

Что будет использоваться в лабораторной работе:

- **Аппаратная часть:**
 - Отладочная плата AT91SAM9M10G45-EK;
 - Ноутбук с DVD-приводом и возможностью загрузки с DVD-диска, а также с 2-мя (как минимум) USB-портами;
 - Кабель micro USB, кабель RS232, адаптер питания 5В (входят в комплект разработчика AT91SAM9M10G45-EK);
 - Переходник USB-COM;
 - Flash-карта памяти.
- **Программная часть:**
 - Лабораторная работа будет вестись в ОС Ubuntu 10.04 с Live DVD-диска. Никакое дополнительно программное обеспечение на ноутбук устанавливать не требуется.
- **Приблизительное время работы:** 90 минут.

Введение

Целью данной практической работы является знакомство с встраиваемой операционной системой Linux на примере отладочной платы AT91SAM9M10G45-EK. В процессе работы будут рассмотрены следующие вопросы:

- ✓ Обзор основных компонентов встроенного программного обеспечения Linux-системы;
- ✓ Сборка составных частей ПО:
 - Компиляция предзагрузчика Bootstrap;
 - Компиляция загрузчика U-Boot, редактирование config-файла;
 - Конфигурирование и компиляция ядра Linux;
 - Сборка образа файловой системы.
- ✓ Работа с программой SAM-BA, «прошивка» собранных компонентов в память микроконтроллера;
- ✓ Работа с программой MINICOM и запуск полученной системы. Задание различных параметров загрузчика U-Boot;
- ✓ Создание простой графической программы в среде QT-Creator, ее кросс-компиляция и запуск на отладочном комплекте.



Обозначения в документе

Важные команды, названия папок, путей и т. д. выделены в тексте ***жирным курсивом***. Команды, которые непосредственно можно скопировать из документа и вставить в терминал, выделены в тексте ***жирным красным курсивом***.

Содержание

Что будет использоваться в лабораторной работе	1
Введение	1
Обозначения в документе	2
1 Аппаратная часть практической работы	4
1.1 Отладочная плата AT91SAM9M10G45-EK	4
2 Практическая часть	6
2.1 Загрузка Ubuntu 10.04 с Live DVD	6
2.2 Структура рабочего каталога	8
2.3 Сборка компонентов системы	8
2.3.1 Кросс-инструменты для компиляции	8
2.3.2 Сборка первичного загрузчика BOOTSTRAP	9
2.3.3 Сборка универсального загрузчика UBOOT	11
2.3.4 Сборка ядра LINUX	13
2.3.5 Сборка консольного образа файловой системы	20
2.4 «Прошивка» собранных компонентов в Nandflash-память отладочного комплекта с помощью программы SAM-BA	22
2.4.1 Настройка программы SAM-BA	24
2.4.2 «Прошивка» BOOTSTRAP	24
2.4.3 «Прошивка» UBOOT	25
2.4.4 «Прошивка» LINUX KERNEL	25
2.5 Подготовка Flash-карты памяти для загрузки	25
2.5.1 Запись tar-архива графического образа файловой системы на Flash-карту памяти	25
2.6 Запуск системы, работа в программе MINICOM, настройка параметров U-BOOT	27
2.6.1 Вариант загрузки с внешней Flash-карты памяти	27
3.7 Создание GUI-приложения в среде разработки QT Creator	30
3.7.1 Создание GUI-проекта в среде QT-Creator	30
3.7.2 Кросс-компиляция проекта	33
3.7.3 Перенос проекта и QT-библиотек в файловую систему микроконтроллера (на Flash-карту)	34
3.7.4 Задание переменных окружения системы на микроконтроллере для запуска графического приложения	35
3.7.5 Запуск программы на контроллере	35
Приложение	37

1. Аппаратная часть практической работы.

1.1 Отладочная плата AT91SAM9M10G45-EK:

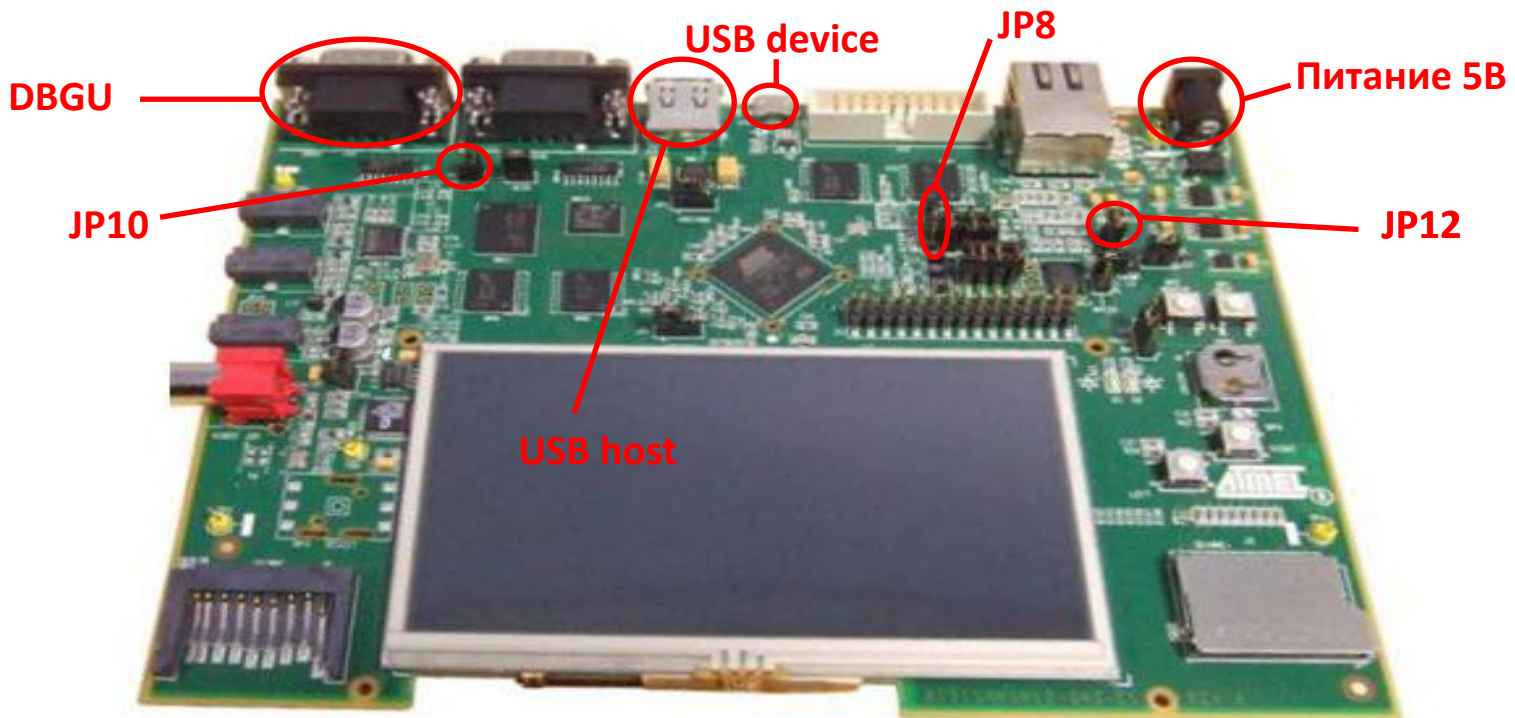


Рис. 1

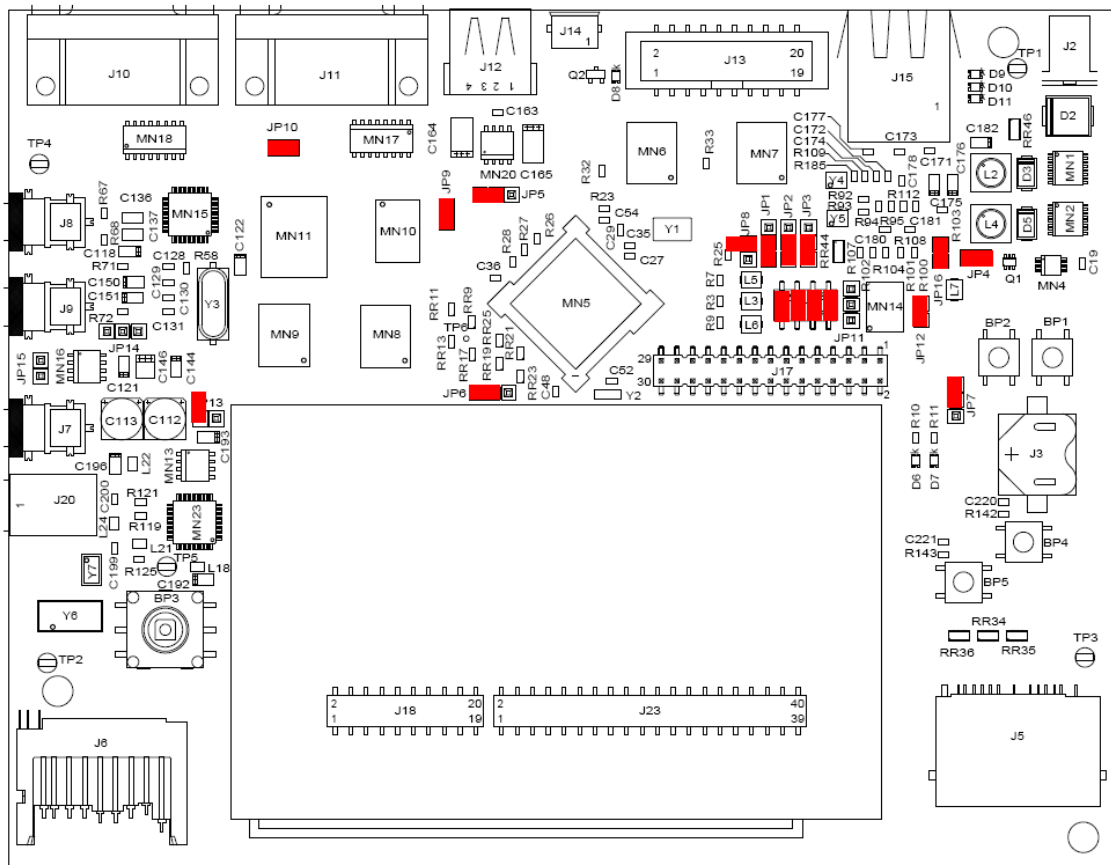


Рис. 2

На рисунке 1 показан внешний вид отладочной платы AT91SAM9M10G45-EK:

JP10 – этот джампер должен быть **закрыт** для доступа к Nandflash-памяти платы;

JP12 – этот джампер должен быть **закрыт** для доступа к Dataflash-памяти платы;

JP8 – BMS (Boot Mode Select – выбор способа загрузки) всегда должен быть **открыт** для загрузки с встроенной ROM-памяти;

DBGU – порт для вывода отладочной информации и работы с платой через терминал;

USB-device – порт микро USB для подключения к ноутбуку при «прошивке» платы через SAM-BA;

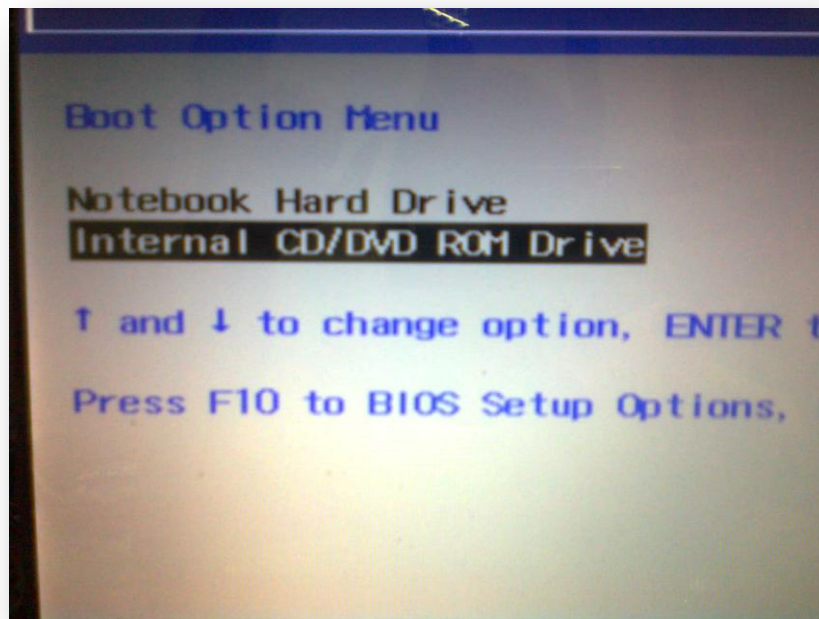
USB-host – порт USB для работы с различными USB-устройствами (в нашей работе с Flash-картой памяти).

На рисунке 2 показаны положения всех джамперов, в которые они должны быть установлены по умолчанию.

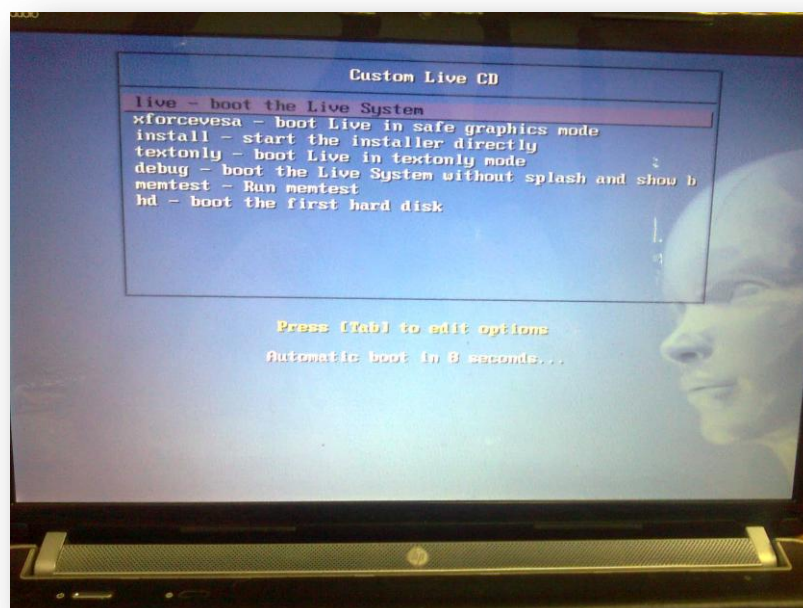
2. Практическая часть

2.1 Загрузка Ubuntu 10.04 с LiveDVD

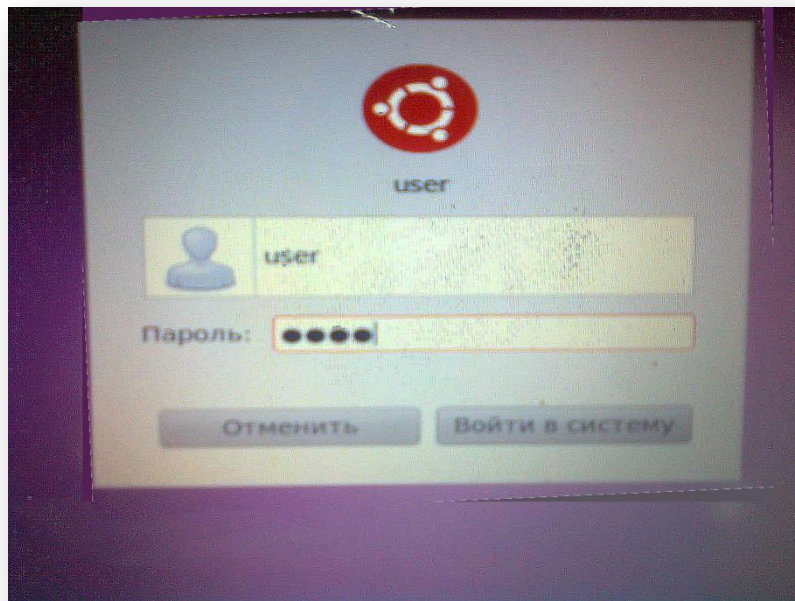
- ✓ Вставьте диск с Ubuntu 10.04 в привод ноутбука и перезагрузитесь, выбрав при этом в BIOS'е вариант загрузки с CD/DVD, если таковой не выбран по умолчанию.



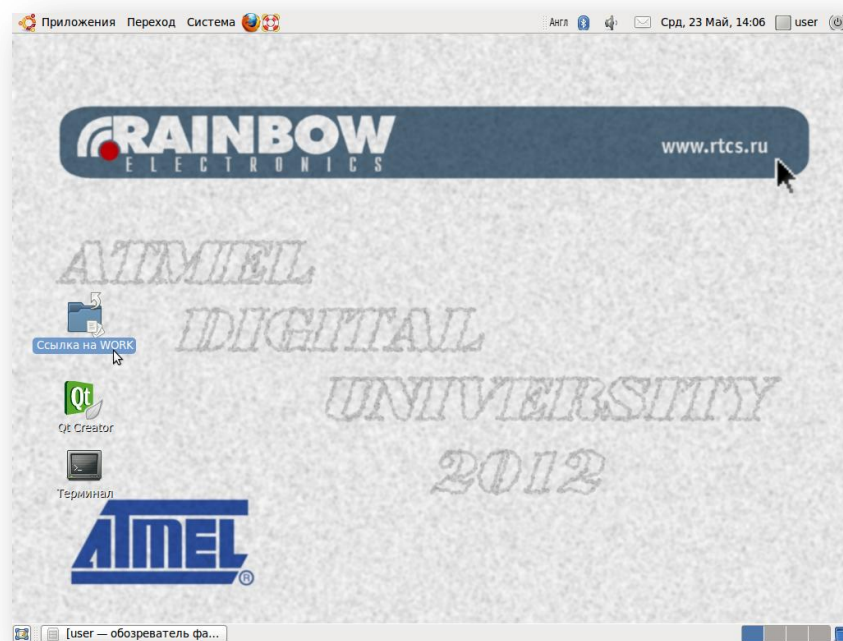
- ✓ Перед Вами появится надпись Boot...Необходимо нажать Enter.
- ✓ Перед Вами появится окно с выбором вариантов загрузки. Вам нужно остановиться на первом (Boot Live CD), нажав клавишу Enter.



