Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления (ИУ)**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии (ИУ7)**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Драйвер для устройств с интерфейсом GPIO»

Студент группы **ИУ7-72Б** _____ **Иванов В.А.**
(Подпись, дата) (И.О. Фамилия)

Руководитель НИР _____ **Рязанова Н.Ю.**
(Подпись, дата) (И.О. Фамилия)

***2021** г.*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данный момент активно ведётся развитие технологий ”умного дома”и ”интернета вещей”. Они направлены на создание общей сети для любого типа домашней техники или механизмов. Целью является их тесная интеграция, создание возможности управления и получения информации об их текущем состоянии для жильца дома.

Разработка подобных ”умных”устройств тесно связана с использованием микроконтроллеров или одноплатных компьютеров. Такие устройства имеют небольшой размер и представляют незначительные вычислительные мощности. Поэтому для них зачастую требуется применение низкоуровневого программирования.

Наиболее распространённым физическим интерфейсом подключения является GPIO - контакты общего назначения, позволяющие подключать к себе широкий спектр различных устройств.

Данная работа посвящена разработке драйвера для взаимодействия с устройствами с интерфейсом GPIO.

1 Аналитическая часть

1.1 Постановка задачи

По заданию требуется разработать драйвер в виде загружаемого модуля ядра, позволяющий управлять устройствами, подключенными с помощью интерфейса GPIO.

Необходимо предоставить пользователю возможность ввода/вывода информации с устройств, управления режимами.

Для достижения данной цели необходимо решить следующие задачи:

1) ознакомиться с основными принципами работы устройств интерфейса GPIO;
2) определить способ управления устройствами;
3) выделить набор действий;
4) реализовать драйвер.

Драйвер должен предоставлять минимальный необходимый функционал для управления контактами:

- ввод / вывод значений;
- захват / освобождение владения процессом;
- установление режима работы.

Такие операции как захват и освобождение владения нужны для того, чтобы обеспечить монопольный доступ на запись для одного процесса. Таким образом, без захвата владения пином возможно только чтение его значения.

1.2 Принцип работы устройств GPIO

GPIO - интерфейс для связи между компонентами компьютерной системы, к примеру, микропроцессором и различными периферийными устройствами[1]. Контакты GPIO могут выступать как в роли входа, так и в роли выхода — это, как правило, конфигурируется. Обычно они используются для подключения датчиков, переключателей, дисплеев и т.п.

В данной работе будет использоваться одноплатный компьютер Raspberry

Pi 2B ввиду отсутствия в распоряжении других компьютеров. Рассмотрим подробнее организацию GPIO на его примере.

На рисунке 1 описывается назначение каждого из контактов (пинов) для этой модели. Можно отметить, что не все из них используются для ввода/вывода. Помимо этого есть контакты предназначенные для подачи определённого напряжения на внешние устройства (3.3V, 5V) или для заземления (GROUND).

