Файл конфигурации компоновщика

- Обзор
- Объявление типа сборки.
- Определение воспоминаний и регионов
- Регионы
- Работа с разделами
- Выбор раздела
- Использование символов, выражений и чисел.
- Структурная конфигурация

Прежде чем читать эту главу, вы должны ознакомиться с концепцией разделов, см. Модули и разделы, стр. 96.

Обзор

Чтобы связать и найти приложение в памяти в соответствии с вашими требованиями, ILINK нужна информация о том, как обрабатывать разделы и как размещать их в доступных областях памяти. Другими словами, ILINK нуждается в конфигурации, переданной ему с помощью файла конфигурации компоновщика.

Этот файл состоит из последовательности директив и обычно предоставляет возможности для:

• *Declaring the build type (Объявление типа сборки)* информирующее компоновщик о том, предназначена ли сборка для традиционной системы ПЗУ или для системы ОЗУ, помогая компоновщику проверять, что только подходящие разделы помещаются в разные области памяти.

• *Defining available addressable memories (Определение доступной адресуемой памяти)* предоставляющей компоновщику информацию о максимальном размере возможных адресов и определение доступной физической памяти, а также работа с памятью, к которой можно обращаться по-разному.

• *Defining the regions of the available memories that are populated with ROM or RAM (Определение областей доступной памяти, которые заполняются ПЗУ или ОЗУ),* с указанием начального и конечного адреса для каждой области.

• *Section groups (Группы разделов)* касающиеся того, как группировать разделы в блоки и наложения в зависимости от требований раздела.

• *Defining how to handle initialization of the application (Определение того, как обрабатывать инициализацию приложения),* предоставляющую информацию о том, какие разделы должны быть инициализированы, и как эта инициализация должна быть выполнена.

• *Memory allocation (Распределение памяти),* определяющее, где - в какой области памяти - должен быть размещен каждый набор разделов.

• *Using symbols, expressions, and numbers (Использование символов, выражений и чисел)*, выражающих адреса, размеры и т. д., в других директивах конфигурации. Символы также можно использовать в самом приложении.

• *Structural configuration (Структурная конфигурация)* означает, что вы можете включать или исключать директивы в зависимости от условия и разбивать файл конфигурации на несколько разных файлов.

• *Special characters in names (Специальные символы в именах).* При указании имени символа или раздела, в котором используются неидентификационные симво-

лы, вы можете заключить имя в обратные кавычки. Пример: ***‘My Name‘***.

Комментарии могут быть записаны как комментарии C (/*...*/) или как комментарии C ++ (// ...).

Объявление типа сборки

Объявление типа сборки в файлах конфигурации компоновщика указывает компоновщику, предназначена ли сборка для традиционной системы ПЗУ (с, среди прочего, инициализации переменных при запуске программы) или для системы ОЗУ, которая будет использоваться для отладки (где другие стили можно использовать инициализацию).

build for директива

```
build for { ram | rom };
```

Параметры

ram Предполагается, что сборка представляет собой отладочную или экспериментальную установку, при которой инициализация некоторых или всех переменных может быть выполнена во время загрузки.

rom Предполагается, что сборка представляет собой традиционную сборку ПЗУ, где инициализация всех переменных выполняется при запуске программы.

Если вы объявляете тип сборки ROM - и особенно если вы также объявляете, какие области памяти являются ROM или RAM - компоновщик может лучше проверять, что только подходящие разделы помещаются в разные области памяти. Если вы не укажете явно директиву инициализации (см. initialize directive, стр. 511), компоновщик будет вести себя так, как если бы вы указали инициализацию с помощью ***copy {rw} ;***.

Если вы объявляете тип сборки RAM, компоновщик не проверяет, какие типы секций помещаются в какую область памяти.

Если вы не включили директиву ***build for*** в файл конфигурации компоновщика, компоновщик выполняет только ограниченную проверку. Это полезно в первую очередь для целей обратной совместимости.

См. также define region directive (директиву определения региона), стр. 498.

Определение memories and regions (памяти и регионов)

ILINK нужна информация о доступных пространствах памяти, или, более конкретно, ему нужна информация о:

•Максимальный размер адресуемой памяти.

Директива ***define memory*** определяет область памяти с заданным размером, который является максимально возможным объемом адресуемой памяти, не обязательно физически доступной. См. директиву define memory, стр. 498.

•Доступная физическая память

Директива ***define region*** определяет область в доступной памяти, в которой могут быть размещены определенные разделы кода приложения и разделы данных приложения. Вы также можете использовать эту директиву, чтобы объявить, содержит ли область RAM или ROM. Это в первую очередь полезно при сборке для традиционной системы ПЗУ. См. директиву define region, стр. 498.

Регион состоит из одного или нескольких диапазонов памяти. Диапазон (range) - это непрерывная последовательность байтов в памяти, и несколько диапазонов могут быть выражены с помощью выражений области. См. Region expression,, стр. 502.

В этом разделе представлена подробная информация о каждой директиве компоновщика, относящейся к определению памяти и регионов.

define memory директива

```
define memory [ name ] with size = size_expr [ ,unit-size ];
```

где размер блока (unit-size) может быть одним из:

unitbitsize = bitsize_expr

unitbytesize = bytesize_expr

Параметры

size_expr Определяет, сколько единиц содержится в пространстве памяти - всегда отсчитывается с нулевого адреса.

bitsize_expr Определяет, сколько битов содержит каждая единица.

bytesize_expr Определяет, сколько байтов содержит каждая единица. Каждый байт содержит 8 бит.

Директива ***define memory*** определяет область памяти с заданным размером, который является максимально возможным объемом адресуемой памяти, не обязательно физически доступной. Это устанавливает ограничения для возможных адресов, которые будут использоваться в файле конфигурации компоновщика. Для многих микроконтроллеров достаточно одной области памяти. Однако для некоторых микроконтроллеров требуется два или более. Например, для архитектуры Гарварда обычно требуются два разных пространства памяти: одно для кода и одно для данных. Если определена только одна память, имя памяти указывать необязательно. Если размер блока не указан, блок содержит 8 бит.

Пример:

```
/* Declare the memory space Mem of four Gigabytes */
define memory Mem with size = 4G;
```

define region директива

```
define [ ram | rom ] name = region-expr;
```

где region-expr - это выражение региона, см. также Regions, стр. 501.

Параметры

ram Регион находится в ОЗУ (RAM memory).

rom Регион находится в ПЗУ (ROM memory).

name Имя региона.

Директива ***define region*** определяет регион, в котором могут быть размещены определенные разделы кода и данные. Область состоит из одного или нескольких диапазонов памяти, где каждый диапазон памяти состоит из непрерывной последовательности байтов в определенной памяти. Несколько диапазонов можно объ-

единить с помощью выражений областей - эти диапазоны не обязательно должны быть последовательными или даже в одной и той же памяти.

Если вы объявляете регионы как ROM или RAM, компоновщик может проверить, что только подходящие разделы помещены в регионы, если вы строите традиционную систему на основе ROM (см. директиву build for, стр. 497).

Пример

```
/* Define the 0x10000-byte code region ROM located at address 0x10000 */
define rom region ROM = [from 0x10000 size 0x10000];
```

logical директива

```
logical range-list = physical range-list
```

где ***range-list*** (список-диапазон) - один из

```
[ region-expr,... ]region-expr
[ region-expr,... ]from address-expr
```

Параметры

region-expr Выражение региона, см. так же Regions, page 501.

address-expr Выражение адреса

Logical директива сопоставляет логические адреса с физическими адресами. Физический адрес обычно используется при загрузке или записи содержимого в память, а логический адрес - это адрес, который видит ваше приложение. Физический адрес совпадает с логическим адресом, если не используются логические директивы или если адрес находится в диапазоне, указанном в логической директиве.

При генерации вывода в формате ELF отображение влияет на физический адрес в заголовках программы. При создании вывода в шестнадцатеричном формате Intel или Motorola S-records используется физический адрес.

Каждый адрес в списке логических диапазонов в указанном порядке сопоставляется с соответствующим адресом в списке физических диапазонов в указанном порядке.

