РТУ МИРЭА (ТУ)

Центр проектирования интегральных схем, устройств наноэлектроники и микросистем

Методическое пособие по лабораторным работам

«Багет-ПЛК1-01»

(ОС Linux, Debian 12)



Москва

2025

Оглавление

Введение **4**

1 Установка ОС **11**

1.1 Подготовка rootfs 11

1.2 Настройка rootfs 13

1.3 Проверка 16

1.4 Задание для самостоятельной работы 20

2 Ознакомительная **21**

2.1 Запуск и подключение к устройству 22

2.2 Подключение по сетевому интерфейсу 23

2.3 Настройка доступа по ключу 26

2.4 Аппаратный Hello world 29

2.5 Задание для самостоятельной работы 32

3 Модуль ядра **34**

3.1 Сборка модуля 36

3.2 Настройка платы 40

3.3 Наложенное дерево устройств 43

3.4 Проверка работы модуля 46

3.5 Задание для самостоятельной работы 49

4 Драйвер GPIO **52**

4.1 Встроенные драйверы 53
4.2 Проверка работы кнопки 57
4.3 Проверка работы светодиодов 58
4.4 Собираем свой драйвер 59
4.5 Задание для самостоятельной работы 64

5 Драйвер SPI устройства **66**
5.1 Запуск и подключение к устройству 67
5.2 Подготовка Device Tree Overlay 68
5.3 Собираем драйвер 72
5.4 Задание для самостоятельной работы 76

6 Драйвер I2C устройства **77**
6.1 Запуск и подключение к устройству 78
6.2 Подготовка наложенного дерева устройств 79
6.3 Работа с I2C из user-space 83
6.4 Задание для самостоятельной работы 85

7 Драйвер Serial устройства **86**
7.1 Запуск и подключение к устройству 86
7.2 Подготовка Device Tree Overlay 87
7.3 Тестовое приложение 90

8 Web интерфейс **92**
8.1 Правка rootfs 92
8.2 Запись на SD карту 94
8.3 Проверка 96
8.4 Задание для самостоятельной работы 99

Рекомендуемая литература **101**

А Установка и настройка виртуальной машины **103**

А.1 Настройка гипервизора 103

А.2 Установка гостевой машины 104

А.3 Настройка гостевой машины 109

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 КБайт), кэш-память второго уровня (128 КБайт), буфер трансляции виртуальных адресов на 64 адреса, контроллер динамической памяти DDR3 с поддержкой DDR3L, восемь таймеров-счетчиков, сторожевой таймер, часы реального времени, два контроллера Ethernet 10/100, контроллеры последовательных интерфейсов UART (5 шт.), I2C (3 шт.), SPI (4 шт.), CAN 2.0 (2 шт.), USB 2.0, контроллер SDHCI. 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).

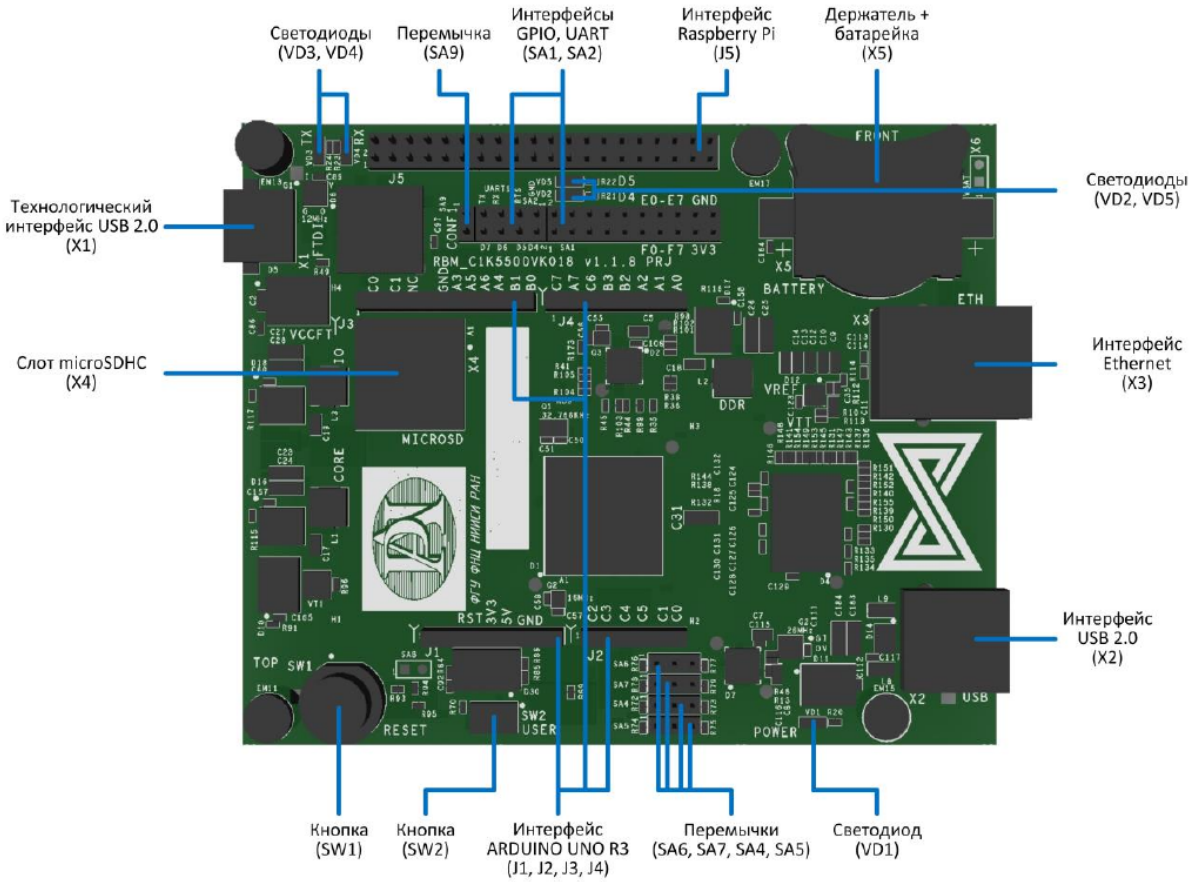


Рисунок 1. Расположение портов ввода-вывода

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.

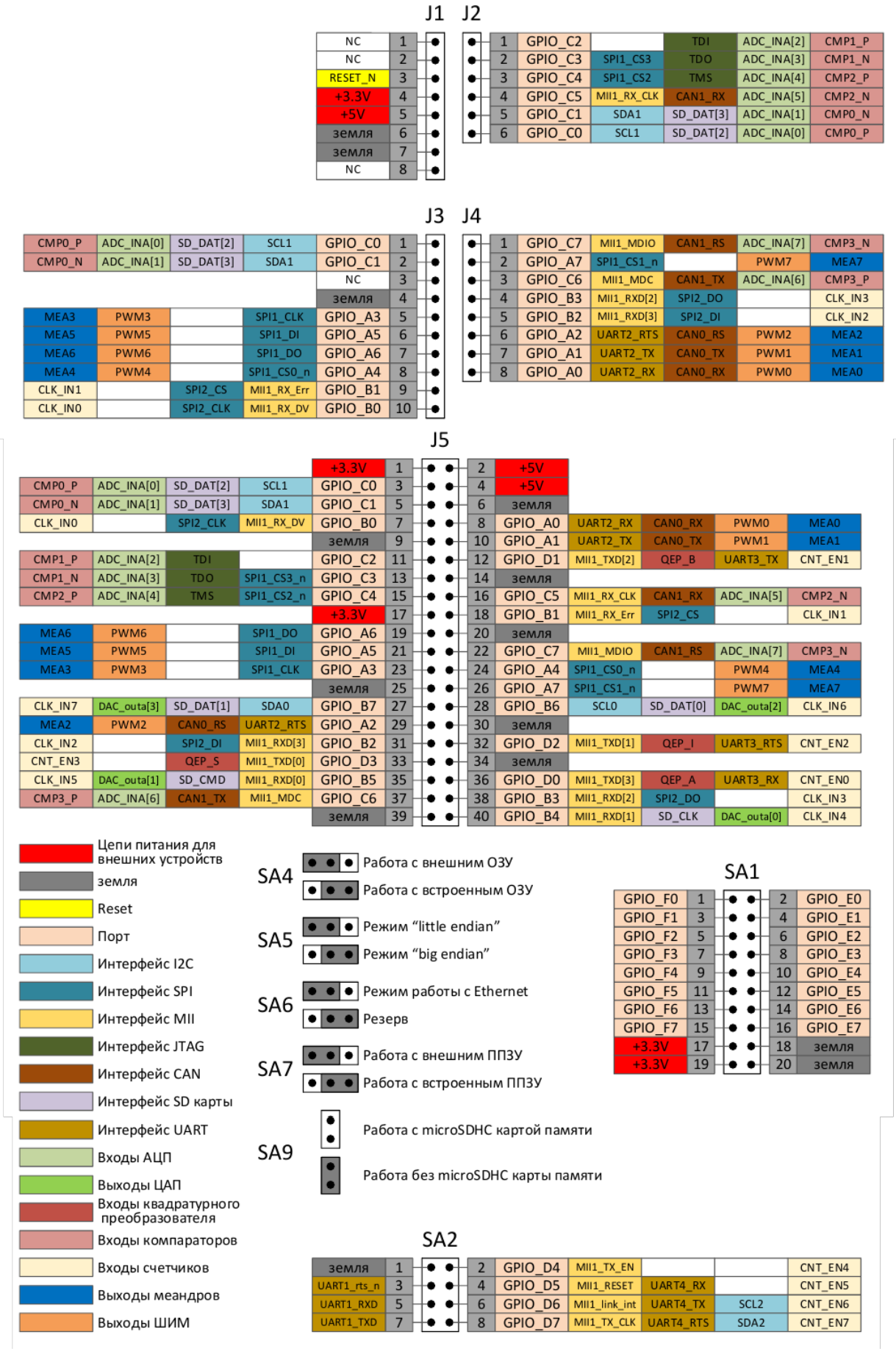


Рисунок 2. Разводка выводов процессора по разъёмам ввода-вывода

Переходная плата

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

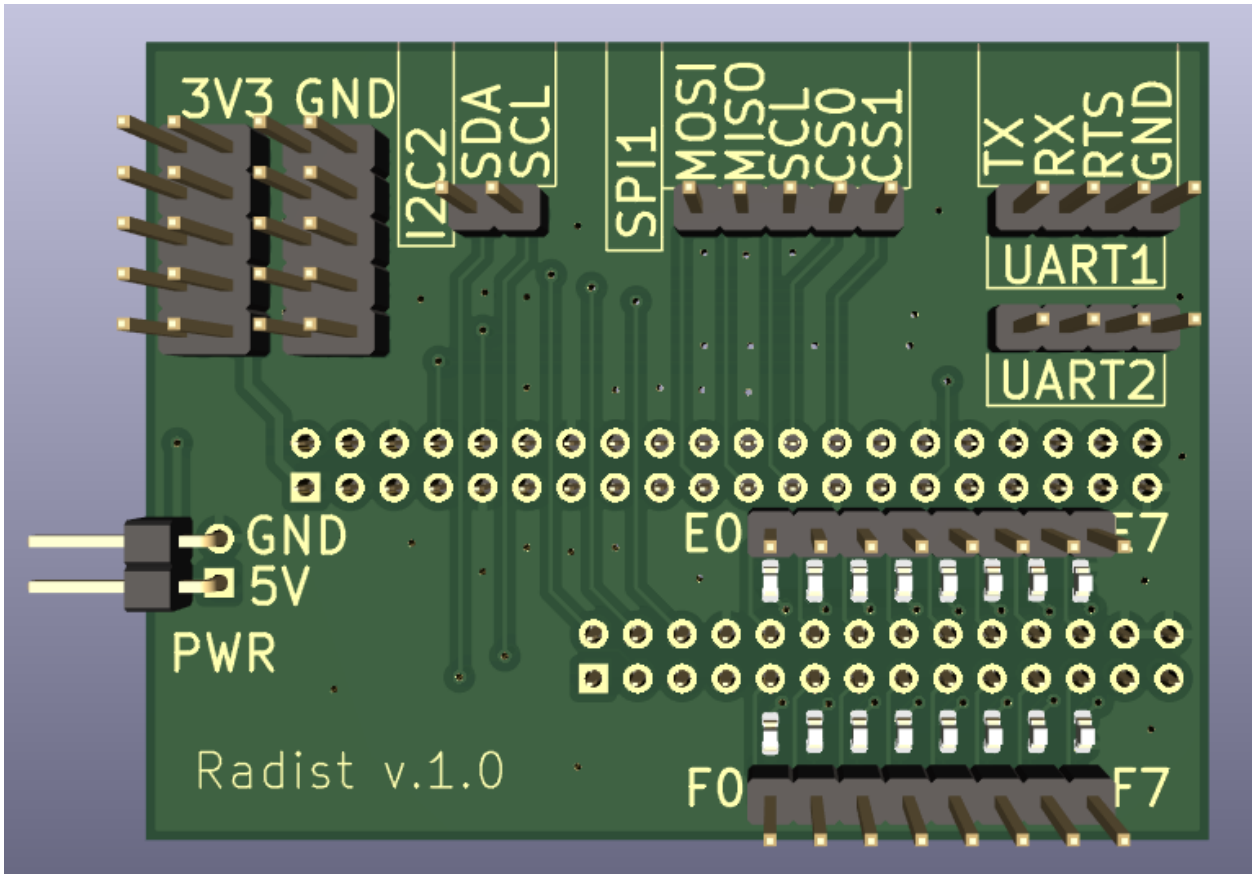


Рисунок 3. Переходная плата

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 интерфейса.

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.

Примеры:

Необходимо ввести в консоли виртуальной машины (ВМ) команду echo

Hello:

```
# echo Hello
```

Необходимо ввести в консоли целевой платы команду echo FOOBAR:

```
$ echo FOOBAR
```

Необходимо ввести в консоли виртуальной машины (ВМ) очень длинную команду (да-да, вечер не обещает быть томным):

```
# echo \
FOOBAR
```

Лабораторная работа № 1

Установка ОС

Цель: Установить дистрибутив Debian 12 на целевую платформу.

Описание: ОС Linux делится 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, содержит скрипты отвечающие за очередность инициализации, запуска сервисов и прочие прикладные вещи. 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, можно найти на wiki проекта Debian (https://wiki.debian.org/ru/Debootstrap).

1.1 Подготовка rootfs

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

1.1.2 При помощи комбинации клавиш на клавиатуре Ctrl+Alt+T откройте консоль.

