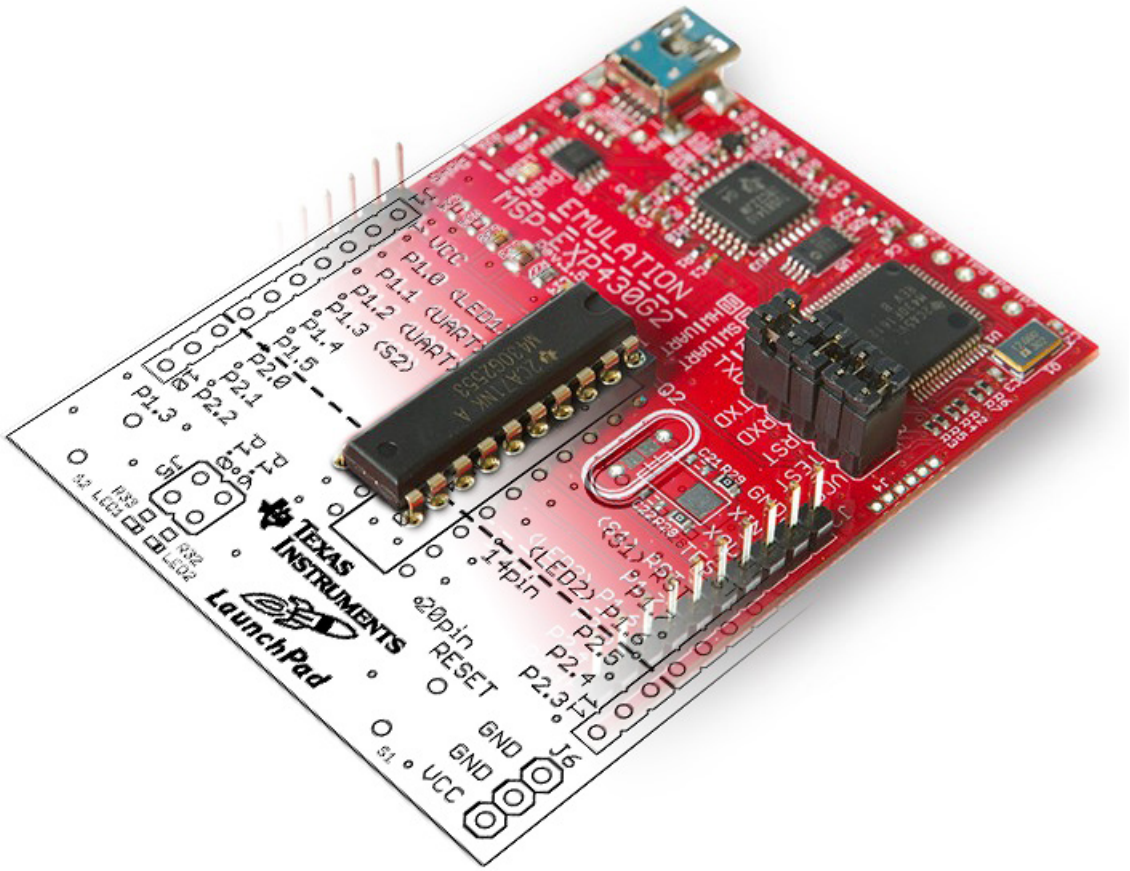
Проектування мікропроцесорних засобів вимірювання



Міністерство освіти і науки України
Вінницький національний технічний університет

Проектування мікропроцесорних засобів вимірювання

Лабораторний практикум

Вінниця
ВНТУ
2017

УДК 004.31:621.317(075)
П79

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 9 від 26.01.2017 р.)

Рецензенти:
В. М. Дубовой, доктор технічних наук, професор
Л. М. Заміховський, доктор технічних наук, професор
С. М. Цирульник, кандидат технічних наук, доцент

Проектування мікропроцесорних засобів вимірювання : Лабораторний практикум / В. Ю. Кучерук, О. М. Василевський, П. І. Кулаков, К. В. Овчинников − Вінниця : ВНТУ, 2017. – 77 с.
П79

Лабораторний практикум призначений для організації лабораторних робіт з дисципліни «Проектування мікропроцесорних засобів вимірювання» у середовищі CCS з використанням макетної плати LaunchPad для студентів спеціальності 6.051001 «Метрологія та інформаційно-вимірювальні технології». Для полегшення самостійного виконання завдань наводяться теоретичні відомості та список рекомендованої літератури.

УДК 004.31:621.317(075)

Зміст

Вступ

Мікропроцесорні прилади дозволяють вирішувати програмним методом частину завдань, які в звичайних приладах вирішуються апаратними засобами. Наприклад, для вимірювання амплітудного, середньовипрямленого і середньоквадратичного значень напруги апаратними методами потрібні відповідні перетворювачі [1, 2]. Це ж завдання можна вирішити мікропроцесорним приладом, перетворивши спочатку аналоговий вхідний сигнал в цифровий [3, 4] з допомогою аналого-цифрового перетворювача (АЦП), а потім, з використанням відповідних програм, обчислити необхідні параметри вимірюваного сигналу. Можливості приладу можна розширити, модифікувавши програмне забезпечення, наприклад, додаючи програмні модулі для статистичної обробки і спектрального аналізу. При цьому апаратна частина, що містить АЦП, не ускладнюється, а змінюється тільки програмне забезпечення. Тому мікропроцесорні прилади легше зробити багатофункціональними [5–10], що дозволяє скоротити парк засобів вимірювання, необхідних для наукових і виробничих цілей.

Проте використання мікропроцесорів має і недоліки, в першу чергу – складність апаратури [11–13] і досить висока її вартість. В перспективі, враховуючи швидке зниження цін на елементи мікропроцесорних систем, можна чекати значного здешевлення мікропроцесорних приладів. В деяких випадках швидкодія АЦП і мікропроцесора виявляються недостатніми для проведення вимірів або розрахунків в реальному масштабі часу. При цьому іноді виявляється доцільним застосувати масштабно-часове перетворення досліджуваного сигналу, зробивши його повільнішим. Тому постійно ведуться роботи з підвищення швидкодії і розрядності мікропроцесорів, що випускаються промисловістю, розширюючи тим самим можливості мікропроцесорних засобів вимірювання.

При розробці мікропроцесорних засобів вимірювання найбільш трудомісткою частиною виявляється розробка програмного забезпечення [14–17], вартість якого може значно перевищувати вартість апаратних засобів. Проте розробці апаратної частини і, зокрема, вибору мікропроцесора також приділяється багато уваги. А зі зростанням розрядності та обчислювальної потужності мікропроцесорів багато уваги приділяється методам зниження споживаної потужності мікропроцесорними засобами вимірювання.

Зростаючий попит на продукцію з автономним (батарейним) живленням викликав необхідність створення мікроконтролерів з низьким і наднизьким енергоспоживанням. Компанія Texas Instruments розробила і серійно випускає сім'ю мікроконтролерів MSP430 [18], яка задовольнить вимоги найвимогливішого розробника пристроїв на мікроконтролерах. Застосовуючи мікроконтролери MSP430, розробники отримують потужний інструмент для роботи з аналоговими і цифровими сигналами при наднизькому споживанні енергії батареї.

RISC (Reduced Instruction Set Computer) – архітектура процесора зі скороченим набором команд. Найбільш важливі відмітні особливості RISC–архітектури: архітектура регістр-регістр, прості способи адресації, прості команди і великий регістровий файл. Мікроконтролер MSP430 має 27 основних інструкцій і 24 додаткових інструкції, що значно спрощує процес генерації команд. Відсутні спеціальні команди звернення до акумулятора, пам'яті або до периферійних пристроїв, що істотно підвищує ефективність роботи процесора. Ядро процесора – 16-бітове RISC ALU і шістнадцять 16-бітових регістрів. Чотири регістри виконують функції програмного лічильника (PC), регістра статусу (SR), покажчика стека (SP) і регістра констант (CG). Інші дванадцять 16-бітових регістри – повністю у розпорядженні користувача. Регістри загального призначення використовуються для зберігання змінних, покажчиків і для операцій з даними. Процесор звертається до цих регістрів безпосередньо, що сприяє високій ефективності роботи мікроконтролера MSP430. Час виконання команд 1–4 машинні цикли (1–4 мкс).

Для ефективного використання енергії батареї сім’я мікроконтролерів MSP430 використовує п'ять режимів енергозбереження: LPM0, LPM1, LPM2, LPM3 і LMP4. Струм, що споживається мікроконтролером MSP430, в нормальному (робочому) режимі складає 250–400 мкА. Процесор (CPU) і всі вбудовані периферійні пристрої працюють в звичайному режимі. Основна особливість сім’ї мікроконтролерів MSP430 полягає в тому, що периферія (модуль АЦП, таймери, порти введення/виведення) може працювати автономно, тобто незалежно від процесора. Тому, якщо впродовж деякого проміжку часу CPU не використовується, його вимикають командою. Струм, що споживається від батареї, знижується до 30 мкА (режим LPM0). За відсутності необхідності використовувати системний тактовий сигнал (MCLK), для тактування CPU, АЦП і таймерів, в режимі LPM3 струм, що споживається від батареї, знижується до 0,8 мкА. Повернення з режимів енергозбереження LPM0–LPM3 в робочий режим відбувається за внутрішнім перериванням, яке генерують периферійні модулі. Повернення з режиму LPM4 (все вимкнено) можливе тільки за зовнішнім перериванням. Режими управління споживаною потужністю перемикаються програмно. Переходи з будь-якого режиму енергозбереження (LPM0–LPM4) в робочий режим відбуваються за 6 мкс. Розвинена система переривань (15 векторів) дозволяє оперативно керувати роботою мікроконтролера, мінімізуючи час «холостої» роботи CPU. Усі периферійні пристрої мають індивідуальні вектори переривання.

Ще одна особливість, властива сім’ї мікроконтролерів MSP430, це генератор з цифровим контролем (DCO). Для роботи мікроконтролера досить зовнішнього кварцевого резонатора з частотою резонансу 32 кГц (частота ACLK). Генератор DCO формує системну частоту MCLK, як добуток частоти ACLK на множник з ряду 1, 2, 4,..., 32. Значення множника задається програмно, при цьому частота MCLK набуває значень від 32 кГц

до 1 МГц. Системна частота MCLK потрібна для CPU, АЦП і таймерів. Програміст повинен сам вибрати значення MCLK, керуючись оптимальним співвідношенням продуктивності мікроконтролера і енергозбереження.

Приємне доповнення до програмного забезпечення пакет FPP (Floating Point Package). Математика з плаваючою крапкою потрібна, якщо розмір вживаних чисел дуже великий. Пакет з плаваючою крапкою оперує з 24-бітовою і 40-бітовою мантисою і розроблений для мікроконтролерів сім'ї MSP430. У пакет входять конвертери, що переводять числа з двійкового та двійково-десяткового формату в числа формату «плаваюча крапка», також передбачено і зворотне перетворення. Пакет з плаваючою крапкою використовує особливості RISC архітектури сім'ї мікроконтролерів MSP430. Під час роботи підпрограми, що використовує пакет FPP, аргументи копіюються в регістри R4–R15 і всі обчислення відбуваються в регістровій пам'яті. Після завершення обчислень результат зберігається у вершині стека.

До того ж компанія Texas Instrument супроводжує всі мікроконтролери сім'ї MSP430 налагоджувальними пристроями. Ці пристрої значно прискорюють процес адаптації розробника до нової для нього сім'ї мікроконтролерів.

Лабораторна робота № 1
ЗАВАНТАЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА НАЛАШТУВАННЯ ОБЛАДНАННЯ

Мета роботи: завантаження та встановлення програмного забезпечення Code Composer Studio, завантаження документації і програмного забезпечення для роботи з MSP430 LaunchPad.

1.1 Теоретичні відомості

LaunchPad є простим у використанні засобом розробки, призначеним для початківців і досвідчених користувачів, які розробляють пристрої на основі мікроконтролерів. LaunchPad забезпечує корстувача всіма необхідними інструментами для початку розробки власних проектів.

Комплект розробника LaunchPad є частиною серії MSP430 Value Line та забезпечений вбудованим DIP (Dual In-line Package) роз'ємом з підтримкою до 20 виводів, що дозволяє встановлювати пристрої серії MSP430 Value Line на макетній платі LaunchPad. Крім того, вбудований засіб емуляції флеш-пам'яті дозволяє, безпосередньо підключаючись до персонального комп'ютера, легко програмувати, відлагоджувати і аналізувати код. У комплект постачання входять безкоштовні середовища розробки програмного забезпечення, призначені для написання і відлагоджування програм. Завдяки клавіатурі, світлодіодам і додатковим виводам входів/виходів, призначеним для простої інтеграції із зовнішніми пристроями, LaunchPad може бути використаний для створення інтерактивних рішень.

Макетна плата TI LaunchPad MSP430 поставляється в наборі з двома мікроконтролерами серії MSP430G2x. Чипи відрізняються один від одного кількістю пам'яті, числом виводів і набором послідовних портів. Основні характеристики чипів наведені в таблиці 1.1.

Таблиця 1.1 – Основні характеристики комплектів поставки

	MSP430G2553	MSP430G2453
Тактова частота	16 МГц	16 МГц
FLASH ROM	16 Кб	8 Кб
SRAM	512 байтів	256 байтів
Цифрові входи/виходи	24	16
Таймери	2	1
Послідовний порт	USCI – I^2C, SPI, UART	USCI – I^2C, SPI
АЦП	8 × 10 бітів	8 × 10 бітів

Окрім перерахованого вище, контролер має у своєму складі датчик температури, вартовий таймер і компаратор, а вбудовані 16-бітові таймери мають підтримку широтно-імпульсної модуляції (ШІМ). На платі також

розміщуються два світлодіоди, під’єднані до виводів P1.0 і P1.6, кнопка скидання і кнопка в розрив виведення P1.3. Загальний вигляд макетної плати LaunchPad зображений на рис. 1.1.

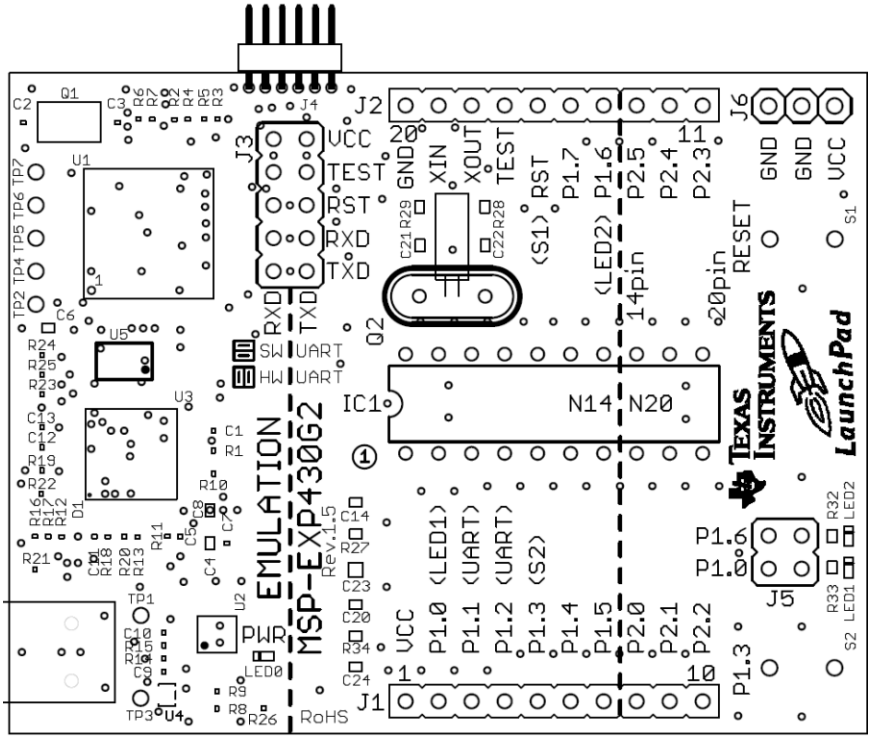


Рисунок 1.1 – Загальний вигляд макетної плати LaunchPad

Серед іншого до складу макетної плати LaunchPad входять:

U1 – MSP430F1612IPMR (16-розрядний мікроконтролер з надмалим споживанням потужності);

U2 – TPS77301DGKR (лінійний стабілізатор з малим спадом напруги вхід/вихід);

U3 – TUSB3410VF (повношвидкісний міст USB – UART);

U4 – TPD2E001DRLR (однокристальна сукупність швидкісних діодів для захисту комунікаційних ліній від розрядів статичної електрики);

U5 – AT24C128-10TU-2.7 (EEPROM з послідовним інтерфейсом I^2C);

LED0, LED1 – зелені світлодіоди;

LED2 – червоний світлодіод;

S1, S2 – тактові кнопки.

1.2 Порядок виконання роботи

Для роботи з макетною платою необхідно завантажити та встановит на комп’ютер відповідне програмне забезпечення.

1.2.1 Завантаження та встановлення Code Composer Studio 4

Натисніть на посилання або скопіюйте URL в рядок адреси браузера: http://processors.wiki.ti.com/index.php/Download_CCS. Знайдіть та виберіть посилання, «скачати останню версію MSP430/C28x з обмеженням коду». Для отримання можливості скачувати файли необхідно увійти в систему (зверніть увагу, для того щоб продовжити свою роботу, ви повинні мати обліковий запис TI). Як тільки ви погоджуєтеся з умовами експорту, вам буде надіслано по електронній пошті посилання на ZIP-файл. Натисніть на посилання і збережіть ZIP-файл на робочому столі. Розпакуйте файл в папку з ім'ям CCS Setup та після завершення встановлення видаліть поштовий файл і папку встановлення CCS Setup. Обов'язково відключіть будь-які плати, що були підключені до USB-портів ПК.

Відкрийте папку встановлення CCS Setup на робочому столі і двічі натисніть на файлі з ім'ям setup_CCS_MC_Core_n.n.n.n.exe. Додержуйтеся інструкцій для встановлення програми у Code Composer Studio. Виберіть «MSP430-only Core Tools» для встановлення в діалоговому вікні «Product Configuration». Натисніть кнопку Next. Вигляд діалогового вікна наведений на рисунку 1.2.

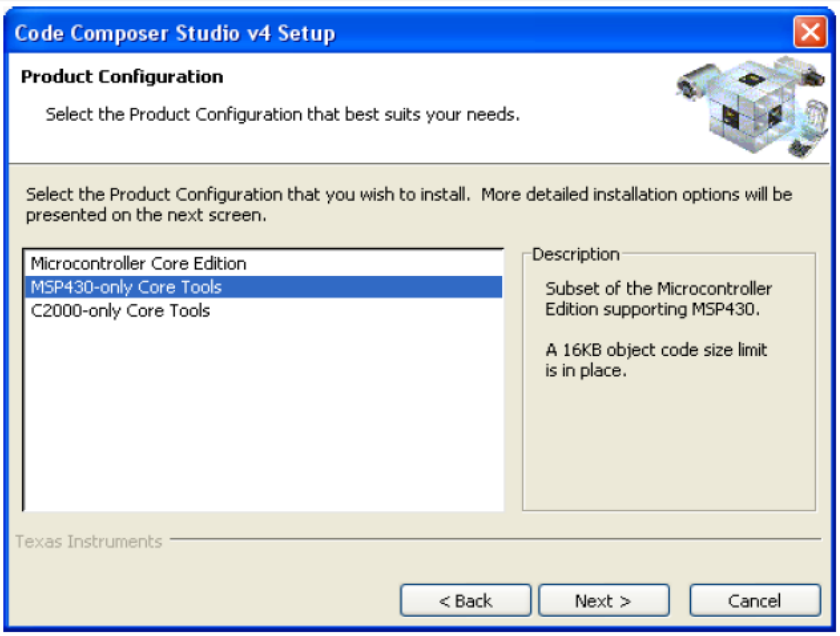


Рисунок 1.2 – Діалогове вікно конфігурації продукту

В діалоговому вікні використовуйте налаштування за замовчуванням. Виберіть компоненти і натисніть кнопку Next. Натисніть кнопку Next у стартовому вікні копіювання файлів. Встановлення займе не більше 10 хвилин. Процес вибору компонентів наведено на рисунку 1.3.

