Пример расширенного копировщика загрузки

В этой секции описывается пример расширенного копировщика загрузки. Вы можете собрать этот пример для запуска либо из CFI флеш, либо из памяти внутри чипа, а образы загрузки хранить во флеш чипах CFI или EPCS. Пример описан на Си и хорошо комментирован, упрощая его модификацию под собственные нужды. Этот пример содержит следующие средства, в дополнении к тем, что предлагает копировщик загрузки по умолчанию:

- Поддержка двух различных образов загрузки
- Поддержка статусных сообщений, используя JTAG UART
- Выполнение проверку на ошибки образа данных загрузки
- Поддержку образов загрузки с не выровненными словами

Инициализация драйвера

Чтобы уменьшить потребность в памяти, примеру расширенного копировщика загрузки необходима минимальная инициализация драйвера, необходимого, собственно, для поддержки средств копировщика загрузки. По умолчанию, пример инициализирует следующие драйверы:

- Системный таймер
- JTAG UART
- Обработчик прерываний процессора

После завершения инициализации этих драйверов, он перенаправляет на основной код приложения в RAM, который выполняет полную инициализацию системных драйверов. Если вы уверены, что вам не нужны эти компоненты во время загрузки, то пример приложения позволяет вам выборочно запретить инициализацию этих драйверов, уменьшив, тем самым, размер кода.

Вывод в JTAG UART

Копировщик загрузки в этом примере выводит информацию во время процесса загрузки в периферию JTAG UART. Вывод прекрасно подходит для отладки копировщика загрузки, а также для мониторинга статуса загрузки вашей системы. По умолчанию, пример выводит базовую информацию, такую как сообщение о запуске системы, адреса во флеш памяти, где он нашёл образ загрузки и отображает образ, выбранный для загрузки. Вы запросто можете добавить в код свои сообщения для вывода.

Пример расширенного копировщика загрузки избегает использования библиотечной функции printf() по следующим причинам:

- Библиотека printf() может остановить копировщик загрузки, если хост не будет читать выход из JTAG UART.
- Библиотека printf() потенциально может задействовать больше памяти программ.

Предотвращение остановки по JTAG UART

JTAG UART отличается от обычного UART. Традиционный UART обычно передаёт последовательные данные вслепую, независимо от того, есть или нет внешнее хост устройство. Если хост не читает последовательные данные, то данные утеряны. JTAG UART – это другое, от записывает свои данные для передачи в выходной буфер и ждёт, когда внешний хост прочитает их из него, чтобы его очистить. По умолчанию, драйвер JTAG UART останавливается, когда выходной буфер полон. Драйвер ждёт, пока внешний хост прочитает их из выходного буфера, прежде чем записать в него новые данные для передачи. Такой процесс предотвращает потерю передаваемых данных.

Во время загрузки, однако, возможно, что не будет хоста, подключенного к JTAG UART. В этом случае, передаваемые данные не будут прочитаны из выходного буфера. Когда выходной буфер заполнен, функция printf() останавливает выполнение программы. Такая остановка является проблемой, поскольку копировщик загрузки должен продолжать создавать систему вне зависимости от того, подключен ли внешний хост к JTAG UART.

Чтобы избежать этой проблемы, пример расширенного копировщика загрузки реализует свою собственную процедуру вывода, называемую my_jtag_write(). Эта процедура содержит настраиваемое пользователем средство таймаута, которое позволяет JTAG UART приостанавливать программу на ограниченный период времени. После истечения времени таймаута, программа продолжит исполнение, более не делая вывода в JTAG UART. Использование этой процедуры взамен printf() защитит копировщик загрузки от остановки, когда хост не подключен к JTAG UART.

