Реализация многозадачного проекта на микроконтроллере ARM7 без использования операционных систем

Константин Оськин (г. Пермь)

В статье описывается способ реализации переключения задач встроенной программы микроконтроллера ARM7 без использования операционной системы. Рассмотрены особенности переключения задач, в том числе организация пятизадачного приложения с использованием карусельного механизма переключения задач.

Использование встроенных операционных систем при разработке приложений для микроконтроллеров предоставляет разработчику набор средств для конфигурирования временных характеристик программы микроконтроллера. Наличие стандартной отлаженной системы позволяет быстро построить новый проект и разграничить в нём выполнение таких задач, как информационный обмен, аналого-цифровое и цифроаналоговое преобразование, вычисление управляющих воздействий по входным параметрам, архивация данных и др. Тем не менее, на ранних этапах освоения новых семейств микроконтроллеров возникает проблема с выбором и освоением новых операционных систем. В данной статье представлен способ организации многозадачных приложений для микроконтроллеров ARM7 без использования встроенных операционных систем. Сам механизм рассмотрен на примере микроконтроллера LPC2134, однако технология применима к любому микроконтроллеру семейства ARM7.

Рассмотрим самый простой механизм переключения задач операционной системой. По возникновению сигнала операционная система определяет источник, требующий передачи управления, определяет его приоритет и, если приоритет источника выше, чем приоритет текущего процесса, передаёт ему управление. Процесс, прерываемый новой задачей, назовём фоновым процессом, а процесс, прервавший её, – приоритетным. При передаче управления операционная система сохраняет контекст прерванной задачи и точку, куда следует возвратиться после окончания выполнения приоритетной задачи.

Организация режимов работы ядра ARM7 достаточно подробно рассмотрена в [1–3]. Не повторяясь, отметим следующее. Переход в режимы FIQ, IRQ осуществляется по запросу от источника прерывания. Поскольку эти режимы являются привилегированными по отношению к пользователю (User), в них доступно обращение к битам режима регистра состояния программы CPSR с последующим переключением режима. Кроме того, каждый режим имеет свой собственный стек, регистр указателя вершины стека и регистр связи, которые автоматически обновляются и переключаются при смене режима работы ядра. Следует уточнить, что в режиме System ядро оперирует данными пользовательского режима, т.е. стек и все регистры R0–R14 у них общие.

Теперь рассмотрим состояние ядра при переходе в режим, например, быстрого прерывания FIQ. Этот режим представляется автору оптимальным, т.к. обеспечивает наименьшее время реакции на прерывания по каналу FIQ. Согласно описаниям [1, 2], программа, переключившись в режим быстрого прерывания и сохранив следующую команду, переходит на вектор быстрого прерывания из таблицы векторов. Авторы упомянутых книг и разработчики систем программирования для микроконтроллеров ARM7 предлагают стартовый код, приведённый в листинге 1:

Листинг 1. «Стандартный» стартовый код ARM7

```
Vectors LDR     PC, Reset_Addr
LDR     PC, Undef_Addr
LDR     PC, SWI_Addr
LDR     PC, PAbt_Addr
LDR     PC, DAbt_Addr
NOP ; Reserved Vector
LDR     PC, IRQ_Addr
LDR     PC, [PC, #-0x0FF0]; Vector
from VicVectAddr
LDR     PC, FIQ_Addr
Reset_Addr DCD Reset_Handler
Undef_Addr DCD Undef_Handler
SWI_Addr DCD SWI_Handler
PAbt_Addr DCD PAbt_Handler
DAbt_Addr DCD DAbt_Handler
DCD 0 ; Reserved Address
IRQ_Addr DCD IRQ_Handler
FIQ_Addr DCD FIQ_Handler

Undef_Handler B Undef_Handler
SWI_Handler B SWI_Handler
PAbt_Handler B PAbt_Handler
DAbt_Handler B DAbt_Handler
IRQ_Handler B IRQ_Handler
FIQ_Handler     B
FIQ_Handler
```

Здесь все переходы надёжно заглушены инструкциями типа FIQ_HandlerB FIQ_Handler. Попав в эту заглушку, ядро содержит в регистрах R0–R8 данные прерванной программы. Значения этих регистров можно назвать контекстом прерванного процесса. Напомним, что согласно стандарту, рекомендованному фирмой ARM, регистры R0–R3 (a1–a4) должны использоваться для передачи параметров подпрограмм и функций, а R4–R11 (v1–v8) – для хранения локальных переменных [1].

Собственный обработчик прерывания должен выполнить обязательные действия – снять флаги прерывания и определить точку перехода на новую задачу. Обработчик не обязательно писать на ассемблере в коде запуска. Достаточно совершить переход на Си-функцию. При этом следует помнить, что не все инструкции перехода сохраняют адрес возврата в регистре связи. Также перед переходом на Си-функцию необходимо сохранить на стеке весь контекст прерванного процесса вместе с регистром связи, чтобы после выхода из Си-функции их можно было восстановить.