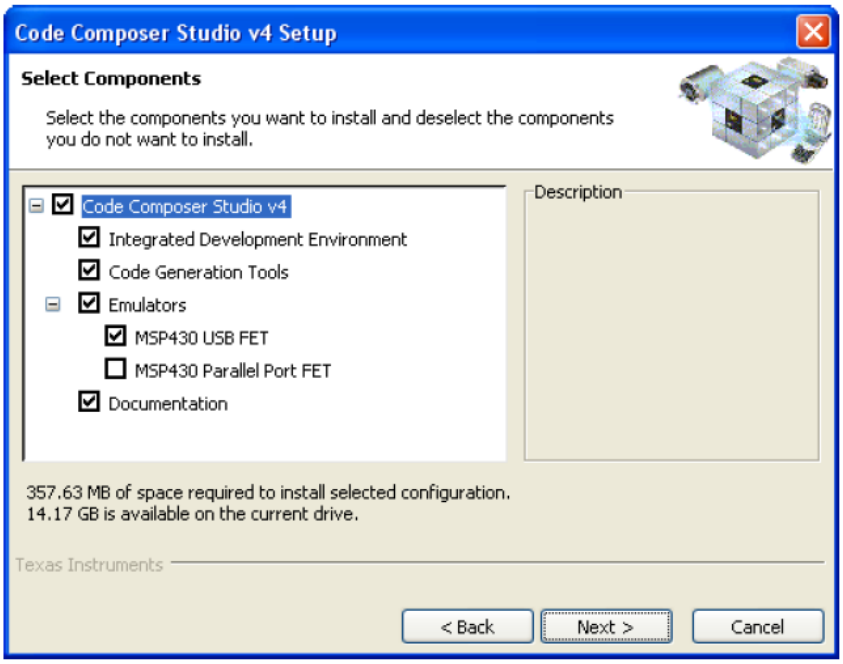


Рисунок 1.3 – Діалогове вікно вибору компонентів

1.2.2 Завантаження та встановлення Workshop Lab

Натисніть на посилання, щоб перейти на MSP430 LaunchPad та скачати файл і зберегти MSP430_LaunchPad_Workshop.exe файл на робочому столі:http://softwaredl.ti.com/trainingTTO/trainingTTO_public_sw/MSP430_LaunchPad_Workshop/MSP430_LaunchPad_Workshop.exe

Двічі натисніть MSP430_LaunchPad_Workshop.exe файл, щоб встановити програмне забезпечення на комп'ютер. Після встановлення ви можете видалити даний файл на робочому столі. Відповідні файли будуть встановлені в C:\MSP430_LaunchPad, а структура каталогів виглядатиме, як зображено на рис. 1.4.

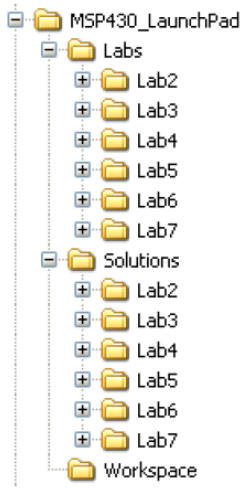


Рисунок 1.4 – Структура каталогів

1.2.3 Завантаження підтверджувальних документів та програмного забезпечення

Завантажте і збережіть нижчевказані документи і програми на вашому комп'ютері:

- LaunchPad «Керівництво користувача»: http://focus.ti.com/lit/ug/slau318/slau318.pdf
- MSP430x2xx «Керівництво користувача»: http://focus.ti.com/lit/ug/slau144f/slau144f.pdf
- C Compiler «Керівництво користувача»: http://www.ti.com/litv/pdf/slau132e
- MSP430G2xx «Приклади коду»: http://www.ti.com/lit/zip/slac463a
- Температурні демонстраційні файли: http://www.ti.com/litv/zip/slac435
- Температурні демонстраційні GUI-файли: http://dl.dropbox.com/u/9458154/LaunchPad_Temp_GUI.zip

Додаткова інформація доступна на MSP430 LaunchPad на http://www.ti.com/launchpadwiki

Перевірте наявність всіх засобів, що повинні входити у комплект лабораторного стенда MSP430 LaunchPad:

- емулятор плати LaunchPad (MSO-EXP430G2);
- міні USB-B кабель;
- MSP430G2231 (попередньо встановлені з передвстановленою демонстраційною програмою) та MSP430G2211;
- 10-контактні PCB-роз'єми (два штекерних і два розеткових);
- 32,768 кГц кварцовий резонатор;
- короткий посібник користувача і дві LaunchPad-наклейки.

1.2.4 Налаштувати обладнання

Плата лабораторного стенда LaunchPad містить в собі запрограмований мікроконтролер MSP430G2231, встановлений у роз'єм на платі. Коли LaunchPad підключений до комп'ютера через універсальну послідовну шину (Universal Serial Bus, USB), демострація починається з послідовним перемиканням світлодіодів. На платі емулятор генерує напругу живлення і всі сигнали, необхідні для запуску демонстраційної версії. Підключіть MSP430 LaunchPad до комп'ютера за допомогою USB-кабелю. Встановлення драйвера запускається автоматично. При появі запиту на програмне забезпечення воно буде встановлено автоматично. Наразі світлодіоди на платі червоного і зеленого кольору повинні послідовно перемикатися. Це свідчить про те, що обладнання працює і було налаштоване правильно.

1.2.5 Запуск демо-програми

Демо-програма, записана в пам'яті, здійснює вимірювання температури за допомогою внутрішнього датчика температури. Цей демонстраційний приклад задіює різні периферійні пристрої мікросхеми MSP430G2231. До задіяних у програмі периферійних пристроїв відносять: 10-розрядний аналого-цифровий перетворювач (Analog to Digital Converter, ADC), що перетворює сигнал внутрішнього температурного датчика, і 16-розрядні таймери, які керують широтно-імпульсною модуляцією (Pulse-Width Modulation, PWM) для варіювання яскравості світлодіодів. Додатково, 16-розрядні таймери використовуються в програмному забезпеченні універсального асинхронного приймача/передавача (Universal Asynchronous Receiver-Transmitter, UART) для обміну даними з персональним комп'ютером.

Натисніть кнопку P1.3 (нижня ліва), щоб переключити програму в режим вимірювання температури. Значення температури визначається на початку цього режиму, а світлодіоди сигналізують про підвищення або зниження температури LaunchPad, змінюючи яскравість червоного або зеленого світлодіодів відповідно. Ви можете повторно калібрувати значення температури, ще раз натиснувши на кнопку P1.3. Спробуйте збільшити або зменшити температуру кристала (потріть палець об палець, щоб нагріти його або помістіть кінчик пальця у холодну воду, щоб навпаки – знизити температуру, потім натисніть у верхній частині контролера). Зверніть увагу на зміну яскравості світлодіода.

Далі ми будемо використовувати графічний інтерфейс для відображення показань температури на ПК. Переконайтеся, що встановлено джерело GUI та завантажені файли (LaunchPad_Temp_GUI.zip)

Перевірте, що COM-порт був використаний для плати шляхом натискання (в Windows) **Win+R**, потім введіть devmgmt.msc в поле і виберіть OK. (У Windows 7 просто введіть в поле пошуку програм і файлів). У вікні Device Manager, що відкривається, натисніть лівою кнопкою на символ (COM & LPT) і запишіть число COM-порту для програм MSP430 UART (COMxx): _____. Закрийте Device Manager.

Запустити GUI, натиснувши на LaunchPad_Temp_GUI.exe. Цей файл знаходиться в `<Install Directory>\LaunchPad_Temp_GUI\application.window`. Можливо, вам доведеться вибрати Run у вікні «Open File – Security Warning».

Для запуску GUI необхідний деякий час. Переконайтесь, що пристрій працює (тобто кнопка P1.3 була натиснута). У GUI виберіть COM-порт і натисніть Enter. Змінний струм повинен бути виведений на екран. Спробуйте збільшити і зменшити температуру на пристрої і подивіться на зміни, що відбуваються на дисплеї. Зверніть увагу на те, що внутрішній температурний датчик не калібрований. Тому виведене на екран значення не

буде точним. Ми просто шукаємо температурні значення, щоб відслідкувати їх зміну.

1.3 Зміст звіту

1. Мета роботи.
2. Налаштоване середовище та обладнання готове до роботи.
3. Висновки.

Контрольні питання

1. З яких основних частин складається MSP430 LaunchPad?
2. Перерахуйте основні характеристики чипів серії MSP430G2x, що ідуть в комплекті з макетною платою.
3. На рисунку 1.1 назвіть модулі, що входять до складу макетної плати.
4. З яких компонентів складається Code Composer Studio?
5. Наведіть структуру каталогів Code Composer Studio.
6. Перерахуйте засоби, що входять у комплект лабораторного стенда.
7. Як запустити на виконання демонстраційну програму, записану в постійну пам’ять мікроконтролера?
8. Як переключити демонстраційну програму в режим вимірювання температури?
9. Як скористатися можливостями графічного інтерфейсу для відображення значення температури?
10. Для чого призначене і що собою являє Code Composer Studio?
11. Які функції виконує Workshop Lab?

Лабораторна робота № 2
ОСНОВНІ ОСОБЛИВОСТІ CODE COMPOSER STUDIO

Мета роботи: вивчення основних особливостей Code Composer Studio. У цій лабораторній роботі необхідно створити новий проект, побудувати код програми за допомогою ПК на MSP430.

2.1 Теоретичні відомості

Code Composer Studio (CCS) – Integrated Development Environment (IDE), інтегроване середовище розробки програмного забезпечення та створення коду для цифрових сигнальних процесорів (Digital Signal Processor – DSP) і/або ARM-процесорів сім'ї TMS320 та інших процесорів Texas Instruments (TI), таких як MSP430.

Інтегральне середовище Code Composer Studio містить операційну систему реального часу DSP/BIOS. Також до складу продукту входять емулятори та підтримка JTAG-орієнтованого відлагодження.

Середовище CCS містить такі інструменти, як редактор вихідного тексту, компілятор, компанувальник (link editor, linker), налагоджувач (debugger), симулятори для всіх типів процесорів TI, засоби візуалізації та багато інших допоміжних інструментів. Це єдиний графічний інтерфейс користувача, який дозволяє проводити покрокову розробку та відлагодження програмного коду. Інтуїтивно зрозумілі інструменти дозволяють розробнику швидко почати свою власну розробку, поступово освоюючи нові функціональні можливості та покращуючи продуктивність власних рішень.

Засіб розробки CCS базується на програмному забезпеченні з відкритим вихідним кодом Eclipse. Використання саме цього продукту обумовлене тим, що Eclipse забезпечує ефективну структуру програмного забезпечення, яку підтримує велика кількість розробників вбудованих рішень.

IDE Code Composer Studio була створена в рамках стандарту eXpressDSP, який розробляється фірмою Texas Instrument і представляє з себе універсальну технологію розробки програмного забезпечення для процесорів TI. Такій IDE притаманний ряд властивостей, найбільш суттєвими з яких є:

- інтеграція усіх засобів розробки (редактор, відлагоджувач, менеджер проектів тощо) в одно застосування, що має зручний інтерфейс;
- потужний промисловий Сі-компілятор, асемблерний оптимізатор, компанувальник;
- масштабоване ядро (DSP/BIOS II) реального часу. Містить набір стандартних інструментів взаємодії DSP-системи з периферією;
- програмний симулятор, що дозволяє вести відлагодження на програмній моделі вибраної сім'ї ЦСП;

- аналізатор реального часу, призначений для моніторингу стану системи без зупинки процесора;
- компілятор, що оптимізує код програми під певну платформу (розмір коду, оптимальність за рахунок використання конвеєризації і паралельних блоків);
- візуальний компанувальник, що дозволяє за допомогою графічного інтерфейсу розташовувати код програми і дані в пам'яті;
- можливість перегляду виду сигналів в різних графічних форматах;
- відкрита вбудовувана архітектура, що дозволяє інтегрувати в систему засоби сторонніх розробників. Основні функціональні можливості IDE CCS наведені на рис. 2.1.

Після запуску CCS відкривається основне вікно, яке поділено на дві ділянки – робочу та ділянку навігації проекту. В ділянці навігації проекту відображається вся інформація про поточний проект – вихідні файли, командний файл компанувальника, підключені бібліотеки і т. ін.

Основною програмною одиницею в CCS є проект (файл з розширенням *.pjt). Проект містить в собі вихідні файли (тексти програм мовою Сі (*.c) або асемблера (*.asm), бібліотеки (*.lib), командний файл компанувальника (*.cmd)) та файл налаштувань (установки компілятора та компонувальника). На рис. 2.2 зображено частину основного вікна проекту в CCS та наведений перелік основних складових проекту.

Рисунок 2.1 – Функціональні можливості Code Composer Studio 4.1

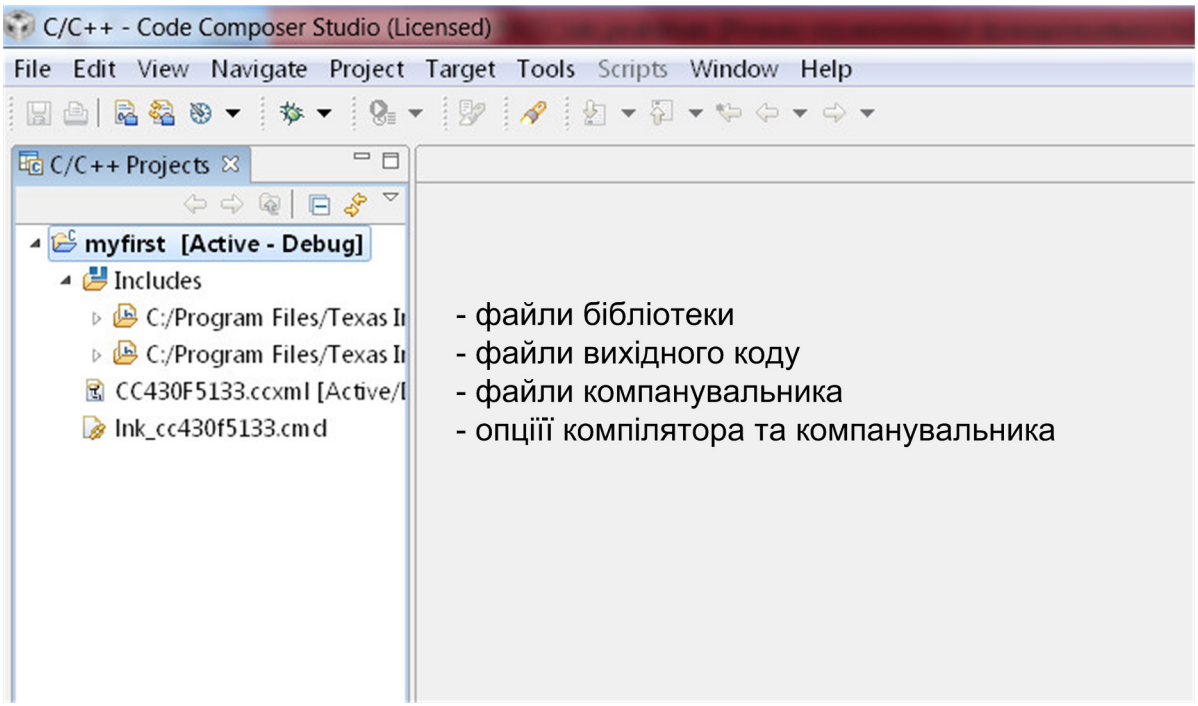


Рисунок 2.2 – Складові проекту

Основні етапи створення нового проекту в інтегрованому середовищі розробки Code Composer Studio наведені на рис. 2.3.

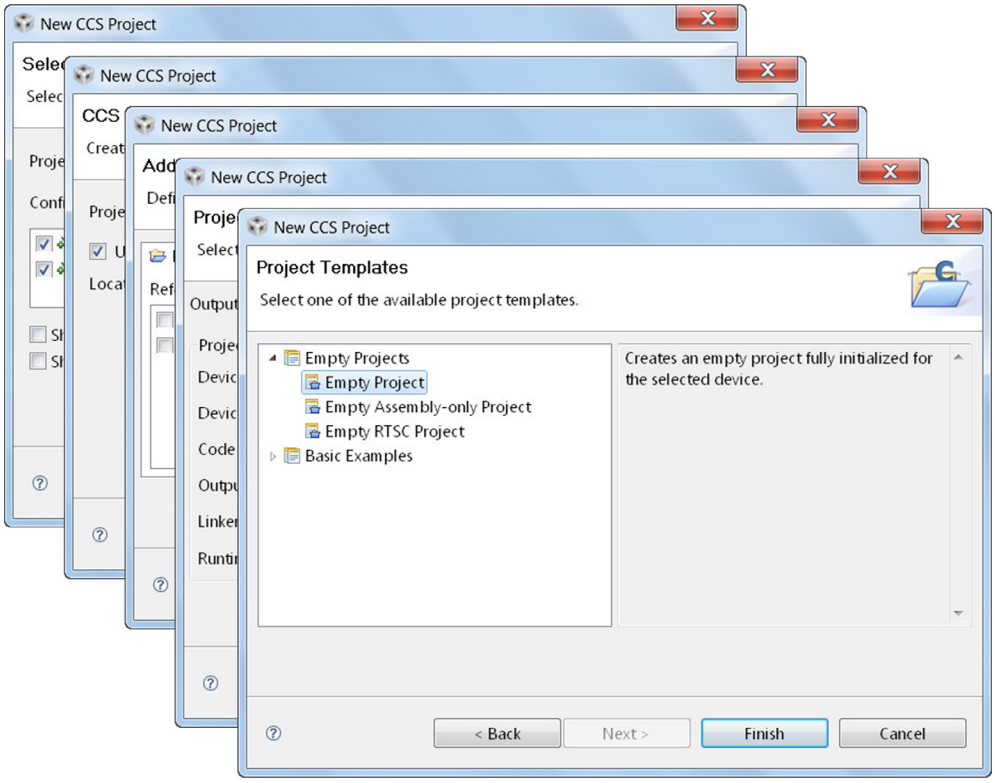


Рисунок 2.3 – Основні етапи створення нового проекту в CCS

2.2 Порядок виконання роботи

Запустіть Code Composer Studio та відкрийте робочий файл

1. Запустіть Code Composer Studio (CCS) шляхом подвійного натискання на значку на робочому столі або вибору його з переліку програм Windows. Перейдіть за такою адресою:

`C:\MSP430_LaunchPad\WorkSpace` та натисніть «OK».

Ця папка містить всі налаштування CCS, які охоплюють налаштування і проекти, що були створені до того, як CCS закрили, а також вони будуть доступними, коли CCS буде відкритий знову. Коли CCS закритий, робочий простір збережено автоматично.

2. Спершу, коли CCS відкривається, з'являється сторінка «Welcome to Code Composer Studio v4». Закрийте сторінку шляхом клацання на значку CCS або шляхом клацання на вкладці «Welcome».

Тепер можна бачити інструментальні засоби CCS. Інструментальні засоби відкриються в «C/C++ Perspective». «Perspective» визначає початкові уявлення про розташування вікон інструментальних засобів, панелей інструментів і меню, які є придатними для певного типу завдання (тобто кодують розробку або налаштовують її). «C/C++ Perspective» використовується, щоб створити або розробити проекти на C/C++. Коли сеанс налагодження буде запущений, «Debug Perspective» буде автоматично включено. Дана операція використовується для того, щоб налаштувати проекти на C/C++.

Створіть новий проект

1. Проект містить всі файли, що будуть необхідними для розробки вихідного файлу (.out), який може бути виконаний на апаратних засобах MSP430. Створення нового проекту можна здійснити за допомогою таких операцій:

`File->New->CCS Project`

Введіть назву проекту: Temperature_Sense_Demo. Відміттє опцію «Use default location» («Використовувати за замовчуванням»). Натисніть кнопку Browse (огляд) і перейдіть до потрібного вам проекту:

`C:\MSP430_LaunchPad\Labs\Lab2\Project-TS`

Після цього необхідно натиснути кнопку «OK», а потім, відповідно, – Next.

2. Коли з'явиться наступне вікно, необхідно обрати відповідну платформу та конфігурацію. «Project type» («Тип проекту») повинен бути встановлений в «MSP430». Натисніть кнопку Next.

3. У наступному вікні між залежностями проекту (якщо такі є) визначити «There are none now» (немає жодних). Натисніть кнопку Next.

4. В останньому вікні визнчіть налаштування проекту CCS. Виберіть «Device variant» за допомогою списку, що з'явиться на екрані, та вкажіть «MSP430G2231». Це дозволить використовувати відповідний файл команд компонувальника, а також бібліотеку підтримки, яка використовується під

час виконання проекту. Встановіть основні опції для компанувальника та компілятора, встановіть цільову конфігурацію. Після цього натисніть кнопку Finish.

5. Після проведення послідовності описаних дій буде створено новий проект. Зверніть увагу, що вікно C/C++ Pprojects містить файли шаблону проекту Temperature_Sense_Demo. Встановлений проект знаходиться в стані Active (активний), а вихідні файли розташуються в папці Debug. У цій конфігурації проект не містить вихідних файлів. Наступний крок повинен додати початкові файли до проекту.

Створіть файл вихідного коду

1. Для додавання вихідного файлу до проекту клацніть правою кнопкою по Temperature_Sense_Demo у вікні C/C++ Projects і виберіть:

`New->Source File` або `File->New->Source File.`

Дайте ім'я вихідному файлу main.c і натисніть Finish. Після чого відкриється порожнє вікно для коду заданого вихідного файлу main.c.

2. Після створення порожнього файлу вихідного коду необхідно додати відповідний код до вихідного файлу main.c. На даному етапі ми будемо використовувати код першоджерела, який був попередньо запрограмований в пристрій MSP430G2231 (тобто програма, яку була створено раніше).

Натисніть `File->Open File...` і перейдіть до

`C:\MSP430_LaunchPad\Labs\Lab2\Files.`

Відкрийте файл `Temperature_Sense_Demo.txt` для редагування. Скопіюйте вміст цього файла в буфер обміну і вставте в main.c. Потім закрийте файл `Temperature_Sense_Demo.txt`. Цей файл більше не потрібен. Обов'язково збережіть main.c, натиснувши кнопку Save (зберегти) в основному меню вікна проекту.

Створіть і завантажте проект

1. На панелі інструментів розташовуються кнопки, про функції яких можна дізнатися таким чином: коли навести курсор на клавішу, яка вам необхідна, тоді у вікні під нею можна прочитати пояснення. Клавіші мають вигляд, показаний на рис. 2.4. Пояснення функціонального призначення кнопок зведено до таблиці 2.1.

