УПРАВЛІННЯ ТЕХНОЛОГІЧНИМИ ПРОЦЕСАМИ

УДК 681.325.65

В.М.Скіданов, Т.О.Забава

Київський національний університет будівництва і архітектури, Київ

АНАЛІЗ ЗАДАЧ СПІВВІДНОШЕННЯ АПАРАТНИХ І ПРОГРАМНИХ ЗАСОБІВ ПРИ ПОБУДОВІ СИСТЕМ КЕРУВАННЯ АВТОНОМНИМИ ОБ'ЄКТАМИ

Обґрунтовано необхідність оптимізації співвідношення апаратної та програмної складової мікропроцесорних систем керування автономних об'єктів, наведено конкретні приклади, запропоновано підходи до програмної реалізації довільних комбінаційних логічних функцій.

Ключові слова: керування, автономний об'єкт, апаратні та програмні засоби.

Вступ

Широке і практично безкомпромісне застосування мікропроцесорів та мікроконтролерів при реалізації керування технологічними об'єктами дещо знижує „чутливість" розробників до проблеми аналізу раціонального співвідношення апаратних і програмних складових у структурі систем керування. При цьому, якщо для стаціонарних об'єктів такі параметри системи керування, як маса, габарити, потужність споживання тощо не викликають будь-яких занепокоєнь і, можливо, ускладнень, то для автономних об'єктів ці показники стають ледь не вирішальними [1] за умови, зрозуміло, повної функціональної адекватності тих чи інших технічних рішень для тих чи інших об'єктів керування.

Метою даної роботи є аналіз реальних ситуацій, де виникає необхідність обґрунтованого вибору програмної або апаратної домінанти при вирішенні задачі керування автономним об'єктом.

Задача перша. Необхідно реалізувати струмообмеження, наприклад, у силовому контурі приводу, який складається з акумуляторної батареї, напівпровідникового ключа-регулятора (технічно це DC-DC імпульсний транзисторний перетворювач (ІП) понижуючого типу) та тягового електродвигуна. Зрозуміло, що найбільш „тендітним" елементом у цій структурі є саме перетворювач, тому що і батарея, і двигун є більш стійкими до струмових перевантажень. Керуючий алгоритм безумовно містить блоки опитування датчика струму у силовому контурі, його АЦП-перетворення, кількісну оцінку та формування керуючої дії – широтно-імпульсного сигналу з певною довжиною імпульсу. Як показує практика, покрокове проходження програмою даної гілки алгоритму може скласти часовий інтервал від 50 до 100 і більше мікросекунд. Частота комутації у сучасних перетворювачів перевищує 10 кГц, тобто період керуючого широтоно-імпульсного сигналу буде не більшим за 100 мкс. При цьому слід враховувати сталу часу силового L-R кола, яка становить 10-15 мс. Таким чином, адекватна реакція приводу на керуючу дію матиме місце через 30-45мс (три сталі часу). І тільки після цього доцільно виконувати наступне вимірювання поточного струму і формувати нову керуючу дію у відповідності до заданого і поточного стану системи. Це означає, що впродовж сотень (в нашому випадку 300-450) періодів комутації перебіг процесів у силовому колі лишається поза увагою системи керування. І в разі виникнення ситуації стрімкого непрогнозованого зростання струму (вище максимального) цього часу може бути цілком достатньо для незворотної руйнації силових напівпровідникових приладів у імпульсному перетворювачі.

Одним із виходів у такій ситуації, який опробовано на практиці, є зміна тактики регулювання у діапазоні струмів, що становлять 0,8-1,0 від максимально припустимого. Для цього діапазону інтервал опитування струму слід зменшити від трьох сталих часу (реальна тривалість перехідного процесу) до однієї сталої часу. При цьому коливні процеси в системі при зміні керуючої дії будуть суттєво згладжуватись за рахунок інерційності механічних складових приводу, а інтервал некерованості відчутно зменшиться. Однак радикальним вирішенням проблеми, так би мовити, ultima ratio regum («останній довід королів», лат.) є застосування незалежного апаратного контуру струмообмеження, наведеного на рис.1. Тут сигнал від датчика струму порівнюється на компараторі з опорною напругою, яка встановлюється пропорційною граничній величині струму навантаження. Поки поточний струм не перевищує заданого опорною напругою припустимого рівня, компаратор має на виході логічну одиницю і

підтримує елемент „І” у провідному стані, що забезпечує проходження керуючого ШІМ-сигналу до вхідного кола імпульсного перетворювача.

Як тільки струм у силовому колі стане більшим за припустимий, компаратор змінить вихідний сигал на логічний нуль і проходження керуючого сигналу через елемент „І” буде припинено, тобто перетворювач вимкне струм у силовому колі. Час спрацьовування такого захисту у найгіршому випадку не перевищить одиниць мікросекунд, що є гарантованим захистом усього силового контуру.