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 неопределённость (к примеру 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 перейдите в рабочий каталог для создания образа 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 bookworm rootfs
```

--include=A,B,C.. - добавить в сборку указанные пакеты (перечисление должно быть без пробелов после запятых!)

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

--arch=mips64el — указываем целевую архитектуру

bookworm — версия сборки (в данном случае 12)

rootfs — путь к папке назначения, где будут размещены файлы

Замечание:

Время работы данной программы зависит от количества данных и качества подключения к сети интернет. Наберитесь терпения.

1.1.5 Для продолжения установки, понадобиться утилита qemu позволяющая эмулировать различные архитектуры. Создайте символьную ссылку на него в директории rootfs (в противном случае нужно было бы удалить этот файл перед переносом на пату):

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

и перейдите в созданную папку 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 Необходимо добавить источники для установки ПО. Откройте файл со списком источников обновления для редактирования выполнив команду:

```
   # nano /etc/apt/sources.list
```

Проверьте и добавьте, если отсутствуют, следующие строки (обратите внимание, что знак '\' писать нельзя, каждая строка начинается с deb или deb-src и заканчивается non-free-firmware)

```
deb http://ftp.debian.org/debian bookworm main contrib \
```

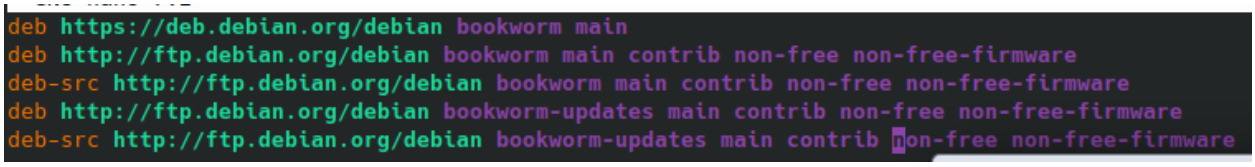
```
non-free non-free-firmware

deb-src http://ftp.debian.org/debian bookworm main contrib \
non-free non-free-firmware

deb http://ftp.debian.org/debian bookworm-updates main contrib \
non-free non-free-firmware

deb-src http://ftp.debian.org/debian bookworm-updates main contrib \
non-free non-free-firmware
```

Файл должен выглядить как на картинке



Сохраните изменения (Ctrl + w) и выйдите из редактора (Ctrl + x).

1.2.3 Обновите список, и установите ряд дополнительных приложений

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

1.2.4 Задайте пароль для пользователя root, для чего вызовите утилиты passwd передав ей в качестве параметра имя 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 завершите ввод нажатием клавиши Enter на клавиатуре.

Затем система попросит повторить пароль, снова введите root и завершите ввод нажатием клавиши Enter на клавиатуре.

1.2.5 Настройте работу сетевого интерфейса со статическим адресом, для этого создайте 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 будет использоваться статический ip адрес, далее идёт конфигурация адресов.

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

1.2.6 Укажите имя компьютера

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

1.2.7 На этом первоначальная настройка файловой системы окончена. Выйдите из chroot.

```
# exit
```

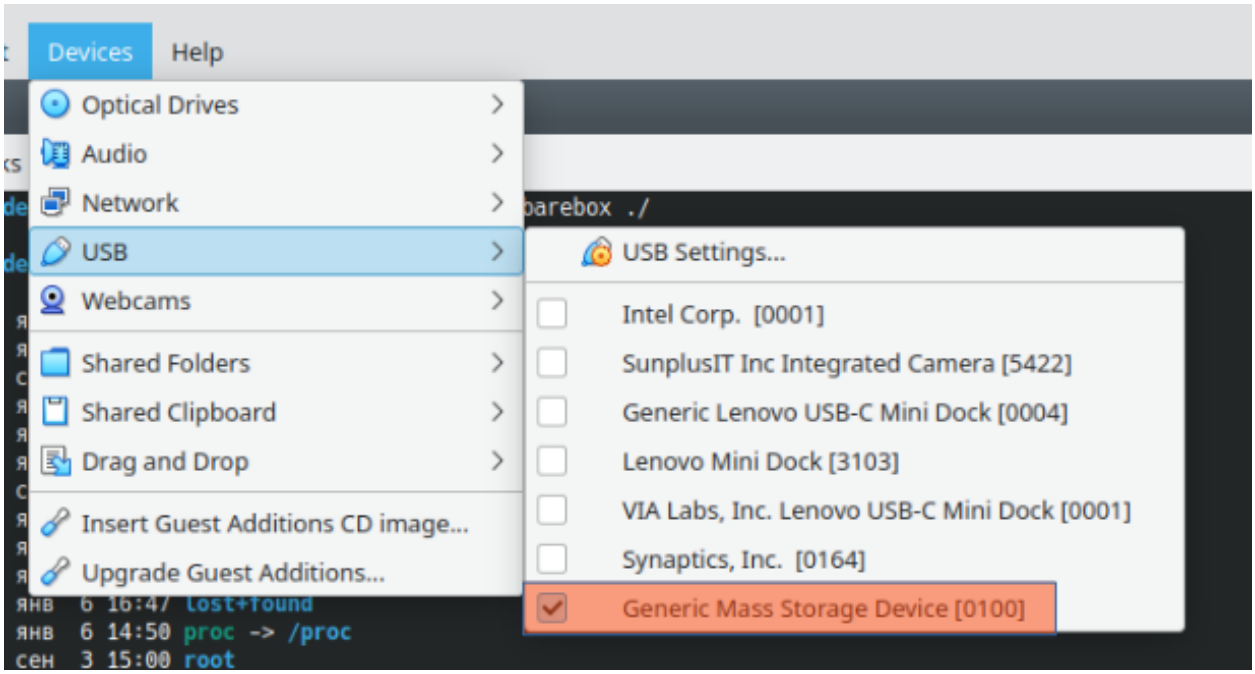
1.2.8 Скопируйте папку barebox из каталога support в корень созданной ранее rootfs

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

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

1.3 Проверка

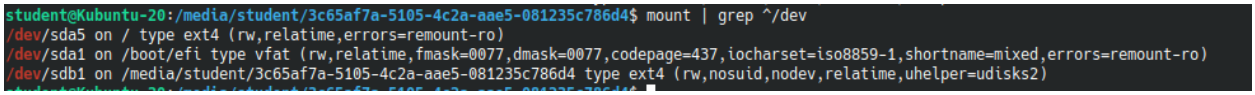
1.3.1 Вставьте SD карту в ПК, и подключите её к виртуальной машине



1.3.2 Необходимо узнать точку монтирования SD карты. Она может отличатся в зависимости от ридера, который используется для работы с картой памяти. Для упрощения ввода, выполните команду в консоли для получения списка примонтированных устройств:

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

Команда grep служит для фильтрация лишнего вывода. На экране будет показан 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, создайте временную системную переменную, которая будет хранить путь к корню на SD карты. Для примера(!) на рисунке выше:

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

Запомните:

> *Эта переменная существует только в том терминале, где был вызван export, в остальных окнах терминала она не существует, если закрыть текущее окно, или понадобится вызвать эту переменную 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/
```

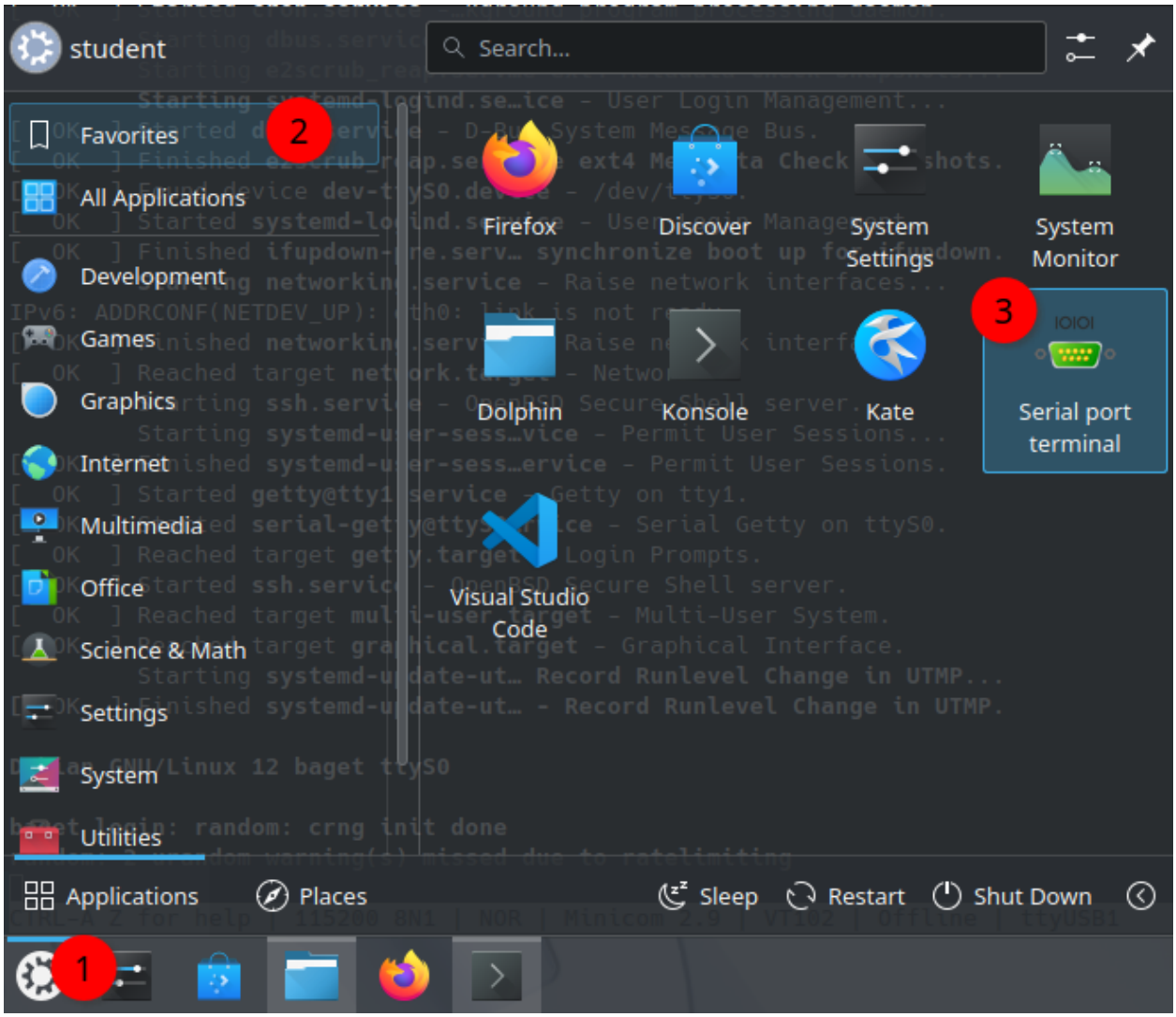
и отмонтируйте SD карту

```
# sudo umount $BAGET_SD
```

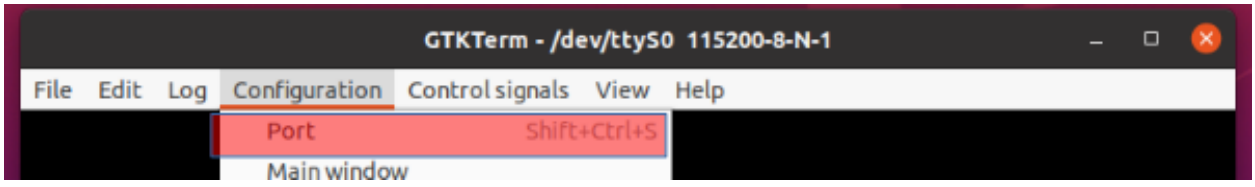
Дождитесь когда команда выполниться, после чего достаньте SD карту и вставьте её в плату. Это важно, так как в ОС Linux запись данных на носитель происходит в фоне, и если отключить накопитель «неправильно» то есть шанс не только не найти данные на носителе, но и повредить файловую систему, с потенциальной потерей всех данных на носителе.

1.3.4 Подключите USB кабелем 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



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



Нажмите кнопку OK. Что бы в будущем не повторять настройку, можно сохранить текущею конфигурацию как начальную. Для этого выберите пункт Configuration->Save configuration.

Или подключитесь при помощи консольной программы minicom, для этого нажмите Ctrl+Shift+T для открытия новой вкладки в окне терминала и выполните команду

```
# minicom -D/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 (так как не правильно была извлечена SD карта). В этом случае, можно наблюдать в терминале множество 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 извлечении SD карты убедитесь, что плата обесточена!**

1.3.7 Введите в качестве логина и пароля root.

Поздравляем, Debian дистрибутив успешно запущен на плате.

1.3.8 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, пока не появится надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

Если к плате подключались через команду minicom, для выхода нажмите Ctrl+A а затем X. Выберите Yes и нажмите Enter.

1.4 Задание для самостоятельной работы

Внесите изменения в копию файловой системы на ПК, и запишите её на SD карту платы. В обновлении должны быть установлены следующие пакеты:

- tree - для вывода в древовидном формате информацию о содержимом текущего каталога, и его подкаталогов
- mc - Midnight Commander, файловый менеджер, консольный файловый менеджер для Linux
- htop - интерактивный консольный монитор ресурсов

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).

Полезные ссылки:

1. Хабр: Коротко об SSH.

2. man7.org: sysfs.

3. EmbeTronicX: sysfs.

4. Kernel Doc: GPIO Sysfs Interface for Userspace.

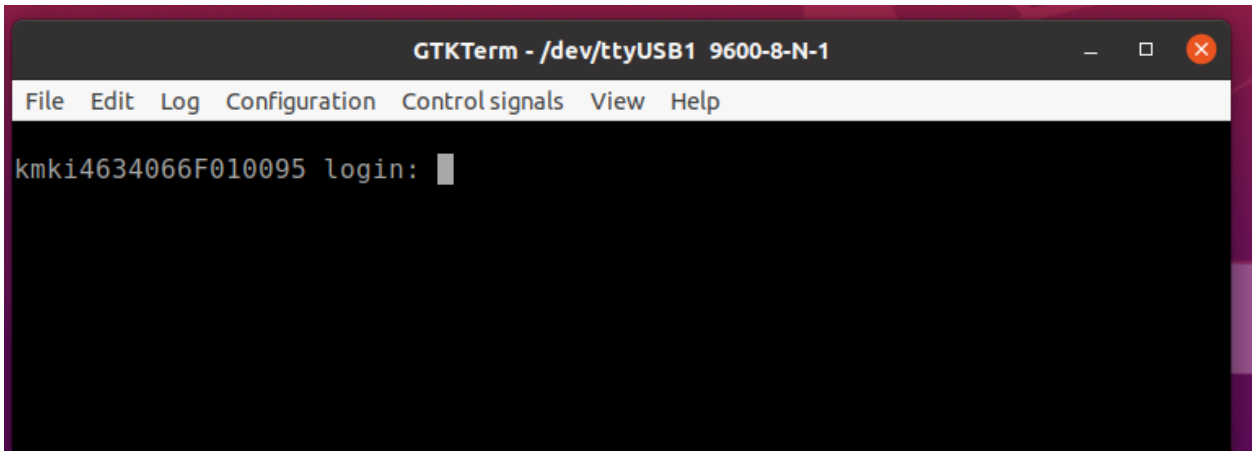
2.1 Запуск и подключение к устройству

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, или откройте консоль в виртуальной машине, и вызовете утилиту minicom.

2.1.4 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. В окне должен появится следующий вывод:



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 -4 addr show dev eth0
```

Проверьте результат, IP адрес платы должен быть 192.168.100.200

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

Результат отображает информацию по выбранному интерфейсу, в данном случае интерес представляет адрес указанный после слова inet. Это IP адрес, который присвоен плате.

