PUMPKIN™
REAL-TIME SOFTWARE

RM-PICC18
Справочное руководство

750 Naples Street • San Francisco, CA 94112 • (415) 584-6360 • http://www.pumpkininc.com
*• **Перевод:** Андрей Шлеенков • http://andromega.narod.ru • mailto:andromega@narod.ru •*

Справочное руководство Salvo для компилятора HI-TECH PICC-18

Введение

Данное руководство предназначено для пользователей Salvo, использующих микроконтроллеры PIC18 PICmicro® компании Microchip (http://www.microchip.com) с компилятором Си PICC-18 компании HI-TECH (http://www.htsoft.com).

Связанные документы

При построении приложений Salvo с компилятором HI-TECH C PICC-18, вместе с данным руководством должны использоваться следующие документы Salvo:

Руководство пользователя Salvo (Salvo User Manual)
Приложение AN-1 (Application Note AN-1) - устаревшее
Приложение AN-3 (Application Note AN-3)
Приложение AN-4 (Application Note AN-4) - устаревшее
Приложение AN-9 (Application Note AN-9)
Приложение AN-17 (Application Note AN-17)
Приложение AN-26 (Application Note AN-26)

Замечание: Пользователям MPLAB-C18 настоятельно рекомендуется обновить Microchip MPLAB IDE до версии 6.30 или более поздней. Вместо AN-1 и AN-4 следует использовать AN-26.

Примеры проектов

Примеры проектов Salvo, с использованием компилятора HI-TECH C PICC-18 и среды разработки Microchip MPLAB IDE версий 5 или 6, могут быть найдены в следующих директориях каждого дистрибутива Salvo для Microchip PICmicro® MCU:

```
\salvo\ex\ex1\sysf
\salvo\tut\tu1\sysf
\salvo\tut\tu2\sysf
\salvo\tut\tu3\sysf
\salvo\tut\tu4\sysf
\salvo\tut\tu5\sysf
\salvo\tut\tu6\sysf
```

Свойства

Таблица 1 иллюстрирует основные особенности реализации Salvo для компилятора Си HI-TECH PICC-18.

основное	
доступные дистрибутивы	Salvo Lite, LE & Pro for Microchip PICmicro® MCUs
поддерживаемые устройства	PIC18 PICmicro® MCUs
заголовочные файлы	`portpicc.h`
другие специфические для процессоров файлы	-
имена поддиректорий проекта	SYSF
salvocfg.h	
специфический для процессора файл требуется?	да
автоопределение компилятора?	да[1]
библиотеки	
директория `\salvo\lib`	`htpicc18`
переключение контекста	
метод	на основе меток при помощи `OSCtxSw(label)`
`_OSLabel()` требуется?	да
объем автоматических переменных и параметров функций в задачах	не ограничен
память	
поддерживаемые модели памяти	`small` и `large`
прерывания	
управляются через	`GIEL` и/или `GIEH` биты. Управляются через опцию конфигурации `OSPIC18_INTERRUPT_MASK`
статус прерывания сохраняется в критических секциях?	нет
используемый метод	прерывания запрещаются на входе и разрешаются при выходе из критических секций
степень вложенности	вложенность не допускается
альтернативные методы возможны?	да[2]
отладка	
отладка в исходных кодах?	только при построении с исходным кодом
компилятор	
поддержка упакованных битовых полей?	да
`printf() / %p` поддерживается?	да / нет
`va_arg()` поддерживается?	да

Таблица 1: Особенности реализации Salvo для компилятора Си HI-TECH PICC-18

Оптимизация компилятора

Несовместимая оптимизация

Ни одна из оптимизаций компилятора HI-TECH PICC-18 не известна как несовместимая с Salvo.

