2011

Системи для малих та мобільних платформ. Конспект лекцій

Доцент МЗ Кривенко С.А.

ХНУРЕ

10.02.2011

ЗМІСТ

Вступ....3
Тематичний модуль 1. Архітектура мікропроцесора ARM....5
1 Ввідна....6
2 МОДЕЛЬ МІКРОПРОЦЕСОРА ARM....7
2.1 Архітектура ARM....7
2.2 Набір інструкцій....8
3 Мікропроцесор V3....11
3.1 Блок схема....11
3.2 Банк регістрів....14
3.3 Виключення....16
Тематичний модуль 2. Операційна система....23
4 Загальна характеристика....24
4.1 Дві форми операційних систем....24
4.2 контекстна схема операційної системи....26
4.3 Процес ініціалізації задач....27
5 Обробка переривань....31
5.1 Схеми потоків даних щодо обробки переривання....31
5.2 Специфікації процесів переривання....33
6 Припинення задач....35
6.1 Схеми потоків даних щодо припинення задачі....35
6.2 Специфікації процесів припинення задачі....37
Тематичний модуль 3. Прикладне програмне забезпечення....40
7 Загальна характеристика GPS....41
7.1 Програмне забезпечення GPS ARCH....41
7.2 Структура програмного забезпечення....42
7.3 Алгоритм обробки сигналу....43
8 Тенденції розвитку....44
8.1 Контекстна схема обслуговування переривання....44
8.2 Вісім схем потоків даних та специфікацій....45
9 Контрольні завдання та запитання....46
Висновки....47
ПЕРЕЛІК ЛІТЕРАТУРИ....48

Вступ

Навчальна дисципліна «Системи для малих та мобільних платформ» читається в восьмому семестрі студентам четвертого курсу, які готуються за напрямом 6.050103 «Програмна інженерія».

Предметом дисципліни є побудова та функціонування систем для малих та мобільних платформ.

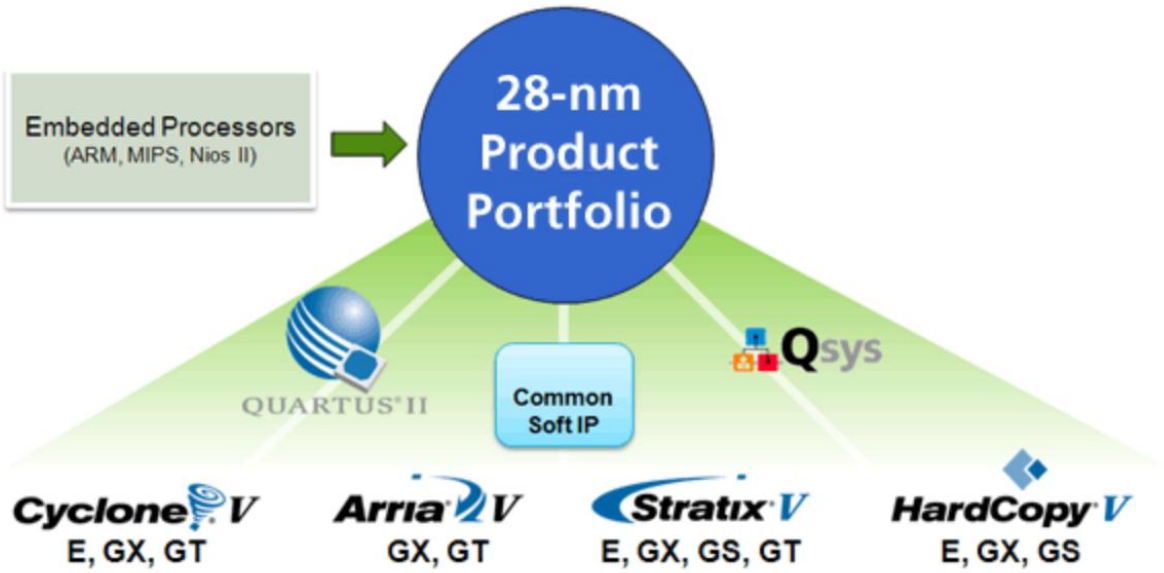
Мета дисципліни полягає у формуванні знань і умінь розробки і експлуатації програмних засобів систем для малих та мобільних платформ на рівні, достатньому для практичної діяльності за спеціальністю.

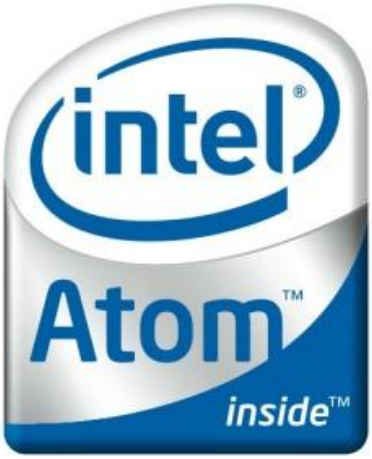
Тематичний модуль 1.
Архітектура мікропроцесора ARM

Лекція 1. Ввідна

Лекція 2. Модель мікропроцесора ARM

Лекція 3. Мікропроцесор V3





1 ВВІДНА

Загальні відомості щодо ARM.

ARM є сімейство мікропроцесорів архітектури RISC, які поділяють ті ж принципи дизайну і загального набору інструкцій.

ARM сам не виробляє процесори, але надає ліцензії, іншим виробникам інтегрувати їх у свою власну систему.

Ядро ARM (CORE) це частина системи-на-кристалі.

Крім GPS, ядро ARM широко використовується в мобільних телефонах, портативних помічниках, і багатьох інших портативних пристроях побутової електроніки.

Залежно від застосування процесорів ARM доступні, наприклад: різні розміри кеша; шини різної ширини; різні тактові частоти.

Різні версії використовують різні архітектури, наприклад, ARM 7: VON NEUMANN; ARM 9: HARVARD. Мова програмування не залежать від базової архітектури.

2 МОДЕЛЬ МІКРОПРОЦЕСОРА ARM

Agenda:

- 1) архітектура ARM;
- 2) набір інструкцій.

2.1 Архітектура ARM

ARM асемблер: мова асемблеру відображає набір команд (майже один до одного); одна інструкція на рядок; етикетки надають імена адрес (як правило, в першому стовпці); інструкції часто починаються в наступній колонці; колонки запускають розрахунок до кінця рядка коду мови асемблера.

Архітектура Von Neumann складається з центрального процесора і єдиної пам'яті, пам'ять зберігає інструкції і дані.

Пам'ять зберігає дані, інструкції.

Центральний процесор CPU отримує інструкції з пам'яті. Окремий процесор і пам'ять відрізняє програмований комп'ютер.

Регістри процесора: лічильник програми PC, реєстр команд IR, регістри загального призначення і т.д.

Архітектура HARVARD.

Складається з центрального процесора і двох окремих модулів пам'яті. В оригінальній Гарвардській архітектурі, один модуль пам'яті зберігає інструкції та інший – дані.

Порівняння VON NEUMANN AND HARVARD:

- гарвардська архітектура дозволяє одночасно готувати два модуля пам'яті;
- гарвардська архітектура не можете використовувати код, який сам модифікується;
- більшість DSP використовують Гарвардську архітектуру для роботи з потоками даних: більшої пропускної здатності пам'яті; більш передбачуваної пропускної здатності;
- додаткове обладнання, оскільки необхідні дві шини адреси і даних;
- модель для програмування.

Регістри доступні в режимі користувача USER MODE.

ARM процесор має 17 активних регістрів в режимі користувача 16 регістрів даних (r0-R15) і 1 регістр стану процесора

Регістри R13-R15 мають спеціальні завдання:

- R13 є покажчик стека SP;
- R14 є регістром зв'язку LR ;
- R15 є програмний лічильник PC.

Регістр стану поточної програми[1] CPSR призначений для спостереження і управління внутрішніми операціями.

ARM використовує 32-бітові адреси.

Слово має 32 біта (4 байти) в довжину.

Адреса посилається на байт (а не слова).

ARM процесор може бути налаштований на використання моделі пам'яті LITTLE-ENDIAN або BIG-ENDIAN. LITTLE-ENDIAN : молодший байт знаходиться в молодших бітах слова. BIG-ENDIAN : молодший байт знаходиться у старших бітах слова.

2.2 Набір інструкцій

Переміщення даних.

ARM має архітектуру завантаження-збереження[2].

Дані повинні бути завантажені в регістри, перш ніж вони можуть бути оброблені ALU.

Дані переміщуються між регістрами шляхом застосування інструкції

MOV r1, r2 ; r1 = r2

MOV r3, #1 ; r3 = 1

Дані переміщуються між модулями за допомогою інструкцій LOAD та STORE.

Одномісний трансфер Пам’ять-Регістр.

Дані повинні бути завантажені в регістри, перш ніж вони можуть бути оброблені ALU.

Інструкції LOAD та STORE можна комбінувати з різними режими адресації.

Базова інструкція LOAD це LDR (завантаження слова в регістр), але є варіації, які застосовують байт[3], півслова[4] і байти із знаком[5].

Базова інструкція STORE це STR (зберегти слово з регістру), варіації STRB і STRH.

Приклад завантаження.

- Before:

r0 = 0x00000000

r1 = 0x00070000

[1] Current Program Status Register, CPSR
[2] Load-Store architecture
[3] byte, LDRB
[4] Half word, LDRH
[5] signed bytes, LDRSB

mem32[0x00070000] = 0x00000005

LDR r0, [r1]

- After:

r0 = 0x00000005

r1 = 0x00070000

Операції зі стеком.

ARM архітектура використовує інструкції LOAD-STORE MULTIPLE INSTRUCTIONS для завантаження та вивантаження даних в стеку. Тут ви повинні вирішити, який стек за збільшенням (A, ASCENDING) або за спаданням (D, DESCENDING) і повний (F, FULL) або порожній (E, EMPTY) стек. Повний стек: Покажчик стека вказує на останній використовуваний адрес. Порожній Стек: Покажчик стека вказує на першу порожню адресу STMFA SP!, {r5,r7} штовхає регістри R5 і R7 за зростанням стека і вказує після інструкції на пам'ять, де зберігається R7. STMFA SP!, {r5,r7} еквівалентна STMIB r13!, {r5,r7}.

Інструкції для обробки даних.

- Інструкції обробки даних маніпулюють даними в регістрах (Move, Arithmetic, Logical, Comparison, Multiply).
- Якщо суфікс S використовується CPSR прапори N, Z, C, V оновлюється.
- ADD r1, r2, r3 does not update CPSR.
- ADDS r1, r2, r3 updates the CPSR.

Інструкції обробки даних та CPSR.

- MOVS r1, #1 ⇒ NZCV = 0000
- MOVS r2, #-1⇒ NZCV = 1000
- ADD r3, r2, r1⇒ NZCV = 1000
- ADDS r3, r2, r1⇒ NZCV = 0110

Інструкції обробки даних.

- Move: MOV, MVN
- Arithmetic: ADD, ADC, SUB, SBC, RSB,RSC
- Logical: AND, ORR, EOR, BIC
- Comparaison: CMP, CMN, TST, TEQ
- Multiply: MUL, MLA, SMLAL, SMULL, UMULL, SMLAL, UMLAL

Формати інструкцій обробки даних.

- Basic format
- SUB r3, r2, r1 ; r3 = r2 – r1
- Immediate Operand
- SUB r3, r2, #3 ; r3 = r2 - 3
- Preprocessing (Barrel-Shifter)
- SUB r3, r2, r1, LSL #1 ; r3 = r2 – (r1 * 2)

Схема циклічного зсуву.

Схема циклічного зсуву дозволяє введення зсуву до вступу даних в ALU.

Shift Operations: LSL, LSR, ASR, ROR, RRX.

Example:

MOV r3, r4, LSL #3; r3 = 8 * r4

Інструкції розгалуження.

- Інструкції розгалуження використовується для зміни потоку виконання (if-then-else, for-loop, while-loop)
- Відділення Інструкції: B, BL, BX, BLX
- Інструкції розгалуження часто використовуються з умовами (EQ, NE, CS, CC, MI, PL, VS, HI, LS, GE, LT, GT, LE)

BEQ label ; Branch to label, if Z = 1

- Адресний ярлик зберігається в інструкції, як PC-відносний зсув і повинно бути в межах 32 Мб від інструкції розгалуження

Subroutines.

- The BL (Branch and Link) instruction can be used for subroutines, since it writes the return address to the link register
- BL subroutine
- ...
- subroutine
- ... ; code for subroutine
- MOV pc, lr ; return by moving lr to pc

Умовне виконання.

- Not only branch instructions can be used with conditions
- ADDEQ r4, r5, r6 is only executed if Z = 1
- Conditional Execution helps to design shorter programs that do not use so much memory

Висновки до лекції 2.

- ARM є сім'я ядер мікропроцесорів.
- Архітектура завантаження / збереження.
- Більшість інструкцій RISC, працюють у єдиному технологічному циклі.
- Деякі операції для багатьох регістрів виконуються довше.
- Всі команди можуть бути виконані умовно.

