PUMPKIN™
REAL-TIME SOFTWARE

RM-PICC18
Reference Manual

750 Naples Street • San Francisco, CA 94112 • (415) 584-6360 • http://www.pumpkininc.com

Salvo Compiler Reference Manual – HI-TECH PICC-18

Salvo™
The RTOS that runs in tiny places.™

© Copyright 2003 Pumpkin, Inc. last updated on Jan 6, 2004
All trademarks mentioned herein are properties of their respective companies.

Introduction

This manual is intended for Salvo users who are targeting Microchip (http://www.microchip.com/) PIC18 PICmicro® MCUs with HI-TECH's (http://www.htsoft.com/) PICC-18 C compiler.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with HI-TECH's PICC-18 C compiler:

Salvo User Manual
Application Note AN-1 (obsolete)
Application Note AN-3
Application Note AN-4 (obsolete)
Application Note AN-9
Application Note AN-17
Application Note AN-26

Note PICC-18 users are strongly advised to upgrade to Microchip's MPLAB IDE v6.30. Use *AN-26* in place of *AN-1* and *AN-4*.

Example Projects

Example Salvo projects for use with HI-TECH's PICC-18 C compiler and the Microchip MPLAB v5 and v6 IDEs can be found in the:

```
\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
```

directories of every Salvo for Microchip PICmicro® MCUs distribution.

Features

Table 1 illustrates important features of Salvo's port to HI-TECH's PICC-18 C compiler.

general	
available distributions	Salvo Lite, LE & Pro for Microchip PICmicro® MCUs
supported targets	PIC18 PICmicro® MCUs
header file(s)	`portpicc.h`
other target-specific file(s)	--
project subdirectory name(s)	SYSF
salvocfg.h	
target-specific header file required?	no
compiler auto-detected?	yes[1]
libraries	
`\salvo\lib` directory	`htpicc18`
context switching	
method	label-based via `OSCtxSw(label)`
`_OSLabel()` required?	yes
size of auto variables and function parameters in tasks	unrestricted
memory	
memory models supported	`small` and `large`
interrupts	
controlled via	`GIEL` and/or `GIEH` bits. Controlled via `OSPIC18_INTERRUPT_MASK` configuration option
interrupt status preserved in critical sections?	no
method used	interrupts disabled on entry and enabled on exit of critical sections
nesting limit	no nesting permitted
alternate methods possible?	yes[2]
debugging	
source-level debugging?	only in source-code builds
compiler	
bitfield packing support?	yes
printf() / %p support?	yes / no
va_arg() support?	yes

Table 1: Features of Salvo Port to HI-TECH's PICC-18 C Compiler

Compiler Optimizations

Incompatible Optimizations

None of HI'TECH's PICC-18 C compiler's optimizations are known to be incompatible with Salvo.

Libraries

Nomenclature

The Salvo libraries for HI-TECH's PICC-18 C compiler follow the naming convention shown in Figure 1. It is similar to that used by HI-TECH for the standard PICC-18 libraries.[3]

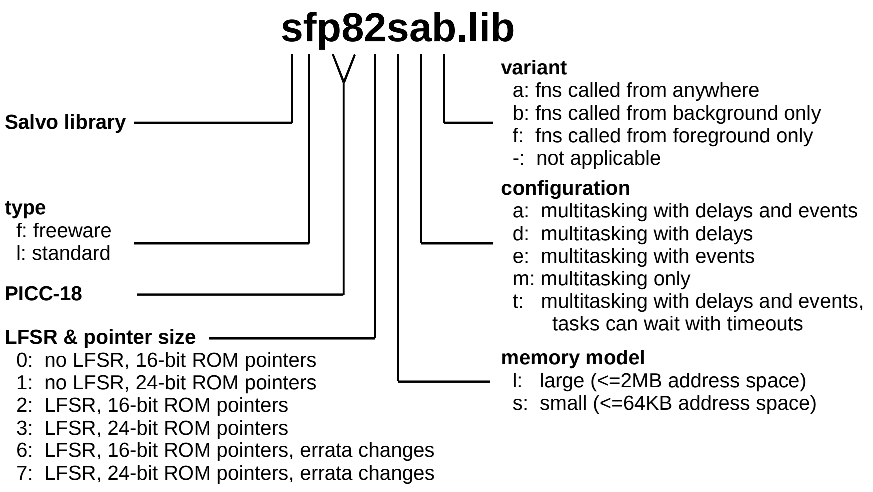
sfp82sab.lib
Salvo library
type
f: freeware
l: standard
PICC-18
LFSR & pointer size
0: no LFSR, 16-bit ROM pointers
1: no LFSR, 24-bit ROM pointers
2: LFSR, 16-bit ROM pointers
3: LFSR, 24-bit ROM pointers
6: LFSR, 16-bit ROM pointers, errata changes
7: LFSR, 24-bit ROM pointers, errata changes
variant
a: fns called from anywhere
b: fns called from background only
f: fns called from foreground only
-: not applicable
configuration
a: multitasking with delays and events
d: multitasking with delays
e: multitasking with events
m: multitasking only
t: multitasking with delays and events, tasks can wait with timeouts
memory model
l: large (<=2MB address space)
s: small (<=64KB address space)

