РТУ МИРЭА (ТУ)

Центр проектирования интегральных схем, устройств наноэлектроники и микросистем

Методическое пособие по лабораторным работам

«Багет-ПЛК1-01»

(ОС Linux, Debian 10)

Москва

2023

Оглавление

Введение **2**

1 Установка ОС **9**

2 Ознакомительная **17**

3 Модуль ядра **25**

4 Драйвер GPIO **41**

5 Драйвер SPI устройства **54**

6 Драйвер I2C устройства **65**

7 Драйвер Serial устройства **74**

8 Web интерфейс **80**

Список литературы **88**

Введение

Методическое пособие описывает основные шаги по разработке программного обеспечения для встраиваемых систем. В качестве целевой платформы используется плата «Багет ПЛК1-01» с установленным микроконтроллером Комдив-МК К5500ВК018, производства ФГУ ФНЦ НИИСИ РАН. В состав К5500ВК018 входит суперскалярное 64-разрядное RISC микропроцессорное ядро с архитектурой КОМДИВ64 (MIPS64-совместимая), включая сопроцессор вещественной арифметики (IEEE754), кэш-память первого уровня (16+16 кБ), кэш-память второго уровня (128 кБ), буфер трансляции виртуальных адресов на 64 адреса, контроллер динамической памяти DDR3L, восемь таймеров-счетчиков, сторожевой таймер, часы реального времени, два контроллера Ethernet 10/100, контроллеры последовательных интерфейсов RS-232C (4 шт.), I2C (2 шт.), SPI (3 шт.), QSPI, CAN 2.0 (2 шт.), USB 2.0. Серийный выпуск начат в 2021 году.

ПЛК1-01 выполнен в виде печатной платы без корпуса, на которой расположены соединители для подключения первичного электропитания и внешних устройств, органы управления (пользовательские кнопки SW1, SW2), светодиодные индикаторы (VD1 – VD5), перемычки для управления режимами работы ПЛК1-01 (SA4, SA5, SA6, SA7, SA9).

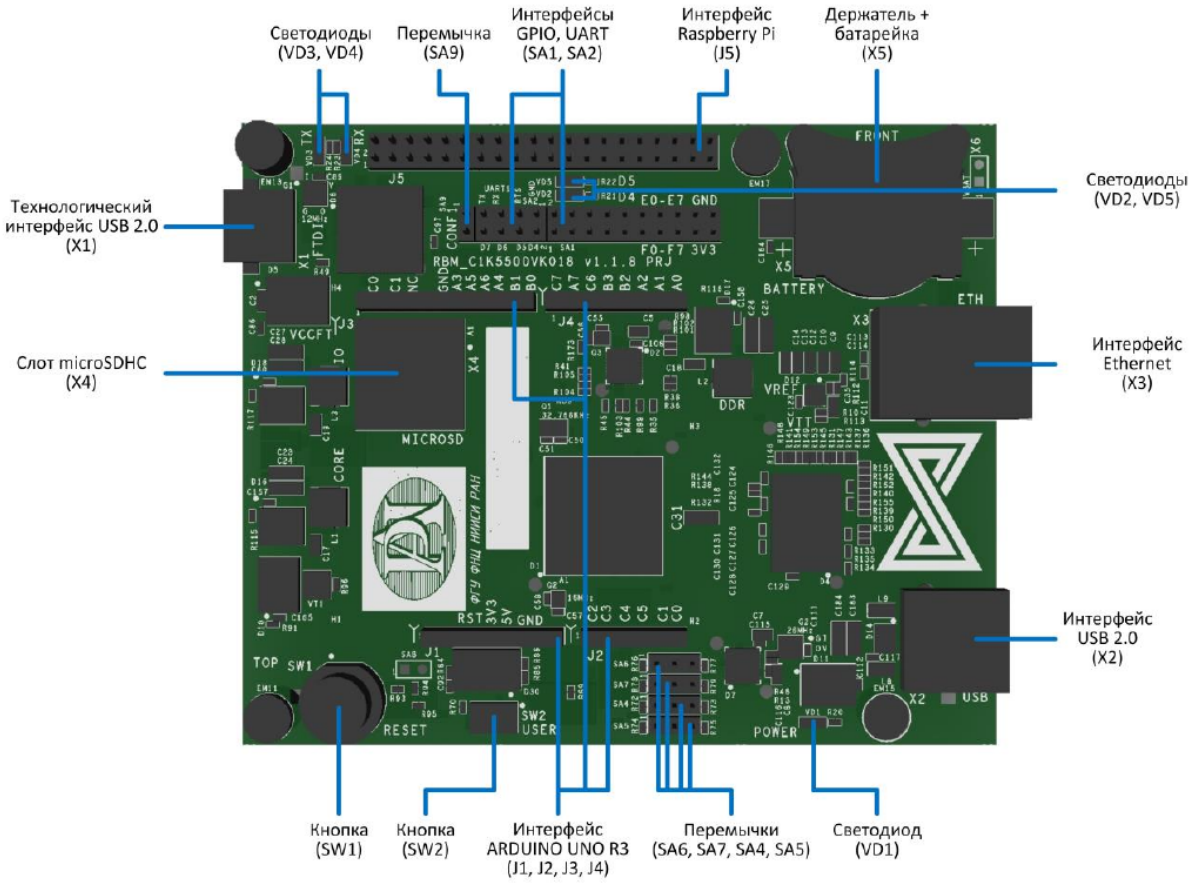


Рисунок 1. Расположение портов ввода-вывода

Назначение	Вывод процессора
Кнопка SW2	GPIOD 6
Зелёный светодиод VD2	GPIOD 4
Жёлтый светодиод VD5	GPIOD 5

Для организации питания часов реального времени (RTC) микроконтроллера К5500ВК018 при отсутствии основного питания предусмотрена установка автономного элемента питания – батарейки типа CR2032 +3 В. Расположение соединителей, органов управления и индикаторов ПЛК1-01 приведено на рисунке 1, назначение выводов на некоторых из разъёмов приведены на рисунке 2.

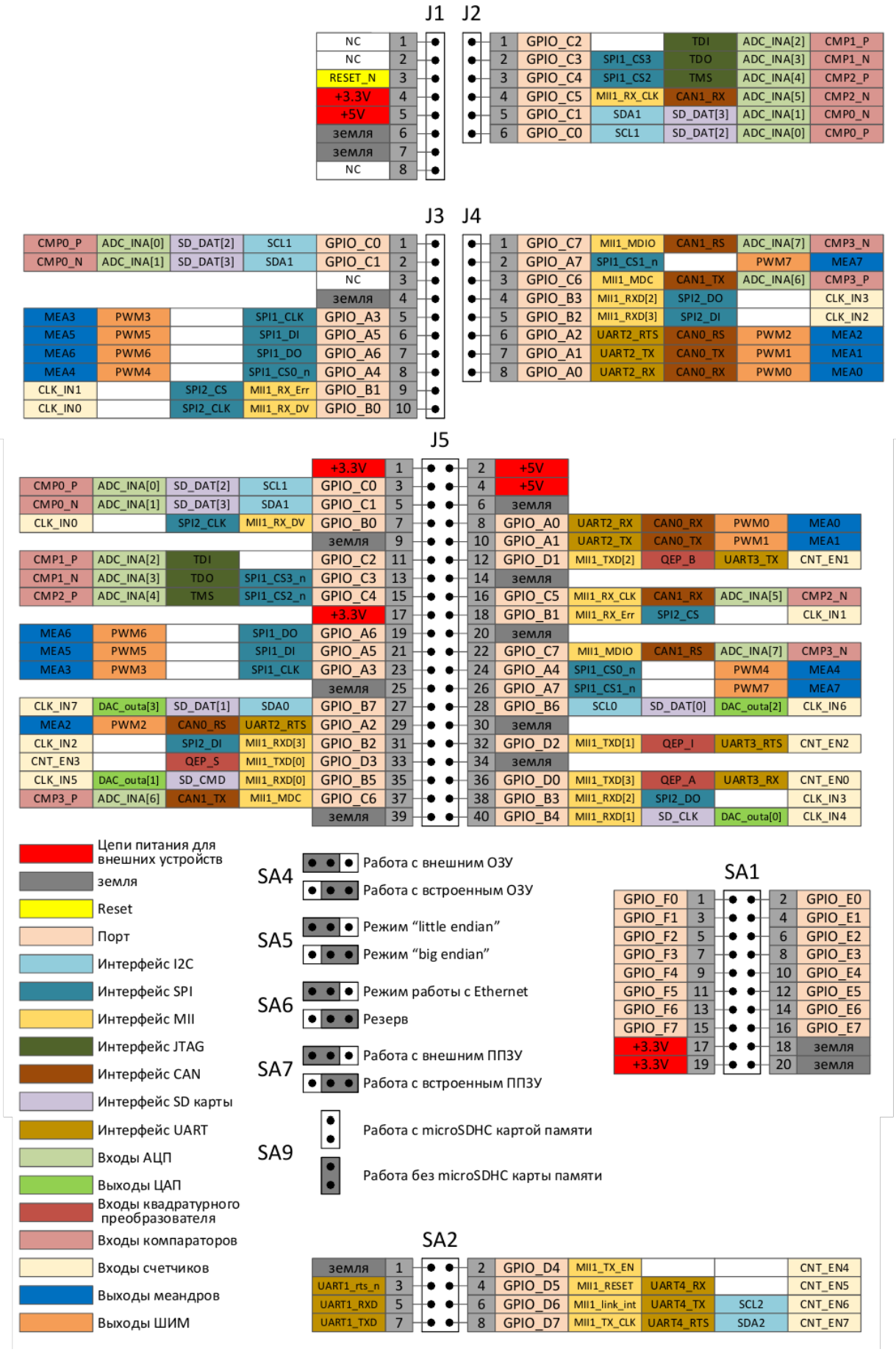


Рисунок 2. Разводка выводов процессора по разъёмам ввода-вывода

Переходная плата

Для удобства подключения к интерфейсам целевой платы, была разработана переходная плата, с выведенными сигнальными линиями, сгруппированными по интерфейсам (рисунок 3).

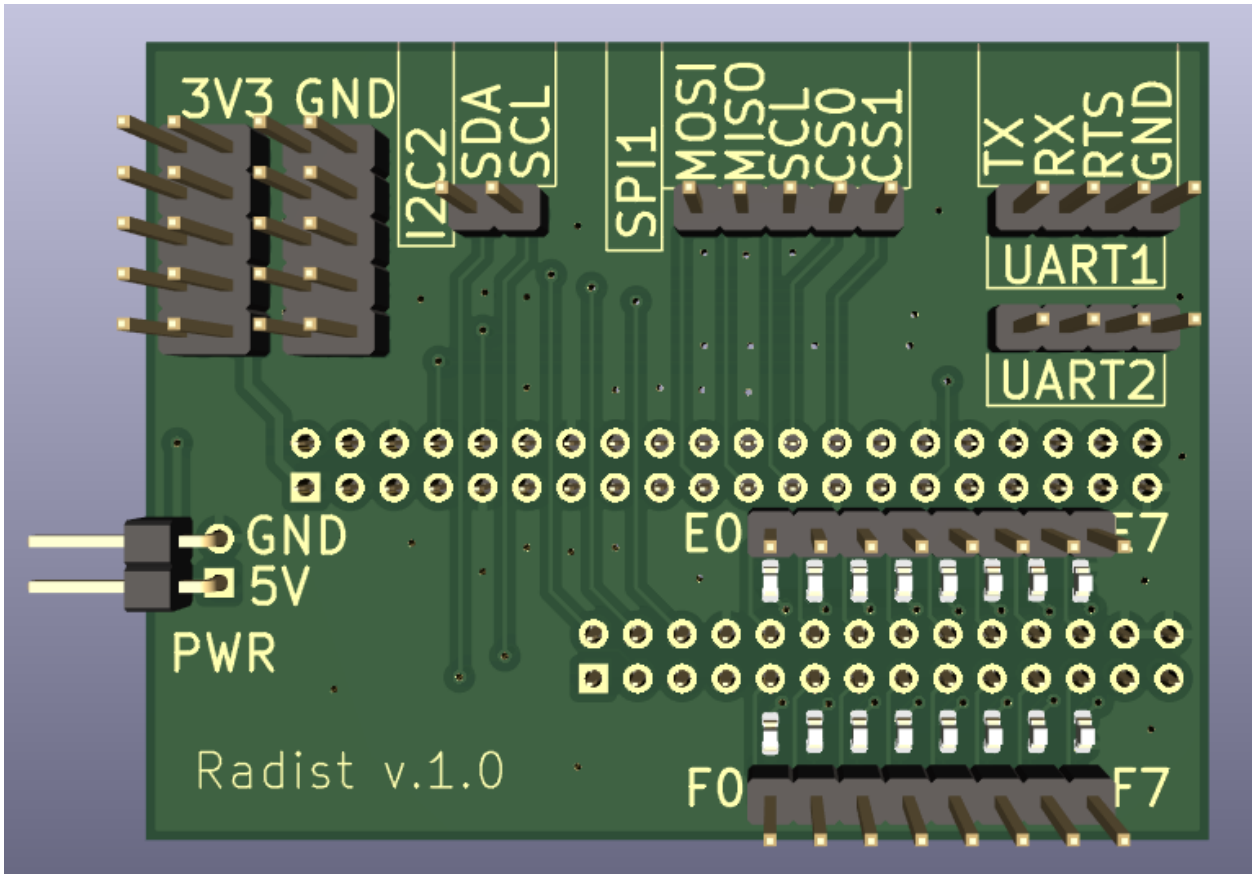


Рисунок 3. Переходная плата

UART2 можно использовать как для работы с интерфейсом UART2, так и для работы с интерфейсом CAN0. Выводы GPIO банков E и F дублируются на светодиоды для отладочных целей. При необходимости, можно подключать к выводам кнопки, или использовать их для формирования управляющих сигналов для модулей. Также обратите внимание, что одна из сигнальных линий I2C интерфейса используется кнопкой SW2 на плате, по этой причине не стоит использовать кнопку при активации этого нтерфейса.

Как запускается Linux?

Для запуска ОС Linux в общем случае должны пройти следующие этапы:

1. ROM BOOT — запуск встроенного загрузчика, определяющего источник дальнейшей загрузки. Этот код нельзя исправить его работа и формат образа (заголовки, значение бит и прочее) зависит от производителя.

2. First Stage Bootloader (FSBL) — загрузчик первой ступени, задача этого загрузчика, начальная инициализация процессора, и загрузка пользовательского кода в DDR память. Данный загрузчик опциональный, и у некоторых производителей может отсутствовать. Работает из встроенной RAM памяти процессора.

3. Second Stage Bootloader (SSBL) — это загрузчик второго уровня, задача которого заключается в подготовке системы к запуску ядра Linux, с пользовательскими параметрами (точка монтирования rootfs, скорость работы терминала для логов и прочее). Наиболее распространённый вариант загрузчика Das U-Boot (https://u-boot.readthedocs.io). В целевой плате используется другой загрузчик, набирающий популярность — Barebox (https://www.barebox.org/).

4. Kernel — на этом этапе за работу процессора отвечает ядро операционной системы, происходит финальная конфигурация, активация дополнительный процессорных ядер (если они есть и активирована функция многоядерной поддержки), монтирование rootfs, запуск сервисов и прочее.

Описание окружения

Для работы с платой используется виртуальная машина на базе Kubuntu 20.04. На виртуальной машине установлены пакеты для компиляции кода под Linux для MIPS64 архитектуры mips64el-linux-gnu-gcc (C компилятор) и mips64el-linux-gnu-g++ (C++ компилятор), компилятор дерева устройств device-tree-compiler, архив с заголовочными файлами ядра Linux установленного на плате, утилита debootstrap, IDE VSCode и прочие полезные инструменты.

Имя пользователя **student** пароль **usrstudent**. Пользователь поддерживает работу через sudo (получение прав суперпользователя он же администратор).

В конце некоторых работ есть задания для самостоятельного выполнения. При этом есть простые задачи, предполагающие незначительное изменение кода, а также есть помеченные *. Последние выполняются по желанию, при этом нужно быть готовым, что придётся активно пользоваться поиском, и расходовать клетки серого вещества.

ВНИМАНИЕ! Запомните, что далее по тексту, если просят выполнить команду, в начале которой написано **$** , значит команду нужно выполнить на целевой плате, а если с # - в консоли виртуальной машины. При этом $ или # писать не нужно!

Если команда заканчивается символом \ и на следующей строчке есть запись, то это запись относиться к одной команде, Вы можете записать как видите, так как символ \ в консоли Linux означает многострочную команду или, опустив символ \, продолжить вводить текст приведённый на следующей строке методички.

Примеры:

Авторы хотят, что бы Вы в консоли виртуальной машины (ВМ) запи-

сали echo Hello и нажали кнопку Enter:

```
# echo Hello
```

Авторы хотят, что бы Вы в консоли целевой платы записали echo FOOBAR и нажали кнопку Enter:

```
$ echo FOOBAR
```

Авторы хотят, что бы Вы в консоли виртуальной машины (ВМ) написали очень длинную команду (да-да, вечер не обещает быть томным) и нажали кнопку Enter:

```
# echo \
FOOBAR
```

Лабораторная работа № 1

Установка ОС

Цель: Установить дистрибутив Debian 10 на целевую платформу.