3 МІКРОПРОЦЕСОР V3

Agenda:

1) блок схема;

2) банк регістрів;

3) виключення .

3.1 Блок схема

ARM60 – це 32-розрядний RISC мікропроцесор загального призначення з низьким енергозабезпеченням. По суті це ядро ARM6, упаковане в плоский пластиковий корпус, який має 100 контактів. Його простий, витончений і повністю статичний проект особливо відповідає застосуванням, що чутливі до вартості і енергозабезпечення.

ARM60 ідеально задовольняє застосування, що вимагають характеристики RISC мікропроцесору, невеликого за розміром та енергозабезпеченням ефективного процесору. Типові застосування включають:

телекомунікації, наприклад, процесор мобільного телефону стандарту GSM;

передавання даних, наприклад конверсія протоколу;

портативні комп’ютери, наприклад комп'ютер типу PALM;

портативні інструменти, наприклад ручний прилад збору даних;

автомобільні застосування, наприклад модуль управління двигуном;

дешевий контролер для споживчих мультимедійних застосувань.

Характеристики мікропроцесора: 32 розрядний RISC процесора; 32 розрядна шина даних; 32 розрядна адресна шина; режими роботи – Big&Little Endian; високо продуктивна RISC архітектура, гарантовано 21 мільйон команд в секунду @ 30MHz (30 мільйонів команд в секунду на піку) @ 5V; низьке енергетичне споживання 1.5mA/MHz @ 5V, яке забезпечене технологією1мкм CMOS; повністю статична дія, що є ідеалом для чутливих до енергетичного споживання застосувань; швидка відповідь на переривання для систем, які працюють в реальному часі; система підтримки віртуальної пам'яті; чудова підтримка мови високого рівня; система простих, але могутніх команд; просте тестування при виробництві за стандартом IEEE 1149.1 (JTAG); ARM60 - частина сім'ї (Advanced RISC Machines – ARM) 32-розрядних мікропроцесорів загального призначення, які забезпечують дуже низьке енергетичне споживання і ціну для мікросхем високої продуктивності.

Архітектура заснована на принципах зменшеного набору команд комп'ютера[6], ця система команд і механізм декодування набагато простіші в порівнянні з мікро програмованою комплексною системою команд[7]. Це приводить до високої командної продуктивності і вражаючої реакції на переривання в реальному часі від малої і рентабельної мікросхеми.

Система команд включає одинадцять основних командних типів:

двоє з них роблять можливим використання сформованих на одному кристалі схем арифметичного логічного устрою (ALU), регістра і устрою множення для виконання швидкохідних дій над даними в банку з 31 регістра, кожен з яких має шину даних 32 розряди уширшки;

три класи інструкцій управляють передачею даних між пам'яттю і регістрами, один оптимальний для гнучкості адресації, інший для швидкого перемикання контексту і третини для обміну даних;

три інструкції управляють потоком і рівнем привілею виконання;

Три типи присвячені контролю зовнішніх співпроцесорів, які дозволяють нарощувати функціональність системи команд, яка може бути розширена навісною мікросхемою у відкритий і загальноприйнятий спосіб.

Система команд ARM – хороша платформа для компіляторів багатьох різних мов високого рівня. Де для критичних кодових сегментів, програмування коду асемблеру також просте, на відміну від деяких RISC процесорів, які залежать від складної технології компілятора, щоб управляти ускладненою взаємозалежністю інструкцій. Конвеєрна обробка використовується таким чином, що всі частини обробки і систем пам'яті можуть діяти безперервно. Звичайно, тоді як одна інструкція виконується, її наступниця розшифрована, а третя інструкція є притягнутою із пам'яті.

Інтерфейс пам'яті проектувався так, щоб дозволити продуктивному потенціалу бути таким без несення високих витрат на систему пам'яті. Критичні до швидкості управляючі сигнали прокладені конвеєром, щоб дозволити системі управляти функціями, які здійснюються в стандартній малопотужній логіці, і ці управляючі сигнали полегшують експлуатацію методів швидкого доступу, запропонованих динамічними RAM промислового стандарту.

ARM60 має 32 розрядну адресну шину. Всі процесори ARM застосовують таку ж систему команд, і ARM60 може бути сформовано, щоб використовувати 26 розрядну адресну шину для сумісності з попередніми процесорами.

[6] Reduced Instruction Set Computer, RISC
[7] Complex Instruction Set Computers, CISC

ARM60 - повністю статичне CMOS виконання ARM, яка дозволяє тактовому генератору зупинятися в будь-якій частині циклу із наднизьким залишковим енергетичним споживанням без жодної втрати стану.

Мікропроцесор ARM60 підтримує різноманітність операційних конфігурацій. Деякі регульовані входами і відомі, як апаратні конфігурації. Інші управляються програмним забезпеченням і ці відомі, як режими роботи.

Процесор ARM60 забезпечує 4 апаратні конфігурації, які можливо змінити, тоді як процесор працює.

Вхід BIGEND встановлює, чи розглядає ARM60 слова в пам'яті у форматі BIG ENDIAN або LITTLE ENDIAN. Пам'ять розглядається як лінійна колекція байтів, пронумерованих вгору з нуля. Байти 0-3 тримають перше збережене слово, байти 4-7 наступне і так далі. У схемі LITTLE ENDIAN найменшому пронумерованому байту одного слова належить бути молодшим істотним байтом слова, а найбільшому пронумерованому байту - старшим істотним. В цій схемі байт 0 системи пам'яті потрібно з'єднати з лініями 7..0 шини даних d[7:0]. У схемі BIG ENDIAN старший істотний байт слова запам'ятовується в низько пронумерованому байті, а молодший істотний байт запам'ятовується у вище всього пронумерованому байті. Таким чином байт 0 системи пам'яті потрібно з'єднати з лініями 31..24 шини даних d[31:24].

Вхід LATEABT встановлює поведінку процесору, коли відбувається виключення даних. Це впливає тільки на поведінку регістрових інструкцій завантаження/збереження і обговорюється більш повно далі в пункті, де розглядається система команд.

Інші два входи, PROG32 і DATA32 використовуються для зворотної сумісності з ранніми процесорами ARM. Для сумісності ARM звичайно потрібно звичайно бути встановити 1. Ця конфігурація розширює адресний простір до 32 розрядів, вводить головні зміни в моделі програмування як описано нижче і забезпечує підтримку роботи, існуючих 26 розрядних програм в 32 розрядному навколишньому середовищі. Цей метод рекомендований для сумісності з майбутніми процесорами ARM і всі нові коди потрібно писати, з використанням тільки 32 розрядного операційного методу. Оскільки оригінальна система команд ARM була пристосована до 32 розрядного застосування є певні додаткові обмеження, про які програмісти повинні бути обізнані. Це вказано по тексту словами повинен і не повинен. Посилання потрібно також зробити до документа постачальника APPLICATION NOTES.

Вибір режиму роботи.

Мікропроцесор ARM60 має 32 розрядну шину даних і 32 розрядну адресну шину. Процесор підтримує наступні типи даних: байти (8 біт) і слова

(32 біта), де слова повинні бути вирівняні до границь чотирьох байтів. Інструкція це точно одне слово, а дії з даними (наприклад додавання ADD) виконуються тільки над словами. Дії завантаження і збереження можуть передати або байти, або слова.

ARM60 підтримує шість режимів роботи:

(1) призначений для користувача режим[8] USR – нормальний режим виконання програми;

(2) режим швидкого переривання[9] FIQ – розроблений, щоб підтримувати процеси перенесення даних в каналі інтерфейсу мікропроцесору;

(3) режим звичайного переривання[10] IRQ – застосовується для обробки переривання загального призначення;

(4) режим супервізора[11] SVC звернення до операційної системи – захищений режим роботи системи;

(5) режим виключення[12] ABT – включається після переривання вибору з попередженням даних або команд;

(6) невизначений режим[13] UND – включається, коли невизначена інструкція виконується.

Зміни режимів можливі під управлінням програмного забезпечення, або вони можливо принесені зовнішніми перериваннями або обробкою виключення. Більшість прикладних програм виконуватимуться в призначеному для користувача режимі USR. Інші режими, відомі, як привілейовані режими, увійдуть до службових переривань або виключень або, щоб звернутися до захищених ресурсів.

3.2 Банк регістрів

Процесор має загалом 37 регістрів, виконаних у вигляді тридцяти одного 32-розрядного основного регістру і 6 регістрів статусу. В любий момент часу тільки 16 основних регістрів (R0 до R15) і один або два регістри статусу видимі для програміста. Видимість регістрів залежить від режиму роботи процесора, інші регістри (регістри банку) підключаються щоб підтримувати наступні режими роботи процесора: IRQ, FIQ, SUPERVISOR, ABORT AND UNDEFINED MODE.

[8] user mode – USR
[9] Fast Interrupt reQuest – FIQ
[10] Irq mode – IRQ
[11] Supervisor mode – SVC
[12] Abort mode – ABT
[13] Undefined mode – UND

У всіх режимах 16 регістрів R0...R15, безпосередньо доступні. Всі регістри окрім R15 – загального призначення і можуть використовуватися, щоб зберігати дані або адреси. Зареєструйте володіння Регістр R15 зберігає лічильник команд[14] PC. Коли регістр R15 читається, біти [1:0] нульові і біти [31:2] містять PC. Сімнадцятий регістр статусу поточної програми (Current Program Status Register – CPSR) також доступний. Він містить прапори коду завершення і біти поточного режиму і, може розглядатися як розширення PC.

R14 використовується як регістр зв'язку підпрограми і приймає копію R15, коли виконується інструкція розгалуження і зв'язку програми. Він може розглядатися як регістр загального призначення у всі інші часи. Регістри R14_svc, R14_irq, R14_fiq R14_abt і R14_und використовуються так само, щоб записувати повертанні значення R15, коли переривання і виключення виникніть, або, коли виконується інструкція розгалуження і зв'язку програми в межах процедури обробки переривання або виключення.

В режимі FIQ застосовується банк, який має сім регістрів, що проектуються на R8-14 (R8_fiq-R14_fiq). Багатьом програмам FIQ не потрібно зберігати будь-які регістри.

Призначений для користувача режим, режими Irq, супервізора виключення і невизначений застосовують кожен по два регістри в банку, що проектуються на R13 і R14. Два регістри банку, дозволяють для кожного режиму мати приватні покажчик вершини стека і регістр зв'язку.

Програми режимів супервізора, IRQ , виключення і невизначений, які вимагають більш ніж ці два регістри банку, як очікується, зберігають деякі або всі регістри (R0...R12) у відповідних стеках. Вони тоді вільні, щоб використовувати ці регістри, які вони відновлять перед поверненням до програми виклику.

Крім того є також п'ять регістрів статусу збереженої програми[15] SPSR, які завантажують з CPSR, коли виключення відбувається. Є один SPSR для кожного привілейованого режиму. Тому CPSR попереднього режиму може бути легко відновлений, коли з поточного привілейованого режиму виходять.

Біти N, Z, C та V це прапори коду завершення. Прапори коду завершення в CPSR, можуть мінятися в результаті арифметичних і логічних операцій в процесорі і, можуть перевірятися усіма інструкціями, щоб визначити, чи винна інструкція виконуватися.

Біти I та F блокують переривання. Біт I блокує переривання IRQ, коли він встановлений, і біт F блокує переривання FIQ, коли він встановлений. Біти M0,

[14] Program Counter – PC
[15] Saved Program Status Registers — SPSR

M1, M2, M3 і M4 (m[4:0]) є бітами режиму і вони визначають режим, в якому процесор діє. Інтерпретацію бітів режиму показано в табл.3.1.

Таблиця 3.1 – Біти режиму

M[4:0]	Режим	Набір доступних регістрів	
10000	User	PC, R14..R0	CPSR
10001	FIQ	PC, R14_fiq..R8_fiq, R7..R0	CPSR, SPSR_fiq
10010	IRQ	PC, R14_irq..R13_irq, R12..R0	CPSR, SPSR_irq
10011	Supervisor	PC, R14_svc..R13_svc, R12..R0	CPSR, SPSR_svc
10111	Abort	PC, R14_abt..R13_abt, R12..R0	CPSR, SPSR_abt
11011	Undefined	PC, R14_und..R13_und, R12..R0	CPSR, SPSR_und

Не всі комбінації біт режиму визначають дійсний режим роботи процесора. Тільки ці явно описані набори повинні використовуватися.

Молодші 28 біт регістрів статусу програм[16] PSR, в тому числі сполучення I, F та M[4:0], відомі колективно, як службові біти. Службові біти змінятся, коли виключення виникає і крім того вони можуть управлятися програмним забезпеченням, коли процесор знаходиться в привілейованому режимі. Невикористані біти в регістрах PSR резервуються і їх стан повинен бути збереженим, коли змінюються прапори або службові біти. Програми не повинні залежати від певних значень резервованих бітів, коли перевіряється статус PSR, з тих пір, як вони, можливо, будуть читатися як один або нуль в майбутніх процесорах.