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

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

затем установите новый адрес командой

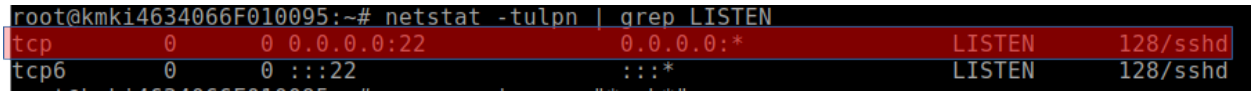
```
$ ip addr add 192.168.100.200/24 dev eth0
```

Для активации изменения необходимо переподключится, 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 вывода, как на картинках выше не появилось, 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 через сетевой интерфейс, 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 интерпретатора команд 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 Проверьте, что сетевое подключение к плате установлено, для этого вбейте команду

```
# ping 192.168.100.200 -c 4
```

Вывод должен быть как в примере ниже (время может отличаться)

```
PING 192.168.100.200 (192.168.100.200) 56(84) bytes of data.
64 bytes from 192.168.100.200: icmp_seq=1 ttl=63 time=1.17 ms
64 bytes from 192.168.100.200: icmp_seq=2 ttl=63 time=1.06 ms
64 bytes from 192.168.100.200: icmp_seq=3 ttl=63 time=1.29 ms
64 bytes from 192.168.100.200: icmp_seq=4 ttl=63 time=1.50 ms

--- 192.168.100.200 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 1.061/1.253/1.497/0.161 ms
```

2.2.7 Введите команду:

```
# ssh netuser@192.168.100.200
```

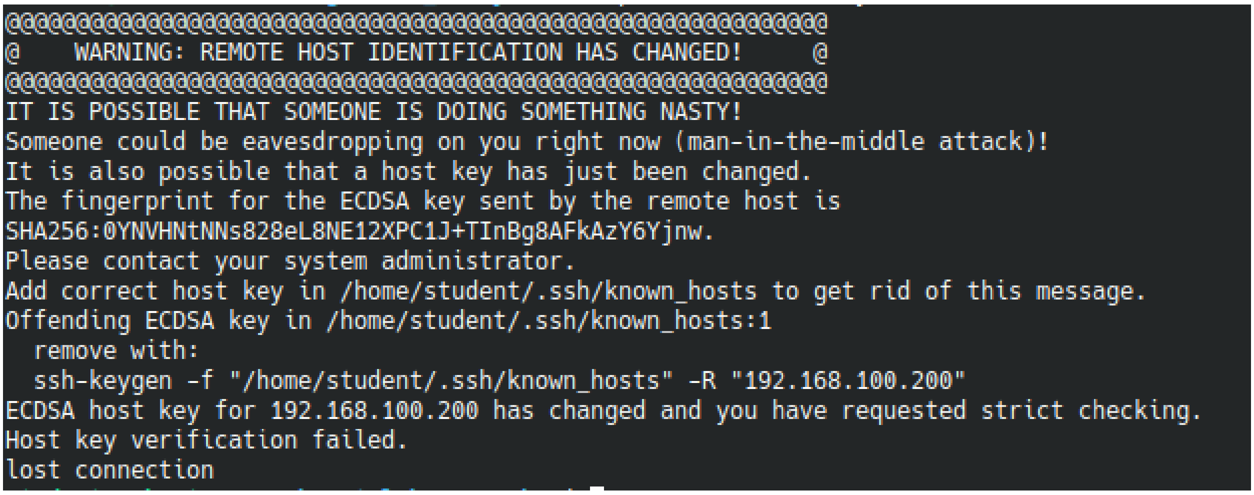
2.2.8 Будет предложено ввести пароль пользователя, вводим тот, что установили в пункте 2.2.3 и получаем доступ к терминалу платы. (в текущем терминале, строка ввода команды будет начинаться с надписи netuser@baget:~$)

Если вместо ввода пароля, в консоли появилась 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 повторно ведите пароль.

Если же получите вывод как на картинке ниже



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:0YNVHNtNNs828eL8NE12XPC1J+TInBg8AFkAzY6Yjnw.
Please contact your system administrator.
Add correct host key in /home/student/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/student/.ssh/known_hosts:1
  remove with:
  ssh-keygen -f "/home/student/.ssh/known_hosts" -R "192.168.100.200"
ECDSA host key for 192.168.100.200 has changed and you have requested strict checking.
Host key verification failed.
lost connection
```

Выполните команду

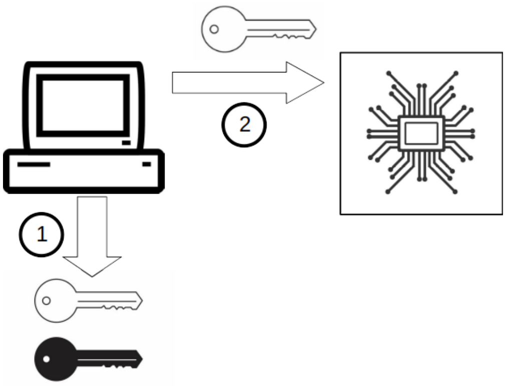
```
# ssh-keygen -f "/home/student/.ssh/known_hosts" -R "192.168.100.200"
```

и вернитесь к пункту 2.2.7

2.2.9 Завершите сеанс, выполнив команду exit.

2.3 Настройка доступа по ключу

SSH подключение можно организовать без необходимости ввода пароля, для этого используются ключи шифрования. При этом создаётся два ключа, один ключ приватный, который остаётся на машине, с которой планируется осуществлять доступ, и второй - публичный, для добавления на сервера (или удалённые системы) к которым необходимо подключится.



2.3.1 В терминале виртуальной машины создайте пару из приватного и публичного ключей:

```
# ssh-keygen -f /home/student/.ssh/id_rsa
```

Будет задан вопрос о пути и имени файла с приватным ключом (публичный будет лежать там же, с тем же именем и дополнительным суффиксом .pub), тут оставляем поля ввода пустым, и жмём кнопку Enter.

Далее будет предложенно указать passphrase. Впишите слово **s3cr3t**, и нажмите Enter. После чего нужно будет вновь повторить его. Это по сути пароль который нужно будет указать, что бы воспользовастся созданным ключом, что позволит дополнительно обезопасить файл-ключ, так как в случае его утечки, злоумышлинику придётся дополнительно узнать эту фразу, перед тем как воспользовастся файл-ключом.

Запомните:

В данном случае, используются стандартные настройки, однако на них можно повлиять, в частности генерируемый ключ на базе RSA шифрования, с длинной ключа в 2048 бит, что считается безопасным, но до 2030 года. В случае, если плата имеет доступ в глобальную сеть, рекомендуется устанавливать длинну ключа не менее 3072 бита:

```
# ssh-keygen -t rsa -b 3072
```

Через ключ t можно выбрать алгоритм шифрования. Информацию о доступных алгоритмах шифрования списке ключей для запуска приложения и т.д. можно получить выполнив следующую команду:

```
# ssh-keygen --help
```

или

```
# man ssh-keygen
```

2.3.2 Скопируйте публичный ключ на плату

```
# scp /home/student/.ssh/id_rsa.pub \
netuser@192.168.100.200:/home/netuser/
```

2.3.3 Перейдите в терминал платы, и допишите ключ в файл с ключами авторизации

```
$ cat /home/netuser/id_rsa.pub >> /home/netuser/.ssh/authorized_keys
```

2.3.4 Удалите файл с публичным ключом

```
$ rm /home/netuser/id_rsa.pub
```

2.3.5 Проверьте подключение к плате по ssh

```
# ssh netuser@192.168.100.200
```

Если теперь запрашивает секретную фразу (passphrase) с указанием пути к приватному ключу: '/home/student/.ssh/id_rsa' значит всё сделано правильно.

2.3.6 Добавьте в кэш сессии виртуальной машины секретную фразу, что бы не вводить её при каждом обращении к плате:

```
# ssh-add
```

впишите секретную фразу из пункта 2.3.1. Теперь для входа по ssh в терминал платы, или копирования данных по ssh программой scp, не нужно будет указывать паролей или секретных фраз.

Запомните:

ssh-agent это менеджер ключей для SSH.

ssh-add позволяет добавить в "память"ssh-agent ключи. Однако эта система работает до перезагрузки виртуальной машины. При новом входе, перед первым взаимодействием с платой нужно будет добавлять ключи в "память"ssh-agent'а.

2.3.7 Финальная проверка

```
# ssh netuser@192.168.100.200
```

После запроса на подключение сразу должен произойти переход в терминал платы, без запроса пароля или секретной фразы.

2.3.8 Завершите сессию по SSH закрыв терминал, или набрав команду.

```
$ exit
```

2.4 Аппаратный Hello world

В этой части будет показано, как можно 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 драйвера.

За формирование наполнения виртуальных файловых систем отвечают

в том числе и модули ядра Linux, которые выполняют роль драйвера, если речь идёт про взаимодействие с периферийным модулями.

2.4.1 Перейдите в каталог /sys/class/gpio:

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

2.4.2 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
```

Первый и последний файлы используются для того, что бы подключать (export) драйвер для управления отдельным выводом в файловую систему sysfs или отключать (unexport).

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 каталога platform. Первая часть имени является базовым адресом периферийного модуля, отвечающего за управление банком выводов. Первому банку выводов A соответствует адрес 0x1B400200, остальные по возрастанию.

Так же обратите своё внимание, что в данной сборке, номера файлов gpiochip уменьшаются(!), при продвижении от выводов в банке A к выводам в банке F.

2.4.3 У выводов общего назначения в рамках ядра Linux нумерация сквозная. Таким образом, можно определить, что вывод подключённый к кнопке SW2 (банк D, вывод 6) находиться под номерам 486 (480 + 6). Выполните следующую команду:

```
$ echo 486 > export
```

2.4.4 Выполните эту же команду для экспорта пары выводов банка F (464 и 465)

2.4.5 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.

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

```
$ echo out > gpio464/direction
$ echo out > gpio465/direction
```

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

```
$ echo 1 > gpio464/value
$ echo 0 > gpio464/value
```

2.4.8 Для взаимодействия с кнопкой менять ничего не нужно. Для удобства отладки запустите 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).

2.4.9 Остановите работу программы watch комбинацией клавиш Ctrl + C

2.4.10 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.

2.5 Задание для самостоятельной работы

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 с задержкой в 1с между

переключениями.

Лабораторная работа № 3

Модуль ядра

Цель: Сборка модуля ядра, работа с device tree.

Описание: В ОС Linux выделяется два пространства: пространство ядра (Kernel Space) и пользовательское пространство (User Space).

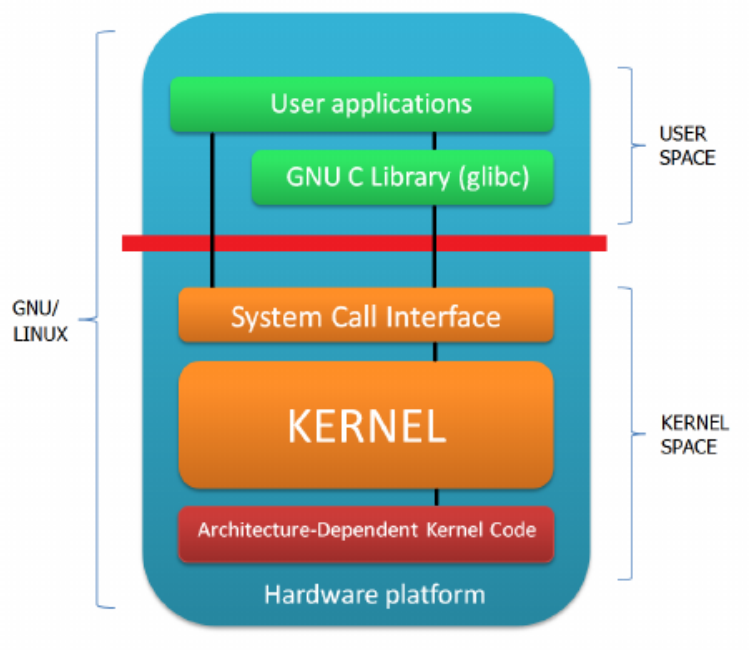
Эти два пространства разделены между собой, и взаимодействие может происходить только при помощи специального системного интерфейса (System Call Interface).

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 ОС Linux выделяют три основных типа устройств:

- символьное устройство — представлены 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, жёсткие диски 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 Добавьте ключи в ssh-agent

```
# ssh-add
```

Впишите фразу заданную в пункте 2.3.1

3.1.5 Создайте рабочий каталог

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

3.1.6 Скопируйте в него рабочие файлы

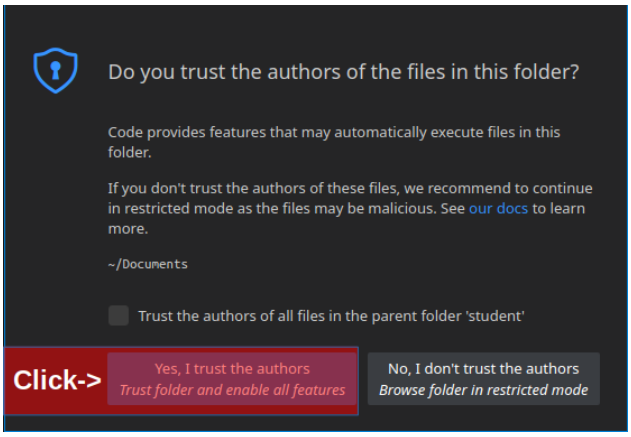
```
# cp -r $BAGET/support/mychar $BAGET/lab_03/
```

3.1.7 Перейдите в рабочий каталог и запустите vscode

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

Замечание:

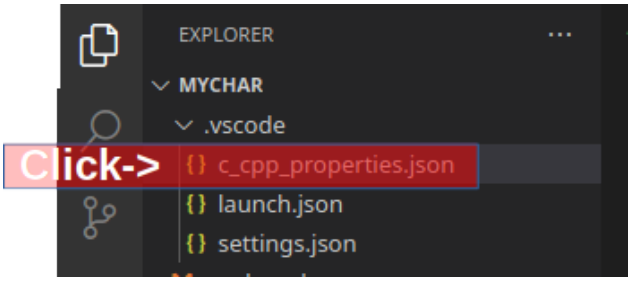
При первом входе, система может спросить о доверенности автора, нужно нажать на кнопку Yes,



3.1.8 Добавьте путей для разрешения части include директив. Откройте .vscode -> c_cpp_properties.json

Замечание:

Если этого файла нет, то необходимо горячими клавишами Shift+Ctrl+P вызвать поле ввода комманд и выбрать C/C++: Edit Configuration (JSON)



3.1.9 Найдите поле includePath, поставьте запятую после "${workspaceFolder}/**" и допишите следующие строки:



для удобства, воспользуйтесь файлом vscode_lines.txt в каталоге support. Ctrl+S для сохранения

3.1.10 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

3.1.11 Изучим содержимое файла mychar_init.c.