Уменьшение использования памяти для вывода

Пример расширенного копировщика позволяет вам полностью запретить вывод в JTAG UART. Это может значительно уменьшить размер необходимой памяти по копировщик загрузки. Чтобы запретить вывод на JTAG UART в этом пример, выполните следующие шаги:

1. Найдите следующую строку в файле <*project*>**/boot_copier_sw/app/ advanced_boot_copier/ advanced_boot_copier.c**:

#define USING_JTAG_UART 1

2. Замените эту линию следующей:

#define USING_JTAG_UART 0

Образы загрузки

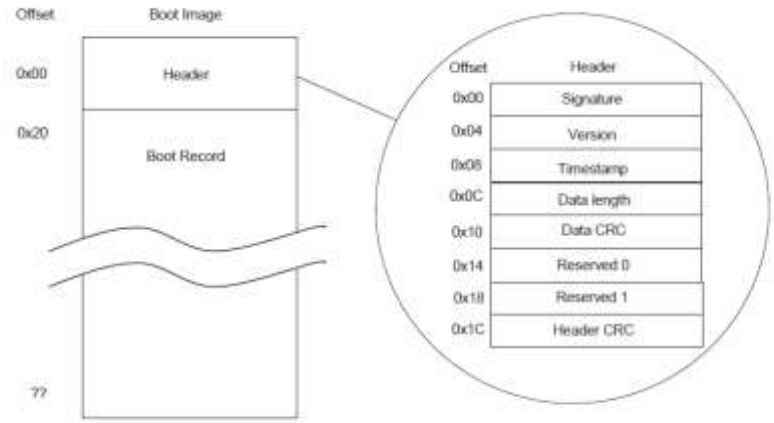
Пример расширенного копировщика загрузки ожидает найти образы загрузки, которые согласованны с определённым форматом и поддерживают до двух образов загрузки, сохранённых во флеш памяти. Он не может допустить, что образ загрузки начинается на границе 32-битных данный во флеш памяти.

Формат образа загрузки

Пример расширенного копировщика загрузки ожидает найти образы загрузки, которые согласованны с определённым форматом. Скрипт **make_flash_image_script.sh** создаёт совместимый с существующим форматом образ загрузки. Скрипт **make_flash_image_script.sh** запускает утилиту **elf2flash** для создания записи загрузки приложения **.elf** файла, и передачи некоторой заголовочной информации этой записи загрузки.

На рис. 1 показан формат загрузочного образа, созданный с помощью скрипта **make_flash_image_script.sh**.

Figure 1. Example Boot Image Format



Формат заголовка образа загрузки

Каждый образ загрузки содержит заголовок по офсету 0x0. Пример копировщика загрузки использует заголовочную информацию, прикреплённую к каждому образу загрузки для получения информации об образе и для создания различных решений процесса загрузки. Скрипт командного процессора **make_flash_image_script.sh** автоматически добавляет заголовочную информацию любому образу загрузки. В табл. 1 описывается информация заголовка образа загрузки.

Табл. 1 Пример формата заголовка образа загрузки

Поле	**Размер**	**Описание**	**Размещение**
Signature	32 бита	Сигнатура (подпись) используется для локализации заголовка во флеш памяти.	Определено в **make_flash_image_script.sh**. Загрузочная сигнатура по умолчанию 0xa5a5a5a5.
Version	32 бита	Двоичное число идентификатора версии приложения	Определено в **make_flash_image_script.sh**.
Timestamp	32 бита	Время создания заголовка. Используется формат целого числа времени Си в секундах с JAN 01, 1970.	Генерируется в **make_flash_image_script.sh**.
Data length	32 бита	Длина данных приложения, находящегося в загрузке, в байтах.	Генерируется в **make_flash_image_script.sh**.
Data CRC	32 бита	Значение CRC32 для целых данных приложения.	Генерируется в **make_flash_image_script.sh**.
Unused 0	32 бита	Неустановленное назначение.	Определено в **make_flash_image_script.sh**.
Unused 1	32 бита	Неустановленное назначение.	Определено в **make_flash_image_script.sh**.
Header CRC	32 бита	Значение CRC32 для данных заголовка.	Генерируется в **make_flash_image_script.sh**.