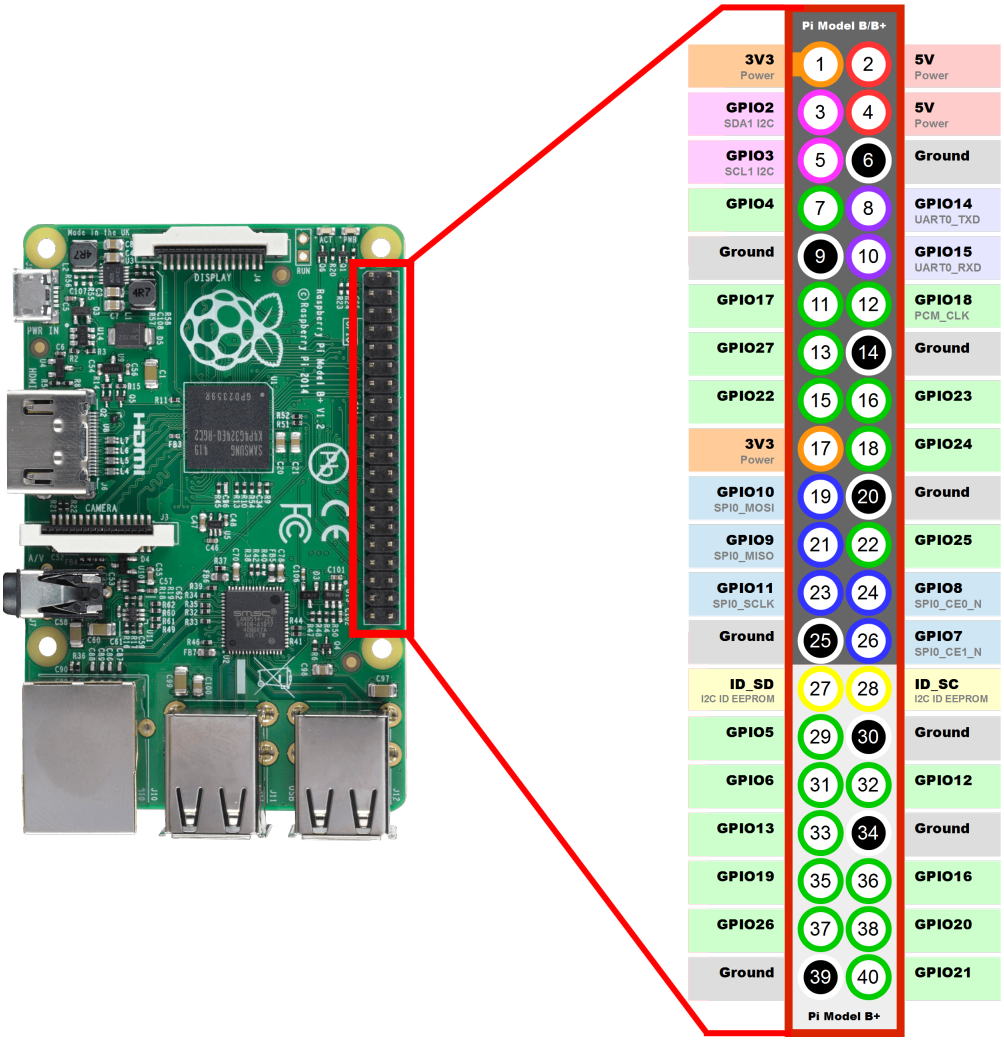

Рисунок 1 – Назначение пинов Raspberry Pi 2B

Пин GPIO имеет два режима:

- **Вход**. Напряжение подаётся внешним устройством. От +0.0В до +1.8В считается уровнем логического нуля, +1.8В-3.3В - логическая единица.
- **Выход**. Напряжение подаётся самим Raspberry Pi. Уровень логических 0 и 1 аналогичный.

По сути, передача и приём информации осуществляется только считыванием и установлением определённого напряжения. Выходы, отмеченные на схеме зелёным цветом имеют наиболее простой принцип действия: режим ввода/вывода в них устанавливается на всё время подключения устройства, а значение напряжения является относительно постоянным, так как по ним передаётся минимальное количество информации.

В отличие от них, контакты имеющие подпись I2C, SPI или UART используются для последовательный синхронной передачи данных в режиме полного или полудуплекса. Это означает, что они используются для передачи уже более большого количества данных и могут переключать режим ввода/вывода по несколько тысяч раз за секунду.

Целью данной работы является разработка базового взаимодействия с внешними устройствами, поэтому драйвер будет ориентирован на более простые интерфейсы GPIO.

1.3 Загружаемый модуль ядра

В изучаемом одноплатном компьютере, как правило, используется операционная система Raspberry Pi OS, основанная на ядре Linux. Поэтому для реализации драйвера требуется написать загружаемый модуль ядра. После загрузки он становится частью ОС и получает доступ к ядрёным функциям и к памяти устройств ввода/вывода, что и требуется в этом случае.

Для обеспечения доступа к драйверу принято решение использовать символьное устройство. Набор возможных действий с ним определяется структурой file_operations, которая частично приведена в листинге 1.

Листинг 1: Структура file_operations

```
1 struct file_operations {
2     struct module *owner;
3     ...
4     long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
5     ...
6     int (*open) (struct inode *, struct file *);
```

```
7     int (*release) (struct inode *, struct file *);
8     ...
9 };
```

Взаимодействие с устройствами фактически производится через функцию unlocked_ioctl. Для её вызова используется функция **ioctl**. В качестве аргументов передаётся структура открытого файла драйвера, номер команды и указатель на данные ввода или вывода. Для задания номеров команд используются макросы описанные в листинге 2. Наличие R или W в названии указывают на то, что операция направленна на вывод или ввод соответственно. В качестве аргументов подаются тип и размер передаваемых данных, а также мäгическое число(уникальное число, позволяющее драйверу обнаружить ошибки некорректности команды с помощью макроса **_IOC_TYPE**).

