УДК 004

О.В.ОГНЄВИЙ, О.О.КОБЛЄНЄВ
Хмельницький національний університет

ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОЦЕСОРІВ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ ПРИ РОЗВ'ЯЗУВАННІ ЗАДАЧ РЕАЛЬНОГО ЧАСУ

У статті досліджено системи реального часу (СРЧ) та сформовано основні вимоги для паралельної оброки даних. Розкрито динамічні властивості програм реального часу. Проаналізовано програми «жорсткого», «м'якого» і «інтерактивного» реального часу та їх ознаки. Описані алгоритми реального часу та їх особливості. Розглянуто основні особливості архітектури процесорів цифрової обробки сигналів при вирішені задач реального часу.

Ключові слова: системи реального часу, архітектури процесорів,паралельне виконання команд.

In the article the real-time(СРЧ) systems are investigational and the basic requirements are formed for parallel obroki of data. Dynamic properties of the programs of the real time are exposed. It is analysed program "hard", "soft" and "interactive" real time and their sign. Described algorithms of the real time and their feature. The basic features of architecture of processors of digital treatment of signals are considered at decided tasks of the real time.

Keywords: real-time, architecture of processors systems, concurrent execution of commands.

Вступ

Архітектура сучасних цифрових процесорів зазнала значних змін в порівнянні з процесорами першого покоління, побудованими по архітектурі фон Неймана. Ці зміни в структурі ядра процесора і організації пам'яті забезпечують відчутний приріст продуктивності при вирішенні завдань цифрової обробки сигналів.

Розподіл пам'яті на пам'ять програм і пам'ять даних з можливістю зберігання даних в пам'яті програм разом з використанням кеша програм дозволяє виконувати будь-яку обчислювальну операцію, що вимагає завантаження двох операндів за 1 такт.

Підвищення продуктивності вдалося досягти за рахунок паралельного виконання декількох команд. При цьому процесори по реалізації паралелізму розділилися на два фронти: в одному підтримується виконання однієї і тієї ж команди для декількох наборів операндів, а в іншому - паралельне виконання декількох команд, операндами для яких є внутрішні регістри процесора. Оригінальний підхід до паралельних обчислень продемонструвала компанія Analog Devices. При створенні ADSP - 2116x компанія модифікувала свій стандартний процесор з плаваючою точкою ADSP - 2106x, додавши додатковий набір виконуючих пристроїв, повністю дублюючий основний. Кожен набір пристроїв в ADSP - 2116x включає помножувач-накопичувач, АЛП, пристрій зсуву і кожен має власний набір регістрів. ADSP - 2116x може сформувати єдину команду і виконати її паралельно для обох наборів різних даних, що ефективно подвоює продуктивність в деяких алгоритмах.

Нарешті, апаратна реалізація операцій часто використовуваних в цифровій обробці сигналів, таких як множення з накопиченням, збільшує продуктивність при їх обчисленні як мінімум удвічі (за рахунок того, що ця операція виконується за 1 такт, а не за 2) в порівнянні з процесорами, що не підтримують таку можливість.

Постановка задачі

Канонічне визначення системи реального часу дано Дональдом Гілліесом і виглядає так: «Системою реального часу є така система, коректність функціонування якої визначається не тільки коректністю виконання обчислень, але і часом, за який отримано потрібний результат. Якщо вимоги за часом не виконуються, то вважається, що відбулась відмова системи ».

Для подібних систем характерно:

- гарантований час реакції на зовнішні події (наприклад на переривання від обладнання);
- жорстка підсистема планування процесів (високо пріоритетні завдання не повинні витіснятися низько пріоритетним, за деякими винятками);
- підвищені вимоги до часу реакції на зовнішні події або реактивності (затримка виклику обробника переривання не більш десятків мікросекунд, затримка при перемиканні задач не більше сотень мікросекунд).

Дослідження систем реального часу дозволяють висунути слідуючі основні вимоги:

1. Підтримка багатонітієвості і диспетчеризації з витісненням. Поведінка СРЧ має бути передбачуваною. Це не означає, що СРЧ повинна бути швидкою, але означає, що максимальний проміжок часу для виконання будь-якої операції має бути відомий заздалегідь і повинен бути узгоджений з вимогами програми. Для забезпечення витіснення на рівні переривань структура обслуговування переривань (і апаратна архітектура) повинна бути багаторівневою.

2. Наявність поняття пріоритету гілки. При побудові конкретної СРЧ розробник повинен вибудувати пріоритети задач таким чином, щоб кожна з них встигла з реакцією до свого критичного терміну.

3. Підтримка передбачуваних механізмів синхронізації гілок.

