Уменьшение размера кода

Размер кода всегда интересует разработчиков встраиваемых систем, поскольку это напрямую определяет стоимость устройства памяти под хранение кода. Возможность контролировать и уменьшать размер кода очень важна для контролирования этой стоимости.

Среда HAL разработана для использования только тех средств, которые вы запрашиваете, чтобы минимизировать общее покрытие кода. Если ваша аппаратная система Nios II состоит из периферии, используемой вашей программой, HAL содержит только драйверы, необходимые для контроля над аппаратными средствами.

В следующих секциях содержатся опции, которые вы рассматриваете, когда вам требуется уменьшить размер кода. Пример проекта **hello_world_small** показывает использование некоторых этих опций для уменьшения размера кода до абсолютного минимума.

Реализация опций в следующих секциях влечёт за собой изменения в настройках BSP. За подробной информацией об управлении BSP настойками обратитесь к секции "Настройки HAL BSP" на стр. 6-2.

Разрешение оптимизации компилятора

Для разрешения оптимизации компилятора используйте уровень -O3 оптимизации компилятора для **nios2-elf-gcc**. Вы можете задать эту опцию в командной строке настроек BSP.

Когда эта опция включена, компилятор Nios II компилирует код по размеру и скорости с максимально возможной оптимизацией.

Вы должны установить эту опцию для проекта BSP и для проекта приложения.

Использование уменьшенных драйверов устройств

Некоторые устройства имеют два варианта драйверов - быстрый и малый. Свойства, предоставляемые этими вариантами драйверов, зависят от устройства. Быстрый вариант является полнофункциональным, а малый вариант уменьшает кодовое покрытие.

По умолчанию, HAL всегда использует быстрый вариант драйвера. Вы можете выбрать уменьшенный драйвер устройства для всех аппаратных компонентов или для конкретного компонента через настройки HAL BSP.

В табл. 6-8 приведён список периферии Altera Nios II, которые предлагаю на текущий момент драйверы с малым размером. Опция small footprint (малый размер) может влиять и на другую периферию. Обратитесь к технической документации на каждую периферию, за деталями поведения драйверов с малым размером.

Табл. 6-8. Периферия Altera, предлагающая драйверы с малым размером

Периферия	Поведение с малым размером драйвера
UART	Операция опроса, приоритетнее, чем управляемая IRQ
JTAG UART	Операция опроса, приоритетнее, чем управляемая IRQ
Контроллер общего интерфейса с флеш памятью	Драйвер работает в режиме с малым размером кода
Контроллер модуля LCD	Драйвер работает в режиме с малым размером кода
Чип последовательной конфигурации EPCS	Драйвер работает в режиме с малым размером кода

Уменьшение накопителя файлового дескриптора

Файловые дескрипторы имеют доступ к устройствам с символьным режимом и к файлам, локализованным в накопителе файлового дескриптора. Вы можете изменить размер накопителя файлового дескриптора в настройках BSP. По умолчанию – 32.

Использование /dev/null

Во время загрузки, стандартный вход, стандартный выход и стандартный выход ошибки перенаправлены на нулевые устройства, то есть, **/dev/null**. Это означает, что вызов printf() на стадии инициализации драйвера не приводит ни к чему, и поэтому не может навредить. После того, как все драйверы инсталлированы, эти потоки перенаправляются на каналы, сконфигурированные в HAL. Покрытие кода может выполнять подобное перенаправление в малом размере, но вы можете устранить его полностью, выбрав **null** для stdin, stdout и stderr. Такой выбор означает, что вы хотите убрать все данные, передаваемые на стандартный выход или стандартный выход ошибки, а ваша программа никогда не будет принимать данные через stdin. Вы можете контролировать назначения для каналов stdin, stdout и stderr, манипулируя настройками BSP.

Использование малого файла I/O библиотеки

Использование малой Си библиотеки newlib

Полная стандартная библиотека newlib ANSI Си, возможно, не нужна для страиваемых систем. Коллекция компилятора GNU (GCC) предлагает уменьшенную реализацию стандартной библиотеки newlib ANSI Си, исключая излишние для страиваемых систем средства newlib. Реализация малой newlib требует малого покрытия кода. Когда вы используете **nios2-elf-gcc** в командной строке, опция **-msmallc** разрешает малую библиотеку Си.

Вы можете выбрать малую библиотеку Си в настройках BSP. В табл. 6-9 обобщены ограничения реализации малой библиотеки newlib Си Nios II.