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 (так к примеру вызов функции вывода текста через стандартный поток вывода - printf, приведёт к ошибке компиляции, и вместо неё нужно использовать её аналог - printk).

Обратите внимание на структура 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 множества различных драйверов. По той же причине рекомендуеться все структуры и функции объявлять 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.

3.1.12 Посмотрите на содержимое файла mychar_dev.c. В этом файле описан процесс инициализации символьного устройства (функция int srisa_pdrv_chinit(struct platform_device *pdev)), создание файла-заглушку в папке dev для доступа к конкретному аппаратному модулю (в данном случае, модуль ядра рассчитан на наличие только одного объекта, файл доступа будет /dev/srisa_pdrv) а так же реализация API вызовов, адресованных данному устройству (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.

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

3.1.14 Перейдите в терминал (можно открыть вкладку TERMINAL в vscode, если её не видно, выберите в меню Terminal -> New Terminal) и соберите модуль ядра командой

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

3.1.15 В результате компиляции в консоли должен быть 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$
```

В рабочей папке должны появится несколько файлов, в том числе chardev.ko

3.2 Настройка платы

Для работы с модулями в Linux есть ряд команд (доступны только пользователю root или через команду sudo):

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. Требуется указывать только имя модуля, без расширения ko. При этом информация о модуле должна 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 Подключитесь к плате через gtkterm или minicom (/dev/ttyUSB1 скорость 115200).

3.2.2 Для функционирования modporbe необходимо выполнить ряд подготовительных работ. В первую очередь создать каталог для хранения модулей и информации о них:

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

3.2.3 Создайте несколько файлов

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

В данном случае, допустим такой ход, но в общем случае, содержимое этих файлов должно отражать реальность, в частности обратите внимание, на наличие одноимённых файлов в рабочей папке, в которой находится исходный код модуля ядра. При этом пути к модулям, записанные в этих файлах

ведутся от каталога, в котором они расположены. Пример наполнения для modules.order и modules.builtin (не для текущей платы!):

```
kernel/kernel/configs.ko
kernel/fs/binfmt_script.ko
kernel/fs/mbcache.ko
```

3.2.4 Вернитесь в консоль виртуальной машины, и скопируйте 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/
```

3.2.5 Вернитесь в терминал платы, и переместите модуль ядра в системную папку

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

3.2.6 Определите зависимости для модуля,выполнив 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 Проверьте работу модуля, выполнив команду

```
$ modprobe chardev
```

в результате должны увидеть надпись

```
srisa_pdrv: srisa_pdrv_init
```

3.2.8 Посмотрите список загруженных модулей

```
$ lsmod
```

вывод должен быть следующим:

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

Size - показывет сколько оперативной памяти выделено для выбранного модуля,

Used by - сколько программ или модулей используют на данный момент указаный модуль (очевидно показываются только активные приложения и модули).

3.2.9 Выполните команду ls /dev/*. Обратите внимание, что отсутствует файл srisa_pdrv, так как он создаётся, только при вызове функции pdrv_probe. Так как в системе отсутствует привязка модуля к периферийному устройству, функция pdrv_probe не вызывается.

3.2.10 Выгрузите из памяти модуль ядра

```
$ rmmod chardev
```

3.3 Наложенное дерево устройств

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 и устройством служит специальная структура - дерево устройств (device tree). В ней описывается основная информация о каждом устройстве, подключенным к процессорному ядру, а так же могут добавлять параметры,

применяемые модулем ядра при работе с тем или иным устройством. Среди прочего, дерево устройств поддерживает функцию наложения (overlay), что позволяет создавать файлы, вносящие исправление в существующее описание, или дополняющее его, в соответствии с требованиями конкретного проекта, без необходимости пересобирать всё дерево устройств целиком.

3.3.1 Создайте рабочую папку в виртуальной машине для реализации наложенного дерева устройств и перейдите в неё

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

3.3.2 Создайте файл и откройте 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

На данном шаге было создано дерево устройств, которое описывает новое виртуальное устройство, совместимое с драйвером, созданном ранее (обратите внимание на поле compatible в написанном фрагменте, и сравните его значение со значением одноименного параметра в исходном коде модуля). Ключевое слово overlay показывает, что данное дерево устройств должно быть наложено на уже существующее.

3.3.5 Скомпилируйте наложенное дерево устройств, для этого выполните следующую команду

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

Запомните:

> *Компиляция наложенного дерева устройств от обычного не отличается.*

Особенность дерева устройств в том, что скомпилированное дерево устройств можно вернуть в «человекочитаемый код» командой

```
# dtc -@ -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 совместимое с модулем устройство.

3.4.2 Посмотрите последние строки лога загрузки ядра

```
$ dmesg | tail
```

В консоли должны быть следующие строки (значение в квадратных скобках могут отличаться, это значение секунд прошедших с момента запуска ядра операционной системы)

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

Строку с init получали ранее, но теперь после неё есть строка srisa-pdrv: pdrv_probe, которая показывает, что модуль успешно связался с устройством описанным в дереве устройств.

3.4.3 Выполните команду ls /dev/* и обратите внимание, что появился файл srisa_pdrv в выдаче.

3.4.4 Отправьте данные в srisa_pdrv, и проанализируем сообщения в консоли

```
$ 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 выполнив команду:

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

Команда dd может часто встречаться 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 драйвер. Чем больше отправим данных, тем точнее получим значение скорости, однако из-за особенностей реализации считывания в 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 меняться от каждого запуска, но порядок величин должен быть одинаковый.

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

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

Тут опять используеться команда dd, но на этот раз не указывается выходной файл (аргумент of), поэтому появляется возмоэность увидеть вывод драйвера 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 присутствует набор букв, который был сгенерирован драйвером.

3.4.7 Оцените скорость чтения данных, но на этот раз перенаправим вывод в файл, в противном случае придётся долго ждать завершения вывода в консоль запрошенного объёма данных (или прервать его комбинацией клавиш Ctrl+C):

```
$ 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 подождите, пока через терминал будет передан 1 мегабайт данных. Вывод можно прервать комбинацией Ctrl+C.

3.4.8 Выгрузим ядро из памяти

```
$ rmmod chardev
```

Обратите внимание, что есть два вывода, один от модуля, который получали ранее, второй сообщает, что была выполнена функция по отключению от устройства (удалению файла заглушки из папки /dev).

3.4.9 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появится надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

3.5 Задание для самостоятельной работы

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 запро-

сил на считывание 1Мбайт, в ответ он должен получить 10 байт: HelloHello. При этом символ переноса каретки (\r) должен быть исключён.

Тут понадобится 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 в проверке работы драйвера. Как один из вариантов, создайте текстовый файл, и впишите в него текст для передачи в драйвер. Через команду dd передаёте этот файл в драйвер, и второй командой dd считываете данные в другой файл, после чего можете сравнить два файла (исходный и полученный) командой diff <path1> <path2>

Пример для записи данных в драйвере с применение структуры kfifo.

```
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 с использованием структуры kfifo.

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

Полезные ссылки:

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

5. Kernel GIT: Input Event Codes

6. Kernel Doc: GPIO Descriptor Consumer Interface

4.1 Встроенные драйверы

Наверное, один из самых распространённых способов использования выводов общего назначения (GPIO) это 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 операционной системы Linux есть готовые драйверы для использования GPIO как кнопок или индикаторов. В данном разделе будет показано 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 Откройте терминал комбинацией клавиш Ctrl+Alt+T

4.1.4 Добавьте ключи в ssh-agent

```
# ssh-add
```

Впишите фразу заданную в пункте 2.3.1

4.1.5 Для начала подготовьте рабочую папку, для разработки наложенного дерева устройств для системы:

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

4.1.6 Создайте и откройте для правки файла

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

4.1.7 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, подключённому 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 текущей сборки ядра триггеры не были активированы (каждый триггер по сути это отдельный модуль ядра), поэтому 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 передаётся под управление драйвером. Входное значение состоит из трёх элементов:

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

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

4.1.9 Скомпилируйте наложенное дерево устройства, командой

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

Замечание:

При компиляции будет выдано предупреждение, его можно проигнорировать.

4.1.10 Скопируйте полученный файл на плату

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

4.1.11 Откройте программу gtkterm, и подключитесь к порту /dev/ttyUSB1.

4.1.12 Переместите скопированный файл в папку barebox

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

4.1.13 Откройте скрипт barebox.sh для редактирования

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

4.1.14 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

4.1.15 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 вывод не совпадает с выводом выше, то скорее всего была допущена ошибка в dts файле. Проверьте его на наличе опечаток, и повторите шаги с пункта 4.1.9.

4.2 Проверка работы кнопки

Работа с кнопкой ведется через детектирования событий генерирующихся входным устройством. В этом плане, схожим образом происходит взаимодействие с клавиатурой, мышкой или сенсорным экраном.

4.2.1 Воспользуемся специальной утилитой для анализа событий - evtest

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

Как только она запустится, то передаст информацию о проверяемом источнике ввода в консоль:

```
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. В консоли появится 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 Прервите работу программы, нажав Ctrl+C.

4.2.4 Дополнительно посмотрите, 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 видели ранее, инструмент evtest занимается преобразованием (парсингом) этого потока, в человекочитаемый вид.

4.2.5 Прервите работу нажав Ctrl+C.

4.3 Проверка работы светодиодов

Светодиоды управляются через виртуальную файловую систему sysfs.

4.3.1 Перейдите в следующий каталог на плате

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

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

```
$ ls -l
```

4.3.3 Для включения светодиод отправьте 1 в файл brightness, 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
```

4.4 Собираем свой драйвер

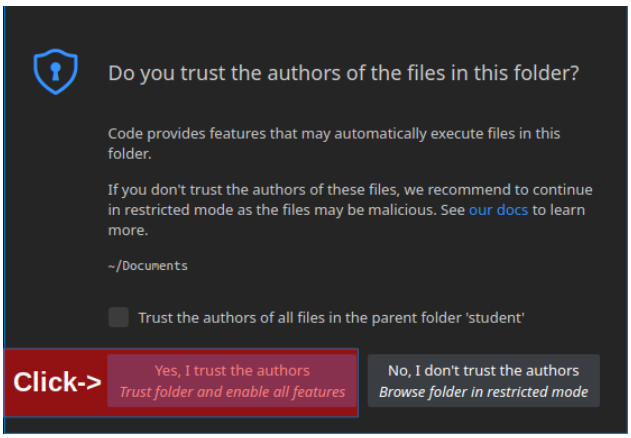
Попробуем теперь использовать вывод общего назначения внутри собственного драйвера, что может часто понадобится, при работе с внешними аппаратными модулями. К примеру, для управления сигналом сброса или для переключения между режимами и т. д.

4.4.1 Скопируйте код демонстрирующий работу с GPIO в пространстве ядра. Вернитесь в консоль виртуальной машины, и выполните следующую команду

```
# cp -r $BAGET/support/gpio_driver $BAGET/lab_04/driver
```

4.4.2 Изучите пример кода модуля ядра

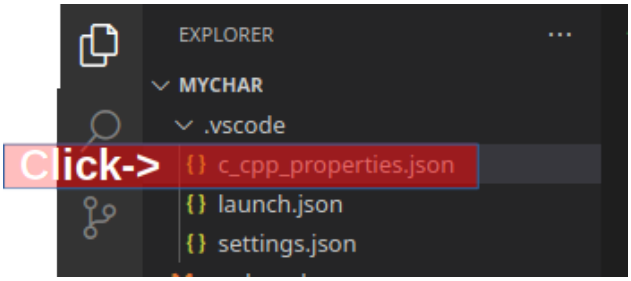
```
# cd $BAGET/lab_04/driver; code .
```



Замечание:

При первом входе, система может спросить о доверенности автора, нужно нажать на кнопку Yes,

4.4.3 Добавьте путей для разрешения части include директив. Откройте .vscode -> c_cpp_properties.json



Замечание:

Если этого файла нет, то необходимо горячими клавишами Shift+Ctrl+P вызвать поле ввода команд и выбрать C/C++: Edit Configuration (JSON)

4.4.4 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.txt в папке support. Ctrl+S для сохранения.

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

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

4.4.6 Посмотрите содержимое файла gpio_temp.c. В нём описан процесс подключения драйвера к выводу.

4.4.7 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.

4.4.8 Перейдите в терминал (можно открыть вкладку TERMINAL в vscode, если её не видно, выберите в меню Terminal → New Terminal) и соберите модуль командой:

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

4.4.9 Проверьте значение системной переменной BRD_PASS

```
# echo $BRD_PASS
```

если вывод был пустой, проинициализируйте переменную

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

4.4.10 Скопируйте модуль на плату

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

Замечание:

Попробуйте сами разобраться, как в Makefile описан процесс копирования файла на плату. Это может Вам помочь, при выполнении самостоятельных работ.

4.4.11 Внесите изменения в наложенное дерево 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 указываем информацию о выводе, которым нужно управлять. Первая часть

(yellowled...) должна совпадать с той меткой, что указана в исходном коде, на 62 строке!

4.4.12 Сохраните изменения Ctrl+S и закройте редактор Ctrl+Q.

4.4.13 Скомпилируйте файл dts, для этого выполните следующую команду

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

4.4.14 Скопируйте полученный файл на плату

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

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

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

4.4.16 Перезагрузите плату командой reboot.

4.4.17 После перезагрузки, подгрузите модуль командой

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

Если не было допущено ошибок, то в течении 20 секунд плата будет попеременно включать светодиоды F0 и F1

4.4.18 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появится надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

4.5 Задание для самостоятельной работы

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, 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.

Проверку работы SPI интерфейса будет проводится при помощи PMOD модуля компании Digilent PmodALS на которой установлен датчик освещённости, на базе микросхемы TEMT6000X01 подключенный к АЦП ADC081S021.

Полезные ссылки:

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

- Vishay Semiconductor's TEMT6000X01
- Texas Instruments ADC081S021

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).