Библиотеки

Номенклатура

Имена библиотек Salvo для компилятора Си HI-TECH PICC-18 следуют соглашениям, показанным на примере имени библиотеки на Рисунке 1. Они подобны именам, используемым HI-TECH для стандартных библиотек PICC-18.[3]

Пример имени библиотеки: **`sfp82sab.lib`**

символы	значение	возможные варианты
s	Salvo	
f	тип	**f**: freeware **l**: standard
p8	PICC-18	
2	размер LFSR и указателей	**0**: нет LFSR, 16-бит указатели ROM **1**: нет LFSR, 24-бит указатели ROM **2**: LFSR, 16-бит указатели ROM **3**: LFSR, 24-бит указатели ROM **6**: LFSR, 16-бит указатели ROM, опечатки исправлены **7**: LFSR, 24-бит указатели ROM, опечатки исправлены
s	модель памяти	**l**: large (<=2 MB адресное простр-во) **s**: small (<=64 KB адресное простр-во)
a	конфигурация	**a**: многозадачность с задержками и событиями **d**: многозадачность с задержками **e**: многозадачность с событиями **m**: многозадачность только **t**: многозадачность с задержками, событиями и ожиданиями
b	вариант	**a**: функции вызываются отовсюду **b**: функции вызываются только из фонового режима **f**: функции вызываются только из режима переднего плана **-**: не применимо

Рисунок 1: Номенклатура библиотек Salvo для компилятора Си Microchip MPLAB-C18

Тип

Дистрибутив Salvo Lite содержит *свободные* (*freeware*) библиотеки. Все остальные дистрибутивы Salvo содержат *стандартные* (*standard*) библиотеки. Дополнительную информацию о типах библиотек см. в главе *Библиотеки* документа *Руководство пользователя Salvo*.

Целевой процессор

Каждая библиотека предназначена для одного или нескольких определенных процессоров. В Таблице 2 перечислены корректные библиотеки для каждого типа PicMicro® MCU.

код процессора	процессоры
`80s, 80l, 81s, 81l:`	Все PIC18 PICmicro® MCU. Рекомендуется только для 18C242, 18C252, 18C442, 18C452
`82s, 82l, 83s, 83l:`	Все PIC18 PICmicro® MCU *исключая* 18C242, 18C252, 18C442, 18C452, 18F252, 18F258, 18F452, 18F458. Рекомендуется для всех остальных PIC18 PICmicro® MCU.
`86s, 86l, 87s, 87l:`[4]	18F252, 18F258, 18F452, 18F458

Таблица 2: Процессоры для библиотек Salvo компилятора Си HI-TECH PICC-18

Замечание: библиотеки Salvo `p80s/p80l/p81s/p81l` предназначены для тех ранних PIC18 PICmicro® MCUs, которые не поддерживают инструкцию LFSR корректно[5] (например, 18C242, 18C252, 18C442, 18C452). *Все* более поздние устройства в семействе поддерживают эту инструкцию правильно.

Для процессоров, которые поддерживают LFSR корректно, используя библиотеки с поддержкой LFSR, будет получен меньший и более быстрый код.

Проверка кода процессора

Вы можете проверить, что выбрали правильную библиотеку Salvo, наблюдая за действиями компилятора PICC-18 C. Откройте `*.map` файл проекта и посмотрите окончание командной строки компоновщика. Там вы увидите, какая библиотека PICC-18 использовалась для создания вашего приложения. Используйте тот же самый код целевого процессора для вашей библиотеки Salvo.

Например, Листинг 1 показывает командную строку компоновщика для проекта PICC-18, выбранного для PIC18F6220:

В данном случае, PICC-18 C компилятор компонует его с библиотекой `pic82l-c.lib`, чтобы создать приложение. Поэтому соответствующий целевой код для библиотеки Salvo будет 82, например `sfp82leb.lib`.[6]

```
Linker command line:

-z -Mtu1lite.map –ol\obj \
  -ppowerup=00h,intcode=08h,intcodelo=018h,init,end_init –ACOMRAM=00h-05Fh \
  -ptemp=COMRAM –ARAM=0-0FFhx15 –ABIGRAM=0-0FFFh –pramtop=0F00h \
  -ACODE=00h-0FFFFh –pconfig=0300000h,idloc=0200000h,eeprom_data=0F00000h \
  -pconst=end_init+0F00h \
  -prbss=COMRAM,rbit=COMRAM,rdata=COMRAM,nvrram=COMRAM,nvbit=COMRAM \
  -pstruct=COMRAM –pnvram=-0FFh \
  -pintsave_regs=BIGRAM,bigbss=BIGRAM,bigdata=BIGRAM –pdata=RAM,param \
  -pidata=CODE,irdata=CODE,ibigdata=CODE –Q18F6620 –h+tu4lite.sym –E \
  -EC:\WINDOWS\TEMP\_3VV1BR5.AAA –ver=PICC18#V8.20PL4 \
  C:\HTSOFT\PIC18\LIB\picrt82l.obj C:\salvo\tut\tu4\main.obj \
  C:\salvo\src\mem.obj C:\salvo\lib\htpicc18\sfp82leb.lib \
  C:\HTSOFT\PIC18\LIB\pic82l-c.lib

Object code version is 3.7

Machine type is 18F6620
```