Рисунок 2.4 – Кнопки на панелі інструментів вікна проекту

2. Натисніть кнопку «Build», за допомогою якої буде показано різні інструменти, розміщені у вікні Console. Перевірте, чи не виникло помилок в процесі створення бінарного файлу. Якщо помилки виникатимуть, з'являтиметься повідомлення про помилку у вікні Problems. Шляхом подвійного клацання на повідомленні про помилку редактор автоматично ві-

дкриє вихідний файл і розташує курсор миші в рядок коду, який містить помилку. Зауважте, що одна помилка в коді може іноді генерувати декілька повідомлень у вікні Problems під час створення проекту.

Таблиця 2.1 – Функціональне призначення кнопок панелі інструментів

Вигляд	Назва	Пояснення
	Build	Створення лише змінених вихідних файлів
	Rebuild	Повне створення всіх вихідних файлів
	Debug	Автоматичне посилання, завантаження і налагоджування

3. CCS може автоматично зберегти змінені вихідні файли, створити програму, завантажити її у флеш-пам'ять, а потім виконати програму до початку основної функції.

Натисніть кнопку «Debug», далі виберіть `Target->Debug Active Project`.

Переконайтесь, що значок Debug (налагодження) розташовується у верхньому правому кутку. Це вказує на те, що проект знаходиться на стадії її налагодження в «Debug perspective». Програма пройшла процес ініціалізації середовища в бібліотеці підтримки і готова до виконання функції `main()` з файлу main.c.

Середовище налагодження

1. Основні кнопки, які керують середовищем налагодження зображені на рис. 2.5. Знаходиться вони у верхній частині IDE CCS.

Рисунок 2.5 – Основні кнопки керування процесом налогодження

Опис функціонального призначення кнопок, які відповідають за ініціалізацію процеса налагоджування, зведено до таблиці 2.2. Опис функціонального призначення кнопок, які відповідають за процес налагодження зведено до таблиці 2.3.

Таблиця 2.2 – Кнопки ініціалізації процесу налагодження

Вигляд	Назва	Пояснення	Наявність
	New Target Configuration (нова цільова конфігурація)	Створює новий цільовий файл конфігурації.	File New Menu Target Menu
	Debug Active Project (Налагодження активного проекту)	Запускає сеанс налагодження на основі активного проекту.	Debug Toolbar Target Menu
	Launch TI Debugger (Запуск TI Debugger)	Запускає відладчик з цільовою конфігурацією за замовчуванням.	Debug Toolbar Target Menu
	Debug (Налагодження)	Відкриває діалогове вікно для зміни існуючих конфігурацій налагодження.	Debug Toolbar Target Menu
	Connect Target (Підключення об’єкту)	З'єднання з апаратними цілями.	TI Debug Toolbar Target Debug View Context Menu
	Terminate All (Завершити все)	Завершує всі активні сеанси налагодження.	Target Menu Debug View Toolbar

Таблиця 2.3 – Кнопки керування процесом налагодження

Вигляд	Назва	Пояснення	Наявність
1	2	3	4
	Halt (Зупинити)	Зупинка.	Target Menu Debug View Toolbar
	Run (Працювати)	Відновлює виконання в даний час завантаженої програми.	Target Menu Debug View Toolbar
	Run to Line (Йти до лінії)	Відновлює виконання в даний час завантаженої програми від поточного розташування.	Target Menu Disassembly Context Menu Source Edi-tor Context Menu
	Go to Main (На головну)	Виконує програму до початку того, коли основна функція до-сягнута.	Debug View Toolbar
	Step Into (Крок в…)	Крок у виділений оператор.	Target Menu Debug View Toolbar
	Step Over (Крок із…)	Пропускає через виділений опе-ратор.	Target Menu Debug View Toolbar

Продовження таблиці 2.3

1	2	3	4
	Step Return (Крок назад)	Кроки з поточної позиції.	Target Menu Debug View Toolbar
	Reset (Скидання)	Скидає вибрану задачу.	Target Menu Debug View Toolbar
	Restart (Перезапуск)	Перезавантажує ПК до точки входу для завантаженої програми.	Target Menu Debug View Toolbar
	Assembly Step Into (Сукупність кроків в…)	Відладчик виконує наступну інструкцію блоку.	TI Explicit Stepping Toolbar Target Advanced Menu
	Assembly Step Over (Сукупність кроків з…)	Відладчик пропускає через єдину інструкцію блоку.	TI Explicit Stepping Toolbar Target Advanced Menu

2. Отже процесор готовий до виконання основної функції `main()` з файлу main.c. Натисніть кнопку Run для виконання коду.

Зауважте, що червоний і зелений світлодіоди перемикаються як очікувалося.

3. Натисніть Halt. Після цього процесор повинен зупинити виконання функції `main()`.

4. Далі натисніть один раз кнопку Step Into, що запустить таймер Interrupt Source Routine (ISR) для перемикання світлодіодів. Натисніть

кнопку Step Into знову та переконайтесь в тому, що червоний і зелений світлодіоди почергово вмикаються і вимикаються.

5. Натисніть Reset CPU, а потім поверніться на початок `main()`.

6. Кнопка Terminate All завершить активний сеанс налагодження, закриє налагоджувач і поверне IDE CCS до вигляду по замовчанню. Натисніть `Target->Terminate All`, або використайте кнопку Terminate All з переліку кнопок управління процесом ініціалізації налагоджування (табл. 2.2). Закрийте сеанс налагодження, натиснувши хрестик на вкладці.

7. Клацніть правою кнопкою по «Temperature _Sense_Demo» у вікні «C/C++ Projects» і виберіть «Close Project» для того, щоб закрити проект.

2.3 Зміст звіту

1. Мета роботи.
2. Створений проект готовий для виконання та відлагодження.
3. Висновки.

Контрольні питання

1. Які інструменти містить в собі IDE Code Composer Studio?
2. Перерахуйте властивості, які притаманні інтегрованому середовищу розробки.
3. Що є основною програмною одиницею Code Composer Studio?
4. Які вихідні файли включає в себе проект?
5. Перерахуйте основні етапи створення нового проекту Code Composer Studio.

Лабораторна робота № 3
ІНІЦІАЛІЗАЦІЯ ТА ПОРТИ ВВЕДЕННЯ/ВИВЕДЕННЯ

Мета роботи: навчитися створювати код ініціалізації і запуску пристрою (MSP430) за допомогою різних ресурсів таймера.

3.1 Теоретичні відомості

Перед тим, як процесор почне виконувати основну функцію `main()`, після процесу апаратного скидання, буде виконаний низькорівневий код ініціалізації периферійних пристроїв. Стан системи ппісля скидання зображений на рис 3.1.

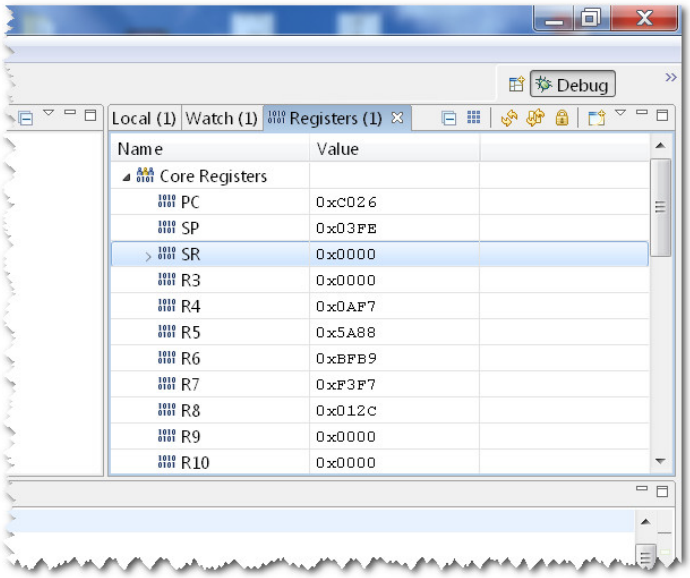


Рисунок 3.1 – Стан системи після скидання

В схемі системного скидання джерелами скидання можуть бути: сигнал скидання при включенні (Power On Reset (POR)) та сигнал очищення при включенні (Power Up Clear (PUC)). Різні події та вихідні умови визначають, який саме з цих сигналів буде сгенеровано.

Сигнал POR скидає пристрій. Він може бути сгенерований в двох наступних випадках:

- ввімкнення пристрою;
- поява сигналу низького рівня на виводі RST/NMI, у випадку, коли вивід сконфігурований як вхід сигналу скидання.

Сигнал PUC генерується завжди при появі сигналу POR, але сигнал POR не генерується сигналом PUC. До появи сигналу PUC призводять наступні події:

- сигнал POR;

- спрацювання «вартового» таймера (за умови, якщо «вартовий» таймер активовано);
- відбулося порушення ключа безпеки «вартового таймера»;
- відбулося порушення ключа безпеки flash-пам'яті.

За будь якого джерела скидання необхідно виконати перезавантаження системи програмного забезпечення. Основні етапи перезавантаження системи програмного забезпечення наведені на рис. 3.2.

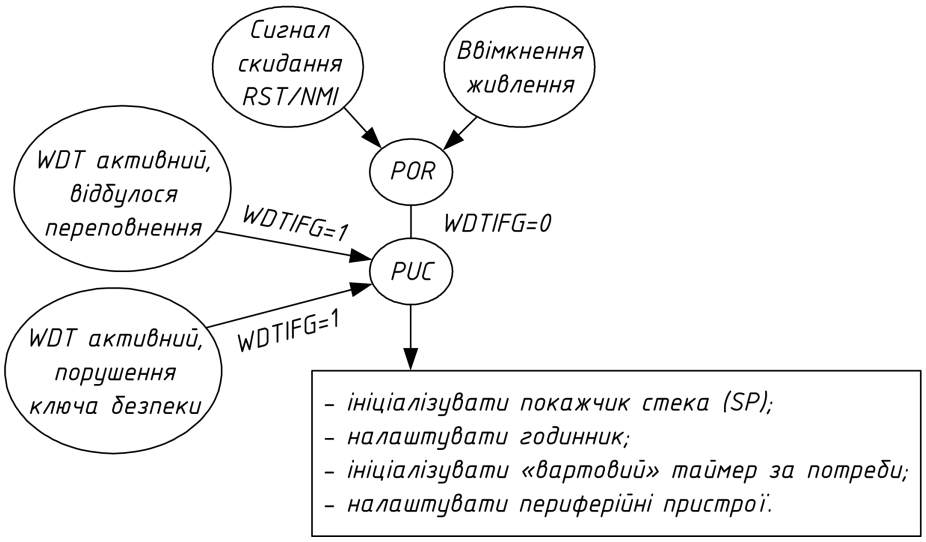


Рисунок 3.2 – Ініціалізація програмного забезпечення

Основний модуль тактування має низьку вартість та наднизький рівень енергоспоживання. Використовуючи три внутрішніх тактових сигнали, користувач може вибрати найкращий баланс співвідношення продуктивності та споживаної потужності. Основний модуль тактування може бути програмно сконфігурований на роботу без використання зовнішніх компонент, з одним зовнішнім резистором, з одним або двома зовнішніми кристалами або резонаторами.

Основний модуль тактування включає в себе два або три джерела тактових імпульсів:

- LFXT1CLK низькочастотний/високочастотний осцилятор, який може використовуватись як з низькочастотними часовими кварцевими резонаторами на 32768 Гц, так і з стандартними кристалами, резонаторами та/або зовнішніми джерелами тактування в діапазоні від 450 кГц до 8 МГц;
- LFXT2CLK додатковий високочастотний осцилятор може використовуватись з стандартними кристалами, резонаторами або зовнішніми джерелами тактових сигналів в діапазоні від 450 кГц до 8 МГц;
- DCOCLK вбудований осцилятор з цифровим керуванням (DCO) RC-типу.

Від основного модуля тактування можна отримати три тактових сигнала:

- допоміжне тактування ACLK. Модуль ACLK – це буферизоване джерело тактових імпульсів LFXT1CLK з подільником на 1, 2, 4 або 8. ACLK програмно вибирається для конкретних програмних модулів;
- основне тактування MCLK. Модуль MCLK програмно вибирається як LFXT1CLK, LFXT2CLK (якщо доступний) або DCOCLK. MCLK ділиться на 1, 2, 4 або 8. MCLK використовується ЦПУ та системою;
- другорядне тактування SMCLK. Модуль SMCLK програмно вибирається як LFXT1CLK, LFXT2CLK (якщо доступний) або DCOCLK. SMCLK ділиться на 1, 2, 4 або 8. SMCLK програмно вибирається для конкретних перефірійних модулів.

Схематично основний модуль татування зображено на рис. 3.3.

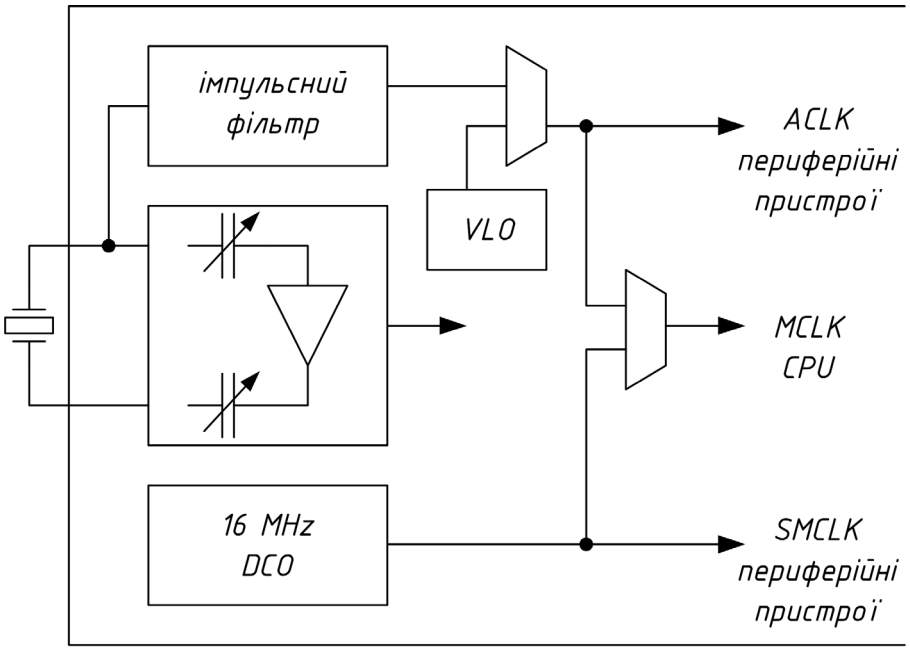


Рисунок 3.3 – Основний модуль тактування

Низькочастотний генератор Very Low-power Oscillator (VLO) є новим для сімейства MSP430, його частота складає 12 кГц. Він може бути використаний для отримання тактового сигналу ACLK для малопотужної периферії. Завдяки низькому рівню споживання цього модуля (500 нА), можна в енергозберігаючому режимі підтримувати роботу таких модулів, як таймер періодичних переривань, «вартовий» таймер і т. ін. При цьому стабільність частоти, яка генерується можна порівняти з характеристиками кварцевих резонаторів. Якщо для обробки будь-якої події мікроконтролеру необхідна повна потужність, то автоматично вмикається основний генератор тактової частоти DCO, що забезпечить вихідну частоту до 16 МГц за 1 мкс. За рахунок такого швидкого перемикання можна більш ефективно

використовувати енергозберігаючи режими роботи контролера. На рисунку 3.4 схематично зображена процедура калібрування VLO.

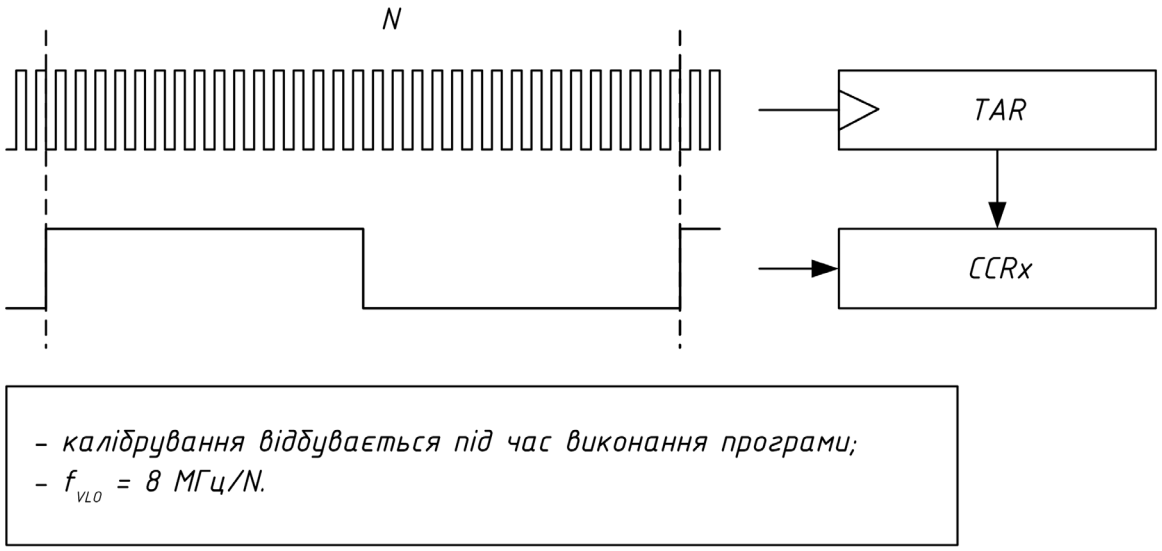


Рисунок 3.4 – Калібрування VLO

Після сигналу PUC основний модуль тактування використовує DCOCLK для формування MCLK. Якщо необхідно, в якості джерела сигналу для MCLK можна використати LFXT1 або XT2.

Для зміни джерела тактування сигналу MCLK з модуля DCO на тактування від кварцевого резонатора (LFXT1CLK або XT2CLK) використовується наступна послідовність команд:

1) переключення на кварцовий резонатор;
2) очищення флагу OFIFG;
3) очікування на протязі приблизно 50 мкс;
4) перевірка OFIFG та повторення кроків 1-4 до тих пір, поки OFIFG залишається очищеним.

В пристроях на основі MSP430x1xx з живленням від батарей зазвичай існують наступні суперечливі вимоги:

- низька тактова частота для економії енергії і збільшення часу роботи від батарей;

- висока тактова частота для швидкої реакції на події і забезпечення можливості швидкої обробки інформації.

Основний модуль тактування дозволяє користувачеві обходити вищенаведені протиріччя шляхом вибору найбільш оптимального з трьох можливих сигналів тактування: ACLK, MCLK і SMCLK. Для оптимальної продуктивності з низьким енергоспоживанням модуль ACLK може бути сконфігурований на роботу від годинникового кварцевого резонатора на 32768 Гц, що забезпечує стабільне тактування для системи і мале споживання в режимі очікування. MCLK може налаштовуватись на роботу від інтегрованого модуля DCO, який активується тільки при появі запиту на

обробку переривання. SMCLK можна конфігурувати на роботу як від годинникового кварцевого резонатора, так і від DCO, залежно від вимог периферії. Гнучкий розподіл тактових сигналів і наявність системи ділення тактової частоти забезпечує тонке налаштування індивідуальних потреб тактування.

На рисунку 3.5 наведена залежність системної частоти від напруги в різних режимах роботи процесора.

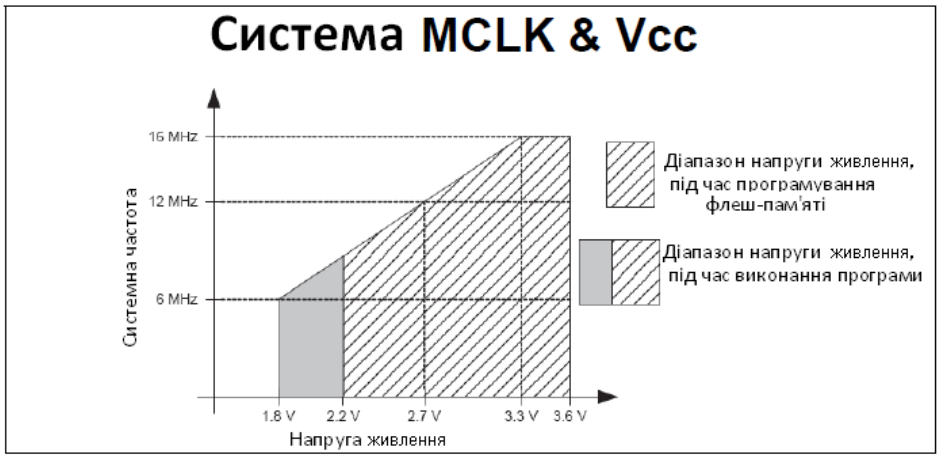


Рисунок 3.5 – Залежність системної частоти від напруги споживання при різних режимах роботи

3.2 Порядок виконання роботи

1. Створіть новий проект (`File->New->CCS Project`) і назвіть його Lab3. Зніміть прапорець, «використовують розташування за замовчуванням». Використовуючи кнопку Browse (огляд) необхідно переміститися до: `C:\MSP430_LaunchPad\Labs\Lab3\Project`. Натисніть «OK» а потім Next. Опції у наступних трьох вікнах повинні прийняти значення за замовчуванням. Використайте значення за замовчуванням, і в останньому вікні натисніть кнопку Finish.
2. Додайте вихідний файл до проекту (`File->New->Source File`) і назвіть його Lab3.c і натисніть кнопку Finish.
3. У порожньому вікні введіть наступний код:

```
#include <msp430g2231.h>
void main(void)
{
//code goes here
}
```

4. Для роботи в подальшому необхідно ознайомитись з вмістом двох файлів:

- msp430g2231.h. Заголовочний файл де описані макроси для доступу до ресурсів MSP430;
- MSP430G2xx User's Guide. Керівництво користувача MSP430.