Figure 1: Salvo Library Nomenclature – HI-TECH's PICC-18 C Compiler

Type

Salvo Lite distributions contain *freeware* libraries. All other Salvo distributions contain *standard* libraries. See the *Libraries* chapter of the *Salvo User Manual* for more information on library types.

Target

Each library is intended for one or more specific processors. Table 2 lists the correct library for each PICmicro® MCU.

target code	processor(s)
`80s, 80l, 81s, 81l:`	all PIC18 PICmicro® MCUs. Recommended only for 18C242, 18C252, 18C442, 18C452
`82s, 82l, 83s, 83l:`	all PIC18 PICmicro® MCUs *except* 18C242, 18C252, 18C442, 18C452, 18F252, 18F258, 18F452, 18F458 Recommended for all other PIC18 PICmicro® MCUs.
`86s, 86l, 87s, 87l:`[4]	18F252, 18F258, 18F452, 18F458

Table 2: Processors for Salvo Libraries – HI-TECH's PICC-18 C Compiler

Note The `p80s/p80l/p81s/p81l` Salvo libraries are for those early PIC18 PICmicro® MCUs that do not support the `LFSR` instruction correctly[5] (e.g. 18C242, 18C252, 18C442, 18C452). *All* later devices in the family support this instruction correctly.

On targets that support `LFSR` correctly, using libraries with `LFSR` support will result in smaller and faster code.

Verifying the Target Code

You can verify that you have chosen the right Salvo library by observing the PICC-18 C compiler's actions. Open the project's `*.map` file and look towards the end of the Linker command line entry. There, you will see which PICC-18 library was used to build your application. Use the same target code for your Salvo library.

For example, Listing 1 shows the linker command line entry for a PICC-18 project built for the PIC18F6220:

```
Linker command line:

-z -Mtu4lite.map -ol.obj \
  -ppowerup=00h,intcode=08h,intcodelo=018h,init,end_init -ACOMRAM=00h-05Fh \
  -ptemp=COMRAM -ARAM=0-0FFhx15 -ABIGRAM=0-0EFFh -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=-f00h \
  -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
```

Listing 1: Example Linker Command Line for PIC18F6220 (from *.map file)

In this case, the PICC-18 C compiler is linking to its `pic82l-c.lib` library in order to build the application. Therefore the appropriate target code for the Salvo library is `82`, e.g. `sfp`*`82`*`leb.lib`.[6]

Memory Model

The HI-TECH PICC-18 C compiler's `small` and `large` memory models are supported. In library builds, the memory model applied to all of the source files must match that used in the library. For source-code builds, the same memory model must be applied to all of the source files.

Note Unlike the library configuration and variant options specified in the `salvocfg.h` file for a library build, none is specified for the selected memory model. Therefore particular attention must be paid to the memory model settings used to build an application. The memory model is usually specified on a node-by-node basis inside an IDE (e.g. MPLAB).

Configuration

Different library configurations are provided for different Salvo distributions and to enable the user to minimize the Salvo kernel's footprint. See the *Libraries* chapter of the *Salvo User Manual* for more information on library configurations.

Variant

Because PICmicro® MCUs do not have a general-purpose stack, the Salvo source code must be properly configured via the appropriate configuration parameters. The Salvo libraries for HI-TECH's PICC-18 C compiler are provided in different variants as shown in Table 3.

If your application does not call any Salvo services from within interrupts, use the *b* variant. If you wish to these services exclusively from within interrupts, use the *f* variant. If you wish to do this from both inside and outside of interrupts, use the *a* variant. In each case, you must call the services that you use from the correct place in your application, or either the linker will generate an error or your application will fail during runtime.

variant code	description
`a / OSA:`	Applicable services can be called from *a*nywhere, i.e. from the foreground and the background, simultaneously.
`b / OSB:`	Applicable services may only be called from the *b*ackground (default).
`f / OSF:`	Applicable services may only be called from the *f*oreground.
`- / OSNONE:`	Library has no variants.[7]

Table 3: Variants for Salvo Libraries – HI-TECH's PICC-18 C Compiler

See the `OSCALL_OSXYZ` configuration parameters for more information on calling Salvo services from interrupts.

See *Multiple Callgraph Issues*, below, for more information on using library variants.

