Алгоритмическая многозадачность в микроконтроллерных разработках

Если я поднимусь на тот холм, я увижу сразу весь сад. А вот и тропинка, она ведет прямо наверх... Нет, совсем не прямо... Надеюсь, она приведет меня все же наверх... Как она кружит! Прямо штопор, а не тропинка. Пойду-ка я лучше назад.

Льюис Кэрролл, «Алиса в Стране Чудес»

В статье представлен один из вариантов алгоритма параллельного выполнения нескольких задач для микроконтроллерных систем. Также приведен макет программы для микроконтроллеров фирмы Microchip.

Алексей Узенгер

uzenger@chat.ru

Мы постоянно наблюдаем за бурным развитием микропроцессорной техники. Разработчики микроконтроллеров (МК), наращивая вычислительный потенциал своих микросхем, расширяют области их применения. Еще недавно программа, написанная для МК, размещалась в одном файле, так как ее структура была обычно несложной — основной программный модуль, выполняемый в цикле, и процедуры для обработки прерываний. И если программа увеличивалась в размерах, то нередко становилась загадкой для самого программиста, а постороннему человеку разобраться с ней вообще не представлялось возможным.

С расширением аппаратных и вычислительных возможностей МК актуальными становятся задача усложнения структуры программы и данных и проблема параллельности выполнения нескольких задач. Понятие «многозадачность» и многопоточность», тесно связанное с программными продуктами для компьютеров, по наблюдениям автора, мало распространено в области программирования МК.

В системах автоматизации технологических процессов часто используются контроллеры, задачи которых предельно просты: например, зафиксировать значение сигнала с АЦП и по запросу от управляющего компьютера передать по последовательному протоколу информацию. Такие системы, как правило, осуществляют информационный сбор данных. Всю основную работу по обработке информации выполняет компьютер либо оператор, что встречается чаще, а контроллеры выполняют одну конкретную задачу. Наше внимание будет уделено МК, которые действуют локально в системе (приборе) и выполняют целый набор задач — от мгновенного принятия решения по каким-либо входным параметрам до выполнения сервисных функций, некритичных ко времени выполнения, например таких, как отображение информации на ЖКИ.

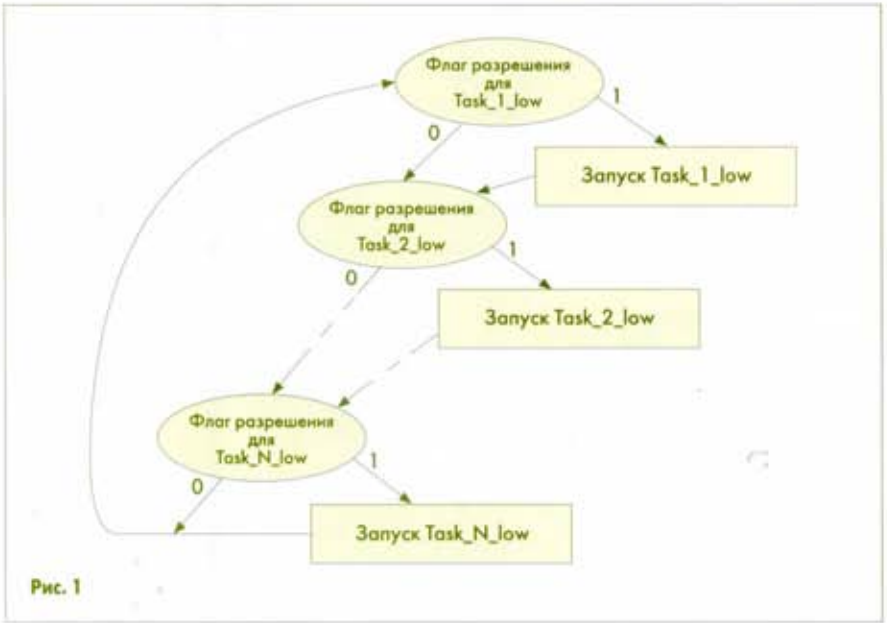

Рис. 1

Начнем с определения термина «задача». Под «задачей» будем понимать функционально и логически законченную последовательность действий (команд) МК. Например, первая задача читает и анализирует аналоговые выводы МК, вторая — выводит информацию на ЖКИ, третья — опрашивает клавиатуру (цифровые выводы) и т. п.

Программирование на ассемблере сложное и трудоемкое занятие, если не сформулировать для себя правила программирования, которые хочется не хочется, а выполнять нужно. Поэтому рассмотрим некоторые ограничения для каждой задачи:

- четко определенная структура поведения (например, никаких команд безусловного перехода на метку другой задачи);
- определен приоритет выполнения задачи в общем ансамбле;
- если задача неактивна, то она не влияет на выполнение активных задач…

Список ограничений можно продолжить в зависимости от сложности задач и требований к системе.

Задачи мы будем подразделять на два типа. Первый тип — высокоскоростные (high) задачи, которые выполняются за относительно короткий промежуток времени. Они имеют наивысший вектор приоритетов по сравнению со вторым типом, критичны к моменту запуска и ко времени выполнения. Высокоскоростные задачи строго привязаны к прерываниям аппаратных модулей МК — например, обслуживание прерывания по фронту вывода порта МК.

Ко второму типу относятся низкоскоростные (low) задачи, которые выполняются за длительный промежуток времени. Они имеют более низкий вектор при-

оритетов по сравнению с высокоскоростными и не критичны к моменту запуска и времени выполнения. Задачи этого типа строго не привязаны к аппаратным прерываниям МК. Их выполнение может разрешаться как аппаратными средствами МК, так и программными модулями выполняемой программы (например, обслуживание ЖКИ).

