УДК

BUILDROOT ДЛЯ СИСТЕМ НА КРИСТАЛЛЕ СЕМЕЙСТВА ALLWINNER F1CX00S

Ценцевицкий Д.А., магистрант гр.255741

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Шемаров А.И. – канд. техн. наук

Аннотация. Программирование встраиваемых систем является не простой задачей само по себе: ограниченое количество постоянной и оперативной памяти, отсутствие многих аппаратных блоков, малопроизводительные и низкочастотные архитектуры процессорных ядер. При всем при этом написание и отладка программного кода по так называемой baremetal технике доставляет множество неудобст в том числе связанных с отладкой. Для решения данной проблемы чаще всего прибегают к использованию операционных систем, таких как FreeRTOS или mbedOS, однако и они не всегда удобны и функциональны. Если же разработчику необходимо максимально абстрагироваться от аппаратной составляющей, то наилучшим решением будет использование Linux.

Ключевые слова. Linux, Allwinner, UBoot, buildroot, embedded, встраеваемые системы.

Прежде всего необходимо рассмотреть, что делает каждая часть Buildroot. На высоком уровне, рабочий процесс, который Buildroot автоматизирует, представлен на рисунке 1.

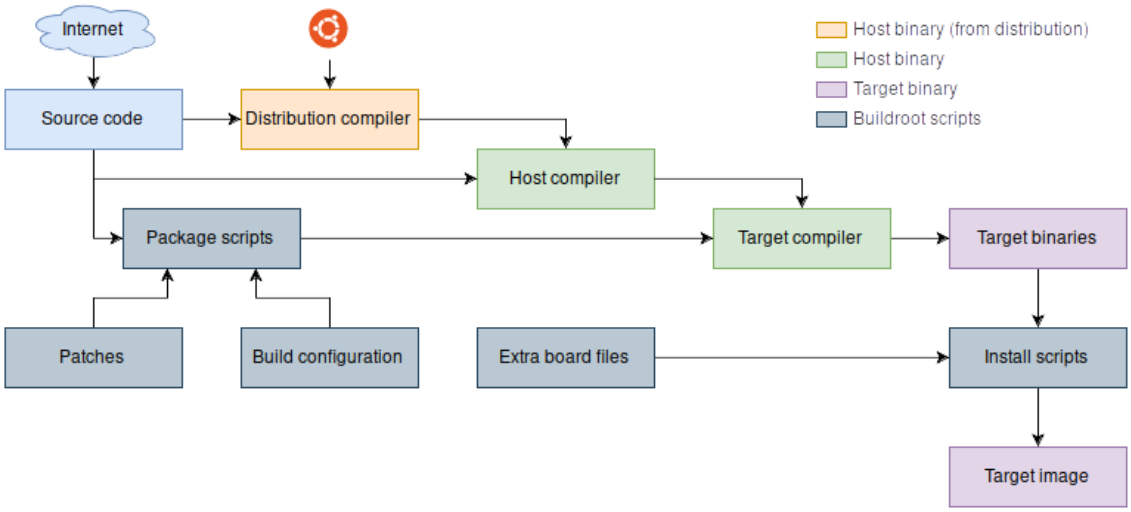

Рисунок 1 – Рабочий процесс автоматизируемый при помощи Buildroot

Типичный процесс сборки образа системы состоиз из следующих шагов:

1. Buildroot создает набор инструментов, который представляет собой кросс-компиляторы и другие инструменты, необходимые для компиляции целевой системы (зеленые прямоугольники).
2. Исходный код (синие прямоугольники) для каждой части программного обеспечения загружается из интернета.
3. Используя скрипты Buildroot (серые прямоугольники), исходный код распаковывается, исправляется, настраивается, компилируется и устанавливается в целевой выходной каталог, который формирует корневую файловую систему (rootfs) для целевого объекта (фиолетовые прямоугольники).
4. Дополнительные файлы, такие как файлы конфигурации на устройстве, также копируются в целевой выходной каталог.
5. Наконец, скрипты собирают окончательный образ прошивки из этого корневого файла.