- ✓ В окне приглашения входа в систему выберите пользователя user и введите для него пароль user.



- ✓ Перед Вами появится рабочий стол Ubuntu 10.04, где и будет вестись вся дальнейшая работа.



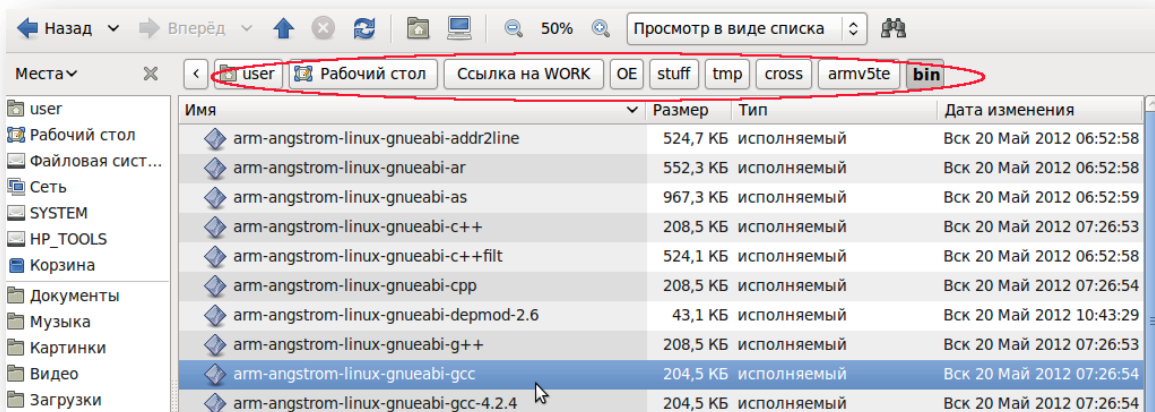
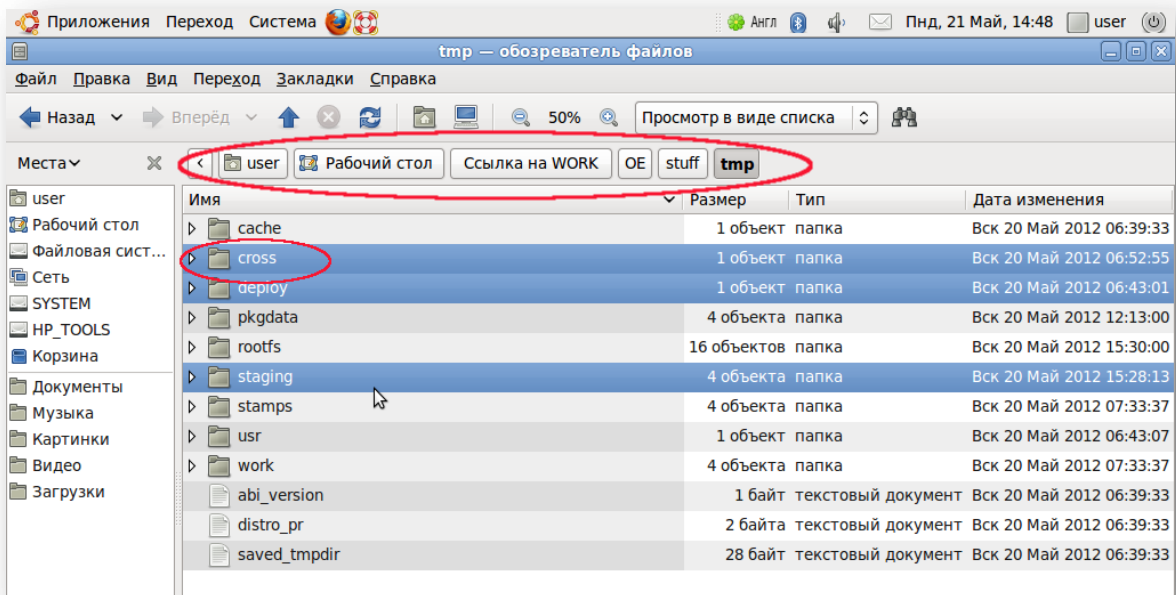
2.2 Структура рабочего каталога

1. На рабочем столе найдите ярлык к папке WORK – каталогу, в котором будет производиться вся дальнейшая работа;
2. Двойным щелчком откройте его и Вы окажетесь в директории WORK, в которой увидите следующие папки: **BOOTSTRAP**, **LINUX_KERNEL**, **OE**, **SAMBA**, **UBOOT** и **arm9**.

2.3 Сборка компонентов системы

2.3.1 Кросс-инструменты для компиляции

Для сборки всех компонентов встроенного программного обеспечения для ARM9-платформы мы будем пользоваться кросс-инструментами (в частности, кросс-компилятором), созданными с помощью системы Openembedded. Более подробно о том, что такое Openembedded, будет рассказано в теоретической части. Исполняемые файлы кросс-инструментов находятся в папке `/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin` и все начинаются с префикса **arm-angstrom-linux-gnueabi-**

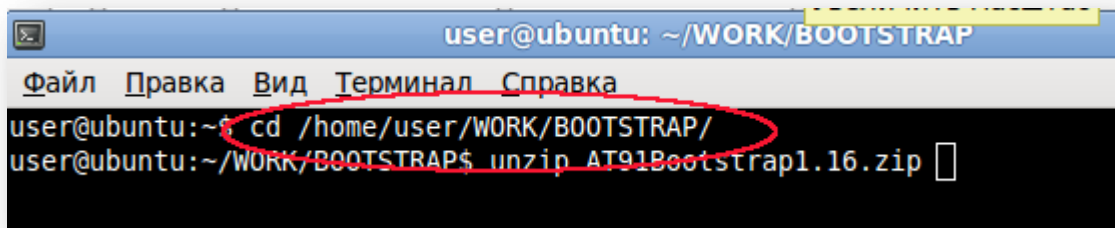


2.3.2 Сборка первичного загрузчика BOOTSTRAP

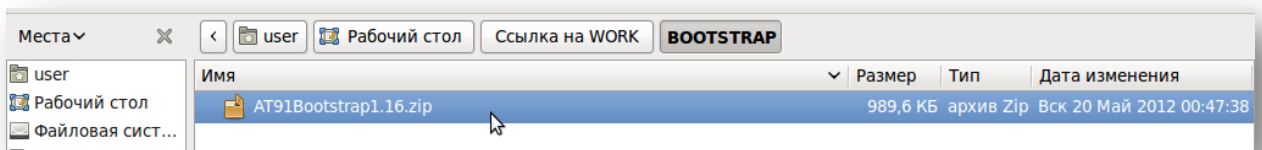
1. На рабочем столе найдите кнопку запуска программы «Терминал» и двойным щелчком откройте ее:



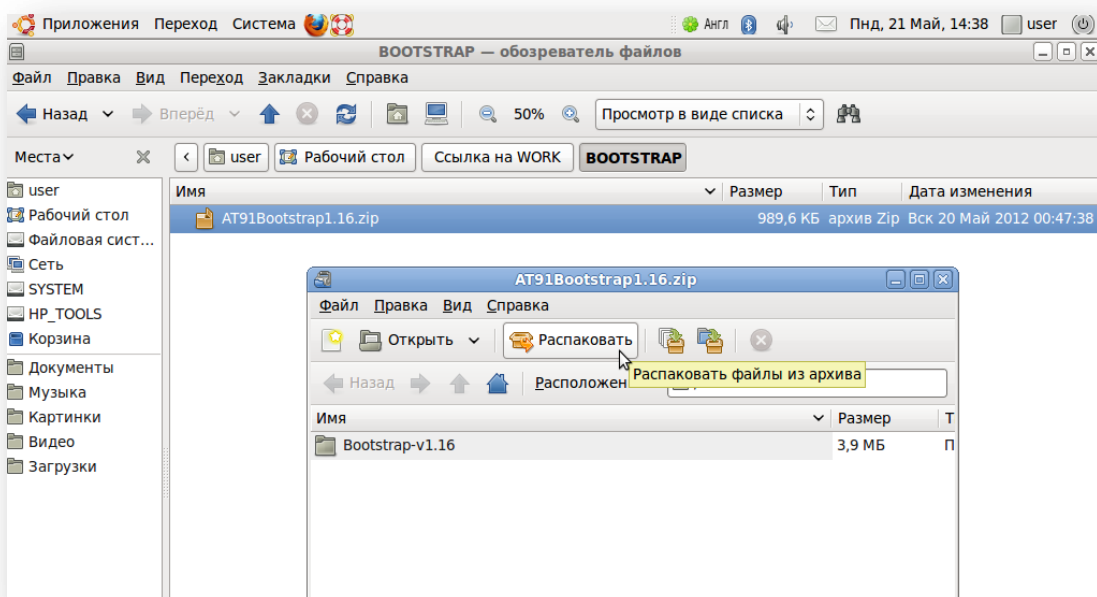
2. Зайдите в папку BOOTSTRAP, набрав в командной строке терминала строку ***cd /home/user/WORK/BOOTSTRAP/***



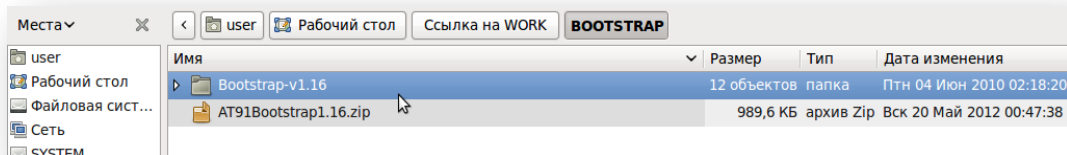
3. В папке BOOTSTRAP находится архив с исходными кодами, который нужно разархивировать.



4. Делаем это, дважды щелкнув по файлу архива:



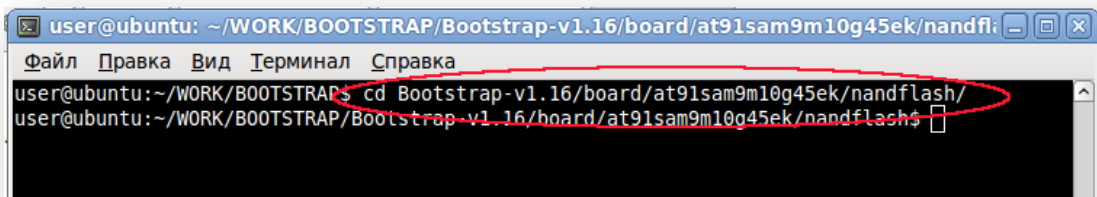
5. Итак, после распаковки в каталоге BOOTSTRAP появляется папка **Bootstrap-v1.16**:



6. Заходим в папку, набрав в терминале команду

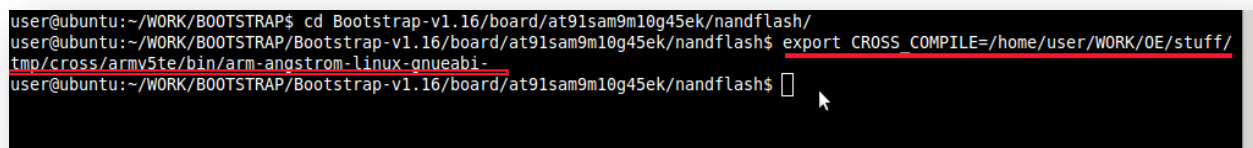
```
cd Bootstrap-v1.16/board/at91sam9m10g45ek/nandflash
```

Мы заходим именно в эту папку, так как в данной работе будем загружать все компоненты системы из установленной на плату Nandflash-памяти. Поэтому необходимо произвести сборку Bootstrap именно для этого варианта.

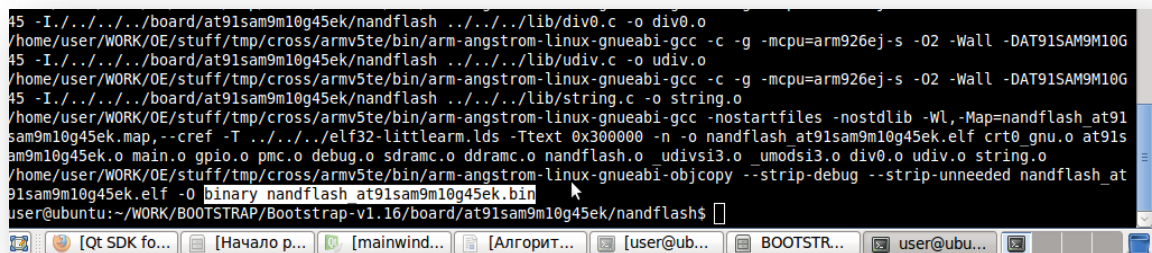


7. Теперь нужно задать переменную окружения **CROSS_COMPILE** для указания пути к кросс-компилятору. Делаем это командой

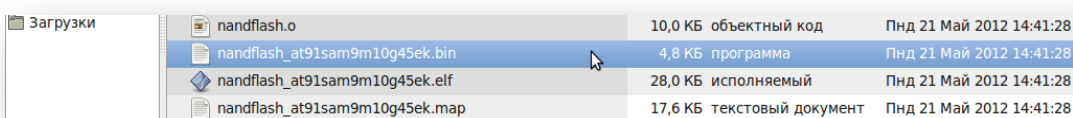
```
export CROSS_COMPILE=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-
```



8. Теперь можно запустить непосредственно сборку исполняемого файла предзагрузчика командой **make** (набираем в терминале make). Начинается процесс кросс-компиляции, который завершается следующим образом:



9. В результате в папке **Bootstrap-v1.16/board/at91sam9m10g45ek/nandflash** появляется файл **nandflash_at91sam9m10g45ek.bin**, который позже мы загрузим в нашу плату:

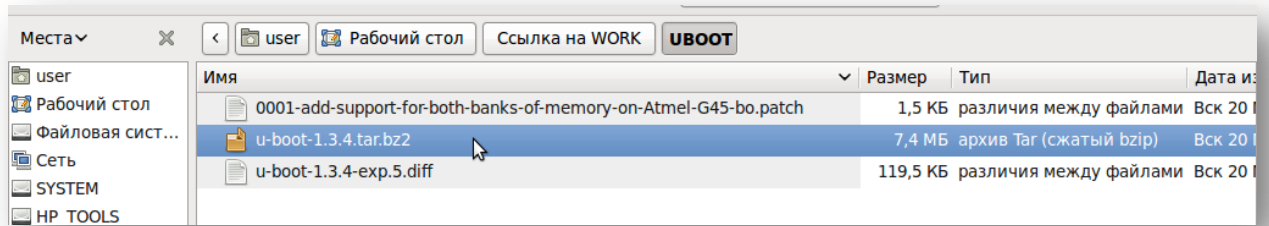


2.3.3 Сборка универсального загрузчика U-BOOT

1. Зайдите в папку UBOOT, набрав в терминале команду

`cd /home/user/WORK/UBOOT`

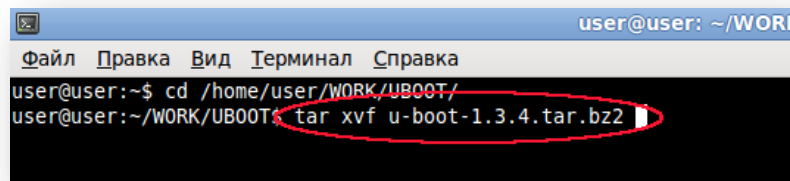
Содержимое этого каталога выглядит следующим образом:



где **`u-boot-1.3.4.tar.bz2`** – архив с «исходниками» загрузчика, **`u-boot-1.3.4-exp.5.diff`** – патч, добавляющий в U-Boot поддержку нашей отладочной платы, а также некоторые другие особенности, **`0001-add-support-for-both-banks-of-memory-on-Atmel-G45-b0.patch`** – патч, добавляющий в U-Boot возможность использования двух банков оперативной памяти.

