Лекция 5.1 gpio, последовательный драйвер, i2c

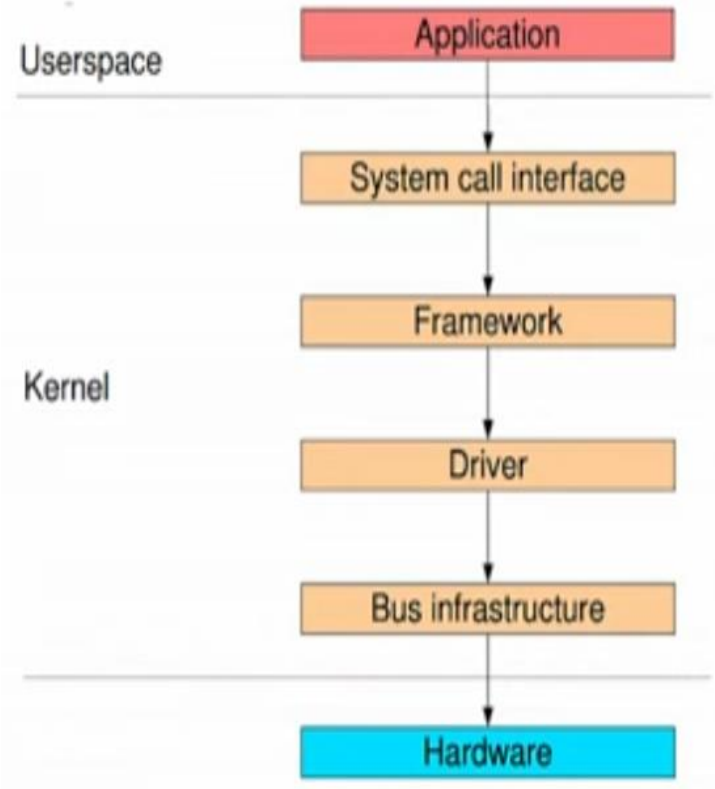
Разработал: Максимов А.Н.

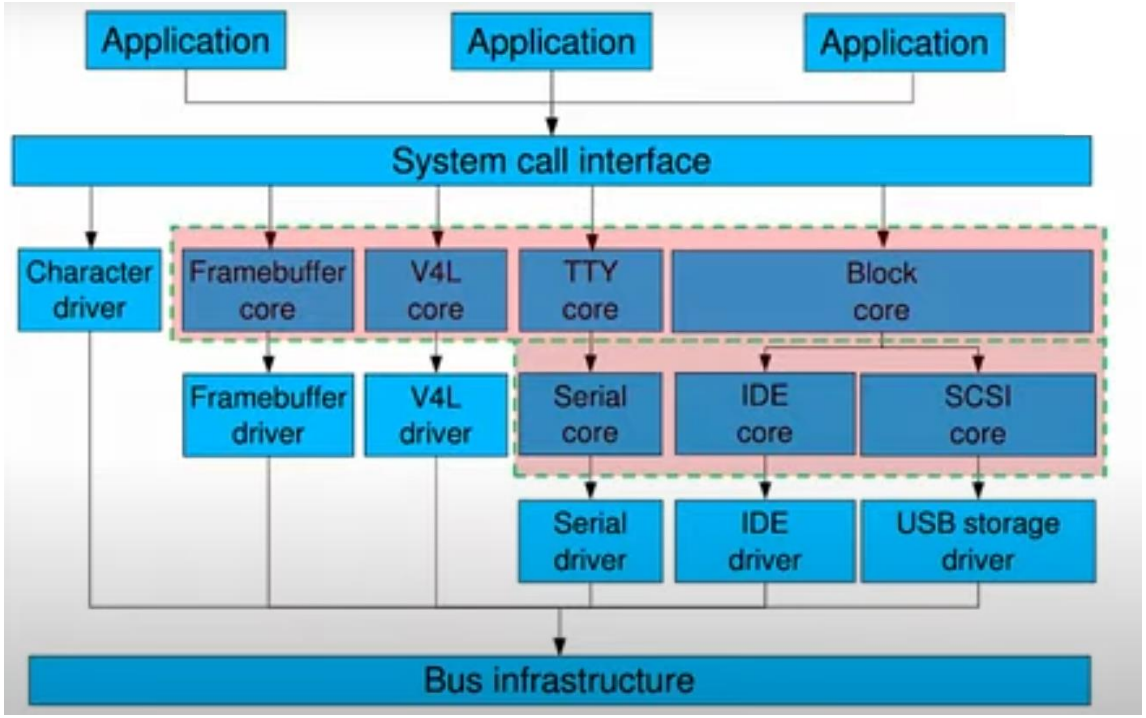
Версия 2.