3.3 Виключення

Виключення виникають кожного разу, коли є потреба припинити нормальне виконання програми, таким чином, що(наприклад) процесор може відхилитися, щоб обробити переривання від зовнішнього пристрою. Стан процесора тільки до обробки виключення повинен бути збережений таким чином, що оригінальна програма може бути відновлена, коли процедура виключення завершилася. Багато виключень, можливо, виникає в той же час.

[16] Program Status Registers — PSR

ARM60 обробляє виключення, використовуючи регістри банків, щоб зберегти стан. Старий вміст PC і CPSR копіюються у відповідний R14 і SPSR, а PC і біти режиму в бітах CPSR приймають значення, які залежать від виключення. Переривання блокує прапори встановлені, де потрібно для запобігання інших некерованих вкладень виключень. У разі обробки декількох програм переривають, регістри R14 і SPSR потрібно зберегти у стек в оперативній пам'яті перед повторним наданням можливості переривання, переміщаючи регістр SPSR в та із стеку, важливо, щоб перенести повне 32-розрядне значення, а не тільки прапор або контрольні поля. Коли багатократні виключення виникають одночасно, встановлений пріоритет визначає порядок, в якому вони управляються. Пріоритети внесені в список пізніше в цьому пункті.

Виключення FIQ виробляється зовні, коли вхідний сигнал nFIQ набуває низький рівень. Цей вхід може прийняти асинхронні переходи, і затриматися на один цикл годинника для синхронізації перед тим, як це зможе впливати на потік виконання програми процесором. Це спроектовано, щоб підтримувати процес переміщення даних або канальні процеси, мається достатньо приватних регістрів, щоб виключити потребу в регістрів для збереження в таких застосуваннях (таким чином зменшується кількість перемикань контексту). Виключення FIQ можливо блокувати встановлюючи F прапор в CPSR (але це не можливо з Призначеного для користувача режиму). Якщо прапор F відсутній, ARM60 перевіряє НИЗЬКИЙ рівень на виході синхронізатора FIQ в кінці кожної інструкції. Коли FIQ виявлений, ARM60 виконує наступне: зберігає адресу наступної інструкції плюс 4 в R14_fiq, зберігає CPSR в SPSR_fiq; задає m[4:0]=10001 (режим FIQ) і встановлює F та I біти в CPSR; встановлює PC таким, щоб забрати наступну інструкцію за адресою 0x1C. Щоб повернутися нормально з FIQ, використовуйте команду SUBS PC, R14_fiq,#4, яка відновить як PC з R14, так і CPSR з SPSR_fiq) і дозволить виконання перерваного коду. R14_fiq - символ для регістра R14, який при використанні потрібно оголосити в прикладній програмі споживачів.

Виключення IRQ – нормальне переривання, викликане НИЗЬКИМ рівнем на nIRQ вході. Воно має нижчий пріоритет, ніж FIQ, і маскується, коли послідовність FIQ входить. Його результат можливо маскувати в будь-який час, встановлюючи I біт CPSR (але це не можливо з Призначеного для користувача режиму). Якщо I сигналізує чисто, ARM60 перевіряє НИЗЬКИЙ рівень на виході синхронізатора IRQ в кінці кожної інструкції. Коли IRQ виявлений, ARM60 виконує наступне: зберігає адресу наступної інструкції плюс 4 в R14_irq, зберігає CPSR в SPSR_irq; задає m[4:0]=10010 (режим IRQ) і встановлює I біт CPSR; змушує PC забрати наступну інструкцію за адресою

0x18. Щоб повернутися нормально з Irq, використовуйте команду SUBS PC,R14_irq,#4 , яка відновлять як PC, так і CPSR і дозволить виконання перерваного коду. R14_fiq - символ для регістра R14 і, якщо його використано потрібно це оголосити в прикладній програмі споживачів.

Про режим ABORT може бути повідомлено через зовнішній вхід ABORT. Сигнал ABORT указує, що поточний доступ до пам'яті не може завершитися. Наприклад, у фактичній системі пам'яті дані, відповідні поточній адресі, можливо, були переміщені з пам’яті на диск, і значна діяльність процесора, можливо, потрібна щоб повернути дані перед тим, як доступ зможе виконуватися успішно. ARM60 перевіряє ABORT впродовж циклів звернення до пам'яті. Успішно перерваний ARM60 відповість в одному з двох випадків.

Перший випадок. Якщо переривання відбувається протягом команди вибору з попередженням[17], команда відмічається, як недійсна, але виключення не відбувається негайно. Якщо команда не виконується, наприклад в результаті виклику підпрограми під час роботи конвеєра, переривання не відбуватиметься. Переривання матиме місце, якщо команда досягає початку конвеєра.

Другий випадок. Якщо переривання відбувається протягом доступу до даних[18], дія залежить від типу команди.

Команди (LDR, STR) переміщення простих даних перериваються неначе команда не виконувалася, якщо процесор сформований для раннього переривання. Коли сформовано пізнє переривання, ці команди можуть записати назад змінені базові регістри і обробник переривання повинен бути обізнаний про це.

Команда (SWP) перевантаження переривається так наче воно не виконалося, хоч зовні доступ для читання, можливо, має місце.

Команди (LDM, STM), які переміщають блокові дані, завершують, але якщо відкладений запис встановлено, база модифікується.

Якщо команда звичайно записала б поверх бази з даними (тобто LDM з базою в списку переміщень), цей перезапис виконується превентивно. Весь перезапис регістрів виконується превентивно після того, як переривання вказано, що означає зокрема, що R15 (який завжди останнім переміщається) збережений в перерваній команді LDM.

Коли відбувається переривання або вибору команди, або даних, ARM60 виконує наступне: зберігає адресу перерваної команди плюс 4 (prefetch aborts) або 8 (data aborts) в регістр R14_abt, зберігає CPSR в SPSR_abt; заряджає m[4:0]=10111 (ABORT) і встановлює I біт CPSR; змушує PC отримати

[17] a Prefetch Abort
[18] a Data Abort

наступну команду або за адресою 0x0C (prefetch abort), або звернутися до 0x10 (data abort).

Щоб повернутися після фіксації причини для переривають, використовуйте команду SUBS PC,R14_abt,#4 (for a prefetch abort) або SUBS PC,R14_abt,#8 (for a data abort). Це відновить як PC, так і CPSR і повторить перервану команду. R14_fiq - символ для регістра R14 і, якщо його використано це потрібно бути оголошеним в прикладній програмі користувачів. Механізм переривання дозволяє здійснювати віртуальну систему пам'яті з нумерацією сторінки на вимогу (demand paged virtual memory system), коли відповідне програмне забезпечення управління пам'яті доступне. Процесор дозволяють генерувати довільні адреси, і, коли дані за адресами недійсні сигнали MMU переривають. Трапи процесора в системному програмному забезпеченні, яке повинне розв'язати причину переривань, роблять дані, які запросили, доступними, і повторюють перервану команду. Прикладній програмі не потрібне знання кількості пам'яті, доступної для неї, ні її стан при виникненні будь-якого переривання.

Команда програмного переривання[19] SWI використовується для отримання режиму Supervisor, звичайно, щоб запросити специфічну функцію супервізора. Коли команда SWI виконується, ARM60 робить наступне: зберігає адресу команди SWI плюс 4 в R14_svc, зберігає CPSR в SPSR_svc; заряджає m[4:0]=10011 (Supervisor) і встановлює прапор I в CPSR; вимушує PC отримати наступну команду за адресою 0x08.

Щоб повернутися з SWI, використовуйте MOVS PC,R14_svc. Це відновить PC і CPSR і поверне до команди, наступної за SWI.

Коли до ARM60 надходить команда, яку не можна обробити, вона пропонується будь-якому співпроцесору, які, можливо, присутні. Якщо співпроцесор може виконувати цю команду, але зайнятий в цей час, ARM60 чекатиме, поки співпроцесор не буде готовий або, поки переривання не відбудеться. Якщо ніякий співпроцесор не може обробити команду, тоді ARM60 візьме невизначений командний трап. Трап може використовуватися для програмної емуляції співпроцесора в системі, яка не має технічних засобів співпроцесора, або для продовження системи команд загального призначення програмною емуляцією. Коли ARM60 прийняв невизначений командний трап, він виконує наступне: зберігає адресу Undefined або команди співпроцесора плюс 4 в R14_und, зберігає CPSR в SPSR_und; завантажує m[4:0]=11011 (Undefined) і встановлює I біт CPSR; вимушує PC отримати наступну команду за адресою 0x04.

[19] software interrupt instruction – SWI

Щоб повернутися після емуляції невдалої команди, використовуйте MOVS PC,R14_und. Це відновить CPSR і поверне до команди, наступної за невизначеною командою.

В табл.3.2 наведений звід векторів виключень.

Таблиця 3.2 – Звід векторів виключень

Адреса	Виключення	Режим на вході
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	-- reserved --	
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

За наведеними адресами звичайно зберігається команда переходу, яка вказує на доречну підпрограму.

Підпрограма FIQ, можливо, постійно знаходиться за адресою 0x1C, і таким чином уникає потреби в команді переходу.

Коли численні виняткові стани з'являються одночасно, фіксована система пріоритету визначає порядок, в якому вони будуть оброблені:

(1) Reset (найвищий пріоритет);

(2) Data abort;

(3) FIQ;

(4) IRQ;

(5) Prefetch abort;

(6) Undefined Instruction, Software interrupt (найнижчий пріоритет).

Не всі виняткові стани можуть відбуватися відразу. Виключення Undefined Instruction, Software interrupt взаємовиключні, кожне з яких відповідає специфічним декодуванням поточної команди.

Якщо виключення data abort відбувається в той же час як FIQ, і FIQ дозволені (тобто Вперед прапорець F в CPSR чистий), ARM60 запровадить

обробника виключення data abort, а потім негайно виконає вектора FIQ. Нормальне повернення від виключення FIQ викличе обробника виключення data abort та дозволить. Розміщення виключення data abort у вищому пріоритеті, ніж FIQ необхідне, щоб гарантувати, що помилка переміщення не буде псувати її виявлення; час для входу до цього виняткового стану потрібно додати до часу очікування обчислень FIQ для гіршого випадку.

Найгірший випадок часу очікування для виключення FIQ, припускаючи, що воно активне, складається з щонайдовшого часу, щоб пройти через синхронізатор (Tsyncmax), плюс час для завершення щонайдовшої команди (Tldm, щонайдовша команда – LDM, яка завантажує всі регістри, зокрема PC), плюс час для входу до режиму data abort (Texc), плюс час для входу до FIQ (Tfiq). Наприкінці цього часу ARM60 виконуватиме команду в 0x1C. Час Tsyncmax складає 3 цикли процесора, Tldm складає 20 циклів, Texc складає 3 цикли, і Tfiq складає 2 цикли. Повний час складає тому 28 циклів процесора. Це тільки понад 1.4 мікросекунд в системі, яка використовує безперервний годинник процесора з частотою 20МГц.

Підрахунок максимального часу очікування виключення IRQ подібне, але необхідно врахувати факт, що FIQ має вищий пріоритет і міг би затримати вхід в підпрограму оброблювання IRQ на довільну довжини часу. Мінімальний час очікування для FIQ або IRQ складається з найкоротшого часу синхронізатору (Tsyncmin) плюс Tfiq. Це 4 циклу процесора. Щоб скоротити час очікування переривання, час Tldm може бути зменшений, використовуючи параметр в компіляторі, який розбиває команди LDM таким чином, що вона буде тільки завантажувати або зберігати в визначене користувачем число (між 3 і 16) регістрів в будь-який час. Якщо цей параметр використовується, то команда MUL або MLA може потенційно стати самою довгою аж 17 циклів, залежно від даних, що обробляються.

Коли сигнал nRESET приймає низький рівень, ARM60 відмовляється від команди, що виконується, а потім продовжує вибірку команд за адресами, які збільшуються на один.

Коли nRESET приймає високий рівень знову, ARM60 робить наступне:

Накладення записів R14_svc і SPSR_svc, копіюючи поточні значення PC і CPSR в них; значення збереженого PC і CPSR не визначені; завантажує m[4:0]=10011 (Supervisor) і встановлює біти I та F в CPSR; змушує PC отримати наступну команду за адресою 0x00.

Всі цикли перенесення в пам'яті можуть бути розміщені в одній з чотирьох категорій:

(1) Непослідовний цикл. ARM60 запросив перенесення до або з пам’яті за адресою, яка не зв’язана з адресою, що використовувалася в попередньому циклі.

(2) Послідовний цикл. ARM60 запросив перенесення до або з пам’яті за адресою, яка є або такою ж як адреса в попередньому циклі, або, плюс одне слово після попередньої адреси.

