МАТЕРИАЛ 5. ОТЛАДКА В СИСТЕМЕ МИКРОПРОГРАММИРОВАНИЯ СЕМЕЙСТВА MCS-51 ОТ KEIL ELEKTRONIK GmbH.

Цель – изучить основные приёмы работы с отладчиком Keil Elektronik GmbH для разработки ПО микропроцессоров семейства MCS-51 и производных; освоить операционную систему реального времени RTX51 Tiny и работу с ней.

ОТЛАДЧИК KEIL ELEKTRONIK GmbH.

Два режима отладки: встроенный симулятор-отладчик (Use Simulator) и продвинутая отладка с эмулятором (Use) (например: Keil Monitor 51 и т.п.). Выбираются в окне **Option for Target – Debug**. Левая панель диалогового окна устанавливает опции для встроенного симулятора-отладчика, а правая для продвинутой отладки.

Use Simulator – чисто программная отладка, позволяющая проводить отладку микроконтроллерной системы до завершения аппаратной разработки. В этом режиме средствами отладчика моделируются порты микроконтроллера и необходимая периферийная аппаратура; без каких-либо аппаратных средств.

Use – отладка разработанных аппаратно-программных средств микроконтроллерной системы в комплексе. Необходим эмулятор и соответствующие ему программные средства. (Вариант: собственная программа компонуется с дополнительным специальным модулем (например: Monitor 51) и отладка осуществляется посредством обмена через последовательный интерфейс.)

Характеристики: 1). Отладчик поддерживает моделирование до 16 Мбайт памяти. 2). Поддерживает встроенные аппаратные средства различных представителей семейства MCS-51, которые представлены в базе данных менеджера проекта, (регистры специальных функций (SFR) и регистры выводов ножек кристалла (VTREG).)

Категории команд отладчика: -1. команды ведения точек останова; -2. команды общей отладки; -3. команды работы с областями памяти; -4. команды исполнения программы; -5. команды окна наблюдения (см. лекция №2).

Выполнение команд отладки.

Команды отладки в массе своей можно выполнить несколькими путями: -1) (основной) в командной строке окна вывода **Output Window – Command** закладка; -2) меню **Debug** для основных команд; -3) «горячие клавиши» для важных команд; -4) команды локального меню в своих окнах (правая кнопка мыши). Тот или иной путь выбирается по удобству пользователя.

Действия при отладке.

В процессе отладки, как правило, выполняются команды отладки и вычисляются выражения. Последовательность отладочных действий может составлять **программу отладки** (порядок действий). Программа отладки может быть (и действительно написана) в виде командного отладочного файла для его интерпретации отладчиком. Эта программа очень схожа с программой в стандартной нотации языка программирования Си.

Программа состоит из строк (предложений):

1) команды отладки – составляют основное содержание отладочных мероприятий(см. лекция №2);

2) выражения – помогают проведению основных мероприятий отладки, уточняют операндную часть для команд отладки и функций пользователя;

3) задание глобальных переменных для собственной отладки – команда DEFINE, (не обязательно, но удобно, ибо глобальные переменные запрещены);

4) объявление функций – вспомогательная информация отладчику (задать повторяющиеся действия и локальные переменные для работы);

5) вызовы функций (возможны 3 типа функций: предопределённые, пользователя и сигнальные) – дополнительный специализированный сервис (особенно для аппаратной части).

Синтаксис для программы отладки соответствует синтаксису языка Си. Есть следующие **отличия**: -1) не поддерживаются агрегатные типы (структуры, объединения, массивы); -2) нет указателей в программе отладки, указатели в отлаживаемой программе представляются своим значением; -3) используются только команды и функции отладчика (аналог стандартных библиотек); -4) для всех символических имён нет отличия больших и малых букв.

Пример: 1) учёт затрат ресурсов и проверка временной диаграммы в коде программы формирования периодического отклика на запрос или 2)фрагмент отладки программы с вычислением чисел последовательности Фибоначчи, раздел определения затрат ресурсов на доступ к данным из разных классов памяти:

```
DEFINE long oldStat                                   рабочая переменная для подсчёта циклов
oldStat=states
DEFINE int oldAddr                                    рабочая переменная для подсчёта размера кода
oldAddr=$ & 0xFFFF
FUNC void MyLS( void ) {                              функция пользователя
  int work;
  work = $ & 0xFFFF;
  printf( "такты = %d\n", states - oldStat );
  oldStat = states;
  printf( "длина кода = %d\n", work - oldAddr );
  oldAddr = work;
}
P 2                                          (или F10, F10) для       ;   dValue = 536l;
MyLS( )                                                                    ; SOURCE LINE # 33
            такты = 4                                                  MOV     dValue,#02H
```

```
            длина кода = 6                                  MOV     dValue+01H,#018H
P 2                                     (или F10, F10) для      ;   bValue = 0xAA;
MyLS( )                                                         ; SOURCE LINE # 34
            такты = 4                                       MOV     bValue,#00H
            длина кода = 6                                  MOV     bValue+01H,#0AAH
P 4                                     (или F10, F10, F10, F10) для    ;   iValue = 537u;
. . . . . .
```

Общие команды отладчика.