Для налагодження, було б зручно зупинити таймер:

```
WDTCTL = WDTPW + WDTHOLD;
```

WDTCTL – watchdog timer control register (регістр управління). Ця команда задає пароль (WDTPW), щоб зупинити таймер (WDTHOLD).

5. Далі необхідно налаштувати світлодіод, який підключений до лінії GPIO. Зелений світлодіод розташований за адресою: порт 1, біт 6. Світлодіод включається, коли біт встановлено в рівень логічної «1». Потім у код програми необхідно ввести наступні два рядки коду:

```
P1DIR = 0x40;
P1OUT = 0;
```

6. Тепер налаштуємо системний годинник. В новому рядку введіть:

```
BCSCTL3 |= LFXT1S_2;
```

7. BCSCTL3 є одним з основних регістрів системи Clock Control. В наступному рядку введіть:

```
IFG1 &= ~OFIFG;
```

де IFG1 це прапор переривання, а OFIFG – прапор помилки (перша буква «О», а не нуль).

8. Необхідно витримати паузу близько 50 мкс для того, щоб система могла зреагувати. Зупинка DCO дозволить виграти час. В наступному рядку введіть:

```
_bis_SR_register(SCG1 + SCG0);
```

де SR – регістр статусу.

9. В наступному рядку розташуйте інструкцію зіставного присвоєння:

```
BCSCTL2 |= SELM_3 + DIVM_3;
```

10. На даний момент, ваш код повинен виглядати так як наведено нижче. Натисніть кнопку Save на панелі меню, щоб зберегти файл.

```
#include <msp430g2231.h>
void main(void)
{
 WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
 P1DIR = 0x40;             //P1.6 output (green
LED)
 P1OUT = 0;                // LED off
 BCSCTL3 |= LFXT1S_2;      // LFXT1 = VLO
 IFG1 &= ~OFIFG;           // Clear OSCFault flag
 __bis_SR_register(SCG1 + SCG0);    // Stop DCO
 BCSCTL2 |= SELM_3 + DIVM_3;        // MCLK = VLO/8
}
```

11. Для того, щоб перемикати світлодіод, необхідно додати наступний цикл:

```
{
 P1OUT = 0x40;                        // LED on
 _delay_cycles(100);
 P1OUT = 0;                           // LED off
 _delay_cycles(5000);
}
```

12. Тепер, повний код повинен виглядати наступним чином:

```
#include <msp430g2231.h>
void main(void)
{
 WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
 P1DIR = 0x40;             //P1.6 output (green
LED)
 P1OUT = 0;                // LED off
 BCSCTL3 |= LFXT1S_2;      // LFXT1 = VLO
 IFG1 &= ~OFIFG;           // Clear OSCFault flag
 __bis_SR_register(SCG1 + SCG0);    // Stop DCO
 BCSCTL2 |= SELM_3 + DIVM_3;        // MCLK = VLO/8
 while(1)
 {
  P1OUT = 0x40;                // P1.6 on (green
LED)
  _delay_cycles(100);
  P1OUT = 0;                   // green LED off
  _delay_cycles(5000);
 }
}
```

13. Натисніть кнопку «Build» і дивіться як працює програма у вікні консолі. Перевірити на наявність помилок можна виконати у вікні Problems.

14. Натисніть кнопку «Debug». Програма в постійну пам’ять мікроконтролера завантажиться автоматично, а процес виконання зупиниться на початку функції `main()`.

15. Необхідно запустити код програми на виконання і якщо все працює правильно, зелений світлодіод повинен перемикатись приблизно раз в три секунди. Запуск процесора від інших джерел тактування значно прискорить системний годинник. Цей процес описується в другій частині лабораторної роботи.

16. Натисніть на кнопку Save, щоб зупинити відлагодження і повернутися до C/C++. Збережіть роботу, натиснувши `File->Save As` і виберіть зберегти в папці `C:\MSP430_LaunchPad\Labs\Lab3\Files`. Ім'я

файлу Lab3a.c. Натисніть кнопку Save. Закрийте вкладку Lab3a і двічі клацніть на Lab3.c в панелі проекту.

17. Натисніть кнопку Terminate all щоб припинити налагодження і повернутися до C/C++. Збережіть роботу натиснувши File->Save As і виберіть Save в папці: C:\MSP430_LaunchPad\Labs\Lab3\Files. Назвіть файл Lab3a.c, натисніть Save. Закрийте вкладку Lab3a і двічі клацніть на Lab3.c в області Projects.

Запуск процесора в схемі

Кварцевий резонатор частотою 32768 Гц, приблизно в три рази швидший, ніж VLO. Якщо ми запустимо код програми з використанням резонатора, зелений світлодіод повинен блимати приблизно один раз в секунду.

1. Ця частина лабораторної роботи використовує попередній код програми в якості відправної точки. Почніть з редагування верхньої частині коду для забезпечення можливості використовувати обидва світлодіода. Необхідно зробити обидва контакти мікроконтролера (P1.0 і P1.6) виходами. Замініть рядок `P1DIR = 0x40;` на `P1DIR = 0x41;`

2. Якщо починати з червоного світло діода на лінії P1.0, то необхідно замінити `P1OUT = 0;` на `P1OUT = 0x01;`

3. Необхідно обрати зовнішній кристал з низькою вхідною тактовою частотою. Для цього замініть рядок `BCSCTL3 |= LEXT1S_2;` на `BCSCTL3 |= LEXT1S_0;`

4. У попередньому варіанті коду програми скидається прапорець OSCFault. За замовчуванням системний годинник буде так чи інакше тактуватися VLO. Наразі необхідно переконатися, що прапорець залишився скинутим, а це свідчитиме про те, що кристал дійсно працює. Для цього замість рядка `IFG1 &= ~OFIFG;` застосовуйте цикл з перевіркою:

```
while(IFG1&OFIFG)
 { IFG1 &= ~OFIFG;
   _delay_cycles (100000);
 }
```

В циклі **while**`(IFG1&OFIFG)` перевіряється біт `OFIFG` у регістрі `IFG1`. Якщо цей біт буде встановлено в нуль цикл завершиться. Необхідно зачекати 50 мкс після зкидання прапорця, щоб перевірити його ще раз. За допомогою рядка `_delay_cycles(100000);` цикл триватиме набагато довше. Зазначений цикл має тривати на стільки довго, щоб ви могли спостерігати червоне світло на початку роботи програми. В протилежному випадку цикл завершиться дуже швидко і людське око не зможе побачити як спалахне червоний світлодіод.

5. Після циклу потрібно додати ще один рядок коду, який вимкне червоний світлодіод:

```
P1OUT = 0;
```

6. Оскільки в код внесено багато змін (є можливість зробити помилку), переконайтеся, що ваш код буде виглядати як показано нижче.

```
   #include <msp430g2231.h>
   void main(void)
   {
    WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
    P1DIR = 0x41;              // P1.0 and P1.6 output
    P1OUT = 0x01;              // red LED on
    BCSCTL3 |= LFXT1S_0;       // LFXT1 = 32768
crystal
    while(IFG1 & OFIFG)
     {
      IFG1 &= ~OFIFG;          // Clear OSCFault flag
      _delay_cycles(100000);   // delay for flag
     }
     P1OUT = 0; // red LED off
     __bis_SR_register(SCG1 + SCG0); // Stop DCO
     BCSCTL2 |= SELM_3 + DIVM_3;     // MCLK = 32768/8
     while(1)
      {
       P1OUT = 0x40;                 // green LED on
       _delay_cycles(100);
       P1OUT = 0;                    // green LED off
       _delay_cycles(5000);
      }
   }
```

7. Натисніть кнопку Built та скомпілюйте проект. Результат процесу компіляції можна спостерігати у вікні Console. Перевірте наявність помилок у вікні Problems.

8. Натисніть кнопку Debug для переходу в режим від лагодження програми. Програма завантажиться в мікроконтролер автоматично, а процес виконання зупиниться на початку функції `main()`. Запустіть програму на виконання.

9. Подивіться уважно на світлодіоди на макетній платі LaunchPad. Якщо все працює правильно, червоний світлодіод буде блимати дуже швидко, а зелений світлодіод повинен блимати кожну секунду або в близько до цього значення. Очевидно, що зі збільшенням частоти тактування мікроконтролера, швидкість поблискування світлодіодів збільшилася принаймні в три рази. Зупиніть виконання коду.

10. Натисніть на кнопку Terminate All щоб зупинити налагодження і поверніться до C/C++ проекту. Збережіть файл програми, натиснувши `File->Save As`, і виберіть папку, в якій файл буде збережено. Для збереження рекомендується `C:\MSP430_LaunchPad\Labs\Lab3\Files`. Назвіть файл Lab3b.c і натисніть Save. Закрийте Lab3b.c і двічі клацніть на Lab3.c в області Projects.

Запуск процесора на DCO

Найменша частота, за допомогою якої можна керувати DCO складає 1 МГц (це також є швидкістю по замовчанню). Можливо налаштувати систему тактування мікроконтролера MCLK на роботу від DCO. У більшості випадків тактування доцільно налаштувати таким чином, щоб ACLK керував VLO оскільки майже всі системи ACLK працюють або на VLO або на кристалі з частотою 32768 Гц. В даному випадку тактовий генератор ACLK розміщений на кристалі, тож його необхідно просто увімкнути.

1. Можна просто залишити схему в такому вигляді, як вона є, але доцільно було б її відкалібрувати. Відразу після рядка коду, який зупиняє «вартовий» таймер, додайте наступний код:

```
   if(CALBC1_1MHZ == 0xFF || CALDCO_1MHZ == 0xFF)
    {
     while(1);     //If cal constants erased, trap
CPU!!
    }
   BCSCTL1 = CALBC1_1MHZ;  // Set range
   DCOCTL = CALDCO_1MHZ;   // Set DCO step+modulation
```

Зверніть увагу на деякі особливості. На даному етапі можна стерти сегменти інформації з флеш-пам'яті, то ж будьте уважні при введенні коду програми.

2. В коді програми необхідно закоментувати рядок, який зупиняє DCO. Зазначений рядок тепер має виглядати наступним чином:

```
      //__bis_SR_register(SCG1 + SCG0); // Stop DCO
```

3. Також необхідно змінити джерело тактування DCO на MCLK. Для цього потрібно рядок `BCSCTL2 |= SELM_3 + DIVM_3;` замінити на `BCSCTL2 |= SELM_0 + DIVM_3;`

4. Після внесених змін код повинен виглядати як показано нижче.

```
   #include <msp430g2231.h>
   void main(void)
   {
    WDTCTL = WDTPW + WDTHOLD;  //Stop watchdog timer
    if(CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
     {
      while(1);              //If cal const erased,
TRAP!
     }
    BCSCTL1 = CALBC1_1MHZ;     //Set range
    DCOCTL = CALDCO_1MHZ;      //Set DCO step
    P1DIR = 0x41;              //P1.0&6 out(r./gr.
LEDs)
```

```
    P1OUT = 0x01;                 //red LED on
    BCSCTL3 |= LFXT1S_0;          //LFXT1 = 32768 crystal
    while(IFG1 & OFIFG)
     {
      IFG1 &= ~OFIFG;             //Clear OSCFault flag
      _delay_cycles(100000);      //delay for flag and
     }
    P1OUT = 0;                    // red LED off
    //__bis_SR_register(SCG1 + SCG0);     // Stop DCO
    BCSCTL2 |= SELM_0 + DIVM_3;           // MCLK =
DCO
    while(1)
     {
      P1OUT = 0x40;                       // green LED on
      _delay_cycles(100);
      P1OUT = 0;                          // green LED off
      _delay_cycles(5000);
     }
   }
```

В разі необхідності код може бути знайдений в файлі `DCO_XT.txt`.

5. Натисніть кнопку Built та скомпілюйте проект. Результат процесу компіляції можна спостерігати у вікні Console. Перевірте наявність помилок у вікні Problems.

6. Натисніть кнопку Debug для переходу в режим від лагодження програми. Програма завантажиться в мікроконтролер автоматично, а процес виконання зупиниться на початку функції `main()`. Запустіть програму на виконання

7. Подивіться уважно на світлодіоди на макетній платі LaunchPad. Якщо все працює правильно, червоний світлодіод буде блимати дуже швидко, а зелений світлодіод повинен блимати ще швидше. DCO працює на частоті 1 МГц, і це приблизно в 33 рази більше, ніж частота годинникового кристалу 32768 Гц. Таким чином, зелений світлодіод повинен мерехтіти близько 30 разів за секунду.

8. Натисніть на кнопку Terminate All щоб зупинити налагодження і повернутися до C/C++ проекту. Збережіть файл програми, натиснувши `File->Save As`, і виберіть папку, в якій файл буде збережено. Для збереження рекомендується `C:\MSP430_LaunchPad\Labs\Lab3\Files`. Назвіть файл Lab3c.c і натисніть Save. Закрийте Lab3c.c і двічі клацніть на Lab3.c в області Projects.

Оптимізований код запуску процесора на DCO та на кристалі

Попередній код програми не оптимізований, тому необхідно провести його оптимізацію. Видаліть код з вікна редактора (клацніть у будь-якому місці тексту, натисніть Ctrl-A, потім натисніть Delete). Скопіюйте та встав-

те код з файлу OPT_XT.txt в Lab3.c. Перевірте код, визначте як все працює. Додана функція, яка об'єднує несправності питання, видаляє затримки і виправляє код. Код повинен працювати, як і раніше. Для перевірки правильності роботи функції, закоротіть контакти XIN і XOUT, перш ніж натиснути кнопку Run. Це має гарантовано викликати помилку на кристалі. Вам необхідно привести в дію цикл LaunchPad, щоб перезавантажити помилку.

Натисніть на кнопку Terminate All щоб зупинити налагодження і повернутися до C++ проекту. Збережіть роботу, натиснувши File->Save As, і виберіть зберегти в папці C:\MSP430_LaunchPad\Labs\Lab3\Files Назвіть файл Lab3d.c і натисніть Save. Закрийте Lab3d.c.

Управління центральним процесором на DCO без кристалу.

Найменша частота, за допомогою якої можна керувати DCO складає 1 МГц. Майже всі системи ACLK працюють або на VLO або на кристалі з частотою 32768 Гц. ACLK в даному випадку виконаний на кристалі, тож його необхідно просто увімкнути і відкалібрувати DCO.

1. Двічі клацніть на Lab3.c у вікні Projects. Видаліть весь код з файлу. Скопіюйте та вставте код з раніше збереженого Lab3a.c в Lab3.c.
2. Можна просто залишити схему в такому вигляді, якому вона є, але ми будемо її калібрувати. Відразу після коду, який зупиняє watchdog timer, додайте наступний код:

```
if(CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
  {
   while(1);     //If cal constants erased, trap
CPU!!
  }
BCSCTL1 = CALBC1_1MHZ;    //Set range
DCOCTL = CALDCO_1MHZ;     //Set DCO step +
modulation
```

Зверніть увагу на деякі особливості саме на даному етапі. Можна стерти сегменти інформації з флеш-пам’яті.

3. Ми повинні закоментувати рядок, який зупиняє DCO. Закоментуйте наступні рядки:

```
// __bis_SR_register(SCG1 + SCG0); // Stop DCO
```

4. Нарешті, ми повинні впевнитися, що MCLK є джерелом DCO.

Замінити: `BCSCTL2 |= SELM_3 + DIVM_3;`

на `BCSCTL2 |= SELM_0 + DIVM_3;`

5. Код повинен виглядати так:

```
#include <msp430g2231.h>
void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;     //Stop watchdog timer
```

```
  if(CALBC1_1MHZ==0xFF || CALDCO_1MHZ==0xFF)
   {
    while(1);                 //If cal const erased,trap
   }
  BCSCTL1 = CALBC1_1MHZ;     // Set range
  DCOCTL = CALDCO_1MHZ;      // Set DCO step + mod
  P1DIR = 0x40;              // P1.6 output(green LED)
  P1OUT = 0;                 // LED off
  BCSCTL3 |= LFXT1S_2;       // LFXT1 = VLO
  IFG1 &= ~OFIFG;            // Clear OSCFault flag
  //__bis_SR_register(SCG1 + SCG0); // Stop DCO
  BCSCTL2 |= SELM_0 + DIVM_3;       // MCLK = DCO/8
  while(1)
   {
    P1OUT = 0x40;                   // green LED on
    _delay_cycles(100);
    P1OUT = 0; // green LED off
    _delay_cycles(5000);
   }
}
```

Код може бути знайдений в разі необхідності в DCO_XT.txt.

6. Натиснувши кнопку Built, можна спостерігати вікно Console. Перевірку на помилки можна здійснити у вікні Problems.

7. Натисніть кнопку Debug. Має відкритися Debug Perspective, програма завантажиться автоматично, і курсор буде знаходитись на початку `main()`.

8. Виконати код. Якщо все працює правильно, зелений світлодіод повинен блимати дуже швидко. DCO працює на частоті 1 МГц, що приблизно в 30 разів швидше, ніж кристал з частотою 32768 Гц. Таким чином, зелений світлодіод повинен мерехтіти близько 30 разів на секунду. Коли це зроблено зупиніть код.

9. Натисніть на кнопку Terminate All щоб зупинити налагодження і повернутися до C++ проекту. Збережіть роботу, натиснувши `File->Save As`, і виберіть зберегти в папці *C:\MSP430_LaunchPad\Labs\Lab3\Files*. Назвіть файл Lab3e.c і натисніть Save. Закрийте Lab3e.c і двічі клацніть на Lab3.c в області Projects.

Оптимізований код запуску процесора на DCO і VLO.

Видаліть код з вікна редактора Lab3.c. Скопіюйте та вставте код з OPT_XT.txt в Lab3.c. Перевірте код, визначте як все працює. Додана функція, яка об'єднує несправності питання, видаляє затримки і виправляє код. Код повинен працювати, як і раніше.

Натисніть на кнопку Terminate All щоб зупинити налагодження і повернутися до C++ проекту. Збережіть роботу, натиснувши `File->Save As`,

і виберіть зберегти в папці *C:\MSP430_LaunchPad\Labs\Lab3\Files*. Назвіть файл Lab3f.c і натисніть Save. Закрийте Lab3f.c і двічі клацніть на Lab3.c в області Projects.

Потім закрийте проект, клацнувши правою кнопкою миші на Lab3 в C/C++ вікні Projects і виберіть Close Project.

3.3 Зміст звіту

1. Мета роботи.
2. Програми запуску для різних версій тактування кристалу.
3. Висновки.

Контрольні питання

1. Перерахуйте джерела скидання програмного забезпечення.
2. В яких випадках може бути сгенерований сигнал POR?
3. В яких випадках може бути сгенерований сигнал PUC?
4. Що містить в собі основний модуль тактування?
5. Що таке VLO і де він може бути застосований?
6. Перерахуйте послідовність команд, яку слід використати для зміни джерела тактування.
7. Назвіть найменшу частоту, за допомогою якої можна керувати DCO.

Лабораторна робота № 4
АНАЛОГО-ЦИФРОВИЙ ПЕРЕТВОРЮВАЧ

Мета роботи: вивчити основні принципи роботи вбудованого аналого-цифрового перетворювача.

4.1 Теоретичні відомості

Мікроконтролери MSP430 мають у своєму складі периферійний модуль ADC10, який представляє собою 10-розрядний аналого-цифровий перетворювач (АЦП). А у деяких контролерів їх два – 10 і 12-розрядний. До основних особливостей аналого-цифрового перетворювача можна віднести те, що перетворення може бути запущене за допомогою сигналу від таймера А.

Очевидно, для роботи АЦП потрібна опорна напруга. MSP430 дає можливість використати одне з двох внутрішніх джерел опорної напруги (1.5 В і 2.5 В) або зовнішнє джерело опорної напруги. В окремих випадках ADC10 може працювати з двома джерелами опорної напруги, верхня напруга може набувати значень від 1.4 В до значення напруги живлення, а нижня – від нуля до 1.2 В.

Сигнал модуль ADC10 може брати як з одного із зовнішніх входів, так і з внутрішніх (наприклад, з вбудованого датчика температури). Як і для будь-якої іншої периферії, для аналого-цифрового перетворювача необхідно задати джерело тактування. Всі налаштування проводять в регістрах ADC10.

АЦП в MSP430 може працювати в одному з чотирьох режимів:

- одноразове перетворення сигналу з одного з каналів;
- одноразове перетворення сигналів з декількох входів;
- перетворення сигналу, що повторюються, з одного з входів;
- перетворення, що повторюються, з декількох каналів.

Режим потрібно вибирати відповідно до поставленої задачі. Структурна схема вбудованого в мікроконтролер АЦП наведена на рисунку 4.1.

Всі налаштування роботи вбудованого АЦП проводять з регістрами управління. Регістр ADC10CTL0 – регістр контролю і управління. В цьому регістрі проводиться налаштування і вибір джерела опорної напруги. Ввімкнення модуля ADC10, запуск перетворення, дозвіл переривань – усі ці налаштування проводять в цьому регістрі. Крім того, в регістрі ADC10CTL0 можна задати тривалість вибірки – 4, 8, 16 або 64 тактів ADC10. Очевидно, що чим більше тривалість вибірки, тим перетворення точніше. Але разом з цим збільшується вірогідність того, що сигнал на вході зміниться за час виміру.