Описание: Нужно понимать, что ОС Linux делиться на две части. Ядро (Kernel) и файловую систему (rootfs). Ядро определяет какие возможности по взаимодействию с железом доступны пользователю и приложениям. Rootfs определяет какие инструменты есть в системе, порядок инициализации, запуска сервисов и прочие прикладные штуки. Дистрибутив это в большей степени наполнение rootfs. Не каждый дистрибутив имеет поддержку иной от x86 архитектур.

Для установки debian воспользуемся инструментом под названием debootstrap, который отвечает за создание rootfs. Подробнее про этот инструмент, и возможности по его управлению, Вы можете найти на wiki проекта Debian.

1.1 Подготовка rootfs

1.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

1.1.2 Запустите консоль нажав комбинацию клавиш на клавиатуре Ctrl+Alt+T

Внимание! В консоли есть возможность автоматического продолжения ввода. Для этого необходимо ввести первые символы команды, или пути, и нажать клавишу TAB. Ввод продолжиться до тех пор, пока не появиться неопределённостью (к примеру у Вас есть два файла foo_bar1 и foo_bar2, при нажатии TAB будет вставлен текст до цифры). Двойное нажатие TAB приведёт к выводу всех возможных вариантов продолжения, если таковые есть.

1.1.3 Создайте и перейдти в рабочий каталог в котором будет создан образ rootfs.

```
# mkdir -p $BAGET/lab_01
# cd $BAGET/lab_01
```

1.1.4 Запустите утилиту debootstrap (при необходимсоти введите пароль usrstudent):

```
# sudo debootstrap --include=aptitude,nano,wget \
--foreign \
--arch=mips64el buster rootfs
```

--include=A,B,C.. - добавить в сборку указанные пакеты

--foreign — только сгенерировать наполнение, применяется когда архитектура на которой запускается утилита отлична от архитектуры назначения.

--arch=mips64el — указываем целевую архитектуру

buster — версия сборки

rootfs — путь к папке назначения, где будут размещены файлы

1.1.5 Для продолжения установки, нам понадобиться утилита qemu позволяющая эмулировать различные архитектуры. Скопируем исполняемый

файл qemu:

```
# sudo cp /usr/bin/qemu-mips64el-static ./rootfs/usr/bin
```

и перейдём в созданную rootfs (привет дедушка контейнеров, chroot)

```
# sudo chroot ./rootfs
```

1.2 Настройка rootfs

1.2.1 Завершим работу debootstrap

```
# export LANG=en_US.UTF-8
# /debootstrap/debootstrap --second-stage
```

1.2.2 Добавим источники для установки ПО, для этого выполним следующие команды

```
# echo deb http://ftp.debian.org/debian buster \
main contrib non-free  >> /etc/apt/sources.list

# echo deb-src http://ftp.debian.org/debian buster \
main contrib non-free >> /etc/apt/sources.list

# echo deb http://ftp.debian.org/debian buster-updates \
main contrib non-free >> /etc/apt/sources.list

# echo deb-src http://ftp.debian.org/debian buster-updates \
main contrib non-free >> /etc/apt/sources.list
```

1.2.3 Обновим список, и установим ряд приложений

```
# apt-get update
# apt-get install -y dialog sudo less i2c-tools evtest mc \
openssh-server resolvconf hwinfo net-tools
```

1.2.4 Зададим пароль для root пользователя

```
# passwd root
```

После чего введите root (внимание, курсор двигаться не будет, это политика безопасности Linux, при вводе пароля курсор не перемещается, что бы нельзя было установить количество символов). И нажмите Enter

Затем Вас попросят повторить пароль, снова введите root и нажмите Enter

1.2.5 Настроим работу сетевого интерфейса со статическим адресом, для этого создадим файл в каталоге /etc/network/interfaces.d/

```
# nano /etc/network/interfaces.d/eth0
```

и впишите туда следующие строки:

```
auto eth0
   iface eth0 inet static
   address 192.168.100.200
   netmask 255.255.255.0
   network 192.168.100.0
```

Первая строка определяет когда настраивается интерфейс. Auto означает, что при загрузке ОС. Для USB адаптеров, лучше выбирать allow-hotplug, что бы сократить время загрузки ОС.

Вторая строка определяет, что мы будем использовать статический ip адрес, далее идёт конфигурация адресов.

Выходим из редактора nano, нажав комбинацию клавиш Ctrl+X. Вам будет задан вопрос о сохранении изменения в файле, жмём Y и клавишу Enter.

1.2.6 Настроим имя компьютера

```
# echo baget > /etc/hostname
```

1.2.7 Завершим настройку rootfs

```
# exit
# sudo rm ./rootfs/usr/bin/qemu-mips64el-static
```

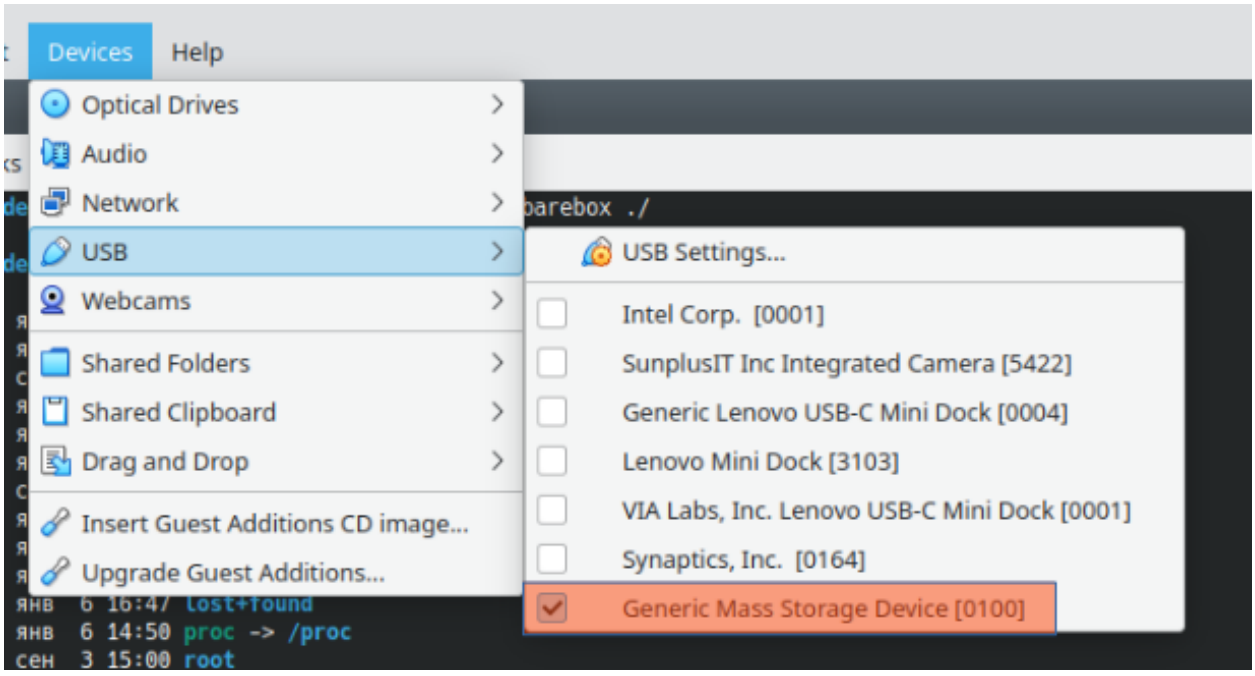
1.2.8 Скопируйте папку barebox из каталога support в корень созданной rootfs

```
# sudo cp -r $BAGET/support/barebox/ ./rootfs/
```

В данной папке лежит скомпилированное ядро Linux (vmlinux. . .), скомпилированное дерево устройств (k5500vk018_rbm.dtb) и скрипт выполняемый загрузчиком (barebox.sh).

1.3 Проверка

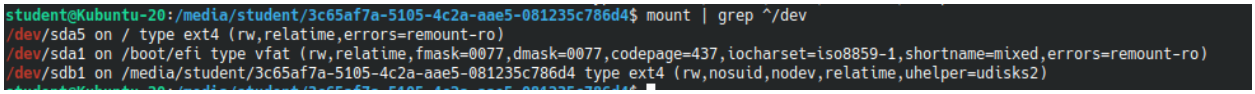
1.3.1 Вставьте SD карту в ПК, и подключите её к виртуальной машине



1.3.2 Необходимо узнать точку монтирования SD карты. Она может отличаться в зависимости от ридера, что Вы используете. Для упрощения ввода, выполните команду в консоле:

```
# mount | grep ^/dev
```

для получения списка примонтированных устройств, при этом командой grep происходит фильтрация лишнего вывода, и отображается только список блочных устройств. Вывод будет примерно как на рисунке



Первые две строчки, это разделы виртуального жёсткого диска, далее точка монтирования SD карты (/media/student/...)

Для упрощения ввода, создадим временную системную переменную, которой назначим путь, для примера(!) на рисунке выше:

```
# export BAGET_SD=\
/media/student/3c65af7a-5105-4c2a-aae5-081235c786d4
```

Нужно помнить, что эта переменная существует только в том терминале, где был вызван export, в остальных окнах терминала она не существует, если Вы закроете текущее окно, или Вам понадобиться эта переменная в другом окне, необходимо будет повторить процедуру присвоения.

1.3.3 Удалите все файлы с sd карты

```
# sudo rm -rf $BAGET_SD/*
```

скопируйте созданную rootfs на sd карту

```
# sudo cp -a $BAGET/lab_01/rootfs/* $BAGET_SD/
```

и отмантируйте SD карту

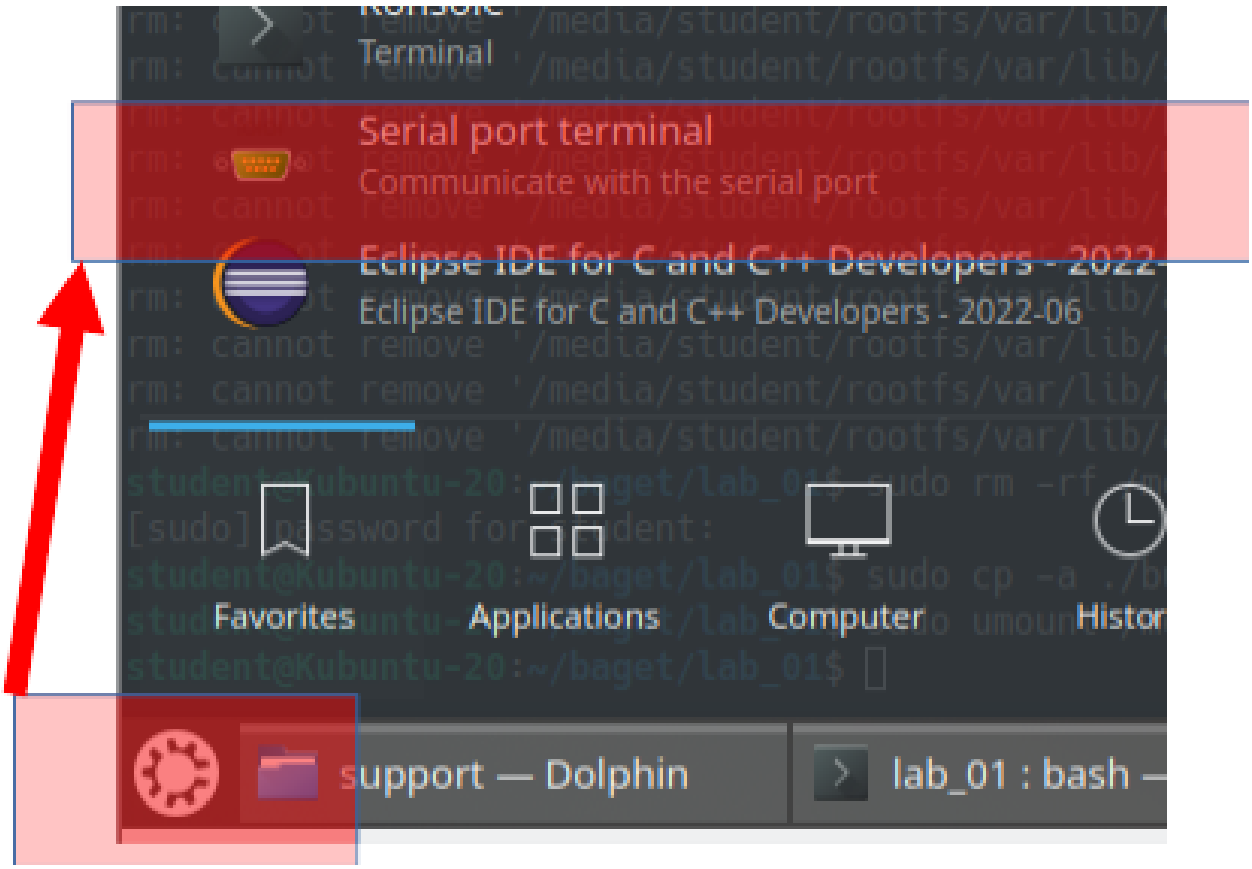
```
# sudo umount $BAGET_SD
```

(это важно, так как в ОС Linux запись данных на носители происходит в фоне, и если вы отключите накопитель «неправильно» то нет гарантии,

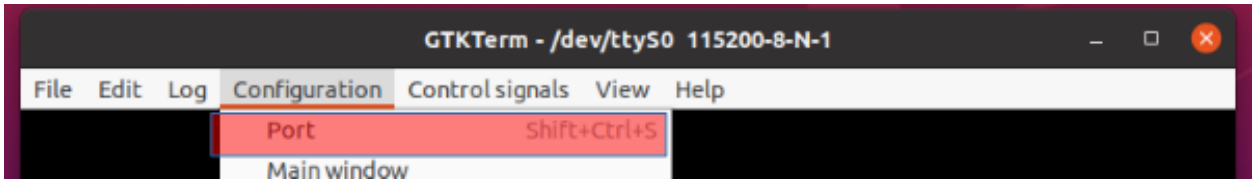
что данные успели записаться на носитель!). Дождитесь когда команда выполниться, после чего достаньте SD карту и вставьте её в плату.

1.3.4 Подключите USB шнуром плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

1.3.5 Запустите программу gtkterm или нажмите Ctrl + Shift + T для открытия новой вкладки в окне терминала и выполните команду minicom (это консольная утилита, в виртуальной машине эта утилита уже настроена на нужный порт, с нужными параметрами)



1.3.6 Выберите порт /dev/ttyUSB1 через меню Configuration->Port.



Нажмите кнопку OK.

Если в окне не появился текст, и на нажатие клавиши Enter так же нет реакции, нажмите на кнопку Reset на плате. Изучите вывод, там будет написано, что пошло не так.

Один из вариантов, когда при записи файлов на SD карту что-то побилось. В этом случае, вы увидите много записей как в примере ниже:

```
srisa-sdhci 1b50b000.sdhci@1b50b000.of: registered as mci0
srisa-sdhci 1b50b000.sdhci@1b50b000.of: SDHCI timeout while waiting for
   done
srisa-sdhci 1b50b000.sdhci@1b50b000.of: error on command 8
srisa-sdhci 1b50b000.sdhci@1b50b000.of: state = 0400 03ff, interrupt =
   8001 0001
srisa-sdhci 1b50b000.sdhci@1b50b000.of: r0 00000000 r1 00000000 r2
   00000000 r3 00000000
```

Нужно повторить все шаги этого раздела. **При вынимании SD карты убедитесь, что плата обесточена!**

1.3.7 Введите в качестве логина и пароля root

Поздравляем, Вы запустили Debian дистрибутив на плате.

1.3.8 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

Лабораторная работа № 2

Ознакомительная

Цель: Запустить плату, подключиться к ней через консоль и по сетевому интерфейсу, работа с кнопкой и светодиодами через sysfs.

Описание: работа с ОС Linux на целевой платформе мало чем отличается от работы с данной ОС на обычном компьютере. Среди особенностей можно выделить наличие меньшего набора инструментов, и рабочих программ, что делается для экономии свободного места, и ускорения загрузки ОС. Так же может отсутствовать привычный рабочий стол, а вся работа сводиться к взаимодействию через рабочий терминал, она же консоль. Для подключения к консоли можно воспользоваться интерфейсом UART (он же COM порт, или последовательный порт), или при помощи сетевого соединения (ssh или реже telnet).

При этом нужно понимать, что возможность работы через консоль настраивается, и не всегда может быть доступна по-умолчанию (обычно возможность сетевого подключения отключают, для повышения безопасности).