Листинг 1: Пример командной строки компоновщика для PIC18F6220 (из *.map file)

Модель памяти

Компилятор HI-TECH PICC-18 C поддерживает модели памяти `small` и `large`. При использовании библиотек, модель памяти всех исходных файлов должна соответствовать модели, используемой в библиотеке. При использовании исходного кода, одна и та же модель памяти должна быть применена ко всем исходным файлам.

Замечание: В отличие от конфигурации библиотеки и вариантов опций, определенных в файле `salvocfg.h`, для построения с библиотеками для выбранной модели памяти ничего не определено. Поэтому особое внимание должно быть уделено установкам модели памяти, используемым для создания приложения. Модель памяти обычно определяется на основе установок среды разработки IDE (например, MPLAB).

Конфигурация

С разными дистрибутивами Salvo поставляются разные конфигурации библиотек, позволяющие пользователю минимизировать объем ядра Salvo. См. главу *Библиотеки* документа *Руководство пользователя Salvo* для получения дополнительной информации о конфигурации библиотек.

Вариант

Поскольку PICmicro® MCU не имеют стека общего назначения, исходный текст Salvo обязан быть должным образом сконфигурирован соответствующими параметрами конфигурации. Библиотеки Salvo для компилятора Си HITECH PICC-18 предоставляются в различных вариантах как показано в Таблице 3.

Если ваше приложение не вызывает какой-либо сервис Salvo из прерываний, используйте вариант *b*. Если требуется вызывать эти сервисы только из прерываний, используете вариант *f*. Если вы желаете вызывать их из любого места, используйте вариант *a*. В каждом случае, вы должны вызывать сервисы из правильного места в вашем приложении, или либо компоновщик сгенерирует ошибку, либо ваше приложение потерпит неудачу во время выполнения.

вариант кода	описание
`a / OSA:`	Применяемые сервисы можно вызывать из любого места, то есть из функций переднего плана или фонового режима одновременно.
`b / OSB:`	Применяемые сервисы можно вызывать только из фонового режима (по умолчанию).
`f / OSF:`	Применяемые сервисы можно вызывать только из режима переднего плана.
`- / OSNONE:`	Библиотека не имеет вариантов.[7]

Таблица 3: Варианты для библиотек Salvo для компилятора Си HI-TECH PICC-18

См. параметры конфигурации `OSCALL_OSXYZ` для получения дополнительной информации о вызове сервисов Salvo из прерываний.

См. *Вопросы множественных Вызовов* ниже, для получения дополнительной информации об использовании вариантов библиотеки.

Установки компиляции

Библиотеки Salvo для компилятора Си HI-TECH PICC-18 построены, используя настройки по умолчанию, описанные в главе *Библиотеки* документа *Руководство пользователя Salvo*. Зависимые от процессора настройки и их замены перечислены в Таблице 4.

ограничения компиляции	
макс. число задач	3
макс. число событий	5
макс. число флагов событий[8]	1
макс. число очередей сообщений[9]	1
специфические для процессора установки	
размер задержки	8 бит
холостой ход	разрешен
биты разрешения прерываний в течение критических секций	`GIEH = GIEL = 0`
уровень прерывания[10]	`0`
указатели на сообщения	могут указывать на ROM или RAM
объекты Salvo[11]	`persistent`
счетчик системного времени	доступен, 32 бита
приоритеты задач	разрешены
сторожевой таймер	очищается в `OSSched()`

Таблица 4: Установки и замены для библиотек Salvo для компилятора Си HI-TECH PICC-18

Замечание: Поскольку для построения этих библиотек используется квалификатор банка `persistent`, `OSInit()` *должен* использоваться во всех приложениях, которые используют эти библиотеки. Без этого, переменные Salvo будут неинициализированы, и будут иметь непредсказуемые значения.