В регістрі ADC10CTL1 вибирається джерело, з якого на вхід АЦП подається сигнал, задається джерело тактування для модуля ADC10 і встано-

влюється попередній подільник частоти. Крім того, бітами цього регістра вибирається один з чотирьох режимів роботи перетворювача.

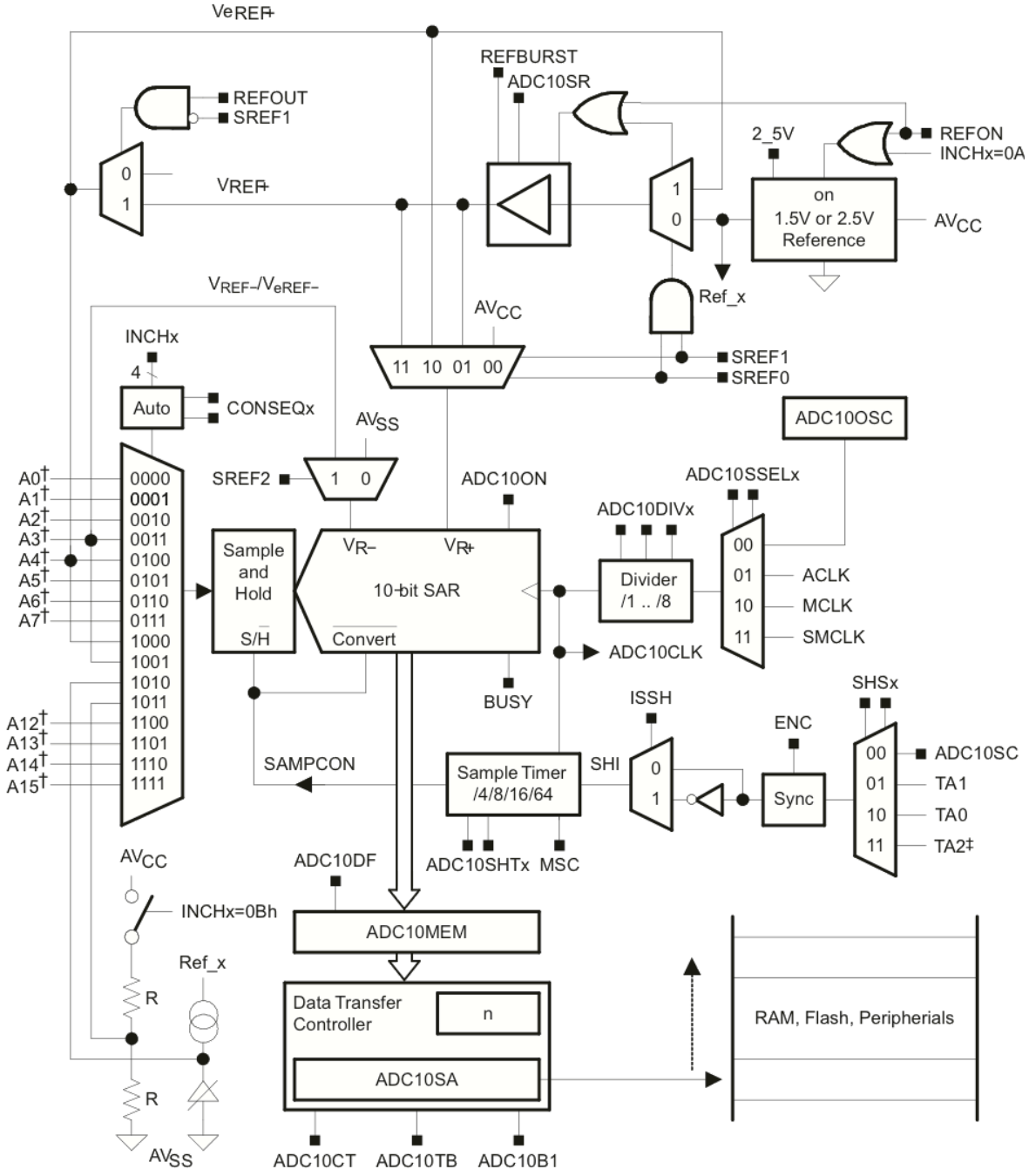


Рисунок 4.1 – Структурна схема вбудованого АЦП

Регістри ADC10AE0 і ADC10AE1 дозволяють включити/виключити будь-який із зовнішніх каналів АЦП. Нульовий біт регістра ADC10AE0 відповідає входу A0, перший – A1, і так далі. В регістрі ADC10MEM зберігається результат перетворення. Більш детальний опис регісрів за допомогою яких можна налаштувати АЦП на роботу можна знайти в детальному описі мікроконтролера MSP430.

4.2 Порядок виконання роботи

1. Створіть новий проект (File->New->CCS Project) і назвіть його Lab4. Зніміть прапорець «use default location» («Використовувати за замовчуванням»). Використовуючи кнопку Browse, перейдіть до папки: *C:\MSP430_LaunchPad\Labs\Lab4\Project*. Натисніть кнопку ОК, а потім натисніть кнопку Next. Наступні параметри у трьох вікнах вже повинні бути встановленими за замовчуванням (project type MSP430, no inter-project dependencies selected, та device variant set to MSP430G2231). Наступні параметри також встановіть за замовчуванням, а в останньому вікні натисніть кнопку Finish.

2. Додайте вихідний файл до проекту (File->New->Source File) та назвіть його Lab4.c та натисніть кнопку Finish.

3. Відкрийте наступні 2 файли, щоб використати їх у даній лабораторній роботі (File->Open File):

C:\MSP430_LaunchPad\Labs\Lab3\Files\OPT_VLO.txt

C:\MSP430_LaunchPad\Labs\Lab2\Files\Temperature_Sense_Demo.txt

4. Скопіюйте весь код з файлу OPT_VLO.txt та вставте його в файл Lab4.c.

5. Переконайтесь, що SMCLK налаштовано правильно. Замініть рядок коду `BCSCTL2 |= SELM_0+DIVM_3;` на `BCSCTL2 |= SELM_0 + DIVM_3 + DIVS_3;`

По замовчанню, після скидання сигнал SMCLK надходить на вхід DCO, а біт DIVS_3 встановлює значення подільника SMCLK на 8. Після виконання приведених вказівок годинник буде налаштований наступним чином:

- ACLK – VLO;
- MCLK – DCO/8 (1 МГц/8 = 125 кГц);
- SMCLK – DCO/8 (1 МГц/8 = 125 кГц).

6. Виконайте завантаження і запуск коду. Якщо все працює правильно, зелений світлодіод повинен мерехтіти. Зупиніть код і натисніть кнопку Terminate all, щоб повернутися в C/C ++.

7. У файлі Temperature_Sense_Demo.txt скопіюйте перші чотири рядки коду з функції `ConfigureAdcTempSensor()`, та вставте їх на початок циклу `while(1)`, трохи вище рядка `P1OUT`. Ці рядки коду наведені нижче:

```
ADC10CTL1 = INCH_10 + ADC10DIV_3;
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON +
ADC10IE;
_delay_cycles(1000);
ADC10CTL0 |= ENC + ADC10SC;
```

8. Розглянемо ці рядки коду, щоб переконатися, що вони працюють коректно. Потрібно знову відкрити посібник користувача та файл для довідки (тримайте відкритим головний файл у редакторі для швидкої довідки).

ADC10CTL1 один з контрольних регістрів ADC10. Бітом INCH_10 обирається внутрішній сенсор температури. Бітом ADC10DIV_0 обирається відповідна тактова частота ADC10. Вибір тактової частоти АЦП відбувається встановленям бітів у регістрі, а джерело може бути внутрішнім ADC10OSC (5MHz), ACLK, MCLK або SMCLK. Внутрішнє джерело частоти ADC10OSC вибирається по замовчанню після процедури скидання. Встановіть наступні біти в регістрі ADC10CTL0:

```
ADC10CTL0=SREF_1+ADC10SHT_3+REFON+ADC10ON+ADC10IE;
```

Регістр ADC10CTL0 містить біти налаштування ADC10:

- бітом SREF_1 обирається проміжок від V_{SS} до V_{REF+};
- бітом ADC10SHT_3 вибирається максимальний час вибірки-зберігання
- бітом REFON налаштовується опорний генератор;
- бітом ADC10ON налаштовується периферія ADC10;
- бітом ADC10IE встановлюється переривання для ADC10.

У даній лабораторній роботі переривання від АЦП не використовується, тому рядок ініціалізації необхідно замінити на наступний:

```
ADC10CTL0=SREF_1+ADC10SHT_3+REFON+ADC10ON;
```

Наступний рядок встановлює час затримки.

```
_delay_cycles(1000);
```

Цикл затримки не є кращим способом для цього, але цілком доречний в рамках даної лабораторної роботи. Час перетворення для аналого-цифровогоперетворення складає не більше 30 мкс. MCLK працює з частотою DCO/8 (1 МГц/8 або 125 кГц), а значення аргументу 1000 провокує затримку 8 мс, і є занадто великим. Значення аргументу 5 буде провокувати затримку в 40 мкс. Змініть значення аргументу як показано нижче:

```
_delay_cycles(5);
```

Наступний рядок коду запускає АЦП на перетворення:

```
ADC10CTL0 |= ENC + ADC10SC;
```

Згідно з вимогами до роботи АЦП в MSP430 встановіть затримку в тринадцять циклів ADC10CLK перш ніж відбудеться зчитування результату перетворення. Тринадцять циклів ADC10CLK з частотою 5 МГц дорівнюватимуть 2.6 мкс, що значно швидше одного циклу DCO/8. Залишіть світлодіод увімкненим і використайте ту ж затримку. Код, що виконує описану процедуру повинен виглядати як показагно нижче:

```
P1OUT = 0x40;
_delay_cycles(100);
```

По завершенню перетворення тактовий генератор повинен бути вимкнений. Біт ENC повинен бути скинутий, для того щоб відбулась зміна біту REF.

Додайте наступні два рядки відразу після _delay_cycles (100);

```
ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);
```

10. Результат перетворення може бути зчитаним з регістру ADC10MEM. Додайте до коду наступний рядок:

```
tempRaw = ADC10MEM;
```

Об'явіть змінну tempRaw після деректив передпроцесора #include на початку коду:

```
volatile long tempRaw;
```

Модифікатор volatile змушує компілятор не оптимізувати поведінку програми пов'язану зі змінною tempRaw.

11. Останні два рядки циклу while(1) вимикають зелений світлодіод і встановлюють затримку для наступного зчитування температури вбудованим датчиком. Цей час може бути будь-яким, але доцільно використати затримку близько 1 секунди між зчитуваннями. Частота таймера становить 125 кГц отже затримка повинна бути 125000 циклів:

12. На даний момент код повинен виглядати як показано нижче. До коду було додано коментарі, для кращого розуміння. Натисніть кнопку Save на панелі меню, щоб зберегти файл.

```
#include <msp430g2231.h>
volatile long tempRaw;
void FaultRoutine(void);
void main(void)
 {
  WDTCTL = WDTPW + WDTHOLD; //Зупинка вартового
таймера
  P1DIR = 0x41;             //P1.0 & 6 виводи
  P1OUT = 0;                //LEDs вимкнені
  if(CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
  FaultRoutine();           //Запит на виклик паст-
ки
  BCSCTL1 = CALBC1_1MHZ;    //Встановлення діапазо-
ну
  DCOCTL = CALDCO_1MHZ;  // Встановлення DCO step +
mod
  BCSCTL3 |= LFXT1S_2;   // LFXT1 = VLO
  IFG1 &= ~OFIFG;        // Очиш. Прапор OSCFault
                         // MCLK=DCO/8,SMCLK=DCO/8
  BCSCTL2 |= SELM_0 + DIVM_3 + DIVS_3;
  while(1)
   {
                         //Temp Sensor ADC10CLK
    ADC10CTL1 = INCH_10 + ADC10DIV_0;
```

```
      ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON +
ADC10ON;
      _delay_cycles(5);    //Затримка врегулювання
Ref
      ADC10CTL0 |= ENC + ADC10SC;  //Запуск перетво-
рення
      P1OUT = 0x40;           //P1.6 вкл. (зелений
LED)
      _delay_cycles(100);     //Затримка для претво-
рення
      ADC10CTL0 &= ~ENC;      //Припиняє перетворення
ADC
      ADC10CTL0 &= ~(REFON + ADC10ON); //Ref і ADC10
OFF
                              //Зчитує перетворене зна-
чення
      tempRaw = ADC10MEM;
      P1OUT = 0;              //викл. зелений LED
      _delay_cycles(125000);    //затримка 1 сек.
     }
   }
  void FaultRoutine(void)
  {
   P1OUT = 0x01;                  //вкл. червоний LED
   while(1);                      //TRAP
  }
```

Для перевірки коду можна відкрити файл Lab4.txt у папці Files.

13. Закрийте файли OPT_VLO.txt і Temperature_Sense_Demo.txt, оскільки вони більше не знадобляться.

14. Натисніть Build і спостерігайте як програма працює у вікні Console. Перевірте код на наявність помилок у вікні Problems.

15. Натисніть кнопку Debug. Відкриється Debug Perspective і програма завантажиться автоматично.

16. Запустіть код програми і спостерігайте за світлодіодами. Якщо все працює правильно то зелений світлодіод повинен мерехтіти приблизно 1 раз за секунду. Зупиніть код натиснувши Halt.

7. Проведіть перевірку роботи аналого-цифрового перетворення. Для того щоб впевнитись у його коректній роботі знайдіть у коді рядок

```
tempRaw = ADC10MEM;
```

Двічі клацніть на tempRaw щоб його виділити. Клацніть правою кнопкою миші і оберіть Add Watch Expression.

18. Клацніть правою кнопкою миші на наступному рядку коду:

```
P1OUT = 0;
```

Оберіть Toggle Breakpoint. Коли програма буде запущена на виконання, вона зупиниться у цій точці, що дозволить виконати читання і оновлення змінної.

19. Запустіть код натиснувши клавішу F8. Він зупиниться у цій точці і значення tempRaw буде оновлено. Повторіть це кілька разів, спостерігаючи за зміною значення. Показання повинні бути стабільними. Типове значення зчитування становить близько 734_{10}, хоча це значення може бути іншим. Якщо необхідно, клацніть по змінній правою кнопкою миші та змініть формат на шістнадцятковий.
20. Піднесіть руку до плати MSP430 та натисніть F8, щоб запустити код. Спостерігатиметься підвищення температури, що підтвердить коректність роботи процесу аналого-цифрового перетворення.
21. Завершіть активний сеанс налагодження за допомогою кнопки Terminate All.
22. Закрийте проект натиснувши правою кнопкою миші по Lab4 у вікні C/C++ Projects і оберіть Close Project.

4.3 Зміст звіту

1. Мета роботи.
2. Програми запуску для різних версій тактування кристалу.
3. Висновки.

Контрольні питання

1. Перерахуйте режими в яких може працювати АЦП MSP430.
2. Опишіть структуру АЦП та особливості його функціонування.
3. Перерахуйте регістри, що відповідають за роботу та налаштування АЦП.
4. В яких регістрах зберігається результат аналого-цифрового перетворення?
5. Яким бітом в регістрі ADC10CTL1 обирається внутрішній сенсор температури?
6. Яким бітом в регістрі ADC10CTL1 обирається відповідна тактова частота?
7. Які налаштування можна зробити регістром ADC10CTL0?
8. Чи використовуються переривання від АЦП в даній лабораторній роботі?
9. Що змушує робити компілятор модифікатор volatile?
10. Як впевнитись в коректній роботі АЦП?

Лабораторна робота № 5
ТАЙМЕР І ПЕРЕРИВАННЯ

Мета роботи: набути навички роботи з вбудованими таймерами для вимірювання та відтворення часових інтервалів.

5.1 Теоретичні відомості

Після налаштування джерела тактування MSP430, можна використовувати його, для управління периферією. Периферія, це пристрої, які вбудовані в мікроконтролер, чи встановлені ззовні і апаратно пов'язані з контролером інтерфейсом. Робота з периферійними пристроями, це задачі для вирішення якийх і створені контролери. Одним з таких периферійних пристроїв є таймер.

В середині мікроконтролера MSP430G2533 є периферійний модуль Timer_A, що представляє з себе 16-розрядний таймер. Оскільки він має шістнадцять двійкових розрядів, максимальне значення до якого може долічити лічильник таймера – 0xFFFF (65535_{10}).

Особливостями таймера Timer_A в MSP430 є:

- три різні режими роботи;
- можливість вибору і налаштування джерела тактування;
- 3 регістри захоплення/порівняння;
- можливість генерувати ШІМ сигнал;
- налаштування переривання на кожну подію.

У таймера Timer_A три режими роботи. Який з них використовується, залежить від програми.

Перший – режим безперервної лічби. В такому режимі роботи таймер лічить від 0 до 0xFFFF, потім розпочинає з початку, і так до нескінченності.

Другий – режим прямої лічби. В такому режимі роботи таймер працює аналогічно першому, але лічить до встановленого значення (верхньої межі), і починає знову з 0. Тільки у цьому режимі ви можете вибирати, верхню межу, до якої рахуватиме таймер.

Третій – режим реверсивного рахунку. Схожий на режим прямого рахунку тим, що в такому режимі роботи також можна встановити верхню межу таймера. Відмінність його в тому, що досягнувши межі, таймер починає лічити вниз, потім, досягнувши 0, знову вгору і так далі.

На додаток до трьох режимів рахунку в Timer_A, є декілька шляхів його використання. Існує можливість встановлювати ключові точки, після досягнення яких, таймер генеруватиме запит на переривання. Мікроконтролери серії Value Line, мають дві/три ключові точки, залежно від моделі. У документації вони називаються регістрами захоплення/порівняння. Їх використання полягає у встановленні прапорців, за якими мікроконтролер

виконуватиме визначені дії. Перший з цих регістрів використовується для установки верхньої межі в режимах прямої і реверсивної лічби. Інші регістри, є регістрами прапорів для процесора, і не впливають на роботу таймера. У режимі безперервного рахунку, перший регістр також є регістром прапорів для процесора.

В кожному мікроконтролері MSP430 для управління роботою периферійного модуля Timer_A використовується п’ять регістрів:

1. TACTL – Timer_A Control Register. Регістр управління Таймера А. Використовується для зв'язку таймера з тактовими сигналами і вибору режимів роботи;

- TASSELx, біти 8 і 9, вказують таймеру, який з тактових сигналів використати;
- IDx, біти 6 і 7, вказують, який подільник частоти тактового сигналу використати, 2, 4 або 8. Ділиться частота, отримана вже після застосування дільника в самому генераторі тактового сигналу;
- MCx, біти 4 і 5, вказують на режим роботи таймера. Якщо вони дорівнюють 0 (встановлено по замовчанню) таймер повністю зупинений;
- TACLR, біт 2. Якщо його встановити в 1, це приведе до скидання таймера. Мікроконтролер автоматично вписує в цей біт 0, після перезапуску таймера;
- TAIE і TAIFG, біти 0 і 1, відповідно. Контролюють переривання таймера.

2. TAR – Timer_A Register. Регістр лічильника Таймера А, в ньому міститься поточне значення таймера;

3. TACCRx – Timer_A Capture/Compare Registers. Регістри захоплення /порівняння Таймера А. Їх може бути два (TACCR0 і TACCR1), чи три (TACCR0, TACCR1 і TACCR2) залежно від моделі мікроконтролера. У режимі порівняння, в цих регістрах знаходиться значення, після досягнення якого, таймер повинен виконати запит на виконання визначеної дії. TACCR0 часто використовується для вказівки верхньої межі рахунку. У режимі захоплення, процесор записує в них поточне значення TAR, по сигналу на вході.

4. TACCTLx – Timer_A Capture/Compare Control Registers. Регістр управління блоком захоплення/порівняння Таймера А. Від його значення, залежать режими роботи регістрів захоплення/порівняння:

- CMx, біти 14 і 15, визначають тип сигналу, по якому відбувається захоплення (по зростаючому, спадаючому, обом фронтам);
- CCISx, біти 12 і 13, обирають джерело сигналу захоплення;
- SCS і SCCI, біти 11 і 10 відповідно, визначають тип синхронізації сигналу захоплення з тактовим сигналом таймера;
- CAP, біт 8, вибір режиму роботи, 1 – режим захоплення, 0 – режим порівняння;

- OUTMODx, біти 5-7, визначають режим роботи модуля виведення, тобто тип реакції на подію захоплення або порівняння;

- CCIE і CCIFG, біти 4 і 0 відповідно, обробка переривань по захопленню/порівнянню;

- CCI і OUT, біти 3 і 2 відповідно визначають вхід і вихід захоплення/порівняння;

- COV, біт 1, сигналізую переповнення захоплення.

5. TAIV – Taimer_A Interrupt Vector Register. Регістр вектору переривання таймера Timer_A. Оскільки переривання від таймера може виникати з різних причин, вміст цього регістра вказує на причину виникнення переривання.

- TAIVx, біти 1-3 кодують причину переривання, запит на яке буде сформований таймером.

На рисунку 5.1 зображена структурна схема периферійного модуля Timer_A.

