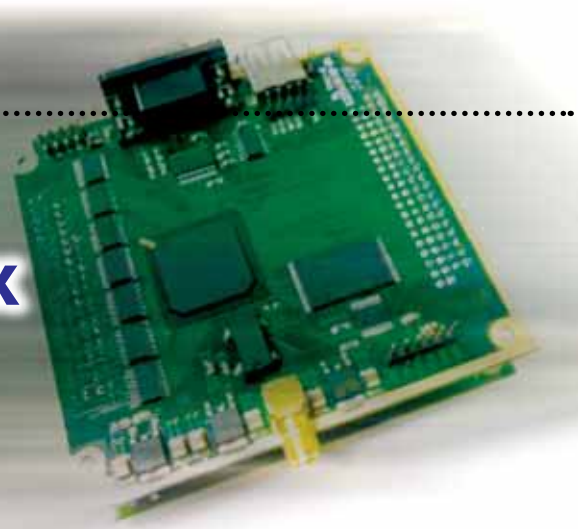

РЕАЛИЗАЦИЯ МНОГОЗАДАЧНЫХ ПРИЛОЖЕНИЙ НА МК СЕРИИ 1886

С. Гусев, gusev.stas@ic-design.ru
С. Шумилин, shumilin.sergei@ic-design.ru

Компания ЗАО "ПКК Миландр" занимается разработкой отечественных микросхем для аппаратуры специального назначения, в том числе для нужд Минобороны. В настоящее время разработан и освоен в серийном производстве широкий ассортимент изделий, включенных в перечень МОП, среди которых – микроконтроллеры, схемы статической памяти, интерфейсные, радиочастотные и многие другие. Все разрабатываемые микросхемы отвечают жестким требованиям, предъявляемым к специальной элементной базе. Помимо самих микросхем, большое внимание уделяется также средствам разработки и технической поддержке потребителей. Для этого компания разрабатывает и поставляет демонстрационно-отладочные комплекты, примеры программ и программные средства для разработки приложений.

В 2009 году компания ЗАО "ПКК Миландр" приобрела лицензию у норвежской компании B Knudsen Data (www.bknd.com) на право распространения компилятора CC7A, предназначенного для разработки приложений для микроконтроллеров серии 1886 на языке Си.

Серия микроконтроллеров 1886ВЕ содержит микроконтроллеры для различных применений. Так, первые микроконтроллеры 1886ВЕ1 и 1886ВЕ2 разрабатывались в рамках ОКР по импортозамещению микроконтроллеров PIC17C756 фирмы Microchip. Микроконтроллер 1886ВЕ3 предназначен для создания защищенных криптографических USB-накопителей и имеет блок аппаратной поддержки криптографического алгоритма (ГОСТ 28147-89), контроллер USB-интерфейса, контроллер внешней NAND флеш-памяти и SD/MMC-карт памяти. Микроконтроллер 1886ВЕ4 предназначен для реализации USB-интерфейса в различных системах связи и управления. Микроконтроллер 1886ВЕ5 имеет в своем составе контроллер CAN- и LIN-интерфейсов и применяется в автомобильной технике. Микроконтроллер 1886ВЕ6 содержит 12-разрядные ЦАП и АЦП, а также схему аппаратного компаратора, схемы ШИМ и регистрации событий и предназначен для различных аналоговых систем. Микроконтроллер 1886ВЕ7 разработан по технологии сверхмалого потребления, выпускается в миниатюрном корпусе и предназначен для систем с батарейным и критичным к уровню потребления питанием. Основные характеристики микроконтроллеров серии 1886 представлены в табл.1.

Компилятор CC7A поставляется в двух вариантах: полный (поставляется отдельно) и урезанный(поставляется вместе со средой IDE). Оба варианта поддерживают все инструкции серии 1886 и генерируют оптимальный ассемблерный код.

Полная версия компилятора поддерживает 8-, 16-, 24- и 32-бит знаковые и беззнаковые переменные и переменные типа bit. Компилятор позволяет использовать структуры, объединения, битовые поля, переопределение типов, массивы и указатели.