2. Распакуем архив командой

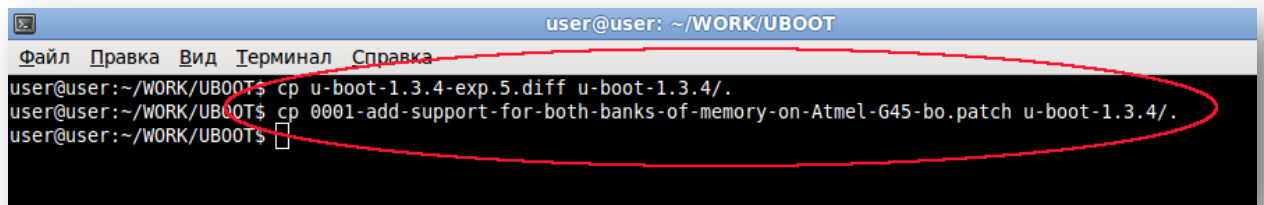
`tar xvf u-boot-1.3.4.tar.bz2`



3. Скопируем оба патча в полученную после разархивирования папку командами

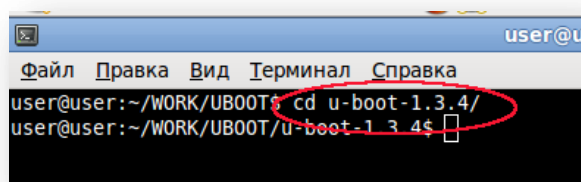
`cp u-boot-1.3.4-exp.5.diff u-boot-1.3.4/.`

`cp 0001-add-support-for-both-banks-of-memory-on-Atmel-G45-bo.patch u-boot-1.3.4/.`

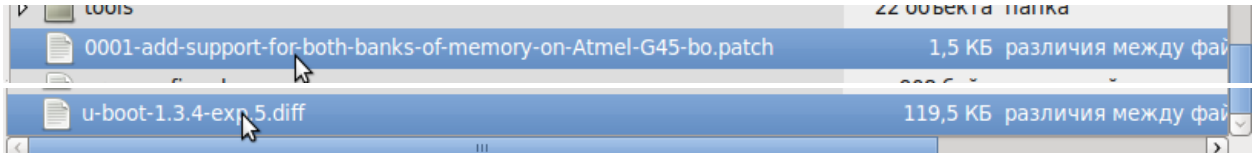


4. Перейдем в папку u-boot-1.3.4

`cd u-boot-1.3.4`

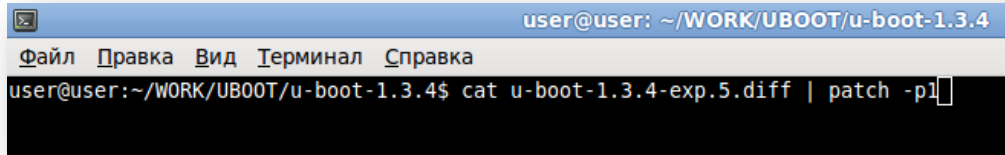


5. В результате в папке появились скопированные нами патчи:



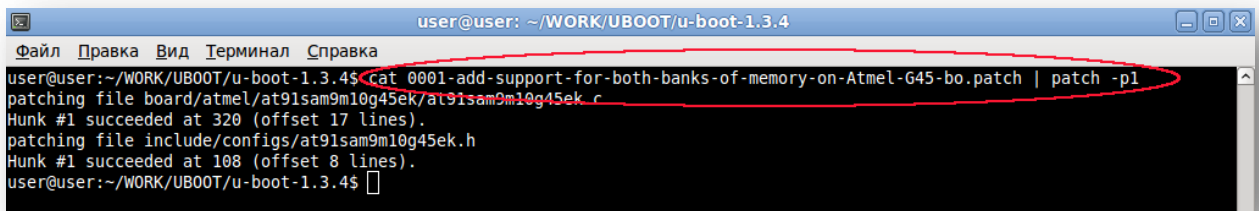
6. Применим первый патч, набрав команду

cat u-boot-1.3.4-exp.5.diff | patch -p1

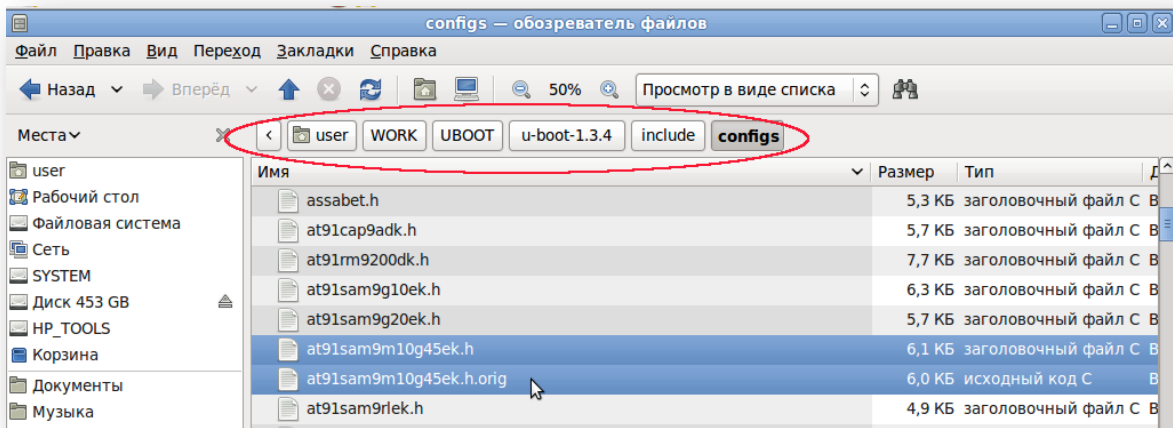


7. Применим второй патч, набрав команду

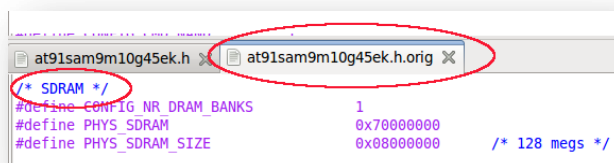
cat 0001-add-support-for-both-banks-of-memory-on-Atmel-G45-bo.patch | patch -p1



8. Второй патч нужен для добавления возможности использования двух банков памяти на отладочной плате. Он меняет конфигурационный файл U-Boot. В результате в каталоге ***include/configs/*** теперь лежат два файла ***at91sam9m10g45ek.h*** (измененная конфигурация U-Boot) и ***at91sam9m10g45ek.orig*** (начальная конфигурация U-Boot):



9. Отличия состоят в следующем:



```
at91sam9m10g45ek.h
/* SDRAM */
#define CONFIG_NR_DRAM_BANKS 2
#define PHYS_SDRAM 0x00000000
#define PHYS_SDRAM_BANK_1 0x20000000
#define PHYS_SDRAM_SIZE 0x08000000 /* 128 megs */
```

10. Далее, как и в случае с Bootstrap, необходимо задать переменную окружения **CROSS_COMPILE** командой

export CROSS_COMPILE=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-

11. Теперь сконфигурируем проект U-boot с указанием того, что в будущем он будет загружаться из Nandflash. В терминале наберем команду

make at91sam9m10g45ek_nandflash_config

```
user@user: ~/WORK/UBOOT/u-boot-1.3.4
Файл Правка Вид Терминал Справка
user@user:~/WORK/UBOOT/u-boot-1.3.4$ make at91sam9m10g45ek_nandflash_config
... with environment variable in NAND FLASH
Configuring for at91sam9m10g45ek board...
user@user:~/WORK/UBOOT/u-boot-1.3.4$
```

12. Теперь все готово для сборки и можно запустить команду **make**

13. Сборка завершается появлением в каталоге **u-boot-1.3.4** исполняемого файла u-boot, который позже мы «прошьем» в Nandflash-память:

```
10g45ek.a --end-group -L /home/user/WORK/OE/stuff/tmp/cross/armv5te/lib/gcc/arm-angstrom-linux-gnueabi/4.2.4 -lgcc \
-Map u-boot.map -o u-boot
/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-objcopy --gap-fill=0xff -O srec u-boot u-boot.srec
/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
user@user:~/WORK/UBOOT/u-boot-1.3.4$
```

System.map	22,6 КБ	текстовый документ
u-boot	596,9 КБ	исполняемый
u-boot.bin	158,6 КБ	программа
u-boot.map	147,0 КБ	текстовый документ
u-boot.srec	476,0 КБ	текстовый документ

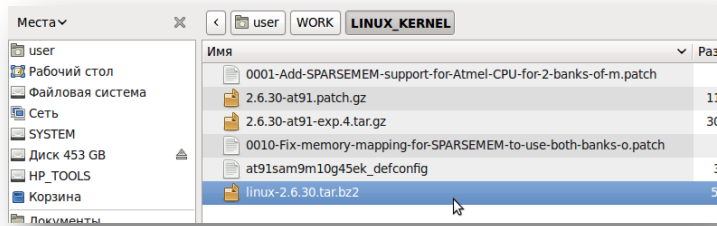
2.3.4 Сборка ядра LINUX

1. Зайдите в папку LINUX, набрав в терминале команду

cd /home/user/WORK/LINUX_KERNEL

```
user@us
Файл Правка Вид Терминал Справка
user@user:~$ cd /home/user/WORK/LINUX_KERNEL/
user@user:~/WORK/LINUX_KERNEL$
```

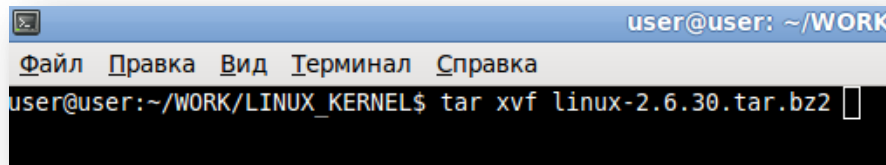
2. Содержимое папки выглядит следующим образом:



где **linux-2.6.30.tar.bz2** – архив с исходными кодами ядра linux для всех архитектур, **2.6.30-at91.patch.gz** – архив патча от Atmel, **2.6.30-at91-exp4.tar.gz** – архив с набором патчей от Atmel, **at91sam9m10g45ek_defconfig** – стандартный конфигурационный файл для нашей отладочной платы, **0001-Add-SPARSEMEM-support-for-Atmel-CPU-for-2-banks-of-m.patch** и **0010-fix-memory-mapping-for-SPARSEMEM-to-use-both-banks-o.patch** – два патча для добавления в ядро возможности работы с обоими банками SDRAM, установленной на плату.

3. Распакуйте архив ядра командой

tar xvf linux-2.6.30.tar.bz2



4. Скопируйте все файлы в полученную папку linux-2.6.30 командами

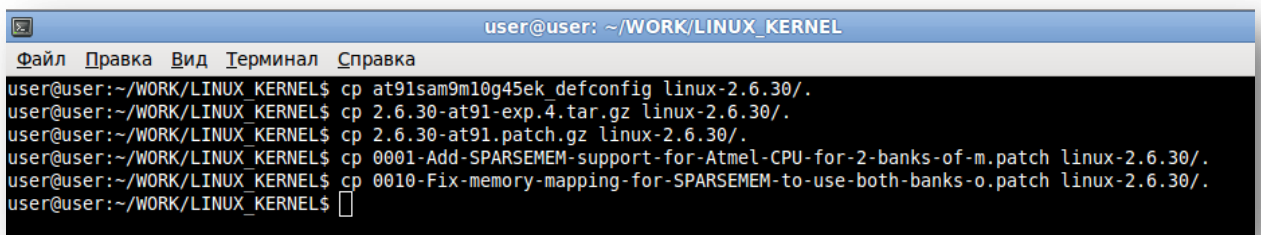
cp at91sam9m10g45ek_defconfig linux-2.6.30/.

cp 2.6.30-at91.patch.gz linux-2.6.30/.

cp 2.6.30-at91-exp.4.tar.gz linux-2.6.30/.

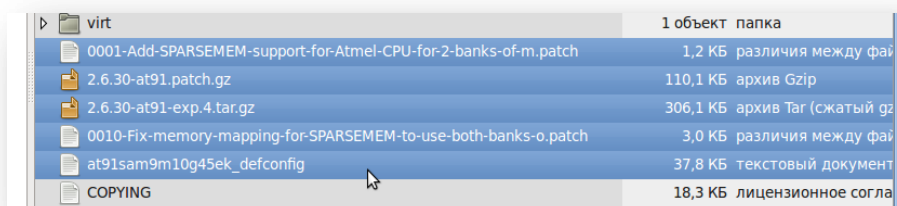
cp 0001-Add-SPARSEMEM-support-for-Atmel-CPU-for-2-banks-of-m.patch linux-2.6.30/.

cp 0010-Fix-memory-mapping-for-SPARSEMEM-to-use-both-banks-o.patch linux-2.6.30/.



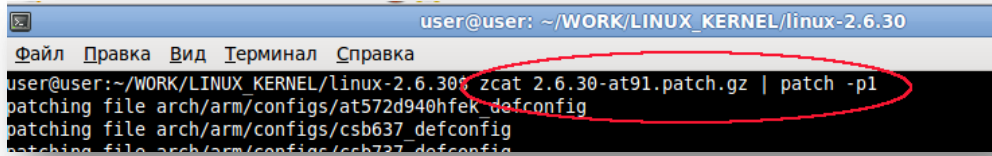
5. В результате в папке **linux-2.6.30** появятся скопированные файлы. Перейдем в папку, набрав в терминале команду

cd linux-2.6.30



6. Применим первый патч от Atmel командой

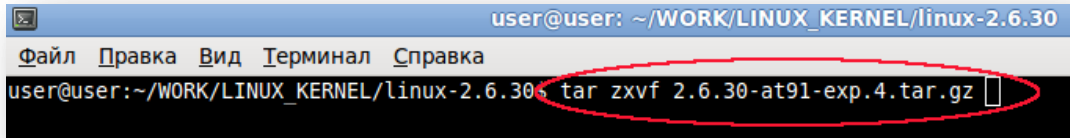
zcat 2.6.30-at91.patch.gz | patch -p1



```
user@user: ~/WORK/LINUX_KERNEL/linux-2.6.30
Файл Правка Вид Терминал Справка
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$ zcat 2.6.30-at91.patch.gz | patch -p1
patching file arch/arm/configs/at572d940hfeek_defconfig
patching file arch/arm/configs/csb637_defconfig
patching file arch/arm/configs/csb727_defconfig
```

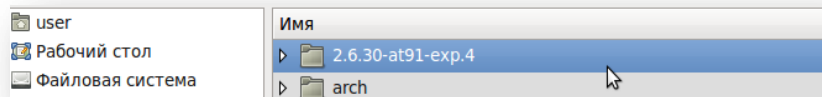
7. Распакуем архив экспериментальных патчей командой

tar zxvf 2.6.30-at91-exp.4.tar.gz



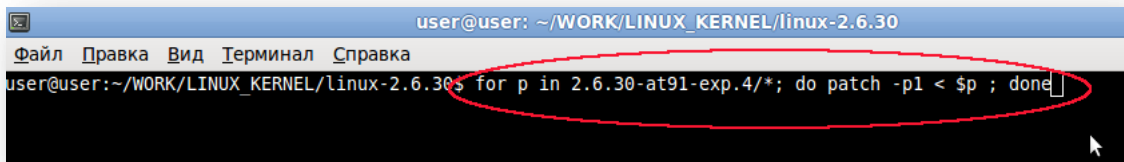
```
user@user: ~/WORK/LINUX_KERNEL/linux-2.6.30
Файл Правка Вид Терминал Справка
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$ tar zxvf 2.6.30-at91-exp.4.tar.gz
```

8. Появилась папка с патчами



9. Применим все патчи из этой папки одной командой

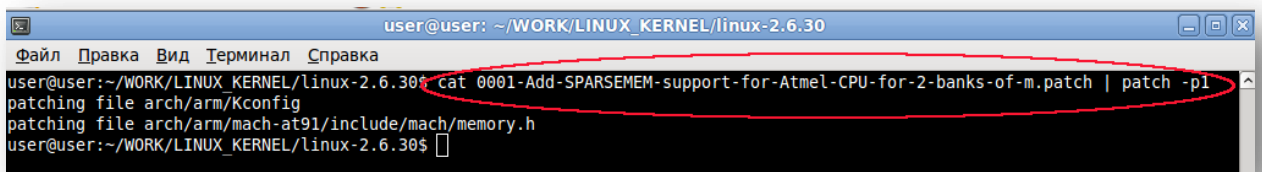
for p in 2.6.30-at91-exp.4/*; do patch -p1 < \$p ; done



```
user@user: ~/WORK/LINUX_KERNEL/linux-2.6.30
Файл Правка Вид Терминал Справка
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$ for p in 2.6.30-at91-exp.4/*; do patch -p1 < $p ; done
```

10. Применим патч

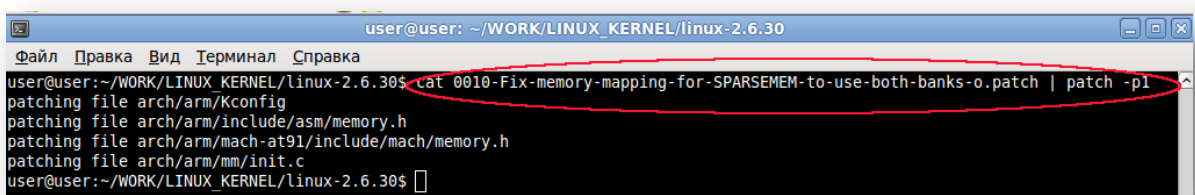
cat 0001-Add-SPARSEMEM-support-for-Atmel-CPU-for-2-banks-of-m.patch | patch -p1



```
user@user: ~/WORK/LINUX_KERNEL/linux-2.6.30
Файл Правка Вид Терминал Справка
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$ cat 0001-Add-SPARSEMEM-support-for-Atmel-CPU-for-2-banks-of-m.patch | patch -p1
patching file arch/arm/Kconfig
patching file arch/arm/mach-at91/include/mach/memory.h
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$
```