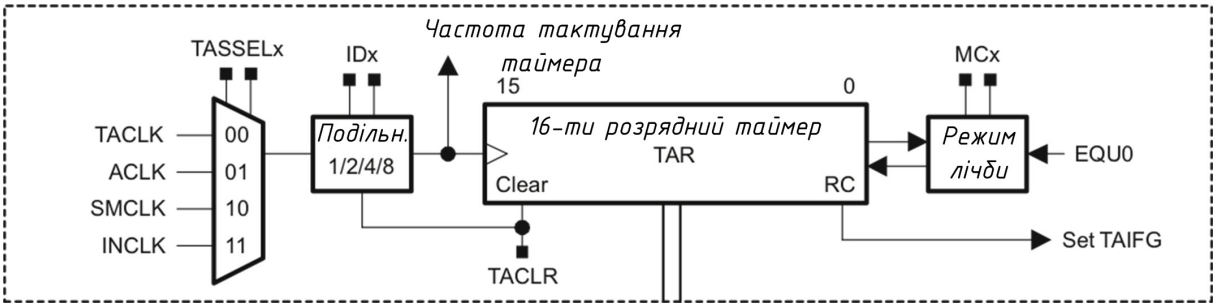


Рисунок 5.1 –Структурна схема периферійного модуля Timer_A

Переривання – це процес переривання поточної послідовності команд, що виконує процесор при настанні події, яка провокує формування сигналу запиту на переривання перефирійним пристроєм.

Механізм переривань створений для забезпечення максимально оперативної реакції програми на певні події. Це дуже важлива частина будь-якого мікроконтрола.

Через необхідність контролювати багато джерел сигналу одночасно, мікроконтролери MSP430 потребують певної організації пріоритетів важливості різної периферії. Тому для MSP430 розроблена чітка і ефективна процедура обробки переривань:

а) якщо сталося переривання від периферійного пристрою, відповідний сигнал встановлює прапор переривання в одному з периферійних регістрів;

б) якщо в процесорному модулі дозволені переривання, то наявність прапора від периферійного переривання, встановлює головний прапор переривань в ЦП;

в) ЦП завершує виконання усіх поточних інструкцій;

г) ЦП зберігає адресу наступної інструкції, що знаходиться в регістрі лічильника команд Program Counter (PC), в стеку;

д) ЦП зберігає поточний статус помістивши вміст регістру стану (Status Register (SR)) в стек;

е) якщо підняті декілька прапорів переривань – обробляється найбільш пріоритетне з них. Пріоритет переривань можна дізнатися в специфікації на мікроконтролер;

ж) прапор переривання занулюється, за винятком ситуації, коли одне переривання провокує встановлення декількох прапорів. В цьому випадку, прапор повинен занулитись програмно обробником переривання;

и) регістр стану SR очищується тому інші переривання не будуть оброблятись поки працює процедура обробки переривання (ISR). ЦП виводиться з режиму зниженого енергоспоживання (Low Power Mode (LPM)), якщо знаходився в цьому режимі;

к) адреса вектору переривання копіюється в лічильник команд PC перенаправляючи процесор на виконання коду, що знаходиться за цією адресою.

Від появи сигналу переривання до початку його виконання проходить 6-12 тактів. Це не особливо велика затримка, але при обробці переривань, критичних до часу обробки, її треба враховувати. Якщо мікроконтролер працює на частоті 1 МГц, то пройде 6-12 мкс, перш ніж мікроконтролер зреагує на подію.

На рисунку 5.2 схематично зображена процедура виклику обробки переривання. Коли процедура обробки переривання (ISR) завершує свою роботу процесор ініціює повернення до виконання основного коду, яке складається з двох кроків:

а) переміщення вмісту збереженого регістра стану зі стека в регістр стану SR;

б) переміщення збереженої адреси наступної команди в лічильник команд мікроконтролера.

Ці кроки займають ще 5 тактів після чого продовжиться виконання основного коду програми. Якщо мікроконтролер знаходився в одному з режимів зниженого енергоспоживання LPM до виникнення переривання він повертається в цей режим автоматично. Якщо повернення в режим енергозбереження не вимагається, у програмі мають бути прописані відповідні інструкції.

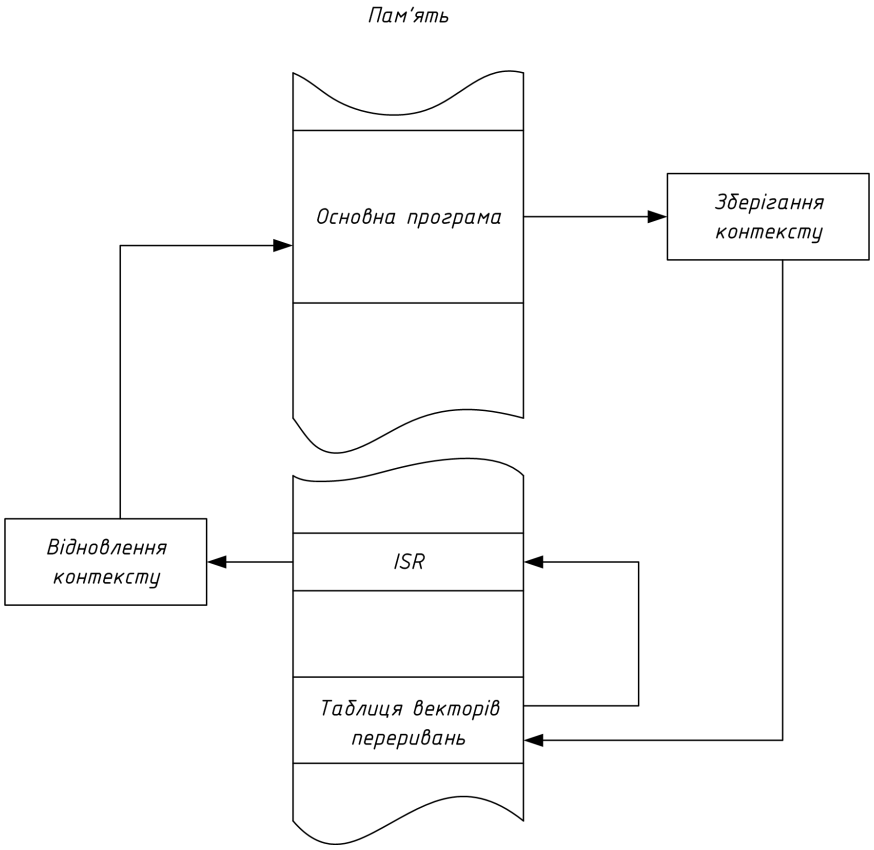


Рисунок 5.2 – Особливості функціонування переривань та стеку

5.2 Порядок виконання роботи

1. Створіть новий проект (`File->New->CCS Project`) і назвіть його Lab5. Зніміть прапорець «use default location». Перейдіть до папки: *C:\MSP430_LaunchPad\Labs\Lab5\Project*. Натисніть кнопку ОК, а потім натисніть кнопку Next. Наступні параметри у трьох вікнах вже повинні бути встановленими за замовчуванням (project type MSP430, no inter-project dependencies selected, та device variant set to MSP430G2231). Наступні параметри також встановіть за замовчуванням, а в останньому вікні натисніть кнопку Finish.

2. Додайте файл до проекту (`File->New->Source File`), назвіть його Lab5.c та натисніть кнопку Finish.

Результуючий файл з попередньої лабораторної роботи буде використовуватися в якості відправної точки. Змінимо вміст файлу, щоб зробити його більш структурованим, організувавши код ініціалізації у вигляді окремих функцій.

3. Відкрийте Lab5_Start.txt використовуючи (`File->Open File`) *C:\MSP430_LaunchPad\Labs\Lab5\Files\Lab5_Start.txt*

4. Скопіюйте весь код з Lab5_Start.txt та вставте його в Lab5.c. Це буде першим етапом до виконання лабораторної роботи.

5. Закрийте файл Lab5_Start.txt він більше не знадобиться.

6. В якості тесту виконайте завантаження і запуск коду. Якщо все працює правильно то зелений світлодіод буде мерехтіти приблизно один раз в секунду (він повинен функціонувати так само, як і в попередній лабораторній роботі). Зупиніть виконання програми і натисніть кнопку Terminate all, щоб повернутися в C/C ++.

7. Знайдіть в коді програми рядок `_delay_cycles(125000);` та видаліть його.

Необхідно реалізувати затримку в одну секунду на базі таймера. В попередній роботі це було виконано з використанням звичайного циклу.

8. Створимо функцію для налаштування Timer_A2. Додадайте прототип функції у верхній частині коду `void ConfigTimerA2(void);`

Потім додайте виклик функції `ConfigTimerA2();` в середині функції `main()` і додайте шаблон для реалізації функції в нижній частині програми:

```
void ConfigTimerA2(void)
 {
 }
```

9. Необхідно реалізувати функцію `ConfigTimerA2()` для налаштування таймера. Додайте наступний код в якості першого рядка функції:

```
CCTL0 = CCIE;
```

Виконання такої інструкції дозволяє мікроконтролеру обробляти переривання лічильника/компаратора у відповідності до стандартної процедури. На відміну від попередньої лабораторної роботи на цей раз буде використовуватись переривання. Додайте в функцію наступні два рядки:

```
CCR0 = 12000;
TACTL = TASSEL_1 + MC_2;
```

Виконання цих інструкцій переведе таймер в режим безперервної лічби з перериванням по досягненню визначеного значення. Зверніть увагу, що `CCR0` та `TACTL` це регістри мікроконтролера, що відповідають за налаштування таймеру Timer_A. `TASSEL_1` та `MC_2` – макровизначення для встановлення та скидання відповідних бітів в регістрах.

Коли таймер досягне значення записаного в регістрі CCR0, буде сгенерований запит на переривання. Оскільки джерелом тактування є VLO з частотою 12 кГц значення для порівняння повинно бути 12000.

10. Встановіть біт дозволу глобальних переривань. Для цього перед нескінченним циклом `while(1)` додайте рядок:

```
_BIS_SR(GIE);
```

11. Попередні налаштування дозволяють процесору обробляти запит на переривання від таймера. На разі необхідно створити функцію обробки запиту на переривання (ISR). Додайте наступний код в нижній частині файлу Lab5.c:

```
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void)
 {
 }
```

12. Видаліть код тіла циклу `while(1)` і вставте його в шаблон функції ISR. Цикл `while(1)` залишіть порожнім.

13. Майже все що потрібно для роботи першого переривання виконано. Для того, щоб реалізувати 2-ге, 3-тє, 4-те переривання з інтервалом в одну секунду, необхідно виконати наступне:

а) прапор переривання повинен бути очищений (це відбувається автоматично);

б) до значення, що знаходиться в регістрі `CCR0` має бути додано 12000. Додайте останнім рядком функції ISR наступне: `CCR0 += 12000;`

14. Необхідно створити програму, яка буде перериватись при досягненні лічильником заданого значення. Додайте в тіло циклу `while(1)` наступний:

```
P1OUT |= BIT0;
for(i = 100; i > 0; i--);
P1OUT &= ~BIT0;
for(i = 5000; i > 0; i--);
```

Оскільки в тілі циклу використовується інструкції, які не мають функціонального навантаження з точки зори корисної дії, вони імовірно будуть видалені з тексту програми на стадії оптимізації. Для того, щоб компілятор не оптимізував шматок коду розташований в тілі циклу, необхідно змінити поведінку оптимізатора щодо змінної лічильника циклу. Змінну лічильник циклу необхідно оголосити з специфікатором типу `volatile`. На початку коду програми розташуйте рядок `volatile unsigned int i;`

15. Для зручності сприйняття коду необхідно зробити деякі зміни:

- в описі функції `FaultRoutine()` замініть `P1OUT=0x01;` на `P1OUT=BIT0;`
- в описі функції `ConfigLEDs()` замініть `P1DIR=0x41;` на `P1DIR=BIT6+BIT0;`
- в функції обробки переривання від таймера (ISR), замініть `P1OUT=0x40;` на `P1OUT |= BIT6;` та замініть: `P1OUT = 0;` на `P1OUT&=~BIT6;`

16. Після модифікацій вихідний код повинен виглядати так, як наведено нижче. Для кращого розуміння в код було додано коментарі.

```
#include <msp430g2231.h>
volatile long tempRaw;
volatile unsigned int i;
void FaultRoutine(void);
void ConfigWDT(void);
void ConfigClocks(void);
void ConfigLEDs(void);
void ConfigADC10(void);
void ConfigTimerA2(void);
void main(void)
 {
  ConfigWDT();
  ConfigClocks();
  ConfigLEDs();
  ConfigADC10();
  ConfigTimerA2();
  _BIS_SR(GIE);                 // Вкл. переривання
  while(1)
   {
    P1OUT |= BIT0;              // Вкл. червоний LED
    for(i = 100; i > 0; i--); // очікув.
    P1OUT &= ~BIT0;             // Викл. червоний LED
    for(i = 5000; i > 0; i--);// очікув.
   }
 }                               // зав. main()
void ConfigWDT(void)
 {
  WDTCTL = WDTPW + WDTHOLD;    // Зупин. сторожовий
                               //таймер
 }
void ConfigClocks(void)
 {
 if(CALBC1_1MHZ==0xFF || CALDCO_1MHZ==0xFF)
  FaultRoutine();              // Якщо калібрув.
                               //дані стираються
                               //run FaultRoutine()
  BCSCTL1 = CALBC1_1MHZ;       //Встанов. діапазон
  DCOCTL = CALDCO_1MHZ;        //Встнов. DCO step +
                               //модуляція
  BCSCTL3 |= LFXT1S_2;         // LFXT1 = VLO
  IFG1 &= ~OFIFG;              // Очист. прапор
                               //OSCFault
  BCSCTL2 |= SELM_0+DIVM_3+DIVS_3; //MCLK =
                              //DCO/8, SMCLK = DCO/8
}
```

```
void FaultRoutine(void)
 {
  P1OUT = BIT0;             // P1.0 на (черв. LED)
  while(1);
 }
void ConfigLEDs(void)
 {
  P1DIR = BIT6 + BIT0;     // P1.6 і P1.0 виводи
  P1OUT = 0;               // LEDs викл.
 }
void ConfigADC10(void)
 {
  ADC10CTL1 = INCH_10 + ADC10DIV_0;// Temp Sensor
                                   //ADC10CLK
 }
void ConfigTimerA2(void)
 {
  CCTL0 = CCIE;             // CCR0 вкл. переривання
  CCR0 = 12000;             // одна секунда
  TACTL = TASSEL_1 + MC_2; // ACLK, безпер. режим
 }
// Timer_A2 переривання
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void)
{
 ADC10CTL0=SREF_1+ADC10SHT_3+REFON+ADC10ON;
 _delay_cycles(5);               // Затримка
 ADC10CTL0 |= ENC+ADC10SC;       // Відбір проб і
                              //перетворення початку
 P1OUT |= BIT6;           // P1.6 вкл. (зелений LED)
 _delay_cycles(100);
 ADC10CTL0 &= ~ENC;  //Відключення перетворення АЦП
 ADC10CTL0 &= ~(REFON+ADC10ON);   // Ref та ADC10
                                  //викл.
 tempRaw = ADC10MEM;  //Зчитув. перетв. значення
 P1OUT &= ~BIT6;       // зелений LED викл.
 CCR0 += 12000;        // Додає одну секунду CCR0
}
```

17. Запустіть програму і спостерігайте за світлодіодами. Якщо все працює правильно, то червоний світлодіод повинен мерехтіти приблизно 2 рази за секунду. Зелений світлодіод повинен мерехтіти один раз за секунду.

18. Переконайтеся в тому, що змінна `tempRaw` все ще знаходиться у вікні перегляду. Якщо це не так – двічі натисніть на ідентифікаторі `tempRaw` у коді правою кнопкою миші і виберіть Add Watch Expression. Якщо необхідно, натисніть на вкладку Watch щоб побачити змінні у вікні перегляду.
19. В середині функції обробки переривання від таймера Timer_A2, знайдіть рядок `P1OUT&=~BIT6;`. Щоб розмістити точку зупинки в цьому місці програми клацніть правою кнопкою миші на цьому рядку і виберіть пункт Toggle Breakpoint.
20. Запустіть код. Виконання програми зупиниться в точці зупинки, а значення змінної `tempRaw` буде оновлено. Зверніть увагу на вікно спостереження за датчиком температури, як і в попередній лабораторній роботі запуск коду відбудеться після натискання клавіші F8.
21. Завершіть активний сеанс налагодження натиснувши кнопку Terminate All.
22. Закрийте проект натиснувши правою кнопкою миші по файлу Lab5 у вікні C/C++ Projects і оберіть пункт Close Project.

5.3 Зміст звіту

1. Мета роботи.
2. Програми запуску для різних версій тактування кристалу.
3. Висновки.

Контрольні питання

1. Що із себе представляє перефірійний модуль Timer_A?
2. Як визначити максимальне значення до якого може долічити лічильник таймера?
3. Перерахуйте основні характеристики таймера.
4. Перерахуйте режими роботи таймера Timer_A.
5. Що таке регістри захоплення/порівняння?
6. Яка процедура переривання реалізована в MSP430?
7. Назвіть особливості функціонування переривань і стеку.

Лабораторна робота № 6

РЕЖИМИ НИЗЬКОГО ЕНЕРГОСПОЖИВАННЯ

Мета роботи: вивчення різних методів використання режимів низького енергоспоживання.

6.1 Теоретичні відомості

За забезпечення контролю живлення в МК MSP430 відповідає спеціалізований модуль. До його складу входить регулятор напруги, що забезпечує роздільне живлення периферії і процесорного ядра. За рахунок регулювання напруги живлення ядра існує можливість керувати енергоспоживанням мікроконтролера, оскільки від рівня напруги змінюється максимальна тактова частота процесора. На рисунку 6.1 приведений графік робочих областей мікроконтролера залежно від напруги живлення процесорного ядра. Для досягнення мінімального споживання (0,1 мкА) вбудований регулятор можна відключити.

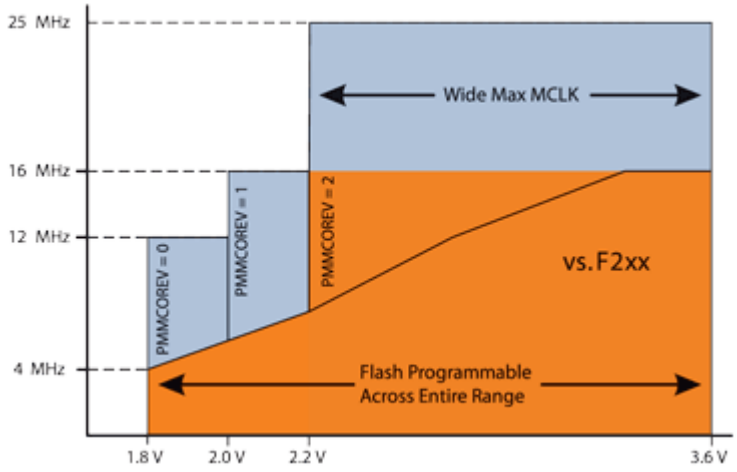


Рисунок 6.1 – Робочі області роботи МК MSP430

Два додаткові супервізори можуть відстежувати напругу живлення мікроконтролера на декількох рівнях і при досягненні їм певного значення або сповіщати систему про те, що рівень напруги змінився (наприклад, для контролю заряду/разяда батарейки), генерувати переривання або апаратне скидання. Ланцюг BOR забезпечує коректне апаратне скидання при стрибках напруги живлення (наприклад, у момент заміни батареї), але при цьому він нічого не споживає.

Всього мікроконтролер має 6 режимів енергоспоживання, діаграма станів приведена на рисунку 6.2, а їх характеристики представлені в таблиці 6.1.

Як вже відмічалось, в режимах зниженого енергоспоживання, коли процесорне ядро відключене, периферія продовжує функціонувати. Наприклад, для MSP430F4xx в режимі LPM3 можуть бути ввімкнені годинник реального часу, драйвер РК-дисплея для відображення інформації, також зберігається вміст ОЗП.

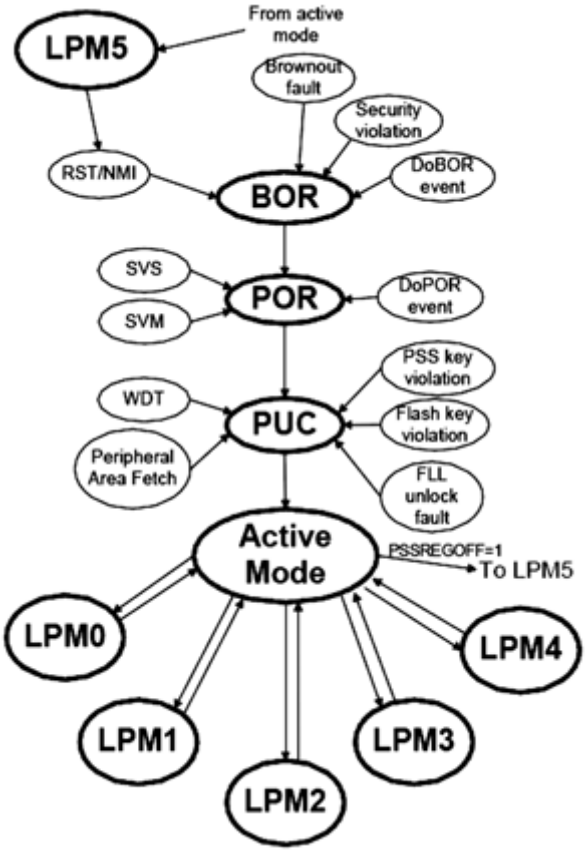


Рисунок 6.2 – Діаграма станів режимів заниженого енергоспоживання

Споживання в такому випадку буде складати приблизно 1 мкА. Для MSP430F5xx при функціонуванні годинника реального часу і вбудованого регулятора напруги споживання в режимі LPM3 складатиме приблизно 2,6 мкА. Час переходу в активний стан займатиме менше 5 мкс, в результаті система практично вмить готова до повноцінної роботи.