Если один или оба списка диапазонов не заканчиваются формой ***from***, общий размер логических диапазонов и физических диапазонов должен быть одинаковым. Если одна сторона заканчивается формой ***from***, а не другая, то сторона, которая заканчивается формой ***from***, будет включать окончательный диапазон размера, который обеспечивает совпадение общих размеров, если это возможно. Если обе стороны оканчиваются формой ***from***, диапазоны будут расширяться до максимально возможного адреса, при котором общие размеры совпадают.

Настройка сопоставления логических адресов с физическими может повлиять на размещение разделов и другого содержимого. Никакой контент не может перекрывать более одного отдельного логического или физического диапазона. Кроме того, если есть отображение из другого логического диапазона в соответствующий физический диапазон, любой логический диапазон, для которого не было задано сопоставление с физическими диапазонами - не упомянутый в логической директиве - исключается из размещения.

Все логические директивы применяются вместе. Использование одной или нескольких директив для указания одного и того же сопоставления не влияет на результат.

Пример

```
// Logical range 0x8000-0x8FFF maps to physical 0x10000-0x10FFF.
// Логический диапазон 0x8000-0x8FFF соответствует физическому диапазону 0x10000-0x10FFF.
// No content can be placed in the logical range 0x10000-0x10FFF.
// Никакой контент не может быть помещен в логический диапазон 0x10000-0x10FFF.
logical [from 0x8000 size 4K] = physical [from 0x10000 size 4K];
// Another way to specify the same mapping
// Другой способ указать такое же сопоставление
logical [from 0x8000 size 4K] = physical from 0x10000;
// Logical range 0x8000-0x8FFF maps to physical 0x10000-0x10FFF.
// Логический диапазон 0x8000-0x8FFF соответствует физическому диапазону 0x10000-0x10FFF.
// Logical range 0x10000-0x10FFF maps to physical 0x8000-0x8FFF.
// Логический диапазон 0x10000-0x10FFF соответствует физическому 0x8000-0x8FFF.
// No logical range is excluded from placement because of this mapping.
// Никакой логический диапазон не исключен из размещения из-за этого сопоставления.
logical [from  0x8000 size 4K] = physical [from 0x10000 size 4K];
logical [from 0x10000 size 4K] = physical [from  0x8000 size 4K];
// Logical range 0x1000-0x13FF maps to physical 0x8000-0x83FF.
// Logical range 0x1400-0x17FF maps to physical 0x9000-0x93FF.
// Logical range 0x1800-0x1BFF maps to physical 0xA000-0xA3FF.
// Logical range 0x1C00-0x1FFF maps to physical 0xB000-0xB3FF.
// No content can be placed in the logical ranges 0x8000-0x83FF,
// Никакой контент не может быть помещен в логические диапазоны 0x8000-0x83FF,
// 0x9000-0x9FFF, 0xA000-0xAFFF, or 0xB000-0xBFFF.
logical [from 0x1000 size 4K] = physical [from 0x8000 size 1K repeat 4 displacement 4K];
// Another way to specify the same mapping.
// Другой способ указать такое же отображение.
logical [from 0x1000 to 0x13FF] = physical [from 0x8000 to 0x83FF];
logical [from 0x1400 to 0x17FF] = physical [from 0x9000 to 0x93FF];
logical [from 0x1800 to 0x1BFF] = physical [from 0xA000 to 0xA3FF];
logical [from 0x1C00 to 0x1FFF] = physical [from 0xB000 to 0xB3FF];
```

Regions (Регионы)

Regions - это набор неперекрывающихся диапазонов памяти. Выражение региона строится из литералов региона и операций над регионами (union - объединение, intersection - пересечение и difference - различие).

Region literal

```
[ memory-name: ][from expr { to expr | size expr } [ repeat expr [ displacement expr ]]]
```

где ***expr*** - выражение, см. выражения, стр. 525.

Параметры

memory-name Имя области памяти, в которой будет расположен литерал региона. Если есть только одна память, имя указывать необязательно.

from expr - начальный адрес диапазона памяти (включительно).

to expr - конечный адрес диапазона памяти (включительно).

size expr - размер диапазона памяти .

repeat expr - определяет несколько диапазонов в одной памяти для литерала региона .

displacement expr - это смещение от начала предыдущего диапазона в повторяющейся последовательности. Смещение по умолчанию - это то же значение, что и размер диапазона.

Литерал региона состоит из одного диапазона памяти. Когда вы определяете диапазон, необходимо указать память, в которой он находится, начальный адрес и размер. Размер диапазона можно указать явно, указав размер, или неявно, указав конечный адрес диапазона. Конечный адрес включается в диапазон, а диапазон нулевого размера будет содержать только адрес. Диапазон может охватывать нулевой адрес, и такой диапазон может даже быть выражен беззнаковыми значениями, потому что известно, где находится память.

Параметр повторения создаст литерал региона, содержащий несколько диапазонов, по одному для каждого повторения. Это полезно для банков или удаленных регионов.

Пример

```
/* The 5-byte size range spans over the address zero */
/* Диапазон размера 5 байт охватывает нулевой адрес.  */
Mem:[from -2 to 2]
/* The 512-byte size range spans over zero, in a 64-Kbyte memory */
/* Диапазон размера 512 байт переходит через ноль в 64-килобайтной памяти.  */
Mem:[from 0xFF00 to 0xFF]
/* Defining several ranges in the same memory, a repeating literal */
/* Определение нескольких диапазонов в одной и той же памяти, повторяющийся литерал */
Mem:[from 0 size 0x100 repeat 3 displacement 0x1000]
/* Resulting in a region containing: В результате получается регион, содержащий:
  Mem:[from 0 size 0x100]
  Mem:[from 0x1000 size 0x100]
  Mem:[from 0x2000 size 0x100]
*/
```

См. также директиву определения региона, стр. 498, и выражение региона, стр. 502.

Region expression (Выражение области)

region-operand
  | region-expr | region-operand
  | region-expr - region-operand
   region-expr & region-operand

где регион-операнд является одним из:

(region-expr)

region-name

region-literal

empty-region

где region-name - это регион, см. директиву определения региона, стр. 498
где region-literal - это региональный литерал, см. литерал региона, стр. 501
а empty-region - это пустой регион, см. пустая область, стр. 503.

Обычно регион состоит из одного диапазона памяти, что означает, что для его выражения достаточно литерала региона. Когда регион содержит несколько диапазонов, возможно, в разной памяти, вместо этого необходимо использовать выражение региона для его выражения. Выражения области на самом деле являются выражениями набора в наборах диапазонов памяти.

Для создания выражений области доступны три оператора: объединение (|), пересечение (&) и разность (-). Эти операторы работают как в теории множеств. Например, если у вас есть наборы A и B, то результатом операторов будет:

- A | B: все элементы в наборе A или наборе B
- A & B: все элементы в наборе A и B.
- A - B: все элементы в наборе A, но не в B.

Пример

```
/* Resulting in a range starting at 1000 and ending at 2FFF, in memory Mem */
/* В результате получается диапазон от 1000 до 2FFF в памяти Mem.  */
Mem:[from 0x1000 to 0x1FFF] | Mem:[from 0x1500 to 0x2FFF]
/* Resulting in a range starting at 1500 and ending at 1FFF, in memory Mem */
/* В результате получается диапазон от 1500 до 1FFF в памяти Mem.  */
Mem:[from 0x1000 to 0x1FFF] & Mem:[from 0x1500 to 0x2FFF]
/* Resulting in a range starting at 1000 and ending at 14FF, in memory Mem */
/* В результате получается диапазон от 1000 до 14FF в памяти Mem.  */
Mem:[from 0x1000 to 0x1FFF] - Mem:[from 0x1500 to 0x2FFF]
/* Resulting in two ranges. The first starting at 1000 and ending at 1FFF,
the second starting at 2501 and ending at 2FFF. Both located in memory Mem */
/* В результате в двух диапазонах. Первый начинается с 1000 и заканчивается 1FFF,
второй начинается с 2501 и заканчивается 2FFF. Оба находятся в памяти Mem  */
Mem:[from 0x1000 to 0x2FFF] - Mem:[from 0x2000 to 0x24FF]
```

Empty region (Свободная область)

```
[ ]
```

Пустая область не содержит диапазонов памяти. Если пустая область используется в директиве размещения, которая фактически используется для размещения одного или нескольких разделов, ILINK выдаст ошибку.