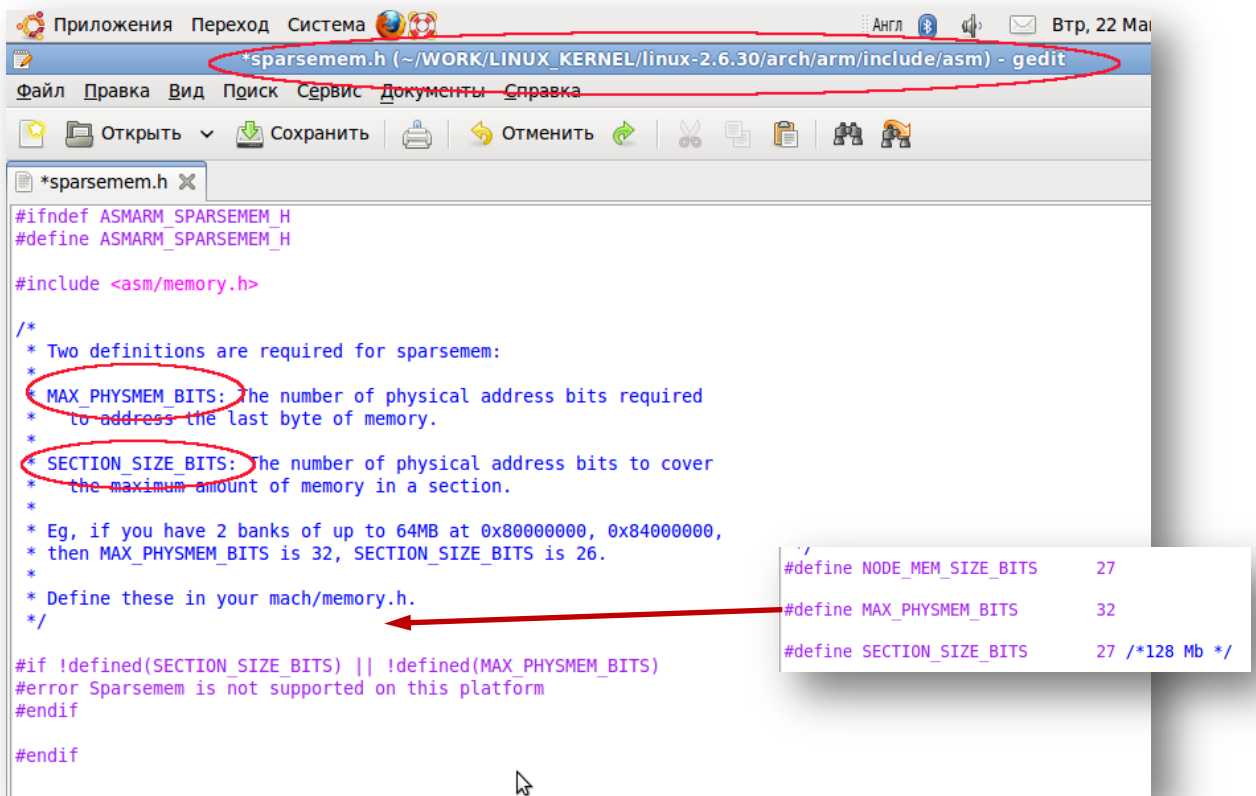
11. Применим патч

cat 0010-Fix-memory-mapping-for-SPARSEMEM-to-use-both-banks-o.patch | patch -p1



```
user@user: ~/WORK/LINUX_KERNEL/linux-2.6.30
Файл Правка Вид Терминал Справка
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$ cat 0010-Fix-memory-mapping-for-SPARSEMEM-to-use-both-banks-o.patch | patch -p1
patching file arch/arm/Kconfig
patching file arch/arm/include/asm/memory.h
patching file arch/arm/mach-at91/include/mach/memory.h
patching file arch/arm/mm/init.c
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$
```

12. Необходимо отредактировать файл `linux-2.6.30/arch/arm/include/asm/sparsemem.h`, добавив туда определение переменных `MAX_PHYSMEM_BITS` и `SECTION_SIZE_BITS`:



```
#!/bin/bash
export ARCH=arm
export CROSS_COMPILE=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-
export PATH=$PATH:/home/user/WORK/OE/stuff/tmp/staging/i686-linux/usr/bin
```

```
/*
 * Two definitions are required for sparsemem:
 *
 * MAX_PHYSMEM_BITS: The number of physical address bits required
 * to address the last byte of memory.
 *
 * SECTION_SIZE_BITS: The number of physical address bits to cover
 * the maximum amount of memory in a section.
 *
 * Eg, if you have 2 banks of up to 64MB at 0x80000000, 0x84000000,
 * then MAX_PHYSMEM_BITS is 32, SECTION_SIZE_BITS is 26.
 *
 * Define these in your mach/memory.h.
 */

#if !defined(SECTION_SIZE_BITS) || !defined(MAX_PHYSMEM_BITS)
#error Sparsemem is not supported on this platform
#endif

#endif
```

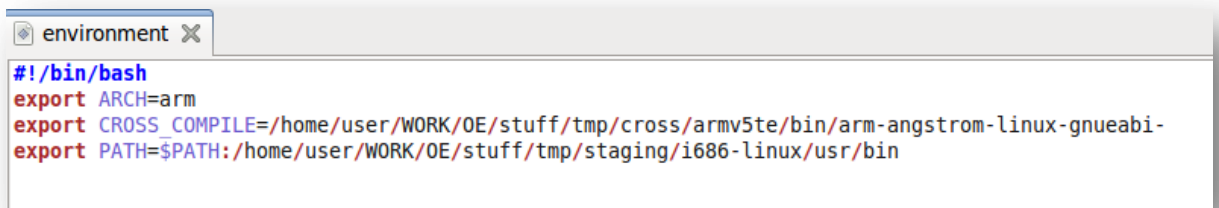
```
#define NODE_MEM_SIZE_BITS 27
#define MAX_PHYSMEM_BITS 32
#define SECTION_SIZE_BITS 27 /*128 Mb */
```

13. Зададим три необходимых для сборки переменных окружения. Для этого создадим скрипт `environment` командой

`gedit environment`

в котором пропишем:

```
#!/bin/bash
export ARCH=arm
export CROSS_COMPILE=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-
export PATH=$PATH:/home/user/WORK/OE/stuff/tmp/staging/i686-linux/usr/bin
```



```
#!/bin/bash
export ARCH=arm
export CROSS_COMPILE=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin/arm-angstrom-linux-gnueabi-
export PATH=$PATH:/home/user/WORK/OE/stuff/tmp/staging/i686-linux/usr/bin
```

14. Исполним скрипт командой

`source environment`

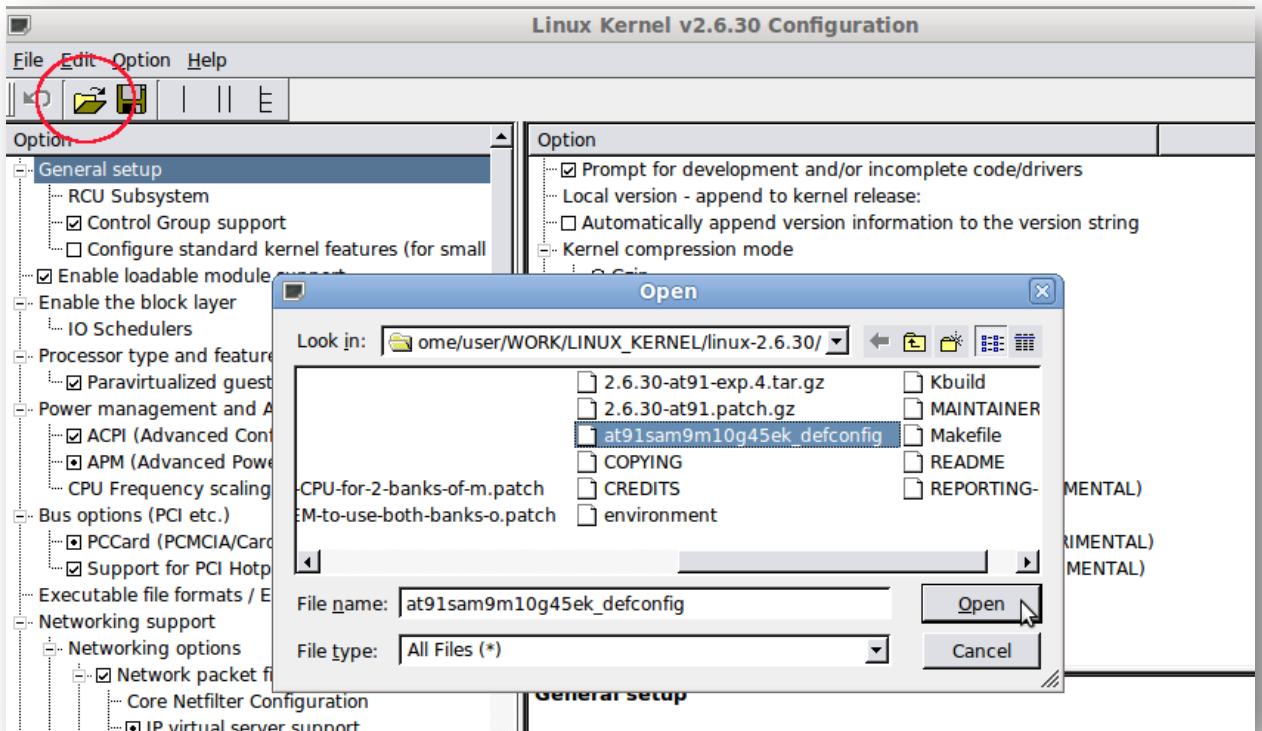
Убедиться в правильности настройки переменных окружения можно, введя в терминале команды `echo $ARCH`, `echo $CROSS_COMPILE`, `echo $PATH`.

15. Теперь можно запустить графический конфигуратор ядра linux, набрав команду

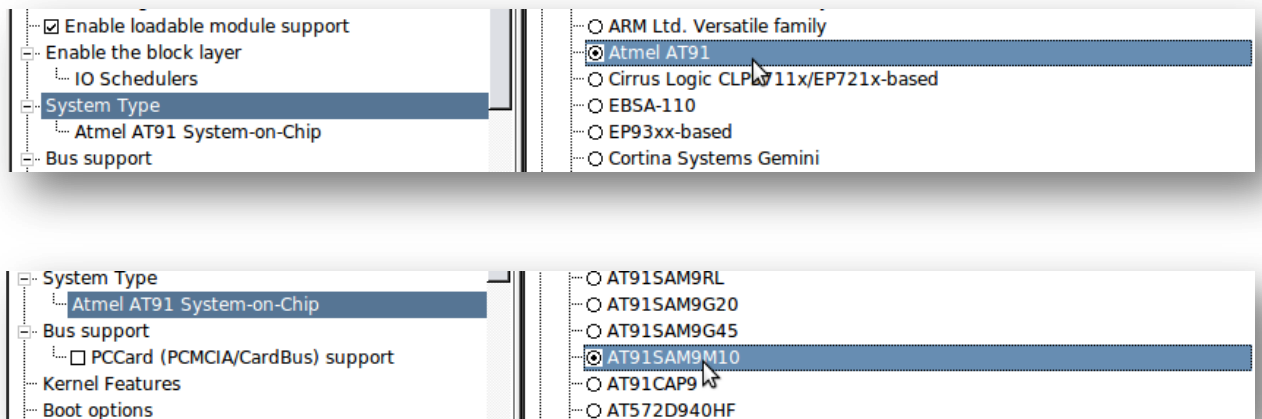
`make xconfig`

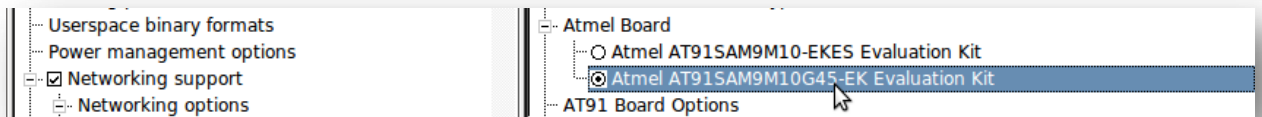

```
user@user: ~/WORK/LINUX_KERNEL/lin
Файл Правка Вид Терминал Справка
user@user:~/WORK/LINUX_KERNEL/linux-2.6.30$ make xconfig
HOSTCC scripts/basic/fixdep
scripts/basic/fixdep.c: In function 'traps':
scripts/basic/fixdep.c:377: warning: dereferencing type-punned pointer will
scripts/basic/fixdep.c:379: warning: dereferencing type-punned pointer will
HOSTCC scripts/basic/docproc
HOSTCC scripts/basic/hash
CHECK qt
HOSTCC scripts/Makefile.conf.o
```

16. Перед Вами появится окно конфигуратора, в котором необходимо нажать кнопку Open и в качестве config-файла выбрать **at91sam9m10g45ek_defconfig**:

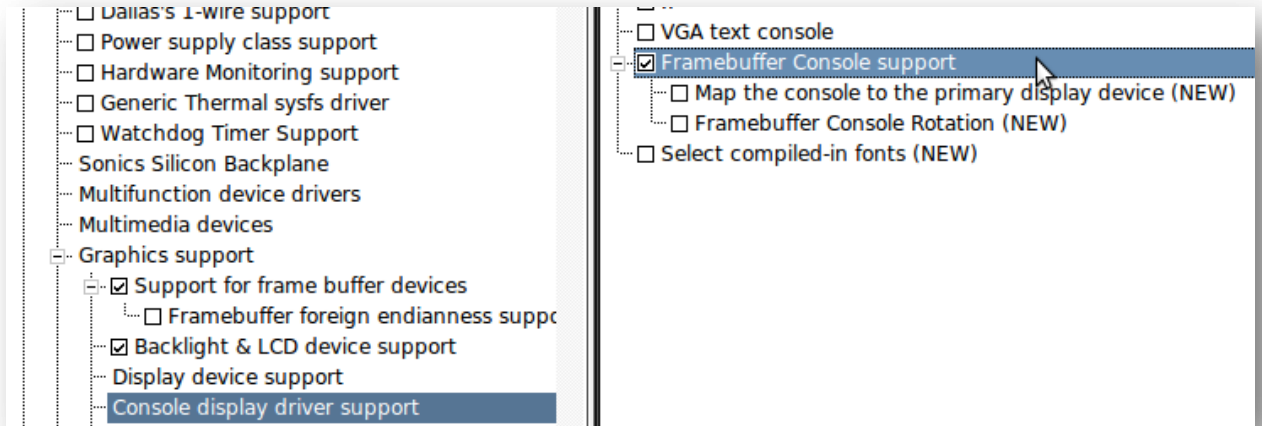


17. Теперь можно проверить, для той ли платформы мы конфигурируем ядро:

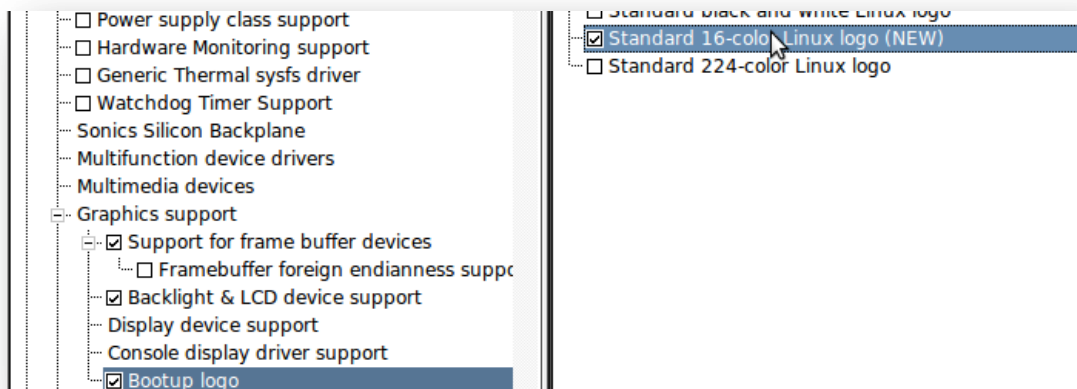




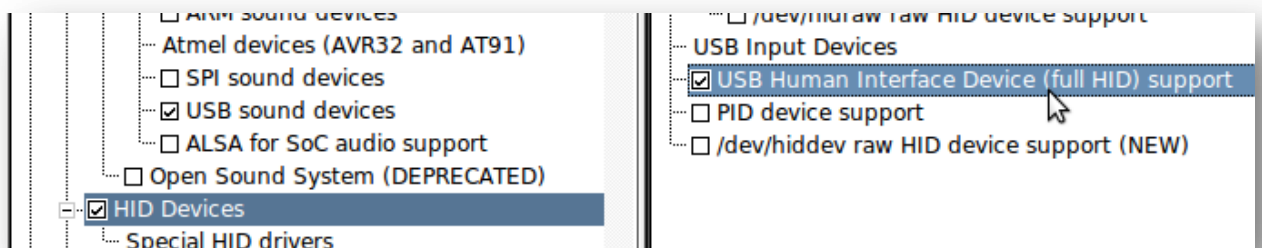
18. Теперь необходимо изменить кое-какие настройки. Для работы с графическим дисплеем, установленным на плате, в пункте **Grafics support** выделите **Console Display driver support** и в окне справа поставьте галочку в строке **Framebuffer Console support**.