15 команд общего назначения дополняют весь сервис при выполнении отладки.

Команды	Краткое описание
ASSIGN	Задать источники ввода/вывода для окна последовательного порта.
DEFINE	Определяет (добавляет) кнопку на панель инструментов.
DIR	Справочная информация о всех символических именах, включая внутренние имена.
EXIT	Выход из режима отладки.
INCLUDE	Выполнить последовательность команд из заданного командного файла.
KILL	Удаляет функции отладчика и кнопки с панели инструментов.
LOAD	Загрузить .obj файл на отладку, или HEX файл или символьную информацию.
LOG	Ведёт журнал выполнения команд отладки.
MODE	Задаёт установки для PC COM портов.
RESET	Сбрасывает отладчик в исходное состояние, переустанавливает доступ к регионам памяти и т.д.
SAVE	Сохраняет содержимое памяти в файле формата Intel HEX386.
SCOPE	Отображает адреса функций в программе и составляющих модулях.
SET	Устанавливает строковое значение предопределённым переменным.
SIGNAL	Отображает список или удаляет сигнальные функции.
SLOG	Обеспечивает ведение файла с содержимым окна последовательного порта.

Набор программы отладки.

1). Набирать программу отладки можно построчно непосредственно в командной строке окна **Output Window – Command** закладка.

2). Набрать можно в текстовом редакторе и исполнить с помощью команды **INCLUDE** набранный командный файл, который может включать другие командные файлы.

3). Интегрирует второй способ. Набрать программу отладки можно в редакторе функций (меню **Debug – Function Editor (Open Ini File) …**), а затем и выполнить её. Окно редактора функций приведено на рис. 1.

Окно редактора функций

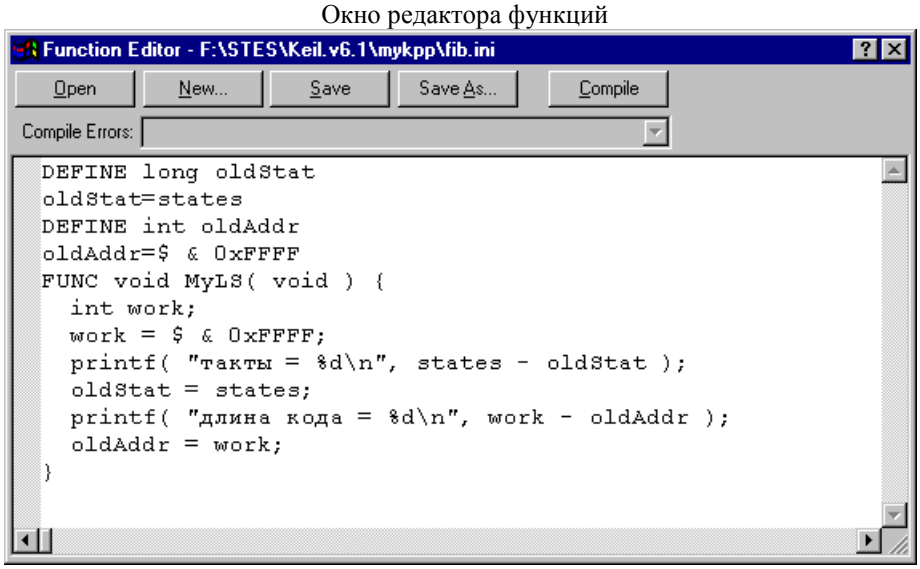

Рис. 1.

Редактор функций аналогичен обычному текстовому редактору со стандартными действиями редактирования и кнопками управления: открыть, новый файл, сохранить и сохранить как…

Посредством кнопки компилирования проводится интерпретация команд в файле на соответствие синтаксису командной строки. Если есть ошибки, то они будут выведены в строке ошибок. Если ошибок нет, то компиляция аналогична действию команды INCLUDE с отработкой командного файла.

Имя командного файла для исполнения можно задать в диалоговом окне **Option for Target – Debug – Initialization File.** При запуске отладчика этот файл будет исполнен.

ВЫРАЖЕНИЯ ОТЛАДЧИКА.

Большинство команд отладки содержат **параметры**, которые в общем случае задаются выражениями. Кроме того, **выражения** могут использоваться в качестве самостоятельной команды при отладке. (Это единые позиции как для языков высокого уровня Си, так и Ассемблера.)

(Всякая величина имеет значение и меру). Выражения могут представлять следующие **типы значений: -1)** константа, представляется исключительно своим значением; **-2)** адрес в своём пространстве памяти; **-3)** системные переменные, общие для всех представителей семейства MCS-51; **-4)** символические имена аппаратных устройств периферии, специфичные для каждого из представителей семейства MCS-51 (SFRs и VTREGs); **-5)** спецификации

типа для явного приведения типа. Во всех выражениях могут использоваться программные переменные.

Адреса могут быть: **-1)** константа с префиксом пространства памяти (**B:** для пространства Bit; **C:** для пространства Code; **Bx:** для пространства Code Bank; **D:** для пространства Data; **I:** для пространства Idata; **X:** для пространства Xdata); **-2)** адрес бита (<выражение>.<позиция бита>); **-3)** адрес данных в памяти (программные переменные для данных); **-4)** адрес программы (программные имена функций и меток); **-5)** номер линии строки (как правило, приводится к адресу программы). Разные типы выражений применяются по назначению для соответствующих команд отладки.