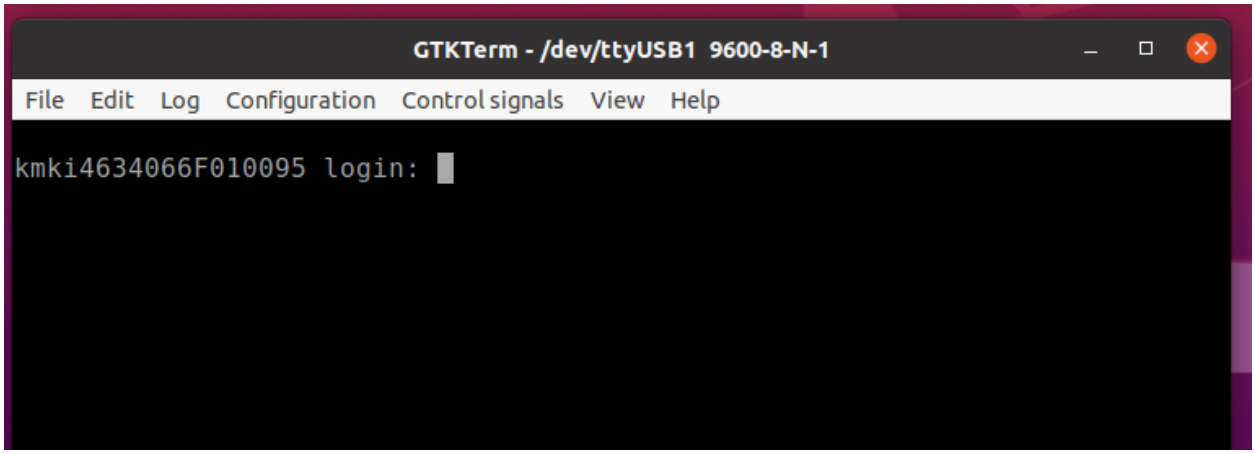
5.1.3 Добавьте ключи в ssh-agent

```
# ssh-add
```

Впишите фразу заданную в пункте 2.3.1

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

5.1.5 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. Должна появиться надпись следующего вида:



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 Создайте рабочую папку в виртуальной машине для наложенного дерева устройств и перейдите в неё

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

5.2.2 Создайте новый файл

```
# kate of_spi1.dts
```

и запишите 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>;
         };
      };

   };
};
```

Данное дополнение состоит из нескольких фрагментов, так как переписать нужно две разные ноды, однако изменения эти должны быть синхро-

низированы 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, вводиться два дополнительных параметра. Первый 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 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 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 дерева устройств 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 групп, 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 Скомпилируйте файл и скопируйте 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 скрипт 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 добавив в начало символ #)

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

Сохраните изменения и выйдите из редактора (Ctrl+S, Ctrl+X)

5.2.8 Перезагрузите плату командой reboot.

5.2.9 После перезагрузки войдите в систему (root/root). И выполните команду

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

Вывод должен показать два имени: spi0 (это контроллер qspi flash-памяти) и spi1 (тот что активировали при помощи наложенного дерева устройств). Если второго имени нет, значит была допущена ошибка в dts файле, или в barebox.sh скрипте. Так же изучите вывод платы во время загрузки.

5.3 Собираем драйвер

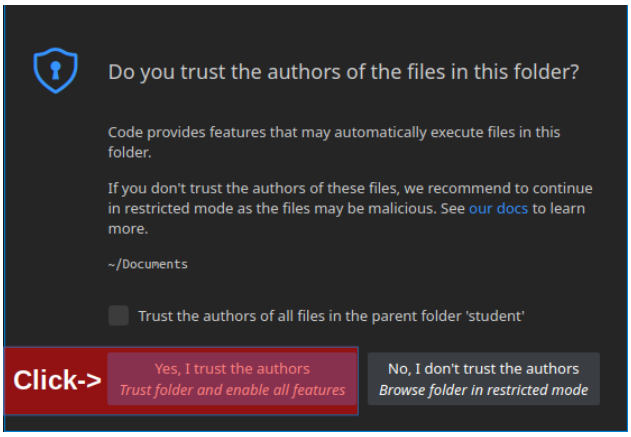
В Linux есть драйвер для управления spi контроллером из пространства пользователя, он называется spidev. Однако у него есть ряд недостатков, которые мешают использовать его в боевых проектах. В системе целевой платы этот драйвер отсутствует, и посмотреть на его работу не получится. Поэтому сразу же начнём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

5.3.1 Подготовьте папку для сборки драйвера, перейдите в неё и откройте в редакторе vscode

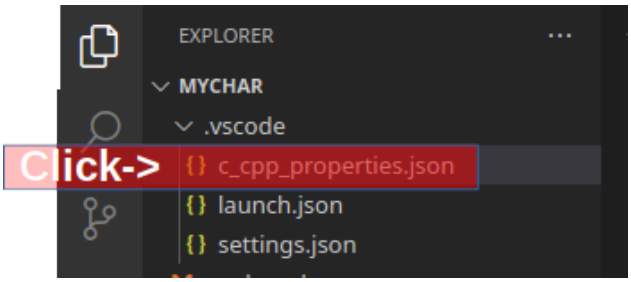
```
# cp -r $BAGET/support/spi_driver $BAGET/lab_05/driver
# cd $BAGET/lab_05/driver; code .
```

Замечание:

При первом входе, система может спросить о доверенности автора, нужно нажать на кнопку Yes,



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



Замечание:

Если этого файла нет, то необходимо горячими клавишами Shift+Ctrl+P вызвать поле ввода команд и выбрать C/C++: Edit Configuration (JSON)

5.3.3 Допишите в поле includePath следующие строки:



для удобства, воспользуйтесь файлом vscode_lines.txt 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 данном случае 1, так как 0 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 можно получить, изучив ссылки приведённые в начале лабораторной работы.

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

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

5.3.6 Проверьте значение системной переменной BRD_PASS

```
# echo $BRD_PASS
```

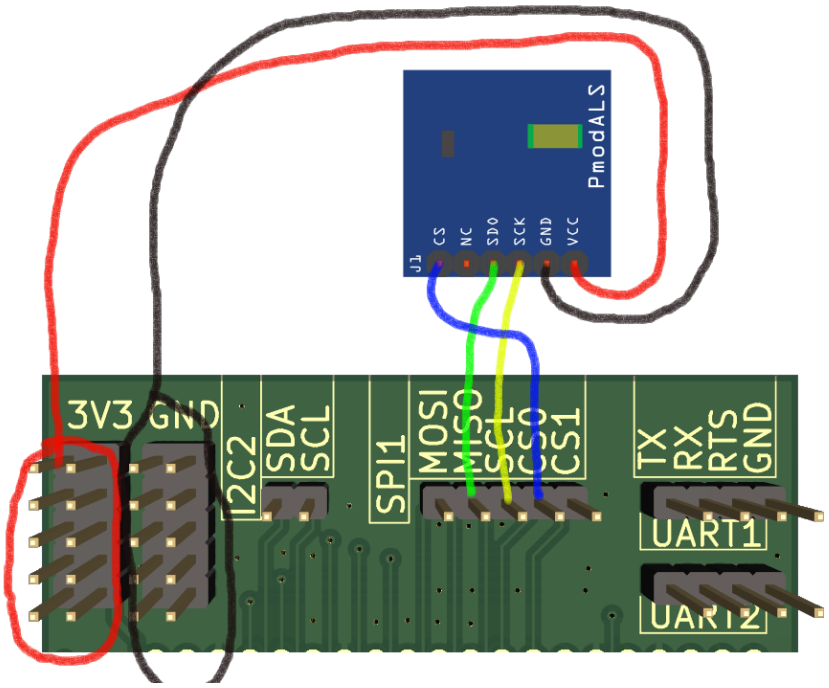
если вывод был пустой, проинициализируйте переменную

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

5.3.7 Скопируйте модуль на плату

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

5.3.8 Подключите модуль PmodALS как показано на рисунке



5.3.9 Перейдите в терминал платы, и активируйте драйвер

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

5.3.10 В терминале должна появится информацию о том, что драйвер зарегистрирован, и 10 раз считано «сырое» значение освещённости с датчика.

5.3.11 Выгрузите драйвер командой

```
$ rmmod spi_temp
```

5.3.12 Выключите плату, для чего в начале введите команду

```
$ poweroff
```

дождитесь, как появится надпись

```
reboot: System halt
```

после чего отключите USB кабель от ПК или платы.

5.4 Задание для самостоятельной работы

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

В этом 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 GY-87 предсталяющем собой навигационный модуль с 10-ю степенями свободы (3-х осевой гироскоп и 3-х осевой акселерометр MPU6050, 3-х осевой датчик магнитного поля HMC5883L и барометр BMP180).

Полезные ссылки:

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

- Описание модуля GY-87
- TDK MPU6050: описание
- TDK MPU6050: конфигурационные регистры
- Honeywell HMC5883L: описание
- Bosch BMP180: описание

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).

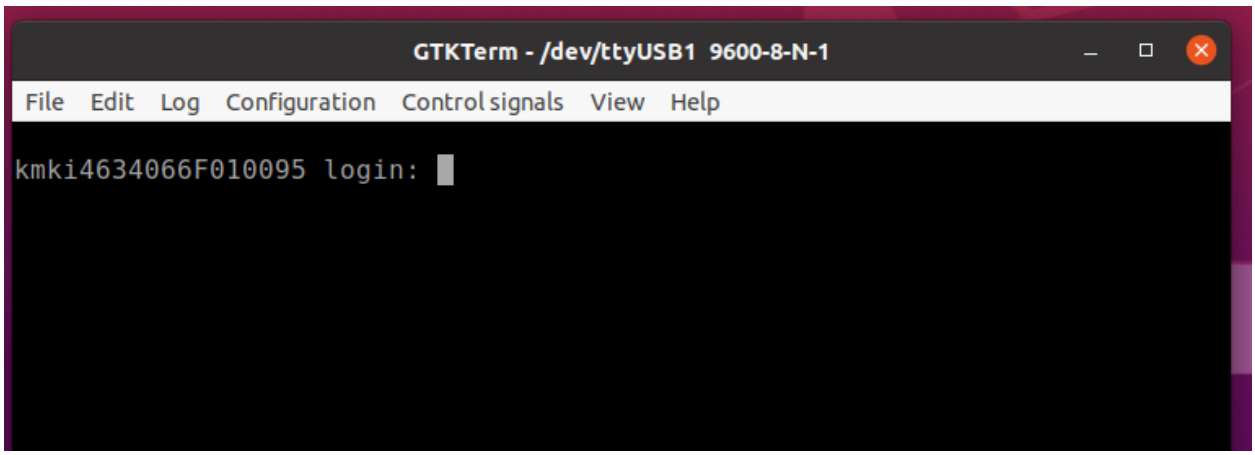
6.1.3 Добавьте ключи в ssh-agent

```
# ssh-add
```

Впишите фразу заданную в пункте 2.3.1

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

6.1.5 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре. Вывод в терминале должен быть похож на тот, что на рисунке ниже:



Введите логин root и пароль root.

6.2 Подготовка наложенного дерева устройств

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

Процессор на плате содержит три контролера I2C, однако воспользоваться возможно только одним, так как выводы двух оставшихся заняты контроллером карты памяти (sdhci), и при их назначении на I2C интерфейс, ОС Linux не сможет загрузится.

6.2.1 Создайте рабочую папку в виртуальной машине для создания наложенного дерева устройств и перейдите в неё

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

6.2.2 Создайте новый файл

```
# kate of_i2c2.dts
```

и запишите 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>;
         };
      };
   };
};
```

Первый фрагмент, это дополнение к ноде описывающей банк выводов D. В этом дополнении создается конфигурация, 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, вводится 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 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 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 Скомпилируйте и отправьте 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 Исправьте файл 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
```

Сохраните изменения и выйдите из редактора (Ctrl+S, Ctrl+X).

6.2.7 Перезагрузите плату командой reboot.

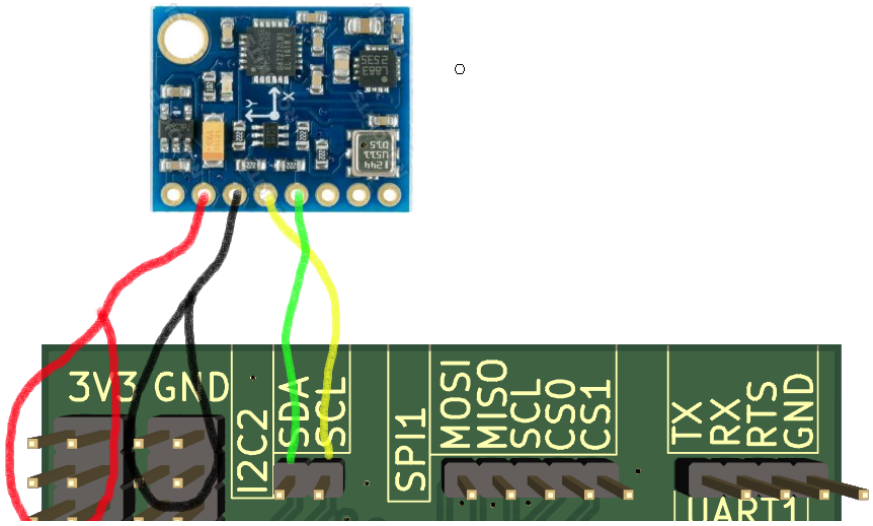
6.2.8 После перезагрузки выполните команду

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

В выдачи должен присутствовать файл i2c-0. Если его нет, выполните команду dmesg на целевой платформе, и изучите лог загрузки на предмет ошибок инициализации I2C контроллера, при их отсутствии, перезагрузите плату, и изучите лог работы загрузчика, на пример ошибок в наложенном дереве устройств.

6.3 Работа с I2C из user-space

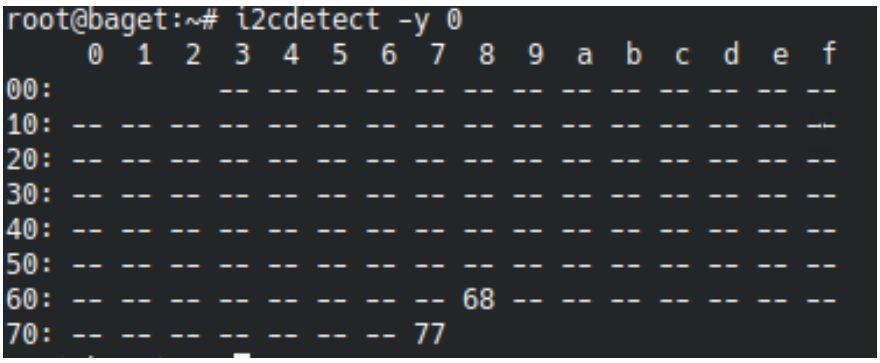
6.3.1 Подключите модуль HW-290 к плате, как показано на рисунке



6.3.2 Воспользуйтесь командой i2cdetect для поиска адресов подключенных к выбранной I2C шине устройства:

```
# i2cdetect -y 0
```

где 0 это индекс i2c адаптера, вспомните что выдала команда ls /dev/i2c*.



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

6.3.3 Скопируйте пример программы работающей с I2C и откройте 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 Скомпилируйте приложение