Возможна работа с переменными с плавающей запятой. Коды библиотечных функций быстрые и компактные, а генерируемый код можно оптимизировать как по размеру, так и по скорости.

Табл.2 демонстрирует основные различия полной и урезанной версий компилятора CC7A.

ПРИНЦИП МНОГОЗАДАЧНОСТИ

Традиционные программы строятся по принципу последовательного выполнения команд с прерываниями и переходами. Подобные приложения часто помещаются в одном файле, содержащем закольцованную основную функцию и небольшой набор процедур, последовательно вызываемых из основной функции. Подобный подход пригоден для небольших приложений. По мере роста размера и сложности кода программы возникает необходимость в специальных подходах к организации структуры приложений. При этом, если удается из общего кода программы выделить отдельные, независящие друг от друга задачи, такое приложение можно организовать по принципу многозадачности.

Код программы при таком подходе разбивается на несколько независимых друг от друга частей, которые называются задачами или процессами. Ядро многозадачности или некоторый его аналог, содержащийся обычно в ос-

Таблица 1. Характеристики микроконтроллеров серии 1886

Параметр	1886BE1	1886BE2	1886BE3	1886BE4	1886BE5	1886BE6	1886BE7
Процессор	8-бит RISC	8-бит RISC	8-бит RISC	8-бит RISC	8-бит RISC	8-бит RISC	8-бит RISC
Тактовая частота	33 МГц	33 МГц	33 МГц	33 МГц	35 МГц	40 МГц	10 МГц
Питание ядра	4,5–5,5 В	4,5–5,5 В	4,5–5,5 В	4,5–5,5 В	4,5–5,5 В	4,5–5,5 В	4,5–5,5 В
Питание портов	4,5–5,5 В	4,5–5,5 В	3,0–5,5 В	3,0–5,5 В	4,5–5,5 В	4,5–5,5 В	4,5–5,5 В
Память программ	Mask ROM 32K×16	FLASH 32K×16	FLASH 32K×16	FLASH 32K×16	EEPROM 4K×16	EEPROM 4K×16	EEPROM 2K×16
ОЗУ	902×8	902×8	902×8	902×8	902×8	902×8	902×8
Память данных	Нет	Нет	EEPROM 256 байт	EEPROM 256 байт	EEPROM 256 байт	EEPROM 256 байт	EEPROM 256 байт
Таймеры	4	4	1	1	3	3	1
Схема захвата	4	4	Нет	Нет	2	2	–
ШИМ	3	3	Нет	Нет	2	2	–
АЦП	12 каналов/ 10 разрядов	12 каналов/ 10 разрядов	Нет	Нет	8 каналов/ 10 разрядов	8 каналов/ 10 разрядов	Нет
ЦАП	Нет	Нет	Нет	Нет	Нет	12 разрядов/ 2 канала	Нет
Компаратор	Нет	Нет	Нет	Нет	Нет	Есть	Нет
USART	2	2	1	1	1 + LIN	2 + LIN	1 + LIN
SPI	1	1	Нет	1	Нет	Нет	Нет
I2C	1	1	Нет	Нет	Нет	Нет	Нет
USB	Нет	Нет	2 точки	4 точки	Нет	Нет	Нет
CAN	Нет	Нет	Нет	Нет	6 буферов RX/TX	Нет	Нет
Специальные блоки	–	–	Блок поддержки ГОСТ 28147-89	–	–	–	–
Встроенный регулятор напряжения	–	–	$U_{вх}$ = 5 В $U_{вых}$ = 3,3 В $I_{вых}$ = 40 мА	$U_{вх}$ = 5 В $U_{вых}$ = 3,3 В $I_{вых}$ = 40 мА	–	–	$U_{вх}$ = 5В...16 В $U_{вых}$ = 5 В $I_{вых}$ = 10 мА
Температура	-60–85°C	-60–85°C	-60–85°C	-60–85°C	-60–85°C	-60–125°C	- 60–125°C
Тип корпуса	H18.64	H18.64	H16.48 /LQFP 64	H16.48 /LQFP 64	H14.42	H16.48	H09.28
Статус	ОКР сдана	ОКР сдана	ОКР сдана	ОКР сдана	Опытные образцы	Опытные образцы	Опытные образцы

новной функции приложения, определяет, которая из задач будет выполняться следующей. Разрабатывать код для каждой задачи можно независимо от разработки всего остального приложения. Таким образом, достигается модульность кода программы, что существенно упрощает процесс проектирования.