Содержание

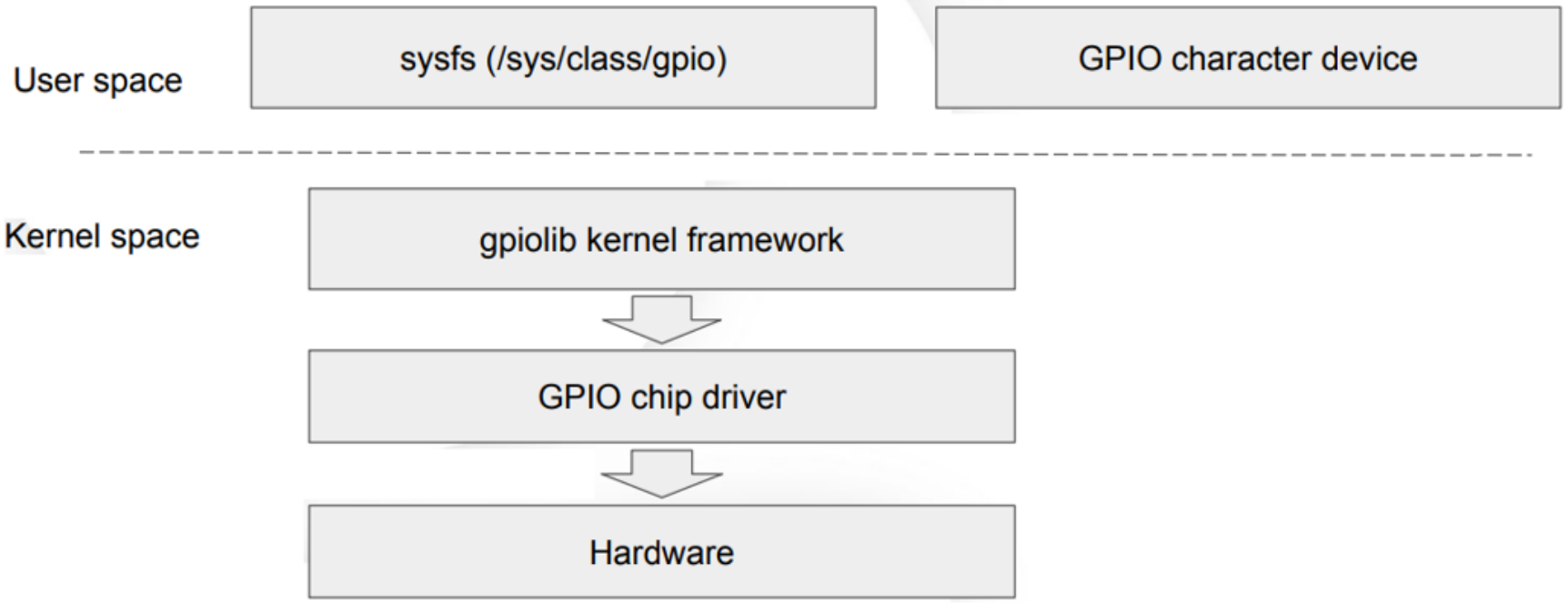
- GPIO
- Serial
- i2c

Взаимодействие подсистем на пример i2c





Пример драйвера в ядре - GPIO



Пример интерфейса GPIO через sysfs

экспортируем пин, без этого шага им не получится управлять:
echo 2 > /sys/class/gpio/export

Делаем его out-пином, то есть, он будет либо подавать, либо не подавать напряжение в 3.3 вольта:
echo out > /sys/class/gpio/gpio2/direction