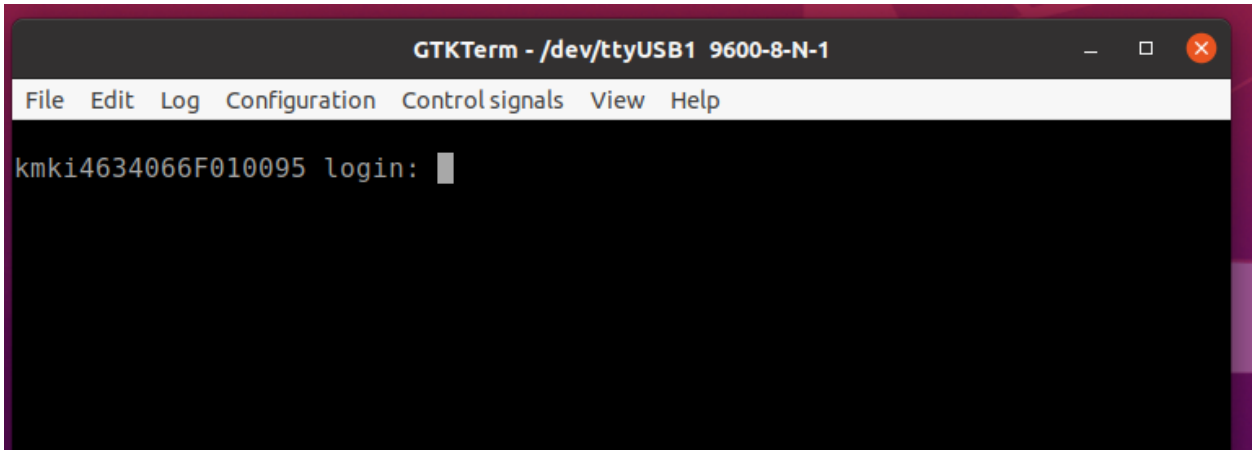
2.1 Запуск и подключение к устройству

2.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

2.1.2 Подключите по USB плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

2.1.3 Откройте программу gtkterm, и подключитесь к порту /dev/ttyUSB1

2.1.4 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. Вы должны увидеть следующий вывод:



Вы подключились к консоли устройства. Введите логин root и пароль root.

2.2 Подключение по сетевому интерфейсу

Рассмотрим какие шаги необходимо выполнить, для того, что бы можно было организовать сетевое подключение к плате, для получение доступа к консоли, и файловой системе через SSH соединение. Так же настройки, которые будут выполнены в данном разделе помогут в дальнейшем копировать файлы на плату через команду scp.

2.2.1 Проверьте текущие настройки интерфейса eth0 выполнив команду:

```
$ ip addr show dev eth0
```

Вы должны увидеть, что выдаче появилась строчка:

```
   ...
inet 192.168.100.200/24 scope global eth0
   ...
```

Если inet адрес отличается, то удалите текущий адрес (вместо <inet_val> впишите адрес, который отобразился у вас)

```
$ ip addr del <inet_val> dev eth0
```

затем установите новый адрес командой

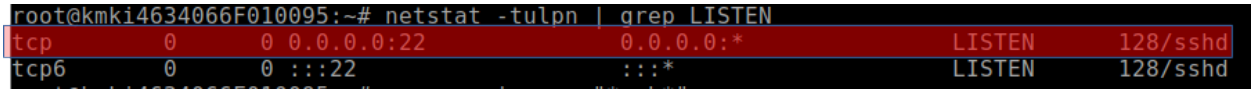
```
$ ip addr add 192.168.100.200/24 dev eth0
```

Для активации изменения необходимо переподключиться, для чего выполните следующие команды

```
$ ip link set eth0 down
$ ip link set eth0 up
```

2.2.2 Проверьте, что на плате запущен SSH сервер. Проверить открытые порты устройства, выполнив команду:

```
$ netstat -tulpn | grep LISTEN
```



Если вы не увидели, вывода, как на картинках выше, то выполните команду:

```
$ systemctl start sshd
```

и проверьте снова.

2.2.3 Заходить через сеть в пользователя root не безопасно, поэтому создадим пользователя для дальнейшей работы, для этого введите следующие команды (пароль для пользователя netuser - usrnetuser):

```
$ useradd -s /bin/bash -m netuser
$ passwd netuser
```

Первая команда для создания нового пользователя netuser с созданием домашнего каталога (/home/netuser) и также назначаем для него использовать интерпретатора bash.

2.2.4 Подключите сетевой шнур к рабочему ПК и плате

2.2.5 Откройте на виртуальной машине консоль, нажав на клавиатуре комбинацию клавиш Ctrl + Alt + T.

2.2.6 В открывшемся окне терминала введите команду:

```
# ssh netuser@192.168.100.200
```

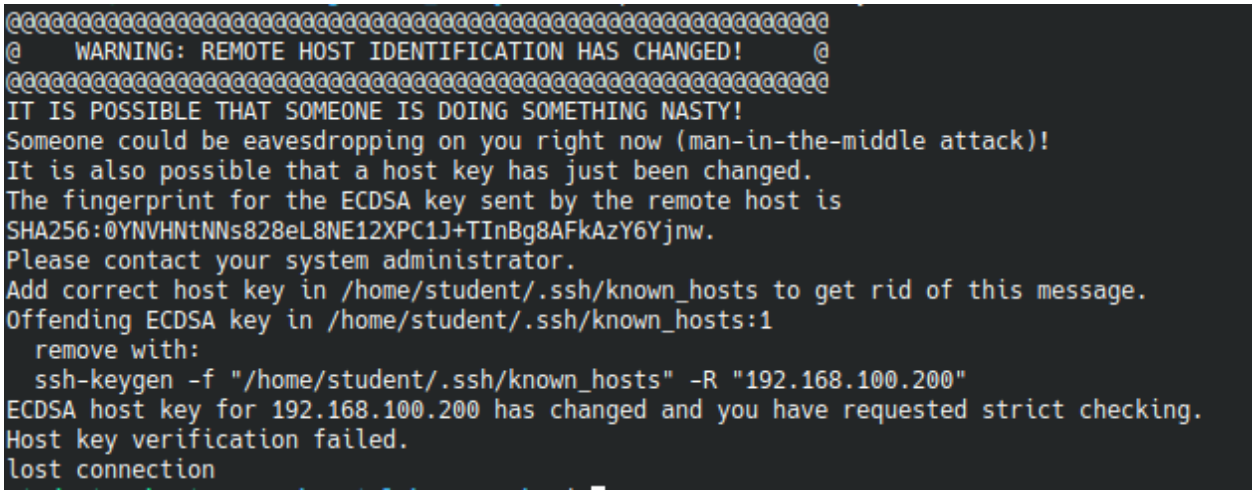
2.2.7 Вам предложат ввести пароль пользователя, вводим тот, что установили в пункте 2.2.3 и получаем доступ к терминалу платы.

Если вместо ввода пароля, Вы увидете надпись похожую на:

```
The authenticity of host '192.168.100.200 (192.168.100.200)' can't be
   established.
ECDSA key fingerprint is
   SHA256:+OSEJhr2dIiTvlWZxq5fZ0UFdYYY+egbTgabe6F7zZE.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Введите yes и нажмите Enter, и Вам предложат ввести пароль.

Если же получите вывод как на картинке ниже



Выполните команду

```
$ ssh-keygen -f "/home/student/.ssh/known_hosts" -R "192.168.100.200"
```

и вернитесь к пункту 2.2.6

2.2.8 Завершите сеанс, введя команду exit

2.3 Аппаратный Hello world

В этой части Вы будете управлять выводами микроконтроллера, для переключения светодиодов и считывания состояния кнопки, при помощи виртуальной файловой системы sysfs.

Для работы с периферией в ОС Linux помимо прочего есть две виртуальные файловые системы: procfs (точка входа /proc) и sysfs (точка входа /sys).

Sysfs экспортирует в пространство пользователя информацию ядра Linux о присутствующих в системе устройствах и драйверах, что позволяет не только считывать значение (например с датчиков температуры), но и менять поведение, если это было заложено разработчиками.

За формирование наполнения виртуальных файловых систем отвечает модуль ядра, который выполняет роль драйвера, если речь идёт про

взаимодействие с периферийным модулями (подробно будет рассмотрен в следующих лабораторных работах).

2.3.1 Перейдите в каталог /sys/class/gpio:

```
$ cd /sys/class/gpio
```

2.3.2 Выведите список фалов и каталогов в текущей директории:

```
$ ls -l
```

```
root@kmki4634066F010095:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Feb 14 10:12 export
lrwxrwxrwx 1 root root    0 Feb 14 10:12 gpiochip464 -> ../../devices/platform/1b400480.gpio/gpio/gpiochip464
lrwxrwxrwx 1 root root    0 Feb 14 10:12 gpiochip472 -> ../../devices/platform/1b400400.gpio/gpio/gpiochip472
lrwxrwxrwx 1 root root    0 Feb 14 10:12 gpiochip480 -> ../../devices/platform/1b400380.gpio/gpio/gpiochip480
lrwxrwxrwx 1 root root    0 Feb 14 10:12 gpiochip488 -> ../../devices/platform/1b400300.gpio/gpio/gpiochip488
lrwxrwxrwx 1 root root    0 Feb 14 10:12 gpiochip496 -> ../../devices/platform/1b400280.gpio/gpio/gpiochip496
lrwxrwxrwx 1 root root    0 Feb 14 10:12 gpiochip504 -> ../../devices/platform/1b400200.gpio/gpio/gpiochip504
--w------- 1 root root 4096 Feb 14 10:12 unexport
```

Разберём то, что Вы видите. Первый и последний файлы используются для того, что бы подключать драйвер для управления отдельным выводом в файловую систему sysfs или отключать.

Далее идёт серия символьных ссылок на каталоги, каждый из которых описывает один банк сигналов ввода-вывода общего назначения (GPIO). Тут нужно отметить, что в рабочем чипе присутствует 6 банков (A, B, C, D, E и F), по 8 выводов в каждом, по этой причине мы видим 6 каталогов.

Обратите внимание на имена каталогов, внутри platform. Первая часть имени является базовым адресом периферийного модуля, отвечающего за управление банком выводов. Первому банку выводов A соответствует адрес 0x1B400200, остальные по возрастанию.

2.3.3 У выводов общего назначения в рамках ядра Linux нумерация сквозная. Таким образом, можно определить, что вывод подключённый к кнопке SW2 находиться под номерам 486. Выполним следующую команду:

```
$ echo 486 > export
```

2.3.4 Выполните эту же команду для экспорта выводов пары выводов банка F (504 и 505)

2.3.5 После каждой команды будет создан каталог ./gpioXXX где вместо XXX будет номер вывода. Если зайти в любой из этих каталогов, то можно увидеть там одинаковый набор файлов, но нас интересуют следующие:

- direction — задаёт направление вывода (in вывод настроен как вход, out — как выход)
- value — если вывод настроен как вход, то читая этот файл можем узнать уровень логического сигнала на выводе. Если настроен на выход, то запись в этот файл 0 или 1 будет устанавливать соответствующий уровень выходного сигнала.

2.3.6 Так как два вывода банка F управляют светодиодами, изменим значение в файле direction для них на out

```
$ echo out > gpio504/direction
$ echo out > gpio505/direction
```

2.3.7 Переключать светодиоды можно записью 0 или 1 в файл value соответствующего вывода

```
$ echo 1 > gpio504/value
$ echo 0 > gpio504/value
```

2.3.8 Для взаимодействия с кнопкой менять ничего не нужно. Для удобства отладки запустить считывание файла через утилиту watch позволяющую выполнять заданную команду через указанный интервал времени.

```
$ watch -n 1 cat gpio486/value
```

После чего можете нажимать кнопку, и смотреть как меняется значение считанное из файла. В данном примере файл будет считываться каждую секунду (из-за параметра -n 1 переданного при запуске утилите watch).

2.3.9 Остановите работу программы watch комбинацией клавиш Ctrl + C

2.3.10 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

2.4 Задание для закрепления

Экспортируйте все выводы банков F и E, и включите все светодиоды.

* Написать bash скрипт, который будет делать экспорт нужных выводов, если необходимо, и в зависимости от того, нажата кнопка или нет поочерёдно включать или выключать все светодиоды.

Лабораторная работа № 3

Модуль ядра

Цель: Сборка модуля ядра, работа с device tree.

Описание:Для начала нужно понять, что в ОС Linux выделяется два пространства: пространство ядра (Kernel Space) и пользовательское пространство (User Space).

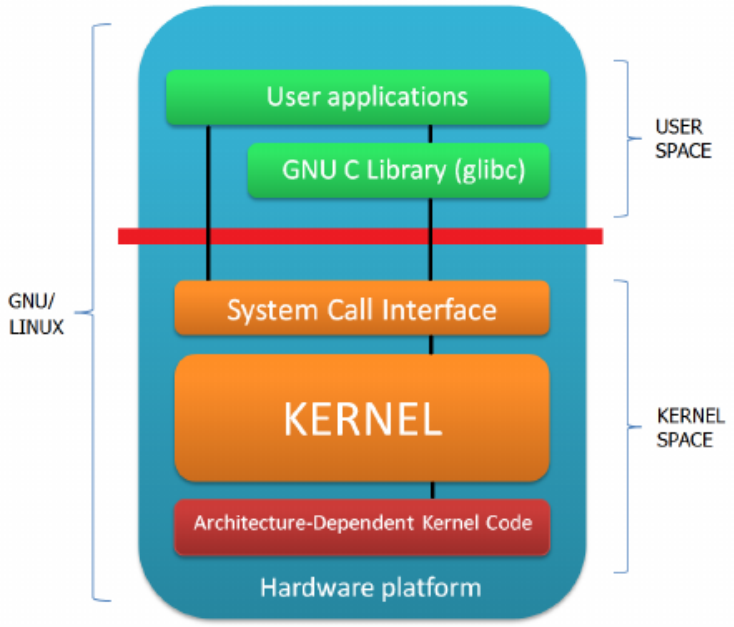
Эти два пространства разделены между собой, и взаимодействие может происходить только при помощи специального системного интерфейса (System Call Interface).

Полезные ссылки:

- Linux Kernel Labs: Character device drivers.
- DiGi: Use Device Tree Overlays to Patch Your Device Tree.

В пространстве ядра работает, как не удивительно, само ядро ОС, а так же его модули (которые могут быть встроены в основное ядро, или могут быть представлены файлами в rootfs с расширением ko).

Пользовательские приложения и библиотеки работают в пользовательском пространстве.



Ядро ОС Linux умеет работать с подгружаемыми модулями (если эта опция была активирована при сборке ядра). В качестве отдельных модулей могут выступать как определённые сервисы, предоставляемые ядром, так и драйверы устройств.

В Linux выделяют три типа устройств:

- символьное устройство — представлено в файловой системой, минимальный объём данных один символ (или байт), примеры: клавиатура, мышь, принтер, тачскрин, экран, камера, spi устройство, i2c устройство и т.д.
- блочное устройство — представлено в файловой системе, минимальный объём это блок данных, работа ведётся путём монтирования устройства. Примеры: карты памяти, SD карты и пр.
- пакетное устройство — не представлено в файловой системе, устройство для взаимодействия пакетами, примеры: сетевой интерфейс, Wi-Fi, Bluetooth и т.д.

В данной работе, мы создадим шаблон для символьного устройства, с

которым можно будет взаимодействовать через пользовательское приложение.

3.1 Сборка модуля

3.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

3.1.2 Подключите по USB плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

3.1.3 Откройте консоль комбинацией клавиш Ctrl+Alt+T

3.1.4 Создайте рабочий каталог

```
# mkdir -p $BAGET/lab_03
```

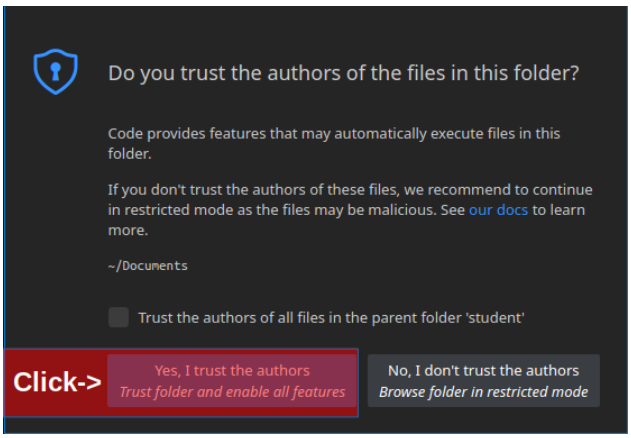
3.1.5 Скопируйте в него рабочие файлы

```
# cp -r $BAGET/support/mychar $BAGET/lab_03/
```

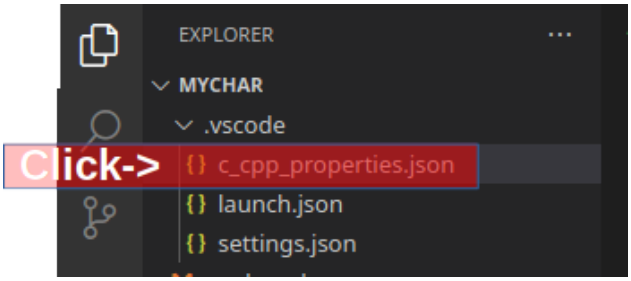
3.1.6 Перейдите в рабочий каталог и запустите vscode

```
# cd $BAGET/lab_03/mychar; code .
```

Замечание: При первом входе, Вас могут спросить, доверяете ли вы автору, нужно нажать на кнопку Yes,



3.1.7 Добавим путей для разрешения части include директив. Откройте .vscode -> c_cpp_properties.json



3.1.8 Допишите в поле includePath следующие строки:

```
"configurations": [
  {
    "name": "linux-gcc-x64",
    "includePath": [
      "${workspaceFolder}/**",
      "/home/student/baget/support/linux-sources-4.15/include/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/uapi/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/generated/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/asm/mach-baget",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/asm/mach-generic"
    ],
```

для удобства, воспользуйтесь файлом vscode_lines.md в папке support.

Ctrl+S для сохранения

3.1.9 Откройте Makefile. Разберём основные моменты

obj-m - тут мы задаём имя объектного файла верхнего уровня (файла из которого по итогу будет собран модуль).

chardev-y — описываем, из каких объектных файлов будет собираться итоговый.

KERNEL_SRC — тут мы создаём переменную, которая будет хранить путь к заголовочным файлам ядра.

SRC — в этой переменной мы записываем путь к исходникам модуля, через вызов утилиты pwd. Таким образом, можно рабочую папку переносить куда угодно, без ручных правок. При сборке модуля, текущий путь меняется на путь KERNEL_SRC, поэтому путь к исходникам модуля ядра должен быть абослютным.

all: — список команд выполняемых при сборке

clean: — список команд выполняемых при очистке

3.1.10 Посмотрите на содержимое файла mychar_init.c. Разберём основные моменты

В данном файле содержится минимальный набор кода, необходимый для сборки модуля ядра, и для его функционирования в системе. Нужно помнить, что данный код выполняется в пространстве ядра (Kernel space), соответственно не все парадигмы и функции привычные по работе с приложениями тут будут работать, хотя большинство из них имеют тут аналоги.

Обратите внимание на структур в районе 36 строки. Эта структура считывается ядром, при обращении к модулю, для установления связи между командами, отправляемыми через системный интерфейс, и функциями, которые их будут исполнять. Таким образом, достигается возможность исключить коллизий при работе с группой различных ядер. По той же причини, все структуры и функции объявлены с ключевым словом static.

Следующий момент, это структура объявляемая со строки 25. Параметр .compatible используется ядром, при сканировании дерева устройств. И при наличии совпадения, автоматически загружает модуль ядра.

3.1.11 Посмотрите на содержимое файла mychar_dev.c. В этом фале описан процесс инициализации символьного устройства, а так же работа с основными операциями (открытие файла, закрытие файла, считывание и запись данных). Обратите внимание на функции работы с данными (srisa_pdrv_read и srisa_pdrv_write), для обмена данными между user space и kernel space используются специальные функции copy_to_user и copy_from_user.

3.1.12 Если кликать мышкой на вызываемую функцию, при зажатой клавише Ctrl, то редактор откроет вам реализацию этой функции в отдельной вкладке (если реализация находиться в другом файле). Попробуйте, так как далее Вам этот функционал понадобиться, при выполнении самостоятельных работ.

3.1.13 Перейдём в терминал (можно открыть вкладку TERMINAL в vscode, если её не видно, выберите в меню Terminal -> New Terminal) и соберём модуль командой

```
# ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make
```

3.1.14 Вы должны получить следующий вывод

```
student@Kubuntu-20:~/baget/lab_03/mychar$ ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make
make -C /home/student/baget/support/linux-sources-4.15 M=/home/student/baget/lab_03/mychar/
make[1]: Entering directory '/home/student/baget/support/linux-sources-4.15'
  CC [M]  /home/student/baget/lab_03/mychar//mychar.o
In file included from /home/student/baget/lab_03/mychar//mychar.c:1:
./include/linux/module.h:137:7: warning: 'cleanup_module' specifies less restrictive attribute than its target 'srisa_pdrv_exit': 'cold' [-Wmissing-att
  137 |  void cleanup_module(void) __attribute__((alias(#exitfn)));
      |       ^~~~~~~~~~~~~~
/home/student/baget/lab_03/mychar//mychar.c:74:1: note: in expansion of macro 'module_exit'
   74 | module_exit(srisa_pdrv_exit);
      | ^~~~~~~~~~~
/home/student/baget/lab_03/mychar//mychar.c:70:20: note: 'cleanup_module' target declared here
   70 | static void __exit srisa_pdrv_exit(void)
      |                    ^~~~~~~~~~~~~~~~
In file included from /home/student/baget/lab_03/mychar//mychar.c:1:
./include/linux/module.h:131:6: warning: 'init_module' specifies less restrictive attribute than its target 'srisa_pdrv_init': 'cold' [-Wmissing-attrib
  131 |  int init_module(void) __attribute__((alias(#initfn)));
      |      ^~~~~~~~~~~
/home/student/baget/lab_03/mychar//mychar.c:64:1: note: in expansion of macro 'module_init'
   64 | module_init(srisa_pdrv_init);
      | ^~~~~~~~~~~
/home/student/baget/lab_03/mychar//mychar.c:60:19: note: 'init_module' target declared here
   60 | static int __init srisa_pdrv_init(void)
      |                   ^~~~~~~~~~~~~~~
  LD [M]  /home/student/baget/lab_03/mychar//chardev.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/student/baget/lab_03/mychar//chardev.ko
make[1]: Leaving directory '/home/student/baget/support/linux-sources-4.15'
student@Kubuntu-20:~/baget/lab_03/mychar$
```

А в рабочей папке должны появиться несколько файлов, в том числе chardev.ko

3.2 Настройка платы

Для работы с модулями в Linux есть ряд команд

insmod <path_to_module> - подключение модуля ядра, с явным указанием пути к файлу, при этом не проверяются зависимости (бывает что один модуль требует предварительного запуска другого модуля).

modporbe <module_name> - это «умная» версия insmod, проверяет зависимости (файл modules.dep), и подгружает их при необходимости. Не требуется указывать полный путь к модулю, однако модуль должен находиться в папке /lib/modules/<kernel_version>/

lsmod — выводит список активированных модулей, с указанием счётчика зависимостей (показывает сколько потоков использует данный модуль)

rmmod <module_name> - выгрузить указанный модуль (указывается только имя модуля, без расширения ko)

3.2.1 Откройте gtkterm и подключитесь к плате (/dev/ttyUSB1 скорость 115200)

3.2.2 Создадим каталог, для нормальной работы утилиты modporbe. Для этого выполните команду

```
$ mkdir -p /lib/modules/$(uname -r)/
```

3.2.3 Создадим два пустых файла

```
$ touch  /lib/modules/$(uname -r)/modules.order
$ touch  /lib/modules/$(uname -r)/modules.builtin
```

В данном случае, допустим такой ход, но в общем случае, содержимое этих файлов должно отражать реальность.

3.2.4 Вернёмся в консоль виртуальной машины, и скопируем скомпилированный модуль на плату (проверьте, что плата подключена сетевым шнуром к рабочему ПК)

```
# scp $BAGET/lab_03/mychar/chardev.ko \
netuser@192.168.100.200:/home/netuser/
```

пароль - usrnetuser

3.2.5 Вернитесь в терминал платы, и скопируйте ядро

```
$ cp /home/netuser/chardev.ko /lib/modules/$(uname -r)/
```

3.2.6 Определим зависимости для модуля, для чего выполните команду

```
$ depmod
```

Данная команда просканирует все модули, в папке /lib/modules/$(uname -r)/ и запишет о них информацию. После чего модули могут подгружаться командой modprobe.

3.2.7 Проверим работу нашего модуля, выполнив команду

```
$ modprobe chardev
```

Вы должны увидеть надпись

```
srisa_pdrv: srisa_pdrv_init
```

3.2.8 Просмотрим список загруженных модулей

```
$ lsmod
```

Вы должны увидеть надпись

```
Module                  Size  Used by
chardev                 4620  0
```

Как мы видим, никто не использует наше ядро (Used by равен 0).

3.2.9 Выполните команду ls /dev/* Обратите внимание, что отсутствует файл srisa_pdrv, так как он создаётся, только при вызове функции pdrv_probe, вызываемая только при наличии совместимого периферийного модуля.

3.2.10 Выгрузим из памяти наше ядро

```
$ rmmod chardev
```

3.3 Правка дерева устройств (device-tree)

Мы успешно собрали модуль устройства, однако его сейчас никто не использует. Как говорилось выше, часто модули ядра являются драйвером, для взаимодействия с периферией (как внутрипроцессорной, типа интерфейса spi, так и внешней). Для описания взаимодействия между ядром я устройством служит специальная структура дерево устройств (device tree). В ней описывается основная информация о устройстве, а так же могут добавлять параметры, применяемые модулем ядра, при работе с тем или иным устройством. Среди прочего, дерево устройств поддерживает функцию перезаписи, что позволяет создавать мелкие файлы, вносящие исправление в существующее описание, или дополняющее его, в соответствии с требованиями конкретного проекта.

3.3.1 Создадим рабочую папку в виртуальной машине для дополнения дерева устройств и перейдём в неё

```
# mkdir $BAGET/lab_03/devtree
# cd $BAGET/lab_03/devtree
```

3.3.2 Создадим файл и откроем его для редактирования

```
# kate of_mychard.dts
```

3.3.3 Запишите в файл следующие строки

```
/dts-v1/;

/ {
   fragment@0 {
      target-path = "/";
      __overlay__ {

         pdrv1 {
            compatible = "srisa,platform-device";
            status = "okay";
         };
      };
   };
};
```

3.3.4 Сохраните Ctrl+S и закройте редактор Ctrl+Q

Мы создали дополнение, которое описывает виртуальное устройство, совместимое с драйвером, который мы создали ранее (обратите внимание на поле compatible в написанном фрагменте, и сравните его значение со значением одноименного параметра в исходном коде модуля).

3.3.5 Скомпилируем дополнение к дереву устройств, для этого выполним следующую команду

```
# dtc -@ -O dtb -o ./of_mychard.dtb ./of_mychard.dts
```

Если у Вас есть уже скомпилированное дерево устройств, Вы можете вернуть его в «человеческий вид» командой

```
# dtc --symbol -O dts -o ./out.dts ./source.dtb
```

3.3.6 Скопируйте полученный файл на плату

```
# scp $BAGET/lab_03/devtree/of_mychard.dtb \
netuser@192.168.100.200:/home/netuser/
```

3.3.7 Перейдите в терминал платы, и переместите скопированный файл в папку barebox

```
$ mv /home/netuser/of_mychard.dtb /barebox/
```

3.3.8 Отредактируйте скрипт barebox.sh

```
$ nano /barebox/barebox.sh
```

и впишите в него, после строки DTB=k5500vk018_rbm.dtb

```
fdt_apply -i $DTB -l of_mychard.dtb -o /dtb && DTB=/dtb
```

Ctrl+S, Ctrl+X

3.3.9 Перезагрузите плату командой reboot

3.4 Проверка работы модуля

3.4.1 Войдите как пользователь root, и выполните команду lsmod. Модуль должен быть уже загружен, так как ядро при запуске должно было обнаружить в дереве устройств, совместимое с нашим модулем.

3.4.2 Посмотрим последние строки лога загрузки ядра

```
$ dmesg | tail
```

Вы должны увидеть следующие строки в выводе

```
...
[ 27.263923] srisa_pdrv: srisa_pdrv_init
[ 27.266836] srisa_pdrv: pdrv_probe
...
```

Строку с init мы видели ранее, но теперь после неё есть строка srisa-pdrv: pdrv_probe, которая говорит нам, что наш модуль успешно связался с устройством описанным нами в дополнении к дереву устройств.

3.4.3 Выполните команду ls /dev/* и обратите внимание, что должен появиться файл srisa_pdrv.

3.4.4 Отправим данные в srisa_pdrv, и проверьте ответ

```
$ echo 1234 > /dev/srisa_pdrv
```

```
srisa_pdrv: srisa_pdrv_open
srisa_pdrv: srisa_pdrv_ioctl
srisa_pdrv: srisa_pdrv_write: Get 5 bytes
srisa_pdrv: srisa_pdrv_release
```

Замечание: принято пять байт, так как последний байт это байт перевода каретки (\r).

3.4.5 Посмотрим на скорость записи данных, для этого выполните команду:

```
$ dd count=1 bs=1M if=/dev/zero of=/dev/srisa_pdrv
```

Команда dd может часто встречаться Вам при активной работе с Linux. Она обеспечивает считывание, или копирование данных из одного файла в другой. В данном случае, мы указываем, что ходим один раз (count=1) отправить 1 мегабайт (bs=1M) из файла /dev/zero (генератор 0) в наш драйвер. Чем больше мы отправим данных, тем точнее получим значение

скорости, однако из-за особенностей реализации считывания в нашем драйвере, большой объём данных может привести к печальным последствиям, так что увлекаться не стоит. Результат должен быть таким

```
srisa_pdrv: srisa_pdrv_open
srisa_pdrv: srisa_pdrv_write: Get 1048576 bytes
srisa_pdrv: srisa_pdrv_release
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.0318856 s, 32.9 MB/s
```

Последняя строка показывает сколько данных было передано, и на какой скорости. Результат может менятся от каждого запуска, но порядок величин должен быть одинаковый.

3.4.6 Попробуем считать данные из драйвера, для этого выполним следующую команду:

```
$ dd count=1 bs=10 if=/dev/srisa_pdrv
```

Тут мы опять используем команду dd, но на этот раз не указываем выходной файл, поэтому сможем увидеть вывод в консоли

```
srisa_pdrv: srisa_pdrv_open
srisa_pdrv: srisa_pdrv_read: Send 10 bytes
ABCDEFGHIsrisa_pdrv: srisa_pdrv_release
1+0 records in
1+0 records out
10 bytes copied, 0.0131135 s, 0.8 kB/s
```

Как видите, в выводе мы увидели набор букв, который был сгенерирован нашим драйвером.

3.4.7 Измерим скорость чтения данных, но на этот раз перенаправим вывод в файл:

```
$ dd count=1 bs=1M if=/dev/srisa_pdrv of=/root/drv.out
```

```
srisa_pdrv: srisa_pdrv_open
srisa_pdrv: srisa_pdrv_read: Send 1048576 bytes
srisa_pdrv: srisa_pdrv_release
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.140926 s, 7.4 MB/s
```

Скорость на чтение меньше, чем на запись, так как драйвер в начале заполняет свой внутренний массив, и затем отправляет данные. Так же тормозит и запись файла. Проверьте это, заменив файл назначения на /dev/null Для просмотра вывода выполните команду cat /root/drv.out и немного подождём, пока через терминал будет передан 1 мегабайт алфавита. Вывод можно прервать комбинацией Ctrl+C

3.4.8 Выгрузим ядро

```
$ rmmod chardev
```

И снова мы видим, что есть два вывода, один от модуля, который видели ранее, второй сообщает нам, что была выполнена функция по отключению от устройства.

3.4.9 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

3.5 Задание для закрепления

Внесите правки в модуль ядра, так, что бы командой считывания можно было получить все данные, записанные ранее через команду записи. Другими словами, пользователь два раза записал Hello после чего запросил на считывание 1Мбайт, в ответ он должен получить 10 байт: HelloHello.

Замечание: тут нам понадобиться работа со структурой kfifo (в коде переменная data_fifo). В коде уже вставлены части по её инициализации (mychar_dev.c), Вам остаётся вписать в нужные места вызовы функций для считывания и записи данных. При этом функция copy_from_user и copy_to_user уже не нужны.

Командой dd можно передавать данные из любых файлов, что может помочь Вам в проверке работы драйвера. Как один из вариантов, Вы создаёте текстовый файл, и пишите в него текст, далее, через комманду dd передаёте в драйвер, и второй командой dd считываете данные в другой файл, после чего можете сравнить два файла коммандой diff <path1> <path2>

Пример для записи данных в драйвере.

```
static ssize_t fifo_write(struct file *file, const char __user *buf,
    size_t count, loff_t *ppos)
{
    int ret;
    unsigned int copied;

    ret = kfifo_from_user(&test, buf, count, &copied);

    return ret ? ret : copied;
}
```

Пример для считывания данных из драйвера.