Формат записи загрузки

Запись загрузки следует сразу за заголовком образа загрузки. Запись загрузки – это представление приложения, которое загружается копировщиком загрузки. Запись загрузки содержит индивидуальную запись для каждой секции кода приложения. Секция кода – это непрерывная часть кода, которая компилируется в уникальный регион памяти. Копировщик загрузки читает запись загрузки для определения адреса назначения каждой секции кода программы приложения, и выполнения соответствующей операции копирования.

Запись загрузки необходима потому, что не все секции кода программы приложения могут быть скомпилированы в одном регионе конфигурации памяти. Зачастую секции кода приложения разбросаны по всему адресному пространству. Для загрузки приложения, во флеш памяти должны находиться целое приложение и информация о том, откуда его части были скопированы в память. Однако, флеш память очень мала, чтобы содержать копию всей памяти. Запись загрузки формирует все секции кода приложения в один непрерывный блок флеш памяти.

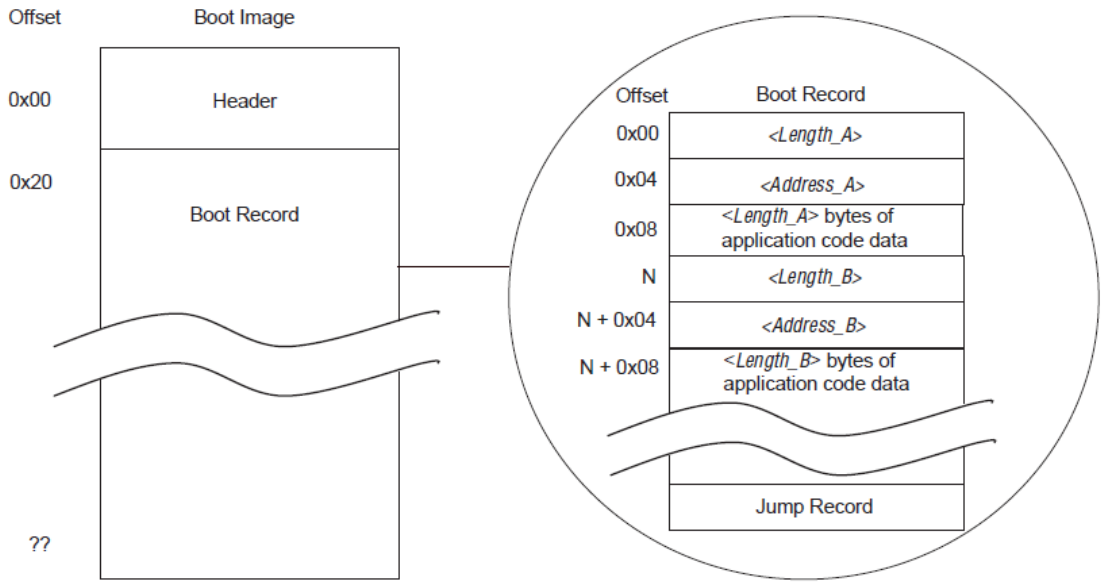
Запись загрузки содержит все секции кода программы приложения в виде непрерывного блока данных, независимо от того, куда в RAM были скомпилированы эти секции кода. Запись загрузки – это последовательность индивидуальных записей, каждая из которых содержит данные для секции кода, упреждённые адресом назначения и их длинной. Во время загрузки, копировщик загрузки читает адрес назначения (<*destination address*>) и длину (<*numbytes*>) из записи загрузки, затем копирует следующие (<*numbytes*>) байты из записи загрузки по <*destination address*>.

В конце индивидуальной записи в записи загрузки находится специальная запись перехода. Чтение этой записи информирует копировщик загрузки о том, что он завершил копирования кода приложения, и что ему сейчас нужно перейти на 32-битный адрес, сохранённый в следующих четырёх байтах. Этот адрес является точкой входа приложения. Запись перехода всегда представлена как 0x00000000.

Третий тип индивидуальной записи – это остановка записи. Остановка записи командует копировщику загрузки остановить собственное исполнение. Остановка записи представлена как 0xFFFFFFFF. Стёртые значения флеш памяти имеют 0xFF в каждом байте. Поэтому, если копировщик загрузки попадает на стёртые места во флеш памяти, он интерпретирует их, как остановку записи и остановку исполнения.