Подаем напряжение:
echo 1 > /sys/class/gpio/gpio2/value

Перестаем подавать напряжение:
echo 0 > /sys/class/gpio/gpio2/value

Узнаем, подается ли сейчас напряжение:
cat /sys/class/gpio/gpio2/value

По завершении работы пину можно сделать unexport:
echo 2 > /sys/class/gpio/unexport

dts

```
gpio0: gpio@44e07000 {
        compatible = "ti,omap4-gpio";
        ti,hwmods = "gpio1";
        gpio-controller;
        #gpio-cells = <2>;
        interrupt-controller;
        #interrupt-cells = <2>;
        reg = <0x44e07000 0x1000>;
        interrupts = <96>;
};
```

Пример интерфейса GPIO через sysfs

```
static const struct of_device_id omap_gpio_match[] = {
{
        .compatible = "ti,omap4-gpio",
        .data = &omap4_pdata,
},
{
        .compatible = "ti,omap3-gpio",
        .data = &omap3_pdata,
},
{
        .compatible = "ti,omap2-gpio",
        .data = &omap2_pdata,
},
{ },
};
MODULE_DEVICE_TABLE(of, omap_gpio_match);
```

Пример интерфейса GPIO через sysfs

```
static struct platform_driver omap_gpio_driver = {
.probe          = omap_gpio_probe,
.remove         = omap_gpio_remove,
.driver         = {
        .name   = "omap_gpio",
        .pm  = &gpio_pm_ops,
        .of_match_table = of_match_ptr(omap_gpio_match),
},
};

/*
 * gpio driver register needs to be done before
 * machine_init functions access gpio APIs.
 * Hence omap_gpio_drv_reg() is a postcore_initcall.
 */
static int __init omap_gpio_drv_reg(void)
{
return platform_driver_register(&omap_gpio_driver);
}
postcore_initcall(omap_gpio_drv_reg);
```

probe

```
static int omap_gpio_probe(struct platform_device *pdev)
{
struct device *dev = &pdev->dev;
struct device_node *node = dev->of_node;
const struct of_device_id *match;
const struct omap_gpio_platform_data *pdata;
struct resource *res;
struct gpio_bank *bank;
struct irq_chip *irqc;
...
if (bank->is_mpuio)
        omap_mpuio_init(bank);

omap_gpio_mod_init(bank);

ret = omap_gpio_chip_init(bank, irqc);
if (ret) {
        pm_runtime_put_sync(dev);
        pm_runtime_disable(dev);
        if (bank->dbck_flag)
                clk_unprepare(bank->dbck);
        return ret;
}
omap_gpio_show_rev(bank);
```