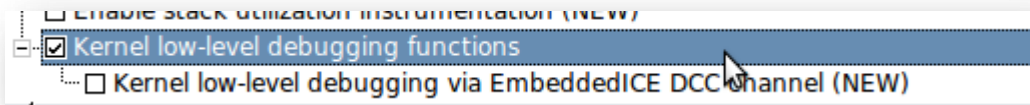
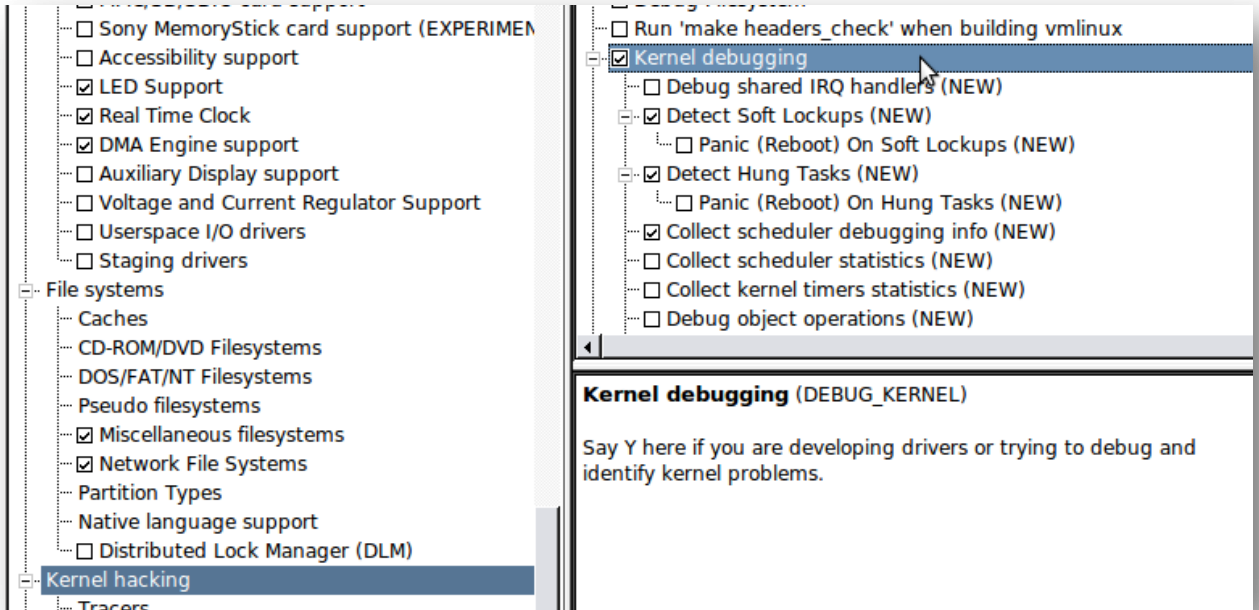
19. Для отображения логотипа linux во время загрузки ядра поставьте галочку в пункте меню **Bootup logo** и в окне справа отметьте строку **Standart 16-color Linux logo**:



20. Для hotplug-работы с HID-устройствами (например, USB-мышью или клавиатурой) выделите раздел **HID Devices** и в окне справа отметьте галочкой строку **USB Human Interface Device (full HID) support**:

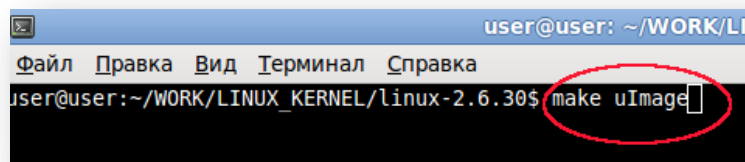


21. Для удобства отладки и выявления ошибок на стадии разработки удобно воспользоваться свойством **Kernel hacking** (выделите в окне слева). В окне справа галочкой отметьте пункт **Kernel debugging**, а в нем ниже **Kernel low-level debugging functions**:

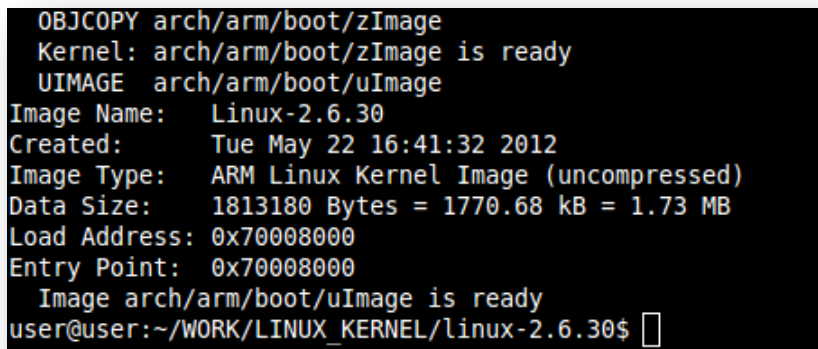


22. Для начальной работы с платой данной конфигурации вполне достаточно. Нажмите кнопку Save и закройте конфигуратор.

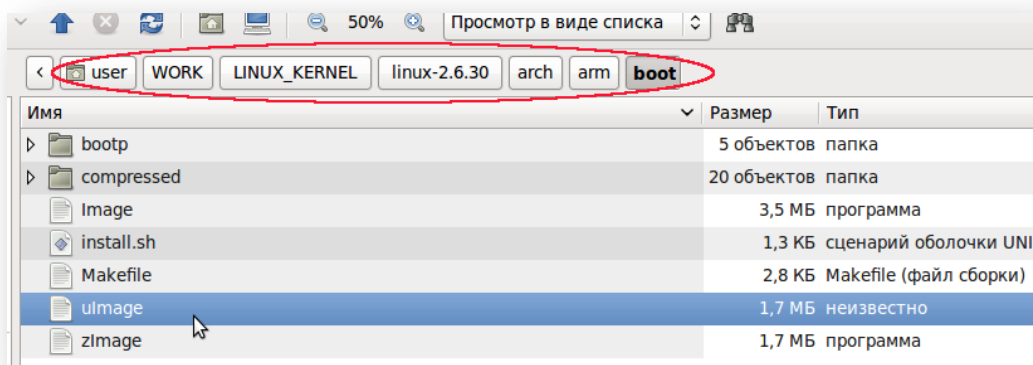
23. Теперь можно запустить сборку образа ядра командой **make uImage**



24. Через небольшой промежуток времени сборка завершается:

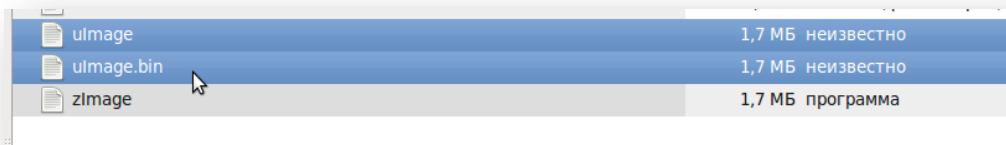


25. Образ ядра, готовый для загрузки при помощи загрузчика U-Boot находится в папке **linux-2.6.30/arch/arm/boot/** и называется **ulmage**:



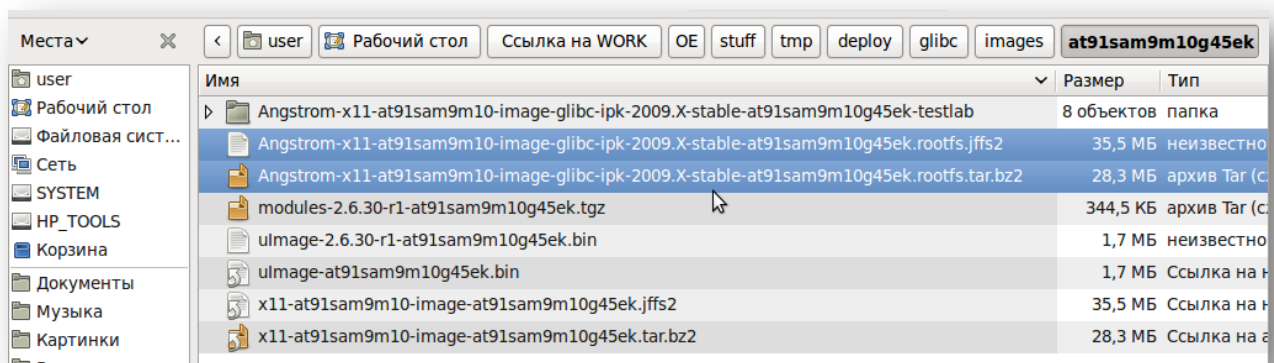
26. В дальнейшем мы будем загружать в Nandflash полученный образ программой SAM-BA, которая видит файлы с расширением .bin. Поэтому для удобства сразу скопируем файл **ulmage** в ту же папку и назовем его **ulmage.bin**:

`cp arch/arm/boot/ulmage arch/arm/boot/ulmage.bin`



2.3.5 Сборка консольного образа файловой системы

1. Графический образ файловой системы уже предварительно собран и находится в папке **/home/user/WORK/OE/stuff/tmp/deploy/glibc/images/at91sam9m10g45ek**. Файл **Angstrom-x11-at91sam9m10-image-glibc-ipk-2009.X-stable-at91sam9m10g45ek** с расширением **.rootfs.jffs2** предназначен для «прошивки» в Nandflash-память отладочного комплекта. Кроме того в папке есть файл с тем же названием, но в виде tar-архива, который мы используем для переноса образа файловой системы на Flash-карту памяти.



2. Мы соберем консольный образ файловой системы (без поддержки оконной системы X11 и рабочего стола). Большинство пакетов уже были готовы на предыдущем этапе сборки графического образа, поэтому процесс не займет много времени.

3. Перейдите в папку **/home/user/WORK/OE/stuff**, набрав в терминале команду

`cd /home/user/WORK/OE/stuff`

```
Файл Правка Вид Терминал Справка
user@user:~$ cd /home/user/WORK/OE/stuff/
user@user:~/WORK/OE/stuff$
```

4. Исполните скрипт `oe_env.sh` командой

`source oe_env.sh`

Данный скрипт устанавливает переменные окружения, в том числе путь к инструменту сборки `bitbake`’у, к файлам-правилам для сборки различных пакетов и т.д. Проверить правильность установки, например, переменной `PATH` можно командой **`echo $PATH`**:

```
Файл Правка Вид Терминал Справка
user@user:~/WORK/OE/stuff$ source oe_env.sh
user@user:~/WORK/OE/stuff$ echo $PATH
/home/user/WORK/OE/stuff/bitbake/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
user@user:~/WORK/OE/stuff$
```

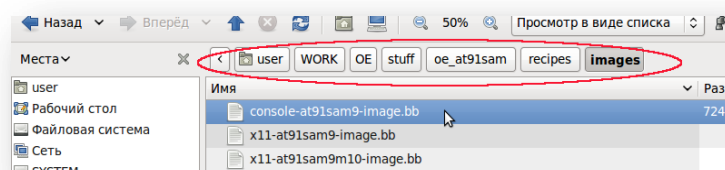
5. Теперь для запуска **`bitbake`** необходимо из-под суперпользователя задать две системные переменные. Для этого в терминале наберите команду **`sudo su`** и в строке предложения введите пароль наберите **`user`**. В терминале введите команду **`echo 0 > /proc/sys/vm/mmap_min_addr`**

```
Файл Правка Вид Терминал Справка
user@user:~/WORK/OE/stuff$ sudo su
[sudo] password for user:
root@user:/home/user/WORK/OE/stuff# echo 0 > /proc/sys/vm/mmap_min_addr
root@user:/home/user/WORK/OE/stuff#
```

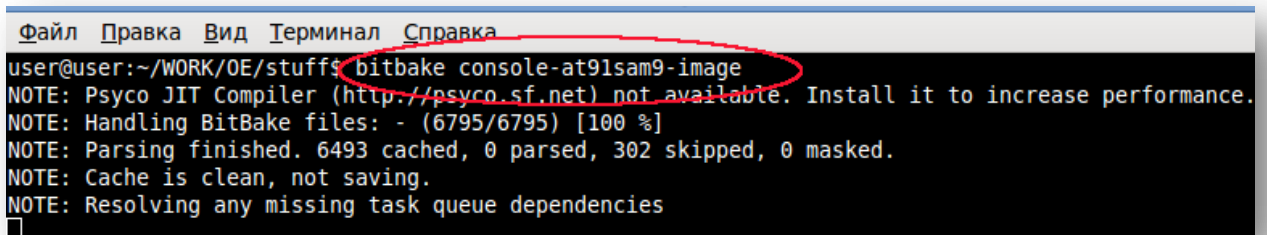
6. После этого наберите в консоли **`sysctl vm.mmap_min_addr=0`**

```
Файл Правка Вид Терминал Справка
root@user:/home/user/WORK/OE/stuff# sysctl vm.mmap_min_addr=0
vm.mmap_min_addr = 0
root@user:/home/user/WORK/OE/stuff#
```

7. Наберите в терминале **`exit`** или нажмите клавиши `Ctrl-D` и выйдите из режим суперпользователя.
8. Правила для `bitbake`, по которым он будет собирать наш образ, хранятся в файле **`console-at91sam9-image.bb`**, который лежит в папке **`/home/user/WORK/OE/stuff/oe_at91sam/recipes/images`**

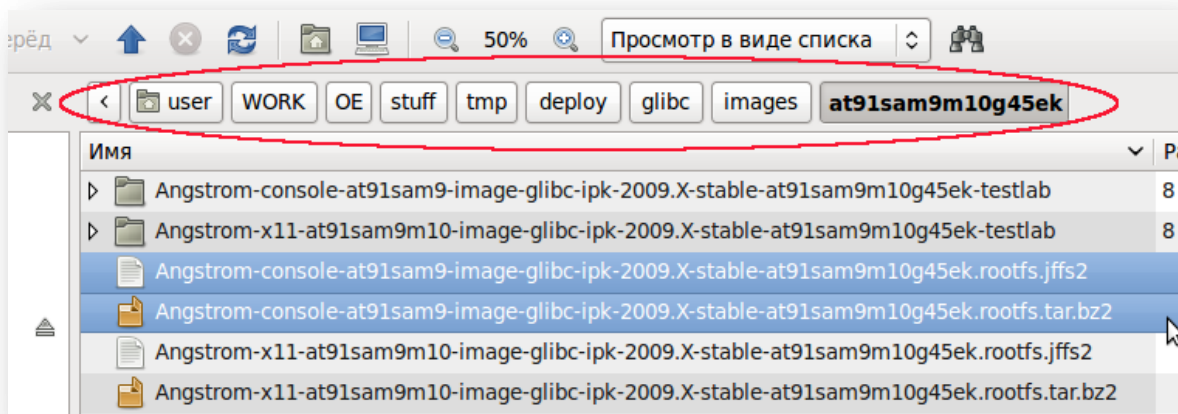


9. Запустите сборку образа командой **bitbake console-at91sam9-image**



```
Файл Правка Вид Терминал Справка
user@user:~/WORK/OE/stuff$ bitbake console-at91sam9-image
NOTE: Psyco JIT Compiler (http://psyco.sf.net) not available. Install it to increase performance.
NOTE: Handling BitBake files: - (6795/6795) [100 %]
NOTE: Parsing finished. 6493 cached, 0 parsed, 302 skipped, 0 masked.
NOTE: Cache is clean, not saving.
NOTE: Resolving any missing task queue dependencies
```

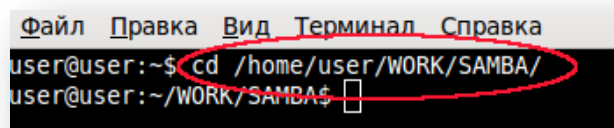
10. После завершения сборки в уже указанной выше папке появляются два нужных нам файла с расширением .jffs2 и tar:



2.4 «Прошивка» собранных компонентов в Nandflash-память отладочного комплекта в программе SAM-BA

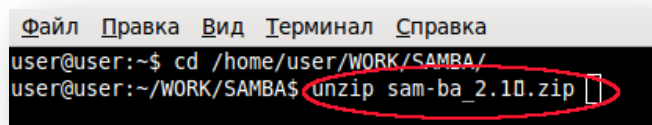
2.4.1 Настройка программы SAM-BA

1. Зайдите в папку /home/user/WORK/SAMBA, набрав команду **cd /home/user/WORK/SAMBA**



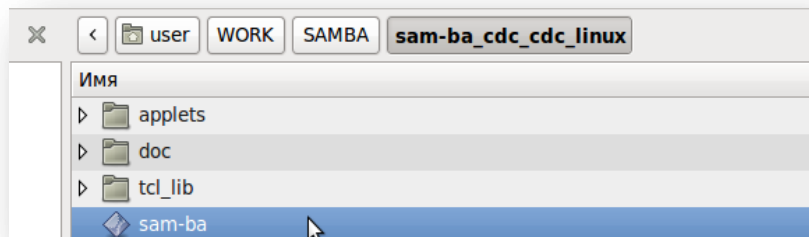
```
Файл Правка Вид Терминал Справка
user@user:~$ cd /home/user/WORK/SAMBA/
user@user:~/WORK/SAMBA$
```

2. В папке лежит архив sam-ba_2.10.zip, который можно распаковать, набрав в терминале команду **unzip sam-ba_2.10.zip**

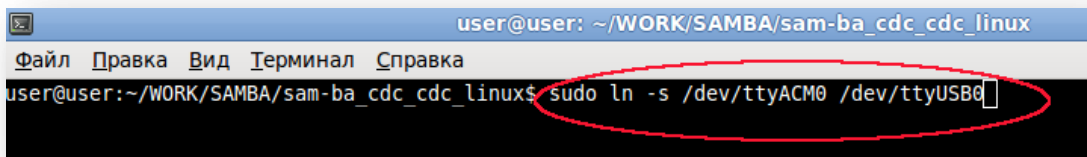


```
Файл Правка Вид Терминал Справка
user@user:~$ cd /home/user/WORK/SAMBA/
user@user:~/WORK/SAMBA$ unzip sam-ba_2.10.zip
```

Зайдите в полученную после распаковки папку **sam-ba_cdc_linux**, набрав в терминале команду **cd sam-ba_cdc_linux/**

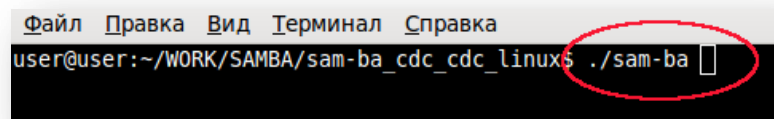


3. В ней лежит исполняемый файл программы `sam-ba`, при помощи которой мы будем «зашивать» собранные компоненты в память.
4. На отладочной плате уберите джамперы **JP12** и **JP10**, чтобы активировать встроенный BOOT-ROM загрузчик для работы с SAM-BA.
5. Соедините плату с ноутбуком кабелем USB.
6. Включите питание платы и нажмите кнопку аппаратного сброса **BP1 (NRST)**.
7. В терминале наберите команду **`sudo ln -s /dev/ttyACM0 /dev/ttyUSB0`**

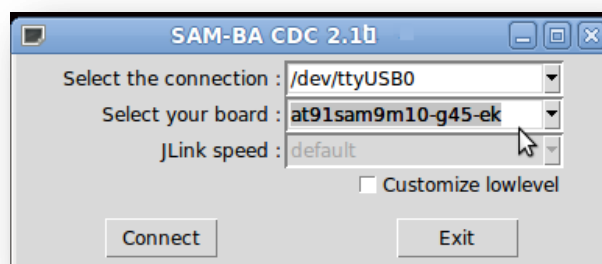


Это делается для того, чтобы SAMBA «увидела» нашу плату. Так как она определяется в системе как устройство `/dev/ttyACM0`, а SAMBA работает с устройствами `/dev/ttyUSB0`, то необходимо создать символическую ссылку на `/dev/ttyACM0` и назвать ее `/dev/ttyUSB0`.