Одним из наиболее распространенных на сегодняшний день алгоритмов распределения вычислительных ресурсов является выполнение каждого процесса в течение определенного промежутка времени. Такой метод, называемый разделением по времени, обладает одним важным недостатком – необходимостью сохранять контекст каждой задачи в момент переключения. Эта особенность накладывает жесткие требования на используемый микроконтроллер: он должен иметь достаточно свободных ресурсов и быть мощным, чтобы за короткое время сохранить контекст текущей задачи, загрузить контекст следующей и при этом сохранять данные о выполнении всех остальных неактивных процессов. Поэтому для обычных 8-разрядных микроконтроллеров такой метод организации многозадачных приложений не подходит.

Другим возможным решением поставленной задачи для 8-разрядных микроконтроллеров является алгоритм разделения по коду. В этом случае переключение с процесса на процесс происходит только тогда, когда текущий процесс выполнил некоторый заранее определенный программистом этап работы. При этом не нужно сохранять контекст каждой задачи, так как к моменту завершения этапа процесс фактически окончен и может быть безболезненно прерван. Однако программисту необходимо заранее продумать структуру приложения и включить в код информацию обо всех состояниях и переходах между его этапами. Иначе такой код будет содержать много информации,

Таблица 2. Характеристики компилятора CC7A

	Демо	Расширенная
Размер кода	2K×16	64K×16
Целочисленные переменные	8, 16, 24 бит	8, 16, 24, 32 бит
Переменные с плавающей запятой	24, 32 бит	16, 24, 32 бит
Переменные с фиксированной запятой	8, 16, 24 бит	8, 16, 24, 32 бит
Поддержка многозадачности	Нет	Есть LEANSLICE
Стоимость	Бесплатно	12 тыс. руб.

не относящейся к основным функциям приложения, и в результате получиться запутанным и трудночитаемым.

КОНЦЕПЦИЯ LEANSLICE

Решением для простого и удобного создания многопоточных приложений по принципу разделения кода может стать режим LEANSLICE в компиляторе CC7A.

Концепция LEANSLICE отличается от традиционных подходов. В ней заложен ряд особенностей, существенно упрощающих многозадачное программирование под небольшие микроконтроллеры:

- возможность реализации многозадачных приложений на микроконтроллерах с фиксированным аппаратным стеком, доступ к которому не может быть получен никакими другими средствами, кроме команд CALL и RETURN;
- возможность независимого написания кода для каждой задачи, что значительно упрощает структуру программы;
- использование алгоритма разделения по коду, наиболее подходящего для небольших 8-разрядных микроконтроллеров;
- автоматизация проектирования конечных программных автоматов. Процесс их проектирования приближен к традиционному программированию на С, а код программы – компактнее и понятнее.

Для определения в коде программы моментов завершения очередных этапов процесса, моментов, когда он может быть остановлен, а также для запуска процессов из основной функции и организации переключения между процессами используются специализированные, встроенные в компилятор функции. Это позволяет создавать для микроконтроллеров более совершенные программы, причем затраты сил и времени такие же, как на обычную программу.

РЕАЛИЗАЦИЯ МНОГОЗАДАЧНОСТИ

Приложение с реализацией многозадачности в режиме LEANSLICE представляет собой набор задач, описанных независимо друг от друга, и основную функцию, определяющую порядок переключения процессов и некоторые общие для всего приложения действия. Каждая задача представляет собой конечный автомат, после каждого выполнения кода текущего состояния которого, приложение переключается на следующую задачу. При этом информация о состоянии задач и необходимых переключениях вставляется в код программы автоматически при компиляции.

