IAR device description file format

IAR Embedded Workbench® for ARM

This guide describes the format of the IAR device description files, also referred to as DDF files.

Introduction

A DDF file is used by the C-SPY Debugger to provide information about the actual target hardware and to add more features. The filename extension used for a device description file is `ddf`. The main topics of the DDF file are:

Memory layout	Used by the C-SPY hardware debugger drivers to set up the memory cache (if implemented in the driver).
Peripheral registers	Used by all C-SPY drivers to display the content of peripheral registers in a symbolic form.
Peripheral register reset values	Used by the C-SPY simulator driver.
Interrupt definition	Used by the C-SPY simulator driver.

All entries are optional, but it is recommended to supply at least the memory layout and peripheral register information. It will simplify debugging.

The DDF files are also used for other, mostly hardware-related, purposes and there might exist definitions that are not described here.

File format

The file format is line-oriented – a definition must be completely described on one line. It is not allowed to split a definition on multiple lines. All comments begin with a semicolon.

Section markers are used for identifying the type of section. The section marker is located in the first column. The available section markers are:

`[Memory]`	Defines the memory layout.
`[SfrInclude]`	Includes a separate file used exclusively for defining the peripheral registers. This can be a DDF or an SVD file.
`[Sfr]`	Defines registers and optionally individual bits of registers.
`[SfrReset]`	Defines SFR reset values. This is only used by the C-SPY simulator driver.
`[SfrGroupInfo]`	Groups a collection of registers into a group. Typically, a group maps to a peripheral unit with a set of registers.
`[InterruptList]`	Defines interrupts. This is only used by the C-SPY simulator driver.

Note that the section marker is case sensitive, which means `[memory]` is not the same as the correct `[Memory]`.

Memory layout, [Memory]

This section starts with the marker `[Memory]`.

The format of the memory definition is:

`Memory = name zone start-address end-address access-type`

name	The name of the memory.
zone	The address space—zone—of the memory.
start-address	The start address of the memory.
end-address	The end address of the memory.
access-type	The access type of the memory: `R`, `W`, or `RW`. `W` denotes a peripheral register area.

The opening `[Memory]` does not need an ordinal number, like `Memory0 =`..., etc.

Inclusion of SFR files, [SfrInclude]

There are two ways of including SFR files:

- Using the `[SFfrInclude]` section marker
- Using the `#include` syntax.

Note that you can use the argument variables when including files, for example `$TOOLKIT_DIR$`.

Using the [SfrInclude] section marker

This type of section starts with the `[SFfrInclude]` section marker.

The format of the SFR file definition is:

`File = filename.txt`

This file should be located in the same directory as the DDF file or be specified with an absolute or relative path. This type of file is considered strong, see below for more information.

Using the #include syntax

For multiple and recursive inclusions of ddf/sfr/svd/*** files, use the `#include` syntax:

```
#include "file1.xxx"
#include weak "file2.xxx"
```

Basically, the filename extension can be anything, but if the file is using the SVD syntax, do not use the `ddf` or `sfr` extensions. Files other than `ddf` and `sfr` will not be parsed for anything other than SFR definitions. The #include file can be defined with an absolute or relative path, or no path at all. The path of the device file defined in the project **Options** dialog box will be used if no path is defined or as a starting point for a relative path.

Weak and strong files

The weak declaration (`#include weak`) informs the system that the SFR registers defined in that file are of a weak type. The strong SFR definitions (`#include`) will be used first in the order they are parsed, then all the weak declarations will be used in the order they are parsed.

The weak SFR definitions will be ignored in the following cases:

- If an SFR register with the same name already exists
- If an SFR register with the same location already exists.

This is for advanced users only.

Example

This is an example of the parsing order of the `#include` directive. The example assumes that the DDF file that you have specified in the project **Options** dialog box has two inclusions. The device description files will be parsed in this order:

Parsing order	DDF	Include directives
1	As specified in the **Options** dialog box.	`#include ddf1` `#include ddf2`
2	`ddf1`	`#include ddf1_1`
3	`ddf2`	`#include ddf2_1` `#include ddf2_2`
4	`ddf1_1`	`#include ddf1_1_1`
5	`ddf2_1`	No files included
6	`ddf2_2`	No files included
7	`Ddf1_1_1`	No files included

Register definitions, [Sfr]

This section starts with the marker `[Sfr]`.