Табл. 6-9. Ограничения малой библиотеки newlib Си Nios II (часть 1 из 2)

Ограничения	Влияние на функции
Нет поддержки плавающей точки для семейства функций printf(). Перечисленные функции поддерживаются, но опции %f и %g не поддерживаются (1).	asprintf() fiprintf() fprintf() iprintf() printf() siprintf() snprintf() sprintf()
Нет поддержки плавающей точки для семейства функций vprintf(). Перечисленные функции поддерживаются, но опции %f и %g не поддерживаются.	vasprintf() vfiprintf() vfprintf() vprintf() vsnprintf() vsprintf()
Нет поддержки семейства функций scanf(). Перечисленные функции не поддерживаются.	fscanf() scanf() sscanf() vfscanf() vscanf() vsscanf()
Нет поддержки поиска. Перечисленные функции не поддерживаются.	fseek() ftell()
Нет поддержки открытия/ закрытия FILE *. Только предоткрытие stdout, stderr и stdin доступны. Перечисленные функции не поддерживаются.	fopen() fclose() fdopen() fcloseall() fileno()

Табл. 6-9. Ограничения малой библиотеки newlib Си Nios II (часть 2 из 2)

Ограничения	Влияние на функции
Нет буферизации выходных функций **stdio.h**.	функции, поддерживаемые без буферизации: fiprintf() fputc() fputs() perror() putc() putchar() puts() printf() неподдерживаемые функции: setbuf() setvbuf()
Нет буферизации входных функций **stdio.h**. Перечисленные функции не поддерживаются.	fgetc() gets() fscanf() getc() getchar() gets() getw() scanf()
Нет поддержки локализации.	setlocale() localeconv()
Нет поддержки C++, поскольку функции в этой таблице не поддерживаются.	

Примечание к табл. 6-9:

(1) Эти функции выходят за границы Nios II. GCC не поддерживает их реализацию в малой библиотеке Си newlib.

Малая библиотека Си newlib не поддерживает MicroC/OS-II.

За подробной информацией о GCC малой библиотеки Си newlib, обратитесь к документации newlib, инсталлированной с Nios II EDS. В меню **Пуск > Программы> Altera** > **Nios II** > **Nios II Documentation**.

Реализация малой библиотеки newlib Си Nios II слегка отличается от GCC. В табл. 6-9 приведены детали отличий.

Использование I/O файлов в стиле UNIX

Если вам требуется ещё больше уменьшить размер кода, вы можете исключить библиотеку Си newlib и использовать API в стиле UNIX. За подробной информацией обратитесь к секции "Интерфейс в стиле UNIX" на стр. 6-5.

Nios II EDS предлагает I/O файлы ANSI Си в библиотеке newlib Си, поскольку они обладают избыточными характеристиками к доступу, ассоциированному с доступными устройствами и файлами, используя I/O файлы функций в стиле UNIX. I/O файлы ANSI Си предлагают буферизированный доступ, поэтому сокращается общее количество выполняемого аппаратного доступа к I/O. Также ANSI C API более гибка, и поэтому, более удобна в использовании. Однако, эти удобства увеличивают размер кода.

Эмуляция функций ANSI Си

Если вы выбрали полный отказ от реализации newlib, но вам требуется ограниченный набор функций в стиле ANSI, вы можете реализовать их, просто используя функции в стиле UNIX. Код в примере 6-15 показывает пример простой реализации небуферизированной функции getchar().

Example 6–15. Unbuffered getchar()

```
/* getchar: unbuffered single character input */
int getchar ( void )
{
  char c;
  return ( read ( 0, &c, 1 ) == 1 ) ? ( unsigned char ) c : EOF;
}
```

Этот пример взят из книги "Язык программирования Си, вторая редакция" Brian W.Kernighan и Dennis M. Ritchie. Эта книга содержит множество других полезных функций.

Использование облегчённого API драйвера устройства

Облегчённый API драйвер устройства позволяет вам минимизировать избыточность при обращении к драйверам устройств. Сам по себе он не влияет на размер драйвера, но он допускает вам исключить ненужные средства API драйвера, уменьшая общий размер HAL кода.

Облегчённый API драйвер устройства доступен для устройств с символьным режимом. Следующие устройства поддерживают облегчённый API драйвер устройства:

- JTAG UART
- UART
- Optrex 16207 LCD

Для этих устройств, облегчённый API драйвер сохраняет пространство кода, игнорируя таблицу динамического файлового дескриптора, и заменяя её тремя статичными файловыми дескрипторами, соответственно stdin, stdout и stderr. Функции библиотеки, относящиеся к открытию, закрытию и управлению файловыми дескрипторами не доступны, зато доступны все другие функции библиотеки. Вы можете ссылаться на stdin, stdout и stderr как на любой другой файловый дескриптор. Вы можете также ссылаться на следующие предопределённые номера файлов:

```
#define STDIN 0
#define STDOUT 1
#define STDERR 2
```

Эта опция подходит, если в вашей программе есть ограниченная потребность в I/O файлах. Файловая система Altera, расположенная в хост-машине, и файловая система только для чтения Altera zip не доступны при использовании уменьшенного API драйвера устройств. Вы можете выбрать уменьшение драйвера устройства в настройках BSP.