Выражения состоят из отдельных термов, соединённых операциями (знаками или операциями). В качестве термов выступают: -1) **константы**; -2) предопределённые имена программы отладчика – **системные переменные**; -3) специальные имена для конкретной модели микроконтроллера, это – **регистры специальных функций** и регистры **VTREG**; -4) программные переменные, это – все **символические имена** отлаживаемой программы; -5) **номера линий** строк отлаживаемой программы; -6) **операторы** и спецификации типа.

Константы.

Имеется **4 категории** констант, представляющих нотации языка Си и Ассемблера. Это – целочисленные константы, константы с плавающей запятой, символьные константы, строковые константы.

Целые константы. Поддерживают 4 системы счисления: 2-ю, 8-ю, 10-ю и 16-ю. Правомерны префиксные и суффиксные нотации. По умолчанию числа 16-разрядные. Суффикс **L** в конце задаёт 32-разрядное число. Число может быть разбито на поля с помощью знака доллара – **$**. Первый нуль в 16-ых числах нужен, если они начинаются с буквы.

Система	Префикс	Суффикс	Пример
Двоичная	нет	Y или y	10100101Y или 1010$0101y
Восьмеричная	нет	Q, q, O, или o	765q или 567Q или 123o
Десятичная	нет	T или ничего	1234T или 1234 или 1$234
Щестнадцатиричная	0x или 0X	H или h	1234H или 0x1234

Числа с плавающей запятой. Должны содержать либо точку, либо экспоненту, либо и то и другое. Перед точкой целая часть обязательна.

123.4567	123e2	0.12e4
0.123456789	234e+3	12.3456e-3
345.6	12e-4	0.023e+2

Символьные константы. Полностью соответствуют стандарту Си. ESC-последовательности допустимы в той же мере, как и в языке Си.

'a', '3', '\n', '\x34', '\012' и т.д.

'\\', '\"'

Строковые константы. Стандарт Си – двойные кавычки. Допустимы вложенные строки когда в команде отладки строка задаёт команду, у которой имеется строка. В этом случае двойные кавычки и обратный слэш отмечаются дополнительным обратным слэшем.

"длина кода = %d\n"

"printf(\"такты = %d\\n\")"

Load "F:\\STES\\Keil_v6.1\\prhelo"

Системные переменные.

Переменная	Тип	Описание
$	unsigned long	Программный счётчик. Допустимо чтение и запись в переменную. $ = C:0x030
radix	unsigned int	Определяет основание системы счисления по умолчанию. Может быть либо 10, либо 16. При запуске отладчика – 16.
states	unsigned long	Текущий счётчик циклов АЛУ. Только чтение.
break	unsigned int	Позволяет останавливать и запускать работу отладчика. Если не ноль, то отладчик остановлен. В противном случае – отладчик работает. Применяется в функциях пользователя и сигнальных функциях.
iip	unsigned char	Индицирует число вложенных прерываний.
itrace	unsigned int	Индицирует включение режима записи трассировки выполняемых инструкций программы. Если не ноль, запись трассировки разрешена. В противном – запрещена.

Регистры выводов ножек кристалла (VTREG).

Перечень регистров **зависит от модели** используемого микроконтроллера. Модель задаётся при создании проекта с помощью фирмы изготовителя и типа микроконтроллера в диалоговом окне **Options for Target – Target**.

Имя	Тип	Описание
XTAL	ulong	Тактовая частота микроконтроллера. Задаётся в диалоговом окне **Options for Target**. Например 12 МГц = 0x00B71B00
CLOCK	ulong	Задаёт число циклов АЛУ, для которых время исполнения программы занимает 1 секунду. Единицы измерения совпадают для переменной **states.** Например: 1000000 = 0x000F4240
PORT0	uchar	Состояние ножек входов порта p0. Например = 0xFF
PORT1	uchar	Состояние ножек входов порта p1. Например = 0xFF
PORT2	uchar	Состояние ножек входов порта p2. Например = 0xFF
PORT3	uchar	Состояние ножек входов порта p3. Например = 0xFF

SIN	uint	Содержимое входного буфера последовательного интерфейса. Например = 0x0002
SOUT	uint	Содержимое выходного буфера последовательного интерфейса. Например = 0x0032 «2»
STIME	uchar	Определяет частоту тактирования последовательного интерфейса. Если равно 1, то используются программные установки. Если – 0, то программные установки игнорируются.