Таким чином, наявність дворівневого (для діапазонів струму навантаження від 0-0,8IН до 0,8-1,0IН) програмного струмообмеження забезпечує нормальну роботу системи в штатному режимі, а нештатні ситуації будуть гарантовано відпрацьовані зовнішнім незалежним апаратним контуром струмообмеження.

Задача друга виникає у разі необхідності реалізації системою керування певних комбінаційних логічних функцій. Подібні задачі можуть виникати досить часто при формуванні вихідних керуючих сигналів і їх вирішення може виконуватись або синтезом і подальшою побудовою схеми на базі елементів жорсткої логіки [2], або створенням програми для мікропроцесора, послідовне виконання якої дасть необхідний результат − керуючий сигнал чи послідовність сигналів. Зазначимо, однак, що ми не розглядатимемо випадки застосування деяких спеціалізованих контролерів, функції яких дозволяють „вбудовувати” логічні елементи в схемотехнічну конфігурацію системи керування. Проблема в тому, що при створенні засобів керування саме автономними об’єктами виникає необхідність побудови цих засобів за принципом необхідної достатності, тобто з мінімальною кількістю схемотехнічних компонентів за умови гарантованого вирішення задачі керування. При такому підході практично виключається можливість застосування серійних засобів керування з огляду на їх недостатню або ж, навпаки, надлишкову конфігурацію. Тому розглядатимемо найбільш загальний випадок побудови бортових систем керування, де треба враховувати вимоги мінімальної маси, об’єму та енергоспоживання, що можливо при оптимальному співвідношенні апаратних та програмних ресурсів системи.

Перед розробником завжди буде стояти задача: що краще − використати зовнішні (по відношенню до процесора) апаратні елементи і заощадити програмні ресурси (час виконання програми, об’єм пам’яті) чи відмовитись від використання додаткових мікросхем, заощаджуючи місце на платі та потужність споживання і програючи при цьому в раціональному використанні „зайнятості” процесора. Зрозуміло, що однозначної відповіді не дасть жоден фахівець, поки не ознайомиться із специфікою конкретної задачі.

У першу чергу слід чітко визначити пріоритети усіх складових загальної задачі керування, оцінити час виконання кожної гілки алгоритму. І, якщо процесор має „вільний” час, його можна використати для програмної реалізації певних складових керуючої процедури. Якщо технічних ресурсів (за швидкісними показниками) процесору не вистачає, то слід мінімізувати його завантаженість і частину задач перекласти на зовнішні засоби.

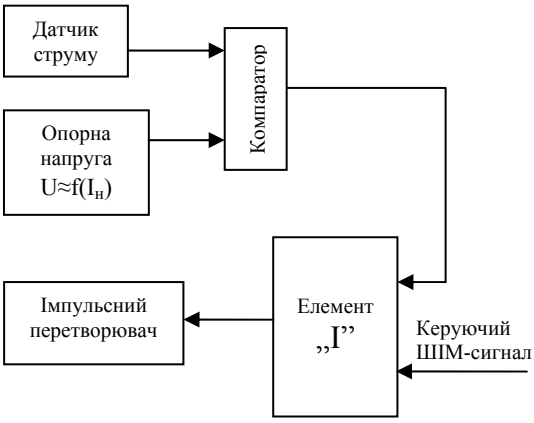

Рис.1. Контур апаратного струмообмеження

Повернемось до задачі реалізації комбінаційної логіки мікропроцесорними системами керування. Реалізація простих комбінаційних функцій типу сума по модулю два, дешифрація двійкового коду, тобто конкретних фіксованих задач, буде виконуватись апаратно. Для більш складних випадків довільної комбінаційної функції, коли варіюється кількість змінних або кількість доданків у диз’юнктивній нормальній формі (ДНФ) функції, доцільною може виявитись її програмна реалізація. Розглянемо деякі прийоми, які дозволяють виконати програмну генерацію комбінаційної логіки.

Виходитимемо з того, що в системі команд будь-якого мікропроцесора є інструкції логічних операцій „І”, „АБО”, „НІ”. Тобто завжди можливо підібрати необхідну послідовність команд для реалізації потрібної логічної функції. Однак не слід орієнтуватись на безпосередню заміну логічних елементів їх програмними аналогами, оскільки такий варіант рішення може бути далекий від оптимального.