Пример

```
define region Code = Mem:[from 0 size 0x10000];
if (Banked)    {
               define region Bank = Mem:[from 0x8000 size 0x1000];
               }
else           {
               define region Bank = [];
               }
define region NonBanked = Code - Bank;
/* Depending on the Banked symbol, the NonBanked region is either one range with
0x10000 bytes, or two ranges with 0x8000 and 0x7000 bytes, respectively. */
/* В зависимости от символа банка, регион вне банка представляет собой либо один
диапазон с байтами 0x10000, либо два диапазона с байтами 0x8000 и 0x7000 соответственно.  */
```

Section handling (Работа с секциями)

Обработка разделов описывает, как ILINK должен обрабатывать разделы исполняемого образа, что означает:

• Placing sections in regions (Размещение разделов в регионах)

Директивы ***place at*** и ***place in*** помещают наборы разделов с похожими атрибутами в ранее определенные области. См. ***place at directive*** на странице 515 и ***place in directive*** на странице 517.

• Making sets of sections with special requirements (Изготовление комплектов секций с особыми требованиями)

Директива ***block*** позволяет создавать empty sections (пустые разделы) с определенными или расширяющимися размерами, определенными выравниваниями, последовательно отсортированными разделами разных типов и т. д.

Директива ***overlay*** позволяет создать область памяти, которая может содержать несколько overlay images (изображений наложения). Смотрите директиву ***define block***, стр. 505, и ***define overlay*** директиву, стр. 510.

• Initializing the application (Инициализация приложения).

Директивы инициализируют но не управляют запуском приложения. С помощью этих директив приложение может инициализировать глобальные символы при запуске и копировать фрагменты кода. Инициализаторы могут храниться несколькими способами, например, они могут быть сжаты. См. ***initialize directive***, стр. 511 и ***do not initialize directive***, стр. 514.

• Keeping removed sections (Сохранение удаленных разделов)

Директива ***keep*** сохраняет разделы, даже если на них не ссылается остальная часть приложения, что означает, что она эквивалентна концепции ***root*** в ассемблере и компиляторе. См. ***keep directive*** на стр. 515.

• Specifying the contents of linker-generated sections (Указание содержимого разделов, созданных компоновщиком).

Директива ***define section*** может использоваться для создания определенных разделов с содержимым и вычислениями, которые доступны только во время ссылки.

• Дополнительные более специализированные директивы:

use init table directive (использовать директиву таблицы инициализации)

В этом разделе представлена подробная информация о каждой директиве компоновщика, относящейся к обработке раздела.

define block директива

```
define    [ movable ] block name
          [ with param, param... ]
                {
                extended-selectors
                }
          [ except
                {
                section-selectors
                } ];
```

где param может быть одним из:

```
size = expr
minimum size = expr
maximum size = expr
expanding size
alignment = expr
end alignment = expr
fixed order
alphabetical order
static base [basename]
```

и где остальная часть директивы выбирает разделы для включения в блок, см. ***Section selection***, стр. 518.

Параметры

name (имя)

Имя определяемого блока.

size (размер)

Настраивает размер блока. По умолчанию размер блока - это сумма его частей в зависимости от его содержимого.

minimum size (минимальный размер)

Задает нижний предел размера блока. Блок, по крайней мере, имеет такой же размер, даже если его содержимое в противном случае не потребовало бы этого.

maximum size (максимальный размер)

Задает верхний предел размера блока. Ошибка генерируется, если разделы в блоке не подходят.

expanding size (увеличивающийся размер)

Блок расширится, чтобы использовать все доступное пространство в диапазоне памяти, в котором он размещен.

alignment (выравнивание)

Задает минимальное выравнивание для блока. Если какая-либо секция в блоке имеет более высокое выравнивание, чем минимальное выравнивание, блок будет иметь это выравнивание.

end alignment (концевое выравнивание)

Задает минимальное выравнивание для конца блока. Обычно конечный адрес блока определяется его начальным адресом и его размером (который может зависеть от его содержимого), но если этот параметр используется, конечный адрес увеличивается, чтобы соответствовать указанному выравниванию, если это необходимо.

fixed order (фиксированный порядок)

Размещает разделы в указанном порядке. Каждый расширенный селектор добавляется в отдельный вложенный блок, и эти блоки хранятся в указанном порядке.

alphabetical order (алфавитный порядок)

Размещает разделы в алфавитном порядке по названию раздела. В блоках в алфавитном порядке разрешены только шаблоны выбора раздела, например, без вложенных блоков. Все разделы в конкретном блоке в алфавитном порядке должны использовать один и тот же тип инициализации (***read-only, zero-init, copy-init*** или

no-init и другие эквиваленты). Вы не можете использовать ***__section_begin*** и т. д. Для отдельных разделов, содержащихся в блоке в алфавитном порядке.

static base [basename] (статическая база [базовое имя])

Указывает, что статическая база с именем ***basename*** будет размещена в начале блока или в середине блока, в зависимости от конкретной статической базы. Код запуска должен гарантировать, что регистр, содержащий статическую базу, инициализирован правильным значением. Если есть только одна статическая база, имя можно не указывать.

block directive определяет непрерывную область памяти, которая содержит, возможно, пустой набор разделов или других блоков. Блоки без содержимого полезны для выделения места для стеков или куч. Блоки с содержимым обычно используются для группировки разделов, которые должны быть последовательными.

Вы можете получить доступ к началу, концу и размеру блока из приложения с помощью операторов ***__section_begin, __section_end*** или ***__section_size***. Если блока с указанным именем нет, но есть разделы с таким именем, компоновщик создаст блок, содержащий все такие разделы.

movable blocks (подвижные блоки) предназначены для использования с независимостью позиции только для чтения и чтения-записи. Создание перемещаемых блоков позволяет компоновщику проверять использование адресов приложением. Подвижные блоки расположены точно так же, как и другие блоки, но компоновщик проверяет, используются ли соответствующие перемещения при обращении к символам в подвижных блоках.

Блоки с увеличивающимся размером чаще всего используются для куч или стеков.

Примечание. Вы не можете разместить блок с увеличивающимся размером внутри другого блока с увеличивающимся размером, внутри блока с максимальным размером или внутри наложения.

Пример

```
/* Create a block with a minimum size for the heap that will use
all remaining space in its memory range */
/* Создайте блок с минимальным размером кучи, который будет использовать
все оставшееся пространство в своем диапазоне памяти. */

define block HEAP with minimum size = 4K, expanding size, alignment = 8 { };
```

См. Также «Взаимодействие между инструментами и вашим приложением», стр. 219. Пример доступа см. В разделе overlay directive, стр. 510.

define section директива

```
define [ root ] section name
  [ with alignment = sec-align ]
{
  section-content-item...
};
```

где каждый section-content-item (элемент-содержимого-раздела) может быть одним из:

```
udata8 { data | string };
sdata8 data [ ,data ] ...;
udata16 data [ ,data ] ...;
sdata16 data [ ,data ] ...;
udata24 data [ ,data ] ...;
sdata24 data [ ,data ] ...;
udata32 data [ ,data ] ...;
sdata32 data [ ,data ] ...;
udata64 data [ ,data ] ...;
sdata64 data [ ,data ] ...;
pad_to data-align;
[ public ] label:
if-item;
```

где ***if-item***:

```
if ( condition ) {
  section-content-item...
[} else if (condition] {
  section-content-item... ]...
[} else {
  section-content-item...]
}
```

name Название раздела.

sec-align Выравнивание раздела, выражение.

root По желанию. Если указан root, раздел всегда включается, даже если на него нет ссылки.

udata8 {data|string}; Если параметр является выражением (данными), он генерирует однобайтовый элемент без знака в разделе. Выражение данных оценивается только при перемещении и только в том случае, если значение необходимо. Если значение данных слишком велико, чтобы поместиться в байт, это вызывает ошибку перемещения,. Возможный диапазон значений от 0 до 0xFF.

Если параметр является строкой в кавычках, он генерирует один однобайтовый член в разделе для каждого символа в строке.

sdata8 data; Как данные udata8, за исключением того, что он генерирует однобайтовый член со знаком.