(3) Внутрішній цикл. ARM60 не вимагає перенесення, тому що він виконує внутрішню функцію і жодна корисна наперед вибірка не може виконуватися в той же час.

(4) Перенесення регістра співпроцесора. ARM60 бажає використовувати шину даних, щоб зв'язатися із співпроцесором, але не вимагає ніякої дії з системою пам'яті.

Ці чотири класи розрізняють звертання до системи пам'яті інспекцією ліній управління nMREQ і SEQ, як це наведено в табл. 3.3.

Таблиця 3.3 – Типи звертань до системи пам’яті

nMREQ	SEQ	Тип циклу
0	0	Non-sequential cycle (N-cycle)
0	1	Sequential cycle (S-cycle)
1	0	Internal cycle (I-cycle)
1	1	Coprocessor register transfer (C-cycle)

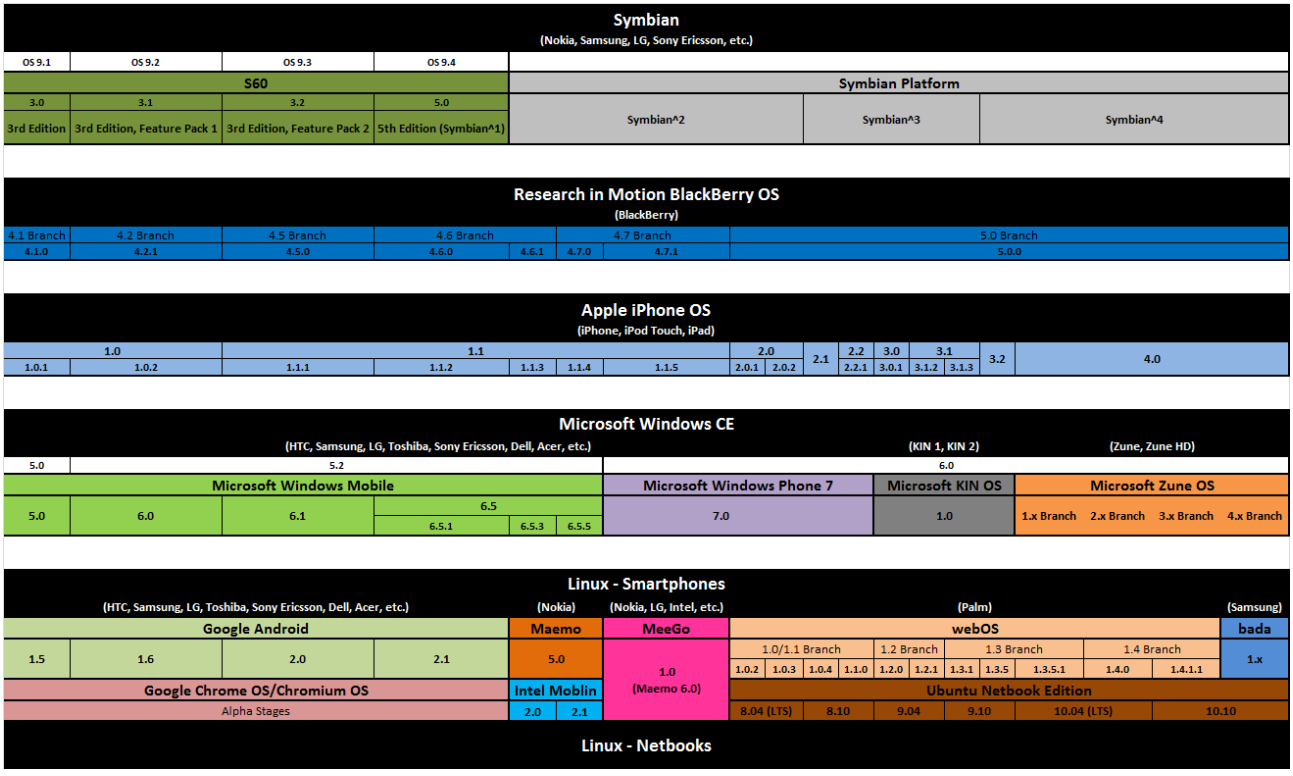
Ці контрольні лінії генеруються протягом фази 1 циклу перед циклом, чиї характеристики вони передбачають, і ця конвеєрна обробка контрольної інформації надає системі пам'яті достатній час, щоб вирішити, може або не може вона використовувати доступ в режимі сторінки.

Тематичний модуль 2. Операційна система

Лекція 4. Загальна характеристика

Лекція 6. Обробка переривань

Лекція 7. Призупинення задач



4 ЗАГАЛЬНА ХАРАКТЕРИСТИКА

Agenda:

1) дві форми операційних систем;
2) контекстна схема операційної системи;
3) процес ініціалізації задач.

4.1 Дві форми операційних систем

Взагалі кажучи операційна система може мати дві форми: паралельну та послідовну. Кожна з цих форм має свої переваги та недоліки в залежності від застосування в якому вона використовується. Дійсно, перевага однієї з систем має тенденцію бути недоліком іншої.

Послідовна операційна система.

Послідовна операційна система заснована на розподілі часу мікропроцесора на послідовність інтервалів. На любому інтервалі явно визначено які можуть виконуватися або звертатися до мікропроцесора. Програмні операції які виконуються на інтервалі не повинні вимагати більше часу чим передбачено системою для нормального завершення задачі. Послідовна операційна система застосовується в системах, в яких не визначені вимоги до операцій реального часу та в яких може бути точно передбачено всі операції та час їх здійснення. Концепція послідовної операційної системи добре задовольняє програмні функції, які виконуються послідовно в заданому порядку в визначений заздалегідь час. Як тільки програмна задача активована, вона утримує мікропроцесор до свого завершення, а операційній системи нема потреби зберігати контекст програмної задачі та мікропроцесора. Виключення можливе коли в системі, коли нормальна програмна операція зазнає дію фізичного переривання, в цьому випадку необхідно зберегти контекст, виконати обробку переривання та відновити контекст. Внаслідок згаданого вище послідовна операційна система застосовує мінімум системних ресурсів: простір пам’яті та мікропроцесор, який завантажується зверху донизу. Основний недолік системи – це її нездатність працювати в наступних системах: реального часу; де необхідно виконувати множину задач з широким діапазоном часу виконання та де задачі мають різні пріоритети. Послідовна операційна система стабільна за своєю природою, але погано пристосована до додавання нових програмних задач, які не передбачались оригінальним проектом. Доступ мікропроцесора до любої програмної задачі можна заборонити в випадку

необхідності виконати новий розподіл програмних операцій на часових інтервалах.

Паралельна операційна система.

Паралельна операційна система призначена для підтримки набору програмних задач, які виконуються паралельним методом. В цій системі кожній задачі надається час мікропроцесора, а коли виникає конфлікт задач виконується задача з найбільшим пріоритетом. Паралельна операційна система є системою реального часу. Для реалізації концепції паралельного перемикання задач операційна система повинна зберігати контекст програмної задачі та мікропроцесора, а потім поновлювати їх перед тим як поновити виконання задачі. У зв'язку з цим паралельна операційна система повинна мати достатній об'єм пам'яті для збереження контексту усіх задач. Залежно від структури програмних задач та проекту це може вимагати значну частину загальної пам'яті прикладної програми. Паралельні операційні системи мають велику гнучкість для модифікації існуючих та доданню нових задач в програмний проект. Єдина вимога з завантаження мікропроцесора – це достатність часу на обробку для завершення необхідних програмних операцій в межах часу активізації задачі.

Вибір операційної системи.

Вади та переваги кожної з операційних систем наведені в табл.4.1.

Таблиця 4.1 – Порівняльна характеристика операційних систем

Тип	Переваги	Недоліки
Послідовна	Проста структура для послідовних задач	Не влаштовує виконання операцій у реальному часі
	Легше виконувати завантаження мікропроцесора	Не влаштовує задачі з різними пріоритетами
	Мінімальні вимоги до пам'яті для збереження контекстів задач	Відсутня гнучкість при внесенні змін в програмну структуру
	Легке перевантаження мікропроцесора	Не дозволяє легко вирішувати призначення пріоритетів
	--	Трудно повно використовувати потужність процесору
Паралельна	Пасує до операцій у реальному часі	Необхідна пам'ять великої місткості для збереження контексту задачі
	Пасує до задач з різними пріоритетами	Додаткове завантаження мікропроцесора для перемикання задач
	Вирішує задачу надання пріоритетів	Велика та складна структура
	Гнучка до додавання нових програмних задач	Важко передбачити навантаження мікропроцесора

Зазвичай програмне забезпечення мобільних засобів повинно виконувати ряд різноманітних функцій сигналів різної тривалості від 1мс до кількох секунд. Ця загальна вимога та бажання мати гнучку систему розробки прикладних програм дозволяє обрати в якості базової для GSM ARCH паралельну операційну систему, яка використовує метод перемикання задач.

Операційна система GSM ARCH комплект визначених розробником мобільних засобів задач, які розташовані ним у відповідності пріоритетам разом програмою обробки переривань[20] ISR.

Програмна задача може перебувати в одному з двох станів: активний, коли інтервал зупинки дорівнює нулю;тимчасово зупиненому, коли інтервал зупинки нерівний нулю.

В любий момент часу тільки одна задача може вирішуватися мікропроцесором. Коли більше одної задачі активно, задача з найбільшим пріоритетом отримує час мікропроцесора. Далі управління переходить до задачі з наступним пріоритетом. Задача, яка в даний час вирішується буде тимчасово втрачати управління мікропроцесора, коли задача з вищім пріоритетом стане активною.

4.2 контекстна схема операційної системи

Контекстна схема операційної системи наведена на рис. 4.1.

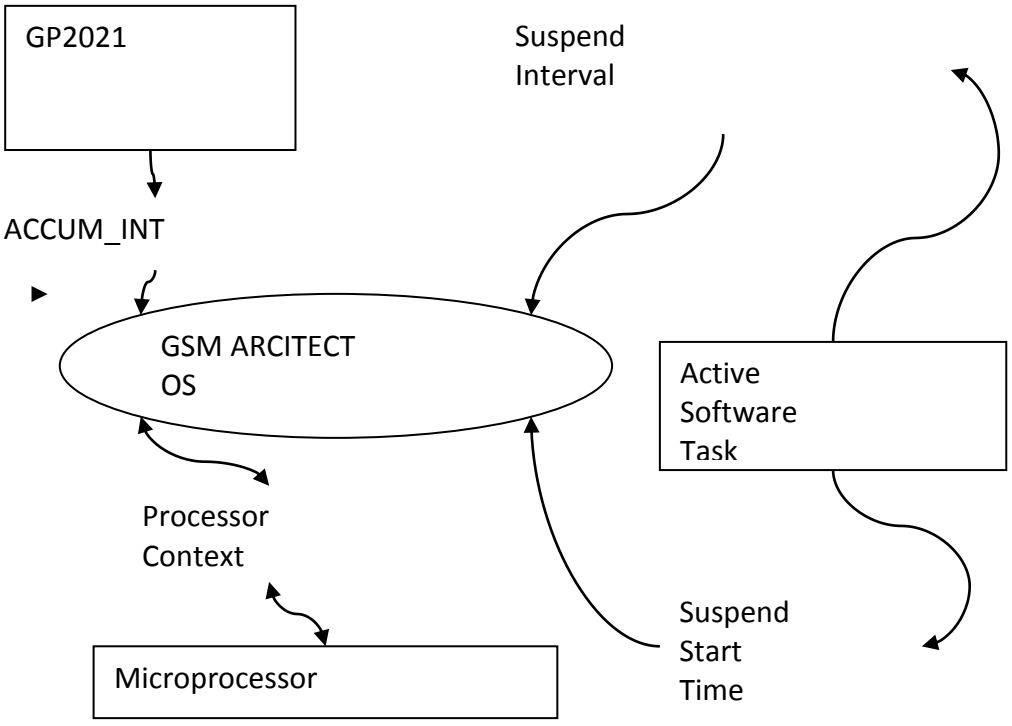


Рисунок 4.1 – Контекстна схема операційної системи GSM ARCH

[20] Interrupt Service Routine – ISR

В першу чергу система управляється перериванням ACCUM_INT GP2021. Це переривання використовується як механізм для знаходження маркерів часу TIC(0.1c), модифікації лічильника цих маркерів та зміни станів програмних задач: активний чи тимчасово зупинений. Інший стимул для операційної системи надходить від самозупинки програмної задачі після завершення всіх програмних операцій задачі. Задача надає операційній системі термін та час старту інтервалу зупинки. Під час тимчасової зупинки та її активізації контекст мікропроцесора повинен бути доступним.

4.3 Процес ініціалізації задач

Перед тим як операційна система почне нормальну роботу необхідно виконати ініціалізацію для завдання конфігурації управляючого блоку задач (Task Control Block – TCB). Схема потоків для цієї операції наведена на рис. 4.2.