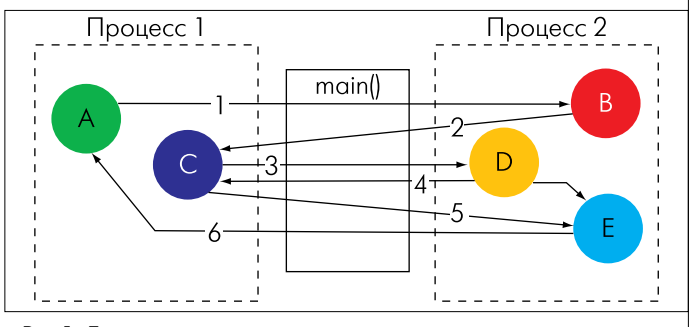

Рис.1. Диаграмма переключения процессов

Рассмотрим диаграмму, приведенную на рис.1. Приложение содержит два параллельных процесса. Кружками на нем обозначены состояния процессов, некоторые законченные этапы работы, а стрелочками показаны переключения между ними. Допустим, приложение начинает выполняться с этапа A первого процесса. После завершения этого этапа выполнение кода управления будет передано этапу B второго процесса. После завершения этапа B, если сформированы условия запуска этапа C, произойдет переход на соответствующий код. Если условие не выполнено, то управление снова передается второму процессу на этап D. Таким образом, пока не сформированы условия запуска для первого процесса, продолжается выполнение второго процесса, причем после каждого этапа будет иметь место попытка передать управление первому процессу.

Для описания состояний конечного автомата процесса разработчику требуется лишь вставить специальные функции waitState() для завершения каждого этапа в коде задачи, а для переключения процессов – одну или несколько функций taskSlicer() в основной функции. Заметим, что ни waitState() ни taskSlicer() не являются функциями в обычном понимании, это своеобразные директивы компилятора, заменяемые при компиляции требуемым кодом.

Функция taskSlicer() может иногда явно указывать на следующий процесс, в этом случае необходимо передать в качестве параметра имя следующей задачи. Код основной функции может выглядеть, например, следующим образом:

```
void main();
startTask(<Процесс 1>);
startTask(<Процесс 2>);
<инициализация переменных>
taskSlicer(<Процесс 1>);
While(1)
{
        if (<условие>)
        taskSlicer(< Процесс 2>);
        else taskSlicer(<Процесс 1>);
}
```

Если функции taskSlicer не указывают явно номер процесса, то компилятор автоматич ески построит код таким образом, что процессы будут переключаться по кругу. В этом случае основная функция должна иметь вид:

```
void main();
startTask(<Процесс 1>);
startTask(<Процесс 2>);
<инициализация переменных>
While(1)
{
taskSlicer();
}
```

В обоих примерах программа, выполнив инициализацию всех задач и переменных, входящих в основной этап первой задачи, производит возврат в основную программу, и далее выполняется следующая задача в соответствии с кодом основной функции.

Код текущего состояния задачи считается законченным при достижении функций waitState() или changeState(). Разница между ними состоит в том, что waitState() обычно включается в тело цикла <условия перехода к следующему состоянию>, а changeState() жестко переключает состояние автомата на следующее (указанное явно или не явно). Например, в приведенном ниже примере операторы состояния 1 будут выполняться до достижения <условия 1>, при этом после каждого цикла выполнения приложение будет переходить из задачи в основную функцию. После удовлетворения <условия 1>, выполнив <последовательность действий 2>, задача переключит состояния на указанное в <номер состояния перехода>, но перед этим произойдет выход в основную функцию и выполнение остальных задач приложения.

```
task <имя задачи>(void)
{
while(<условие 1>)
{
<последовательность действий состояния1>;
waitState();
}
<последовательность действий состояния2>;
changeState(<номер состояния перехода>);
.....
}
```

Формат статьи не позволяет подробно описать все возможности компилятора для работы с процессами. Компилятор может также останавливать процессы, "подвешивать" их, перезапускать, получать состояние текущего этапа процесса. В общем, при помощи LEANSLICE можно реализовать все необходимые функции для написания оптимального кода многозадачного приложения.