Перше, що можна рекомендувати для практичного використання, це послідовна генерація, яка буде ефективною для відносно простих функцій. Для реалізації такого підходу необхідно виконати формування кожного диз’юнктивного члена функції шляхом послідовного виконання операції „І” до кожної змінної (прямої чи інверсної), що йому

належить. При цьому кожна змінна вже занесена до своєї окремої комірки пам'яті. Далі виконується перевірка рівності цього члена одиниці. Якщо рівність має місце, то на виході порту, який відповідає даній функції, формується логічна одиниця і подальша обробка диз'юнктивних складових припиняється. Якщо ж поточний диз'юнктивний член має значення логічного нуля, то починаються формування та аналіз наступного доданку і так далі, поки не будуть проаналізовані усі диз'юнктивні члени функції. Програму слід будувати так, щоб першими аналізувались найбільш прості (за кількістю змінних) диз'юнктивні складові функції, хоча в цілому результат не залежить від порядку формування і аналізу доданків. Такий підхід забезпечує досить просту та швидку генерацію функції з невеликою кількістю змінних та диз'юнктивних складових. Певний недолік полягає в необхідності витрачати на зберігання однієї змінної як мінімум вісім біт пам'яті, крім того, кожна нова функція потребує нової програми.

Далі розглянемо підхід, що буде більш заощадливим, щодо використання програмної пам'яті. Для цього в пам'яті формується таблиця слів, кожне з яких являє собою набір змінних, сформованих у наперед встановленій послідовності. Диз'юнкція усіх табличних слів визначатиме логічну функцію f1. Слово змінних, сформоване на вході (один з портів) системи, порівнюється послідовно з кожним словом таблиці. Якщо слово вхідних змінних співпадає з якимось табличним словом, на відповідному функції f1 виході формується одиниця.

Математичне обґрунтування ще більш ефективного підходу можна представити таким чином. Для порівняння слів A = am … a1a0 та B = bm … b1b0 необхідно реалізувати функцію рівнозначності

$$F_1 = \left(a_m b_m \vee \bar{a}_m \bar{b}_m\right) \times \ldots \times \left(a_0 b_0 \vee \bar{a}_0 \bar{b}_0\right).$$

Якщо змінні у кожному розряді рівні, то F1=1, тобто слово A співпадає зі словом B. Системи команд більшості промислових контролерів мають в своєму складі операцію „виключальне АБО" (команди типу XRL), яка виконується порозрядно, тобто між однойменними розрядами двох слів виконується перетворення, яке відповідає функції

$$F_2 = \left(a_m \bar{b}_m \vee \bar{a}_m b_m\right) \times \ldots \times \left(a_0 \bar{b}_0 \vee \bar{a}_0 b_0\right),$$

яка є функцією нерівнозначності. Відомо, що $F_2 = \bar{F}_1$, тобто, якщо A = B, то F2=0, а наявність в системі команд умовного переходу по нулю дозволяє організувати програмний цикл послідовного порівняння.

Однак перш ніж розпочати безпосередньо порівняння, необхідно врахувати такі фактори: в загальному випадку функція F1 є сумою мінтермів, кожен з яких містить усі змінні (прямі та інверсні). Після мінімізації деякі диз'юнктивні члени будуть мати не всі змінні, тобто зміниться довжина слова, а даний підхід передбачає закріплення за кожною змінною одного певного розряду. Тому операції порівняння повинна передувати операція маскування, результатом якої буде виключення (тут не видалення, а так би мовити - нейтралізація) зі слова змінних „небажаних" або „байдужних" розрядів шляхом примусової установки їх в нуль для збереження довжини слова. Наприклад, при склеюванні (операція логічного додавання) слів $\bar{a}\bar{b}c$ і $\bar{a}bc$ отримаємо $\bar{a}c$, а для збереження фіксованої довжини слова запишемо результат як $\bar{a}xc$, де x позначає байдужний розряд, який таким чином маскується. Для операції маскування виконуємо порозрядно операцію „I" з нулем для байдужних розрядів та з одиницею для інших розрядів. Будується таблиця, яка складається з масок і слів змінних. Таблицю доцільно розмістити у стеку (область основної пам'яті), тоді адреса першої комірки задаватиметься регістром ВВС – вказівником верхівки стеку. В таблиці спочатку розміщується маска, а потім відповідне слово змінних; маска з нулів позначає кінець таблиці. Дані зі стеку можуть передаватись в операційні регістри процесора, при цьому ВВС завжди показуватиме адресу нової верхівки стеку, тобто першого із слів, що залишились.

Для функції восьми змінних F1 = φ(a,b,c,d,e,k,m,n), яка після мінімізації має кінцевий вигляд $F_1 = a\bar{b}cd \vee ac\bar{d} \vee \bar{a}bd \vee \bar{c}$ початковий стан стеку буде заданий у вигляді табл.1.