Возможный диапазон значений от –0x80 до 0x7F.

udata16 data; Как sdata8, за исключением того, что он генерирует беззнаковый двухбайтовый член. Возможный диапазон значений от 0 до 0xFFFF.

sdata16 data; Как sdata8, за исключением того, что он генерирует подписанный двухбайтовый член. Возможный диапазон значений от –0x8000 до 0x7FFF.

udata24 data; Как sdata8, за исключением того, что он генерирует беззнаковый трехбайтовый член. Возможный диапазон значений от 0 до 0xFFFFFF.

sdata24 data; Как sdata8, за исключением того, что он генерирует трехбайтовый член со знаком. Возможный диапазон значений от –0x800000 до 0x7FFFFF.

udata32 data; Как sdata8, за исключением того, что он генерирует беззнаковый четырехбайтовый член. Возможный диапазон значений от 0 до 0xFFFFFFFF.

sdata32 data; Как sdata8, за исключением того, что он генерирует подписанный четырехбайтовый член. Возможный диапазон значений: от –0x80000000 до 0x7FFFFFFF.

udata64 data; Как sdata8, за исключением того, что он генерирует беззнаковый восьмибайтовый член. Возможный диапазон значений от 0 до 0xFFFFFFFFFFFFFFFF.

sdata64 data; Как sdata8, за исключением того, что он генерирует подписанный восьмибайтовый член. Возможный диапазон значений: от -0x8000000000000000 до 0x7FFFFFFFFFFFFFFF.

pad_to data_align; Создает байты заполнения, чтобы текущее смещение от начала раздела было выровнено по выражению data-align.

[public] label: Определяет метку при текущем смещении от начала раздела. Если указано ***public***, метка видна другим программным модулям. В противном случае он виден только другим выражениям данных в файле конфигурации компоновщика.

if-item Выбор элементов во время конфигурации.

condition Выражение.

data Выражение, которое оценивается только при перемещении и только в том случае, если требуется значение.

Используйте директиву ***define section*** для создания разделов с содержимым, которое недоступно на языке ассемблера или C / C ++. Примерами этого являются результаты анализа использования стека, размера блоков и арифметических операций, которые не существуют как перемещения.

Предполагается, что неизвестные идентификаторы в выражениях данных являются метками.

Примечание. Только выражения данных могут использовать метки, результаты анализа использования стека и т. д. Все остальные выражения вычисляются сразу после чтения файла конфигурации.

Пример

```
define section data {
        /* The application entry in a 16-bit word, provided it is less than 256K and 4-byte aligned. */
        /* Метка приложения в 16-битном слове при условии, что оно меньше 256 КБ
        и выровнено по 4 байта.  */
  udata16 __iar_program_start >> 2;
        /* The maximum stack usage in the program entry category. */
        /* Максимальное использование стека в категории метки программы . */
  udata16 maxstack("Application entry");
        /* The size of the DATA block Размер блока DATA */
  udata32 size(block DATA);
};
```

define overlay директива

```
define overlay name [ with param, param... ]
        {
        extended-selectors;
        }
```

```
[ except
        {
        section-selectors
        } ];
```

Информацию о расширенных селекторах и предложениях except см. В разделе ***Section selection***, стр. 518.

name Название оверлея.

size Настраивает размер наложения. По умолчанию размер наложения - это сумма его частей в зависимости от его содержимого.

maximum size Задает верхний предел размера наложения. Ошибка генерируется, если разделы в наложении не подходят.

alignment Задает минимальное выравнивание для наложения. Если какой-либо раздел наложения имеет более высокое выравнивание, чем минимальное выравнивание, наложение будет иметь это выравнивание.

fixed order (фиксированный порядок) Размещает разделы в фиксированном порядке - если не указан, порядок разделов будет произвольным.

Директива ***overlay*** определяет именованный набор разделов. В отличие от директивы ***block***, директива ***overlay*** может определять одно и то же имя несколько раз. Затем каждое определение будет сгруппировано в памяти в том же месте, что и все другие определения с тем же именем. Это создает наложенную область памяти, что может быть полезно для приложения, имеющего несколько независимых субприложений.

Поместите каждый образ субприложения в ПЗУ и зарезервируйте оверлейную область ОЗУ, которая может содержать все субприложения. Чтобы выполнить вспомогательное приложение, сначала скопируйте его из ПЗУ в оверлей ОЗУ.

Примечание: ILINK не помогает вам управлять взаимозависимыми определениями оверлеев, кроме создания диагностического сообщения для любой ссылки из одного оверлея на другой оверлей.

Размер оверлея будет таким же, как и наибольшее определение этого имени оверлея, а требования к выравниванию будут такими же, как и для определения с наивысшими требованиями к выравниванию.

Примечание: разделы, которые были наложены, должны быть разделены на части RAM и ROM, и вы должны позаботиться обо всем необходимом копировании.

Код в оверлейных областях памяти не может быть отлажен; отладчик C-SPY не может определить, какой код загружен в данный момент.

См. также Manual initialization, стр. 119.

initialize директива

```
initialize { by copy | manually }
        [ with param, param... ]
                {
                section-selectors
                }
        [ except
                {
                section-selectors
                } ];
```

где ***param*** может быть одним из:
packing = algorithm (упаковка = алгоритм)
simple ranges (простые диапазоны)
complex ranges (сложные диапазоны)
no exclusions (без исключений)

Информацию о селекторах разделов и предложениях except см. в разделе Section selection, стр. 518.

Параметры

by copy Разделение раздела на разделы для инициализаторов и инициализированных данных и автоматическая обработка инициализации при запуске приложения.

manually (вручную) Разбивает раздел на разделы для инициализаторов и инициализированных данных. Инициализация при запуске приложения не выполняется автоматически.

algorithm (алгоритм) Определяет, как обрабатывать инициализаторы. Выбери между:

none - Отключает сжатие содержимого выбранного раздела. Это метод инициализации вручную по умолчанию.

zeros - сжимает последовательные байты с нулевым значением.

packbits - сжимается с помощью алгоритма ***PackBits***. Этот метод дает хорошие результаты для данных с большим количеством идентичных последовательных байтов.

lz77 - Сжимает по алгоритму ***Lempel-Ziv-77***. Этот метод хорошо обрабатывает большее количество входных данных, но имеет немного больший декомпрессор.

auto - ILINK оценивает результирующий размер, используя каждый метод упаковки (кроме авто), а затем выбирает метод упаковки, обеспечивающий наименьший предполагаемый размер. Обратите внимание, что размер декомпрессора также включен. Это метод по умолчанию для инициализации копированием.

smallest (самый маленький) - это синоним авто.

Initialize директива разделяет каждый выбранный раздел на два: раздел, содержащий данные инициализатора, и раздел, содержащий пространство для инициализированных данных. Раздел, содержащий пространство для инициализированных данных, сохраняет исходное имя раздела, а раздел, содержащий данные инициализатора, получает суффикс имени ***_init***. Вы можете выбрать, должна ли инициализация при запуске выполняться автоматически (initialize by copy) или вы должны обрабатывать ее самостоятельно (initialize manually).

Когда вы используете автоматический метод упаковки (по умолчанию для инициализации копированием), ILINK автоматически выбирает соответствующий алгоритм упаковки для инициализаторов. Чтобы переопределить это, укажите другой метод упаковки. Параметр инициализации ***--log*** показывает, как ILINK решает, какой алгоритм упаковки использовать.

Когда инициализаторы сжимаются, к изображению автоматически добавляется декомпрессор.

У каждого декомпрессора есть два варианта: один, который может обрабатывать только один исходный и целевой диапазон за раз, и другой, который может обрабатывать более сложные случаи. По умолчанию компоновщик выбирает ва-

риант распаковщика в зависимости от того, определяют ли связанные директивы размещения разделов одно- или многодиапазонную область памяти. В общем, это желаемое поведение, но вы можете использовать модификатор со сложными диапазонами или модификатор с простыми диапазонами в директиве инициализации, чтобы указать, какой вариант декомпрессора использовать. Вы также можете использовать параметр командной строки ***--default_to_complex_ranges***, чтобы директивы инициализации по умолчанию использовали сложные диапазоны. Декомпрессоры простых диапазонов обычно на сотни байтов меньше, чем варианты сложных диапазонов.