4. Наявність механізму спадкування пріоритетів. Щоб уникнути проблеми інверсії пріоритетів СРЧ повинна бути забезпечена механізмом успадкування пріоритетів, тобто блокуюча гілка повинна враховувати пріоритет гілки, яку вона блокує, але тільки в тому випадку, якщо заблокована гілка має більш високий пріоритет).

5. Поведінка ОС повинна бути передбачуваною. Розробник повинен знати всі часи виконання системних викликів і вміти передбачати поведінку системи в будь-яких ситуаціях.

Загальні характеристики СРЧ:

- великі і складні системи;
- розподілені системи;
- жорстка взаємодія з апаратурою;
- виконання завдань залежить від часу;
- складність тестування.

Основна частина

Динамічні властивості програм реального часу прийнято характеризувати трьома визначеннями: програми «жорсткого», «м'якого» і «інтерактивного» реального часу.

Жорсткий реальний час. Передбачає наявність гарантованого часу відгуку системи на конкретну подію, наприклад, апаратне переривання, видачу команди управління і т.п. Абсолютна величина часу відгуку великого значення не має. Так, якщо необхідно, щоб програма відпрацювала деяку команду за 1 мілісекунду, але вона справляється з цим завданням лише в 95% випадків, а в 5% не вкладається в норматив, таку систему можна охарактеризувати як таку, що працює в жорсткому реальному часі. Якщо ж команду потрібно відпрацювати протягом години, що і відбувається в 100% випадків - у наявності жорсткий реальний час. У більшості російськомовної літератури такі системи називають системами з детермінованим часом. При практичному застосуванні час реакції має бути мінімальним.

М'який реальний час. В цьому випадку очікуваний час відгуку системи є величиною швидше індикативною, ніж директивною. Звичайно, передбачається що в більшості випадків (відсотків 80 - 90) відгук вкладеться в задані межі. Однак і інші варіанти - у тому числі повна відсутність реакції системи - не повинні призводити до плачевних результатів. Зазвичай вважається, що якщо часовий норматив перевищений на один порядок, то це ще терпимо. М'який реальний час також називається квазіреальним.

Універсальна СРЧ повинна вміти виконувати довільні (заздалегідь невизначені) часові завдання без застосування спеціальної техніки. Розробка таких систем є найскладнішим завданням, хоча зазвичай вимоги до таких систем м'якші, ніж вимоги до спеціалізованих систем.

У системах реального часу необхідне введення деякого директивного терміну (deadline), до закінчення якого задача повинна обов'язково (для систем м'якого реального часу - бажано) виконатися. Цей директивний термін використовується планувальником завдань як для призначення пріоритету задачі при її запуску, так і при виборі завдання на виконання. Практично всі системи промислової автоматизації є системами реального часу.

В залежності від відмінностей, системи реального часу прийнято розрізняти як системи «жорсткого» та «м'якого» реального часу. Системою «жорсткого» реального часу називається система, де нездатність забезпечити реакцію на будь-які події в заданий час є відмовою і веде до неможливості вирішення поставленого завдання.

Ознаки систем жорсткого реального часу:

- неприпустимість жодних затримок ні за яких умов;
- марність результатів при запізненні;
- катастрофа при затримці реакції;
- ціна запізнення нескінченно велика.

Приклад системи жорсткого реального часу - бортова система управління літаком.

Так як система «м'якого» реального часу може не встигати все робити завжди в заданий час, виникає проблема визначення критеріїв успішності її функціонування. В залежності від функцій системи це може бути максимальна затримка у виконанні будь-яких операцій, середня своєчасність обробки подій і т. п. Більше того, ці критерії впливають на те, який алгоритм планування завдань є оптимальним.

Система м'якого реального часу характеризується наступними ознаками:

- за запізнення результатів доводиться платити;
- зниження продуктивності системи, спричинене запізненням реакцій, прийнятне.

Прикладами можуть служити автомат роздрібної торгівлі та підсистема мережевого інтерфейсу. В останньому випадку можна відновити пропущений пакет, використовуючи мережевий протокол, що повторює передачу пропущених пакетів. При цьому, звичайно, відбудеться зниження продуктивності системи.

Основна відмінність між системами жорсткого і м'якого реального часу визначається наступними вимогами: система називається системою жорсткого реального часу, якщо вона «не має права запізнюватися», і м'якого реального часу, якщо їй «не слід запізнюватися».