Build Settings

Salvo's libraries for HI-TECH's PICC-18 C compiler are built using the default settings outlined in the *Libraries* chapter of the *Salvo User Manual*. Target-specific settings and overrides are listed in Table 4.

compiled limits	
max. number of tasks	3
max. number of events	5
max. number of event flags[8]	1
max. number of message queues[9]	1
target-specific settings	
delay sizes	8 bits
idling hook	enabled
interrupt-enable bits during critical sections	`GIEH = GIEL = 0`
interrupt level[10]	0
message pointers	can point to ROM or RAM
Salvo objects[11]	persistent
system tick counter	available, 32 bits
task priorities	enabled
watchdog timer	cleared in `OSSched()`.

Table 4: Build Settings and Overrides for Salvo Libraries for HI-TECH's PICC-18 C Compiler

Note Because the `persistent` bank qualifier is used to build these libraries, `OSInit()` *must* be used in all applications that use these libraries. Without it, Salvo's variables will be uninitialized, with unpredictable results.

Note PIC18 Salvo libraries are configured for 16-bit pointers (PICC-18's default pointer type), and therefore message pointers can point to RAM and ROM.

Note The compiled limits for tasks, events, etc. in Salvo libraries can be overridden to be less (all Salvo distributions) or more (all Salvo distributions except Salvo Lite) than the library default. See the *Libraries* chapter of the *Salvo User Manual* for more information.

Available Libraries

There are 360 Salvo libraries for HI-TECH's PICC-18 C compiler. Each Salvo for Microchip PICmicro® MCUs distribution contains the Salvo libraries of the lesser distributions beneath it.

salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for different Salvo for PICmicro® MCUs distributions targeting the PIC18C452.

Note When overriding the default number of tasks, events, etc. in a Salvo library build, `OSTASKS` and `OSEVENTS` (respectively) *must also be defined* in the project's `salvocfg.h`. If left undefined, the default values (see Table 4) will be used.

Salvo Lite Library Build

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

Listing 2: Example salvocfg.h for Library Build Using sfp80lab.lib

Salvo LE & Pro Library Build

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

Listing 3: Example salvocfg.h for Library Build Using slp80lab.lib

Salvo Pro Source-Code Build

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

Listing 4: Example salvocfg.h for Source-Code Build

Performance

Memory Usage

tutorial memory usage[12]	total ROM[13]	total RAM[14]
`tu1lite`	378	24
`tu2lite`	618	29
`tu3lite`	696	33
`tu4lite`	1462	43
`tu5lite`	2374	96
`tu6lite`	2658	103
`tu6pro`[15]	1994	84

Table 5: ROM and RAM requirements for Salvo Applications built with HI-TECH's PICC-18 C Compiler

Special Considerations

Stack Issues

For architectural reasons, HI-TECH's PICC-18 C compiler does not pass parameters on the stack. Nor does it allocate memory for auto (local) variables on the stack. Instead, it employs a *static overlay* model. This has advantages in speed and memory utilization, but it precludes recursion and has other impacts.

Multiple Callgraph Issues

By default, it is expected that Salvo services will only be called from the background / main loop / task level. This is the default configuration for source-code builds. *b*-variant libraries allow service calls only from the background level. Should you wish to call certain services from the foreground / interrupt level, you will need to set `OSCALL_OSXYZ` configuration options for source-code builds or use a different library (see Table 3) for library builds.

From *Variant*, above, we find that the `f`-variant libraries allow you to call event-reading and –signaling services from the foreground. Similarly, the `a`-variant libraries allow you to call the applicable services from anywhere in your code.

The interrupt_level Pragma

When using the `a`-variant libraries, each instance of an applicable service in use must be called from the foreground, i.e. from an interrupt. Also, PICC-18's `interrupt_level` pragma must be set to 0 and placed immediately ahead of the application's interrupt routine, like this:

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

Listing 5: Setting the HI-TECH PICC-18 interrupt_level Pragma for an ISR when Using a-variant Libraries

PICC-18 requires this in order to manage the parameter overlay areas for functions located on multiple call graphs.

Note This pragma has no effect if there aren't any functions located on multiple call graphs. Therefore it's OK to add it to any application compiled with PICC-18.

Example: Foreground Signaling of One Event Type

In a library build, if you were to move a call to `OSSignalBinSem()` from a Salvo task (i.e. from the background) to an interrupt handler (i.e. to the foreground) without changing the library variant, you'd find that the application crashes from a stack overflow almost immediately. This is because the default interrupt control[17] in `OSSignalBinSem()` is incompatible with being placed inside an interrupt. To circumvent this, you must change

OSLIBRARY_VARIANT to OSF and link an f-variant library (e.g. sfp42Caf.lib — note the f for *foreground* in the variant field) in order to properly support event service calls in the foreground.