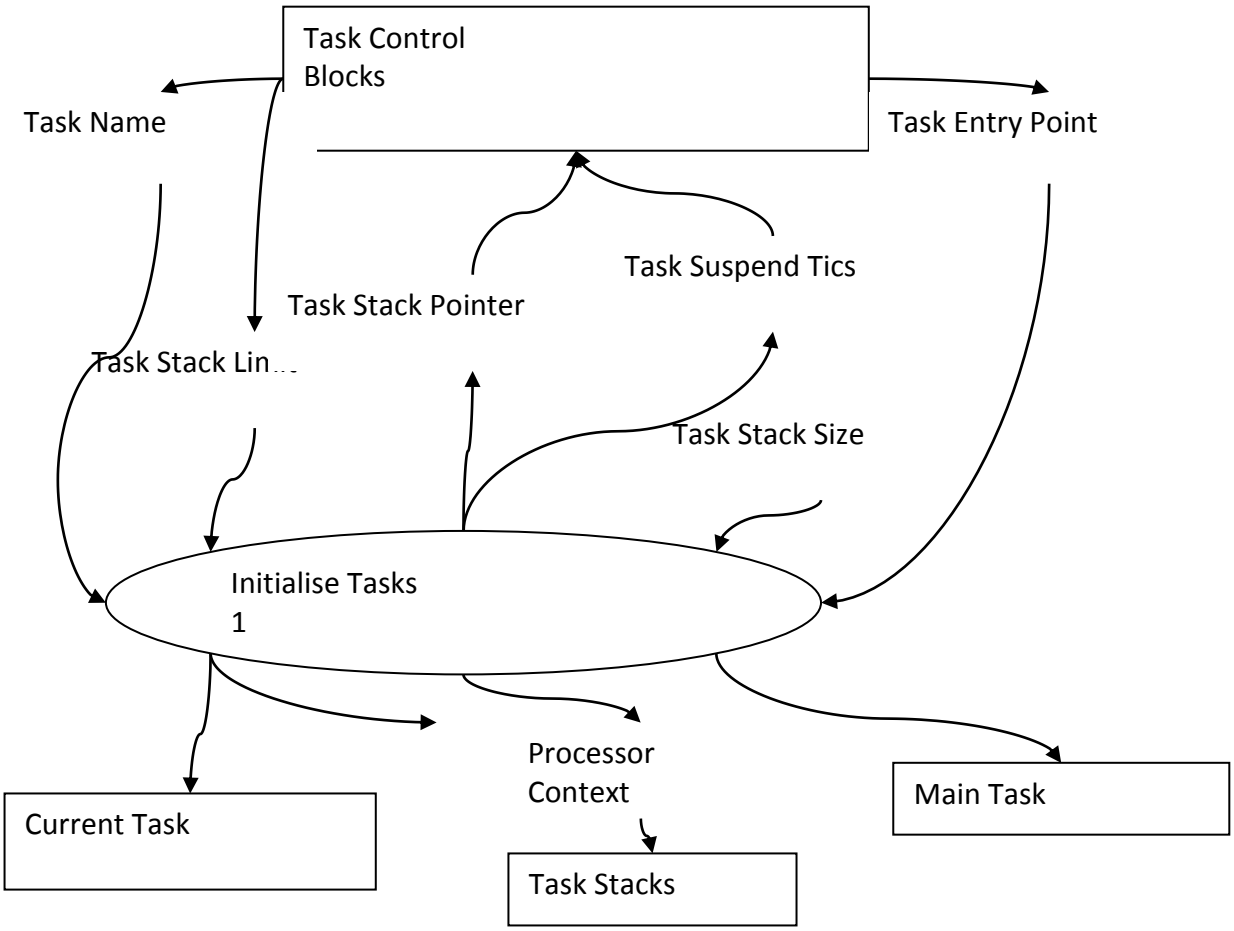


Рисунок 4.2 – Схема потоків даних при ініціалізації задачі (Parent: GSM ARCH OS)

При ініціалізації кожної задачі виконуються наступні операції: розподіл простору пам’яті під стек задачі, завдання покажчика вершини стеку та ліміту

стеку задачі, а також збереження контексту мікропроцесору в стек задачі, як це наведене на рис.4.3.

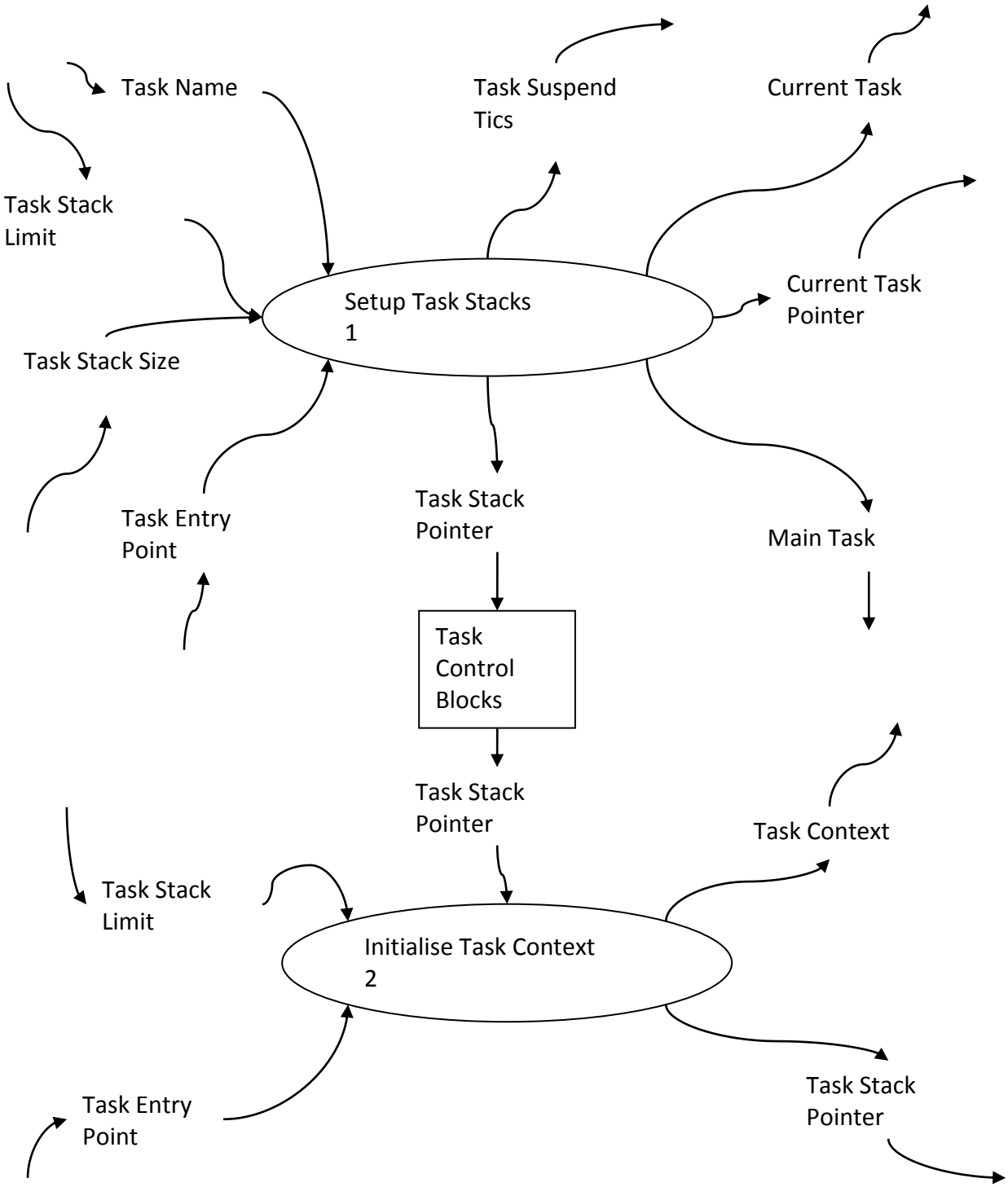


Рисунок 4.3 – Схема потоків даних при формуванні стеків та ініціалізації контексту задачі (Parent: Initialise Tasks)

Пріоритет задачі визначається номером задачі в масиві TCB структур. Найбільший пріоритет має перша задача, а самий низький пріоритет має остання задача, якою завжди виступає задача MAIN.

Оригінальний контекст задачі, який завантажується до мікропроцесора при першій активації задачі, зберігається в стеку задачі від бази вверх. Контекст задачі складається з 16 основних регістрів (r0-r15) та регістра стану поточної програми CPSR для даної задачі. При ініціалізації більшість цих регістрів можуть зберігати довільні величини. Тільки наступні регістри вимагають вірної ініціалізації: R10(sl) – регістр ліміту стеку задачі(застосовується для перевірки переповнення стека); R13(sp) – покажчик вершини стеку (оригінальне значення база стеку плюс 0x44); R15(PC) – лічильник задачі(оригінальна точка входу в задачу); CPSR – регістр статусу поточної задачі. Оригінальне визначення стеку наведено в табл.4.2.

Таблиця 4.2 – Оригінальне визначення стеку

Покажчик стеку	Регістр	Значення
База стеку	r0..r9	Don’t care
База стеку8– 0x28	r10(sl)	Ліміт стеку задачі
База стеку – 0x2C	R11, r12	Don’t care
База стеку – 0x34	R13(sp)	Покажчик стеку задачі (оригінальне значення „База стеку + 0x44)
База стеку – 0x38	R14	Don’t care
База стеку – 0x3C	R15(PC)	Точка входу в задачу
База стеку – 0x40	CPSR	Регістр статусу поточної програми

Специфікація процесу завдання стеків програмної задачі наведена нижче.

```
@IN = Task Name
@IN = Task Stack Limit
@IN = Task Stack Size
@IN = Task Entry Point
@OUT = Current Task
@OUT = Current Task Pointer
@OUT = Main Task
@OUT = Task Stack Pointer
@OUT = Task Suspend Tics
@PSPEC Setup Task Stacks
//For each task, (except MAIN), get a pointer to the stack limit
//and stack pointer
task counter = 0;
DO
    Task Stack Limit = memory allocation (Task Stack Size);
    Task Stack Pointer = Task Stack Limit + Task Stack Size
```

```
      task counter = task counter + 1
WHILE (Task Name != "MAIN")

// Set the current task and current task pointer to be MAIN and MAIN TCB
Current Task = Main Task = task counter;
Current Task Pointer = Task Entry Point of Main Task
Main Task Suspend Tics = 0
FOR task counter 0 TO Main Task -1
       Task Suspend Tics = 0
        Initialise Task Context()
ENDFOR
@
```

Специфікація процесу ініціалізації контексту програмної задачі наведена нижче.

```
@IN = Task Stack Limit
@IN = Task Entry Point
@INOUT = Task Stack Pointer
@OUT = Task Context
@PSPEC Initialise Task Context
Save Task Context to Task Stack Pointer in descending stack:
R0-r15,CPSR
Where:
       R10 = Task Stack Limit
       R13 = Task Stack Pointer
       R15 = Task Entry Point
       Others registers = don’t care
Update Task Stack Pointer with new stack pointer
«
```

5 ОБРОБКА ПЕРЕРИВАНЬ

Agenda:

1) схеми потоків даних щодо обробки переривання;

2) специфікації процесів переривання.

5.1 Схеми потоків даних щодо обробки переривання

Програма обробки переривань ISR зберігає контекст поточної задачі, якою управляє мікропроцесор. Коли надходить TIC зменшуються на один інтервали тимчасової зупинки задач та поновлюється виконання з найвищим пріоритетом. Коли TIC не надходить поновлюється задача, яка була збережена до входу в ISR. Коли TIC надійшов перемикач задачі може змінитися. Схеми роботи операційної системи при виникненні переривань наведені на рис. 5.1 та рис.5.2.