Есть некоторые исключения; иногда кросс-компилятор загружается без необходимости компиляции. Иногда производитель поставляет целый Board Support Package (BSP) , где вся компиляция уже выполнена за нас. Но, в конце концов, это просто сокращает эту блок-схему; все шаги все равно должны быть каким-то образом выполнены.

Наиболее важные каталоги в верхней части дерева Buildroot представлены в таблице 1. Во время своей работы Buildroot генерирует множество других директорий, однако представленные в таблице являются самыми частоиспользуемыми и полезными необходимыми в работе.

Таблица 1 – Результаты тестирования моделей на MS COCO 2017 на устройстве с Intel i7-8550U

Директория	Описание
board/	Файлы и скрипты для поддержки каждой целевой платы
configs/	Конфигурации сборки, такие как suniv-f1c100s_defconfig
package/	Определения пакетов
output/host/	Инструменты сборки, работающие на хотстовой машине
output/target/	Целевой выходной каталог, в котором размещаются целевые двоичные файлы
output/images/	Сюда выводятся образы файловой системы и окончательный образ прошивки

Файл конфигурации suniv-f1c100s_defconfig является одним из основных конфигурационных файлов при сборке системы, он содержит информацию о конкретных версиях ядра и uboot-a, пути к патчам, настройки целевой файловой системы, а так же ссылку на конфигурационный файл uboot.

1.Портирование U-Boot и Linux

Первым шагом при портировании Linux на любую плату является установка и запуск U-Boot.

Из-за ошибки около 70% усилий по программному обеспечению было потрачено на то, чтобы U-Boot общался с флэш-памятью. У U-Boot есть драйвер для работы SPI периферии, но он не работал. Симптом заключался в том, что периферийное устройство SPI могло обнаружить флэш-чип, но всегда считывало с нее мусор. Оказалось, что U-Boot сбрасывает периферийное устройство SPI, когда завершает работу с ним. По-видимому, это не вызвало никаких проблем с другими чипам Allwinner. Однако на F1C100 сброс периферийного устройства, что неудивительно, стирает его регистры конфигурации, особенно те, которые используются для настройки скорости. На рисунке 2 представлен патч устраняющий этот баг.

```
 drivers/spi/spi-sunxi.c | 10 +++++++++-
 1 file changed, 9 insertions(+), 1 deletion(-)

diff --git a/drivers/spi/spi-sunxi.c b/drivers/spi/spi-sunxi.c
index dbfeac77eec..2d02289d04d 100644
--- a/drivers/spi/spi-sunxi.c
+++ b/drivers/spi/spi-sunxi.c
@@ -35,6 +35,10 @@

 DECLARE_GLOBAL_DATA_PTR;

+/* Forward declarations of some reused functions */
+static int sun4i_spi_set_speed(struct udevice *dev, uint speed);
+static int sun4i_spi_set_mode(struct udevice *dev, uint mode);
+
 /* sun4i spi registers */
 #define SUN4I_RXDATA_REG        0x00
 #define SUN4I_TXDATA_REG        0x04
@@ -300,7 +304,8 @@ static inline int sun4i_spi_set_clock(struct udevice *dev, bool enable)

 static int sun4i_spi_claim_bus(struct udevice *dev)
 {
-    struct sun4i_spi_priv *priv = dev_get_priv(dev->parent);
+    struct udevice *bus = dev->parent;
+    struct sun4i_spi_priv *priv = dev_get_priv(bus);
     int ret;

     ret = sun4i_spi_set_clock(dev->parent, true);
@@ -317,6 +322,9 @@ static int sun4i_spi_claim_bus(struct udevice *dev)
     setbits_le32(SPI_REG(priv, SPI_TCR), SPI_BIT(priv, SPI_TCR_CS_MANUAL) |
              SPI_BIT(priv, SPI_TCR_CS_ACTIVE_LOW));

+    sun4i_spi_set_speed(bus, priv->freq);
+    sun4i_spi_set_mode(bus, priv->mode);
+
     return 0;
 }
```

Рисунок 2 – Патч для работы с SPI flash

После этого патча все заработало. Linux 5.2 загрузился практически без проблем.