Example: Foreground and Background Signaling of One Event Type

If we call OSSignalBinSem() from a task and from within an interrupt handler without addressing the callgraph issues, the compiler issues an error message:

```
Error[ ] file : function _OSSignalBinSem appears
in multiple call graphs: rooted at intlevel0 and
_main Exit status = 1
```

To resolve this, add the interrupt_level 0 pragma to your interrupt handler (see Listing 5, above) and use the a-variant library after setting OSLIBRARY_TYPE to OSA.

OSProtect() and OSUnprotect()

HI-TECH's PICC-18 C compiler requires that when a function is contained in multiple callgraphs, interrupts must be disabled "around" that function to prevent corruption of parameters and/or return values.[18] Therefore you must call OSProtect() immediately before and OSUnprotect() immediately after all background instances of every Salvo service that is called from both the background and foreground levels, e.g.:

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

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

Tip Wrapping OSProtect(), the affected Salvo service and OSUnprotect() within another function can make your code more legible. The wrapper may only be called from mainline code – i.e. it can only have a single callgraph. A wrapper function might look like this:

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

and a wrapper macro might look like this:

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

Example: Mixed Signaling of Multiple Event Types

The library variants affect all event services equally – that is, an `f`-variant library expects all applicable event services to be called from the foreground, i.e. from within interrupts. If you wish to call some services from the background, and others from the foreground, you'll have to use the `a`-variant library, as explained above.

A complication arises when you need an `a`-variant library for a particular event type, and you also are using additional event types. In this case, each instance of an applicable event service in use *must be called from the foreground*. If it's not called from the foreground, the compiler issue this error message:

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

However, it need not be called from the background. If you have the "opposite" situation, e.g. you are using an `a`-variant library for one type of event and you need to call an event service for a different event type only from the background, one solution is to place the required foreground call inside an interrupt handler, with a conditional that prevents it from ever happening, e.g.:

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

```
        {
          OSSignalBinSem(OSECBP(1));
        }
      }
```

This creates a call graph acceptable to HI-TECH's PICC-18 C compiler and allows a successful compile and execution. Interestingly, the optimizer will remove the call from the final application.

Interrupt Control

The PIC18 architecture supports two distinct priority levels. When enabled, two separate global-interrupt-enable bits, `GIEH` and `GIEL`, are used to control high- and low-priority interrupts, respectively.

Interrupts are automatically disabled within Salvo's critical sections. By default, both `GIEH` and `GIEL` are reset (i.e. made `0`) during critical sections. This is controlled by Salvo's `OSPIC18_INTERRUPT_MASK` configuration option (default value: `0xC0`).

Salvo Pro users can reconfigure the way in which interrupts are disabled during critical sections by redefining `OSPIC18_INTERRUPT_MASK` in the project's `salvocfg.h`. For example, *if Salvo services* (e.g. `OSTimer()`) *are called only from low-priority interrupts,* then a value of `0x40` for `OSPIC18_INTERRUPT_MASK` ensures that only low-priority interrupts are disabled during a Salvo critical section. In this configuration, high-priority interrupts will therefore be unaffected by Salvo. This is especially useful when high-rate interrupts are present.

---

1 This is done automatically through the `HI_TECH_C` and `_PIC18` symbols defined by the compiler.

2 The lack of an addressable stack severely limits the scope of alternate methods.

3 As of PICC-18 v8.00.

4 Specifically, the PIC18FXX2 Rev B3 and PIC18FXX8 Rev B4 parts. Consult PICC-18 readme files for more information.

5 See Microchip's PIC18CXX2 Errata.

6 Note that the PICC-18 library is automatically added to the linker command line. The Salvo library must be added manually by the user as part of setting up the project.

7 A library may have no variants if the target processor does not support interrupts.

8 Each event flag has RAM allocated to its own event flag control block.

9 Each message queue has RAM allocated to its own message queue control block.

10 Argument for PICC-18's `#pragma interrupt_level` for those services that can be called from within an ISR.

11 By making Salvo's variables `persistent`, the PICC-18 compiler is able to omit some initialization code and thus reduce ROM requirements.

12 Salvo v3.2.0 with PICC-18 v8.20PL4.

13 In bytes.

14 In bytes, all banks.

15 Salvo Pro build differs slightly from Salvo Lite build due to configuration – see tutorial's `salvocfg.h`.

16 Salvo always uses level 0.

17 `OSSignalBinSem()`, like many other user services, disables interrupts on entry and (blindly) re-enables them on exit. The re-enabling of interrupts, if placed inside a PICmicro interrupt routine, causes problems. `OSSignalBinSem()` in the `f-` and `a`-variant libraries control interrupts differently.

18 See PICC-18 manual for more information.