Листинг 2: Макросы команд ioctl

```
1 _IO(type,nr);
2 _IOR(type,nr,size);
3 _IOW(type,nr,size);
4 _IOWR(type,nr,size);
```

1.4 Способ управления

Следующей задачей является определение способа чтения и изменения состояния интерфейсов GPIO.

Для работы с пинами используется способ отображения в память (memory maping). Для чтения или изменения состояния устройства требуется взаимодействовать с определённым участком оперативной память, имеющей постоянный физический адрес.

Рассмотрим устройство отображения GPIO в память для Raspberry Pi 2B. В листинге 3 приведены используемые константы[3].

Листинг 3: Адреса отображения в память

```
1 #define BCM2708_PERI_BASE   0x3F000000
2 #define GPIO_BASE           (BCM2708_PERI_BASE + 0x200000)
```

```
/* GPIO register offsets */
#define GPFSEL0      0x0      /* Function Select */
#define GPSET0       0x1c     /* Pin Output Set */
#define GPCLR0       0x28     /* Pin Output Clear */
#define GPLEV0       0x34     /* Pin Level */
...
```

Адрес **BCM2708_PERI_BASE** определяет начало участка memory mapping для периферийных устройств, участок GPIO имеет смещение $0x200000$. Далее представленны смещения участков памяти относительно базового адреса **GPIO_BASE**, каждое из которых отвечает соответственно за режим ввода/вывода, установку, сброс и чтение значения с пинов.

1.5 Получение памяти ввода/ввода

Для того, чтобы память ввода/вывода стала доступной для использования драйвером требуется вызвать функции представленные в листинге 4 [4].

Листинг 4: Получение доступа к памяти IO

```
1  struct resource *request_mem_region(unsigned long start, unsigned long
   len, char *name);
2  void *ioremap(unsigned long phys_addr, unsigned long size);
3  u32 readl(const volatile void __iomem *addr);
4  void writel(u32 b, volatile void __iomem *addr);
```

С помощью функции **request_mem_region** производится выделение участка физической памяти для использования модулем. Однако, данный участок памяти не будет доступен напрямую. Для его использования требуется вызвать функцию **ioremap**. Она возвращает виртуальный адрес, который используется для получения доступа к физическому. Чтение и запись из такого адреса производится через команды **readl**, **writel**.

1.6 Выводы

В этом разделе были сформулированы цель и задачи, выделен функционал. Было принято решение использовать char драйвер. Изучено устройство интерфейса GPIO и принцип взаимодействия с ним.

2 Конструкторская часть

2.1 Описание функционала

В соответствии с выделенным в предыдущем разделе функционалом, в качестве необходимых для взаимодействия с устройством выделим команды, указанные в таблице 1 (с указанием их аргументов и возвращаемого значения):

Таблица 1 – Команды ввода/вывода

Команда	**Аргументы**	**Возвращаемое значение**
Чтение	№ пина	Значение пина
Запись	№ пина, значение	-
Переключение	№ пина	Значение пина
Установка / Сброс	№ пина	-
Захват владения	№ пина	-
Освобождение владения	№ пина	-
Установка режима	№ пина, режим	-

3 Технологическая часть

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GPIO [Электронный ресурс] Режим доступа: http://ru.wikipedia.org/wiki/GPIO (дата обращения 10.12.2021).

2. Знакомство с GPIO в Raspberry Pi Режим доступа: https://ph0en1x.net/86-raspberry-pi-znakomstvo-s-gpio-perekluchatel-i-svetodiod.htmlgpio-header-layout-pins (дата обращения 10.12.2021).

3. pinctrl-bcm2835.c Режим доступа: https://elixir.bootlin.com/linux/latest/source/drive
bcm2835.cL49 (дата обращения 10.12.2021).

4. Использование памяти ввода/вывода Режим доступа: http://dmilvdv.narod.ru/Translate/LDD3/ldd_using_io_memory.html (дата обращения 10.12.2021).

ПРИЛОЖЕНИЕ А