На рис. 2 показана схема памяти в примере записи загрузки.

Figure 2. Example Boot Record Memory Map



Выбор образа загрузки

Пример расширенного копировщика загрузки поддерживает до двух образов загрузки, сохранённых во флеш памяти. Копировщик загрузки проверяет два размещения во флеш памяти, просматривает правильность образа загрузки по каждому размещению, затем выбирает один образ для копирования в RAM и исполняет. Два размещения запланированы как размещение под номером 1 и 2. Для выбора образа загрузки, копировщик использует следующие критерии, в последовательности их применения:

- Правильность образа
  - Если один правильный образ найден, копировщик загрузки использует его.
  - Если правильный образ не найден, копировщик загрузки ждёт пять секунд, затем переходит обратно на адрес сброса Nios II.
- Номер ревизии
  - Если оба образа загрузки правильны, копировщик загрузки просматривает номер версии каждого образа.
  - Копировщик загрузки выбирает образ с наивысшим номером версии.
- Временная метка
  - Если оба образа загрузки имеют один и тот же номер версии, копировщик загрузки просматривает временную метку каждого образа.
  - Копировщик загрузки выбирает самую последнюю временную метку.
- По умолчанию
  - Если оба образа загрузки имеют одну и ту же временную метку, копировщик загрузки выбирает образ, находящийся под номером 2.

Выравнивание по границе слова

В большинстве случаев, ваша программа образа загрузки Nios II запускается с границы 32-битных данных во флеш памяти. Такое размещение позволяет копировщику загрузки копировать данные приложения, используя трансферт 32-битными словами. Однако, пример расширенного копировщика загрузки не допускает такого выравнивания. Если копировщик загрузки найдёт правильный образ загрузки без 32-битного выравнивания по границе слова во флеш памяти, он всё равно точно скопирует приложение в RAM. Копировщик загрузки использует библиотечную функцию memcpy() для выполнения копирования. Функции memcpy() требуется немного памяти, а использование функции memcpy() – это быстрый и устойчивый метод копирования данных, не зависимо от их выравнивания в памяти.

Методы загрузки

Пример расширенного копировщика загрузки поддерживает следующие методы загрузки:

- Прямо из CFI флеш – Загрузка из CFI флеш памяти, копирование расширенного копировщика загрузки из CFI флеш памяти в RAM, запуск копии расширенного копировщика загрузки, которая копирует приложение из CFI флеш памяти и запуск образа приложения из RAM.
- Из CFI флеш, запуск из внутри чиповой памяти – Загрузка из внутри чиповой RAM, копирование приложения из CFI флеш памяти и запуск образа приложения из RAM.
- Из EPCS флеш, запуск из внутри чиповой памяти - Загрузка из внутри чиповой RAM, копирование приложения из EPCS флеш памяти и запуск образа приложения из RAM.

Загрузка прямо из CFI флеш

Этот метод использует два копировщика загрузки. Первый копировщик загрузки запускается из CFI флеш, а второй запускается из RAM. Адрес сброса Nios II устанавливается на адрес в CFI флеш памяти. Копировщики загрузки CFI – расширенный и по умолчанию – все программируются по этому адресу во флеш. Копировщик загрузки CFI по умолчанию начинает исполняться, когда сбрасывается процессор Nios II. Копировщик загрузки CFI по умолчанию загружает второй копировщик загрузки из CFI флеш в RAM и переводит на точку входа второго копировщика загрузки. Расширенный копировщик загрузки копирует приложение из CFI флеш в RAM и переводит на точку входа приложения.

Контролируйте, чтобы кодовое пространство расширенного копировщика загрузки ни перекрывало пространство кода приложения. В этом примере, мы предотвращаем перекрывание, деля SDRAM на две части – верхний и нижний регион. Расширенный копировщик загрузки загружается в верхний регион SDRAM, а приложение загружается в нижний регион SDRAM.