Когда инициализаторы сжимаются, точный размер сжатых инициализаторов неизвестен, пока не станет известно точное содержимое несжатых данных. Если эти данные содержат другие адреса, и некоторые из этих адресов зависят от размера сжатых инициализаторов, компоновщик выдает ошибку ***Lp017***. Чтобы избежать этого, помещайте сжатые инициализаторы в последнюю очередь или в область памяти вместе с разделами, адреса которых не нужно знать.

Из-за внутренней зависимости генерация сжатых инициализаторов также может завершиться ошибкой (с ошибкой ***LP021***), если адрес инициализированной области зависит от размера ее инициализаторов. Чтобы избежать этого, поместите инициализаторы и инициализированную область в разные части памяти (например, инициализаторы помещаются в ПЗУ, а инициализированная область - в ОЗУ).

Если вы укажете параметр ***no exclusions*** (без исключений), выдается ошибка, если какие-либо разделы исключены (поскольку они необходимы для инициализации). никакие исключения нельзя использовать только с инициализацией копированием (автоматическая инициализация), но не с инициализацией вручную.

Если initialize manually не используется, ILINK организует инициализацию во время запуска системы, включая таблицу инициализации. Код запуска вызывает процедуру инициализации, которая читает эту таблицу и выполняет необходимые инициализации.

На секции с нулевой инициализацией не влияет директива инициализации.

Директива инициализации обычно используется для инициализированных переменных, но может использоваться для копирования любых разделов, например, для копирования исполняемого кода из медленного ПЗУ в быстрое ОЗУ или для оверлеев. Другой пример смотрите в директиве define overlay, стр. 510.

Директива initialize by copy не влияет на разделы, необходимые для инициализации. Сюда входит функция ***__low_level_init*** и все, на что она ссылается.

Все, что доступно из метки входа программы, считается необходимым для инициализации, если оно не достигнуто через фрагмент раздела с меткой, начинающейся с ***__iar_init $$ done***. Параметр --log разделы, помимо записи в журнал маркировки фрагментов разделов, подлежащих включению в приложение, также регистрирует процесс определения, какие разделы необходимы для инициализации.

```
/* Copy all read-write sections automatically from ROM to RAM at program start */
/* Автоматическое копирование всех разделов чтения-записи из ПЗУ в ОЗУ при запуске программы */
initialize by copy { rw };
place in RAM { rw };
place in ROM { ro };
```

См. также Initialization at system startup, стр. 102, и директива do not initialize directive, стр. 514.

do not initialize директива

```
do not initialize
    {
    section-selectors
    }
[ except
     {
    section-selectors
    } ];
```

Информацию о section selectors (селекторах разделов) и предложениях except см. в разделе Section selection (Выбор раздела), стр. 518.

Используйте директиву ***do not initialize***, чтобы указать разделы, которые вы не хотите, чтобы код запуска системы автоматически инициализировал нулями. Директива может использоваться только в разделах с нулевым начальным значением.

Как правило, это полезно, если вы хотите обработать нулевую инициализацию каким-либо другим способом для всех или некоторых секций с нулевой инициализацией.

Это также может быть полезно, если вы хотите полностью подавить нулевую инициализацию переменных. Обычно это обрабатывается автоматически для переменных, указанных как ***__no_init*** в источнике, но если вы связываетесь с объектными файлами, созданными более старыми инструментами от IAR Systems или других поставщиков инструментов, вам может потребоваться подавить нулевую инициализацию специально для некоторых разделов.

```
/* Do not initialize read-write sections whose name ends with _noinit at program start */
/* Не инициализировать разделы чтения-записи, имена которых заканчиваются на _noinit,
при запуске программы */
do not initialize { rw section .*_noinit };
place in RAM { rw section .*_noinit };
```

См. также Initialization at system startup, стр. 102, и initialize directive, стр. 511.

keep директива

```
keep
        {
        [ { section-selectors | block name }
                [ , {section-selectors | block name }... ] ]
        }
        [ except
        {
        section-selectors
        } ];
```

Информацию о selectors and except clauses см. В Section selection, стр. 518.

Директива ***keep*** может использоваться для включения блоков, оверлеев или разделов в исполняемый образ, которые в противном случае были бы отброшены, поскольку во включенных частях приложения не существует ссылок на них. Обратите внимание, что эта директива всегда приводит к включению целых входных разделов, а не только соответствующего фрагмента раздела, при сопоставлении с именем символа.

При этом рассматриваются только разделы из включенных модулей. Директива ***keep*** не требует включения дополнительных модулей в ваше приложение.

Чтобы включить модуль, который определяет определенный символ, или только фрагмент раздела, который определяет символ, используйте опцию компоновщика **Keep symbols** (или опцию ***--keep*** в командной строке) или директиву компоновщика ***keep symbol***.

Пример

```
keep { section .keep* } except {section .keep};
```

place at директива

```
[ "name": ]
place [ noload ] at { address [ memory: ] address |
    start of region_expr [ with mirroring to mirror_address ] |
    end of region_expr [ with mirroring to mirror_address ] }

{
  extended-selectors
}
[ except
    {
    section-selectors
    } ];
```

Информацию о extended selectors (расширенных селекторах) и предложениях except см. в разделе Section selection, стр. 518.

name По желанию. Если он указан, он используется в файле карты, в некоторых сообщениях журнала и является частью имени любых выходных разделов ELF, являющихся результатом директивы.

noload По желанию. Если он указан, он предотвращает загрузку разделов директивы в целевую систему. Чтобы использовать разделы, вы должны поместить их в целевую систему каким-либо другим способом. ***noload*** можно использовать, только если указано имя.

memory: address Определенный адрес в определенной памяти. Адрес должен быть доступен в предоставленной памяти, определенной директивой ***define memory***. Спецификатор памяти не является обязательным, если есть только одна память.

start of region_expr Выражение региона, которое приводит к единственному внутреннему региону. Используется начало интервала.

end of region_expr Выражение региона, которое приводит к единственному внутреннему региону. Используется конец интервала.

mirror_address Если указано с зеркалированием на, предполагается, что содержимое любых разделов зеркально отражается на этот адрес, поэтому отладочная информация и символы будут отображаться в зеркальном диапазоне, но фактические байты содержимого помещаются так, как если бы зеркалирование на не было указано.

Примечание. Эта функция предназначена для поддержки внешнего (target-specific - целевого) зеркалирования.

Директива ***place at*** размещает разделы и блоки либо по определенному адресу, либо в начале или в конце региона. Один и тот же адрес не может использоваться для двух разных мест в директивах. Также невозможно использовать пустую область в месте в директиве. При размещении в регионе разделы и блоки будут размещены перед любыми другими разделами или блоками, размещенными в том же регионе с указанием места в директиве.

Примечание: с зеркалированием в можно использовать только вместе с началом и концом.

```
/* Place the RO section .startup at the start of code_region */
/* Поместите раздел RO .startup в начало code_region */
"START": place at start of ROM { readonly section .startup };
```

См. также place in directive, стр. 517.

place in директива

```
[ "name": ]
place [ noload ] in region-expr
                [ with mirroring to mirror_address ]
{
  extended-selectors
}
        [ except{
        section-selectors
        } ];
```

где ***region-expr*** - это выражение региона, см. также Regions, стр. 501.

и где остальная часть директивы выбирает разделы для включения в блок. См. раздел Section selection, стр. 518.

name По желанию. Если он указан, он используется в файле карты, в некоторых сообщениях журнала и является частью имени любых выходных разделов ELF, являющихся результатом директивы.

noload По желанию. Если он указан, он предотвращает загрузку разделов директивы в целевую систему. Чтобы использовать разделы, вы должны поместить их в целевую систему каким-либо другим способом. noload можно использовать, только если указано имя.

mirror_address Если указано с зеркалированием на, предполагается, что содержимое любых разделов зеркально отражается на этот адрес, поэтому отладочная информация и символы будут отображаться в зеркальном диапазоне, но фактиче-

ские байты содержимого помещаются так, как если бы зеркалирование на не было указано.

Примечание. Эта функция предназначена для поддержки внешнего (целевого) зеркалирования.

place in директива размещает разделы и блоки в определенном регионе. Разделы и блоки будут размещены в регионе в произвольном порядке.

Чтобы указать конкретный порядок, используйте ***block directive*** (директиву блока). В регионе может быть несколько диапазонов.