Алгоритми реального часу та їх особливості. В даний час для вирішення задачі ефективного планування в СРЧ найбільш інтенсивно розвиваються два підходи:

- статичні алгоритми планування (RMS) - використовують пріоритетне витіснення планування; пріоритет присвоюється кожній задачі до того, як вона почала виконуватися; перевага віддається завданням із самими короткими періодами виконання;
- динамічні алгоритми планування (EDF) - пріоритет завданням привласнюється динамічно, причому перевага віддається завданням з найбільш раннім граничним часом початку (завершення) виконання.

При великих завантаженнях системи EDF більш ефективні, ніж RMS. Більшість СРЧ виконують планування завдань, керуючись наступною схемою. Кожній задачі в додатку ставиться у відповідність деякий пріоритет. Чим більше пріоритет, тим вище повинна бути реактивність завдання. Висока реактивність досягається шляхом реалізації підходу пріоритетного витіснення планування, суть якого полягає в тому, що планувальником дозволяється зупиняти виконання будь-якої задачі в довільний момент часу, якщо встановлено, що інша задача повинна бути запущена негайно.

Описана схема працює за наступним правилом: якщо два завдання одночасно готові до запуску, але перше має високий пріоритетом, а друге низьким, то планувальник віддасть перевагу першому. Друге завдання буде запущене тільки після того, як завершить свою роботу перше.

Можлива ситуація, коли завдання з низьким пріоритетом вже запущене, а планувальник отримує повідомлення, що інша задача з більш високим пріоритетом готова до запуску. Причиною цьому може бути зовнішній вплив (переривання від обладнання), як, наприклад, зміна стану перемикача пристрою, керованого СРЧ. У такій ситуації планувальник завдань поведеться згідно підходу пріоритетного витісння планування наступним чином. Задачі з низьким пріоритетом буде дозволено виконати до кінця поточну асемблерну команду (але не команду, описану в коді програми мовою високого рівня), після чого виконання завдання зупиняється. Далі запускається завдання з високим пріоритетом. Після того, як воно опрацьовано, планувальник запускає перерване перше завдання з асемблерної команди, наступної за останньою виконаною.

Кожен раз, коли планувальник завдань отримує сигнал про настання деякої зовнішньої події (тригер), причина якого може бути як апаратна, так і програмна, він діє за наступним алгоритмом:

1. Визначає, чи повинно поточно виконуване завдання продовжувати працювати.
2. Встановлює, яке завдання повинно запускатися наступним.
3. Зберігає контекст зупиненого завдання (щоб вона потім відновило роботу з місця зупинки)
4. Встановлює контекст для наступного завдання.
5. Запускає цю задачу.

Ці п'ять кроків алгоритму також називаються перемиканням завдань.

Особливості сучасних СРЧ. Сучасні СРЧ мають ряд особливостей в порівнянні з системами поділу часу.

Обмежений час відповіді. По суті, система реального часу - це апаратно-програмний комплекс, що реагує в передбачувані часи на непередбачуваний потік зовнішніх подій. Це означає, що:

- ОС повинна встигнути відреагувати на подію, що сталася на об'єкті, протягом часу, критичного для цієї події. Величина критичного часу для кожної події визначається об'єктом і самою подією, і, природно, може бути різною, але час реакції системи має бути передбачений (обчислено) при створенні системи. Відсутність реакції на передбачений час вважається для СРЧ помилкою.
- система повинна встигати реагувати на події що одночасно відбуваються. Навіть якщо дві або більше зовнішніх подій відбуваються одночасно, система повинна встигнути зреагувати на кожну з них протягом інтервалу часу, критичного для цих подій.

Застосування СРЧ завжди конкретно. Якщо ОС загального призначення зазвичай сприймається користувачами (не розробниками) як вже готовий набір додатків, то операційна система реального часу служить тільки інструментом для створення конкретного апаратно-програмного комплексу реального часу.

Для більшості СРЧ передбачається, що основна частина оброблюваних даних апріорно відома. Тому найбільш широкий клас користувачів операційних систем реального часу - розробники комплексів реального часу. Проектуючи і розробляючи конкретну систему реального часу, програміст завжди знає точно які події можуть відбутися на об'єкті, знає критичні терміни обслуговування кожного з цих подій.

Портування. Управління прокатними станами, роботами, рух на автомагістралях, контроль за станом навколишнього середовища, управління атомними і космічними станціями та багато іншого - область задач реального часу. Для різних областей застосування ОС РЧ існують різні апаратні платформи і для кожної необхідно портування, тобто процес «стикування» програмної частини ОС та її апаратного забезпечення.

При виборі апаратної платформи для систем реального часу основними моментами є жорсткі вимоги до часових характеристик і гнучкості системи. Вимоги до апаратного забезпечення в даний час досить чітко визначені. Більшість проектів реального часу здійснюється в рамках архітектурних рішень магістрально-модульних систем (ММС).