Безсумнівною перевагою запропонованого підходу є можливість використання однієї програми для реалізації декількох різних функцій. Припустимо, що крім функції F1, яка представлена у табл.1, необхідно реалізувати ще ряд функцій: F2, F3, F4. Для цього в стеку сформуємо таблиці (аналогічні табл.1) для відповідних функцій та додаткову табл.2, де для кожного масиву функцій F2, F3, F4 вказується адреса початку його первинної табличної реалізації (відповідно табл.1-F2, ..., 1-F4). Основній програмі лишається ідентифікувати функцію, яка підлягає реалізації. Для економії пам'яті код функції доцільно задавати таким, щоб він співпадав з адресою початку таблиці для цієї функції. Якщо це за якихось причин неможливо, то можна організувати непряму (посередню) адресацію. Слід також відрізняти код функції від слова змінних, яке вводиться в пам'ять у першу чергу (передбачається, що всі функції, які

реалізуються, визначаються одним набором змінних).

Коли число змінних перевищує розрядність процесора, слід розділити змінні за байтами. Таблиця типу табл.1 для довільної функції Fi формується традиційно з тією лише різницею, що кожний диз'юнктивний член функції Fi представляється двома байтами, кожен з яких має поперед себе відповідну маску.

Таблиця 1

Формування функції F1

Адреса	Змінні nmkedcba
100	00001111
101	xxxx1101
102	00001101
103	xxxx01x1
104	00001011
105	xxxx1x10
106	00000100
107	xxxxx0xx
108	00000000
109	xxxxxxxx

Таблиця 2

Формування масивів функцій

Адреса початку	Вміст
100	Масив F1
200	Масив F2
300	Масив F3
400	Масив F4

Якщо якийсь доданок містить змінні лише в одному байті, то другий байт може бути довільним, але перед ним формується маска з нулів. Так, для функції $F = ab\bar{c}dkp \vee aem \vee k\bar{l}n\bar{p}$ треба сформувати масив, наведений в табл..3. Слово вхідних змінних записується в пам'ять за два кроки і з різних пристроїв вводу-виводу у строгій відповідності до розподілу змінних за байтами. За необхідності можна продовжити нарощування розрядності функцій, що реалізуються, хоча це може суттєво ускладнити програму і збільшити час її виконання.

На закінчення розглянемо програмну реалізацію багатовихідних комбінаційних логічних функцій, де на виході треба мати декілька керуючих сигналів. Для генерування багатовихідної (з кількістю виходів $\leq$ 8 для восьмирозрядних мікропроцесорів і мікроконтролерів) функції слово вхідних змінних доцільно розглядати як адресу комірки пам'яті, в якій зберігається інформація, що являє собою вісім вихідних величин. Це найпростіший і найшвидший спосіб. Однак, якщо його реалізація ускладнюється через зайнятість пам'яті з потрібними адресами, можна скористуватись непрямою адресацією. Для цього кодові комбінації, які задають значення багатовихідної функції, треба розмістити в одній групі комірок пам'яті, адреси яких розподіляються по іншим коміркам у такий спосіб, щоб адреси останніх у цифровому виразі відповідали значенням, які визначають дані вхідні комбінації змінних багатовихідної функції. Восьмирозрядне вхідне слово змінних забезпечує адресацію 256 можливих значень восьмивихідної функції.

Таблиця3

Формування двобайтної функції

Адреса	Змінні mlkedcba - - - - - - np
100	00101111
101	xx1x1011
102	00000010
103	xxxxxx1x
104	10010001
105	1xx1xxx1
106	00000000
107	xxxxxxxx
108	01100000
109	x01xxxxx
110	00000011
111	xxxxxx01
112	00000000
113	xxxxxxxx
114	00000000
115	xxxxxxxx

Якщо запропонований підхід використовувати при реалізації одновихідних функцій, то для заощадження пам'яті стани виходів слід розмістити не в 256, а в 32 словах (комірках). При цьому п'ять молодших розрядів вхідного слова змінних визначають одне з цих тридцяти двох слів, а три старших розряди вибирають потрібний біт вихідного слова. Зауважимо, що застосування логічних операцій зсуву може підвищити ефективність програмної процедури.

Висновки

Розглянуті загальні теоретичні положення та практичні приклади є безумовним підтвердженням необхідності ретельного аналізу варіантів вирішення задач раціональної побудови засобів керування автономними об'єктами, особливо, коли результат спрямований на підвищення технічної безпеки та надійності системи в цілому (перша задача) або на оптимальне співвідношення апаратних і програмних ресурсів (друга задача) для мінімізації зовнішніх схем і часу виконання керуючої процедури.

Список літератури

1. *Полупроводниковые преобразователи в автономном электроприводе постоянного тока /В.Б. Павлов, А.К.Шидловский, В.М.Скиданов, В.А.Рычков. - К.: Наук.Думка, 1987. - 284 с.*
2. *Лазарев В.Г., Пийль Е.И. Синтез управляющих автоматов/В.Г. Лазарев, Е.И. Пийль. - М.: Энергия, 1978.- 408с.*

Стаття надійшла до редколегії: 16.02.2010

Рецензент: канд. техн. наук, доц. С.В. Іносов, Київський національний університет будівництва і архітектури, Київ