Большинство из VTREG-регистров может быть задано (и просмотрено так же) с помощью диалоговых окон для соответствующих периферийных устройств. Эти диалоги доступны из меню **Peripherals.**

1-ый порт

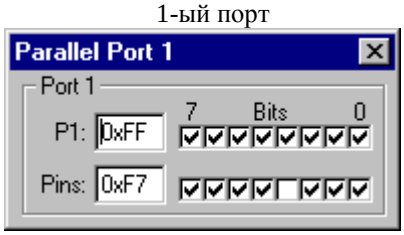

Рис. 2

Пример для отладки программы контроллера температуры приведен на рис. 2. Здесь 3-ий бит порта P1 задаёт переменную REGUPR, которая определяет режим управления автоматический – бит = 0 (если 1, то ручной режим измерения). Меню – **Peripherals – I/O-Ports – Port 1.**

Команда отладки могла быть простым выражением, вроде:

PORT1 &= ~0x08	сбросить бит
PORT1 |= 0x04	установить бит
PORT1 ^= 0x02	инвертировать бит

Другие разновидности микроконтроллеров семейства MCS-51 имеют аналоговые входы/выходы и соответствующие регистры **AINx**, и т.п..

Символические имена

Два варианта записи символических имён отлаживаемой программы: **простое** имя и **полностью квалифицированное** имя.

Простое имя в написании соответствует полностью тексту в программе.

Полностью квалифицированное имя в написании содержит имя модуля и имя функции, если последняя имеет место быть, с предшествующим обратным слэшем. Синтаксис:

\<имя модуля>\<имя переменной>
\<имя модуля>\<имя функции>
\<имя модуля>\<имя функции>\<имя переменной>

Например:

\izmtemp\FLAGDT	имя переменной
\izmtemp\SOSTKON	имя функции.

Полностью квалифицированное имя нужно в тех случаях, когда одним именем названы разные объекты (это плохая манера писать текст программы). В случае простых имён и наличия совпадений действует следующий порядок поиска имён:

1. – имена регистров;
2. – имя локальной переменной в функции;
3. – статическая переменная в текущем модуле;
4. – глобальное или внешнее имя в программе;
5. – символическое имя отладчика (DEFINE);
6. – системная переменная отладчика;
7. – имя VTREG.

Литеральное символическое имя – простое имя с предшествующей обратной кавычкой (`). Литеральное символическое имя используется в случае совпадающих названий. Для литеральных имён порядок поиска изменяется. Позиции 1 и 7 меняются местами.

Пример из программы вычисления чисел последовательности Фибоначчи:

`iN `iValCT `NumFib0

Номера линий строк

Номер линии строки в программе задаёт её адрес в памяти кода. Синтаксис:

\<имя модуля>\<номер линии>
\<номер линии>

Например:

\izmtemp\162	строка 162 в программе контроллера температуры
\163	строка 163 там же.

Операции и операторы.

Все операции и операторы соответствуют стандарту языка Си.

R1 = --R2 PORT1 &= ~8
& main P1 = 0x5A + 1
& \fib2\main

При вычислении выражений отладчик автоматически приводит полученные значения к необходимому типу. При необходимости можно провести явное приведение типа:

(unsigned int) 12.3	получим целое 12
(long) 12	получим 32-разрядное число.

ФУНКЦИИ ОТЛАДКИ.

Особенности функций отладки.

-1). Поддерживают в основе своей лишь стандарт ANSI C.

-2). Поддерживают операторы: **goto, if, else, switch, case, while, do, break, continue**.
-3). Нет препроцессора.
-4). Нет глобальных объявлений. Замена – команда отладки **DEFINE**.
-5). Ведение функций не поддерживает нотацию Керниган-Ричи (K&R).
-6). Допустимы только скалярные переменные, нет указателей, агрегаты не применимы.
-7). Функции не поддерживают рекурсию.
-8). Функции вызываются прямо по имени (нет указателей).
-9). Функции возвращают только скалярные типы.
-10). Локальные переменные не инициализируются при объявлении.
-11). Нет различия между прописными и строчными буквами.

В функциях отладки могут использоваться только выражения и функции отладки. Команды отладки вызываются специальной предопределённой функцией - **exec()**. Вызов функции отладки соответствует синтаксису Си - имя с параметрами в скобках. Функции отладки удобнее объявлять с определением в редакторе функций.

Отладчик поддерживает **три класса функций**: предопределённые функции, функции пользователя, сигнальные функции.

Команда **DIR** с параметром **FUNC** распечатывает все имеющиеся в отладчике функции на текущий момент. Параметр **BFUNC** задаёт распечатку всех предопределённых функций в отладчике. Параметр U**FUNC** задаёт распечатку всех функций пользователя в отладчике. Параметр **SIGNAL** задаёт распечатку всех сигнальных функций в отладчике. Активны в сеансе отладки могут быть только до 64 сигнальных функций, для пользовательских функций ограничений нет.

Предопределённые функции.