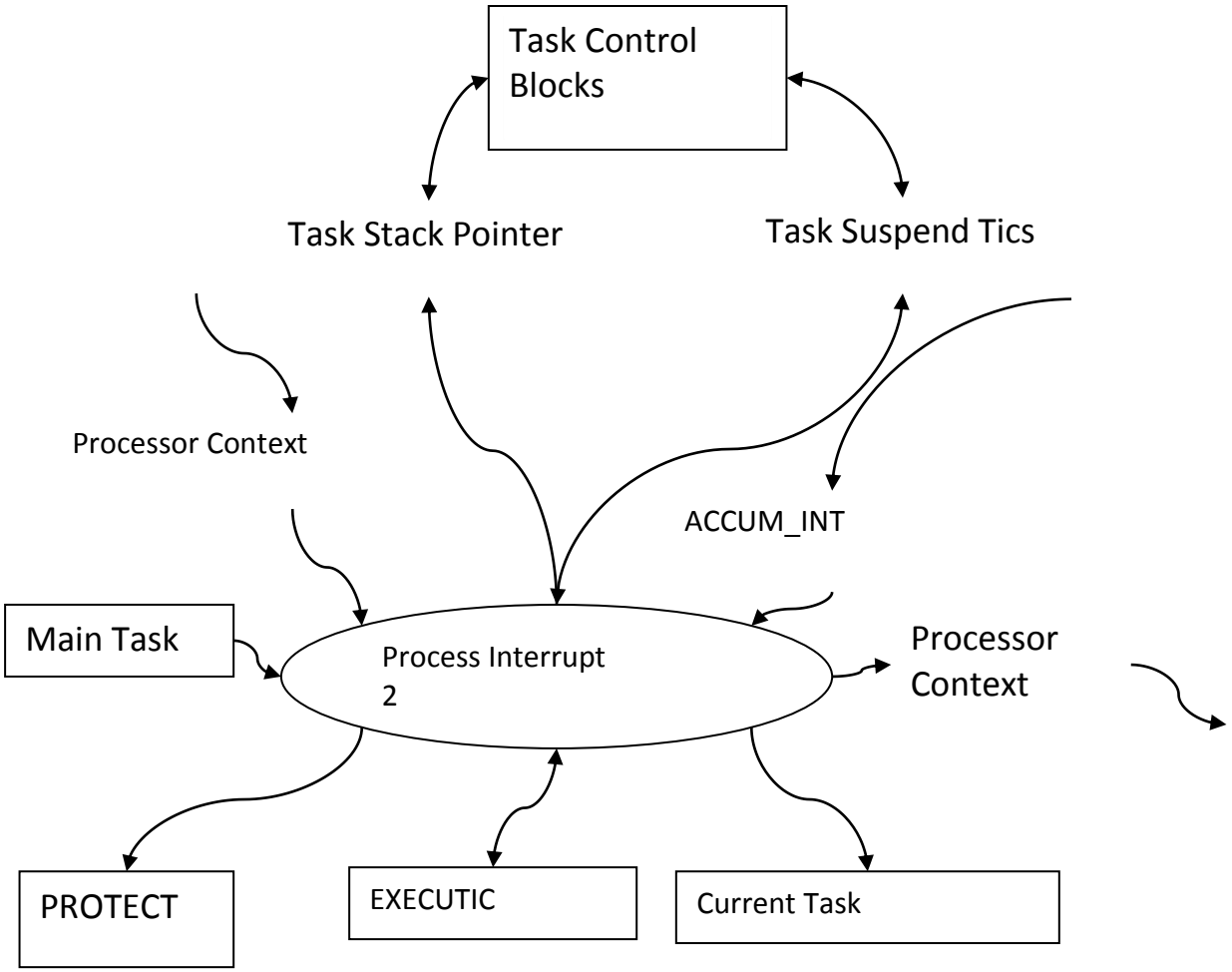


Рисунок 5.1 – Схема потоків даних при обробці переривань (Parent: GPS ARCH OS)

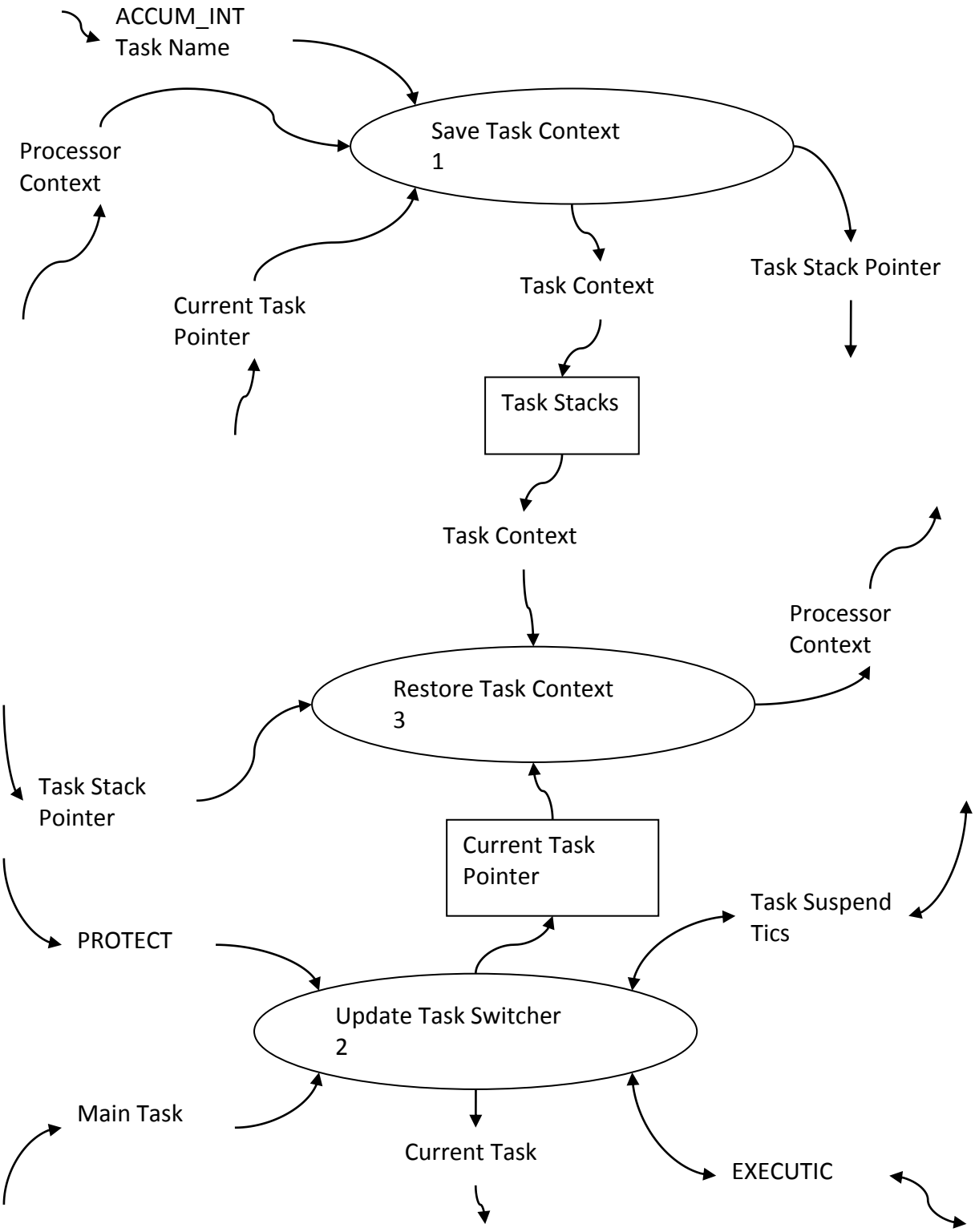


Рисунок 5.2 – Схема потоків даних при збереженні контексту задачі, виборі задачі та поновлені контексту задачі (Parent: Process Interrupt)

5.2 Специфікації процесів переривання

Специфікація процесу збереження контексту програмної задачі наведена нижче.

```
@IN = ACCUM_INT
@IN = Current Task Pointer
@IN = Processor Context
@OUT = Task Stack Pointer
@OUT = Task Context
@PSPEC Save Task Context
Upon an ACCUM_INT( the microprocessor is in IRQ mode).
Store the User Mode Processor Context (the interrupt task Task Context )
R0-r14, SPSR&pc
to the stack of the interrupted task.
Store the User Mode Stack Pointer
(the new Task Stack Pointer of the interrupted task)
to the TCB of the interrupted task (pointed to by Current Task Pointer)
@
```

Специфікація процесу зміни перемикача програмної задачі наведена нижче.

```
@IN = PROTECT

@IN = Main Task Point

@INOUT = Task Suspend Tics

@INOUT = EXECUTIC

@OUT = Current Task

@OUT = Current Task Pointer

@PSPEC Update Task Switcher

//If task switching is allowed and TICs have occurred.
IF PROTECT==0 and EXECUTIC!=0
    //Decrement the suspend intervals for all tasks.
    FOR Task = 0 TO Main Task -1
        IF Suspend Tics for task > EXECUTIC
          Task Suspend Tics for task -= EXECUTIC
        ELSE
          Task Suspend Tics for task = 0
        ENDIF
    ENDFOR

   //Find the highest priority task with zero suspend interval.
```

```
    FOR Task=0 TO Main Task)
        IF Task Suspend Tics for task == 0)
           Current Task = Task
           break
        ENDIF
    ENDFOR

// The Current Task Pointer points to the new (or current) task.
Current Task Pointer = Pointer to TCB of Current Task
EXECUTIC = 0;
ENDIF
@
```

Специфікація процесу поновлення контексту програмної задачі наведена нижче

```
@IN = Current Task Pointer

@IN = Task Stack Pointer

@IN = Task Context

@OUT = Processor Context

@PSPEC Restore Task Context

The microprocessor is in IRQ mode.

Restore the User Mode Stack Pointer (sp of the restored task)

from the TCB Task Stack Pointer for restored task

(pointed to by Current Task Pointer)

Restore the User Mode Processor Context (the interrupt task Task Context )
R0-r14, SPSR&pc
from the stack of the restored task.

Store the User Mode Stack Pointer
( the new Task Stack Pointer of the interrupted task)
to the TCB of the interrupted task (pointed to by User Mode Stack Pointer).

Switch the processor to User Mode.
@
```

6 ПРИПИНЕННЯ ЗАДАЧ

Agenda:

1) схеми потоків даних щодо припинення задачі;

2) специфікації процесів припинення задачі.

6.1 Схеми потоків даних щодо припинення задачі

Задача може тимчасово зупинити себе в любу мить, але звичайно зупинка виконується коли завершено виконання всіх операцій для поточної її активації. Після зупинки задача з найвищім пріоритетом отримує управління мікропроцесору. Схеми потоків зупинки задач наведені на рис.6.1 та на рис.6.2.

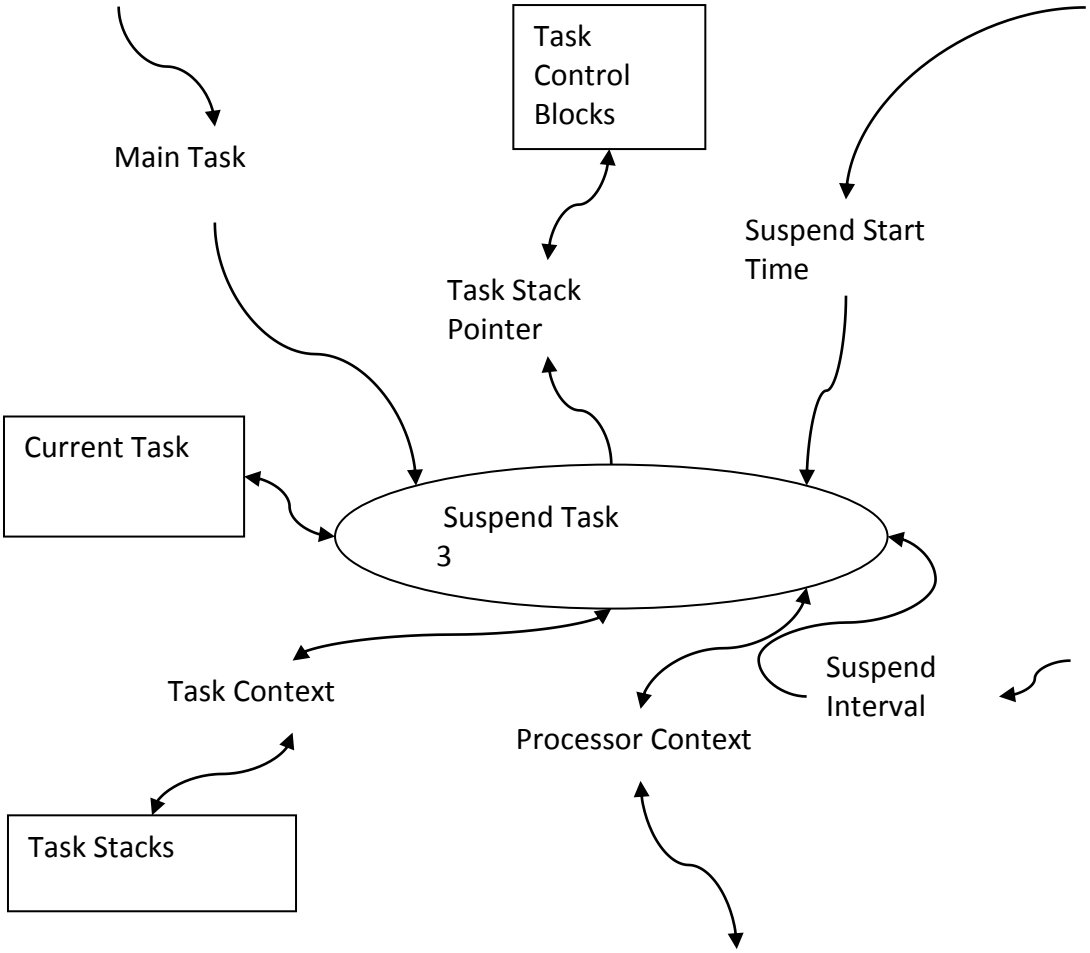


Рисунок 6.1 – Схема припинення задачі (Parent: Gps ARCH OS)