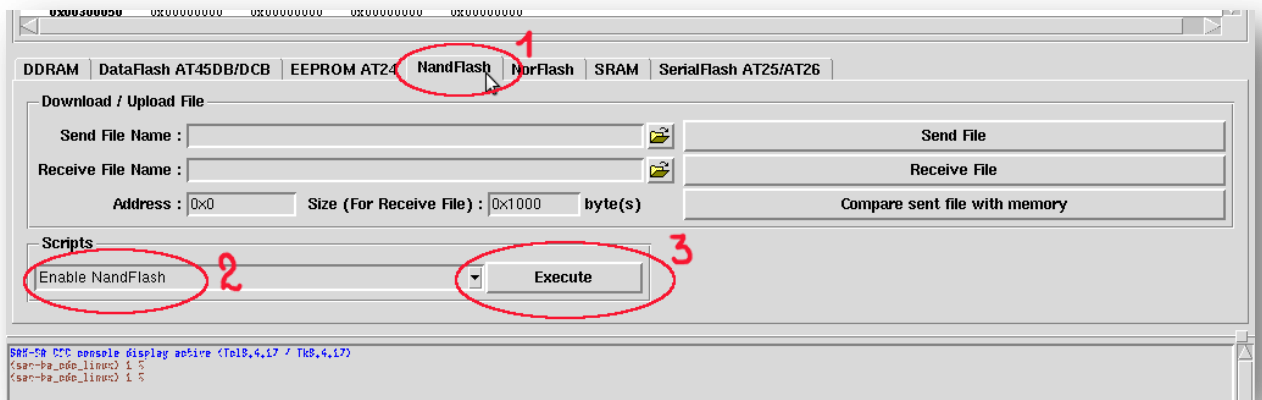
8. Запустите в терминале исполняемый файл программы, набрав **`./sam-ba`**



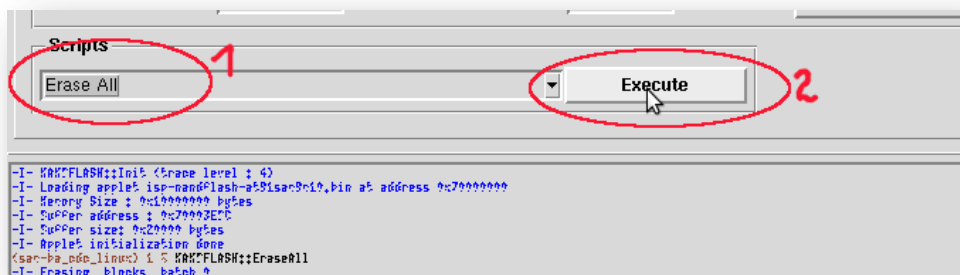
9. Перед Вами появится окно, в котором необходимо выбрать из выпадающего списка плату `at91sam9m10-g45-ek` и нажать Connect:



10. В появившемся рабочем окне выберите вкладку Nandflash.
11. Верните на место джамперы JP10 и JP12.
12. В меню Scripts выберите пункт Enable Nandflash и нажмите кнопку Execute:

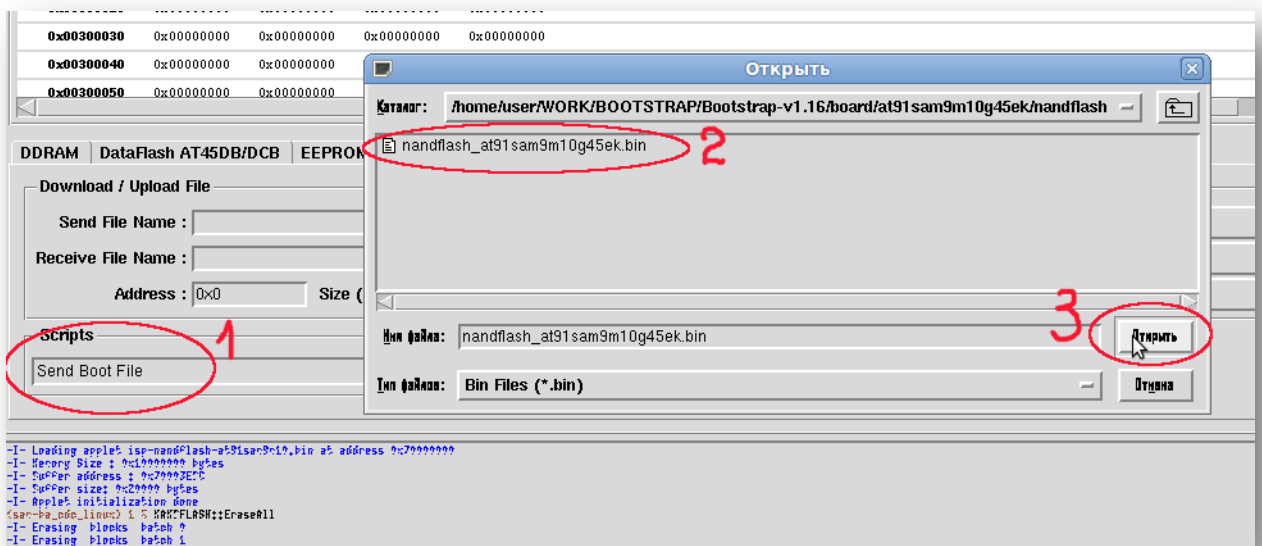


13. Сотрите всю Nandflash, выбрав в окне Scripts пункт **Erase All**, и нажмите кнопку **Execute**.



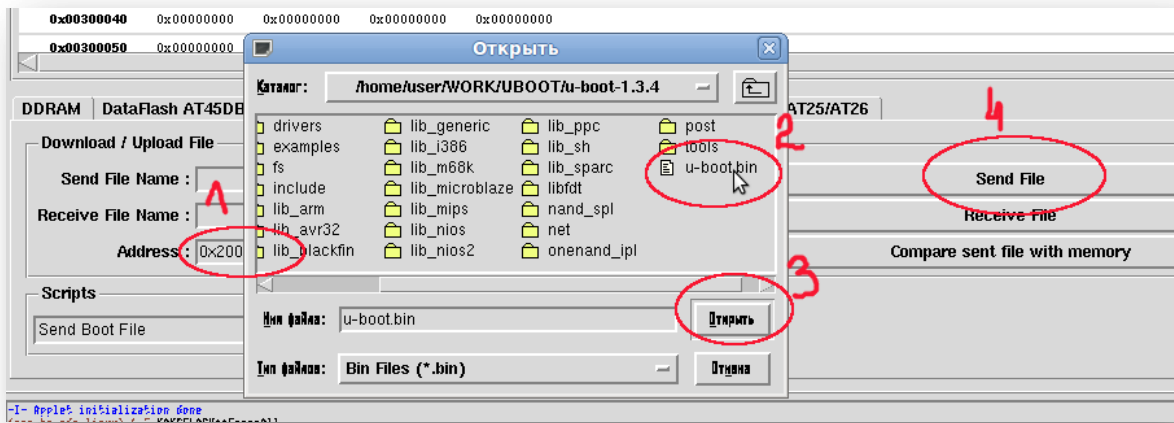
2.4.2 «Прошивка» BOOTSTRAP

1. В окне Scripts выберите пункт Send Boot File и нажмите кнопку Execute. В появившемся окне выбора файла выберите собранный ранее файл предзагрузчика:



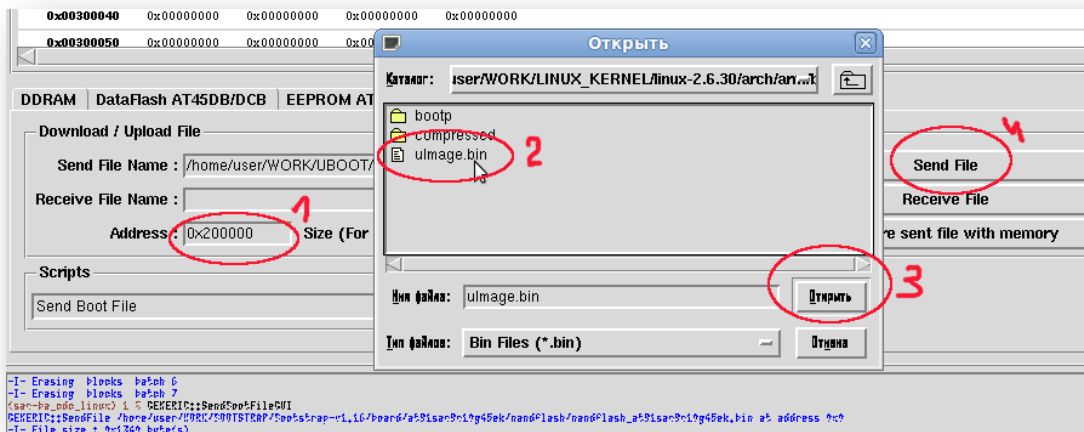
2.4.3 «Прошивка» UBOOT

1. В поле **Address** введите значение **0x20000** и справа от строки **Send File Name** нажмите кнопку выбора файла. В появившемся окне выберите скомпилированный ранее файл **uboot.bin**, нажмите кнопку **Send File**:



2.4.4 «Прошивка» LINUX KERNEL

1. В поле **Address** введите значение **0x200000** и справа от строки **Send File Name** в появившемся окне выберите файл собранного ранее образа ядра **ulmage.bin**, нажмите кнопку **Открыть**, а затем **Send File**:

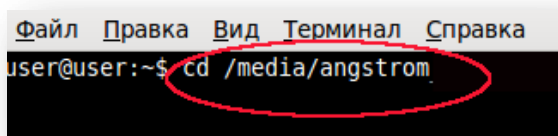


2. Закройте окно программы.

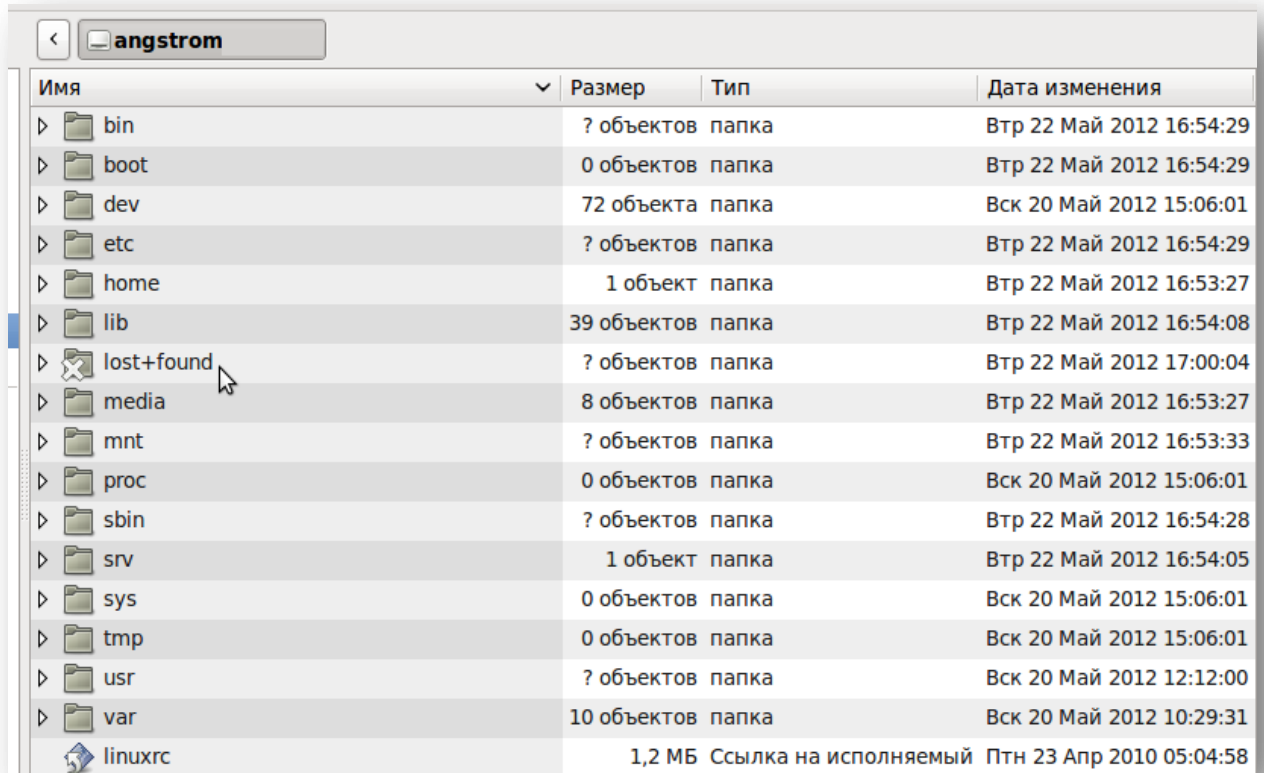
2.5 Подготовка Flash-карты памяти для загрузки

2.5.1 Запись tar архива графического образа файловой системы на Flash-карту

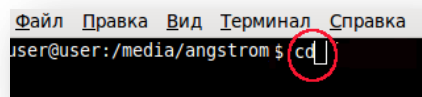
1. Часто при разработке удобно файловую систему хранить на съемных носителях, чтобы оперативно вносить требуемые изменения. В данной работе мы поступим таким образом с графической файловой системой.
2. Вставьте Flash-карту в USB-порт ноутбука. Она отформатирована в ext2-формат и определяется в системе как **/media/angstrom**.
3. Зайдите на «флешку», набрав в терминале команду **cd /media/angstrom**



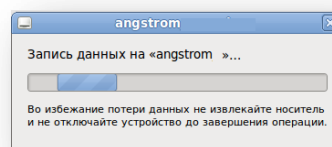
- Из-под суперпользователя распакуйте tar-архив графического образа файловой системы, набрав в терминале команду
`sudo tar xvf /home/user/WORK/OE/stuff/tmp/deploy/glibc/images/at91sam9m10g45ek/Angstrom-x11-at91sam9m10-image-glibc-ipk-2009.X-stable-at91sam9m10g45ek.rootfs.tar.bz2`
- В результате на «флешке» появилось дерево папок – файловая система:



- Flash-карту необходимо «освободить», то есть необходимо набрать в терминале команду `cd`, чтобы покинуть «флешку»:



- «Флешку» необходимо безопасно извлечь. Для этого правой кнопкой мыши кликните на значок `angstrom` в файловом браузере (или на рабочем столе) и выберите пункт **«Извлечь»**.

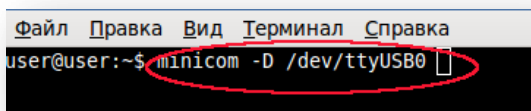


- Вставьте подготовленную Flash-карту в USB-host порт платы.