2.Создание браза флеш памяти

На отладочной плате установлено 8 МБ флэш-памяти. Она должна вмещать загрузчик, ядро, корневую файловую систему и небольшой постоянный раздел. Я решил использовать UBI для размещения всего, кроме загрузчика, который должен располагаться в самом начале флэш-памяти. Используя genimage, я определяю физические разделы как показано на рисунке 3.

```
image flash.bin {
    flash {}
    flashtype = w25q64
    partition uboot {
        image = "u-boot-sunxi-with-spl.bin"
        size = 256K
    }
    partition rootubi {
        image = root.ubi
        size = 0
    }
}
```

Рисунок 3 – Физические разделы флеш памяти

Затем root.ubi создается из образов разделов как показано на рисунке 4.

```
image root.ubi {
    ubi {}
    partition kernel {
        image = "zImage"
        read-only = true
    }
    partition dtb {
        image = f1c100s.dto
        read-only = true
    }
    partition root {
        image = "rootfs.squashfs"
        read-only = true
    }
    partition flashdrive {
        image = flashdrive.img
        read-only = true
    }
    partition persist {
        image = persist.ubifs
    }
}
```

Рисунок 4 – Разделы root.ubi

Во время сборки genimage использует это определение для автоматической компиляции двоичного файла, готового к загрузке на чип флеш памяти.

3.Прошивка флеш памяти

Прошивка флеш памяти достаточно проста. При подключении процессора по аппаратному USB к компьютеру происходит следующее: процессор не может найти ничего для загрузки (флэш-память пуста), поэтому он запускает встроенный режим FEL, который позволяет использовать инструмент под названием sunxi-fel для отправки сгенерированного образа по USB прямо во флеш память, выполнив команду «`output/host/bin/sunxi-fel -p spiflash-write 0 output/images/flash.bin`». Прошивка нескольких мегабайт занимает около двух минут, потому что NOR flash имеет низкую скорость записи - около 100 КБ /с.

Таким образом после загрузки (около 6 сек) мы получаем полноценный Linux готовый к разработке новых программных модулей в знакомом окружении с возможностью удаленной отладки и прочих удобств унифицированной ОС в виде большого количества готовых утилит и драйверов.

Список использованных источников:

1. George Hilliard's blog about embedded systems and software engineering. [Электронный ресурс]. – Режим доступа: https://www.thirtythreeforty.net/. – Дата доступа: 12.04.2023.

2. Buildroot. – Режим доступа: https://buildroot.org/. – Дата доступа: 12.04.2023.

3. Linux kernel source tree. [Электронный ресурс]. – Режим доступа: https://github.com/torvalds/linux. – Дата доступа: 12.04.2023.

4. Das U-Boot Source Tree. [Электронный ресурс]. – Режим доступа: https://github.com/u-boot/u-boot. – Дата доступа: 12.04.2023.

UDC

SOFTWARE MODULE FOR PATTERN RECOGNITION FOR MICROCOMPUTERS BASED ON NEURAL NETWORKS OF YOLOV5 FAMILY

Tsentsevitski D.A.[1]

Belarusian State University of Informatics and Radioelectronics[1]*, Minsk, Republic of Belarus*

Shemarov A.I. – Candidate of Technical Sciences

Annotation. Programming embedded systems is not an easy task in itself: a limited amount of ROM and RAM, the absence of many hardware units, low-performance and low-frequency architectures of processor cores. With all this, writing and debugging program code using the so-called baremetal technique brings a lot of inconvenience, including those associated with debugging. To solve this problem, they most often resort to using operating systems such as FreeRTOS or mbedOS, however, they are not always convenient and functional. If the developer needs to abstract as much as possible from the hardware component, then the best solution would be to use Linux.

Keywords. Linux, Allwinner, UBoot, buildroot, embedded..