```
# make
```

6.3.5 Если компиляция прошла успешно, отправьте полученное приложение на плату:

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

6.3.6 Запустите 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 самостоятельной работы

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 данных об измеренном ускорении по осям 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).

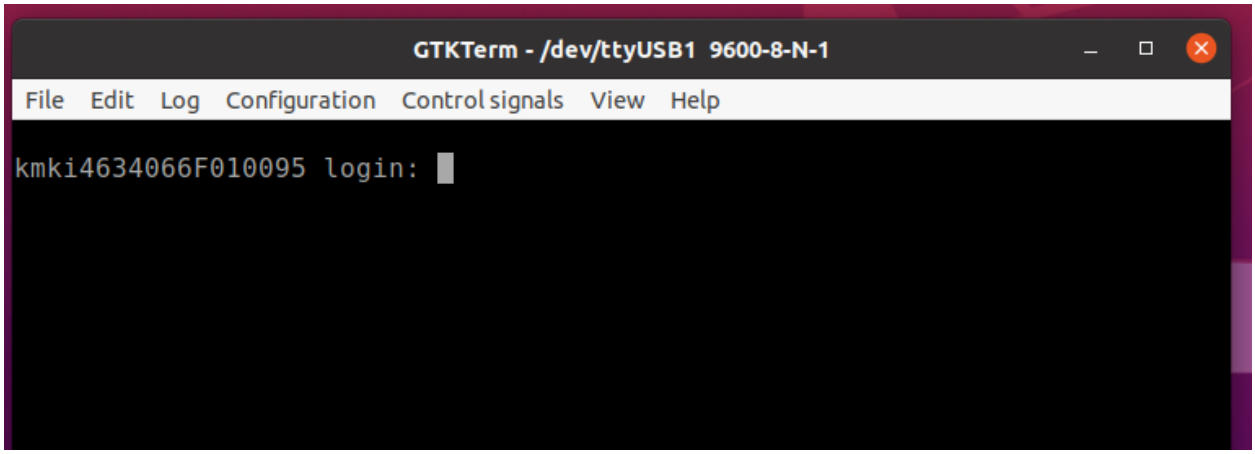
7.1.3 Добавьте ключи в ssh-agent

```
# ssh-add
```

Впишите фразу заданую в пункте 2.3.1

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

7.1.5 Если в окне терминала нет текста, нажмите клавишу Enter на клавиатуре.



Введите логин root и пароль root.

7.2 Подготовка Device Tree Overlay

7.2.1 Создайте рабочую папку в виртуальной машине для device-tree overlay и перейдите в неё

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

7.2.2 Создайте новый файл

```
# kate of_serial2.dts
```

и запишите 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 Скомпилируйте фрагмент, для этого выполните 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 скрипт 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 войдите в систему (root/root). И выполните команду

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

папка должна ссылаться на устройство с адресом активированного контроллера, если это не так, значит была допущена ошибка в описании наложенного дерева устройств, или при редактировании файла barebox.sh. Изучите содержимое файлов и лог загрузки.

```
/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 Перейдите в рабочий каталог, и запустите vscode

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

7.3.3 Изучите файл main.c. В нём только две функции. Одна для инициализации UART контроллера, в которой отключается вся предобработка отправляемых и принимаемых данных операционной системой, что позволяет передавать различные данные через интерфейс без потерь. Так же тут настраивается скорость передачи и приёма.

В основном коде создаётся два буфера, для приёма и отправки данных. Затем один буфер инициализируется начальными 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 не успеть 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
```

В консоли должно появиться сообщение 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, будет создан простой web сервер для мониторинга ресурсов, и управления железом удалённо. Данный проект покажет, 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.

Полезные ссылки:

1. Apache off. page.
2. Apache Tutorial: Dynamic Content with CGI
3. ReadHead Doc: CGI.

8.1 Правка rootfs

В данной работе будет производиться работа с сервером Apache2 и CGI интерпретатором. В современной WEB разработке предпочтение отдают

FastCGI или uWSGI в связке с nginx сервером, исходя из вопросов быстродействия. Однако настройка подобных систем более комплексная, и не относиться к вопросу проектирования встраиваемых систем.

8.1.1 Перейдите в папку первой лабораторной работы, для добавления необходимых компонентов

```
# cd $BAGET/lab_01
# sudo chroot ./rootfs
```

8.1.2 Установите 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
```

8.1.3 Активируйте модуль обработки cgi скриптов apache сервера

```
# a2enmod cgi
```

8.1.4 По-умолчанию, cgi скрипты ищутся в папке /usr/lib/cgi-bin, исправьте путь поиска на /var/www/cgi-bin, для чего откройте файл описания сайта:

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

и внутри ноды VirtualHost допишите

```
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).

8.1.5 Завершите настройку rootfs