Примечание. Если указано с зеркалированием в, выражение-регион должно приводить к одному диапазону.

```
/* Place the read-only sections in the code_region */
/* Поместите разделы read-only в code_region  */
"ROM": place in ROM { readonly };
```

См. также place at directive, стр. 515.

use init table директива

```
use init table name for
    {
    section-selectors
    }
    [ except
        {
        section-selectors
        } ];
```

Информацию о section selectors и предложениях except см. в разделе Section selection, стр. 518.

Параметры

name Имя таблицы инициализации.

Обычно все записи инициализации генерируются в единую таблицу инициализации (называемую таблицей). Используйте эту директиву, чтобы некоторые записи помещались в отдельную таблицу. Затем вы можете использовать эту таблицу инициализации в другое время или при других обстоятельствах, чем обычная таблица инициализации.

Записи инициализации для всех переменных, не упомянутых в директиве use init table, помещаются в обычную таблицу инициализации. Имея несколько директив таблиц инициализации, вы можете иметь несколько таблиц инициализации.

Доступ к началу, концу и размеру таблицы инициализации можно получить в прикладной программе с помощью ***__section_begin, __section_end*** или ***__section_size "Region$$name"***, соответственно, или с помощью символов ***Region $$ name $$ Base, Region $ $ name $$ Limit*** и ***Region $$ name $$ Length***.

```
use init table Core2 for { section *.core2};
/* __section_begin("Region$$Core2") can be used to get the start   of the Core2 init table. */
/* __section_begin ("Region $$ Core2") может использоваться для получения начала таблицы инициализации Core2.  */
```

Section selection (Выбор раздела)

Цель выбора раздела - указать - с помощью селекторов разделов и предложений except - разделы, к которым должна применяться директива ILINK. Все разделы, соответствующие одному или нескольким селекторам разделов, будут выбраны, и ни один из селекторов разделов в предложении except, если таковой имеется. Каждый селектор раздела может сопоставлять разделы по атрибутам раздела, имени раздела и имени объекта или библиотеки.

Некоторые директивы предоставляют функциональные возможности, требующие более детальных возможностей выбора, например директивы, которые могут применяться как к разделам, так и к блокам. В этом случае используются расширенные селекторы.

В этом разделе представлена подробная информация о каждой директиве компоновщика, относящейся к выбору раздела.

section-selectors

```
[ section-selector [ , section-selector... ] ]
```

раздел-селектор:

```
[ section-attribute ][ section-type ]
[ symbol symbol-name ][ section section-name ]
[ object module-spec ]
```

атрибут раздела:

```
ro [ code | data ] | rw [ code | data ] | zi
```

Тип раздела:

```
[ preinit_array | init_array ]
```

Параметры

section-attribute

Будут выбраны только разделы с указанным атрибутом.

раздел-атрибут может состоять из:

ro|readonly, для разделов ROM.

rw|readwrite, для разделов RAM.

В каждой категории разделы можно разделить на те, которые содержат код, и те, которые содержат данные, в результате чего можно выделить четыре основные категории:

ro code, для нормального кода

ro data, для констант

rw code, для кода, скопированного в RAM

rw data, для переменных

readwrite data также есть подкатегория—

zi|zeroinit—для разделов, которые инициализируются нулем при запуске приложения.

section-type

Будут выбраны только разделы с этим типом раздела ELF. ***section-type*** может быть:

pre_init array области предварительной инициализации типа раздела ELF ***SHT_PREINIT_ARRAY.***
init_array области инициализации типа раздела ELF ***SHT_INIT_ARRAY.***

symbol symbol-name

Будут выбраны только те разделы, которые определяют хотя бы один общедоступный символ, соответствующий шаблону имени символа. имя-символа - это шаблон имени символа. Допускаются два подстановочных знака:

? соответствует любому одиночному символу.

* соответствует нулю или более символам.

section section-name

Будут выбраны только разделы, имена которых соответствуют имени раздела. Допускаются два подстановочных знака:

? соответствует любому одиночному символу

* соответствует нулю или более символам.

object module-spec

Будут выбраны только те разделы, которые происходят из библиотечных модулей или объектных файлов, соответствующих спецификации модуля. ***module-spec*** может быть в одной из двух форм:

module, имя в форме

objectname(libraryname). Выбираются разделы из объектных модулей, в которых имя объекта и имя библиотеки соответствуют их соответствующим шаблонам. Шаблон имени пустой библиотеки выбирает только разделы из объектных файлов. Если имя библиотеки: ***sys***, шаблон будет соответствовать только разделам из системной библиотеки.

filename, имя объектного файла или объекта в библиотеке.

Допускаются два подстановочных знака:

? соответствует любому одиночному символу

* соответствует нулю или более символам.

Селектор раздела выбирает все разделы, которые соответствуют атрибуту раздела, типу раздела, имени символа, имени раздела и имени модуля. Можно пропустить до четырех из пяти условий.

Также можно использовать только {} без каких-либо селекторов разделов, что может быть полезно при определении блоков.

Примечание. Селектор раздела с более узкой областью действия имеет более высокий приоритет, чем селектор более общего раздела. Если для одной и той же метки соответствует несколько селекторов разделов, один из них должен быть более конкретным. Селектор раздела более конкретен, чем другой, если в порядке приоритета:

- Он определяет имя символа без подстановочных знаков, а другой - нет.
- Он указывает имя раздела или имя объекта без подстановочных знаков, а другое - нет

• В нем указывается тип раздела, а в другом - нет.

• Могут быть разделы, соответствующие другому селектору, которые также соответствуют этому, однако обратное неверно.

Selector 1	Selector 2	More specific
ro	ro code	Selector 2
symbol mysym	section foo	Selector 1
ro code section f*	ro section f*	Selector 1
section foo*	section f*	Selector 1
section *x	section f*	Neither
init_array	section f*	Selector 1
section .intvec	ro section .int*	Selector 1
section .intvec	object foo.o	Neither

Table 41: Examples of section selector specifications

```
{ rw }                      /* Selects all read-write sections Выбирает все разделы rw*/
{ section .mydata* }        /* Selects only .mydata* sections Выбирает только разделы .mydata * */
/* Selects .mydata* sections available in the object special.o */
/* Выбирает разделы .mydata *, доступные в объекте special.o  */
{ section .mydata* object special.o }
```

Предполагая, что раздел в объекте с именем foo.o в библиотеке с именем lib.a, любой из этих селекторов выберет этот раздел:

```
object foo.o(lib.a)
object f*(lib*)
object foo.o
object lib.a
```

См. также initialize directive, стр. 511, not initialize directive, стр. 514 и keep directive, стр. 515.

extended-selectors

```
[ extended-selector [ , extended-selector...]]
```

где extended-selector (расширенный селектор):

```
[ first | last | midway ]
                { section-selector |
                block name [ inline-block-def ] |
                overlay name }
```

где inline-block-def:

```
[ block-params ] extended-selectors
```

Параметры

first

Помещает выбранные разделы, блок или оверлей первыми в содержащую директиву размещения, блок или оверлей.

last

Помещает выбранные разделы, блок или оверлей последними в содержащую директиву размещения, блок или оверлей.

midway (на полпути)

Помещает выбранные разделы, блок или наложение так, чтобы они находились не дальше половины максимального размера содержащего блока от любого края блока. Обратите внимание, что этот параметр можно использовать только внутри блока с максимальным размером.

name

Имя блока или оверлея.

Используйте расширенные селекторы, чтобы выбрать контент для включения в директиву размещения, блок или оверлей. Помимо использования шаблонов выбора разделов, вы также можете явно указать блоки или наложения для включения.

Используя ключевое слово ***first*** или ***last***, вы можете указать один шаблон, блок или оверлей, которое должно быть помещено первым или последним в директиве размещения, блоке или оверлее. Если вам нужен более точный контроль порядка размещения, вы можете вместо этого использовать блок с фиксированным порядком.

Блоки могут быть определены отдельно, используя директиву ***define block***, или ***inline***, как часть расширенного селектора.