```
static ssize_t fifo_read(struct file *file, char __user *buf, size_t
    count, loff_t *ppos)
{
    int ret;
```

```
    unsigned int copied;

    ret = kfifo_to_user(&test, buf, count, &copied);

    return ret ? ret : copied;
}
```

* Модифицировать код таким образом, что бы можно было независимо обрабатывать два устройства (для проверки добавьте в device tree ещё одно устройство, pdrv2)

Лабораторная работа № 4

Драйвер GPIO

Цель: Управление выводами через встроенные в ядро драйверы и написание своего драйвера, для работы с выводами.

Описание:В предыдущей работе мы узнали, как можно управлять выводами через виртуальную файловую систему sysfs. Однако, при работе с внешними модулями, помимо интерфейса для передачи данных, могут понадобиться выводы, для управления состоянием модуля (сигнал сброса), или для проверки события (вывод ALARM на некоторых промышленных устройствах). Так же нам могут понадобиться выводы для индикации, или ввода информации. В этой работе Вы узнаете, как можно работать с выводами общего назначения через стандартные драйверы ядра, и как ими пользоваться в своих драйверах.

Полезные ссылки:

- Kernel Doc: GPIO Keys.
- Kernel Doc: LED's Common.
- Kernel Doc: Devicetree Dynamic Resolver Notes
- Kernel Doc: General Purpose Input/Output (GPIO)

- Kernel GIT: Input Event Codes
- Kernel Doc: GPIO Descriptor Consumer Interface

4.1 Встроенные драйверы

Наверное, одно из самые распространённые способы использования выводов общего назначения (GPIO) для считывания состояния кнопок (клавиатура, мышь, кнопка сброса и выключения), и простейшей индикации средствами светодиодов (доступ к диску, индикация наличия питания, активации режима Caps Lock и пр.). В ядре Linux есть готовые драйверы для использования GPIO как кнопок или индикаторов. В данном разделе, мы с Вами посмотрим, как можно управлять выводами общего назначения через эти драйверы.

4.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

4.1.2 Подключите по USB плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

4.1.3 Откроете терминал комбинацией клавиш Ctrl+Alt+T

4.1.4 Для начала подготовим рабочую папку, для разработки дополнения к дереву устройств системы:

```
# mkdir -p $BAGET/lab_04/of_dtb/
# cd $BAGET/lab_04/of_dtb/
```

4.1.5 Создадим и приступим к правке дополнения

```
# kate ./of_gpio.dts
```

4.1.6 Впишите следующие строки

```
/dts-v1/;
/plugin/;
/ {
   fragment@0 {
      target-path = "/";
      __overlay__ {
         grnled {
            compatible = "gpio-leds";
            led1 {
               label = "led1";
               gpios = <&gpiof 0 0>;
               linux,default-trigger = "heartbeat";
               default-state = "off";
            };
         };

         button {
            compatible = "gpio-keys";
            status = "okay";
            #address-cells = <1>;
            #size-cells = <0>;
            autorepeat;
            btn {
               label = "btn";
               linux,code = <4>;
               gpios = <&gpiod 6 1>;
            };
         };
      };
   };
};
```

Первая часть, отвечает за передачу управления светодиодом, подклю-

чённому к выводу F0, драйверу gpio-leds. Данный драйвер позволяет не только включать и выключать светодиод, но и активизировать его, при срабатывании одного из триггеров. К сожалению в текуще сборки ядра триггеры не были активированы (каждый триггер по сути это отдельный модуль ядра), поэтом мы не сможем посмотреть их в действии.

Вторая часть отвечает за настройку кнопки. В данном случае, при нажатии кнопки будет формироваться событие отправки символа Q (код кнопки 16). Полный список кодов кнопок можно посмотреть в файле по ссылке 5 в начале лабораторной работы.

Параметр gpios определяет, какой вывод мы отдаём под управление драйвером. Входное значение представления из себя массив из трёх элементов:

1. ссылка на банк выводов,
2. номер вывода в указанном банке,
3. полярность сигнала (0 — прямая, 1 — инверсная)

4.1.7 Сохраните файл Ctrl+S и закройте редактор Ctrl+Q

4.1.8 Скомпилируем наш фрагмент, для этого выполним следующую команду

```
# dtc --symbol -O dtb -o ./of_gpio.dtb ./of_gpio.dts
```

Замечание: при компиляции будет выдано предупреждение, его можно проигнорировать.

4.1.9 Скопируйте полученный файл на плату

```
# scp ./of_gpio.dtb netuser@192.168.100.200:/home/netuser/
```

4.1.10 Откройте программу gtkterm, и подключитесь к порту /dev/ttyUSB1

4.1.11 Переместите скопированный файл в папку barebox

```
$ mv /home/netuser/*.dtb /barebox/
```

4.1.12 Откройте скрипт barebox.sh для редактирования

```
$ nano /barebox/barebox.sh
```

4.1.13 Впишите после строки DTB=k5500vk018_rbm.dtb следующую строчку

```
fdt_apply -i $DTB -l ./of_gpio.dtb -o /dtb && DTB=/dtb
```

при наличии строки от прошлых работ, можно её переписать, удалить или оставить.

Сохраним изменения и выйдем из редактора: Ctl+S, Ctrl+X

4.1.14 Перезагрузите плату командой reboot, и внимательно изучите лог загрузки.

После того, как отработает отключение системы (записи, начинающиеся с [OK]). Система начнёт загрузку. Найдите в логе такие строки:

```
Hit m for menu or any other key to stop autoboot: 0
Booting entry 'mmc'
ext4 ext40: EXT2 rev 1, inode_size 256, descriptor size 32
eth1.ethaddr is not set
add_eth_mac_addr() failed for eth1 (1)
Skip dumping CK64 regs values to overlay: target CPU \freq 0 is less than
   or equal to 400000000
Entry Point: ffffffff807d5cd0
```

Если у Вас вывод не совпадает с выводом выше, то скорее всего Вы допустили ошибку в dts файле. Проверьте, что Вы правильно записали текст

исправления дерева устройств.

4.2 Проверка

4.2.1 Посмотрим на работу кнопки, для чего воспользуемся специальной утилитой для анализа событий

```
$ evtest /dev/input/event0
```

Как только вы её запустите, то увидите информацию о проверяемом источнике ввода:

```
root@baget:/dev/input/by-path# evtest platform-button-event
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "button"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 16 (KEY_Q)
Key repeat handling:
  Repeat type 20 (EV_REP)
    Repeat code 0 (REP_DELAY)
      Value    250
    Repeat code 1 (REP_PERIOD)
      Value     33
Properties:
Testing ... (interrupt to exit)
```

4.2.2 Нажмите и отпустите кнопку. Вы увидите как в консоли появиться несколько записей. Это информация о событии.

```
Event: time 1550139289.238366, type 1 (EV_KEY), code 16 (KEY_Q), value 1
Event: time 1550139289.238366, ---------- SYN_REPORT ------------
Event: time 1550139289.315121, type 1 (EV_KEY), code 16 (KEY_Q), value 0
Event: time 1550139289.315121, ---------- SYN_REPORT ------------
Event: time 1550139292.728959, type 1 (EV_KEY), code 16 (KEY_Q), value 1
Event: time 1550139292.728959, ---------- SYN_REPORT ------------
Event: time 1550139292.776846, type 1 (EV_KEY), code 16 (KEY_Q), value 0
Event: time 1550139292.776846, ---------- SYN_REPORT ------------
Event: time 1550139293.792620, type 1 (EV_KEY), code 16 (KEY_Q), value 1
Event: time 1550139293.792620, ---------- SYN_REPORT ------------
```

Замечание: обратите внимание на значение параметра value, оно кодирует состояние кнопки 0 — отпущена, 1 — нажата, 2 — удерживается

4.2.3 Прервём работу команды, нажав Ctrl+C (рекомендуется её запомнить, при работе с консолью в Linux, эта комбинация используется часто, для прерывания выполнения текущей команды или программы)

4.2.4 Дополнительно посмотрим, что на самом деле считывается из файла при работе с ним, для чего запустим конвейер из двух команд

```
$ cat /dev/input/event0 | hexdump
```

Тут уже при открытии Вы не увидите ничего, однако как только начнёте нажимать и отпускать кнопку, то увидите поток данных. В этом потоке храниться как информация о событии, так и временные метки, что мы видели ранее, инструмент evtest занимается преобразованием (парсингом) этого потока, в человечески воспринимаемый вид.

4.2.5 Прервём работу нажав Ctrl+C.

4.2.6 Светодиоды управляются через виртуальную файловую систему sysfs. Перейдите в следующий каталог

```
$ cd /sys/class/leds
```

4.2.7 Тут располагаются ссылки на все светодиоды доступные в системе. В нашем случае, дожно быть две папки mmc0:: и led1, проверьте выполнив команду

```
$ ls -l
```

4.2.8 Для управления состоянием светодиода отправьте 1 в файл brightness для включения светодиода, и 0 для выключения. При подключении через ШИМ контроллер, этот файл позволяет устанавливать яркость светодиода (например, для управления яркостью подсветки экрана)

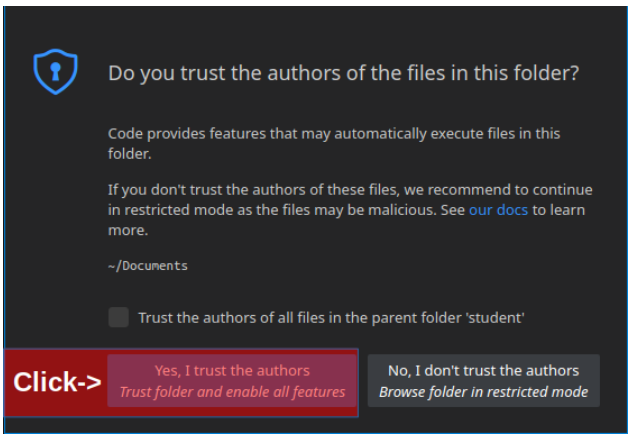
```
$ echo 1 > led1/brightness
$ echo 0 > led1/brightness
```

4.3 Собираем свой драйвер

Попробуем теперь использовать вывод общего назначения внутри собственного драйвера, что может часто понадобиться, при работе с внешними модулями. К примеру, для управления сигнала сброса, или для переключения между режимами и т. д.

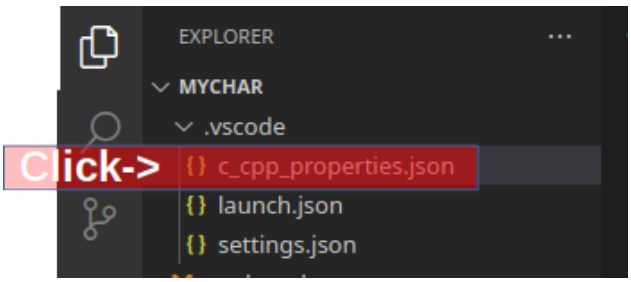
4.3.1 Скопируем код демонстрирующий работу с GPIO в пространстве ядра. Вернитесь в консоль виртуальной машины, и выполните следующую команду, и посмотрим в него

```
# cp -r $BAGET/support/gpio_driver $BAGET/lab_04/driver
# cd $BAGET/lab_04/driver; code .
```



Замечание: При первом входе, Вас могут спросить, доверяете ли вы автору, нужно нажать на кнопку Yes,

4.3.2 Добавим путей для разрешения части include директив. Откройте .vscode -> c_cpp_properties.json



4.3.3 Допишите в поле includePath следующие строки:

```
"configurations": [
  {
    "name": "linux-gcc-x64",
    "includePath": [
      "${workspaceFolder}/**",
      "/home/student/baget/support/linux-sources-4.15/include/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/uapi/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/generated/",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/asm/mach-baget",
      "/home/student/baget/support/linux-sources-4.15/arch/mips/include/asm/mach-generic"
    ],
```

для удобства, воспользуйтесь файлом vscode_lines.md в папке support.

Ctrl+S для сохранения.

4.3.4 Допишем compilerArgs, как на картинке ниже, для корректного перехода к реализации функций gpio драйвера ядра.

```
"compilerArgs": [
  "-DCONFIG_GPIOLIB"
]
```

4.3.5 Посмотрим на содержимое файла gpio_temp.c. В нём описан процесс подключения драйвера к выводу.

4.3.6 Так же откройте, и посмотрите на Makefile. В нём есть раздел install который поможет Вам скопировать на целевую плату скомпилированный модуль ядра. При этом, перед копированием будет запущена сборка

всего проекта. Это полезная практика, так как гарантирует Вам, что при исправлении мелкой опечатки, на плату попадёт код с исправлением, а не его предыдущая версия.

4.3.7 Перейдём в терминал (можно открыть вкладку TERMINAL в vscode, если её не видно, выберите в меню Terminal → New Terminal) и соберём модуль командой:

```
# ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make
```

4.3.8 Проверьте значение системной переменной BRD_PASS

```
# echo $BRD_PASS
```

если вывод был пустой, проинициализируйте переменную

```
# export BRD_PASS="usrnetuser"
```

4.3.9 Скопируйте модуль на плату

```
   # ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make install
```

Замечание: попробуйте сами разобрается, как в Makefile описан процес копирования файла на плату. Это может Вам помочь, при выполнении самостоятельных работ.

4.3.10 Внесите изменения в файл дерева-устройств

```
# kate $BAGET/lab_04/of_dtb/of_gpio.dts
```

удалите ноды led1 и button, и допишите новую ноду:

```
custom {
   compatible = "gpio,dummy";
   status = "okay";
   yellowled-gpios = <&gpiof 1 0>;
```

```
};
```

Замечание: обратите внимание на строку, в которой мы указываем информацию о выводе, которым нужно управлять. Первая часть (yellowled...) должна совпадать с той меткой, что указана в исходном коде, на 62 строке!

4.3.11 Сохраните изменения Ctrl+S и закройте редактор Ctrl+Q

4.3.12 Скомпилируем файл dts, для этого выполним следующую команду

```
# dtc --symbol -O dtb -o $BAGET/lab_04/of_dtb/of_gpio.dtb \
$BAGET/lab_04/of_dtb/of_gpio.dts
```

4.3.13 Скопируйте полученный файл на плату

```
# scp $BAGET/lab_04/of_dtb/of_gpio.dtb \
netuser@192.168.100.200:/home/netuser/
```

4.3.14 Перейдите в терминал платы, и переместите файл в папку barebox

```
$ mv /home/netuser/*.dtb /barebox/
```

4.3.15 Перезагрузите плату командой reboot

4.3.16 После перезагрузки, подгрузите модуль командой

```
$ insmod /home/netuser/gpio_temp.ko
```

Если не было допущено ошибок, то в течении 20 секунд плата будет перемигиваться с Вами светодиодами F0 и F1

4.3.17 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

4.4 Задание для закрепления

Измените код таким образом, что бы светодиод начинал мигать, когда кнопка нажата. Для этого воспользуйтесь таймером.

- Добавьте функцию инициализирующую кнопку (нужно сделать всё то же, что и при инициализации светодиода, в функции int myled_init(int gpio_id), за исключением того, что направления вывода должно быть GPIOD_INPUT
- Для считывания состояния кнопки, используйте функцию gpio_get_value(<gp Помните, что кнопка на плате имеет инверсную логику, другими словами в отпущенном состоянии вывод прижат к питанию, и возвращает 1.
- Пример кода по инициализации таймера

```
      #include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
#include <linux/init.h> /* Needed for the macros */
#include <linux/timer.h>
#include <linux/jiffies.h>

int g_time_interval = 250; // miliseconds
struct timer_list g_timer;

void _TimerHandler(struct timer_list *tm)
```

```
{
   static bool state = true;
   printk("%s: %s", DRIVER_NAME, __func__);
   state = !state;
   mod_timer(tm, jiffies + msecs_to_jiffies(g_time_interval));
}

static int __init my_init(void)
{
   printk(KERN_INFO "My module inserted into kernel!!!.\n");

   /*Starting the timer.*/
   setup_timer(&g_timer, _TimerHandler, 0);
   mod_timer( &g_timer, jiffies +
      msecs_to_jiffies(g_time_interval));

   return 0;
}

static void __exit my_exit(void)
{
   del_timer(&g_timer);
   printk(KERN_INFO "My module exited from kernel!!!\n");
}

module_init(my_init);
module_exit(my_exit);
```

- не забудьте добавить освобождение ресурсов для кнопки и таймера.

Лабораторная работа № 5

Драйвер SPI устройства

Цель: Активировать SPI контроллер, и написать простой драйвер для устройства подключённого к SPI интерфейсу.

Описание:В данной работе, мы познакомимся со структурой ядра, позволяющей взаимодействовать с spi контроллером целевой платы. Данная структура является программным интерфейсом, позволяющим отвязать реализацию взаимодействия с внешним аппаратным модулем по интерфейсу SPI, от особенностей реализации SPI контроллера на конкретной аппаратной платформе.

Полезные ссылки:

- Последовательный интерфейс SPI.
- Kernel Doc: Devicetree Dynamic Resolver Notes
- Kernel Doc: SPI API

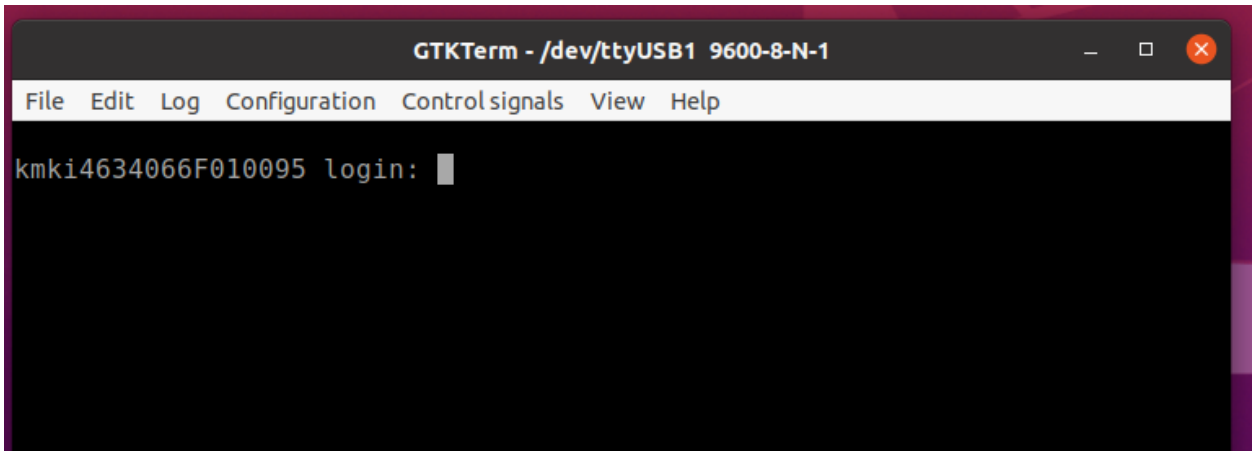
5.1 Запуск и подключение к устройству

5.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

5.1.2 Подключите по USB плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

5.1.3 Откройте программу gtkterm, и подключитесь к порту /dev/ttyUSB1

5.1.4 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. Вы должны увидеть следующий вывод:



Вы подключились к консоли устройства. Введите логин root и пароль root.

5.2 Подготовка Device Tree Overlay

Основная конфигурация целевой платы предполагает, что практически все выводы настроены для управления через блок GPIO. Для работы с SPI, нам необходимо ввести две правки: разрешить работу SPI контроллера и передать под его управления выводы, для коммуникации с внешним миром.

5.2.1 Создадим рабочую папку в виртуальной машине для создания device-tree overlay и перейдём в неё