Предопределённые функции предоставляют первоначальный сервис для двух других категорий функций. Они не могут быть удалены из памяти. Они не могут быть переопределены в отладчике. 22 функции:

`int printf (char *format, ...)`

Аналогична библиотечной функции Си. Выводит строку в окне вывода

```
> printf( "такты = %d\n", states - oldStat );
такты = 4
```

`void twatch (ulong cycles)`

Используется исключительно в сигнальных функциях. Отрабатывает задержку в циклах АЛУ. Помогает формировать сигналы внешней периферии или для внешней периферии на ножках микроконтроллера. **Пример:** формирование меандра с частотой 1 Гц и скважностью 1 на выводе P1.3 порта первого порта (P1).

```
signal void p13_signal (void) {
  while (1) {
    PORT1 |= 0x08; /* 3-ий вывод порта №1 в 1 */
    twatch (CLOCK / 2); /* 0.5 секунды задержка */
    PORT1 &= ~0x08; /* 3-ий вывод порта №1 в 0 */
    twatch (CLOCK / 2); /* 0.5 секунды задержка */
  }
}
```

`int exec (char *cmd)`

Обеспечивает вызов команды отладчика из тела функции пользователя или сигнальной функции. Команды отладчика задаются строкой и разделяются точкой с запятой (**;**), если их несколько.

```
> exec ("dir vtreg; eval R0")
```

`uchar _RBYTE (ulong nAdr)`

Читает содержимое памяти байтов (8 разрядов) с указанным адресом и распечатывает его в окне вывода в формате байта.

```
> _rbyte(0x30)
```

`uint _RWORD (ulong nAdr)`

Читает содержимое памяти слов (16 разрядов) с указанным адресом и распечатывает его в окне вывода в формате слова.

`ulong _RDWORD (ulong nAdr)`

Читает содержимое памяти двойных слов (32 разряда) с указанным адресом и распечатывает его в окне вывода в формате двойного слова.

`float _RFLOAT (ulong nAdr)`

Читает содержимое памяти слов с плавающей запятой (32 разряда) с указанным адресом и распечатывает его в окне вывода в формате слова с плавающей запятой одинарной точности.

`double _RDOUBLE (ulong nAdr)`

Читает содержимое памяти слов с плавающей запятой (64 разряда) с указанным адресом и распечатывает его в окне вывода в формате слова с плавающей запятой двойной точности.

`void _WBYTE (ulong Adr, uchar val)`

Записывает величину заданного типа в указанное место (по адресу). Записывает байт.

```
> _wbyte(0x30,12)
```

void _WWORD (ulong Adr, uint val)

Записывает слово (16 разрядов).

void _WDWORD (ulong Adr, ulong val)

Записывает двойное слово (32 разряда).

void _WFLOAT (ulong Adr, float val)

Записывает слово с плавающей запятой одинарной точности (32 разряда).

void _WDOUBLE (ulong Adr, double val)

Записывает слово с плавающей запятой двойной точности (64 разряда).

int GetInt (char *title)

Запрашивает оператора с помощью указанной строки в функции для ввода числового значения и возвращает данное число в формате слова.

```
> ii = getint( "введите целое ii = " )
```

long GetLong (char *title)

Запрашивает оператора с помощью указанной строки в функции для ввода числового значения и возвращает данное число в формате длинного слова (32 разряда).

double GetDbl (char *title)

Запрашивает оператора с помощью указанной строки в функции для ввода числового значения и возвращает данное число в формате с плавающей запятой.

int rand (int seed)

Генерирует случайное 16-разрядное число с равновероятным законом распределения (диапазон от -32768 до 32767). Если параметр не ноль, то датчик инициализируется с этого значения. Если параметр равен нулю, то получается следующее число из последовательности

```
> rand (0)          > rand (10)
0x0029              0x0047
```

void memset (ulong addr, ulong many, uchar val)

Заполнить блок памяти заданным значением по аналогии с библиотечной функцией Си.

int _TaskRunning_ (ulong addr)

Проверяет, является ли указанная функция действующей в качестве текущей задачи. Возвращает 1, если текущая функция, в противном случае – 0. Используется совместно с системой реального времени RTX.

void wwatch (ulong addr)

Ожидание по записи в память данных. Эта функция полезна для моделирования устройств, отображаемых на память. Например:

управление 3-им выводом порта P1 SIGNAL

```
void p13w( void ) { //...
  printf( "Start p13w()\n" );
  PORT1 &= ~4;            // инициализация
  twatch( 5 );
  PORT1 |= 4;
  while( 1 ){             // работа
//   wwatch( X:8000h );
    wwatch( &cGlob1 );
    PORT1 ^= 4;
    printf( "Переключили P1=%x  такт=%d\n", PORT1, states );
  }
}
```

```
#pragma CODE
#pragma BROWSE
#pragma SMALL
 xdata char cGlob1 _at_ 0x8000;
 xdata char cGlob2 _at_ 0x8400;
 xdata char cGlob3 _at_ 0x8800;
 xdata char cGlob4 _at_ 0x8C00;
 //***** main *****
 void main ( void ) {
  char cLoc1, cLoc2;
  int iLoc;
  cGlob2 = cGlob1 = 0;
  cLoc1 = cGlob3;
  cLoc2 = cGlob4;
  iLoc = 10;
  cGlob2 = 1;
  while( iLoc-- ) ;
  cGlob1 = 1;
  iLoc = 15;
  cGlob2 = 2;
  while( iLoc-- ) ;
  cGlob1 = 2;
  while( 1 );
 }
```

void rwatch (ulong addr)

Ожидание по чтению памяти данных. Работа аналогична предыдущей функции.

void swatch (float nSec)

Ожидание в секундах. Функция полезна для задания больших промежутков времени. Например, генерация меандра на 7 выводе порта P1 может выглядеть следующим образом:

```
SIGNAL void p13s( void ) { // 25 kHz, но лучше будет для инфранизких частот!
  while( 1 ){
```

```
    PORT1 |= 0x80;
    swatch( 0.00002 );
    PORT1 &= ~0x80;
    swatch( 0.00002 );
  }
}
```

Функции пользователя.