Загрузка из CFI флеш, запуск из внутри чиповой памяти

В этом методе, адрес сброса Nios II устанавливается на базовый адрес загрузочной ROM, реализованной как внутри чиповая память FPGA. Исполняемый код копировщика загрузки загружается в загрузочную ROM при конфигурировании FPGA, после завершения разработки аппаратного проекта в программе Quartus II. Копировщик загрузки начинает исполняться, когда сбрасывается процессор Nios II. Он копирует код приложения из CFI флеш памяти в RAM и переводит на точку входа приложения.

Загрузка из EPCS флеш, запуск из внутри чиповой памяти

Этот метод очень похож на предыдущий. Отличие в том, что образ загрузки хранится в EPCS флеш, а не в CFI флеш. В этом методе, адрес сброса Nios II также устанавливается на базовый адрес загрузочной ROM, реализованной как внутри чиповая память FPGA. Исполняемый код копировщика загрузки загружается в загрузочную ROM при конфигурировании FPGA, после завершения разработки аппаратного проекта в программе Quartus II. Копировщик загрузки начинает исполняться, когда сбрасывается процессор Nios II. Он копирует код приложения из EPCS флеш памяти в RAM и переводит на точку входа приложения.

Установка метода загрузки

Пример расширенного копировщика загрузки поддерживает все три метода, описанный выше. Следующая строка в файле *<project>***/boot_copier_sw/app/advanced_boot_copier/advanced_boot_copier.c** контролирует реализуемый метод:

#define BOOT_METHOD <*boot method*>

Следующие опции доступны в поле <*boot method*>:

- BOOT_FROM_CFI_FLASH
- BOOT_CFI_FROM_ONCHIP_ROM
- BOOT_EPCS_FROM_ONCHIP_ROM

Предотвращение перекрытия данных во флеш

Когда вы устанавливаете вашей системе загрузку из флеш памяти, вы должны принимать к рассмотрению другие данные, хранящиеся во флеш. Платы разработки Altera специально разработаны для поддержки хранения образов конфигурации FPGA и образов загрузки программы вместе в одном чипе флеш памяти - CFI или EPCS. Когда вы храните несколько образов во флеш памяти, вы должны следить за тем, чтобы образы не перекрывались.

Перекрытие данных в CFI флеш

Используйте утилиту **nios2-elf-size** для подсчёта размера каждого образа флеш, затем выберите офсеты во флеш памяти для каждого образа, основываясь на их размерах (или предполагаемых размерах), этим вы предотвратите перекрытие.

Перекрытие данных в EPCS флеш

В EPCS флеш, образ конфигурации FPGA всегда должен начинаться с офсета 0x0. Чтобы избежать программирования любого загрузочного образа в начале образа конфигурации FPGA, вы должны определить офсета конца образа конфигурации FPGA. Конвертируйте ваш **.sof** файл образа конфигурации в **.flash** образ, используя утилиту **sof2flash**, затем запустите **nios2-elf-size** для этого флеш образа. Результатом будет офсет конца образа конфигурации FPGA в EPCS флеш. Проследите за тем, чтобы любой образ загрузки программы, который вы программируете в EPCS флеш, начинался с офсета выше конца образа конфигурации FPGA.

Размер кода копировщика загрузки

Пример расширенного копировщика загрузки, не модифицированный, компилируется в исполняемый файл размером примерно 6500 байт. Если вы исключаете функции JTAG UART и системного таймера, исполняемый файл уменьшается примерно на 2000 байт.

Для сравнения, размер кода копировщика загрузки по умолчанию, описанного в секции "Копировщик загрузки Nios II по умолчанию" на стр. 2, - примерно 200 байт, когда компилируется для загрузки из CFI флеш, и примерно 500 байт, когда компилируется для загрузки из EPCS флеш.

Если вам требуется разработать копировщик загрузки размером менее 2000 байт, обратитесь к секции "Пример малого копировщика загрузки" на стр. 18. Малый копировщик загрузки написан на языке Nios II ассемблер и содержит очень мало функций. Когда он исполняется, он просто копирует запись загрузки из CFI в RAM, а затем переводит на скопированное приложение. Размер скомпилированного кода примерно 200 байт.