Setup complete."));

}

void loop(void)

{

if (watchdogActivated)

{

watchdogActivated = false; sleepIterations += 1;

if (sleepIterations >= MAX\_SLEEP\_ITERATIONS) { sleepIterations = 0;

if (enableWiFi()) { logSensorReading();

}

shutdownWiFi();

}

}

sleep();

}