Кроме того, компилятор обладает набором библиотек функций, необходимых разработчику многопоточных приложений. В их состав входят:

- библиотека задержек и таймеров – содержит 8-, 16- и 24-разрядные таймеры, которые могут быть использованы для получения необходимых задержек;
- библиотека событий – позволяет процессам управлять друг другом;
- библиотеки семафоров и двоичных семафоров – используется для защиты общих для всех процессов данных и ресурсов;
- почтовые ящики – позволяют процессам обмениваться сообщениями.

Для более подробного ознакомления со всеми функциями и возможностями компилятора CC7A и режима поддержки многозадачности LEANSLICE можно скачать руководство по эксплуатации с сайта www.milandr.ru.

ПРИМЕР РЕАЛИЗАЦИИ МНОГОЗАДАЧНОГО ПРИЛОЖЕНИЯ

Рассмотрим подробнее методику созданий многозадачной программы на примере небольшого приложения, реализующего функции электронных часов, секундомера и будильника на базе

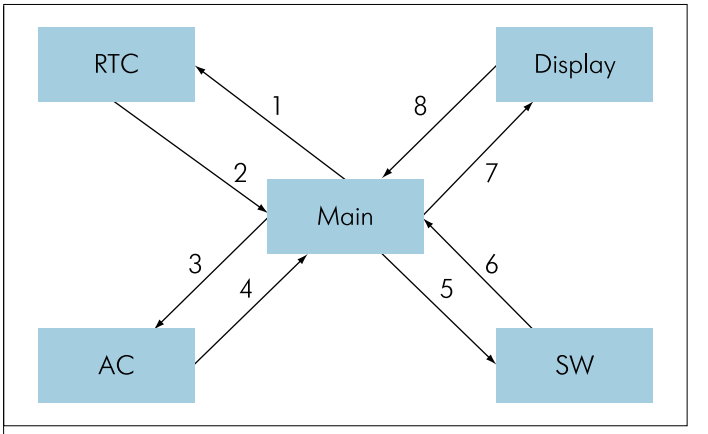

Рис.2. Диаграмма переключения процессов приложения часы-секундомер-будильник

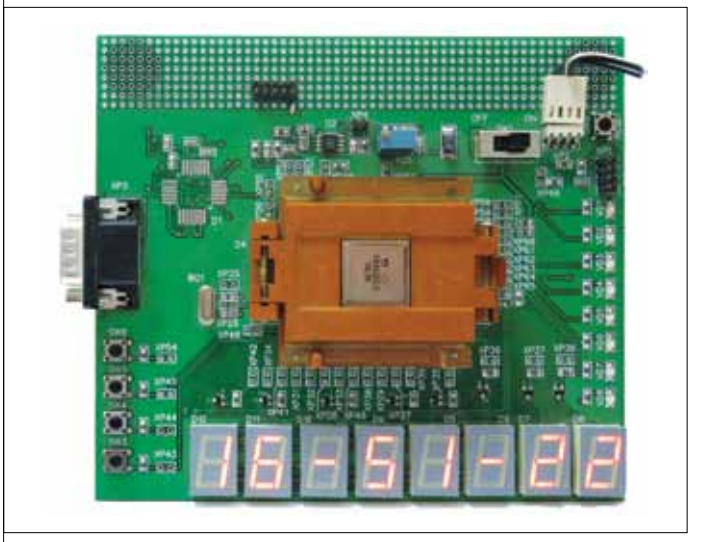

Рис.3. Отладочная плата для микроконтроллера 1886BE2

демонстрационно-отладочной платы для микроконтроллера 1886BE2. В данной задаче функции часов, будильника и секундомера могут быть представлены как отдельные, независимые друг от друга процессы, с помощью которых демонстрируются возможности поддержки многозадачности компилятора CC7A, а реализация на стандартном демонстрационно-отладочном наборе позволит пользователям самостоятельно повторить и расширить приложение с целью освоения новых возможностей.