Функция пользователя объявляется с ключевым словом FUNC. Позволяет выполнять команды отладки посредством одиночного вызова. Синтаксис объявления:

FUNC <тип возврата> <**имя функции**> **(** <список параметров> **) {**
<тело функции>
}

Тип возврата может быть: **bit, char, int, long, uchar, ulong, float, void.**

Важно: в функции пользователя нельзя вызывать сигнальную функцию и другие функции ожидания.

Сигнальные функции.

Сигнальная функция объявляется с ключевым словом SIGNAL. Позволяет моделировать электрические сигналы на ножках микроконтроллера для отладки с предполагаемым аппаратным окружением. Синтаксис объявления:

SIGNAL <тип возврата> <**имя функции**> **(** <список параметров> **) {**
<тело функции>
}

Особенности сигнальных функций:

- возвращаемое значение **void**;
- может принимать максимум до 8 параметров;
- может вызывать предопределённые функции и функции пользователя;
- не может вызывать другие сигнальные функции;
- не может быть вызвана из функции пользователя;
- должна вызывать функции ожидания.

Важно: сигнальные функции обязаны иметь в своём теле вызовы функций ожидания. В противном случае отладчик никогда не начнёт интерпретировать строки отлаживаемой программы.

Пример 1: функция моделирует ввод данных через последовательный интерфейс каждые 500 циклов АЛУ:

```
signal void inputser (void) {
  while (1) {
    sin = 2;
    twatch (500);
  }
}
```

Пример 2: функция моделирует меандр с частотой 10 Гц и скважностью 1 на втором входе порта P1:

```
signal void p123_signal (void) {
  while (1) {
    PORT1 |= 4;                 /* 2-ой вывод порта №1 в 1 */
    twatch (CLOCK / 20);        /* 0.05 секунды задержка   */
    PORT1 &= ~4;                /* 2-ой вывод порта №1 в 0 */
    twatch (CLOCK / 20);        /* 0.05 секунды задержка   */
  }
}
```

Наблюдать работу функции можно с помощью диалогового окна периферии (окно для порта №1) и точки останова по записи в регистр PORT1, например:

bs write PORT1, 1, "printf(\"Порт 1 =%x, такт =%d\\n\", PORT1, states)"

в окне вывода будет видно значение сигнала для порта и момент времени переключения в циклах процессора.

Пример 3: меандр принимается с внешнего устройства, который легко отслеживается:

```
SIGNAL void p10( void ) { // 25 kHz
  long cl;
  printf( "Start 25kHz p10()\n" );
  cl = 0;
  while( 1 ){
    printf( "Установили меандр =%d\n", cl++ );
    PORT1 |= 1;
    twatch( CLOCK / 50000 );
    PORT1 &= ~1;
    twatch( CLOCK / 50000 );
  }
}
```

Пример 4:

Команда отладчика **SIGNAL** отображает или удаляет сигнальные функции в сеансе отладки.

SIGNAL STATE — отображает текущее состояние сигнальных функций в сеансе отладки;

SIGNAL KILL <имя функции> — удаляет указанную сигнальную функцию.

ОПЕРАЦИОННАЯ СИСТЕМА РЕАЛЬНОГО ВРЕМЕНИ (RTOS).

Ряд задач для микроконтроллеров можно разбить на набор относительно самостоятельных подзадач, которые могут выполняться одновременно. Для таких приложений ОС реального времени (**ОСРВ**) или **RTOS** предоставляет средства гибкого планирования и постановки задач на выполнение.

Многозадачность поддерживается реализацией алгоритма разделения времени. Режим реального времени поддерживается механизмом реагирования на события.

Есть **две версии RTOS**: RTX51 Tiny и RTX51 Full. Полная версия системы в дополнение поддерживает систему приоритетов задач до 4 уровней, обмен сообщениями, семафоры и работу в сети CAN.

ОСРВ RTX51 Tiny.

Система RTX51 Tiny реализована в малой (SMALL) модели памяти. Планировщик системы использует аппаратный таймер 0 и его прерывания. Каждой задаче поочерёдно выделяется 1 квант времени. Стандартный режим: 1 такт (tick) равен 10000 циклов процессора, 1 квант времени = 5 тактов. Эти настройки можно изменять под нужды пользователя. Отметим также, что система разделения времени может быть отключена, тогда работа осуществляется под управлением событиями. Глобальное запрещение прерываний в пользовательском приложении останавливает и планировщик системы RTX51 Tiny.