По умолчанию, облегчённый API драйвер устройства запрещён.

За подробной информацией об облегчённом API драйвере устройства, обратитесь к главе "Разработка драйверов устройств для слоя аппаратной абстракции" в настольной книге программиста.

Использование минимального API в символьном режиме

Если вы можете ограничить себе использование I/O с символьным режимом для очень простых средств, вы можете уменьшить размер кода, используя минимальный API в символьном режиме. Такой API состоит из следующих функций:

- alt_printf()
- alt_putchar()
- alt_putstr()
- alt_getchar()

Эти функции подходят для программ, которым требуется принимать строки команд и посылать простые текстовые сообщения. Большинство их полезно только совместно с облегчённым API драйвером устройства, описанным в секции "Использование облегчённого API драйвера устройства" на стр. 6-34.

Для использования минимального API в символьном режиме, включите в проект заголовочный файл **sys/alt_stdio.h**.

В следующих секциях в общих чертах описано влияние этих функций на размер кода.

alt_printf()

Эта функция похожа на printf(), но поддерживает только %c %s, %x и %% строки подстановки. Функция alt_printf() занимает значительно меньше размера кода, чем printf(), независимо от того, выбрали ли вы облегчённый API драйвер устройства или нет. Функция alt_printf() занимает менее 1 Кб с уровнем оптимизации компилятора -O2.

alt_putchar()

Эквивалентно putchar(). Совместно с облегчённым API драйвером устройства эта функция значительно уменьшает размер кода. При отсутствии облегчённого API, она вызывает putchar().

alt_putstr()

Похожа на puts() за исключением того, что она не может добавлять символ новой строки для строки. Совместно с облегчённым API драйвером устройства эта функция значительно уменьшает размер кода. При отсутствии облегчённого API, она вызывает puts().

alt_getchar()

Эквивалентно getchar (). Совместно с облегчённым API драйвером устройства эта функция значительно уменьшает размер кода. При отсутствии облегчённого API, она вызывает getchar ().

За дополнительной информацией об использовании этих функций, обратитесь к главе "Справка по HAL API" в настольной книге программиста Nios II.

Исключение неиспользованных драйверов устройств

Если аппаратное устройство представлено в системе, то по умолчанию разработка программы Nios II подразумевает потребность в драйвере устройства, и соответствующей конфигурации HAL BSP. Если HAL может найти соответствующий драйвер, он создает элемент из этого драйвера. Если ваша программа никогда не обращается к устройству, система всё равно может затрачивать ресурсы на инициализацию драйвера устройства.

Если аппаратная часть содержит устройство, никогда не используемое программой, рассмотрите возможность удаления этого устройства из аппаратной части. Это сократит размер кода и использование ресурсов FPGA.

Однако, имеются случаи, когда устройство должно быть представлено в системе, но рабочей программе не требуется драйвер. Показательный пример – флеш память. Пользовательская программа может загружаться из флеш памяти, но не может использовать её на стадии прогона программы; таким образом, ей не нужен флеш драйвер. Вы можете выборочно исключать любой драйвер, выбирать специфическую версию драйвера или заменять его собственным драйвером.

За дополнительной информацией о контроле над конфигурацией драйверов, обратитесь к главе "Инструмент разработки программы Nios II" в настольной книге программиста Nios II.

Другой способ контролировать процесс инициализации драйвера устройства – это использовать автономную среду. Подробнее в секции "Последовательность загрузки и точка входа" на странице 6-37.

Исключение ненужного кода завершения

HAL вызывает функцию exit() при выключении системы, чтобы предоставить чистый выход из программы. Функция exit() выключает из работы все внутренние I/O буферы библиотеки Си и вызывает любые C++ функции, регистрированные вместе с atexit(). В частности exit() вызывается при выходе из main(). Две опции HAL позволяют вам минимизировать или исключить этот код.

Исключение чистого выхода

Чтобы избежать избыточности, связанной с предоставлением чистого выхода, ваша программа должна использовать функцию _exit() вместо exit(). Этой функции не требуется изменение исходного кода. Вы можете выбрать функцию _exit() в настройках BSP.

Исключение всего кода завершения

Множество встраиваемых систем никогда не завершают работы. В таких случаях код завершения не нужен. Вы можете исключить весь код завершения в настройках BSP.

Если вы разрешаете эту опцию, проследите, чтобы ваша функция main() (или alt_main()) не возвращала значения.

Выключение поддержки C++

По умолчанию, HAL поддерживает программы C++, включая конструкции и деструкторы по умолчанию. Вы можете запретить поддержку C++ в настройках BSP.