Параметр ***midway*** в первую очередь полезен вместе со статической базой, которая может иметь как отрицательные, так и положительные смещения.

```
define block First { ro section .f* };
/* Define a block holding any read-only section matching ".f*" */
/* Определите блок, содержащий любую доступную только для чтения секцию,
соответствующую ".f *"  */
define block Table { first block First, ro section .b };
/* Define a block where the block First comes before the sections matching ".b*". */
/* Определите блок, в котором блок сначала стоит перед секциями,
соответствующими ".b *".  */
```

Вы также можете определить блок ***First inline*** вместо отдельной директивы ***define block***:

```
define block Table { first block First { ro section .f* },
ro section .b* };
```

См. также директиву define block на стр. 505, директиву define overlay на стр. 510 и директиву place at директиву на стр. 515.

Использование символов, выражений и чисел

В файле конфигурации компоновщика вы также можете:

- Определить и экспортировать символы.

Директива ***define symbol*** определяет символ с указанным значением, которое может использоваться в выражениях в файле конфигурации. Символ также можно экспортировать для использования приложением или отладчиком. См. Директиву ***define symbol*** на стр. 524 и ***export directive*** на стр. 525.

- Используйте выражения и числа.

В файле конфигурации компоновщика выражения и числа используются для указания адресов, размеров и т. д. См. ***expressions***, стр. 525.

В этом разделе представлена подробная информация о каждой директиве компоновщика, относящейся к определению символов, выражений и чисел.

check that директива

```
check that expression;
```

expression Логическое выражение.

Вы можете использовать директиву c***heck that*** для сравнения результатов анализа использования стека с размерами блоков и регионов. Если выражение равно нулю, выдается ошибка.

Три дополнительных оператора доступны только для проверки этих выражений:

maxstack (category) Глубина стека самой глубокой цепочки вызовов для любой корневой функции графа вызовов в категории.

totalstack(category) Сумма глубин стека самых глубоких цепочек вызовов для каждой корневой функции графа вызовов в категории.

size(block) Размер блока.

```
check that maxstack("Program entry")
              + totalstack("interrupt")
              + 1K
    <= size(block CSTACK);
```

См. также Stack usage analysis (Анализ использования стека), стр. 105.

define symbol директива

```
define [ exported ] symbol name = expr;
```

exported Экспортирует символ, который будет использоваться исполняемым изображением.

name Название символа.

expr Значение символа.

Директива ***define symbol*** определяет символ с указанным значением. Затем символ можно использовать в выражениях в файле конфигурации. Определенные таким образом символы работают точно так же, как символы, определенные с помощью опции ***--config_def*** вне файла конфигурации.

Вариант определения экспортируемого символа этой директивы - это ярлык для использования директивы ***define symbol*** в сочетании с директивой ***export symbol*** экспорта символа. В командной строке для достижения того же эффекта потребуется как параметр ***--config_def***, так и параметр ***--define_symbol***.

Примечание:

- Символ не может быть переопределен.
- Символы с префиксом ***_X***, где X - заглавная буква, или символы, содержа-

щие __ (двойное подчеркивание), зарезервированы для поставщиков наборов инструментов.

```
/* Define the symbol my_symbol with the value 4 */
/* Определите символ my_symbol со значением 4  */
define symbol my_symbol = 4;
```

См. также export directive, стр. 525 и Взаимодействие между ILINK и приложением, стр. 122.

export директива

```
export symbol name;
```

name Имя символа

Директива экспорта определяет экспортируемый символ, поэтому его можно использовать как из исполняемого образа, так и из глобальной метки. Приложение или отладчик может затем обратиться к нему для настройки и т. д.

```
/* Define the symbol my_symbol to be exported */
/* Определите символ my_symbol для экспорта  */
export symbol my_symbol;
```

expressions (выражения)

Выражение состоит из следующих составляющих:

```
expression binop expression
unop expression
expression ? expression : expression
(expression)
number
symbol
func-operator
```

где binop - один из этих бинарных операторов:

+, -, *, /, %, <<, >>, <, >, ==, !=, &, ^, |, &&, ||

где unop - один из этих унарных операторов:

+, -, !, ~

где ***number*** - это число, см. numbers, стр. 527

где ***symbol*** - это определенный символ, см. директиву define symbol, стр. 524 и --config_def, стр. 331

и где ***func-operator*** - один из этих функционально-подобных операторов, доступных во всех выражениях:

alignndown (expr, align) Значение expr, округленное в меньшую сторону до ближайшего кратного align. Выравнивание должно быть степенью двойки.

alignup (expr, align) Значение expr, округленное в большую сторону до ближайшего кратного значения align. Выравнивание должно быть степенью двойки.

end(region) Самый высокий адрес в регионе.

isdefinedsymbol (name) Истина (1), если имя символа определено, в противном случае - Ложь (0).

isempty (region) Истина (1), если область пуста, в противном случае - Ложь (0).
max (expr [, expr ...]) Самый большой из параметров.
min (expr [, expr ...]) Наименьший из параметров.
size (region) Общий размер всех диапазонов в регионе.
start (region) Самый низкий адрес в регионе.
где ***align*** и ***expr*** - выражения, а ***region*** - выражение региона, см. Region expression, стр. 502.

func-operator также может быть одним из этих операторов, которые доступны только в выражениях для размера или выравнивания блока или наложения, при проверке этих выражений и в выражениях данных в директивах ***define section***:

imp (name) Если имя - это имя символа с постоянным значением, этот оператор является этим значением. Этот оператор можно использовать вместе с ***#pragma public_equ*** (см. Public_equ, стр. 417) для импорта значений из модулей вашего приложения, например размера определенного типа структуры.

tlsalignment () Выравнивание локальной области хранения потока.
tlssize () Размер области локального хранения потока.

В файле конфигурации компоновщика выражение представляет собой 65-битное значение в диапазоне от -2 ^ 64 до 2 ^ 64. Синтаксис выражения полностью соответствует синтаксису C с некоторыми незначительными исключениями. Здесь нет присваиваний, приведений, операций до и после, а также операций с адресом (*, &, [], -> and.). Некоторые операции, извлекающие значение из выражения области и т. д., Используют синтаксис, напоминающий синтаксис вызова функции. Логическое выражение возвращает 0 (Ложь) или 1 (Истина).

keep symbol директива

```
keep symbol name;
```

name Имя символа

Обычно компоновщик сохраняет символ только в том случае, если он нужен вашему приложению. Используйте эту директиву, чтобы гарантировать, что символ всегда включен в окончательное приложение.

См. также директиву keep, стр. 515 и --keep, стр. 344.

numbers

nr [nr-suffix]
где ***nr*** - десятичное или шестнадцатеричное число (0x ... или 0X ...).
и где ***nr-suffix*** является одним из:

```
K       /* Kilo = (1 << 10) 1024 */
M       /* Mega = (1 << 20) 1048576 */
G       /* Giga = (1 << 30) 1073741824 */
T       /* Tera = (1 << 40) 1099511627776 */
P       /* Peta = (1 << 50) 1125899906842624 */
```

Число может быть выражено обычными средствами языка Си или добавлением к нему набора полезных суффиксов, что обеспечивает компактный способ указания чисел.

1024 совпадает с 0x400, что совпадает с 1K.

Structural configuration (Структурная конфигурация)

Структурные директивы предоставляют средства для создания структуры внутри конфигурации, например:

• Conditional inclusion (Условное включение)

Директива if включает или исключает другие директивы в зависимости от условия, что позволяет иметь директивы для нескольких различных конфигураций памяти в одном файле. См. if directive, стр. 528.

• Разделение файла конфигурации компоновщика на несколько разных файлов.

Директива include позволяет разделить файл конфигурации на несколько логически различных файлов. См. include directive, стр. 529.

• Сообщение об ошибке в неподдерживаемых случаях.

В этом разделе представлена подробная информация о каждой директиве компоновщика, относящейся к структурной конфигурации.

error директива

```
error string
```

string Сообщение об ошибке.

Директива ***error*** может использоваться для сигнализации об ошибке, если директива встречается в активной части условной директивы.

```
error "Unsupported configuration"
```

if директива

```
if (expr) {
        directives
[ } else if (expr) {
        directives ]
[ } else {
        directives ]
}
```

где expr - выражение, см. expressions, стр. 525.

directives Любая директива ILINK.

Директива ***if*** включает в себя или исключает другие директивы в зависимости от условия, что позволяет иметь директивы для нескольких различных конфигураций памяти, например, конфигурации как банковской, так и небанковской памяти, в одном и том же файле.