Приложение для ОСРВ состоит из одной или более задач, которые имеют свой идентификатор - ID. RTX51 Tiny управляет до 16 задач одновременно. Задача - это простая функция Си, которая не имеет параметров и не возвращает результирующего значения. Определение задачи имеет следующий формат:

```
void MyFunc (void) _task_ id {   // где id - идентификатор задачи
… … …
}
```

По прошествии аппаратного сброса на выполнение ставится задача под номером 0. Задача 0 ставит на выполнение другие задачи по своему усмотрению в порядке реализации своего алгоритма. Например, работа в стандартном режиме разделения времени может выглядеть так:

Пример 1

```
// Работа в ОСРВ
#pragma CD
#pragma LC
#pragma SB
#include <rtx51tny.h>
// номера задач от 0 до максимального (15)
// Датчик случайных чисел!
static unsigned long int counter0;
static unsigned long int counter1;

#define lA1 ( 3 * 5 * 7 * 11 * 13 * 17L )
#define lC1 ( 5 )

static void MySimplJob0( void ) _task_ 0 {
  os_create_task( 2 );              /* поставить задачу task 2 к готовым */
  while( 1 ) {                      /* бесконечный цикл                  */
    counter0 *= lA1;                /* увеличить счётчик counter         */
    counter0 += lC1;                /* нарастить счётчик counter         */
  }
}

static void MySimplJob1( void ) _task_ 2 {
  while( 1 ) {                      /* бесконечный цикл                  */
    counter1++;                     /* нарастить счётчик counter1        */
  }
}
```

Система RTX51 Tiny поддерживает следующие функции:

```
char os_create_task ( unsigned char task_id )
char os_delete_task ( unsigned char task_id )
char os_wait ( unsigned char typ, unsigned char ticks, unsigned int dummy )
char os_wait1 ( unsigned char typ )
char os_wait2 ( unsigned char typ, unsigned char ticks )
char os_send_signal ( unsigned char task_id )
char os_clear_signal ( unsigned char task_id )
char isr_send_signal ( unsigned char task_id )
char os_running_task_id ( void )
```

Для поддержки этих функций система RTX51 Tiny состоит из следующих компонентов: 1). Системные часы - обработчик прерываний таймера 0, работающего в режиме 1 (16 разрядов); 2). Планировщик, осуществляющий переключение задач; 3). Процедура **main()**, инициализирующая систему; 4). Процедура **os_create()**, которая ставит

задачи в очередь; 5). Процедура **os_wait()**, которая переводит задачу в ожидание; 6). Процедура **os_delete()**, которая удаляет задачу из очереди; 7). Процедура **os_send_signal()** - передатчик сигнала адресату; 8). Процедура **os_clear_signal()**, которая сбрасывает сигнал для адресата; 9). Процедура **os_running_task_id()**, которая возвращает индекс текущей исполняемой задачи.

Система RTX51 Tiny управляет до 16 задач. Требует 7 байт памяти DATA; 3*<число задач> байт памяти IDATA для управления стеками и статусами задач; менее 900 байт памяти CODE. Эксплуатирует аппаратный таймер Timer 0. Продолжительность такта (tick) 1000 - 65535 машинных циклов. Реакция на события менее 20 машинных циклов. Переключение задач занимает 100 - 700 машинных циклов в зависимости от состояния стека.

Управление задачами.

Каждая задача в системе RTX51 Tiny может находиться в одном из 5 состояний. В состоянии исполнения в любое время может находиться только одна задача. Из состояния выполнения задача переключается в двух случаях: во-первых, если вызывает функцию os_wait(), во-вторых, если истекает её квант времени. Другая задача ставится на исполнение при двух условиях: во-первых, если нет других исполняемых задач и, во-вторых, задача находится либо в состоянии готовности, либо в состоянии тайм-аута.

Состояния задач следующие:

Running	- задача в текущий момент времени исполняется;
Ready	- задача готова к исполнению и ждёт своего кванта времени для выполнения;
Waiting	- задача ждёт совершения события, после которого переходит в разряд готовых к исполнению;
Deleted	- задача удалена из очереди, но не из программы; впоследствии может быть включена в очередь;
Time-out	- задача отработала свой квант времени и её ничто не прервало, переходит в разряд готовых.

Простая многозадачность - это режим разделения времени. Например:

Пример 2

```
// номера задач от 0 подряд
#include <rtx51tny.h>
static long int counter0;
static long int counter1;
static long int counter2;
static long int counter3;

static void MyJob0( void ) _task_ 0 {
  os_create_task( 1 );      /* поставить задачу task 1 к готовым */
  os_create_task( 2 );      /* поставить задачу task 2 к готовым */
  os_create_task( 3 );      /* поставить задачу task 3 к готовым */
  while( 1 ) {              /* бесконечный цикл                  */
    counter0++;             /* нарастить счётчик counter          */
  }
}

static void MyJob1( void ) _task_ 1 {
  while( 1 ) {              /* бесконечный цикл                */
    counter1++;             /* нарастить счётчик counter1      */
  }
}


static void MyJob2( void ) _task_ 2 {
  while( 1 ) {              /* бесконечный цикл                */
    counter2++;             /* нарастить счётчик counter1      */
  }
}

static void MyJob3( void ) _task_ 3 {
  while( 1 ) {              /* бесконечный цикл                */
    counter3++;             /* нарастить счётчик counter1      */
  }
}
```

Совместно с механизмом разделения времени может работать **механизм управления событиями**. Последний механизм может работать и самостоятельно, когда разделение времени отключено. В качестве событий выступают: сигнал, таймаут и интервал.

```
#define K_SIG     1           /* Wait for Signal     */
#define K_TMO     2           /* Wait for Timeout    */
#define K_IVL   128           /* Wait for Interval   */
```