```
# mkdir -p $BAGET/lab_05/devtree
# cd $BAGET/lab_05/devtree
```

5.2.2 Создадим новый файл

```
# kate of_spi1.dts
```

и запишем в него следующий текст

```
/dts-v1/;
/plugin/;
/ {
   fragment@0 {
      target-path = "/gpio@1b400200";

      __overlay__ {

         spi1_pins {
            phandle = <0x01>;

            spi1_clk {
               function = "spi1_clk";
               groups = "spi1_clk";
            };

            spi1_di {
               function = "spi1_di";
               groups = "spi1_di";
            };

            spi1_do {
               function = "spi1_do";
               groups = "spi1_do";
            };

            spi1_cs0 {
               function = "spi1_cs0";
               groups = "spi1_cs0";
            };
         };
      };
   };
```

```
   fragment@1 {
      target-path = "/spi@1a800000";

      __overlay__ {
         status = "okay";
         pinctrl-names = "default";
         pinctrl-0 = <0x01>;
      };
   };


   __local_fixups__ {

      fragment@1 {
         __overlay__ {
            pinctrl-0 = <0x00>;
         };
      };

   };
};
```

Разберём, что мы тут создали. Данное дополнение состоит из нескольких фрагментов, так как переписать нам нужно две разные ноды, однако изменения эти должны быть синхронизированы между собой, по этой причине они находятся в одном файле.

Первый фрагмент, это дополнение к ноде описывающей банк выводов A. В этом дополнении мы создаём конфигурацию, для передачи управления частью выводов SPI контроллеру. Будьте внимательны, не любой вывод может быть передан SPI контроллеру.

Следующий фрагмент правок, касается дополнения к описанию spi контроллера. Первое, это его активация, записью в параметр status значения okay. Так же, мы вводим два дополнительных параметра, один хранит в себе название конфигурации выводов, второй — для указания идентификатора конфигурации, обратите внимание, что значение у него совпадает

(и это важно) со значением phandle у описания выводов из первого фрагмента.

Последняя строка гарантирует, что после применения этого дополнения, значение параметра pinctrl-0 будет соответствовать значению phandle нашей группы spi1_pins. Дело в том, что значение phandle из первого фрагмента, будет актуализировано, для разрешения коллизий (так как в исходном device-tree уже есть нода, с таким значением phandle), и последняя строка в нашем описании исправлений поясняет системе, что после разрешения коллизии, необходимо обновить значение указанного параметра (значение 0 зарезервировано, и в данном контексте означает ссылку на значение параметра phandle).

Узнать, какой банк выводов, и какие выводы могут быть использованы с тем или иным интерфейсом, какие значения phandle уже использованы, и просто для любопытства можно одним из двух способов:

1. Из самой системы, изучив подкаталоги gpio, в каталоге /proc/device-tree/ (рекомендуется для считывания значение того или иного параметра использовать команду cat).

2. Преобразовав бинарный файл device-tree системы в текстовый вид

```
# dtc --symbol -O -dts -o  /tmp/root.dts \
$BAGET/support/barebox/k5500vk018_rbm.dts
# kate /tmp/root.dts
```

Посмотрите на ноды, описывающие банки GPIO выводов. Так же обратите внимание, на объявление конфигураций выводов для sdhci контроллера, особенно, как эта конфигурация разбита, между двумя банками выводов, на значение параметра phandle каждой группы. Сравните значение параметров phandle конфигурационных груп, и значения перечисленные в параметре pinctrl-0 у ноды для контроллера sdhci. (не стесняйтесь использовать поиск по файлу).

5.2.3 Сохраните изменения Ctrl+S и закройте редактор Ctrl+Q

5.2.4 Скомпилируем файл и скопируем его на плату

```
# dtc --symbol -O dtb -o ./of_spi1.dtb ./of_spi1.dts
# scp ./of_spi1.dtb netuser@192.168.100.200:/home/netuser/
```

5.2.5 Перейдите в терминал платы, и переместите файл в папку barebox

```
$ mv /home/netuser/of_spi1.dtb /barebox/
```

5.2.6 Откройте скритп barebox.sh для редактирования

```
$ nano /barebox/barebox.sh
```

5.2.7 Впишите после строки DTB=k5500vk018_rbm.dtb следующую строчку (при наличии других строк, закомментируйте их символом #)

```
fdt_apply -i $DTB -l of_spi1.dtb -o /dtb && DTB=/dtb
```

Сохраним изменения и выйдем из редактора (Ctrl+S, Ctrl+X)

5.2.8 Перезагрузите плату командой reboot.

5.2.9 После перезагрузки входим в систему (root/root). И выполняем команду

```
$ ls /sys/class/spi_master/
```

Вы должны увидеть, два имени: spi0 (это контроллер qspi flash-памяти) и spi1 (тот что мы активировали). Если второго имени нет, значит дополнение к device-tree не отработало. Проверьте, что Вы правильно написали команду в файле /barebox/barebox.sh, а так же изучите вывод платы во время загрузки.

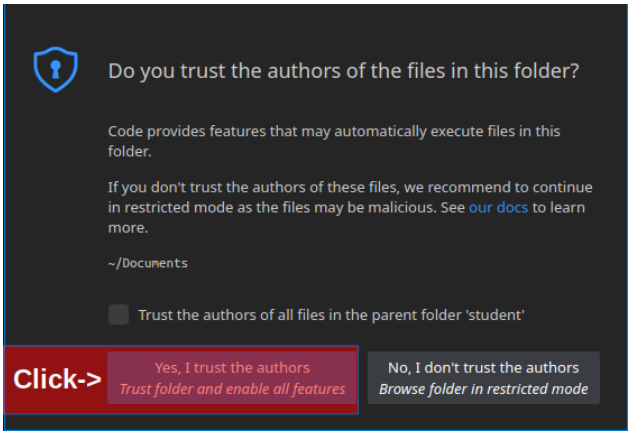
5.3 Собираем драйвер

В Linux есть драйвер для управления spi контроллером из пространства пользоватея, он называется spidev. Однако у него есть ряд недостатков, которые мешают использовать его в боевых проектах. В системе целевой платы этот драйвер отсутствует, и посмотреть на его работу мы не сможем. Поэтому сразу же начнём собирать свой драйвер.

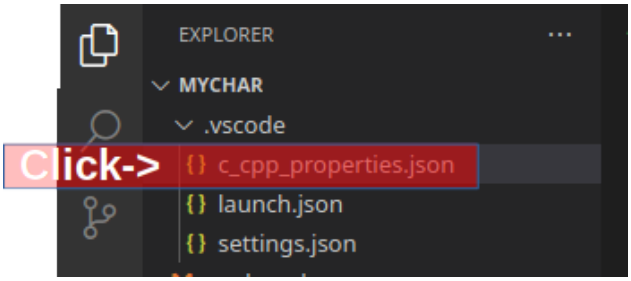
5.3.1 Подготовим папку для сборки драйвера, перейдём в неё и откроем в редакторе vscode

```
# cp -r $BAGET/support/spi_driver $BAGET/lab_05/driver
# cd $BAGET/lab_05/driver; code .
```

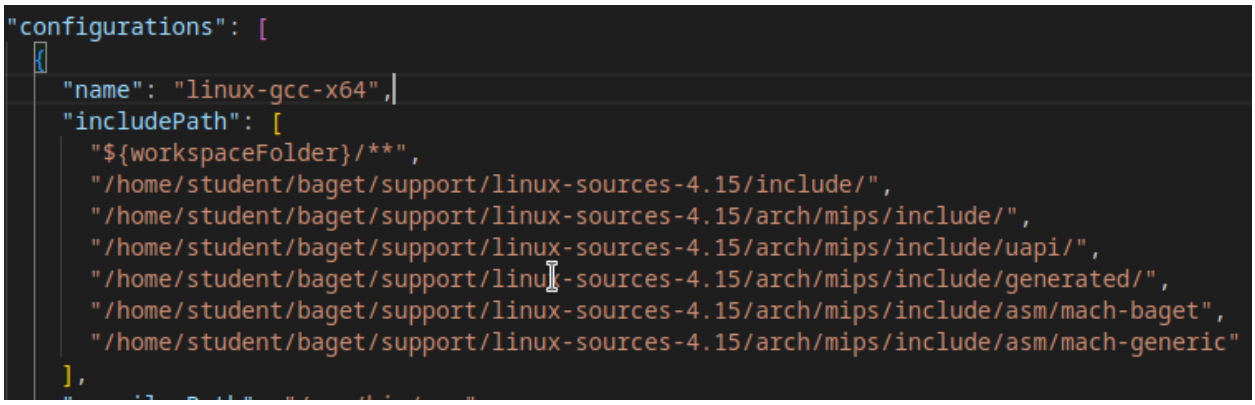
Замечание: При первом входе, Вас могут спросить, доверяете ли вы автору, нужно нажать на кнопку Yes,



5.3.2 Добавим путей для разрешения части include директив. Откройте .vscode -> c_cpp_properties.json



5.3.3 Допишите в поле includePath следующие строки:



для удобства, воспользуйтесь файлом vscode_lines.md в папке support. Ctrl+S для сохранения.

5.3.4 Изучите исходны код в файле spi_temp.c. Для работы с SPI интерфейсом, в Linux есть готовые структуры, которые в данной работе используются.

Из особенностей, которые хотелось бы отметить, это структура описание которой начинается с 11 строки. В ней описывается режим работы SPI интерфейса, в том числе номер интерфейса в системе. В нашем случае 1, так как 0 будет занят контроллером QSPI памяти.

Так же обратите внимание на организацию транзакции, описанную в функции ext_spi_write (начало описания с 28 строки). В ней используется структура spi_transfer, информацию о которой Вы можете получить, перейдя по ссылке 3 приведённой в начале лабораторной работы.

5.3.5 Перейдём в терминал (можно открыть вкладку TERMINAL в vscode, если её не видно, выберите в меню Terminal → New Terminal) и соберём модуль командой

```
# ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make
```

5.3.6 Перейдём в терминал (можно открыть вкладку TERMINAL в vscode, если её не видно, выберите в меню Terminal → New Terminal) и соберём модуль командой

```
# ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make
```

5.3.7 Проверьте значение системной переменной BRD_PASS

```
# echo $BRD_PASS
```

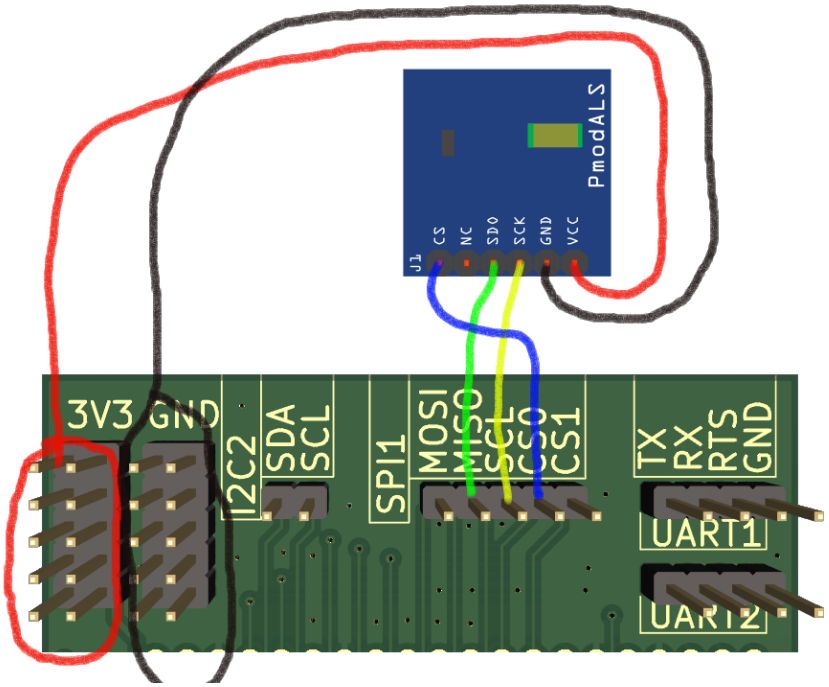
если вывод был пустой, проинициализируйте переменную

```
# export BRD_PASS="usrnetuser"
```

5.3.8 Скопируйте модуль на плату

```
# ARCH=mips CROSS_COMPILE=mips64el-linux-gnuabi64- make install
```

5.3.9 Подключите модуль PmodALS как показано на рисунке



5.3.10 Перейдите в терминал платы, и активируйте драйвера

```
$ insmod /home/netuser/spi_temp.ko
```

5.3.11 Вы должны увидите в терминале, информацию о том, что драйвер зарегистрирован, и 10 раз считано «сырое» значение освещённости с датчика.

5.3.12 Выгрузите драйвер командой

```
$ rmmod spi_temp
```

5.3.13 Выключите плату, для чего в начале введите команду

```
   $ poweroff
```

дождитесь, как появиться надпись

```
   reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

5.4 Задание для закрепления

Внесите изменения в текущий драйвер таким образом, что бы создавался файл в папке /dev, чтение которого давало бы текущее значение освещённости.

В этом Вам может помочь код из лабораторной работы 3.

* Внесите изменение в драйвер устройств таким образом, что бы частота, номер spi шины, и индекс вывода Chip Select передавались через описание в дереве устройств:

```
pmodals {
   compatible = "digilent,pmodals";
```

```
    spi_cfg = <400000 1 0>;
    status = "okay";
};
```

Лабораторная работа № 6

Драйвер I2C устройства

Цель: Активировать I2C контроллер, и написать простой драйвер для устройства подключённого к I2C шине.

Описание:Работа с I2C интерфейсом не требует разработки модуля ядра, при условии, что внешний модуль не требует дополнительных управляющих линий. В ОС Linux есть набор программных продуктов, позволяющих сканировать (i2cdetect) подключённые модули, с целью определения адресов устройств на шине, а так же для контроля исправности. Так же модули позволяющие отправлять (i2c_set) и принимать (i2c_get) данные .

В данной работе, будет приведён пример приложения, обеспечивающего настройку и взаимодействие с внешним модулем.

Полезные ссылки:

- Последовательный интерфейс I2C.
- Kernel Doc: Devicetree Dynamic Resolver Notes
- Kernel Doc: I2C API

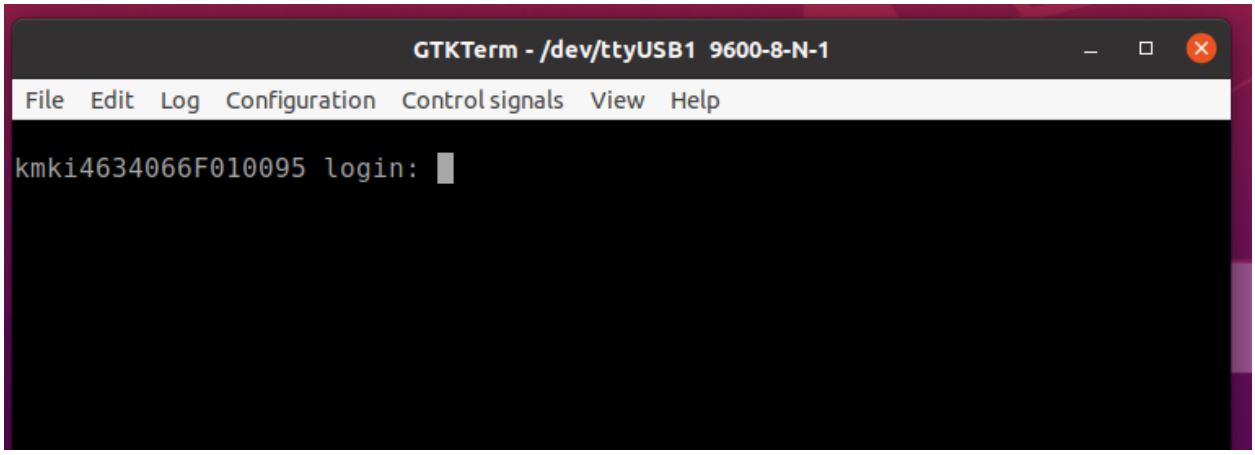
6.1 Запуск и подключение к устройству

6.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

6.1.2 Подключите по USB плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

6.1.3 Откройте программу gtkterm, и подключитесь к порту /dev/ttyUSB1

6.1.4 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. Вы должны увидеть следующий вывод:



Вы подключились к консоли устройства. Введите логин root и пароль root.