Таблиця 6.1 – Характеристики режимів споживання

SCG1	SCG0	OSCOFF	CPUOFF	Режим	Стан системи	Споживання
1	2	3	4	5	6	7
0	0	0	0	Активн	CPU, MCLK, SMCLK, ACLK активні	25 МГц – 5,6 мА
0	0	0	1	LPM0	CPU, MCLK відключені; SMCLK/ACLK активні; DCO включений, якщо від нього тактується ACLK або SMCLK. FLL включене якщо працює DCO	Залежить від кількості активних перефер. модулів.

Продовження таблиці 6.1

1	2	3	4	5	6	7
0	1	0	1	LPM1	CPU, MCLK відключені; SMCLK/ACLK активні; DCO включений, якщо від нього тактується ACLK або SMCLK. FLL відключена	Залежить від кількості активних перефер. модулів.
1	0	0	1	LPM2	CPU, MCLK, SMCLK відключені; ACLK активна; DCO включений, якщо від нього тактується ACLK. FLL відключена	
1	1	0	1	LPM3	CPU, MCLK, SMCLK відключені; ACLK активна; DCO та FLL відключена	2,6 мкА (працюють часи реального часу)
1	1	1	1	LPM4	CPU і усі тактові домени відключені.	1,6 мкА
1	1	1	1	LPM5	CPU і усі тактові домени відключені. Відключений регулятор напруги.	0,1 мкА

6.2 Порядок виконання роботи

1. Створіть новий проект (`File->New->CCS Project`) і назвіть його Lab6. Зніміть прапорець «use default location» («Використовувати за замовчуванням»). Використовуючи кнопку «Browse», перейдіть до папки *C:\MSP430_LaunchPad\Labs\Lab6\Project*. Натисніть кнопку ОК, а потім натисніть кнопку Next. Наступні параметри у трьох вікнах вже повинні бути встановленими за замовчуванням (project type MSP430, no inter-project dependencies selected, and device variant set to MSP430G2231). Наступні параметри також встановіть за замовчуванням, а в останньому вікні натисніть кнопку Finish.

2. Додайте файл вихідного коду до проекту (`File->New->Source File`) та назвіть його Lab6.c та натисніть кнопку Finish.

Результуючий файл з останньої лабораторної роботи буде використовуватися в якості відправної точки.

3. Відкрийте Lab5_Finish.txt використовуючи (`File->Open File…`) *C:\MSP430_LaunchPad\Labs\Lab5\Files\Lab5_Finish.txt*

4. Скопіюйте весь код з Lab5_Finish.txt та вставте його в Lab6.c. Це буде першим етапом до виконання завдання.

5. Закрийте файл Lab5_Finish.txt. Він більше не знадобиться.

6. В файлі Lab6.c знайдіть та замініть рядки коду:

- в функції `ConfigTimerA2()` замініть `CCR0=12000;` на `CCR0=36000;`

- в функції обробки переривання від таймера `Timer_A0()` замініть `CCR0+=12000;` на `CCR0+=36000;`

7. Струм який буде протікати через червоний світлодіод буде спотворювати результати вимірювання, тому за коментуйте два рядка `P1OUT` у циклі `while(1)`.

8. Запустіть програму. Якщо все працює правильно, то зелений світлодіод буде мерехтіти приблизно один раз в 3 секунди. Зупиніть код і натиснувши кнопку Terminate all, щоб повернутися в C/C ++.

9. Виміряйте напругу між Vcc і GND на виводах перемички J6. Очікуване значення має бути близько 3,7 В постійного струму. Результат вимірювання запишіть тут: _____

10. Наразі MCLK і SMCLK встановлені на 125 kHz (DCO/8) з використанням подільника. Необхідно збільшити частоту MCLK до значення 1 MHz, для чого замініть в функції ConfigClocks() рядок `BCSCTL2|=SELM_0+DIVM_3+DIVS_3;` на `BCSCTL2=0;`

11. Запустіть код на виконання. Якщо все працює правильно, то зелений світлодіод буде мерехтіти приблизно один раз в 3 секунди.

12. Потрібно повністю ізолювати мікроконтролер від емулятора, за виключенням землі. Видаліть всі п'ять перемичок з роз'єму J3. Налаштуйте мультиметр на вимірювання струму в межах міліампер. Підключіть червоний дріт мультиметра до верхнього контакту напруги живлення на роз'ємі J3 і чорний дріт мультиметра до нижнього контакту напруги живлення на роз'ємі J3. Далі натисніть кнопку скидання на платі. Якщо мультиметр має достатньо низький внутрішній опір, то зелений світлодіод на платі буде мерехтіти, а на дисплеї мультиметра зявиться результат вимірювання.

Виміряйте струм споживання за умови відсутності емулятора та не працюючих світлодіодів. Для цього виміряйте струм між миготінням зеленого світлодіода. Очікуване значення повинно приблизно дорівнювати 362 мкА. Запишіть результати вимірювання тут: _____

Обережно встановіть чотири перемички на роз’ємі J3 за винятком Vcc. Переконайтесь в тому, що всі виводи налаштовані для роботи у режимі низького енергоспоживання. Згідно з електричною схемою LaunchPad, Port1 з’єднаний з інтерфейсом GPIO (General Purpose Input/Output). Лише лінія P1.3 налаштована на вхід для використання кнопки-перемикача S2, а інші налаштовані як виходи. Входи P2.6 і P2.7 налаштуємо як GPIO.

13. Перейменуйте `ConfigLEDs()` на `ConfigPins()`.

14. Видаліть вміст `ConfigPins()`, та додайте в функцію наступні рядки:

```
P1DIR=~BIT3;
P1OUT=0;
```

15. Дві лінії другого порту (Port2) суміщені з лініями XIN і XOUT. Наразі кварцові резонатори використовуватись не будуть, тому необхідно налаштувати ці лінії порту як GPIO. Для цього біти 6 і 7 регістру P2SEL мають бути очищені. Додайте наступний код в функцію `ConfigPins()`:

```
P2SEL=~(BIT6+BIT7);
P2DIR|=BIT6+BIT7;
P2OUT=0;
```

16. Після всіх змін функція `ConfigPins()` має виглядати як наведено нижче. В вихідному коді також розміщені коментарі для кращого розуміння.

```
void ConfigPins(void)
{
 P1DIR=~BIT3;     // P1.3 введення, інші - виведення
 P1OUT=0;              // очищує контакти виведення
 P2SEL=~(BIT6+BIT7);   // P2.6 і 7 GPIO
 P2DIR|=BIT6+BIT7;     // P2.6 і 7 виведення
 P2OUT=0;              // очищує контакти виведення
}
```

17. Запустіть програму на виконання. Якщо все працює правильно, то зелений світлодіод буде мерехтіти приблизно один раз в 3 секунди.

18. Видаліть чотири джемпери з роз'єму J3. Натисніть кнопку скидання на платі LaunchPad і виміряйте значення струму між мерехтінням зеленого світлодіода. Очікуване значення струму має бути близько 362 мкА. Запишіть результати вимірювання тут: _____

Обережно встановіть чотири джемпери на роз'єм J3, за винятком Vcc. Наразі схема споживання струму MSP430G2231 складається з трьох складових. Процесор споживає 360 мкА на протязі 3 секунд. Струм необхідний для ADC10 складає 250 мкА і споживається мікроконтролером на протязі 33 мкс. Збільшення струму до 600 мкА відбувається за 3 мкс. Якщо б можливо було обмежити кількість часу, на протязі якого процесор активний, загальне споживання струму було б значно менше.

Основна частина потужності споживається програмою під час виконання циклу `while(1)`. Можливо встановити процесор в режим зниженого енергоспоживання і заощадити значну кількість електроенергії.

19. Видаліть весь код з тіла циклу `while(1)`. Видаліть рядок `_BIS_SR(GIE)`. Видаліть рядок `volatile unsigned int i;` з верхньої частини Lab6.c. Додайте рядок у тіло циклу `while(1)`, який буде вмикати переривання і переводити процесор в режим LPM3:

```
_bis_SR_register(LPM3_bits + GIE);
```

20. Наразі код функції `main()` повинен мати наступний вигляд:

```
void main(void)
 {
   ConfigWDT();
   ConfigClocks();
   ConfigPins();
```

```
    ConfigADC10();
    ConfigTimerA2();
    while(1)
     {
      // Встановлення режиму LPM3 з перериваннями
      _bis_SR_register(LPM3_bits + GIE);
     }
  }
```

Ідентифікатор `LPM3_bits` описаний як макрос наступним чином:

```
#define LPM3_bits SR_SCG1+SR_SCG0+SR_CPUOFF
```

а рядок `_bis_SR_register(LPM3_bits + GIE);` встановлює в регістрі стану (SR) наступні біти:

- SCG0 (біт вимикає SMCLK);
- SCG1 (біт вимикає DCO);
- CPUOFF (біт вимикає CPU).

21. Додайте до коду функції обробки переривання (ISR) від таймера `Timer_A0` наступний рядок:

```
_bic_SR_register_on_exit(LPM3_bits);
```

Цей макрос очищує в регістрі статусу (SR) біти описані вище. Після описаних змін функція `Timer_A0` повинна виглядати наступним чином:

```
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A0(void)
 {
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON +
ADC10ON;
  _delay_cycles(5);
  ADC10CTL0 |= ENC + ADC10SC;
  P1OUT |= BIT6;
  _delay_cycles(100);
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~(REFON + ADC10ON);
  tempRaw = ADC10MEM;
  P1OUT &= ~BIT6;
  CCR0 += 36000;
  _bic_SR_register_on_exit(LPM3_bits);
}
```

22. Виконайте завантаження і запуск коду. Якщо все працює правильно, то зелений світлодіод буде мерехтіти приблизно один раз в 3 секунди. Зупиніть код і натисніть кнопку Terminate all, щоб повернутися в C/C ++.

23. Потім видаліть чотири джемпери з роз'єму J3. Натисніть кнопку скидання на платі LaunchPad і виміряйте значення струму між мерехтін-

ням зеленого світлодіода. Очікуване значення стуму має бути близько 0,6 мкА. Запишіть результати вимірювання тут: _____ . Обережно встановіть чотири джемпери на роз'єм J3, за винятком Vcc.

24. Останнім кроком в оптимізації коду, буде видалення затримки програмного забезпечення в ISR. Використаємо Timer_A2 щоб це реалізувати та заощадити ще більше енергії. Без осцилографа виміряти струм споживання не вийде, але ми зможемо переконатися, що струм не збільшиться.

25. Оперційна затримка у Timer_A0 ISR повинна бути відповідною до частоти 1 МГц MCLK яка була встановлена раніше. Для встановлення затримки в 30 мкс замініть рядок `_delay_cycles(5);` на `_delay_cycles(500);`

26. Виконайте завантаження і запуск коду. Зелений світлодіод буде мерехтіти приблизно один раз в 3 секунди, проте дуже швидко. Якщо у вас виникли проблеми з кодом, тоді його можна знайти у папці Files, файл Lab6a.txt.

27. Переконайтеся, що код все ще зчитує температуру, встановіть tempRaw у вікні перегляду. Встановіть точку зупину на рядку `CCR0 += 36000;` в функції обробки переривання від таймера Timer_A0.

28. Запустіть програму в покроковому режимі (F8) та спостерігайте за значенням tempRaw у вікні перегляду. Значення має змінюватись. Зупиніть код і натисніть кнопку Terminate all, щоб повернутися в C/C ++.

29. Використаємо Timer_A2 для реалізації затримки в 3 секунди та 30 мкс. Для реалізації використаємо регістри захоплення і порівняння CCR0 і CCR1. Вони несуть різне функціональне навантаження, але обидва здатні виступати джерелом запиту на переривання. В той момент, коли Timer A Register (TAR) досягне значення, яке записане в CCR0, відбудеться скидання (занулення) регістра TAR. Тому використаємо CCR1 для реалізації затримки в 3 секунди, а CCR0 (3 секунди + 30 мкс).

30. Наразі функція обробки переривання Timer_A2 запускається при досягненні значенням TAR значення, записаного в CCR0. Для зменшення плутанини, змінимо заголовок функції обробки переривання від Timer_A2 наступним чином:

```
//Timer_A2 CCR0 обслуговування переривання (ISR)
#pragma vector=TIMERA0_VECTOR __interrupt void
Timer_A0 (void)
```

31. Змініть вміст функції `ConfigTimerA2()` як показано нижче:

```
void ConfigTimerA2(void)
 {
  CCTL1 = CCIE+OUTMOD_3; //CCR1 int. вкл./скидання
  CCR1 = 36000;          // три секунди
  CCTL0 = CCIE;          // CCR0 int. вкл.
  CCR0 = 36100;          //3 сек.+час встановлення
  TACTL = TASSEL_1+MC_1; // ACLK
}
```

32. Додайте шаблон ISR у кінець Lab6.c.

```
//Timer_A2 CCR1 обслуговування переривання (ISR)
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1(void)
 {
 }
```

33. Зкопіюйте перший та останній рядок функції Timer_A0 і помістіть їх в функцію Timer_A1:

```
// Timer_A2 CCR1 обслуговування переривання (ISR)
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1(void)
 {
  ADC10CTL0=SREF_1+ADC10SHT_3+REFON+ADC10ON;
  _bic_SR_register_on_exit(LPM3_bits);
 }
```

34. У функції Timer_A0, видаліть перші два рядки та рядок `CCR0+=36000;` для усунення затримки. Для того, щоб прапори автоматично не скидались при запуску коду повторно, на початку функцій, додайте наступний рядок `CCTL0 &= ~CCIFG;`. Порівняйте ваш код з наведеним нижче:

```
//Timer_A2 CCR0 обслуговування переривання (ISR)
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A0(void)
 {
  CCTL0 &= ~CCIFG;
  ADC10CTL0 |= ENC + ADC10SC;
  P1OUT |= BIT6;
  _delay_cycles(500);
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~(REFON + ADC10ON);
  tempRaw = ADC10MEM;
  P1OUT &= ~BIT6;
  _bic_SR_register_on_exit(LPM3_bits);
 }

// Timer_A2 CCR1 обслуговування переривання
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1 (void)
 {
  CCTL1&=~CCIFG;
  ADC10CTL0=SREF_1+ADC10SHT_3+REFON+ADC10ON;
  _bic_SR_register_on_exit(LPM3_bits);
 }
```

35. Виконайте завантаження і запуск коду. Якщо все працює правильно, то зелений світлодіод буде мерехтіти настільки швидко, що це буде важко побачити. Зупиніть код і натисніть кнопку Terminate all, щоб повернутися в C/C ++.

36. Щоб отримати підтвердження про завершення перетворення, додайте шаблон функції обробки переривання від АЦП в кінці Lab6.c:

```
// ADC10 обслуговування переривання
#pragma vector=ADC10_VECTOR
__interrupt void ADC10(void)
 {
 }
```

37. Скопіюйте всі рядки функції Timer_A0, які знаходяться нижче `delay_cycles` і вставте їх у функцію `ADC10(void)`.

38. Видаліть рядок, що починається з `P1OUT` у функції Timer_A0.

39. У верхній частині функції для очищення прапору переривання. `ADC10(void)` додайте рядок `ADC10CTL0 &= ~ADC10IFG;`

40. У функції Timer_A1, змініть рядок:

```
ADC10CTL0 = SREF_1+ADC10SHT_3+REFON+ADC10ON;
```

на

```
ADC10CTL0 =
SREF_1+ADC10SHT_3+REFON+ADC10ON+ADC10IE;
```

Після всіх змін код функцій обробки переривань має вигядати наступним чином:

```
//Timer_A2 CCR0 обслуговування переривання (ISR)
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A0(void)
 {
  CCTL0 &= ~CCIFG;
  ADC10CTL0 |= ENC + ADC10SC;
  P1OUT |= BIT6;
  _bic_SR_register_on_exit(LPM3_bits);
 }

//Timer_A2 CCR1 обслуговування переривання (ISR)
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1(void)
 {
  CCTL1 &= ~CCIFG;
  ADC10CTL0=
  SREF_1+ADC10SHT_3+REFON+ADC10ON+ADC10IE;
  _bic_SR_register_on_exit(LPM3_bits);
 }
```

```
//ADC10 обслуговування переривання (ISR)
#pragma vector=ADC10_VECTOR
__interrupt void ADC10(void)
 {
  ADC10CTL0 &= ~ADC10IFG;
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~(REFON + ADC10ON);
  tempRaw = ADC10MEM;
  P1OUT &= ~BIT6;
  _bic_SR_register_on_exit(LPM3_bits);
 }
```

41. Ліквідуйте точки зупину і запустіть код. Якщо все працює правильно, то зелений світлодіод буде мерехтіти настільки швидко, що це буде важко побачити. Встановіть точку зупину на рядку `P1OUT` в функції обробки переривання від АЦП `ADC10(void)` і переконайтеся, що код працює правильно. Зупиніть код і натисніть кнопку Terminate all, щоб повернутися в C/C ++.

42. Видаліть чотири джемпери з роз'єму J3. Натисніть кнопку скидання на платі LaunchPad і виміряйте значення струму між мерехтінням зеленого світлодіода. Очікуване значення вимірюваного струму має бути близько 0,7 мкА. Додаткові 0,1 мкА споживаються Timer_A2 для запуску VLO. Запишіть результати вимірювання тут: _____. Обережно встановіть чотири джемпери на роз'єм J3, за винятком Vcc.

43. Далі, закрийте проект натиснувши правою кнопкою миші по Lab6 у вікні C/C++ Projects і оберіть Close Project.

6.3 Зміст звіту

1. Мета роботи.
2. Програми запуску для різних версій тактування кристалу.
3. Висновки.

Контрольні питання

1. Який модуль відповідає за забезпечення контролю живлення?
2. Перерахуйте режими енергоспоживання які реалізовані в MSP430.
3. Охарактеризуйте кожний режим енергоспоживання.
4. Назвіть приблизне значення струму споживання, яке очікується за умови відсутності емулятора?
5. Назвіть необхідне значення струму для роботи ADC10?
6. В якому режимі енергоспоживання очікуваний струм складатиме 0,7 мкА?

Лабораторна робота № 7
ПОСЛІДОВНИЙ ЗВ'ЯЗОК

Мета роботи: вивичення принципів організації послідовного зв'язку мікроконтролера MSP430 з персональним комп'ютером.

7.1 Теоретичні відомості

В асинхронному режимі USART (універсальний синхроний/ асинхронний приймач/передавач (УСАПП)) підключає MSP430 до зовнішньої системи через дві зовнішні лінії введення/виведення: URXD і UTXD. Режим асинхронної передачі UART задається встановленням біту SYNC в нульове значення.

Режим UART має наступні особливості:

- 7- або 8-розрядні дані з перевіркою парності/непарності і без контролю парності;
- незалежні регістри зсуву передачі і прийому;
- роздільні буферні регістри передачі і прийому;
- передача і прийом розпочинаються з молодшого біта даних;
- вбудовані комунікаційні протоколи вільної лінії і адресного біта для багатопроцесорних систем;
- визначення в приймачі стартового фронту сигналу для автоматичного пробудження з режимів заниженого енергоспоживання;
- програмована швидкість передачі з модуляцією для підтримки дробових величин швидкостей;
- прапори статусу для виявлення помилок, блокування і визначення адреси;
- можливі незалежні переривання для прийому і передачі.

В режимі UART модуль USART передає і приймає символи на швидкості, асинхронній іншому пристрою. Синхронізація кожного символу грунтована на вибраній швидкості передачі USART. Для виконання функцій передачі і прийому використовується однакова швидкість у бодах.

Модуль USART скидається сигналом PUC або при встановленні біта SWRST. Після процедури скидання біт SWRST автоматично встановлюється, залишаючи USART в стані скидання. Коли біт SWRST встановлений, біти URXIEx, UTXIEx, URXIFGx, RXWAKE, TXWAKE, RXERR, BRK, PE, OE, FE скинуті, а біти UTXIFGx і TXEPT встановлені. Прапори дозволу прийому і передачі URXEx і UTXEx не змінюють свого стану. Модуль USART починає працювати після очищення біту SWRST.

Кадр, який передається по USART відповідає стандарту RS-232 і містить стартовий біт, сім або вісім бітів даних, біт контролю парності, адресний біт (у адресному режимі) і один або два стопових біти. Тривалість пе-

редавання бітів визначається вибраним джерелом тактових імпульсів і налаштуванням регістрів швидкості передачі.

Лінії USART мікроконтролера підключені до перетворювача COM-USB TUSB3410. TUSB3410 забезпечує мостове з'єднання між портом USB і розширеним послідовним портом UART. TUSB3410 містить усю необхідну логіку для обміну даними з хостом (комп'ютером) з використанням шини USB. Всередині модуля – ядро мікроконтролера 8052 з 16 кбайт RAM, причому ця RAM може бути завантажена або з хоста через USB, або із зовнішньої підключеної мікросхеми пам'яті по шині I^2C. У TUSB3410 також міститься 10 кбайт ROM, яка дозволяє конфігурувати порт USB при включенні живлення і завантаженні. Весь функціонал пристрою такий, як декодування команд, налаштування UART, повідомлення про помилки – закладений в програмному забезпеченні (firmware). Це програмне забезпечення працює під управлінням хоста. Мікросхема TUSB3410 може бути використана як інтерфейс між з будь-яким пристроєм, що має традиційний послідовний порт, і комп'ютером з портами USB.