Однак, якою би не була важлива сама СРЧ, зараз в умовах доступності сумісних апаратних засобів основна увага приділяється розробці і налагодженню прикладного програмного забезпечення, чия частка у витратах на розробку систем реального часу становить до 70%.

В залежності від середовища в якому функціонують системи їх можна розділити на класи:

- програмування на рівні мікроконтролерів;
- мінімальне ядро СРЧ;
- ядро СРЧ та інструментальна середа;
- ОС з повним сервісом.

Вбудовані системи реального часу. Відмінність - застосування операційної системи реального часу завжди пов'язане з апаратурою, з об'єктом, з подіями, що відбуваються на об'єкті. Система реального часу, як апаратно-програмний комплекс, включає в себе датчики, які реєструють події на об'єкті, модулі вводу-виводу, перетворюють показники датчиків в цифровий вигляд, придатний для обробки цих показань на комп'ютері, і, нарешті, комп'ютер з програмою, що реагує на події, що відбуваються на об'єкті. Операційна система реального часу орієнтована на обробку зовнішніх подій. Саме це призводить до корінних відмінностей (порівняно з ОС загального призначення) в структурі системи, у функціях ядра, в побудові системи введення-виведення.

Операційна система реального часу може бути схожа по інтерфейсу на ОС загального призначення, однак побудована вона абсолютно інакше.

Останнім часом високопродуктивні мікропроцесори, а з ними і операційні системи реального часу, все частіше використовуються в так званих «глибоко вбудованих» застосуваннях. До таких комп'ютерних систем пред'являються дві основні вимоги: малі габарити і низька вартість. Тому глибоко вбудовані мікропроцесорні системи ставлять дві проблеми на шляху застосування серійних СРЧ: невеликі обсяги використовуваної пам'яті і відсутність «зайвих» інтерфейсів, по яких можна було б пов'язати цільову та інструментальну машину на етапі розробки вбудованого ПЗ.

За структурним характеристикам програмно-апаратні комплекси можна розділити на класи:

- виконавчі системи реального часу;
- ядра реального часу;
- Unix'и реального часу.

Відзначимо основні особливості архітектури процесорів цифрової обробки сигналів при вирішені задач реального часу. :

1. Гарвардська архітектура , основу якої становить фізичний і логічний поділ пам'яті команд і пам'яті даних. Ключові команди DSP-процесора є багатоопераднями, і прискорення їх роботи вимагає одночасного читання декількох комірок пам'яті. Відповідно на кристалі є роздільні шини адреси і даних (у деяких типах процесорів - кілька шин адреси і даних). Це дозволяє поєднувати в часі вибірку операндів і виконання команд. Використання модифікованої гарвардської архітектури припускає, що операнди можуть зберігатися не тільки в пам'яті даних, але і в пам'яті команд разом з програмами. Наприклад, у разі реалізації цифрових фільтрів коефіцієнти можуть зберігатися в пам'яті програм, а значення даних - в пам'яті даних. Тому коефіцієнт і дані можуть вибиратися в одному машинному циклі. Для забезпечення вибірки команди в тому ж машинному циклі використовується або кеш-пам'ять програм, або дворазове звернення до пам'яті програм за час машинного циклу.

2. Для скорочення часу виконання однієї з основних операцій цифрової обробки сигналу - множення - застосовується апаратний помножувач. У процесорах загального призначення ця операція реалізується за кілька тактів зсуву і додавання і займає багато часу, а в DSP-процесорах завдяки спеціалізованому помножувачу потрібен всього один цикл. Вбудована схема апаратного множення дозволяє виконати за 1 такт основну операцію ЦГЗ - множення з накопиченням (MultIPly-Accumulate - MAC) для 16 - та / або 32-розрядних операндів.

3. Апаратна підтримка циклічних буферів. Наприклад, для фільтру, при кожному обчисленні відліку вихідного сигналу використовується новий відлік вхідного сигналу, який зберігається в пам'яті на місці старого. Для такого циркулюючого буфера може використовуватися фіксована область ОЗП. При цьому під час обчислень генеруються лише послідовні значення адрес ОЗП незалежно від того, яка операція - запис або читання - виконується в даний момент. Апаратна реалізація циклічних буферів дозволяє встановити параметри буфера (адреса початку, довжина) в програмі поза тілом циклу фільтрації, що дозволяє скоротити час виконання циклічної ділянки програми.