2.6 Запуск системы, работа в программе MINICOM, настройка параметров U-BOOT

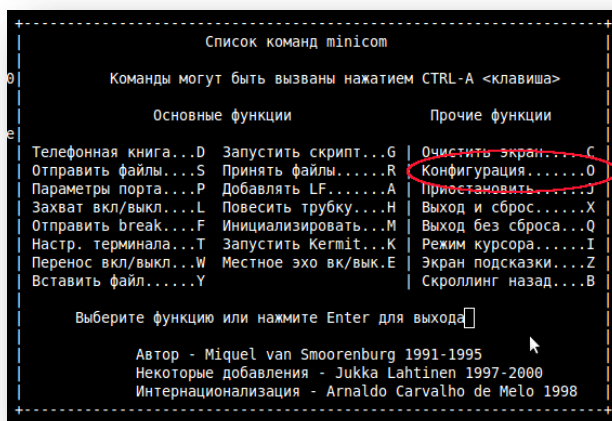
2.6.1 Вариант загрузки с внешней Flash-карты памяти

1. Отсоедините USB-кабель, с помощью которого мы «прошивали» плату, от ноутбука и в тот же USB-порт вставьте USB-часть переходника USB-COM. Второй конец адаптера вставьте в DBGU-порт отладочной платы.
2. Вызовите в терминале программу MINICOM командой ***minicom -D /dev/ttyUSB0***



```
Файл Правка Вид Терминал Справка
user@user:~$ minicom -D /dev/ttyUSB0
```

3. После надписи на экране «Инициализируется модем...» перед Вами появится меню настройки программы, в котором нужно выбрать пункт «Конфигурация», для этого нажмите кнопку O (в английской раскладке) на клавиатуре ноутбука.



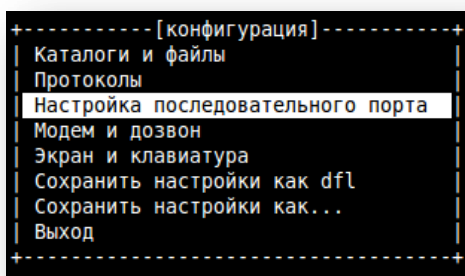
```
-----
Список команд minicom
-----
Команды могут быть вызваны нажатием CTRL-A <клавиша>

Основные функции          Прочие функции
-----
Телефонная книга...D     Запустить скрипт...G     Очистить экран...C
Отправить файлы...S       Принять файлы...R        Конфигурация...O
Параметры порта...P       Добавлять LF...A         Присоединить...J
Захват вкл/выкл...L       Повесить трубку...H      Выход и сброс...X
Отправить break...F       Инициализировать...M     Выход без сброса...Q
Настр. терминала...T      Запустить Kermit...K     Режим курсора...I
Перенос вкл/выкл...W      Местное эхо вкл/выкл...E  Экран подсказки...Z
Вставить файл...Y         | Скроллинг назад...B

Выберите функцию или нажмите Enter для выхода

Автор - Miquel van Smoorenburg 1991-1995
Некоторые добавления - Jukka Lahtinen 1997-2000
Интернационализация - Arnaldo Carvalho de Melo 1998
-----
```

4. Далее перед Вами появится подменю, в котором клавишами «Вверх-вниз» нужно выбрать пункт «Настройка последовательного порта»:



```
+-----[конфигурация]-----+
| Каталоги и файлы
| Протоколы
| Настройка последовательного порта
| Модем и дозвон
| Экран и клавиатура
| Сохранить настройки как dfl
| Сохранить настройки как...
| Выход
+-----+-----
```

5. В появившемся подменю необходимо только изменить свойство «Аппаратное управление потоком», нажав клавишу F на клавиатуре. Затем нажмите Enter:

```

A+-----+
a| A - Последовательный порт      : /dev/ttyUSB0
t| B - Размещение lock-файла     : /var/lock
| C - Программа при выходе      :
M| D - Программа при запуске     :
| E - Скорость/Чётность/Биты    : 115200 8N1
| F - Аппаратное управление потоком : Нет
| G - Программное управление потоком : Нет
|
| Какую настройку изменить? [ ]
+-----+

```

6. Во вновь появившемся меню «конфигурация» выберите строку «Сохранить настройки как dfl...» и нажмите Enter.

```

----- [ конфигурация ] -----
| Каталоги и файлы
| Протоколы
| Настройка последовательного порта
| Модем и дозвон
| Экран и клавиатура
| Сохранить настройки как dfl
| Сохранить настройки как...
| Выход
+-----+

```

7. Нажмите на плате кнопку BP1 (NRST) и на экране ноутбука теперь Вы можете наблюдать процесс загрузки каждого из собранных компонентов нашей системы. Загрузка Bootstrap не отображается на экране, так как по умолчанию в нем отключена возможность вывода отладочной информации через DBGU. Зато процесс загрузки U-Boot мы можем наблюдать достаточно подробно. Обратим внимание на несколько пунктов. Во-первых, U-Boot уже работает с обоими банками SDRAM, для чего мы и применяли патч в самом начале. Во-вторых строка “Warning – Bad CRC or NAND, using default Environment” означает, что после того, как мы стерли всю Nandflash, не записали переменные окружения для U-Boot (которые стандартно находятся в Nandflash по адресу 0x60000). Не обращайте внимания на данное предупреждение. Позже мы зададим параметры U-Boot и это замечание перестанет появляться.

```

U-Boot 1.3.4 (May 22 2012 - 16:13:36)
DRAM: 256 MB
NAND: 256 MiB
DataFlash:AT45DB321
Nb pages: 8192
Page Size: 528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C00041FF (RO) Bootstrap
Area 1: C0004200 to C00083FF Environment
Area 2: C0008400 to C0041FFF (RO) U-Boot
Area 3: C0042000 to C0251FFF Kernel
Area 4: C0252000 to C041FFFF FS
*** warning - bad CRC or NAND, using default environment
In: serial
Out: serial
Err: serial
Net: macb0
macb0: Starting autonegotiation...

```

8. Остановим загрузку U-boot, нажав любую клавишу на ноутбуке. Зададим сценарий загрузки U-Boot, а именно укажем ему, откуда он должен скопировать образ ядра linux.

Для этого откройте файл /home/user/WORK/UBOOT/Параметры для U-BOOT и скопируйте оттуда строку **set bootcmd 'nand read 0x22200000 0x00200000 0x002729F0; bootm 0x22200000'**. Вставьте эту строку в терминал, нажав клавиши Ctr-Shift-V. Теперь в параметрах укажем ядру, что необходимо монтировать файловую систему со съемного диска /dev/sda1, которым будет являться подготовленная заранее нами «флешка» - **set bootargs 'mem=128M@0x20000000 mem=128M@0x70000000 console=ttyS0, 115200 root=/dev/sda1 rw rootdelay=10'**. Здесь rootdelay – это задержка в секундах перед загрузкой файловой системы, чтобы ядро успело подмонтировать съемный диск. Вставьте параметры в строку U-Boot'а таким же образом.

```

Параметры для U-BOOT x
//Задание сценария загрузки U-Boot
set bootcmd 'nand read 0x22200000 0x00200000 0x002729F0; bootm 0x22200000' 1

//Задание параметров ядра при расположении образа файловой системы на Flash-карте памяти 2
set bootargs 'mem=128M@0x20000000 mem=128M@0x70000000 console=ttyS0,115200 root=/dev/sda1 rw rootdelay=10'

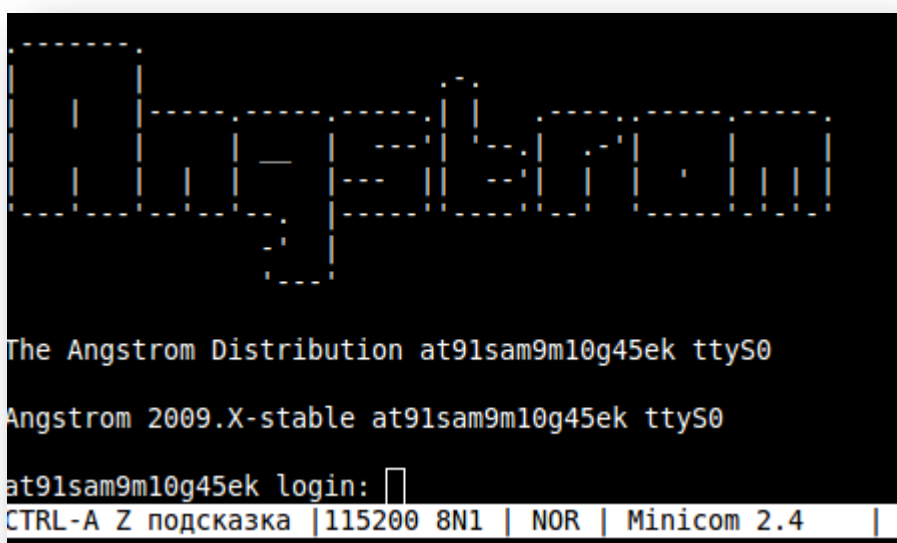
//Сохранение заданных параметров во flash-память
saveenv

//Сброс и перезагрузка контроллера
reset

```

Эти параметры U-Boot передает ядру. Здесь **mem=128M@0x20000000 mem=128M@0x70000000** означает, что мы используем оба банка оперативной памяти, **console=ttyS0, 115200** указывает, через какое устройство и с какой скоростью микроконтроллер выводит отладочную информацию. Сохраним параметры U-Boot в Nandflash памяти, введя команду **saveenv**. Теперь каждый раз, если не прерывать загрузку и не менять параметры вручную, ядро будет загружать файловую систему со съемного носителя.

9. Теперь можно загрузиться. Наберите в терминале команду **boot**.
10. На экране отображается процесс загрузки и распаковки образа ядра, монтирования файловой системы. В результате на экране ноутбука появится приглашение войти в систему Angstrom. Введите имя пользователя root. По умолчанию пароль пустой.



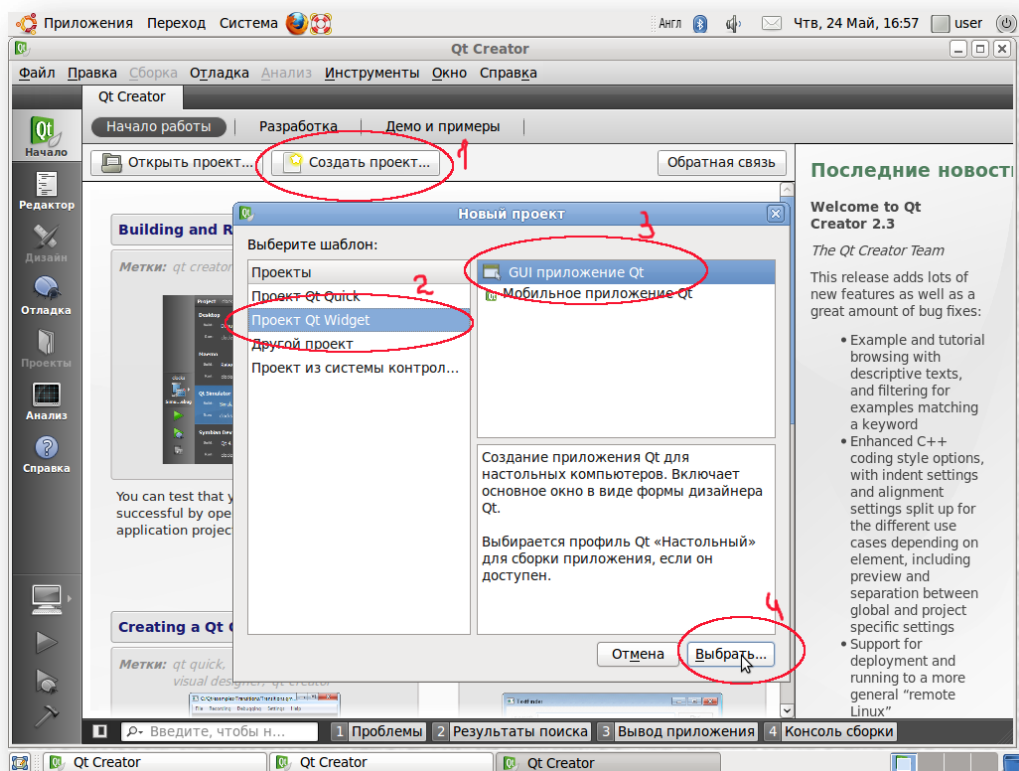
3.7 Создание GUI-приложения в среде разработки QT Creator

3.7.1 Создание GUI-проекта в среде QT-Creator

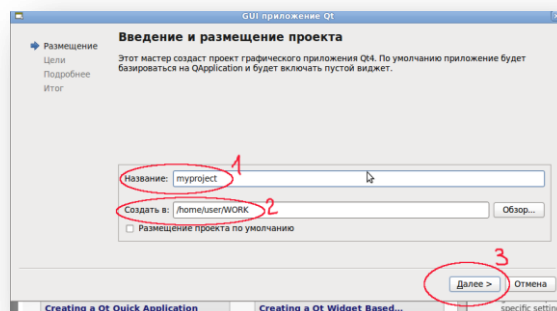
1. Найдите на рабочем столе ярлык QtCreator и двойным щелчком откройте программу:



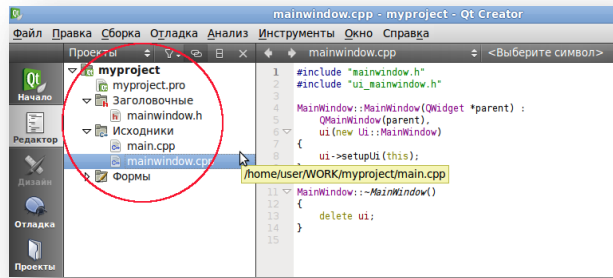
2. В появившемся окне найдите кнопку «Создать проект», далее выделите пункт «Проект QT Widget» и справа «GUI приложение QT». Нажмите кнопку «Выбрать»:



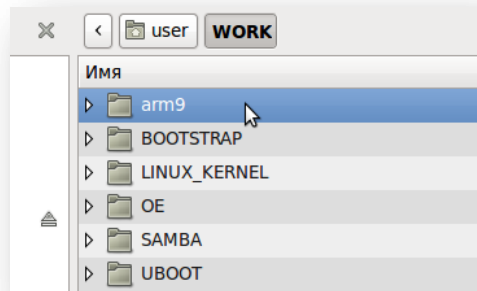
3. Перед Вами откроется окно «Введение и размещение проекта». Введите имя проекта – например, myproject. Для размещения выберите, например, рабочую папку WORK. Нажмите кнопку «Далее».



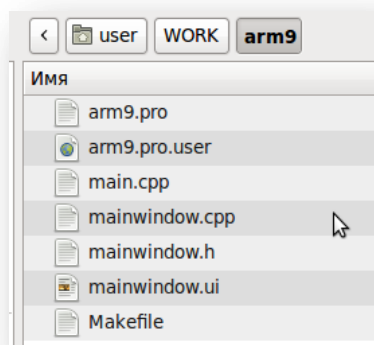
4. Перед Вами откроется окно, в котором слева отображается структура проекта, а в основном поле текстовый редактор:



5. Файл проекта называется myproject.pro. По умолчанию QtCreator создает два сpp-файла с исходными кодами – main.cpp и mainwindow.cpp – и заголовочный файл mainwindow.h. Также в проект входит и файл форм mainwindow.ui, в котором вручную можно добавлять кнопки, текстовые поля, списки и другие элементы пользовательского интерфейса.
6. В рабочем каталоге WORK найдите папку arm9 – это пример простого проекта:



7. Откройте файл main.cpp из папки arm9 и скопируйте все его содержимое в файл main.cpp своего проекта myproject. То же самое сделайте с файлами mainwindow.cpp, mainwindow.h.



8. Программировать в среде QT удобно на языке с++. Поэтому все элементы программы организованы в виде классов – окна, виджеты, элементы управления и т. д. Хотя можно написать программу на С и использовать QT для автоматического создания Makefile'а и компиляции.
9. Рассмотрим кратко содержимое файла main.cpp:

```

main.cpp main(int, char *[])
#include <QtGui/QApplication>
#include <QApplication>
#include <QDesktopWidget>
#include <QWidget>
#include "mainwindow.h"
#include <QTextCodec>
#include <QStyle>
#include <QCleanlooksStyle>
#include <QMotifStyle>
#include <QPlastiqueStyle>
MainWindow *Window_Main; 1
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Window_Main = new MainWindow; 2
    Window_Main->show();
    Window_Main->setWindowTitle(QString::fromLocal8Bit("AT91SAM9ML0G45-EK"));
    return a.exec();
}

```

- 1 – Объявление переменной Window_Main, которая имеет класс MainWindow, описанный в заголовочном файле mainwindow.h.
- 2 – Динамическое создание нового объекта класса MainWindow под именем Window_Main. Вызов функции show для отображения созданного окна на дисплее. Вызов функции SetWindowTitle для присвоения имени созданному окну.

10. Рассмотрим файл mainwindow.h, в котором содержится объявление класса MainWindow:

Здесь мы объявляем все переменные и объекты и функции, принадлежащие данному классу, а также слоты – функции, вызываемые автоматически при наступлении какого-либо стандартного события (например, нажатия кнопки). В данном примере у нас на форме будут две кнопки QPushButton1, QPushButton2, поле ввода текста QLineEdit. Для объединения всех элементов, их структурирования и размещения на форме используется элемент класса QVBoxLayout под именем LayoutMain. Также в разделе private slots мы объявляем два слота – PrintText1 и PrintText2, которые будут вызываться при нажатии кнопок QPushButton1 и PushBitton2 соответственно.