После выхода из функции следует подменить значение, хранящееся в регистре связи, новой точкой перехода, полученной в функции, и совершить выход из обработчика прерывания, сохранив при этом контекст прерванного процесса. Проблема реализации этих действий заключается в том, что после выхода из об-

© СТА-ПРЕСС

работчика ядро меняет режим работы, а соответственно, и стек. Таким образом, возврат из приоритетной задачи, выполняемой в пользовательском режиме, будет затруднён отсутствием на стеке контекста прерванного процесса. Существует множество решений такой проблемы. По мнению автора, проще всего перейти в режим пользователя до выхода из обработчика прерывания и сохранить на стеке контекст. В листинге 2 приведён код программы обработки прерывания FIQ:

Листинг 2. Обработка прерывания FIQ

```
IMPORT FIQ_rtn ;СИ-подпрограмма обработки прерывания
IMPORT ptp          ;точка перехода
IMPORT rp           ;точка возврата
FIQ_Handler
        STMDB   R13!, {R0-R8, R14}
        BL              FIQ_rtn
;СИ-функция
        LDMIA   R13!, {R0-R8, R14}
;восстанавливаем исходное состояние стека
        LDR     R9, =rp
        STR     LR, [R9, #0]!
;Сохранили точку возврата прерванного процесса, R9
        MSR             CPSR_c,
#Mode_USR       ;переход в user
        STMDB   R13!, {R0-R8}
;сохраняем стек для перехода на ptp
        LDR             R9, =rp
        LDR             LR, [R9]
        STMDB   SP!, {LR}
;Помещаем точку возврата прерванного режима на стек
        LDR             R9, =ptp
        LDR             LR, [R9]
;СОХРАНЯЕМ ТОЧКУ ПЕРЕХОДА
        SUBS            PC, LR, #0
```

Переход в режим пользователя и выполнение в нём, а не в FIQ, приоритетной задачи позволяет устранить одно досадное недоразумение, присущее ARM7, – отсутствие вложенных прерываний. Оставшись в режиме FIQ, мы не только пропустим вызовы FIQ, которые могут поступить за время выполнения новой задачи, но и заблокируем обработку всех остальных источников прерываний IRQ. Но после перехода в режим пользователя новая задача может быть также прервана новым прерыванием по каналу FIQ или IRQ. Более того, задача может быть вызвана рекуррентно или может быть прервана более приоритетной задачей.

После завершения работы приоритетной задачи потребуется переключиться на прерванный процесс. Контекст этого процесса как раз сохранён на стеке нашего режима. Таким образом, возврат к прерванному процессу может быть осуществлён, как показано в листинге 3:

Листинг 3. Выход из приоритетной задачи

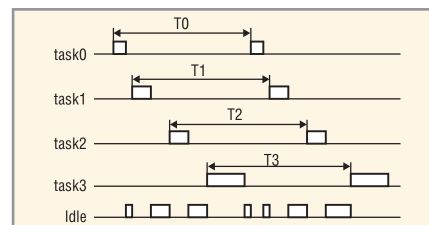
```
EXPORT TASK_OUT
TASK_OUT
        LDMIA   SP!, {LR}
;Восстановили точку возврата
        LDMIA   SP!, {R0-R8}
;Восстановили контекст прерванного процесса
        SUBS    PC, LR, #0x04
;Возврат к прерванному процессу
```

Таким образом, в обратном порядке со стека восстанавливаются точка возврата и контекст прерванного процесса. Если завершить Си-функцию процесса процедурой TASK_OUT, мы вернёмся к команде, на которой было прервано выполнение процесса. Но, поскольку возврат к прерванному процессу происходит при помощи команды SUBS PC, LR, #0x04, объявленная нами функция процесса завершится нештатно с точки зрения компилятора. Иными словами, любой компилятор перед вызовом функции поставит команду сохранения на стеке требуемых данных, а перед выходом из неё – восстановит их обратно. Поэтому на момент вызова TASK_OUT на стеке будут храниться некоторые данные, которые следует сбросить в самом начале выполнения процедуры TASK_OUT. Какие именно данные оставил на стеке компилятор, можно посмотреть в дизассемблированном коде программы.

Представленный механизм можно реализовать, например, в карусельном (round-robin) переключении задач. Для этого назначим прерыванию от системного таймера канал FIQ и настроим его на прерывания от регистров совпадения. В листинге 4 продемонстрирован механизм переключения задач на языке Си:

Листинг 4. Пример передачи управления задачам

```
uint32_t ptp, ptp0, ptp1, ptp2,
ptp3, rp;       // Точки входа
приоритетных задач
void T0isr(void){
        if (T0TC >= T0MR0){
                T0MR0 = T0TC + T0;
                ptp = ptp0;
        }
        if (T0TC >= T0MR1){
                T0MR1 = T0TC + T1;
                ptp = ptp1;
        }
        if (T0TC >= T0MR2){
                T0MR2 = T0TC + T2;
                ptp = ptp2;
        }
        if (T0TC >= T0MR3){
                T0MR3 = T0TC + T3;
                ptp = ptp3;
        }
        T0IR |= 0x000000FF;
// Сбрасываем флаг прерывания
}
void init_tasks (void){
        ptp0 = (unsigned)TASK0;
        ptp1 = (unsigned)TASK1;
        ptp2 = (unsigned)TASK2;
        ptp3 = (unsigned)TASK3;
}
```



Временная диаграмма переключения задач

Каждая задача будет включаться с заданным периодом. Временная диаграмма работы программы представлена на рисунке. Этот лабораторный пример можно усложнить назначением приоритета каждой задаче, блокировками прерываний подпрограмм обработки прерываний и т.д.

Однако предложенный способ имеет ряд недостатков. Во-первых, отсутствуют механизмы приостановки задачи, т.е. каждая новая задача будет запускаться с начала и выполняться до конца. Второй недостаток заключается в том, что данные и точки возврата разных задач приходится хранить на одном и том же стеке. Тем не менее, как было показано выше, предложенный механизм позволяет достаточно просто переключать циклически возникающие задачи.

Литература

1. *Редькин П.П.* Микроконтроллеры ARM7 семейства LPC2000. Руководство пользователя. Додэка-XXI, 2007.
2. *Тревор М.* Микроконтроллеры ARM7. Семейство LPC2000 компании Philips. Додэка-XXI, 2006.
3. LPC2131/2//4/6/8 User manual, Rev.02, www.nxp.com/documents/user_manual/UM10120.pdf.

© СТА-ПРЕСС