7.2 Порядок виконання роботи

1. Створіть новий проект (`File->New->CCS Project`) і назвіть його Lab7. Зніміть прапорець «use default location» («Використовувати за замовчуванням»). Використовуючи кнопку «Browse», перейдіть до папки: *C:\MSP430_LaunchPad\Labs\Lab7\Project*. Натисніть кнопку ОК, а потім натисніть кнопку Next. Наступні параметри у трьох вікнах вже повинні бути встановленими за замовчуванням (project type MSP430, no inter-project dependencies selected, and device variant set to MSP430G2231). Значення інших параметрів також залишить за замовчуванням, а в останньому вікні натисніть кнопку Finish.
2. Додайте source file до проекту (`File->New->Source File`), назвіть його Lab7.c та натисніть кнопку Finish.
3. Відкрийте файл Lab6a.txt використовуючи (`File->Open File…`) *C:\MSP430_LaunchPad\Labs\Lab6a\Files\Lab6a.txt*
4. Скопіюйте весь код з файлу Lab6a.txt та вставте його в файл Lab7.c. Це буде першим етапом до виконання завдання. Цей код не має бути оптимізованим для зменшення енергоспоживання, як це було описано в останній частині попередньої лабораторної роботи. В програмі будуть задіяні Timer_A2 та UART, тому максимально оптимізувати код за енергоспоживанням буде неможливо. Проте незначні корективи можна залишити і при цьому зберегти низьке енергоспоживання.
5. Завантажте і запустіть на виконання код. Якщо все працює правильно, зелений світлодіод буде мерехтіти приблизно один раз в 3 секунди. Код має функціонувати так само, як і в попередній лабораторній роботі. Зупиніть код натиснувши кнопку Terminate all, щоб повернутися в режим C/C++.

6. Обробку переривання від таймера Timer_A2 необхідно видалити, а вартовий таймер (WDT) потрібно налаштувати в режим інтервального таймера. Після змін функція `ConfigWDT()` має виглядати, як наведено нижче:

```
void ConfigWDT(void)
 {
  WDTCTL = WDT_ADLY_250;    // <1 сек інтервал WDT
  IE1 |= WDTIE;             // Вкл. переривання WDT
 }
```

Вибір інтервалів з WDT обмежений, а макрос `WDT_ADLY_250` описаний як:

```
#define WDT_ADLY_250 \
(WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS0)
```

забезпечує затримку 250 мс працюючи від основного джерела тактування. Працюючи від VLO WDT забезпечить затримку трохи менше 1 с.

7. Інструкції, які виконувались раніше при виникненні переивання від Timer_A0 повинні виконуватись при виникненні переривання від WDT. Змініть заголовок функції обробки переривання від таймера:

```
// Timer_A2 обслуговування переривання
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void)
```

на

```
// WDT обслуговування переривання #pragma
vector=WDT_VECTOR
__interrupt void WDT(void)
```

8. Видаліть рядок `CCR0+=36000;`. Наразі немає необхідності обробки переривання від Timer_A2, тому видаліть весь код всередині функції `ConfigTimerA2()`.

9. Виконайте завантаження і запуск коду. Якщо все працює правильно, зелений світлодіод буде мерехтіти приблизно один раз за секунду. Зупиніть виконання коду і натисніть кнопку Terminate all, щоб повернутися в C/C++. Якщо необхідно цей код можна знайти у файлі Lab7a.txt у папці Files.

10. Видаліть обидва рядки `P1OUT…` в функції обробки переривання від WDT. Для продовження виконання роботи знадобляться обидва світлодіода.

11. Необхідно налаштувати лінії передачі і прийому (P1.1 та P1.2), а також інші лінії мікроконтролера відповідним чином. Змініть код функції `ConfigPins(void)` наступним чином:

```
void ConfigPins(void)
 {
  P1SEL |= TXD + RXD;     // P1.1 & 2 TA0, інші GPIO
  P1DIR = ~(BIT3 + RXD); // P1.3 вхід, інші виводи
  P1OUT = 0;              // очищ. виводи
  P2SEL = ~(BIT6 + BIT7);// ств. P2.6 & 7 GPIO
  P2DIR |= BIT6 + BIT7;  // P2.6 & 7 виводи
  P2OUT = 0;              // очищ. виводи
 }
```

12. Необхідно створити функцію, яка б обслуговувала процес передачі даних. Додайте код з файлу Transmit.txt в кінець файлу Lab7.c що знаходиться в папці Files:

```
// Function Transmits Character from TXByte
void Transmit()
 {
  BitCnt = 0xA;
  while(CCR0 != TAR)
  CCR0 = TAR;        // поточний стан лічильника TA
  CCR0 += Bitime;    // певний час до першого біта
  TXByte |= 0x100;  // дод. знак зупинки до TXByte
  TXByte = TXByte << 1;
  CCTL0 = CCIS0 + OUTMOD0 + CCIE;
  while(CCTL0 & CCIE);  // Очікування завершення TX
 }
```

Не забудьте додати прототип функції на початку файлу Lab7.c:

```
void Transmit(void);
```

13. Послідовна передача інформації відбтиметься за допомогою таймера Timer_A, який забезпечує часові інтервали, необхідні для роботи UART на швидкості 2400 бод. Скопіюйте вміст функції обробки переривання Timer_A та вставте його у кінець файлу Lab7.c:

```
// Timer A0 обслуговування переривання
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void)
 {
  CCR0 += Bitime;     // додає Offset до CCR0
  if(CCTL0 & CCIS0)  // TX вкл. CCI0B?
   {
   if(BitCnt == 0)
    {
     CCTL0 &= ~CCIE; // викл. переривання
    } else {
            CCTL0 |= OUTMOD2; // TX простір
            if(TXByte & 0x01)
```

```
            CCTL0 &= ~OUTMOD2;    // TX Mark
            TXByte = TXByte >> 1;
            BitCnt--;
           }
    }
  }
```

14. Налаштуйте Timer_A2. Змініть функцію `ConfigTimerA2()` в Lab7.c наступним на наступний код:

```
void ConfigTimerA2(void)
 {
  CCTL0 = OUT;                  //TXD Idle as Mark
  TACTL = TASSEL_2 + MC_2 + ID_3;
 }
```

15. Для того, щоб код працював, додайте наступне у верхню частину файлу Lab7.c:

```
#define TXD BIT1          // TXD вкл. P1.1
#define RXD BIT2          // RXD вкл. P1.2
#define Bitime 13*4       // 0x0D

unsigned int TXByte;
unsigned char BitCnt;
```

16. Додайте наступне оголошення на початку файлу Lab7.c:

```
volatile long tempSet = 0;
volatile int i;
```

В змінній tempSet буде зберігатись значення температури зчитане за допомогою ADC10. В процесі роботи програми це значення буде порівнюватись з новим виміряним значенням температури для того, щоб зафіксувати її збільшення або зменшення. Моменти зміни температури необхідно фіксувати та відображати зручним для користувача чином. На макетній платі можна задіяти для відображення зелений та червоний світлодіоди, а також передавати інформацію про зміну температури на комп'ютер використовуючи UART.

18. Додайте наступний код управління в цикл **while**(1):

```
_bis_SR_register(LPM3_bits + GIE);
```

Цей код доступний у файлі While.txt:

```
if(tempSet == 0)
 {
  tempSet = tempRaw;          // Set reference temp
```

```
 }
if(tempSet > tempRaw + 5)     // тест для lo
 {
  P1OUT = BIT6;                // зелений LED вкл.
  P1OUT &= ~BIT0;              // червоний LED викл.
  for(i=0; i<5; i++)
   {
    TXByte = TxLO[i];
    Transmit();
   }
 }
if(tempSet < tempRaw - 5)     // тест для hi
 {
  P1OUT = BIT0;                // червоний LED вкл.
  P1OUT &= ~BIT6;              // зелений LED викл.
  for(i=0; i<5; i++)
   {
    TXByte = TxHI[i];
    Transmit();
   }
 }
if(tempSet <= tempRaw + 2 & tempSet >= tempRaw - 2)
 {                            // тест для діапазону in
  P1OUT &= ~(BIT0 + BIT6); // всі LEDs вимкнені
  for(i=0; i<5; i++)
   {
    TXByte = TxIN[i];
    Transmit();
   }
}
```

Програма використовує три стани температури: «LO», «HI» та «IN». Відображаються ці стани з допомогою зеленого і червоного світло діодів, причому, коли значення температури знаходиться в дозволених межах обидва світло діоди не світяться. Як тільки температура підніметься до критичного значення – загориться червоний світло діод. Як тільки температура впаде до мінімально дозволеного значення – загориться зелений світлодіод. Програма також відправляє через UART послідовності ASCII символів, які відповідають кожному з трьох станів. Послідовності складаються з символів режиму та службових символів, для коректної обробки їх термінальною програмою на комп'ютері. Зазначені послідовності ASCII символів зведені до таблиці 7.1.

Таблиця 7.1 – Послідовності ASCII символів, які передаються через UART

Стан	Послідовність
«LO»	`0x4C, 0x4F, 0x0A, 0x08, 0x08`
«HI»	`0x48, 0x49, 0x0A, 0x08, 0x08`
«IN»	`0x49, 0x4E, 0x0A, 0x08, 0x08`

Зі сторони комп'ютера має бути запущена програма, яка прийматиме та відображатиме отриману інформацію на екран у вигляді символів. Це може бути звичайний термінал, який підтримує роботу в режимі ASCII. Для правильного функціонування програми додайте наступні масиви даних на початок файлу Lab7.c:

```
unsigned int TxHI[] = {0x48,0x49,0x0A,0x08,0x08};
unsigned int TxLO[] = {0x4C,0x4F,0x0A,0x08,0x08};
unsigned int TxIN[] = {0x49,0x4E,0x0A,0x08,0x08};
```

19. Виконайте завантаження і запуск коду. Якщо у вас виникли проблеми з запуском програми, порівняйте її код з кодом, що знаходиться у файлі Lab7Finish.txt у папці Files.

20. Переконайтесь в тому, що макетна плата LaunchPad підключена до ПК. В OS Windows, натисніть **Win + R** або виберіть в меню «Пуск» розділ «Виконати» і введіть в діалогове вікно devmgmt.msc. Натисніть OK. Повинен відкритись Диспетчер устройств OS Windows.

21. Відкрийте вкладку «Порти» і знайдіть порт з ім'ям "MSP430 «Application UART». Запишіть номер COM-порту: _____. (наприклад COM45). Закрийте «Диспетчер пристроїв».

22. CCS повинен знаходитись в режимі Debug. Натисніть `View-> Other…`, а у вікні Show View натисніть [+] поруч з Terminal. Оберіть Terminal нижче, і натисніть кнопку OK. Далі натисніть кнопку Settings, і задайте наступні параметри порту:

Connection type – serial;

Port – COMXX (XX – номер відповідного порта);

Baude rate – 2400;

Data bits – 8;

Stop bits – 1;

Parity – none;

Flow control – none.

23. Запустіть код. У вікні Terminal буде відображатися стан температури «IN». Запустіть код на виконання. Коли температура мікроконтролера MSP430 підніметься, засвітиться червоний світлодіод, у вікні Terminal ві-

добразиться стан температури «HI». Коли температура MSP430 спаде, засвітиться зелений світлодіод, а у вікні Terminal відобразиться стан температури «LO». За прийнятної температури обидва світлодіоди будуть вимкнені, і у Terminal pane відобразиться стан температури «IN».

24. Завершіть активний сеанс налагодження за допомогою кнопки Terminate All.

25. Закрийте проект натиснувши правою кнопкою миші по Lab7 у вікні C/C++ Projects і оберіть Close Project.

7.3 Зміст звіту

1. Мета роботи.
2. Програма організації послідовного зв'язку Lanch Pad з персональним комп'ютером.
3. Висновки.

Контрольні питання

1. Які можливості передбачені для роботи USART в MSP430?
2. Які швидкості передавання інформації забезпечує USART в MSP430?
3. Які формати передавання даних підтримуються USART в MSP430?
4. Який таймер забезпечує часові інтервали необхідні для послідовного передавання інформації?
5. Як забезпечити передавання інформації в форматі ASCII?

Англійсько-український словник термінів

ADC (англ. Analog to Digital Converter) – аналого-цифровий перетворювач;

ALU (англ. Arithmetic and Logic Unit) – арифметико-логічний пристрій;

BIOS (англ. Basic Input Output Sistem) – основна система введення/виведення;

BOR (англ. Brown Out Reset) – скидання в наслідок дефіциту напруги живлення;

Breakpoint – точка зупинки;

CCS (англ. Code Composer Studio) – інтегроване середовище розробки;

CLK (англ. Clock) – системна тактова частота, генератор синхронізації;

COM-порт – послідовний порт, сленгове найменування інтерфейсу стандарту RS-232;

Console – сукупність пристроїв інтерактивного введення/виведення;

CPU (англ. Central Processor Unit) – модуль центрального процесора;

Debug – режим запуску програми при якому можна відслідкувати хід її роботи;

Debugger – програма призначена для пошуку помилок в інших програмах;

DSP (англ. Digital Signal Processor) – цифровий сигнальний процесор;

FPP (англ. Float Point Package) – доповнення до програмного забезпечення для реалізації обчислень з плаваючою крапкою;

GUI (англ. Graphic User Interface) – графічний інтерфейс користувача;

GND (англ. Ground) – вузол кола, потенціал якого умовно приймають за нуль;

Halt – апаратна або програмна зупинка комп’ютера без втрати вмісту ОЗП;

ISR – підпрограма обробки переривання;

JTAG (англ. Join Test Action Group) – апаратний інтерфейс на базі стандарту IEEE 1149.1 для тестування та відлагодження;

LED (англ. Light-Emitting Diode) – діод, який випромінює світло, світлодіод;

Linker – компонувальник, редактор зв’язків;

LPM (англ. Low Power Mode) – режим заниженого енергоспоживання;

PC (англ. Program Counter) – програмний лічильник;

POR (англ. Power On Reset) – скидання при появі напруги живлення;

PUC (англ.) – сигнал очищення при включенні;

PWM (англ. Pulse-Width Modulation) – широтно-імпульсна модуляція;

RISC (англ. Reduced Instruction Set Computer) – комп'ютер зі скороченим набором команд;

SP (англ. Stack Pointer) – вказівник на вершину стеку;

SR (англ. Status Register) – регістр статусу;

Terminal – кінцевий пристрій для взаємодії людини з комп'ютером;

Toggle – перемикати;

UART (англ. Universal Asynchronous Receiver Transmitter) – універсальний асинхронний приймач/передавач;

VCC (англ. Voltage at common Collector) – вузол кола, потенціал якого рівний напрузі живлення;

VLO (англ. Very Low-power Oscillator) – генератор синхронізації з надмалим споживанням;

WDT (англ. Watchdog Timer) – вартовий таймер.

Список використаної літератури

1. Кончаловский В. Ю. Цифровые измерительные устройства: Учеб. пособие для вузов / В. Ю. Кончаловский. – М.: Энергоатомиздат, 1985. – 304 с., ил.
2. Винокуров В. И. Электрорадиоизмерения: учеб. пособие для радиотехнич. спец. вузов / В. И. Винокуров, С. И. Каплин, И. Г. Петелин; Под ред. В. И. Винокурова. – 2-е изд., перераб. и доп. – М.: Высш шк., 1986. – 351 с.: ил.
3. Быстродействующие интегральные микросхемы ЦАП и АЦП и измерение их параметров / А. -Й. К. Марцинкявичюс, Э. -А. К. Багданскис, Р. Л. Пошюнас и др.; Под ред. А.-Й. К. Марцинкявичюса, Э. -А. К. Багданскиса. – М.: Радио и связь, 1988. – 224 с., ил.
4. Федорков Б. Г. Микросхемы ЦАП и АЦП / Б. Г. Федорков, В. А. Телец. – М.: Энергоатомиздат, 1990. – 320 с., ил.
5. Клингман Э. Проектирование микропроцессорных систем: Пер. с англ. / Э. Клингман – М.: Мир, 1985. – 363 с., ил.
6. Сташин В. В. Проектирование цифровых устройств на однокристальных микроконтролерах / В. В. Сташин, А. В. Урусов, О. Ф. Мологонцева. – М.: Энергоатомиздат, 1990. – 224 с.
7. Казаринов Ю. М. Применение микропроцессоров и микро-ЭВМ в радиотехнических системах / Ю. М. Казаринов, В. Н. Номоконов, Ф. В. Филиппов – М.: Высш.шк., 1988. – 208 с.
8. Каган Б. М. Основы проектирования микропроцессорных устройств автоматики / Б. М. Каган, В. В. Сташин. – М.: Энергоатомиздат, 1987. – 304 с.
9. Хоуп Г. Проектирование цифровых вычислительных устройств. / Г. Хоуп. – М.: Мир, 1982. – 400 с.
10. Лихтциндер Б. Я. Микропроцессоры и вычислительные устройства в радиотехнике / Б. Я. Лихтциндер, В. Н. Кузнецов – К.: Высшая шк., 1988. – 272 с.
11. Самофалов В. Г. Микропроцессоры / В. Г. Самофалов, О. В. Викторов. – К.: Техника, 1989. – 312 с.
12. СверхБИС универсальных однокристальных микро-ЭВМ / А. В. Кобылинский, Г. П. Липовецкий, Н. Г. Сабадаш и др. – К.: Техника, 1987. – 165 с., ил.
13. Смит Б. Э. Архитектура и программирование микропроцессора 8086. Пер. с англ. / Б. Э. Смит, М. Т. Джонсон – М.: Конкорд, 1992. – 272 с.
14. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT: Пер. с англ. / Р. Джордейн. – М .: Финансы и статистика, 1992. – 544 с.
15. Абель П. Язык ассемблера для IBM PC и программирования. Пер, с англ. / П. Абель. – М.: Высш. шк., 1992. – 423 с.

16. Башков Е. А. Аппаратное и программное обеспечение зарубежных микро-ЭВМ / Е. А. Башков. – К.: Выща шк., 1990. – 207 с.

17. Семейство микроконтролеров MSP430x2xx. Архитектура, программирование, разработка приложений / пер. с англ. Евстифеева А. В. – М.: Додэка-XXI, 2010. – 544 с.: ил.

Список рекомендованої літератури

18. Мячев А. А. Интерфейсы систем обработки даннях / А. А. Мячев, В. Н. Степанов, В. К. Щербо. – М.: Радио и связь. 1989. – 416 с.

19. Науман Г. Стандартные интерфейсы для измерительной техники / Г. Науман, В. Майлинг, А. Щербина. М.: Мир, 1982. – 304 с.

20. Мальцева Л. А. Основы цифровой техники / Л. А. Мальцева, Э. М. Фромберг, В. С. Ямпольский. – М.: Радио и связь, 1986. – 128 с.

21. Мишель Ж. Программируемые контролеры: Пер. с фр. / Ж. Мишель, К. Лоржо, Б. Эспьо. – М.: Машиностроение, 1986. – 320 с.

22. Шило В. Л. Популярные цифровые микросхемы: Справочник / В. Л. Шило. – М.: Радио и связь, 1987. – 352 с.

23. Шевкопляс Б. В. Микропроцессорные структуры. Инженерные решения: Справочник / Б. В. Шевкопляс – М.: Радио и связь, 1990. – 512 с.

24. Токхейм Р. Основы цифровой электроники: Пер. с англ. / Р. Токхейм – М.: Мир, 1988. – 392 с.

25. Микропроцессоры: Справочное пособие для разработчиков судовой РЭА / Г.Г. Гришин, А. А. Мошков, О. В. Ольшанский, Ю. А. Овечкин. – Л.: Судостроение, 1987. – 520 с.

26. Микропроцессорные автоматические системы регулирования. Основы теории и элементы / В. В. Солодовников и др. – М.: Высш. шк., 1991. – 255 с.

27. Майоров В. Г. Практический курс программирования микропроцессорных систем / В. Г. Майоров, А. И. Гаврилов – М.: Машиностроение, 1989. – 272 с.

28. Погорелый С. Д. Программное обеспечение микропроцессорных систем: Справочник / С. Д. Погорелый, Т. О. Слободянюк. –К.: Тех. шк., 1985. – 240 с., ил.

29. Полупроводниковые БИС запоминающих устройств: Справочник / В. В. Баранов и др. – М.: Радио и связь, 1986. – 360 с.

30. Холленд Р. Микропроцессоры и операционные системы: Краткое справочное пособие: Пер. с англ. / Р. Холленд. – М.: Энергоатомиздат, 1991. – 192 с.

31. Проектирование микропроцессорной электронно-вычислительной аппаратуры: Справочник / В. Г. Артюхов и др. – К.: Тех. шк., 1988. – 263 с.

32. Королев Л. Н. Микропроцессоры, микро- и мини-ЭВМ. / Л. Н. Королев. – М.: Изд-во Моск. ун-та, 1988. – 213 с.

Навчальне видання

В. Ю. Кучерук, О. М. Василевський, П. І. Кулаков, К. В. Овчинников

Проектування мікропроцесорних засобів вимірювання

Лабораторний практикум

Редактор В. Дружиніна

Оригінал-макет підготовлено К. Овчинниковим

Підписано до друку 31.08.2017 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Ум. друк. арк. 4,43.
Наклад 50 (1-й запуск 1-20) Зам № 2017-324.

Видавець та виготовлювач
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 59-85-32, 59-87-38.
press.vntu.edu.ua;
Email: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.