Замечание: Библиотеки Salvo для PIC18 сконфигурированы для 16-битовых указателей (тип указателя PICC-18 по умолчанию), и поэтому указатели сообщений могут указывать и на RAM и на ROM.

Замечание: Ограничения компиляции библиотек Salvo могут быть изменены в меньшую сторону (все дистрибутивы Salvo) или в большую (все дистрибутивы Salvo кроме Salvo Lite) по сравнению со значениями по умолчанию. См. главу *Библиотеки* документа *Руководство пользователя Salvo*.

Доступные библиотеки

Существует 360 библиотек Salvo для компилятора Си HI-TECH PICC-18. Каждый дистрибутив Salvo для Microchip PICmicro® MCU содержит также библиотеки Salvo из младших версий дистрибутивов.

Примеры salvocfg.h

Ниже приводятся примеры файлов конфигурации проекта `salvocfg.h` для различных дистрибутивов Salvo для PICmicro® MCU типа PIC18C452.

Замечание: Заменяя заданные по умолчанию число задач, событий и т.д. при использовании библиотек Salvo, `OSTASKS` и `OSEVENTS` соответственно *должны быть также определены* в файле `salvocfg.h` проекта. Если они будут оставлены неопределенными, будут использоваться значения по умолчанию (см. Таблицу 4).

Компиляция с библиотеками Salvo Lite

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSF
#define OSLIBRARY_CONFIG        OSA
#define OSLIBRARY_VARIANT       OSB
```

Листинг 2: Пример salvocfg.h для компиляции с библиотеками, используя sfp80lab.lib

Компиляция с библиотеками Salvo LE & Pro

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSL
#define OSLIBRARY_CONFIG        OSA
#define OSLIBRARY_VARIANT       OSB
```

Листинг 3: Пример salvocfg.h для компиляции с библиотеками, используя slp80lab.lib

Компиляция с исходным кодом Salvo Pro

```
#define OSENABLE_IDLING_HOOK     TRUE
#define OSENABLE_SEMAPHORES      TRUE
#define OSEVENTS                 1
#define OSLOC_ALL                persistent
#define OSTASKS                  3
```

Листинг 4: Пример salvocfg.h для компиляции с исходным кодом

Эффективность

Использование памяти

учебные примеры[12]	всего ROM[13]	всего RAM[14]
tu1lite	378	24
tu2lite	618	29
tu3lite	696	33
tu4lite	1462	43
tu5lite	2374	96
tu6lite	2658	103
tu6pro[15]	1994	84

Таблица 5: Требования памяти ROM и RAM для приложения Salvo, создаваемого компилятором Си HI-TECH PICC-18

Специальные вопросы

Вопросы стека

По архитектурным причинам, компилятор Си HI-TECH PICC-18 не передает параметры через стек и не выделяет в стеке память для локальных переменных. Вместо этого он использует *статическую оверлейную* модель. Это дает преимущества в скорости и в использовании памяти, но это не позволяет применять рекурсию и имеет некоторые другие эффекты.

Вопросы множественных вызовов

По умолчанию, ожидается, что сервисы Salvo будут вызываться только из фонового режима / основного цикла / уровня задачи. Это – конфигурация по умолчанию для построения с исходным кодом. Библиотеки *b-варианта* позволяют вызов сервисов только из фонового уровня. Если вы желаете вызывать определенные сервисы из уровня переднего плана / уровня прерывания, вы будете должны установить опции конфигурации OSCALL_OSXYZ для построения с исходным кодом или использовать другую библиотеку (см. Таблицу 3) для построения с библиотекой.

Из *Вариантов* (см. выше) мы видим, что библиотеки f-варианта позволяют вам вызывать сервисы чтения события и сигнализации из уровня переднего плана. Аналогично a-вариант библиотек позволяет вам вызывать сервисы из любого места вашего кода.