```
# exit
```

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

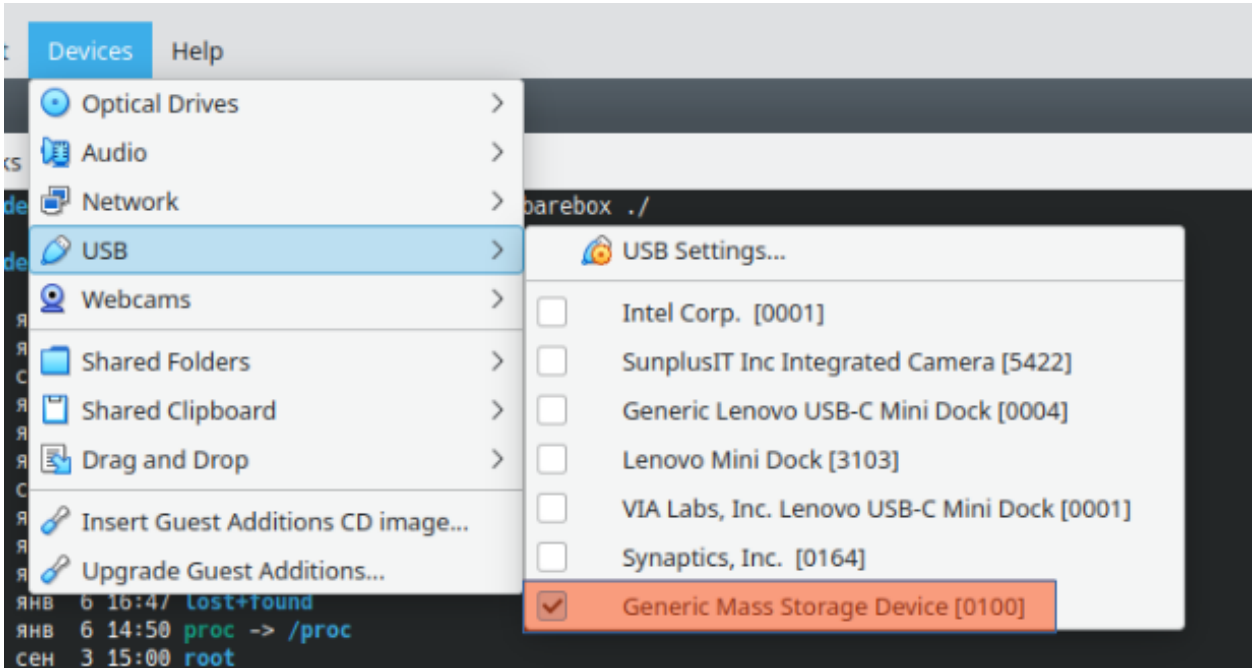
```
# sudo cp -r $BAGET/support/web/* ./rootfs/var/www/
```

8.1.7 Измените владельца для папки с содержимым сайта

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

8.2 Запись на SD карту

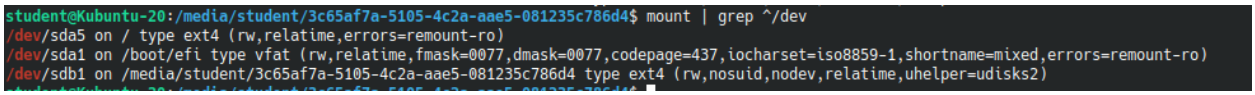
8.2.1 Вставьте SD карту в ПК, и подключите её к виртуальной машине



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

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

Вывод будет примерно как на рисунке



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

Для упрощения ввода, создайте временную системную переменную, которая будет содержать путь к папке, 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 обновленную файловую систему на карту памяти

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

и отмонтируйте 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, 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!**

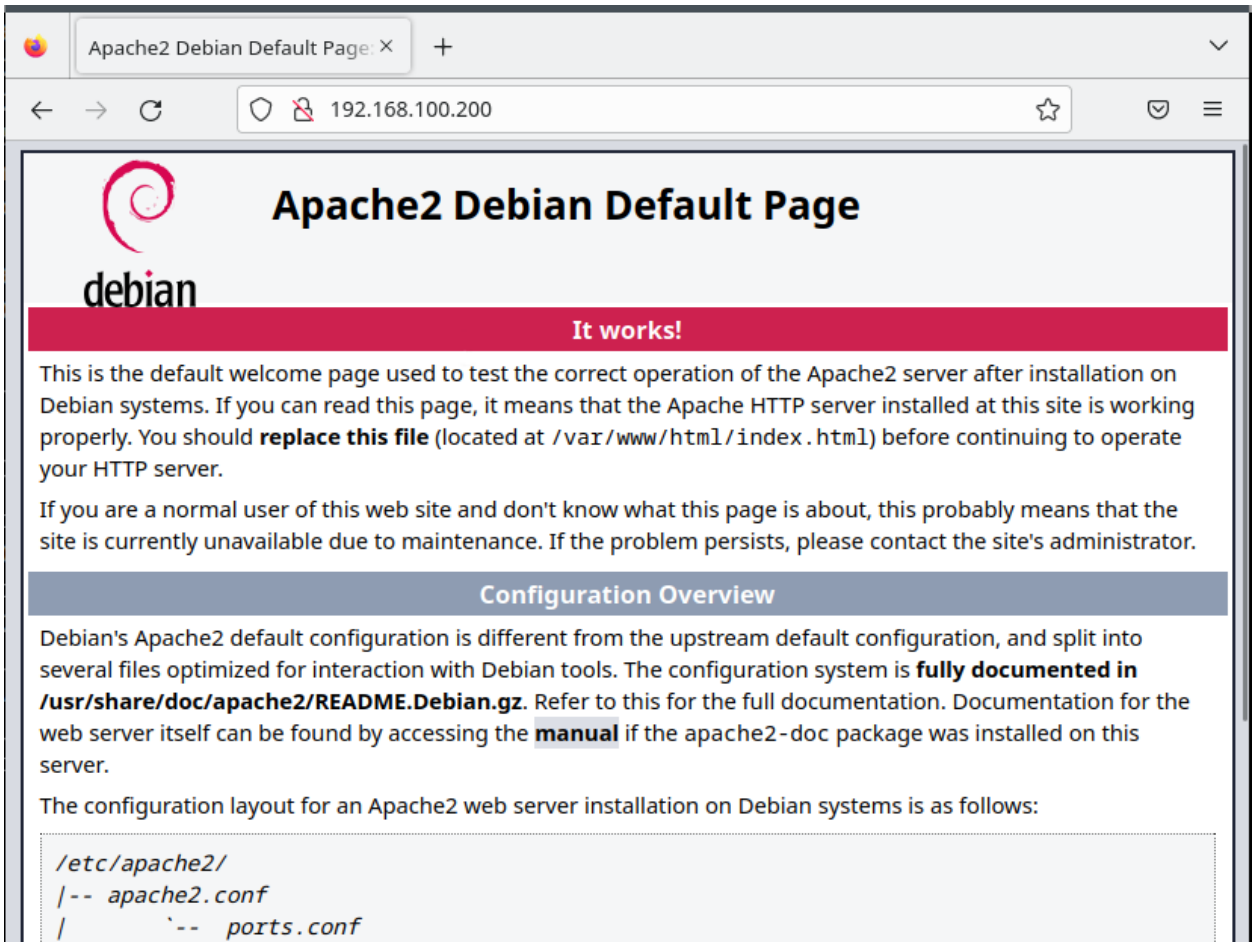
8.3 Проверка

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

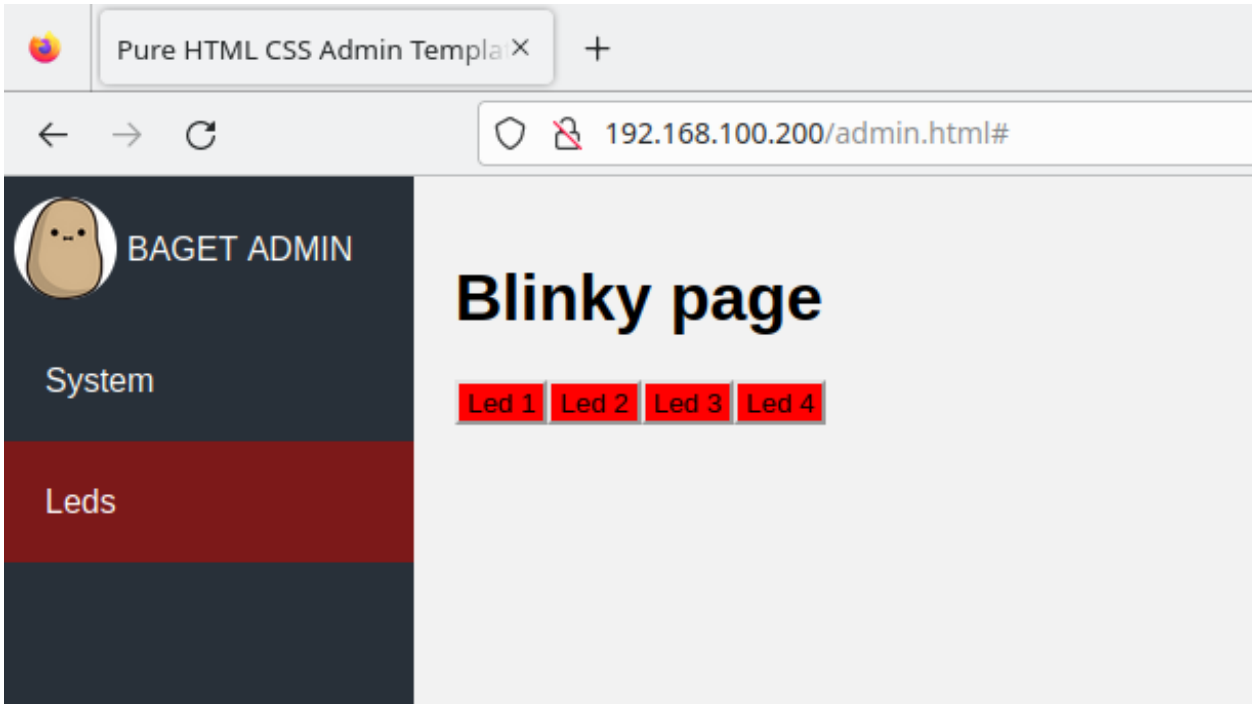


Замечание: если вы получили результат, что страница не найдена, проверьте, что у адреса префикс http и повторите попытку снова. Помните, если браузер не находит ответа от http, он автоматически переключается на https и пробует найти что-то там. А так как префикс определяет порт, к которому отправляются запросы (https это 433 порт), то важно убедиться, что браузер пытается подключиться к нужному порту (в конфигурация описан только 80 порт).

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 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, что веб-сервер в системе работает, как ещё один терминал (вспомните, как подключались к терминалу платы по ssh, тут нечто похожее). При этом подключение ведётся не от имени суперпользователя (так делать никогда нельзя, если конечно не хотите сделать большой проход в систему). Мало того, этот «пользователь», не может получить доступа к корневому каталогу платы, для него корневым каталогом в является папка /var/www/ (в текущих конфигурация сервера).

Так же, можно заметить, что при попытке управлять светодиодами, возникает большая задержка, между тем как происходит нажатие кнопки на веб-странице и непосредственно переключением светодиода. 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.

8.4 Задание для самостоятельной работы

Добавте возможность управления всеми выводами банка F. Можно воспользовтся тектовым редактором nano установленном на целевой платформе.

Необходимо внести правки в следующие файлы:

- led_socekt.py - дописать элементы в строке 6 (если фалй уже запускался, остановите его выполнение командой killall python3, и не запустие его вновь, после редактирования)
- /var/www/js/led_ctrl.js - дописать в первой строке ещё 4 элемента, для проверки достаточно обновить страницу в браузере. Дополнительно, можно посмотреть отладочную информацию кликнув правой клавишей мыши на странице и выбрав пункт Inspect. В открывшемся наборе вкладок выберите Console. Если в результате правок были

допущены ошибки, тут будет о них информация.

Рекомендуемая литература

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

Приложение A

Установка и настройка виртуальной машины

В данном пособие используется гипервизор Oracle VirtualBox настроенный для работы с отладочной платой. В данном разделе рассмотрим настройку рабочего окружения, необходимого для выполнения работ.

A.1 Настройка гипервизора

Необходимо скачать с сайта https://www.virtualbox.org следующие пакеты:

1. VirtualBox Platform Packages - установочный пакет гипервизора.
2. VirtualBox Extension Pack - расширения для гипервизора, позволяющие в том числе использовать USB интерфейсы хоста из гостевой операционной системы.

Запустите установку гипервизора. По завершению установки, запустите гипервизор. В открывшемся окне, выберите File -> Tools -> Extension Pack Manager.

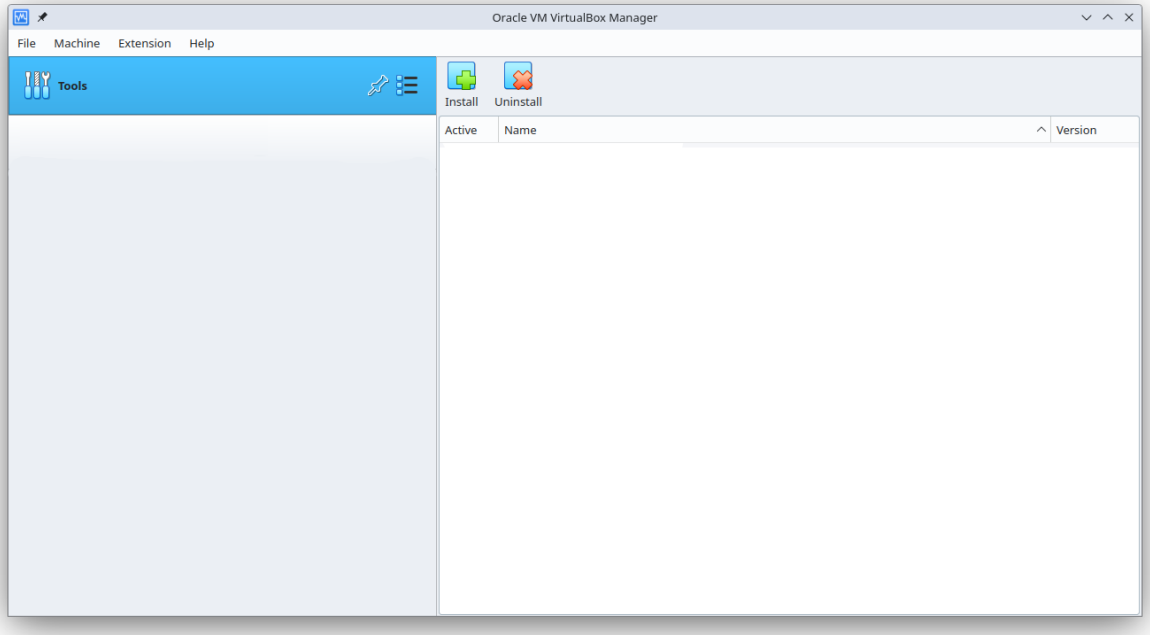


Рисунок А.1. Окно установки расширений

Нажмите, на кнопку Install. В открывшемся диалоговом окне выберите файл расширения для гипервизора (файл с расширением vbox-extpack), и нажмите на кнопку Open.

А.2 Установка гостевой машины

А.2.1 Необходимо скачать установочный образ Kubuntu 24.10 с сайта https://kubuntu.org/getkubuntu/

А.2.2 Запустить гипервизор Oracle VirtualBox, если закрыли его ранее.

А.2.3 В окне гипервизора выберите Machine -> New.

Нажмите на кнопку Expert Mode (рисунок А.2), для перехода в расширенный режим конфигурации.

А.2.4 Вкладка Name and Operation System.

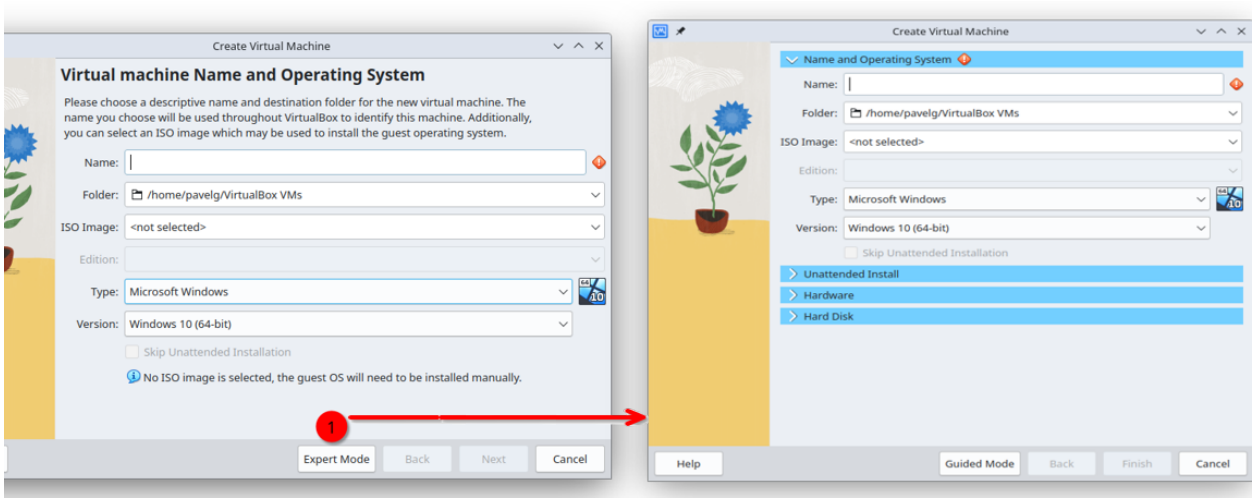


Рисунок А.2. Окно создания новой гостевой машины

1. Name - Название для виртуальной машины BagetWS_kubuntu24.

2. Folder - родительская папка в которой будет создана виртуальная машина (в данном папке появиться вложенная папка с именем виртуальной машины).

3. ISO Image - путь к образу диска с установочными файлами операционной системы, скаченный ранее.

4. Skip Unattended Installation - данный пункт должен быть отмечен.

А.2.5 Вкладка Hardware.

1. Base Memory - выбор доступной для гостевой машины объёма оперативной памяти, выберите не менее 16384 MB, если данное количество попадает в красную зону, то следует расширить объём оперативной памяти рабочего компьютера, или выбрать другой компьютер.

2. Processor - выбор количества потоков доступных для гостевой машины, выберите не менее 4, если данное количество попадает в красную зону, то следует выбрать другой компьютер.

Остальные поля данной вкладки нужно оставить без изменения.

A.2.6 Вкладка Hard Disk. Выберите Create a Virtual Hard Disk Now, и укажите объём не менее 50GB.

Остальные поля данной вкладки нужно оставить без изменения.

A.2.7 Нажмите кнопку Finish, и в окне гипервизора запустите виртуальную машину двойным нажатием левой кнопки мыши на иконке. Дождитесь появление окна как на рисунке А.3.

При этом необходимо, что бы рабочий компьютер имел доступ к сети интернет.

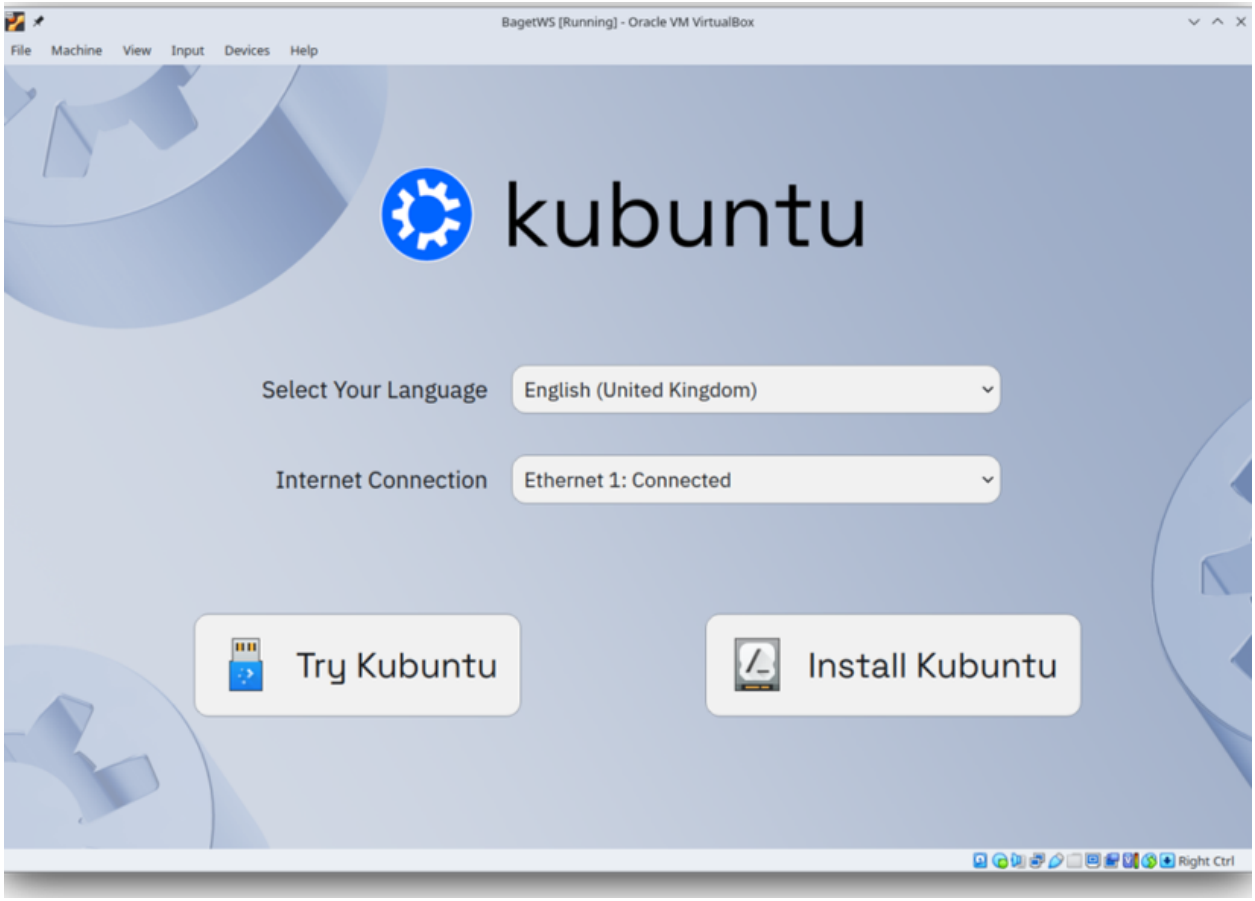


Рисунок А.3. Окно выбора действия

A.2.8 Нажмите на кнопку Install Kubuntu. И далее нажимайте на кнопку Next пока не дойдёте до раздела Partitions. В данном разделе выберите Erase disk, после чего нажмите кнопку Next.

A.2.9 Во кладке Users введите

1. Your name и login - student
2. Computer Name - bagetws
3. Password и Repeat Password - usrstudent
4. Включите опцию Log in automatically without asking for the password

A.2.10 Нажмите кнопку Next, затем кнопку Install. Подтвердите ещё раз начало установки. Дождитесь окончания установки.

A.2.11 Убедитесь, что опция Restart now отмечена, и нажмите кнопку Done.

A.2.12 Закройте окно гостевой машины.

A.2.13 В окне гипервизора выберите виртуальную машину, и нажмите на кнопку Settings.

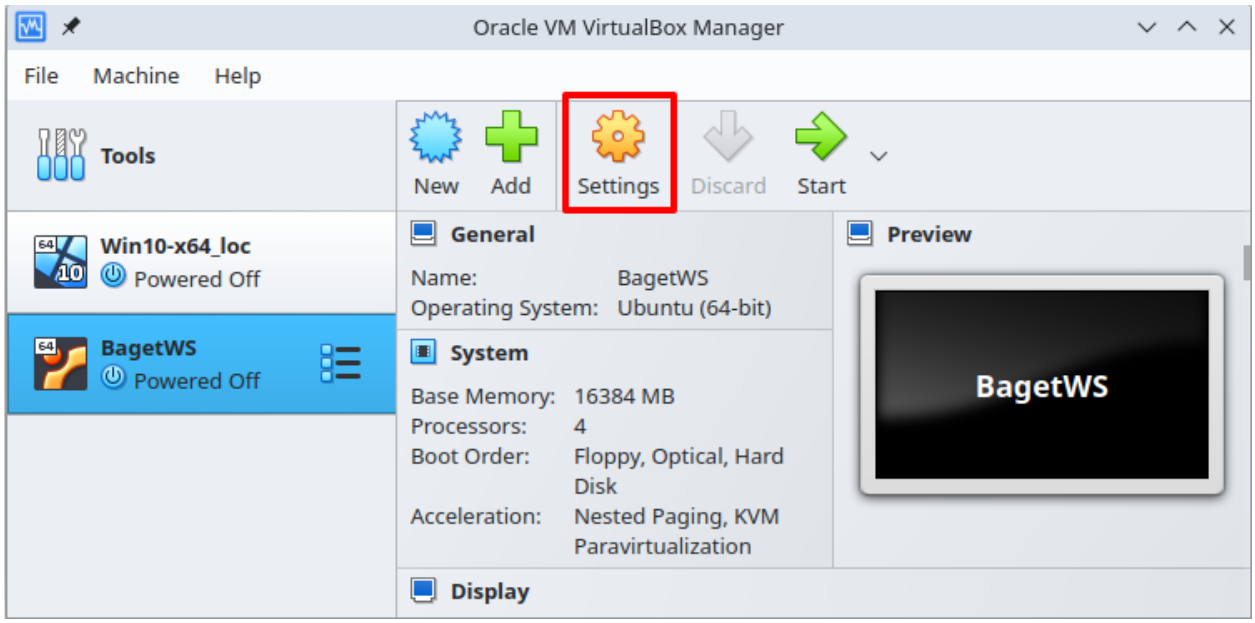


Рисунок А.4. Иконка настройки гостевой машины

А.2.14 Перейдите на вкладку Network (пункт 1 на рисунке А.5), выберите режим Bridget Adapter (2) для проброса физического сетевого адаптера в виртуальную машину, и укажите сетевой интерфейс (3), доступ к которому должна получить виртуальная машина.

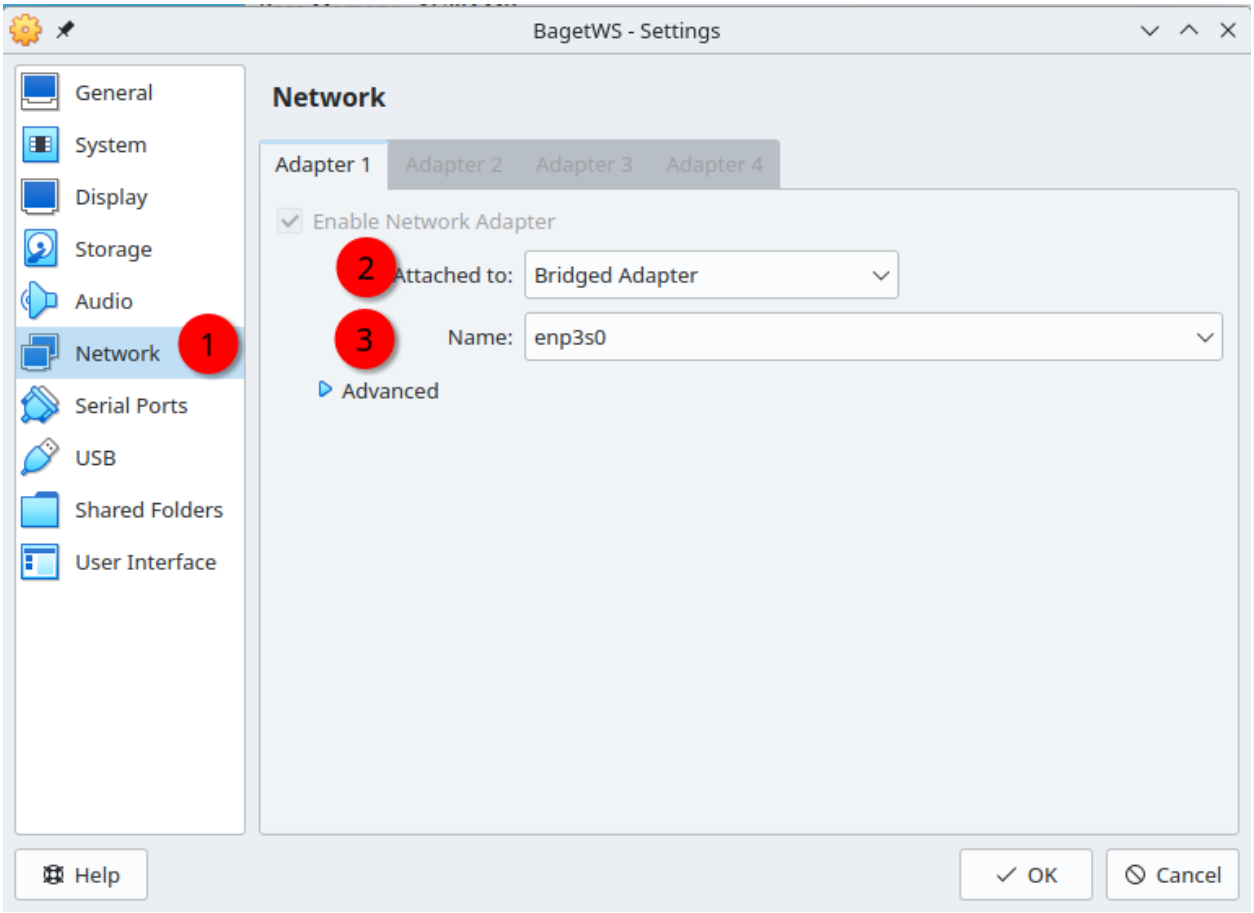


Рисунок А.5. Настройка первого сетевого интерфейса

А.2.15 Выберите вкладку Adapter 2 (пункт 1 на рисунок А.6) отметте Enable Network (2). Выберите тип подключения NAT (3). Через этот интерфейс виртуальная машина будет иметь доступ к сети интернет.

А.2.16 Нажмите кнопку OK что бы изменения вступили в силу.

А.2.17 Закройте окно настройки.

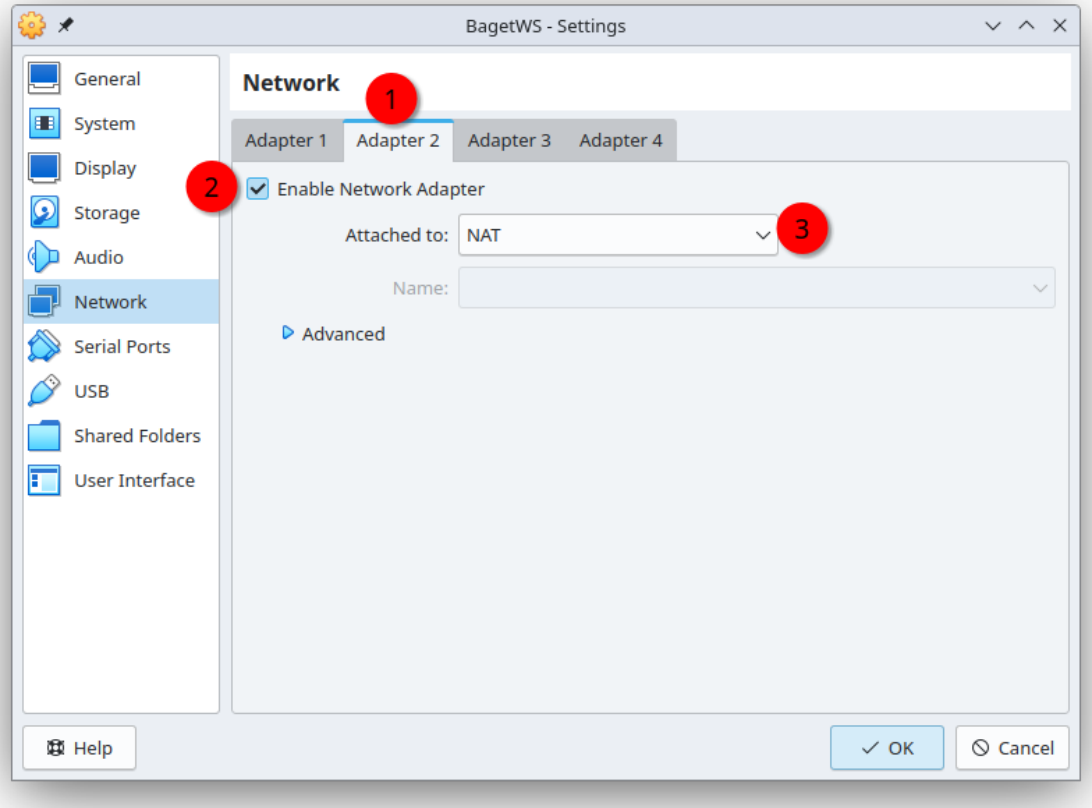


Рисунок А.6. Настройка второго сетевого интерфейса

А.3 Настройка гостевой машины

А.3.1 Запустите гостевую машину. После окончания загрузки, нажмите на иконку  в левом нижнем углу, выберите вкладку Favorites и справа, из списка приложений выберите программу терминал Konsole (так же это приложение можно открыть с помощью комбинации клавиш Ctrl + Alt + T).

А.3.2 В открывшемся терминале вбейте следующую команду:

```
# sudo apt update && sudo apt upgrade -y

# kwriteconfig5 --file kwalletrc  --group 'Wallet' \
--key 'Enabled' 'false'
```

A.3.3 Добавим расширения для взаимодействия между гостевой машиной и хостом.

```
# sudo apt install build-essential dkms -y

# sudo apt install virtualbox-guest-additions-iso -y

# sudo mount /usr/share/virtualbox/VBoxGuestAdditions.iso /mnt

# sudo /mnt/VBoxLinuxAdditions.run
```

После чего нужно перезагрузить гостевую машину:

```
# sudo shutdown -r now
```

A.3.4 Установите дополнительные пакеты

```
# sudo apt install -y gcc-mips64el-linux-gnuabi64 \
g++-mips64el-linux-gnuabi64 qemu-system-mips64el \
qemu-user-static gdb-multiarch gtkterm minicom

# sudo apt install -y git device-tree-compiler debootstrap
```

A.3.5 Склонируйте репозиторий с файлами для лабораторных работ.

```
# git clone -n --depth=1 --filter=tree:0 \
    https://github.com/gpav31/baget_linux_labs /home/student/baget/

# cd /home/student/baget

# git sparse-checkout set --no-cone /support

# git checkout
```

A.3.6 Установите VSCode и необходимые расширения.

```
# sudo snap install --classic code
```

```
# code --install-extension ms-vscode.cpptools

# code --install-extension ms-vscode.cpptools-extension-pack

# code --install-extension ms-vscode.makefile-tools
```

A.3.7 Добавьте переменные среды. Для этого откройте файл bashrc

```
# nano /home/student/.bashrc
```

В начало файла нужно добавить следующею строку:

BAGET="/home/student/baget"

A.3.8 Настройте автоматическое монтирование для USB устройств. Для этого откройте программу Settings  .

В открывшемся окне (рисунок А.7), в строке поиска вбейте Remov, и выберите ниже пункт Removable Devices. Отметьте пункты, как показано на рисунке А.7.

Нажмите кнопку Apply.

A.3.9 Вбейте в строку поиска слово connections и выберите вкладку Connections. В данной вкладке происходит настройка сетевых интерфейсов виртуальной машины.

Выберите тот, что не отвечает за интернет (теоретически это Wired connection 1), и перейдите во вкладку IPv4.

В поле Method выберите Manual.

Нажмите кнопку +Add справа от таблицы. В колонку Address введите 192.168.100.2 и нажмите Enter. В колонка Netmask должна появиться надпись 255.255.255.0, скорректируйте её, если она отличается.

Нажмите кнопку Apply.

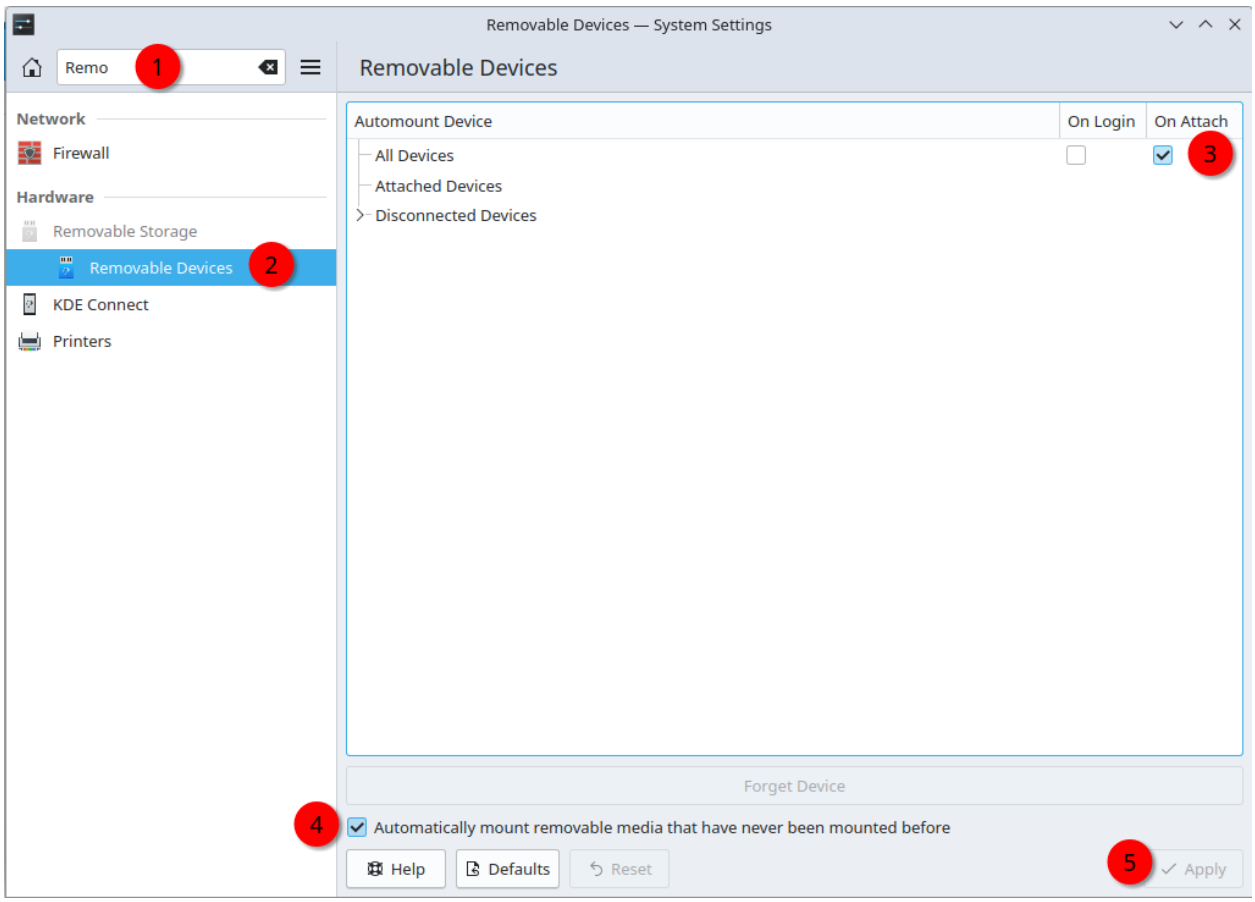


Рисунок А.7. Подключение автоматического монтирования устройств

А.3.10 Добавьте необходимые программы в Избранное. Нажмите на иконку  в левом нижнем углу, и в поле search введите terminal. Нажмите правой кнопкой на программе Serial port terminal и выберите пункт Add to Favorites.

Повторите тоже самое для VSCode, для чего в поле поиска нужно вписать слово code, и найти Visual Studio Code.

А.3.11 Добавьте пользователя student в группу dialout, что бы можно было подключаться к последовательному порту без прав администратора.

```
sudo usermod student -aG dialout
```

А.3.12 Выключите виртуальную машину.

A.3.13 Сохраните состояние виртуальной машины (Snapshot) или скопируйте её, что бы по завершению курса можно было её состояние вернуть в первоначальное, и не повторять все настройки описанные выше.