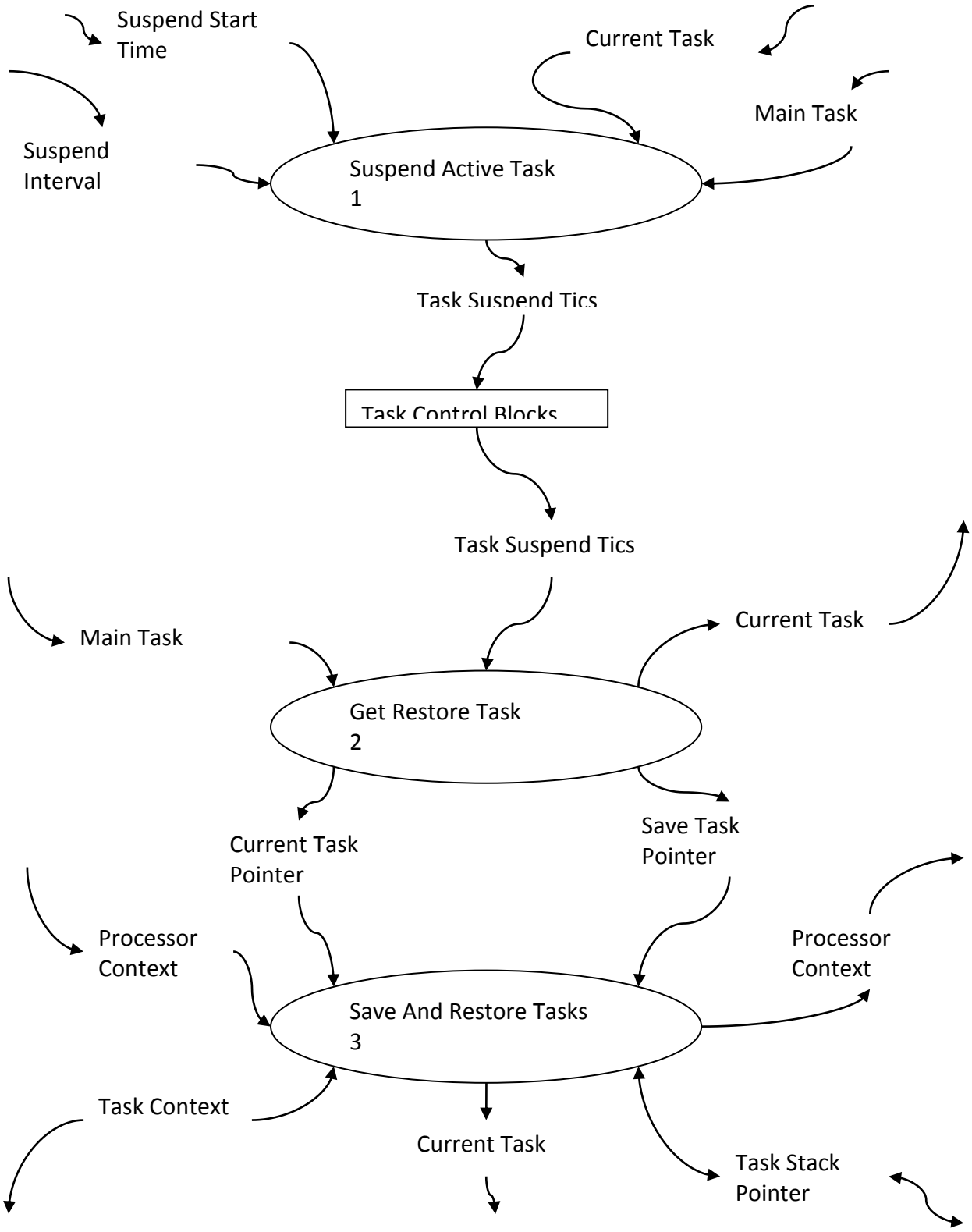


Рисунок 6.2 – Схема припинення активної задачі, підготовки наступної задачі, збереження та поновлення задач

6.2 Специфікації процесів припинення задачі

Специфікація процесу тимчасової зупинки активної програмної задачі наведена нижче.

```
@IN = Suspend Interval
@IN = Suspend Start Time
@IN = Current Task
@IN = Main Task

@OUT = Task Suspend Tics

@PSPEC Suspend Active Task

//Can't suspend MAIN
IF Current Task == Main Task
      RETURN
Get current time.
IF (current time – Suspend Start Time) < (Suspend Interval + 1)
      Task Suspend Tics for Current Task =
       Suspend Interval - (current time -Suspend Start Time)
ELSE
      Task Suspend Tics for Current Task = 1
ENDIF
@
```

Специфікація процесу визначення активної задачі наведена нижче.

```
@IN = Task Suspend Tics
@IN = Main Task
@OUT = Current Task
@OUT = Current Task Pointer
@OUT = Save Task Pointer
@PSPEC Get Restore Task
Save Task Pointer = TCB Pointer of Current Task
FOR Task = 0 TO MainTask)
      IF Task Suspend Tics for Task ==0)
        Current Task = Task
         break;
       ENDIF
ENDFOR
Current Task Pointer = TCB Pointer of Current Task
@
```

Специфікація процесу збереження та поновлення програмної задачі наведена нижче

@IN = Save Task Pointer

@IN = Current Task Pointer

@INOUT = Processor Context

@INOUT = Task Stack Pointer

@INOUT = Task Context

@PSPEC Save And Restore Tasks

Store the Processor Context (Task Context of the save task):
R0-r14, SPSR&pc
to the stack pointer of the TCB Save Task Pointer.

Store the new stack pointer to the TCB Task Stack Pointer of the save task.

Restore the processor context(Task Context of the restore task):
R0-r14, SPSR&pc
from the stack of the TCB pointed at by Current Task Pointer.
@

Кожна паралельна програмна задача може мати одну з двох загальних форм:

Ім’я Задачі()

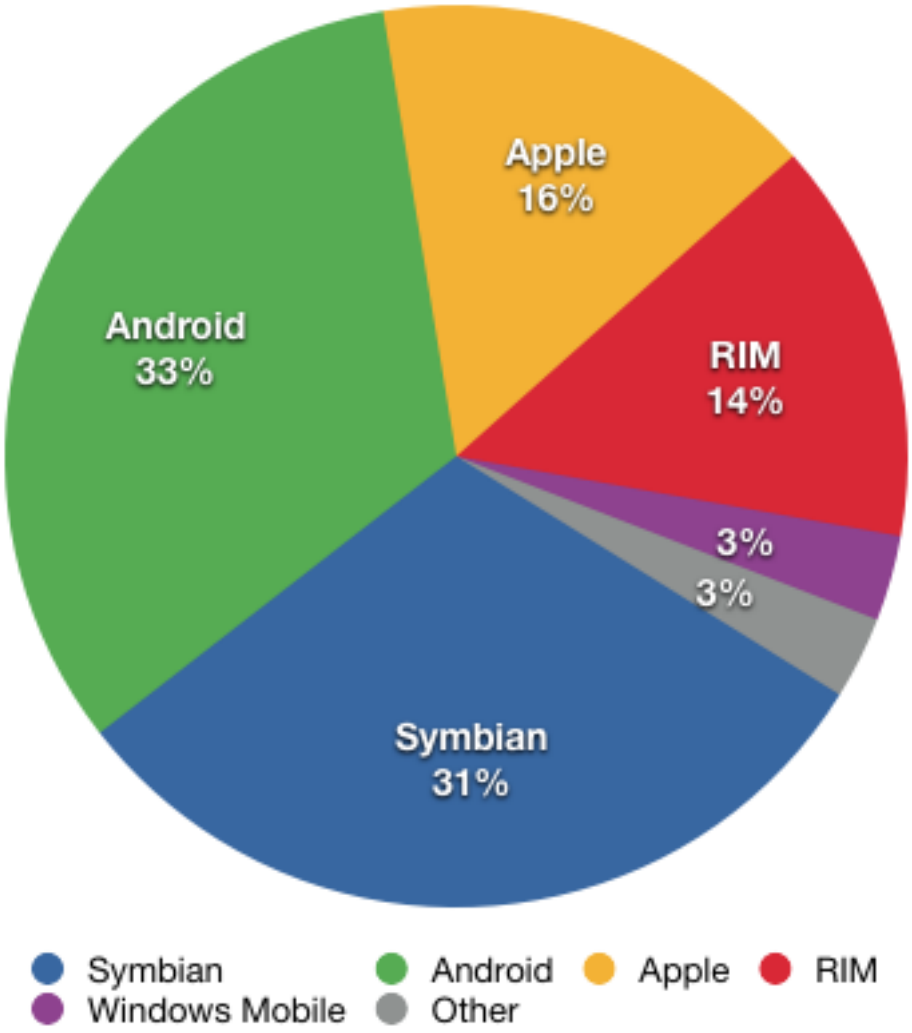
{
	Оголошення необов’язкових локальних змінних;
	Необов’язкові команди ініціалізації;
	Обов’язково
{
y = поточний час;
Команди та виклики функцій;
Зупинити через x TIC, які рахуються з моменту y;
}
}
або
Ім’я Задачі()
{
	Оголошення необов’язкових локальних змінних;
	Необов’язкові команди ініціалізації;
	Обов’язково
{

```
Команди та виклики функцій;
y = поточний час;
Зупинити через x ТІС, які рахуються з моменту y;
}
}
```

Тематичний модуль 3.
Прикладне програмне забезпечення

Лекція 7. Загальна характеристика GPS

Лекція 8. Тенденції розвитку



7 ЗАГАЛЬНА ХАРАКТЕРИСТИКА GPS

Agenda:

1) Програмне забезпечення GPS ARCH

2) Структура програмного забезпечення

3) Алгоритм обробки сигналу

7.1 Програмне забезпечення GPS ARCH

7.1 Програмне забезпечення GPS ARCH

Програмне забезпечення GPS ARCH формує бінарне зображення(EXE файл), яке має бути завантажено з комп'ютера PC до середовища ARM Toolkit.

Пакет ARM Toolkit використовує мову програмування C для розробки вихідного коду.

7.1 Атрибути програмного забезпечення

340 KBytes of ‘C’ source code

82 KBytes of ‘H’ include files

8 KBytes of ‘S’ assembler files

157 KBytes executable (40K332bit)

Compiles with ARM Toolkit v2.0

Конкуруючі завдання.

Програмне забезпечення виконує наступні конкуруючі завдання:

Прочитати акумулятори каналу корелятора в GP2021

Управляти каналом корелятора, що відстежує петлі

Контролювати умови замикання петлі

Оновлювати оцінки позиції та швидкості приймача

Аналізувати дані GPS, одержані від супутників

Збирати блоки вимірювання

Розраховувати позицію супутника щодо спостерігача, засновану на ефемеридних даних

Управляти стратегією вибору супутника

Формувати потік вихідних даних RS232

Перевіряти існування даних RTCM SC-104 DGPS для виправлення і аналізу даних.

7.2 Структура програмного забезпечення

Операційна система перемикання завдань.

Програмне забезпечення діє як просте перемикання завдань операційною системою, щоб забезпечити дію конкуруючих структур, які надають собі час щодо вимог обробки сигналу GPS і програмне забезпечення.

Операційна система перемикання завдань складається з переривань і процедур обробки завдань. Тільки одне завдання може бути активне в будь-який час.

Головне джерело – це переривання одержане від GP2021, сигнал ACCUM_INT (типове значення періоду 505,05μs - у програмі GPS ARCH встановлюється значення 900,025μs) обслуговується програмою обробки переривання (Interrupt Service Routine-ISR). Процедура ISR управляє обслуговуванням операційної системи GPS ARCH і обробкою даних сирих накопичень GP2021 для підтримки сигналу корелятора, що відстежує петлі.

Управління завданнями.

Архітектор GPS має 7 певних завдань і 1 рутина обслуговування переривання. Нові завдання можуть легко бути додані споживачем.

Завдання можуть бути або активними, або зупиненими. Тільки одне завдання обслуговується мікропроцесором в один момент.

Кожне конкуруюче завдання програмного забезпечення бере одну із загальних двох форм:

1). Завдання, яке винне бути відновленим у фіксовані інтервали часу

2). Завдання, яке винне бути підвішеним у фіксовані інтервали часу

Пріоритет завдання визначає яке завдання одержує процесор протягом конфлікту завдань. Завдання Main()завжди визначене, щоб мати найнижчий пріоритет. Затримання процесора може бути досягнуте, виводячи з ладу перемикання задач або блокуючи (маскуючи) переривання GP2021. Рутина припинення завдання діє в межах операційної системи щоб тимчасово зупинити дію завдання для певної кількості TlCs (1TlC – 0,0999999s).

Масив TCB.