6.2 Подготовка Device Tree Overlay

Основная конфигурация целевой платы предполагает, что практически все выводы настроены для управления через блок GPIO. Для работы с I2C, нам необходимо ввести две правки: разрешить работу I2C контроллера и передать под его управления выводы, для коммуникации с внешним миром.

Процессор на плате содержит три конролера I2C, однако воспользоваться мы можем только одним, так как выводы вторых двух заняты контроллером карты памяти (sdhci), и при их назначении на I2C интерфейс, Linux не сможет загрузиться.

6.2.1 Создадим рабочую папку в виртуальной машине для создания device-tree overlay и перейдём в неё

```
# mkdir -p $BAGET/lab_06/devtree
# cd $BAGET/lab_06/devtree
```

6.2.2 Создадим новый файл

```
# kate of_i2c2.dts
```

и запишем в него следующий текст

```
/dts-v1/;
/plugin/;
/ {
   fragment@0 {
      target-path = "/gpio@1b400380";
      __overlay__ {

         i2c2_pins {
            phandle = <0x01>;
            i2c2_scl {
               function = "i2c2_scl";
               groups = "i2c2_scl";
            };
            i2c2_sca {
               function = "i2c2_sca";
               groups = "i2c2_sca";
            };
         };
      };
   };
```

```
   fragment@1 {
      target-path = "/i2c@1b400130";
      __overlay__ {
         status = "okay";
         pinctrl-names = "default";
         pinctrl-0 = <0x01>;
      };
   };


   __local_fixups__ {
      fragment@1 {
         __overlay__ {
            pinctrl-0 = <0x00>;
         };
      };
   };
};
```

Разберём, что мы тут создали. Данное дополнение состоит из нескольких фрагментов, так как переписать нам нужно две разные ноды, однако изменения эти должны быть синхронизированы между собой, по этой причине они находятся в одном файле.

Первый фрагмент, это дополнение к ноде описывающей банк выводов D. В этом дополнении мы создаём конфигурацию, для передачи управления частью выводов I2C контроллеру. Будьте внимательны, не любой вывод может быть передан I2C контроллеру.

Следующий фрагмент правок, касается дополнения к описанию i2c контроллера. Первое, это его активация, записью в параметр status значения okay. Так же, мы вводим два дополнительных параметра, один хранит в себе название конфигурации выводов, второй — для указания идентификатора конфигурации, обратите внимание, что значение у него совпадает (и это важно) со значением phandle у описания выводов из первого фраг-

мента.

Последняя строка гарантирует, что после применения этого дополнения, значение параметра pinctrl-0 будет соответствовать значению phandle нашей группы i2cs_pins. Дело в том, что значение phandle из первого фрагмента, будет актуализировано, для разрешения коллизий (так как в исходном device-tree уже есть нода, с таким значением phandle), и последняя строка в нашем описании исправлений поясняет системе, что после разрешения коллизии, необходимо обновить значение указанного параметра (значение 0 зарезервировано, и в данном контексте означает ссылку на значение параметра phandle).

Узнать, какой банк выводов, и какие выводы могут быть использованы с тем или иным интерфейсом, какие значения phandle уже использованы, и просто для любопытства можно одним из двух способов:

1. Из самой системы, изучив подкаталоги gpio, в каталоге /proc/device-tree/ (рекомендуется для считывания значение того или иного параметра использовать команду cat).

2. Преобразовав бинарный файл device-tree системы в текстовый вид

```
# dtc -@ -O -dts -o  /tmp/root.dts \
$BAGET/support/barebox/k5500vk018_rbm.dts
# kate /tmp/root.dts
```

Посмотрите на ноды, описывающие банки GPIO выводов. Так же обратите внимание, на объявление конфигураций выводов для sdhci контроллера, особенно, как эта конфигурация разбита, между двумя банками выводов, на значение параметра phandle каждой группы. Сравните значение параметров phandle конфигурационных груп, и значения перечисленные в параметре pinctrl-0 у ноды для контроллера sdhci. (не стесняйтесь использовать поиск по файлу).

6.2.3 Сохраните изменения Ctrl+S и закройте редактор Ctrl+Q

6.2.4 Скомпилируем и отправим файл на плату

```
# dtc -@ -O dtb -o ./of_i2c2.dtb ./of_i2c2.dts
# scp ./of_i2c2.dtb netuser@192.168.100.200:/home/netuser/
```

6.2.5 Перейдите в терминал платы, и переместите файл в папку barebox

```
$ mv /home/netuser/of_i2c2.dtb /barebox/
```

6.2.6 Внесити правкив вайл barebox.sh

```
$ nano /barebox/barebox.sh
```

Впишите после строки DTB=k5500vk018_rbm.dtb следующую строчку

```
fdt_apply -i $DTB -l of_i2c2.dtb -o /dtb && DTB=/dtb
```

Сохраним изменения и выйдем из редактора (Ctrl+S, Ctrl+X).

6.2.7 Перезагрузите плату командой reboot.

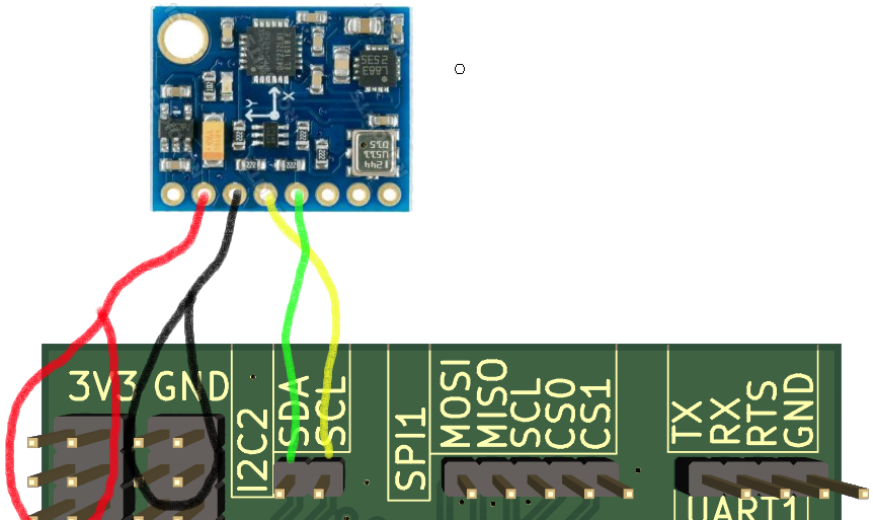
6.2.8 После перезагрузки входим в систему (root/root). И выполняем команду

```
$ ls /dev/i2c*
```

Вы должны увидеть, в выдачи i2c-0. Если Вы его не увидели, выполните команду dmesg на целевой платформе, и изучите лог загрузки, на предмет ошибок инициализации I2C контроллера, или выполнения команды overlay загрузчиком.

6.3 Работа с I2C из user-space

6.3.1 Подключите модуль HW-290 к плате, как показано на рисунке



6.3.2 Просканируем доступные к взаимодействию I2C устройства, а так же узнаем их адреса. Для этого воспользуемся командой i2cdetect

```
# i2cdetect -y 0
```

где 0 это индекс i2c адаптера, в нашем случае он один, поэтому 0 (вспомним что выдала команда ls /dev/i2c*)

```
root@baget:~# i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

Могут возникнуть ошибки при сканировании, убедитесь, что к выводам I2C контроллера подключены подтягивающие к питанию резисторы.

6.3.3 Скопируем пример программы работающей с I2C и откроем её в редакторе vscode

```
# cp -r $BAGET/support/i2c_app $BAGET/lab_06/app
# cd $BAGET/lab_06/app; code .
```

Проект состоит из нескольких файлов. Первый, это основной файл программы appi2c.c. Два других являются обёртками, для взаимодействия с I2C интерфейсом.

6.3.4 Скомпилируем приложение

```
# make
```

6.3.5 И если ошибок не появилось, отправим программу на плату

```
# scp $BAGET/lab_06/app/app \
netuser@192.168.100.200:/home/netuser
```

6.3.6 Запустим скопированное приложение на плате

```
$ /home/netuser/app
```

Программа должна успешно подключиться к модулю, и выдавать в консоль данные с акселерометра.

6.3.7 Остановите работу программы нажав Ctrl+C

6.3.8 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

6.4 Задание для закрепления

Допишите программу так, что бы она сообщала, о направлении перемещения в плоскости XY (вверх, вниз, налево, направо)

* Собрать модуль ядра, который будет обеспечивать коммуникацию через I2C интерфейс с акселерометром. Считывание сырых данных через файл в папке /dev

Лабораторная работа № 7

Драйвер Serial устройства

Цель: Активация Serial устройства, и написание ПО для работы с ним.

Описание:Внести правки в device tree для активации Serial контроллера и передачи выводов под его управление. Собрать и протестировать тестовое ПО, которое обеспечивает работу с устройством через Serial интерфейс.

Полезные ссылки:

- Linux Serial Ports Using C/C++.
- Kernel Doc: Devicetree Dynamic Resolver Notes
- Kernel Doc: SPI API

7.1 Запуск и подключение к устройству

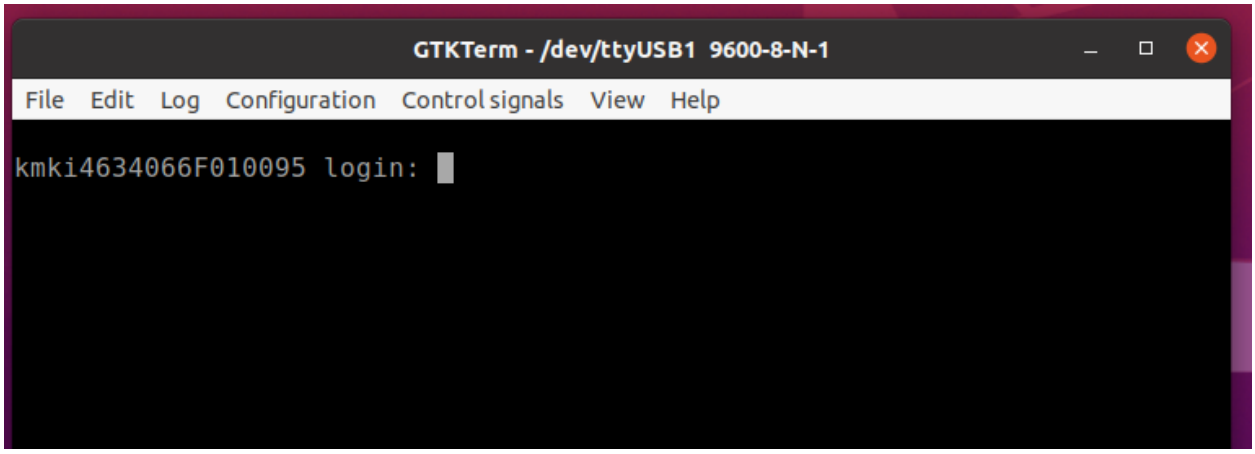
7.1.1 Запустите виртуальную машину. Логин и пароль для входа: student / usrstudent.

7.1.2 Подключите по USB плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной

машине (меню Device→USB).

7.1.3 Откройте программу gtkterm, и подключитесь к порту /dev/ttyUSB1

7.1.4 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. Вы должны увидеть следующий вывод:



Вы подключились к консоли устройства. Введите логин root и пароль root.

7.2 Подготовка Device Tree Overlay

7.2.1 Создадим рабочую папку в виртуальной машине для создания device-tree overlay и перейдём в неё

```
# mkdir $BAGET/lab_07/devtree
# cd $BAGET/lab_07/devtree
```

7.2.2 Создадим новый файл

```
# kate of_serial2.dts
```

и запишем в него следующий текст

```
/dts-v1/;
/plugin/;
/ {
    fragment@0 {
```

```
        target-path = "/gpio@1b400200";
        __overlay__ {

            serail2_pins {
                phandle = <0x01>;
                uart2_rx {
                    function = "uart2_rx";
                    groups = "uart2_rx";
                };
                uart2_tx {
                    function = "uart2_tx";
                    groups = "uart2_tx";
                };
            };
        };
    };


    fragment@1 {
        target-path = "/serial@1b511000";
        __overlay__ {
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <0x01>;
        };
    };


    __local_fixups__ {
        fragment@1 {
            __overlay__ {
                pinctrl-0 = <0x00>;
            };
        };
    };
};
```

7.2.3 Сохраните изменения Ctrl+S и закройте редактор Ctrl+Q

7.2.4 Скомпилируем наш фрагмент, для этого выполним следующую команду

```
# dtc --symbol -O dtb -o ./of_serial2.dtb ./of_serial2.dts
```

7.2.5 Скопируйте полученный файл на плату

```
# scp ./of_serial2.dtb netuser@192.168.100.200:/home/netuser/
```

7.2.6 Перейдите в терминал платы, и переместите файл в папку barebox

```
$ mv /home/netuser/of_serial2.dtb /barebox/
```

7.2.7 Откройте скритп barebox.sh для редактирования

```
$ nano /barebox/barebox.sh
```

и впишите после строки DTB=k5500vk018_rbm.dtb следующую строчку

```
fdt_apply -i $DTB -l of_serial2.dtb -o /dtb && DTB=/dtb
```

7.2.8 Сохраните изменения (Ctrl+S, Ctrl+X), и перезагрузите плату ($ reboot)

7.2.9 После перезагрузки входим в систему (root/root). И выполняем команду

```
$ ls -l /sys/class/tty/ttyS2
```

папка должна ссылаться на устройство с адресом активированного нами контроллера, если это не так, значит Вы допустили ошибку в описании дополнения к дереву устройств

```
/sys/class/tty/ttyS2->../../devices/platform/1b511000.serial/tty/ttyS2
```

7.3 Тестовое приложение

7.3.1 Скопируйте исходники в рабочую папку

```
# cp -r $BAGET/support/serial_app $BAGET/lab_07/app
```

7.3.2 Перейдём в рабочий каталог, и откроем vscode

```
# cd $BAGET/lab_07/app; code .
```

7.3.3 Откроем файл main.c. Как видим тут только две функции. Одна для инициализации UART контроллера, в которой отключается вся предобработка отправляемых и принимаемых данных операционной системой, что позволяет нам передавать различные данные через интерфейс без потерь. Так же тут настраивается скорость передачи и приёма. Обратите внимание, потенциально скорость передачи и приёма может быть различной.

В основном коде создаётся два буфера, для приёма и отправки данных. Затем один буфер инициализируется стартовыми значениями, и запускается передача данных. Тут нужно обратить внимание на то, что аппаратный FIFO буфер UART контроллера имеет ограниченный объём, в данном случае 16 бит (считайте значение из файла /sys/class/tty/ttyS2/xmit_fifo_size). По этой причине, если Вы не успеете считать 16 бит из приёмного буфера, то всё что будет принято далее, будет потеряно.

7.3.4 Скомпилируйте и отправьте код на целевую платформу

```
# make
# scp ./serapp netuser@192.168.100.200:/home/netuser
```

7.3.5 Замкните выводы RX и TX и запустите программу на плате

```
$ /home/netuser/serapp
```

Вы должны получить информацию, об успешном завершении. Если нет, то необходимо проверить подключение, и настройки.

7.3.6 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

Лабораторная работа № 8

Web интерфейс

Цель: Простой WEB интерфейс для взаимодействия с платформой.

Описание:В данной лабораторной работе, мы создадим простой web сервер для мониторинга ресурсов, и управления железом удалённо. Данный проект покажет Вам, как можно организовать графический пользовательский интерфейс для работы с железом, без необходимости тащить на плату гигабайты графических библиотек, и поднимать драйверы для работы с дисплеями.

8.1 Правка rootfs

В данной работе будет производиться работа с сервером Apache2 и CGI интерпретатор. В современной WEB разработке предпочтение отдают FastCGI или uWSGI, исходя из вопросов быстродействия. Однако у нас не критически важный объект, а демонстрация идеи, и настройка Apache, авторам кажется более простой, чем настройка связки nginx + uWSGI. Однако, если читатели заинтересуются этой темой, то авторы будут спокойны, так как указали на то, «как нынче носят».

Главное, дорогой читатель, не трать время на PHP.

8.1.1 Перейдём в папку первой лабораторной работы, для добавления необходимых нам компонентов

```
# cd $BAGET/lab_01
```

8.1.2 Перейдём в папку первой лабораторной работы, для добавления необходимых нам компонентов

```
# sudo cp /usr/bin/qemu-mips64el-static $BAGET/lab_01/rootfs/usr/bin
# sudo chroot $BAGET/lab_01/rootfs
```

8.1.3 Установим необходимые пакеты, веб-сервер apache2, расширение для работы cgi скриптов, а также pyhton3, так как CGI скрипты на bash создавать можно, но больно.