The format of the register definition is:

`sfr = "`*`name`*`", "`*`zone`*`", `*`address`*`, `*`size`*`, base=`*`radix`*`, [, bitRange=`*`range`*`] [, bitShift="`*`bitfield-size`*`,`*`mask`*`[>>|<<]`*`n`*`,`*`mask`*`[>>|<<]`*`n`*`…"`

`name`	The name of the register. A period (`.`) in the name is used for separating the register name and the optional bit name.
`zone`	The memory zone of the register. The only available zone is `Memory`.
`address`	The address of the register.
`size`	The size in bytes of the register: `1`, `2`, `4`, or `8`.

radix	The radix used for displaying the numeric value of the register. Use `10` for decimal and `16` for hexadecimal. `float` and `double` are also available.
range	An optional bit range definition which can be a single bit number or a range of bits in the form `i–j`. For example, `0–2`. As an alternative, `bitRange` can be replaced with `bitMask` to specify the bit mask
bitfield-size	The resulting size of the constructed register or bitfield.
mask	The mask applied to the complete SFR to get the relevant bits.
>>n	Right shift *n* positions. If no shift is needed, the shift declaration can be omitted.
<<n	Left shift *n* positions. If no shift is needed, the shift declaration can be omitted.

The `bitShift` parameter can have up to four actions (that is, mask and shift).

The resulting value of `bitShift` is calculated as:

```
Value=((regVal AND mask0) shift0) OR ((regVal AND mask1) shift1) OR …
```

Example 1

In this example, `TRIX.part` will be a 16-bit bitfield consisting of bits 7-4 shifted 4 bits to the left, which means they will be shown as bits 11-8. `TRIX2` will be a 32-bit register with its value shifted 1 bit to the right:

```
sfr = "TRIX",      "Memory", 0xE000E008, 4, base=16
sfr = "TRIX.part", "Memory", 0xE000E008, 4, base=16, bitShift
="16,0x000000F0<<4"
sfr = "TRIX2",     "Memory", 0xE000E00C, 4, base=16, bitShift
="32,0xFFFFFFFF>>1"
```

Example 2

```
sfr = "IFSR.FS",   "Memory", 0xE000E004, 4, base=2, bitRange=0-4, bitShift
="5,0x0000000F>>0,0x00000400>>6"
```

In this example, `IFSR.FS` will be a 5-bit bitfield consisting of these bits (where bit 0 is the lowest bit):

```
IFSR.FS     IFSR
bit 3-0     bit 3-0     0x0000000F>>0
bit 4       bit 10      0x00000400>>6
```

Register reset value definitions, [SfrReset]

This section starts with the marker `[SfrReset]`.

This information is only used by the simulator and is optional, unless you want to simulate something that depends on SFR register values.

The format of the SFR reset definition is:

```
Reset = address size value
```

address	The address of the register.
size	The size in bytes of the register: `1`, `2`, `4`, or `8`.
value	The register value in hexadecimal notation.

Example

```
Reset = 0xFFFD90 4 0
Reset = 0xFFFD93 4 0
Reset = 0xFFFD94 4 0x90
```

Register group definitions, [SfrGroupInfo]

This section starts with the marker `[SfrGroupInfo]`.

The format of the register group definition is:

`group = "group-name", "name0", "name1", "name2" …`

group-name	The group name.
name0, etc.	The register names that form the group. A register name in the group must be defined in the register definition section `[Sfr]` of the DDF file.

Example

This example shows how to set up a DDF file for a few UART (serial port) registers. The UART is named UART0 and the registers are:

- A 32-bit baud rate register at address `0xF0001600`
- An 8-bit control register at address `0xF0001604`. Bit 0 is `RX` enable, bit 1 is `TX` enable, and bit 2–3 encodes the character length.

```
[Sfr]
sfr = "BAUD",          " Memory", 0xF0001600 , 4, base=16
sfr = "CONTROL",       " Memory", 0xF0001604 , 1, base=16
sfr = "CONTROL.RXEN",  " Memory", 0xF0001604 , 1, base=16, bitRange=0
sfr = "CONTROL.TXEN",  " Memory", 0xF0001604 , 1, base=16, bitRange=1
sfr = "CONTROL.CHAR",  " Memory", 0xF0001604 , 1, base=16, bitRange=2–3
; alternative definition:
;sfr = "CONTROL.RXEN", " Memory", 0xF0001604 , 1, base=16, bitRange=0x0C
[SfrGroupInfo]
Group = "UART0", "BAUD", "CONTROL"
```

Interrupt definitions, [InterruptList]

This section starts with the marker `[InterruptList]`.

This information is only used by the C-SPY simulator driver. If an interrupt is needed during simulation, open the **Interrupt Setup** dialog box and define the desired interrupt or used the **Forced Interrupt** window to force an interrupt. If no interrupts are defined in the DDF file, there are some default interrupts available.

The format of the interrupt definition is:

```
Interrupt = name, vectorNumber
```

name	The interrupt name.
vectorNumber	The interrupt vector to use for this interrupt.

Example

```
[InterruptList]
Interrupt = NMI          0x08
Interrupt = HardDefault  0x0C
Interrupt = MemManage    0x10
```