Завдання можуть бути додані або видалені з програмного забезпечення, виправляючи глобальну структуру Блоку управління завданнями (Task Control Block-TCB). Порядок завдань в декларації TCB є пріоритетом завдання.

Для кожного завдання, TCB має наступні записи:

- 1) покажчик на адресу початку завдання;
- 2) покажчик на межу стеку (ініціалізація NULL);
- 3) покажчик вершини стека (ініціалізація 0);
- 4) ім'я завдання;

- 5) розмір стека завдання;
- 6) інтервал зупинки завдання.

7.3 Алгоритм обробки сигналу

Знаходження коду і стеження за ним.

Для знаходження коду, обидва і фаза коду GP2021 і частота носія повинні відповідати фазі коду і частоті носія прийнятого сигналу на такій тривалості, щоб кореляція була вище порогу виявлення.

GPS ARCH виконує пошук у площині фаза коду, несуча частота, перебираючи всі кодові фази в серіях частотних діапазонів, поки сигнал не виявлений.

Як тільки сигнальне виявлення сигналу відбувається, цикли пошуку коду і носія закриваються.

Для стеження за кодом використовується Фазова Замкнута Петля.

Для стеження за носієм використовується Частотна Замкнута Петля.

Петля, що відстежує носій, допомагає петлі, яка відстежує код.

Пошук коду.

GPS ARCH використовує точну копія, яка ковзає для пошуку кодового придбання.

Частота GP2021 генератору коду DCO програмується злегка вище частоти формування чипів, яка передбачена таким чином, що коди з часом ковзають один за одним. Типове програмоване значення надає пошукову швидкість 0,25 чипа за мілісекунду.

Відтепер, повний пошук коду (1023 чипи) у наданій частоті займає близько 4 секунд для завершення. Частотні діапазони мають 500Hz завширшки.

Тому, щоб знайти частоту у просторі +-10,25kHz необхідно близько 164 секунд для завершення.

8 ТЕНДЕНЦІЇ РОЗВИТКУ

Agenda:

1) Контекстна схема обслуговування переривання

2) Вісім схем потоків даних та специфікацій

8.1 Контекстна схема обслуговування переривання

Короткий огляд процедури обслуговування

Процедура[21] ISR виконує два основних набора функцій: обслуговування операційної системи та обробка даних, які накоплені в мікросхемі GP2021, з метою відтворення прийнятого інформаційного сигналу. Далі розглядається тільки обробка даних, які накопичені в мікросхемі GP2021

Процедура ISR активується перериванням ACCUM_INT, яке надходить з мікросхеми GP2021 з періодом 900.025μs, та має самий високий пріоритет у порівнянні з любою прикладною задачею програмного забезпечення.

Процедура ISR повинна бути виконана до повторного виконання за перериванням ACCUM_INT(non-re-entrant). Очевидно, що час на виконання рутини ISR повинен бути значно менше періоду формування переривань ACCUM_INT, щоб залишився час на виконання інших прикладних програмних задач.

Функції ISR.

Програма GPS ARCH збирає данні I,Q з тих каналів,які доступні для накопичення нових даних. Ці данні використовуються в ряді функцій:

1) визначення значень лічильників епох коду та несучої;

2) визначення індикаторів блокування коду та несучої;

3) стеження за несучою;

4) стеження за кодом;

5) збереження бітових даних повідомлень від супутників.

Процедура ISR активізується ACCUM_INT, тому вона має самий високий пріоритет над усіма програмними задачами.

Тому немає необхідності підтримувати методологію активізації і зупинки задач, які застосовуються іншими задачами програмного забезпечення GPS Architect.

При кожній активації ISR, перша операція – це збір накопичених даних від активних каналів, в яких доступні нові накопичені дані.

[21] Interrupt Service Routine – ISR

Поява нових накопичених даних в каналах асинхронне один до одного та до переривання ACCUM_INT, тому необхідно виконувати опит активних каналів на наявність нових даних.

Регістр стану GP2021 ACCUM_STATUS_A має 12-розрядну карту біт каналів, які мають нові накопичені данні.

Приймання I,Q

При кожній активації ISR, перша операція – це збір накопичених даних від активних каналів, в яких доступні нові накопичені дані.

Поява нових накопичених даних в каналах асинхронне один до одного та до переривання ACCUM_INT, тому необхідно виконувати опит активних каналів на наявність нових даних.

Регістр стану GP2021 ACCUM_STATUS_A має 12-розрядну карту біт каналів, які мають нові накопичені данні.

Розрахунок часу передавання даних.

Важливо щоб читання і збереження нових накопичених даних для всіх каналів відбувалося якомога швидше після переривання ISR, щоб збільшити період повторення ACCUM_INT і зменшити навантаження мікропроцесора.

Номінальний час накопичення коду C/A – 1ms. Якщо час читання і збереження усіх 12 каналів складає ΔT, то період повторення ACCUM_INT необхідно встановити рівним (1-ΔT) ms.

Фактично період буде ще меншим з урахуванням часу входження до ISR і необхідну обробку перед читанням суматорів.

У кожному каналі необхідно прочитати 4 суматора (In-phase Prompt, In-phase Track, Quadrature Prompt, Quadrature Track), тому максимально буде прочитано 48 накопичень.

8.2 Вісім схем потоків даних та специфікацій

Склад специфікацій:

- приймання I,Q;
- приймання лічильника епохи;
- перевірка TIC;
- поновлення лічильника пропущених накопичень;
- стеження за індикаторами захвату;
- поновлення параметрів петлі стеження за кодом;
- поновлення параметрів петлі стеження за несучою;
- формування потоку даних.

9 КОНТРОЛЬНІ ЗАВДАННЯ ТА ЗАПИТАННЯ

Скласти специфікацію процесу при виконані функції: Initialise Tasks ().

Скласти специфікацію процесу при виконані функції: Initialise Context ().

Скласти специфікацію процесу збереження контексту мікропроцесору при виконані функції: IRQ Handler()

Скласти специфікацію процесу зміни задачі при виконані функції: GPISR()

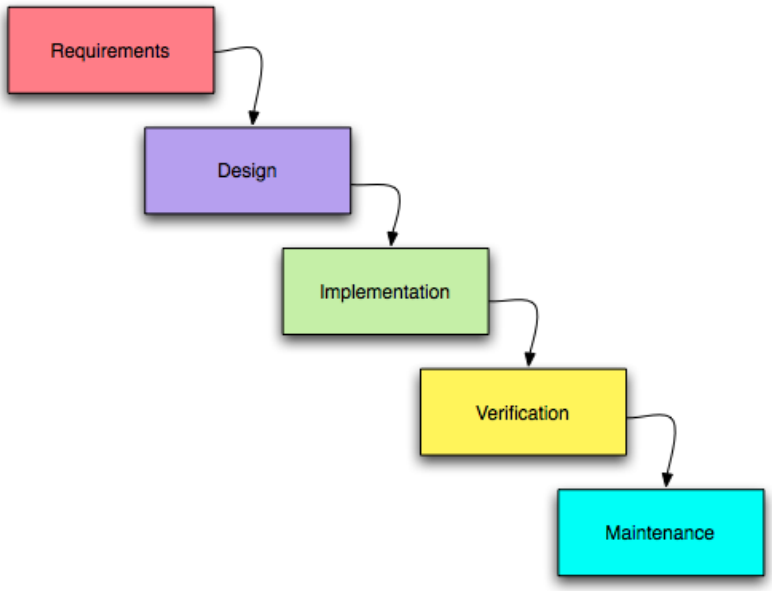
Скласти специфікацію процесу поновлення контексту мікропроцесору процесу при виконані функції: IRQ Handler ()

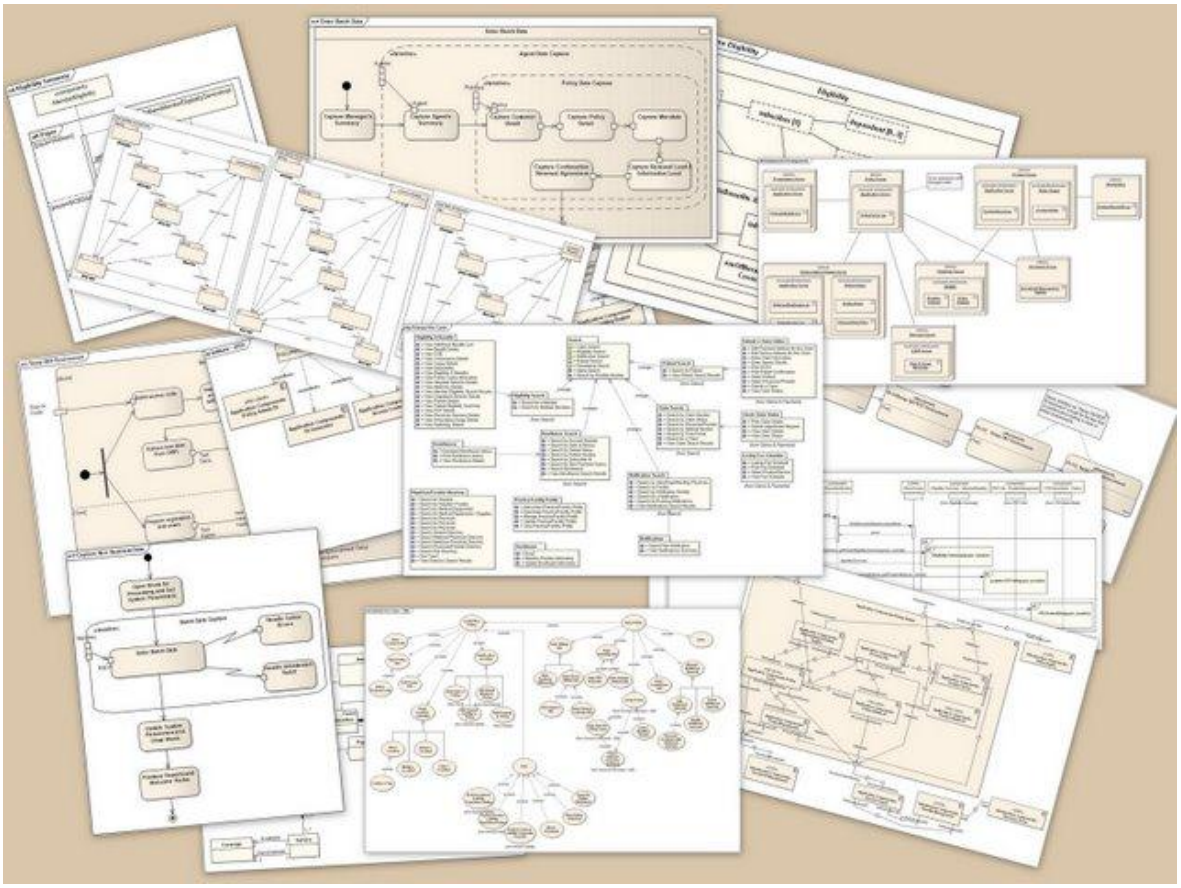
Скласти специфікацію процесу зупинки активної задачі при виконані функції: Suspend ()

Скласти специфікацію процесу визначення активної задачі при виконані функції: Suspend ()

Скласти специфікацію процесу збереження та поновлення контексту мікропроцесору при виконані функції: Save And Restore ().

Висновки


Requirements
Design
Implementation
Verification
Maintenance



ПЕРЕЛІК ЛІТЕРАТУРИ

1. Абель, П. Язык Ассемблера для IBM PC и программирования / Пер. с англ. Ю.В. Сальникова / П. Абель - М.: Высшая школа, 1992.- 447 с.

2. Quartus II Handbook Version 10.0. Volume 1: Design and Synthesis - 1769с.- Режим доступу : www/ URL: http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf - 10.02.2011р. - Загл. с екрана.

3. Скляр, Б. Цифровая связь. Теоретические основы и практическое применение. Изд.2-е, испр.: Пер с англ./ Б. Скляр – М.: Издательский дом «Вильямс», 2007. – 1104с.: ил.

4. Сергиенко, Б.А. Цифровая обработка сигналов/ А.Б.Сергиенко − СПб.: Питер, 2007. – 608с.: ил.

5. Бондарев, В.Н. Цифровая обработка сигналов: методы и средства. Учебное пособие для вузов 2-изд./ В.Н. Бондарев, Г. Трестер, В.С. Чернега - Х.: Конус.2001. - 398с

6. Шиндлер, Д.Л. Основы компьютерных сетей.: Пер. с англ. – М: Издательский дом «Вильямс», 2003. – 656с.

7. Product information and application notes. Mitel Semiconductors. TECHNICAL DOCUMENTATION GP2000 GPS Chipset – Designer’s GuideSupersedes Issue 1.4 in August 1996 Global Positioning Products Handbook, HB3045-1.0 MS4395-2.3 April 1998. © Mitel Corporation 1998 Publication No. MS4393 Issue No. 2.3 April 1998. - 1 електрон. опт. диск (CD-ROM). - Загл. с етикетки диска