```
# apt-get install -y \
apache2 \
python3 \
virtualenv
```

8.1.4 Активируем модуль обработки cgi скриптов apache сервера

```
# a2enmod cgi
```

8.1.5 По умолчанию, cgi скрипты ищутся в папке /usr/lib/cgi-bin, далее мы исправим путь поиска на /var/www/cgi-bin, для чего исправим описание сайта:

```
# nano /etc/apache2/sites-enabled/000-default.conf
```

и внутри описания VirtualHost, допишем

```
ScriptAlias /cgi-bin/ /var/www/cgi-bin/
<Directory "/var/www/cgi-bin">
   Options ExecCGI
```

```
   SetHandler cgi-script
</Directory>
```

Сохраните и закройте файл (Ctrl+S, Ctrl+X)

8.1.6 Завершим настройку rootfs

```
# exit
# sudo rm ./rootfs/usr/bin/qemu-mips64el-static
```

8.1.7 Скопируйте папку html и cgi-bin из каталога support в папку rootfs

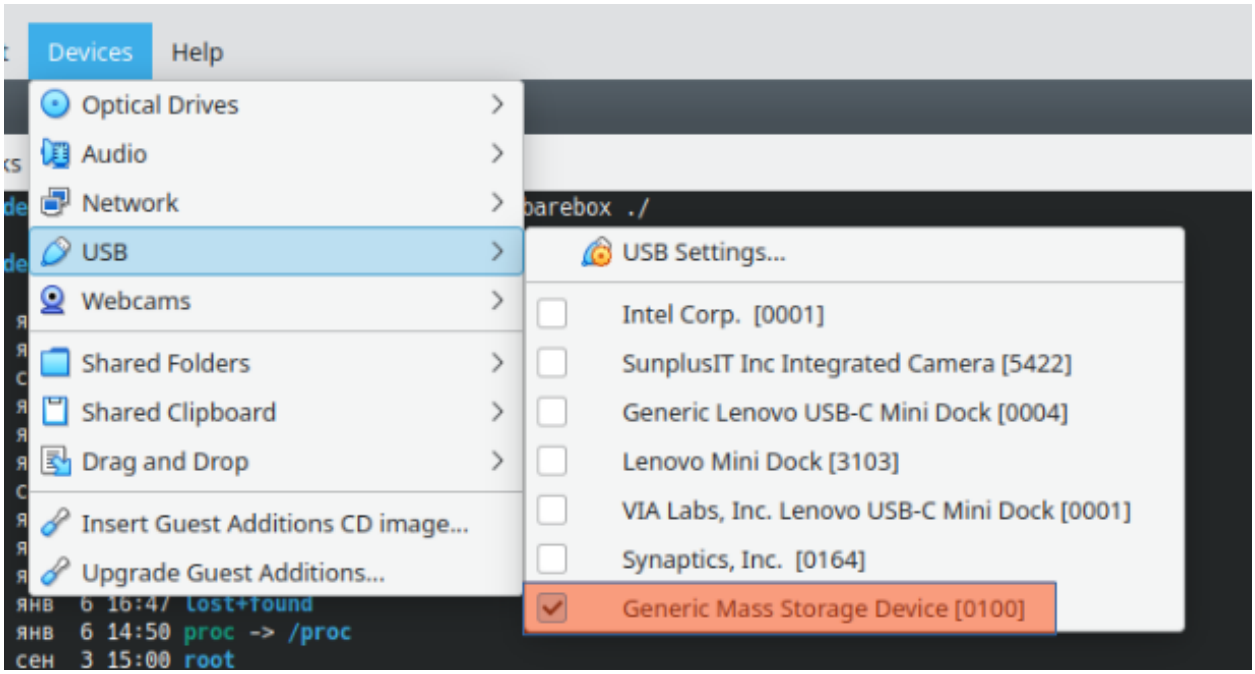
```
# sudo cp -r $BAGET/support/web/* ./rootfs/var/www/
```

8.1.8 Установим владельца для папки с веб сервером

```
# sudo chown -R www-data:www-data ./rootfs/var/www/
```

8.2 Запись на SD карту

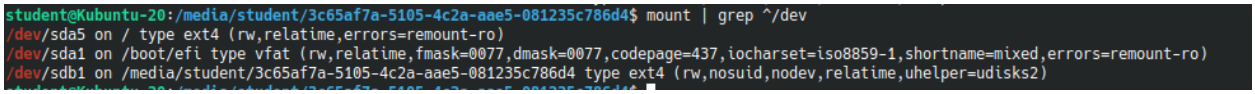
8.2.1 Вставьте SD карту в ПК, и подключите её к виртуальной машине



8.2.2 Необходимо узнать точку монтирования SD карты. Она может отличаться в зависимости от ридера, что Вы используете. Для упрощения ввода, выполните команду в консоле:

```
# mount | grep ^/dev
```

для получения списка примонтированных устройств, при этом командой grep происходит фильтрация лишнего вывода, и отображается только список блочных устройств. Вывод будет примерно как на рисунке



Первые две строчки, это разделы виртуального жёсткого диска, далее точка монтирования SD карты (/media/student/...)

Для упрощения ввода, создадим временную системную переменную, которой назначим путь, для примера(!) на рисунке выше:

```
# export BAGET_SD=\
/media/student/3c65af7a-5105-4c2a-aae5-081235c786d4
```

Нужно помнить, что эта переменная существует только в том терминале, где был вызван export, в остальных окнах терминала она не существует, если Вы закроете текущее окно, или Вам понадобиться эта переменная в другом окне, необходимо будет повторить процедуру присвоения.

8.2.3 Удалите все файлы с sd карты

```
# sudo rm -rf $BAGET_SD/*
```

скопируйте созданную rootfs на sd карту

```
# sudo cp -a $BAGET/lab_01/rootfs/* $BAGET_SD/
```

и отмантируйте SD карту

```
# sudo umount $BAGET_SD
```

(это важно, так как в ОС Linux запись данных на носители происходит в фоне, и если вы отключите накопитель «неправильно» то нет гарантии, что данные успели записаться на носитель!). Дождитесь когда команда выполниться, после чего достаньте SD карту и вставьте её в плату.

8.2.4 Подключите USB шнуром плату к ПК. Проверьте, и при необходимости подключите USB устройство FTDI RBM_C1K5500VK018 к виртуальной машине (меню Device→USB).

8.2.5 Запустите программу gtkterm

8.2.6 Выберите порт /dev/ttyUSB1

Если в окне не появился текст, и на нажатие клавиши Enter так же нет реакции, нажмите на кнопку Reset на плате. Изучите вывод, там будет написано, что пошло не так.

Один из вариантов, когда при записи файлов на SD карту что-то побилось. В этом случае, вы увидите много записей как в примере ниже:

```
srisa-sdhci 1b50b000.sdhci@1b50b000.of: registered as mci0
srisa-sdhci 1b50b000.sdhci@1b50b000.of: SDHCI timeout while waiting
    for done
srisa-sdhci 1b50b000.sdhci@1b50b000.of: error on command 8
srisa-sdhci 1b50b000.sdhci@1b50b000.of: state = 0400 03ff, interrupt =
    8001 0001
srisa-sdhci 1b50b000.sdhci@1b50b000.of: r0 00000000 r1 00000000 r2
    00000000 r3 00000000
```

Нужно повторить все шаги этого раздела. **При вынимании SD карты убедитесь, что плата обесточена!**

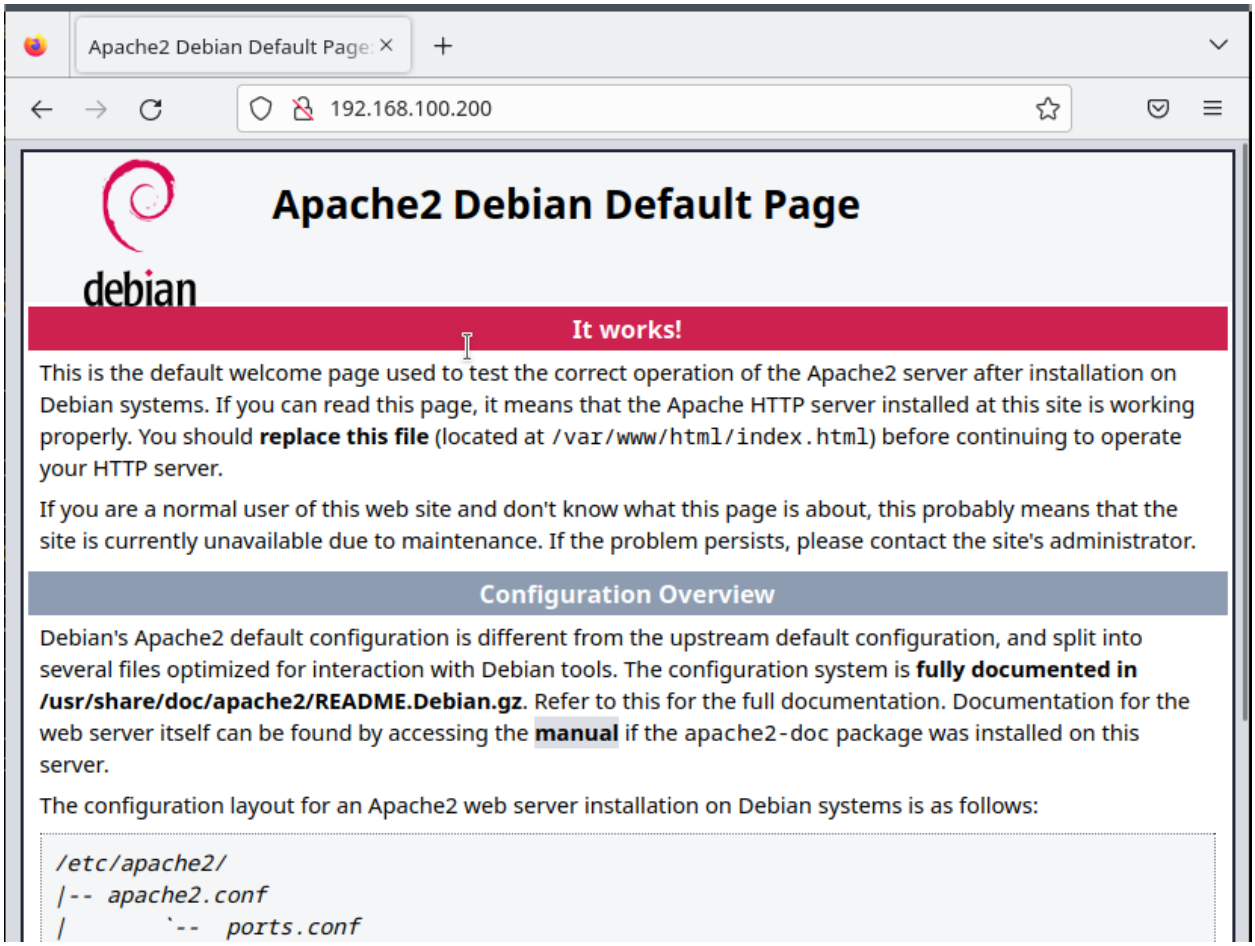
8.3 Проверка

8.3.1 Проверьте, что сервис apache2 работает.

```
$ ps -aux | grep apache2
```

```
root@baget:~# ps -aux | grep apache2
root         189  0.2  1.0   8984  5176 ?        Ss   10:12   0:00 /usr/sbin/apache2 -k start
www-data     190  0.0  0.7   8636  3644 ?        S    10:12   0:00 /usr/sbin/apache2 -k start
www-data     191  0.1  1.0 755160  5200 ?        Sl   10:12   0:00 /usr/sbin/apache2 -k start
www-data     192  0.1  1.0 755160  5200 ?        Sl   10:12   0:00 /usr/sbin/apache2 -k start
root         270  0.0  0.4   8324  2512 ttyS0    S+   10:12   0:00 grep apache2
```

8.3.2 После чего откройте браузер в виртуальной машине, и вбейте в адресное поле ip адрес платы (192.168.100.200) вы должны поучить страницу, как на рисунке ниже



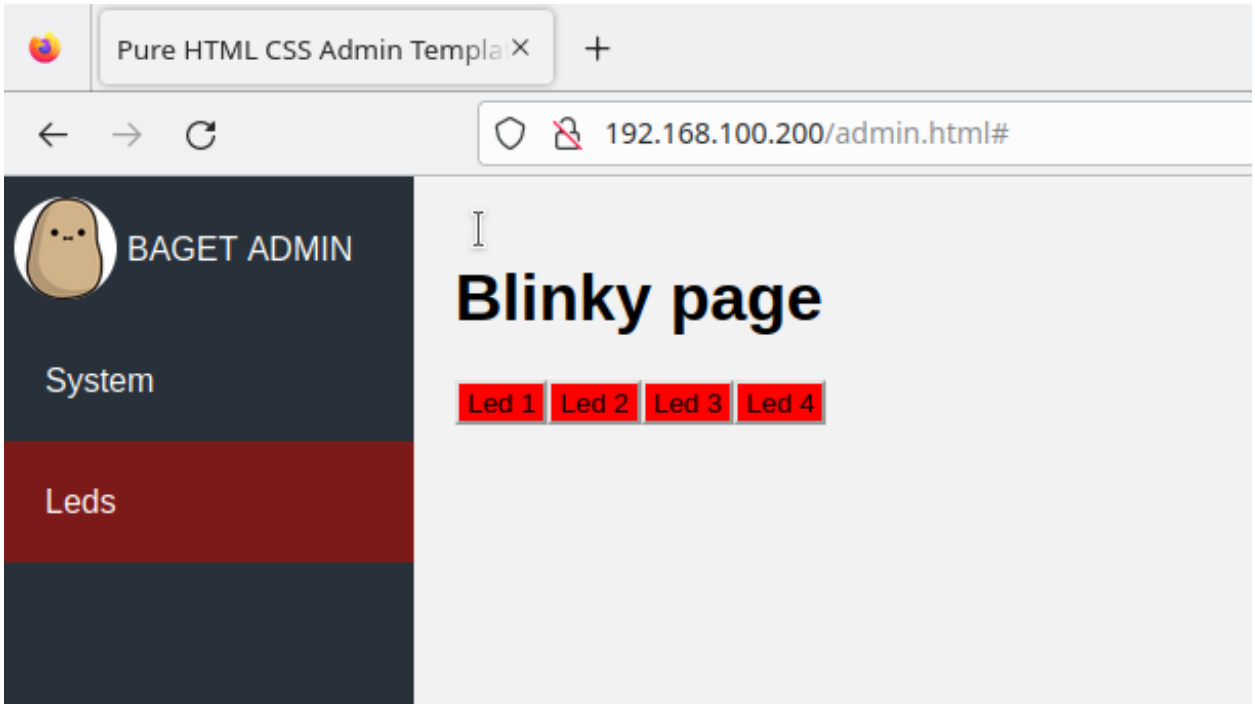
Замечание: если Вы получили результат, что страница не найдена, проверьте, что у ваз стоит префикс http и повторите попытку снова. Помните, если браузер не находит ответа от http, он автоматически переключается на https и пробует найти что-то там. А так как префикс определяет порт,

к которому отправляются запросы, то важно убедиться, что Вы пытаетесь подключиться к нужному порту.

8.3.3 В терминале платы, запустите сокет, для управления светодиодами

```
$ python3 /var/www/led_socekt.py &
```

8.3.4 Откройте в браузере страницу admin.html: 192.168.100.200/admin.html Вы должны увидеть страницу следующего вида



Первый пункт, будет выдавать Вам информацию о плате (свободное место на диске, загруженность процессора, и RAM памяти, IP адреса платы).

Второй пункт позволяет управлять светодиодами, подключённым к первым четырём выводам банка выводов F.

Нужно понимать главную вещь, веб-сервер в системе работает, как ещё один терминал (вспомните, как мы подключались к терминалу платы по ssh, тут нечто похожее). При этом подключение ведётся не от имени суперпользователя (так делать никогда нельзя, если Вы конечно не хотите

сделать себе, и другим большой проход в систему). Мало того, этот «пользователь», не может получить доступа к корневому каталогу платы, для него корневым каталогом в нашем случае, является папка /var/www/.

Так же, Вы можете заметить, что при попытке управлять светодиодами, возникает большая задержка, между тем как Вы нажмёте кнопку, и произойдёт непосредственно переключение. Это связано с тем, что CGI технология не самая быстрая, и в современности применяется FastCGI или uWSGI. Однако их настройка более комплексная, чем CGI, поэтому в рамках данной лабораторной работы не рассматривалась.

8.3.5 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появиться надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

Список литературы

1. Руководство для платы «Багет-ПЛК1-01» [Электронный ресурс] .- URL: https://www.niisi.ru/БАГЕТ-ПЛК1-01_РЭ_v3.3.pdf

2. Описание К5500ВК018 [Электронный ресурс] .- URL: https://www.niisi.ru/Комдив-МК%20описание%20v4.pdf

3. P. J. Salzman The Linux Kernel Module Programming Guide / Peter Jay Salzman, Michael Burian, Ori Pomerantz, Bob Mottram, Jim Huang [Электронный ресурс]. - URL: https://sysprog21.github.io/lkmpg/

4. Навигатор по коду ядра Linux 4.15.18 [Электронный ресурс]. - URL: https://elixir.bootlin.com/linux/v4.15.18/source

5. Документация ядра Linux 4.15 [Электронный ресурс]. - URL: https://www.kernel.org/doc/html/v4.15/index.html

6. Спецификация Device Tree [Электронный ресурс]. - URL: https://www.devicetree.org/specifications/

7. Device Tree для «чайников» [Электронный ресурс]. - URL: https://events.static.linuxfound.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf

8. Руководство по установке Debian [Электронный ресурс]. - URL: https://www.debian.org/releases/stable/armel/

9. Спецификация разъёмов PMOD [Электронный ресурс]. - URL: https://digilent.com/reference/_media/reference/pmod/pmod-interface-specification-1_2_0.pdf