Прагма interrupt_level

Используя библиотеки a-варианта, каждый вызов сервиса должен быть осуществлен из уровня переднего плана, то есть из прерывания. Кроме того, прагма `interrupt_level` для PICC-18 должна быть установлена в 0 и помещена непосредственно перед процедурой обработки прерывания, как показано здесь:

```
#pragma interrupt_level 0[16]
void interrupt IntVector (void)
{
  OSStartTask(TASK_P);
}
```

Листинг 5: Установка прагмы interrupt_level для HI-TECH PICC-18 для процедуры прерывания (ISR) при использовании a-варианта библиотек

PICC-18 требует этого, чтобы управлять областями оверлея параметров для функций, расположенных на схеме множественных вызовов.

Замечание: Данная прагма не дает никакого эффекта, если нет никаких функций, расположенных по схеме множественных вызовов. Поэтому очень хорошо добавить это в любое приложение, компилируемое с PICC-18.

Пример: Сигнал одного типа события на переднем плане

При компиляции с библиотеками, если вы должны были переместить вызов `OSSignalBinSem()` из задачи Salvo (то есть из фонового режима) в обработчик прерывания (то есть на передний план) не меняя варианта библиотеки, вы увидите, что приложение потерпит аварию из-за переполнения стека почти немедленно. Это потому, что управление прерываниями по умолчанию[17] в `OSSignalBinSem()` несовместимо с его размещением в прерывании. Чтобы обойти это, вы должны изменить `OSLIBRARY_VARIANT` на `OSF` и компоновать приложение с библиотеками `f`-варианта (например, `sfp42Caf.lib` – отметьте `f` для *переднего плана* в поле варианта), чтобы должным образом поддерживать вызов сервиса события на переднем плане.

Пример: Сигнал одного типа события на переднем и заднем плане

Если мы вызываем `OSSignalBinSem()` из задачи и из обработчика прерывания, не учитывая проблемы схемы вызовов, компилятор выдаст сообщение об ошибке:

```
Error[ ] file : function _OSsignalBinSem appears in
multiple call graphs: rooted at intlevel 0 and _main
Exit status = 1
```

Чтобы разрешить это, добавьте прагму `interrupt_level 0` в ваш обработчик прерывания (см. Листинг 5, выше), и используйте a-вариант библиотеки после установки `OSLIBRARY_TYPE` в значение `OSA`.

OSProtect() и OSUnprotect()

Компилятор HI-TECH PICC-18 требует, чтобы, когда функция содержит множественные схемы вызова, прерывания должны быть запрещены "вокруг" этой функции, чтобы предотвратить разрушение параметров и/или возвращаемых значений.[18] Поэтому вы должны вызвать `OSProtect()` непосредственно перед и `OSUnprotect()` немедленно после всех случаев фоновых вызовов каждого сервиса Salvo, которые вызываются и из фона и из уровня переднего плана, например:

```
void TaskN( void )
{
  ...
  OSProtect();
  OSSignalBinSem(SEM_P);
  OSUnprotect();
  ...
}

#pragma interrupt_level 0
void interrupt IntVector(void)
{
  OSSignalBinSem(SEM_P);
}
```

Совет: Окружение из функций сервиса Salvo `OSProtect()` и `OSUnprotect()` в пределах функции может сделать ваш код более понятным. Функции окружения можно вызывать только из основной последовательности кода, то есть это должна быть единая схема вызова. Функция окружения могла бы выглядеть следующим образом:

```
OSSignalBinSem_Wrapper(OStypeEcbP ecbP)
{
  OSProtect();
  OSSignalBinSem(ecbP);
  OSUnprotect();
}
```

а макрос окружения мог бы выглядеть так:

```
#define OSSignalBinSem_Wrapper(ecbP) \
  do { OSProtect(); \
       OSSignalBinSem(ecbP); \
       OSUnprotect(); \
     } while(0);
```

Пример: Смешанные сигналы нескольких типов событий

Варианты библиотеки затрагивают все сервисы событий одинаково – то есть, библиотека `f`-варианта ожидает, что все применяемые сервисы событий будут вызваны из уровня переднего плана, то есть из прерываний. Если вы желаете вызывать некоторые сервисы из фонового уровня, а другие из переднего плана, вы должны будете использовать библиотеку `a`-варианта, как объяснено выше.