Текст внутри невыделенной части директивы if не проверяется на синтаксис. Единственное требование к такому тексту - это то, что он может быть токенизирован, и что любой токен открытой фигурной скобки ({) имеет соответствующий токен закрывающей фигурной скобки (}).

Пример см. Empty region, стр. 503.

include директива

```
include "filename";
```

filename Путь, в котором в качестве разделителя каталогов можно использовать как /, так и \.

Директива ***include*** позволяет разделить файл конфигурации на несколько логически различных частей, каждая из которых находится в отдельном файле. Например, могут быть части, которые вам нужно часто менять, и части, которые вы редко редактируете.

Обычно компоновщик ищет файлы включения конфигурации в каталоге конфигурации системы. Вы можете использовать опцию компоновщика ***--config_search***, чтобы добавить больше каталогов для поиска.

См. также --config_search, стр. 333

Section reference (Ссылка на раздел)

- Сводка разделов и блоков.
- Описание разделов и блоков.

Для получения дополнительной информации см. Modules and sections, стр. 96.

Сводка разделов и блоков

В этой таблице перечислены разделы и блоки ELF, которые используются инструментами сборки IAR:

Section (Раздел)	Описание
.bss	Хранит инициализированные нулем статические и глобальные переменные.
CSTACK	Содержит стек, используемый программами C или C ++.
.data	Содержит статические и глобальные инициализированные переменные.
.data_init	Сохраняет начальные значения для разделов .data, когда используется инициализация директивы компоновщика.
.exc.text	Содержит код, связанный с исключениями.
HEAP	Содержит кучу, используемую для динамически выделяемых данных.
__iar_tls$$DATA	Содержит начальные значения для переменных TLS.
.iar.dynexit	Содержит таблицу atexit.
.iar.locale_table	Содержит таблицу языковых стандартов для выбранных языковых стандартов.
.init_array	Содержит таблицу функций динамической инициализации.
.intvec	Содержит таблицу векторов сброса
IRQ_STACK	Содержит стек для запросов прерывания, IRQ и исключений.
.noinit	Содержит статические и глобальные переменные __no_init.
.preinit_array	Содержит таблицу функций динамической инициализации.
.prepreinit_array	Содержит таблицу функций динамической инициализации.
.rodata	Хранит данные констант.
.text	Содержит программный код.
.textrw	Содержит объявленный программный код __ramfunc.

.textrw_init	Содержит инициализаторы для объявленного раздела .textrw.
Veneer$$CMSE	Удерживает безопасные veneers межсетевого экрана.

В дополнение к разделам ELF, используемым в вашем приложении, инструменты используют ряд других разделов ELF для различных целей:

• Разделы, начинающиеся с .debug, обычно содержат отладочную информацию в формате DWARF.

• Разделы, начинающиеся с .iar.debug, содержат дополнительную отладочную информацию в формате IAR.

• Раздел .comment содержит инструменты и командные строки, используемые для создания файла.

• Разделы, начинающиеся с .rel или .rela, содержат информацию о перемещении ELF.

• Раздел .symtab содержит таблицу символов для файла.

• Раздел .strtab содержит имена символов в таблице символов.

• Раздел .shstrtab содержит названия разделов.

Описание разделов и блоков

В этом разделе содержится справочная информация о каждом разделе, где:

• Описание описывает, какой тип содержания содержит раздел и, если необходимо, как этот раздел обрабатывается компоновщиком.

• Размещение памяти описывает ограничения размещения памяти.

Для получения информации о том, как выделить разделы в памяти путем изменения файла конфигурации компоновщика, см. Размещение кода и данных - файл конфигурации компоновщика, стр. 99.

.bss Хранит инициализированные нулем статические и глобальные переменные. Этот раздел можно разместить где угодно в памяти.

CSTACK Блок, содержащий внутренний стек данных.Этот блок можно разместить в любом месте памяти.См. Также Настройка стековой памяти, стр. 118.

.data Содержит статические и глобальные инициализированные переменные. В объектных файлах сюда входят начальные значения. Когда используется инициализация директивы компоновщика, для каждого раздела .data создается соответствующий раздел .data_init, содержащий возможно сжатые начальные значения. Этот раздел можно разместить где угодно в памяти.

.data_init Содержит возможно сжатые начальные значения для разделов .data. Этот раздел создается компоновщиком, если используется директива компоновщика инициализации. Этот раздел можно разместить где угодно в памяти.

.exc.text Содержит код, который выполняется только тогда, когда ваше приложение обрабатывает исключение. В той же памяти, что и .text. См. Также Обработка исключений, стр. 206.

HEAP Содержит кучу, используемую для динамически выделяемых данных в памяти, другими словами, данные, выделенные malloc и free, а в C ++ - new и delete. Этот раздел можно разместить где угодно в памяти. См. Также Настройка кучи памяти, стр. 118.

__iar_tls$$DATA Содержит начальные значения для переменных TLS. Этот раздел создается компоновщиком, если используется параметр компоновщика --threaded_lib.

В дополнение к обычным способам доступа к началу, концу и размеру этого блока (см. Операторы выделенного раздела, стр. 200), вы также можете использовать оператор __iar_tls $$ DATA $$ Align для доступа к выравниванию локального потока. область хранения в исходном коде C / C ++. Этот раздел можно разместить где угодно в памяти. См. Также Управление многопоточной средой, стр.

.iar.dynexit Содержит таблицу вызовов, которые необходимо сделать при выходе. Этот раздел можно разместить где угодно в памяти. См. Также раздел Установка лимита atexit, стр. 118.

.iar.locale_table Содержит таблицу языковых стандартов для выбранных языковых стандартов. См. Также Locale, стр. 164.

.init_array Содержит указатели на подпрограммы, вызываемые для инициализации одного или нескольких объектов C ++ со статической продолжительностью хранения. Этот раздел можно разместить где угодно в памяти.

.intvec Содержит таблицу векторов сброса и векторы исключений, которые содержат инструкции перехода для cstartup, процедуры обслуживания прерывания и т. д. Размещение этого раздела зависит от устройства. См. Руководство производителя оборудования.

IRQ_STACK Содержит стек, который используется при обслуживании исключений IRQ. Другие стеки могут быть добавлены по мере необходимости для обслуживания других типов исключений: FIQ, SVC, ABT и UND. Файл cstartup.s необходимо изменить, чтобы инициализировать используемые указатели стека исключений. *Примечание:* этот раздел не используется при компиляции для Cortex-M. Этот раздел можно разместить где угодно в памяти. См. Также Стек исключений, стр. 216.

.noinit Содержит статические и глобальные переменные __no_init. Этот раздел можно разместить где угодно в памяти.

.preinit_array Подобно .init_array, но используется библиотекой для выполнения некоторых инициализаций C ++ раньше других. Этот раздел можно разместить где угодно в памяти. См. Также .init_array, стр. 534.

.prepreinit_array Подобно .init_array, но используется, когда статическая инициализация C переписывается как динамическая инициализация. Выполняется перед динамической инициализацией C ++. Этот раздел можно разместить где угодно в памяти. См. Также .init_array, стр. 534.

.rodata Хранит постоянные данные. Сюда могут входить постоянные переменные, строковые и агрегатные литералы и т. д. Этот раздел можно разместить где угодно в памяти.

.text Содержит программный код, включая код для инициализации системы. Этот раздел можно разместить где угодно в памяти.

.textrw Содержит объявленный программный код __ramfunc. Этот раздел можно разместить где угодно в памяти. См. Также _ _ramfunc, стр. 393.

.textrw_init Содержит инициализаторы для объявленных разделов .textrw. Этот раздел можно разместить где угодно в памяти. См. Также _ _ramfunc, стр. 393.

Veneer$$CMSE Этот раздел содержит облицовку защищенного шлюза, автоматически создаваемую компоновщиком для каждой функции входа, как определено расширенным ключевым словом __cmse_nonsecure_entry. Этот раздел должен быть помещен в область памяти NSC (незащищенный вызов). Области NSC

могут быть запрограммированы с использованием SAU (блок атрибуции безопасности) или IDAU (блок атрибута, определяемый реализацией). Для получения информации о том, как программировать SAU или IDAU, смотрите документацию к вашему ядру Armv8-M. См. Также Arm TrustZone®, стр. 232, --cmse, стр. 282, __cmse_nonsecure_entry, стр. 387, и --import_cmse_lib_out, стр. 344