Исходное приложение разбито на четыре различных процесса, управляемых из основной функции (рис.2). Управление устройством осуществляется кнопками SW6, SW5, SW4 и SW3 (порт E, выводы PE[3..0] микроконтроллера), а вывод информации – на индикаторы D12...D8 (рис.3). Сигналом будильника является мигание светодиодов VD8... VD1, управляемых портом F.

Рассмотрим подробнее код для основной функции*. После инициализации всех констант, переменных и задач следует основной цикл функции. В цикле после определения текущего режима и состояния таймера происходит перебор всех процессов.

```
void main(void)
{
<инициализация констант кодов символов>;
<включение и конфигурация таймера 0>;
<конфигурация портов C, D, F, E>
<инициализация переменных времени > //Для каждой задачи переменные
//времени(ч, мин, с) должны быть
//различны.
<инициализация переменной режима Mode> //Переменная режима определяет
//только режим индикаторов и кнопок.
startTask(RTC); //активация всех задач.
startTask(SW);
startTask(AC);
startTask(Display);
while(1)
{
if (RE3==0) //Выбор режима осуществляется комбинацией:
{
if (RE0==0) WrkState = 0x00; //часы КН3 и КН0;
else if (RE1==0) WrkState = 0x01; //секундомер КН3 и КН1;
else if (RE2==0) WrkState = 0x02; //таймер КН3 и КН2;
else;
}
else;

if(TMR0H >= 0x09) //Если таймер достиг значения, соответствующего
if(TMR0L >= 0x00) // секундному отсчету, устанавливается флаг Inc.
 {
TMR0H = 0x00; // Далее все задачи должны инкрементировать свои
TMR0L = 0x00; // переменные времени.
Inc = 0xFF;
else Inc = 0x00; //В противном случае флаг
} обнуляется.
taskSlicer();//Функция инициирует перебор всех задач.
}
```

Первая исполняемая задача – это процесс задания времени часов RTC(). Диаграмму автомата процесса можно увидеть на рис.4. Реализации этого автомата на языке C с использованием режима LEANSLICE имеет вид:

```
task RTC(void)
{
 while(!((RE2 == 0) && (RE3 == 1) && (WrkState == 0)))
        { if (Inc == 0xFF)
          {IncrementTime(); //функция инкрементирования переменных времени в
           waitState(); } // соответствии с ходом часов (60с, 60 мин, 24ч) }
    while(!((RE2 == 0) && (RE3 == 1) && (Mode == 0)))
    { if ((RE1 == 0) && (RE3 == 1) && (Mode == 0)) secRTCincrement();
          if ((RE0 == 0) && (RE3 == 1) && (Mode == 0))
secRTCdecrement();
           waitState(); }
    while(!((RE2 == 0) && (RE3 == 1) && (Mode == 0)))
    {     if ((RE1 == 0) && (RE3 == 1) && (Mode == 0))
minRTCincrement();
          if ((RE0 == 0) && (RE3 == 1) && (Mode == 0))
minRTCdecrement();
          waitState(); }
    while(!((RE2 == 0) && (RE3 == 1) && (Mode == 0)))
    { if ((RE1 == 0) && (RE3 == 1) && (Mode == 0))     hourRTCincrement();
          if ((RE0 == 0) && (RE3 == 1) && (Mode == 0))
hourRTCdecrement();
    waitState(); }
```

В нулевом состоянии, т.е. "нормальный ход часов", автомат будет находиться до нажатия клавиши SW5. При этом каждый раз процесс RTC в нулевом состоянии, получив управление, проверяет, требуется ли инкрементировать переменные времени и в случае необходимости, посчитав новое значение переменных, возвращает управление основной функции, которая передает управление следующему процессу. Клавиша SW5 переводит автомат во второе состояние, где происходит установка секунд. При этом, если приложение не находится в режиме часов (Mode!=0), управление блокируется, но процесс не останавливается и продолжает счет времени. Далее следуют этапы и установки минут и часов. При очередном нажатии клавиши SW5 автомат возвращается в исходное состояние нормального хода часов.