Осложнения возникают, когда вы нуждаетесь в библиотеке a-варианта для специфического типа события, и также используете дополнительные типы событий. В этом случае, каждый применяемый сервис события при использовании *нужно вызывать из переднего плана*. Если они вызываются не из переднего плана, компилятор выдает сообщение об ошибке:

```
Error[ ] file : function _OSSignalBinSem is not
called from specified interrupt level
Exit status = 1
```

Однако это нельзя вызывать из уровня фона. Если вы имеете "противоположную" ситуацию, например вы используете библиотеку a-варианта для одного типа события, и вы должны вызывать сервис для другого типа события только из фона, одно из решений состоит в том, чтобы поместить необходимый вызов переднего плана в обработчике прерывания, с условием, которое предотвращает его от каких-либо неполадок, например:

```
#pragma interrupt_level 0
void interrupt IntVector( void )
{
  /* real code is here ...        */
  ...
  /* dummy to satisfy call graph. */
  if( 0 )
  {
    OSSignalBinSem(OSECBP(1));
  }
}
```

Это создает граф вызовов, приемлемый для компилятора Си HI-TECH PICC-18 и позволяет успешную компиляцию и выполнение. Интересно, что оптимизатор удалит вызов из заключительного кода.

Управление прерываниями

Архитектура PIC18 поддерживает два различных уровня приоритетов прерываний. Когда разрешены, два раздельных глобальных бита разрешения прерываний, `GIEH` и `GIEL` используются для управления высоко- и низкоприоритетными прерываниями соответственно.

Прерывания автоматически запрещаются в критических секциях Salvo. По умолчанию, оба бита `GIEH` и `GIEL` сброшены (т.е. равны 0) в течение критических секций. Это управляется значением `OSPIC18_INTERRUPT_MASK` – опцией конфигурации Salvo (по умолчанию: `0xC0`).

Пользователи Salvo Pro могут переконфигурировать способ запрета прерываний в критических секциях переопределением `OSPIC18_INTERRUPT_MASK` в файле проекта `salvocfg.h`. Например, *если сервис Salvo* (например, `OSTimer()`) *вызывается только из низкоприоритетных прерываний*, то величина `0x40` для `OSPIC18_INTERRUPT_MASK` обеспечит то, что только низкоприоритетные прерывания будут запрещены в критических секциях Salvo. В этой конфигурации на высокоприоритетные прерывания Salvo не влияет. Это полезно при использовании высокоприоритетных прерываний.

[1] Выполняется автоматически при помощи символов HI_TECH_C и _PIC18, определяемых компилятором.
[2] Отсутствие адресуемого стека строго ограничивает возможности дополнительных методов.
[3] Как в PICC-18 v8.00.
[4] Специфично, PIC18FXX2 Rev B3 и PIC18FXX8 Rev B4. Проконсультируйтесь с PICC-18 readme файлами.
[5] См. Microchip's PIC18CXX2 Errata.
[6] Заметьте, что библиотека PICC-18 автоматически добавляется к командной строке компоновщика. Библиотека Salvo должна быть добавлена вручную пользователем как часть настройки проекта.
[7] Библиотека может не иметь вариантов, если целевой процессор не поддерживает прерывания.
[8] Каждый флаг события имеет свою память RAM, расположенную в своем блоке управления флагом события.
[9] Каждая очередь сообщения имеет свою память RAM, расположенную в своем блоке управления очередью сообщений.
[10] Аргумент для PICC-18 `#pragma interrupt_level` для тех сервисов, которые можно вызвать из ISR.
[11] Имея переменные Salvo `persistent`, компилятор PICC-18 может опустить некоторый код инициализации и таким образом уменьшить требования к ROM.
[12] Salvo v3.2.0 с PICC-18 v8.20PL4.
[13] В байтах.
[14] В байтах, все банки.
[15] Salvo Pro работает несколько иначе, чем Salvo Lite при конфигурации – см. учебный `salvocfg.h`.
[16] Salvo всегда использует уровень 0.
[17] `OSSignalBinSem()`, как и многие другие пользовательские сервисы, запрещает прерывания на входе и (вслепую) повторно разрешает их на выходе. Переразрешение прерываний, если помещено в процедуру прерывания PICmicro, вызывает проблемы. `OSSignalBinSem()` в `f`- и `a`-вариантах библиотек управляют прерываниями по разному.
[18] См. руководство PICC-18 для получения дополнительной информации.