По какому критерию можно определить тип задачи? Как один из вариантов, задачи можно разделить по времени их выполнения. Так, например, пусть МК фирмы Microchip тактируется от кварца 4 МГц. Время выполнения одной команды составляет 1 мкс (4 периода кварцевого резонатора). Если задача состоит из десятков команд, то есть до 0,1 мс, то ее можно отнести к высокоскоростным. Следовательно, если задача выполняется за время большее, чем 0,1 мс, то ее можно обозначить как низкоскоростную. Для каждого конкретного случая разделение задач будет индивидуальным. Может так случиться, что время выполнения процедуры будет менее 0,1 мс, а ее лучше отнести к низкоскоростным. Также не стоит создавать слишком длительных по времени задач. Лучше либо разделить их на несколько подзадач, либо пересмотреть структуру всей задачи. Возможно, для них лучше использовать временные метки от одного из таймеров МК. С увеличением частоты тактируемого кварца, соответственно, будут изменяться требования к времени выполнения задач.

На рис. 1 представлена структура поведения основного модуля программы. Очевидно, что все задачи, которые заполняют основной модуль, относятся к низкоскоростным. Каждая из них выполняется только тогда, когда будет установлен флаг, разрешающий ей выполняться. Например, если по какому-либо событию одновременно выставились два флага, то выполнение их будет происходить строго в определенной последовательности, определяемой по приоритетам. Приоритет выполнения убывает в данном случае с увеличением порядкового номера задачи.

На рис. 2 как один из вариантов представлено поведение модуля, обслуживающего прерывание. Прерывание выполняет j-ю высокоскоростную задачу и разрешает/запрещает выполнение i-й низкоскоростной задачи. Речь идет об одном флаге разрешения и о выполнении одной задачи, что совсем не обязательно — их может быть и несколько.

Приведем пример реализации многозадачной системы на МК фирмы Microchip. Пусть нам необходимо параллельно выполнять шесть задач — три высокоскоростные и и три низкоскоростные.

Task_A_high выполняется мгновенно по прерыванию от нулевого таймера. Разрешение на выполнение данной задачи происходит сразу же после инициализации системы, т. е. изначально заложено, что задача будет выполняться каждый раз после прерывания нулевого таймера.

Task_B_high и Task_C_high при наличии разрешающего флага выполняются по прерыванию от первого таймера. Задача Task_B_high имеет больший приоритет по сравнению с Task_C_high. Изначально заложено, что данные задачи игнорируются системой. По мере необходимости их можно подключать и, следовательно, отключать.

Task_A_low выполняется по прерыванию от нулевого таймера. Причем выполняется не сразу, а по мере освобождения МК ядра от прерываний.

Разрешение на выполнение задачи Task_B_low происходит сразу же после инициализации системы. То есть изначально заложено, что задача будет выполняться циклически, пока флаг разрешения на выполнение будет установлен.

Задача Task_C_low изначально неактивна. По мере необходимости ее можно будет включить на выполнение.

```
taskVector  EQU 0x20
; — флаги разрешений на прерывание -
#define  bTask_A_high  taskVector, 0
#define  bTask_B_high  taskVector, 1
#define  bTask_C_high  taskVector, 2
#define  bTask_A_low   taskVector, 3
#define  bTask_B_low   taskVector, 4
#define  bTask_C_low   taskVector, 5
; — модули,
; в которых расписан код задач
#include  «init.inc»
#include  «taskA_hi.inc»
#include  «taskB_hi.inc»
#include  «taskC_hi.inc»
#include  «taskA_lo.inc»
#include  «taskB_lo.inc»
#include  «taskC_lo.inc»
  Org  0x00
  Goto  Main
  Org  0x04
;[команды входа в прерывание]
; — прерывание от 0 таймера -
  btfss  INTCON, T0IF
  goto  int0_end
  bcf  INTCON, T0IF
  bsf  bTask_A_low
  btfss  bTask_A_high
  goto  $+3
  Call  Task_A_high
  bcf  bTask_A_high
int0_end
; — прерывание от 1 таймера -
  btfss  PIR1, TMR1F
  goto  int1_end
  bcf  PIR1, TMR1F
  btfss  bTask_B_high
  goto  $+3
  Call  Task_B_high
  bcf  bTask_B_high
  btfss  bTask_C_high
  goto  $+3
  Call  Task_C_high
  bcf  bTask_C_high
int1_end
;[команды выхода из прерывания]
  Retfie
Main:
  Call  Init
  bsf  bTask_A_high
  bsf  bTask_B_low
begin
  btfss  bTask_A_low
  goto  $+3
  Call  Task_A_low
  bcf  bTask_A_low
  btfss  bTask_B_low
  goto  $+3
  Call  Task_B_low
  bcf  bTask_B_low
  btfss  bTask_C_low
  goto  $+3
  Call  Task_C_low
  bcf  bTask_C_low
  goto begin
  END.
```

Рис. 2

Приведенный выше пример иллюстрирует простую реализацию выполнения нескольких задач на МК. Ничего сверхсложного в нем нет. Параллельное выполнение задач можно было бы реализовать и в одном основном модуле и система работала бы. А если число задач увеличится до двадцати, например, то введение данной структуры себя оправдывает. При желании можно развить идею «многозадачности» для МК, попробовать сымитировать возможность изменения приоритетов и т. п. На вопрос «а стоит ли?» у автора ответа нет. Можно утверждать только то, что, понимая логику работы МК и, соответственно, логику программы, которую сам создаешь, можно красиво выполнить любое техническое задание и