4. Скорочення тривалості командного такту. Це багато в чому забезпечується прийомами, характерними для RISC-процесорів. Головними з них є розміщення операндів більшості команд в регістрах, а також конвеєризація на рівні команд і мікрокоманд. Конвеєр має від 2 до 10 ступенів, що дозволяє на різних стадіях виконання одночасно обробляти до 10 команд. При цьому використовується генерація адрес регістрів паралельно з виконанням арифметичних операцій, а також багатопортовий доступ до пам'яті. Сюди ж можна віднести і такий прийом, характерний для універсальних мікропроцесорів з EPIC-архітектурою , як застосування команд зі надвеликою довжиною слова (VLIW), що генеруються на стадії компіляції програми. Цьому ж служить і розглянута вище Гарвардська архітектура процесора, характерна для однокристальних мікроконтролерів.

5. Наявність на кристалі процесора внутрішньої пам'яті, що ріднить ЦСП з однокристальними МК. Вбудована в процесор пам'ять зазвичай має значно більшу швидкодію, ніж зовнішня. Наявність вбудованої пам'яті дозволяє значно спростити систему в цілому, зменшити її розміри, енергоспоживання і вартість. Ємність внутрішньої пам'яті є результатом певного компромісу. Її збільшення веде до подорожчання

процесора і збільшує енергоспоживання, а обмежена ємність пам'яті програм не дозволяє зберігати складні алгоритми. Більшість DSP-процесорів з фіксованою точкою мають малу ємність внутрішньої пам'яті, зазвичай від 4 до 256 Кбайт, і невисоку розрядність зовнішніх шин даних, що зв'язують процесор з зовнішньою пам'яттю. У той же час ЦСП із плаваючою точкою зазвичай передбачають роботу з великими масивами даних і складними алгоритмами і мають або вбудовану пам'ять великої ємності, або велику розрядність адресних шин для підключення зовнішньої пам'яті (а іноді і те, і інше).

6. Широкі можливості по апаратній взаємодії із зовнішніми пристроями, які включають:

- велику різноманітність інтерфейсів, у тому числі контролери локальної промислової мережі CAN, вбудовані комунікаційні (SCI) і периферійні (SPI) інтерфейси, I2C, UART;
- кілька входів для аналогових сигналів і, відповідно, вбудований АЦП;
- вихідні канали широтно-імпульсної модуляції (ШІМ);
- розвинену систему зовнішніх переривань;
- контролери прямого доступу на згадку.

7. В деяких DSP-сімействах передбачені спеціальні апаратні засоби, що полегшують створення мультипроцесорних систем з паралельною обробкою даних для нарощування продуктивності.

8. DSP-процесори широко використовуються в мобільних пристроях, де споживана потужність є основною характеристикою. Для зниження енергоспоживання в сигнальних процесорах застосовується безліч методів, в тому числі зменшення напруги живлення і введення функцій управління споживанням, наприклад, динамічні зміни тактової частоти, сплячий або черговий режим.

Слід зазначити, що ці заходи роблять значний вплив на швидкість роботи процесора і при некоректному використанні можуть привести до непрацездатності проектованого пристрою (як приклад можна згадати деякі мобільні телефони, які в результаті помилок в програмах управління енергоспоживанням іноді переставали включатися) або до погіршення його експлуатаційних характеристик (наприклад, значного часу відновлення працездатності при виході із сплячого режиму).

Висновки

Головною проблемою процесорів, що реалізовують можливість паралельного виконання команд, являється важкість їх програмування. На жаль, ефективність коду, що транслюється в асемблер з мов високого рівня, поки що поступається коду, який міг би написати безпосередньо на асемблері хороший програміст. Тому для максимального використання можливостей архітектури процесора з метою отримати необхідну продуктивність в завданнях реального часу програмісти вимушені реалізовувати свої алгоритми на асемблері, що збільшує трудомісткість і час роботи над програмою.

У процесорів, що не підтримують паралельні обчислення, немає проблем, перелічених вище, але вони і рідко використовуються для вирішення завдань реального часу, оскільки не здатні забезпечити достатню продуктивність.

Література

1. Jennifer Eyre. The Digital Signal Processor Derby - www.bdti.com

2. The BDTImark 2000 : A Summary Measure of DSP Speed - www.bdti.com

3. Якименко Ю.І. Мікропроцесорна техніка / Ю.І.Якименко, Т.О.Терещенко, Є.І.Сокол, В.Я.Жуйков, Ю.С.Петергеря. -К.: ІВЦ “Видавництво «Політехніка»”, 2004. - 440 с.

4. Грень Я.В. Програмування систем реального часу: Навчальний посібник / Я.В.Грень – Львів: Львівська політехніка, 2011. – 324 с.

Надійшла до редакції
26.1.2013 р.