11. Рассмотрим файл mainwindow.cpp, в котором прописан конструктор класса MainWindow, а также слоты PrintText1, PrintText2.

```

mainwindow.h <Выбери>
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLineEdit>
#include <QPushButton>
#include <QLayout>
#include <QString>
#include <QTextCodec>

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0);

    QVBoxLayout *LayoutMain;

    QPushButton *PushButton1;
    QPushButton *PushButton2;
    QLineEdit *LineEdit;

private slots:
    void PrintText1 ();
    void PrintText2 ();
};

#endif // MAINWINDOW_H

```

```

mainwindow.cpp* MainWindow::MainWindow(QWidget *parent)
: QWidget(parent)
{
    LayoutMain = new QVBoxLayout;

    PushButton1 = new QPushButton;
    PushButton1->setText("PUSH1");

    PushButton2 = new QPushButton;
    PushButton2->setText("PUSH2");

    LineEdit = new QLineEdit;

    LayoutMain->addWidget(PushButton1);
    LayoutMain->addWidget(PushButton2);
    LayoutMain->addWidget(LineEdit);

    this->setLayout(LayoutMain);

    connect(PushButton1, SIGNAL(clicked()), this, SLOT(PrintText1()));
    connect(PushButton2, SIGNAL(clicked()), this, SLOT(PrintText2()));
}

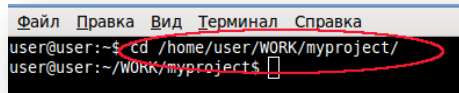
void MainWindow::PrintText1()
{
    LineEdit->setText("TEXT1");
}

void MainWindow::PrintText2()
{
    LineEdit->setText("TEXT2");
}

```

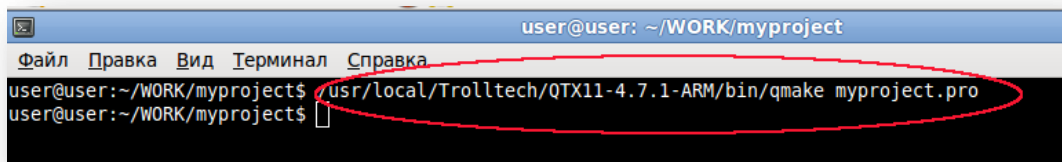

3.7.2 Кросс-компиляция проекта

1. Теперь нужно скомпилировать созданный проект под ARM-архитектуру. Для этого перейдите в папку с проектом, набрав в терминале ***cd /home/user/WORK/myproject***



```
Файл Правка Вид Терминал Справка
user@user:~$ cd /home/user/WORK/myproject/
user@user:~/WORK/myproject$
```

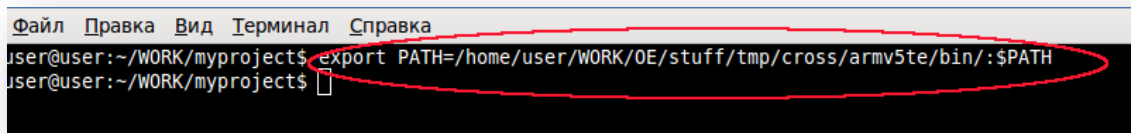
2. В папке ***/usr/local/Trolltech/QTx11-4.7.1-ARM*** находятся инструменты и библиотеки QT, позволяющие работать с графикой в оконной системе X11. Они могут быть сконфигурированы и собраны из исходных кодов под любую платформу. В данном случае библиотеки и инструменты QT предназначены для работы с архитектурой ARM. В папке ***/usr/local/Trolltech/QTx11-4.7.1-ARM/bin*** находится инструмент ***qmake***, который автоматически создает Makefile проекта QT в соответствии со спецификацией, которая была выбрана при конфигурировании библиотек. Вызываем ***qmake*** командой ***/usr/local/Trolltech/QTx11-4.7.1-ARM/bin/qmake myproject.pro***



```
user@user: ~/WORK/myproject
Файл Правка Вид Терминал Справка
user@user:~/WORK/myproject$ /usr/local/Trolltech/QTx11-4.7.1-ARM/bin/qmake myproject.pro
user@user:~/WORK/myproject$
```

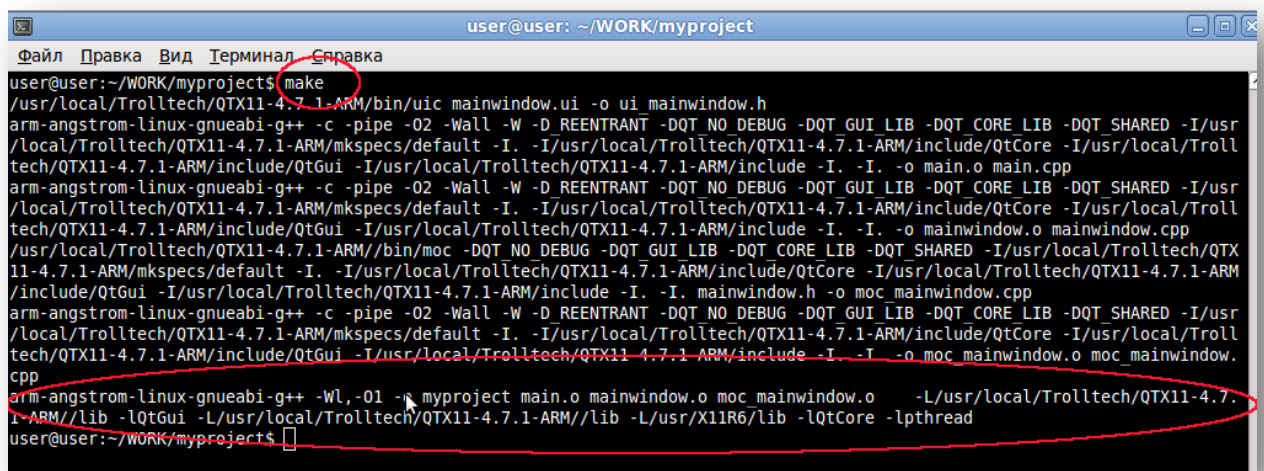
3. Теперь для компиляции необходимо указать путь до кросс-компилятора, как мы уже делали ранее командой

export PATH=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin:\$PATH



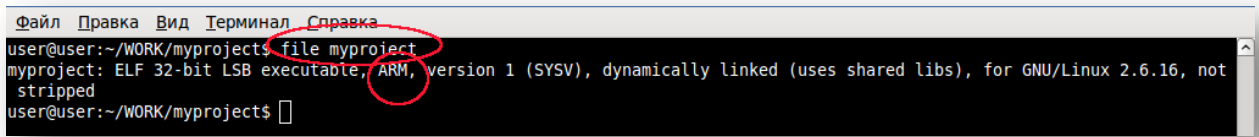
```
Файл Правка Вид Терминал Справка
user@user:~/WORK/myproject$ export PATH=/home/user/WORK/OE/stuff/tmp/cross/armv5te/bin:$PATH
user@user:~/WORK/myproject$
```

4. Запускаем команду ***make*** для сборки проекта.



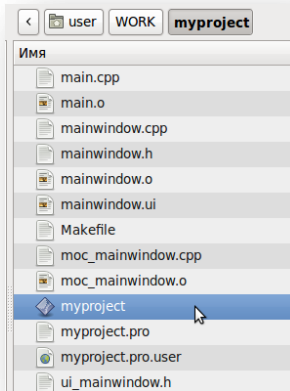
```
user@user: ~/WORK/myproject
Файл Правка Вид Терминал Справка
user@user:~/WORK/myproject$ make
/usr/local/Trolltech/QTx11-4.7.1-ARM/bin/uic mainwindow.ui -o ui_mainwindow.h
arm-angstrom-linux-gnueabi-g++ -c -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/local/Trolltech/QTx11-4.7.1-ARM/mkspecs/default -I. -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtCore -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtGui -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include -I. -I. -o main.o main.cpp
arm-angstrom-linux-gnueabi-g++ -c -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/local/Trolltech/QTx11-4.7.1-ARM/mkspecs/default -I. -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtCore -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtGui -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include -I. -I. -o mainwindow.o mainwindow.cpp
/usr/local/Trolltech/QTx11-4.7.1-ARM/bin/moc -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/local/Trolltech/QTx11-4.7.1-ARM/mkspecs/default -I. -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtCore -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtGui -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include -I. -I. mainwindow.h -o moc_mainwindow.cpp
arm-angstrom-linux-gnueabi-g++ -c -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/local/Trolltech/QTx11-4.7.1-ARM/mkspecs/default -I. -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtCore -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include/QtGui -I/usr/local/Trolltech/QTx11-4.7.1-ARM/include -I. -I. -o moc_mainwindow.o moc_mainwindow.cpp
arm-angstrom-linux-gnueabi-g++ -Wl,-O1 -o myproject main.o mainwindow.o moc_mainwindow.o -L/usr/local/Trolltech/QTx11-4.7.1-ARM/lib -lQtGui -L/usr/local/Trolltech/QTx11-4.7.1-ARM/lib -L/usr/X11R6/lib -lQtCore -lpthread
user@user:~/WORK/myproject$
```

5. В папке myproject появился исполняемый файл myproject. Можно проверить его свойства, введя в терминале команду **file myproject**



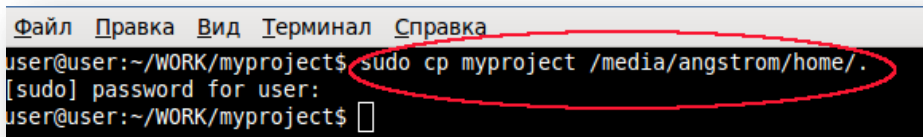
```
Файл Правка Вид Терминал Справка
user@user:~/WORK/myproject$ file myproject
myproject: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.16, not
stripped
user@user:~/WORK/myproject$
```

6. Теперь этот файл можно запускать непосредственно на нашей плате.

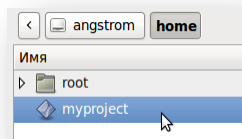


3.7.3 Перенос проекта и QT-библиотек в файловую систему микроконтроллера (на Flash-карту)

1. Вставьте Flash-карту с файловой системой, которую мы создавали на предыдущем этапе. Скопируйте файл myproject в папку /home на «флешке», набрав в терминале команду **sudo cp myproject /media/angstrom/home/**.

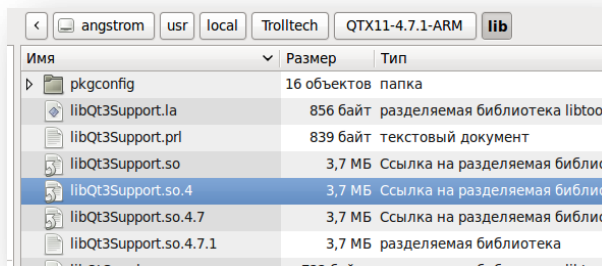


```
Файл Правка Вид Терминал Справка
user@user:~/WORK/myproject$ sudo cp myproject /media/angstrom/home/
[sudo] password for user:
user@user:~/WORK/myproject$
```



2. Кроме самого исполняемого файла необходимо скопировать библиотеки QT, на которые мы ссылались при компиляции, в то же место файловой системы микроконтроллера, где они хранятся в файловой системе ноутбука. Для этого нужно создать в файловой системе angstrom папку /usr/local, затем /usr/local/Trolltech, затем /usr/local/Trolltech/QTX11-4.7.1-ARM и скопировать в нее папку lib из папки /usr/local/Trolltech/QTX11-4.7.1-ARM/lib файловой системы ноутбука:

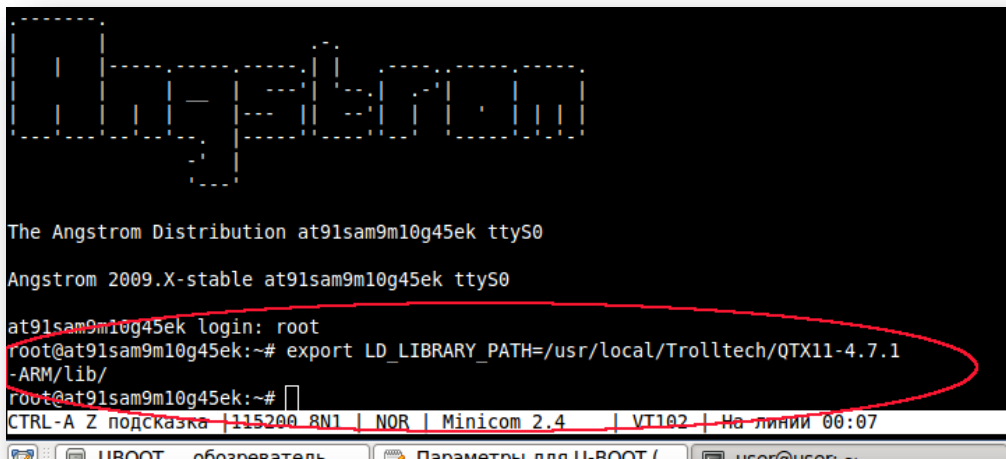
```
sudo mkdir /media/angstrom/usr/local/
sudo mkdir /media/angstrom/usr/local/Trolltech/
sudo mkdir /media/angstrom/usr/local/Trolltech/QTX11-4.7.1-ARM
sudo cp -R /usr/local/Trolltech/QTX11-4.7.1-ARM/lib /media/angstrom/usr/local/Trolltech/QTX11-4.7.1-ARM/.
```



3.7.4 Задание переменных окружения системы на микроконтроллере для запуска графического приложения

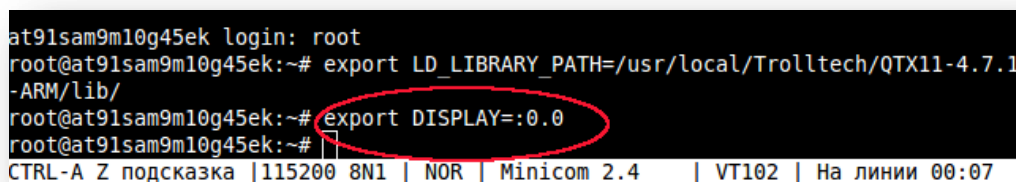
1. Безопасно извлеките Flash-карту из ноутбука и вставьте ее в USB-host порт отладочной платы. Включите питание платы и дождитесь загрузки. Войдите в систему.
2. Задайте переменную окружения LD_LIBRARY_PATH, указывающую путь до библиотек QT, набрав в терминале команду

export LD_LIBRARY_PATH=/usr/local/Trolltech/QT11-4.7.1-ARM/lib



3. Задайте переменную DISPLAY, необходимую для запуска приложений X11, набрав в терминале команду

export DISPLAY=:0.0



3.7.5 Запуск программы на микроконтроллере

1. Перейдите в папку /home и запустите приложение myproject, набрав в терминале последовательно команды ***cd /home*** и ***./myproject***. На дисплее отладочной платы Вы увидите Ваше приложение.

```
root@at91sam9m10g45ek:~# export DISPLAY=:0.0
root@at91sam9m10g45ek:~# cd /home/
root@at91sam9m10g45ek:/home# ./myproject
CTRL-A Z подсказка | 115200 8N1 | NOR | Minicom 2.4
```

Приложение

Ссылки на скачивание исходных текстов

✓ Openembedded

```
mkdir /home/user/WORK/OE/stuff  
cd /home/user/WORK/OE/stuff  
git clone git://git.openembedded.org/openembedded  
cd openembedded  
git checkout origin/stable/2009 -b stable/2009  
cd ..  
wget ftp://ftp.linux4sam.org/pub/oe/linux4sam_2.1/oe_at91sam.tgz  
tar zxvf oe_at91sam.tgz  
source oe_env.sh  
tar zxvf bitbake-1.8.18.tar.gz  
ln -s bitbake-1.8.18 bitbake
```

✓ Bootstrap

```
wget ftp://www.at91.com/pub/at91bootstrap/AT91Bootstrap1.16.zip
```

✓ U-Boot

```
wget ftp://ftp.denx.de/pub/u-boot/u-boot-1.3.4.tar.bz2  
wget ftp://www.at91.com/pub/uboot/u-boot-1.3.4-exp.5/u-boot-1.3.4-exp.5.diff  
wget http://www.linuxconsulting.ro/atmel/at91sam9q45/dual-memory-bank-support/uboot/0001-add-support-for-both-banks-of-memory-on-Atmel-G45-bo.patch
```

✓ Linux kernel

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.30.tar.bz2  
wget http://maxim.org.za/AT91RM9200/2.6/2.6.30-at91.patch.gz  
wget ftp://www.linux4sam.org/pub/linux/2.6.30-at91/2.6.30-at91-exp.4.tar.gz  
wget http://www.linuxconsulting.ro/atmel/at91sam9q45/dual-memory-bank-support/kernel/0001-Add-SPARSEMEM-support-for-Atmel-CPU-for-2-banks-of-m.patch  
wget http://www.linuxconsulting.ro/atmel/at91sam9q45/dual-memory-bank-support/kernel/0010-Fix-memory-mapping-for-SPARSEMEM-to-use-both-banks-o.patch  
wget ftp://www.at91.com/pub/linux/2.6.30-at91/at91sam9m10q45ek_defconfig
```

✓ QT-libraries

```
wget ftp://ftp.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.7.1.tar.gz
```

**Предложения, пожелания и вопросы
отправляют на zaa@rainbow.ur.ru**