1). Координация взаимодействия задач осуществляется посредством **передачи сигналов:**

Пример 3

```
// обмен сигналами
```

```
#include <rtx51tny.h>
static unsigned int counter0;
static unsigned int suma;
static unsigned long int counter1;

#define uC1 ( 511 )

static void MyJob0( void ) _task_ 0 {
  os_create_task( 1 );              /* поставить задачу task 1 к готовым */
  while( 1 ) {                      /* бесконечный цикл                  */
    if( ++counter0 == uC1 ) {       /* нарастить счётчик counter         */
      os_send_signal( 1 );          /* послать сигнал                    */
      counter0 = 0;
      suma++;
    }
  }
}

static void MyJob1( void ) _task_ 1 {
  while( 1 ) {                      /* бесконечный цикл                */
    os_wait1( K_SIG );              /* ждать сигнала для счёта         */
    counter1++;                     /* нарастить счётчик counter1      */
  }
}
```

2). Взаимодействие задач осуществляется посредством задания временных промежутоков на исполнение программного кода, т.е. **задание таймаута**. Таймаут - это временной интервал, заданный в тактах, от момента вызова функции ОСРВ, например:

// задержка задач. Пример 4

```
#include <rtx51tny.h>
static unsigned long int counter0;
static unsigned long int counter1;

static void MyJob0( void ) _task_ 0 {
  os_create_task( 1 );              /* поставить задачу task 1 к готовым  */
  while( 1 ) {                      /* бесконечный цикл                   */
    counter0++;                     /* нарастить счётчик counter          */
    os_wait2( K_TMO, 5 );           /* задержка в тактах ОС               */
  }
}

static void MyJob1( void ) _task_ 1 {
  while( 1 ) {                      /* бесконечный цикл                */
    counter1++;                     /* нарастить счётчик counter1      */
    os_wait2( K_TMO, 7 );           /* задержка в тактах ОС            */
  }
}
```

3). Временной промежуток задаётся в виде **величины интервала**. Интервал аналогичен таймауту с тем исключением, что временной промежуток отсчитывается от момента конца последней задержки, а не от текущего момента. Последнее обстоятельство позволяет с большой легкостью генерировать действительно периодические последовательности временных промежутков.

Функции класса ожидания существуют в виде трёх разновидностей. Основная функция - это os_wait, которая имеет 3 параметра. Используются только 2, а третий нужен для совместимости с полной версией системы RTX51. Первый параметр задаёт тип события ожидания, а второй значение временного промежутка в тактах (tick). Функция os_wait1 работает только лишь для ожидания сигнала. Функция os_wait2 специализирована для временных ожиданий. Тип ожидания можно задавать с помощью логического объединения нескольких событий, например:

```
os_wait2 ( K_TMO | K_SIG, 55 )
```

Функции ожидания **возвращают значение**, которое говорит запрашивающей задаче, какое событие действительно произошло и продолжило исполнение прерванной задачи:

```
#define NOT_OK      255         /* Parameter Error */
#define TMO_EVENT   8           /* Timeout Event   */
#define SIG_EVENT   4           /* Signal  Event   */
```

Управление стеком.

Каждая задача в системе RTX51 Tiny имеет свой собственный стек в пределах физического стека. Это есте-

ственное положение вещей. Но стека всегда мало. Поэтому система RTX51 Tiny для текущей работающей задачи выделяет максимально возможный стек, а остальным задачам стек сжимается до минимально возможных размеров. Операции со стеком выполняются во время переключения задач на выполнение.

ЛАБОРАТОРНАЯ РАБОТА 4. «РАБОТА С ОТЛАДЧИКОМ KEIL ELEKTRONIK GmbH»

Часть 1.

1. Исследовать программу контроллера температуры для системы дозированного нагрева биообъектов из лекции №1. Проверить работу команд отладки.
2. Определите временные интервалы в тактах и секундах выполнения основных блоков кода программы работы контроллера температуры.
3. Точно установить программные задержки в программе для нескольких частот тактирования микроконтроллера.
4. Передавайте команды контроллеру температуры, моделируя работу последовательного интерфейса. Отслеживайте при этом отклики контроллера. Используйте разные сигнальные функции.

Часть 2.

1. Провести балансировку более 3-х датчиков и получить с них измеренные температуры в 8-разрядном и 12-разрядном коде, поступающем с АЦП.
2. Отладить всю программу целиком. Моделировать отключения датчиков.

Часть 3.

1. Исследовать задачу управления дорожным светофором, которая находится в каталоге с примерами для среды разработки .\C51\EXAMPLES\TRAFFIC. Приложение состоит из 3 файлов: **getline.c** - текстовый редактор ввода с последовательного интерфейса; **serial.c** - программы буферизованного ввода/вывода с последовательного интерфейса и соответствующий обработчик прерывания; **traffic.c** - процедуры управления светофором для работы с RTX51 Tiny.
2. Задать свою временную диаграмму работы светофора. Продемонстрировать работу в отладочном режиме.