Процесс "будильник" AC() имеет аналогичную структуру. Отличия заключаются в том, что начальным состоянием для него является установка секунд, далее после каждого нажатия SW5 следует установка переменных минут, часов, а в последнем состоянии процесс сравнивает установленное значение со значением часов. При совпадении значений зажигается линейка светодиодов VD8...VD1. Этот процесс, как и любой другой в этом приложении, можно перевести в любое состояние и переключить режим устройства. Управление процессом при этом будет блокировано, но он продолжает функциониро-

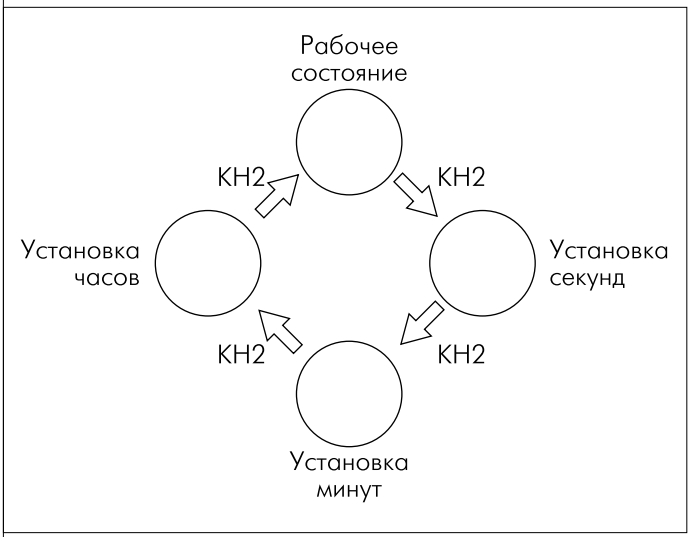

Рис.4. Диаграмма состояний процесса "часы"

* Во всех примерах вырезаны детали, несущественные для демонстрации работы компилятора. Полный код приложения доступен на сайте www.milandr.ru.

вание в том состоянии, в котором был оставлен.

Процесс "секундомер" SW() имеет два состояния – состояние работы и состояние остановки. При этом возможно осуществить его сброс. Процесс функционирует аналогично предыдущим.

Последний процесс Display() выводит на индикаторы состояния переменных времени процесса в соответствии с выбранным режимом. При этом индикаторы управляются при помощи двух 8-разрядных портов микроконтроллера 1886BE2: на порт C выводится код символа для отображения на индикаторе, а текущий индикатор для отображения выбирается с помощью порта D. Таким образом, чтобы отобразить информацию необходимо постоянно обновлять каждый индикатор, последовательно перебирая их, чтобы они продолжали свечение. Эта задача также демонстрирует возможности контроля времени выполнения задач. Так как необходимо постоянно обновлять информацию на индикаторах, повторные выполнения задачи должны осуществляться через короткие промежутки времени, определяемые временем угасания светодиода. Так, для процесса вывода данных время обновления информации на индикаторах составляет суммированное время, за которое выполняются инструкции активного состояния каждого процесса. Это достаточно для получения четкого и яркого свечения. Структура процесса Display:

```
task Display(void)
{
        while(1)
{
        if (WrkState == 0x00)                //в зависимости от режима
        { sec = secRTC; min = minRTC; hour = hourRTC;} //переменным
процесса
        else if (WrkState == 0x01) //display назначаются
        { sec = secSW; min = minSW; hour = hourSW;} //переменные вре-
мени
    else { sec = secBP; min = minBP; hour = hourBP;} //одного из процес-
сов.
    PORTC = <код первого символа>; //далее последовательно
    PORTD = <номер первого индикатора >; //обновляются светодиоды.
    .....
    PORTC = <код последнего символа>;
    PORTD = <номер последнего индикатора >;
    waitState();
    } }
```

Мы рассмотрели пример реализации обычной задачи, разработать которую можно и без многозадачности системы компилирования. Применение режима LEANSLICE компилятора CC7A упрощает структуру приложения и ускоряет реализацию. Код программы не содержит сложной системы условных переходов и множества служебных переменных.

Для более сложных задач с множеством условий переменных и процедур компилятор LEANSLICE также является оптимальным решением.