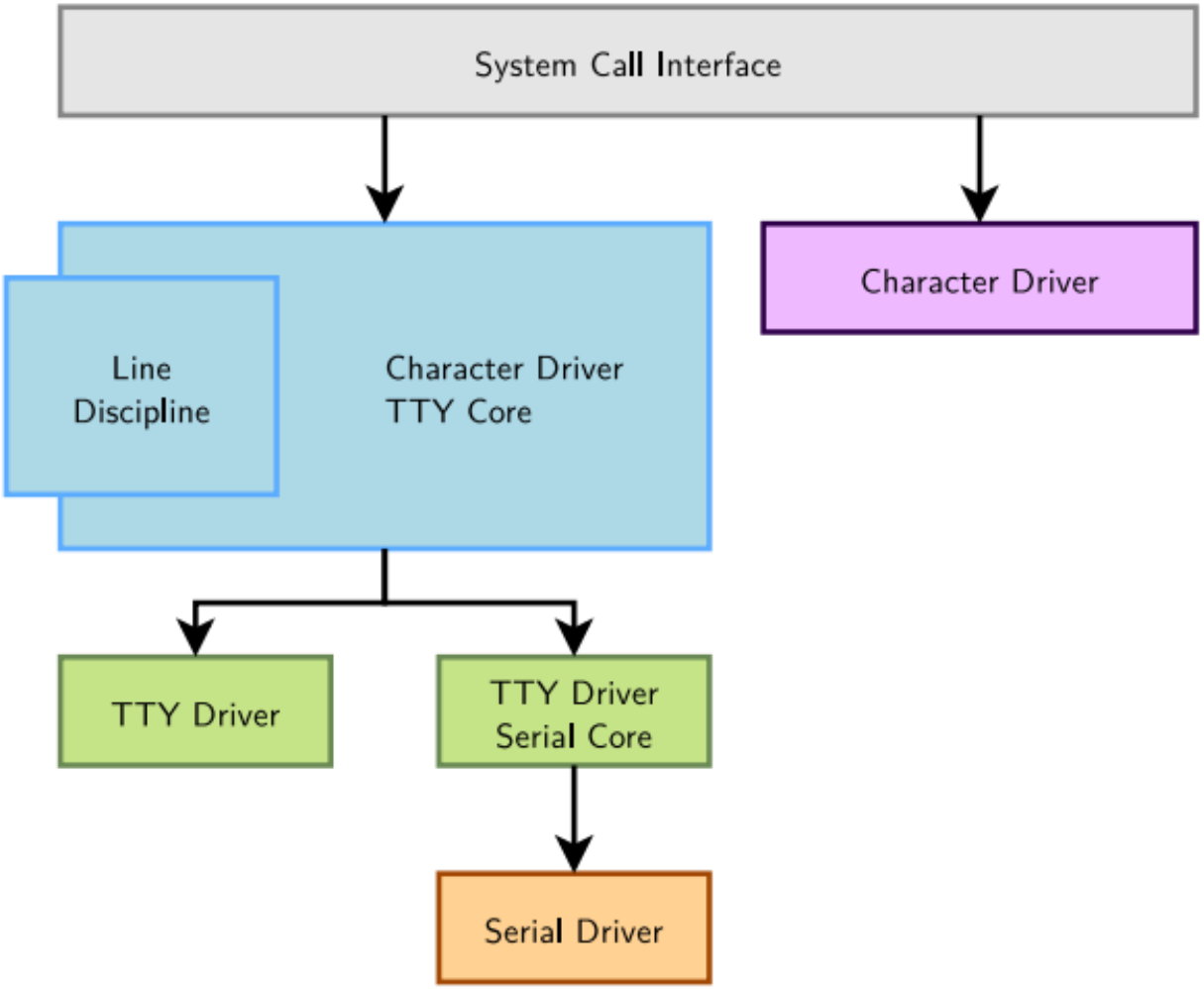
Структура gpio_chip

```
struct gpio_chip {
  /* имя порта gpio */
  const char *label;
  /* функция задания как входа */
  int (*direction_input)(struct gpio_chip *chip, unsigned offset);
  /* состояние контакта */
  int (*get)(struct gpio_chip *chip, unsigned offset);
  /* функция задания как выхода */
  int (*direction_output)(struct gpio_chip *chip, unsigned offset, int value);
  /* задание состояния */
  void (*set)(struct gpio_chip *chip, unsigned offset, int value);
  /* номер первого контакта в контексте ядра, присваивается динамически в случае
значения равном -1 */
  int base;
  /* количество контактов */
  u16 ngpio;
};
```

Регистрация gpiochip_add пример:

```
ret = gpiochip_add(&skel_gc->gchip);
```

serial



https://bootlin.com/doc/legacy/serial-drivers/linux-serial-drivers.pdf

Драйвер serial

Для правильной интеграции в систему Linux последовательные порты должны быть видны как устройства TTY из приложений пользовательского пространства

- Драйвер serial должен быть частью подсистемы TTY ядра
- Начиная с версии 2.6, специализированный драйвер TTY, serial_core, упрощает разработку последовательных
- драйверов
- Основные типы для инфраструктуры serial_core приведены в разделе include/linux/serial_core.h

Структуры данных

Структура данных, представляющая драйвер: uart_driver

Регистрация отдельного экземпляр для каждого драйвера
uart_register_driver() и uart_unregister_driver()

Структура данных, представляющая порт: uart_port
- По одному экземпляру для каждого порта (возможно несколько для каждого драйвера)
- uart_add_one_port() и uart_remove_one_port()
- Структура данных, содержащая указатели на операционную систему.

Особенности

owner, обычно устанавливается в значение THIS_MODULE
►driver_name,
► dev_name префикс имени устройства, обычно ttyS
► major и minor
► Используйте TTY_MAJOR и 64 для получения обычных чисел. Но они могут противоречить номерам, зарезервированным для 8250
► nr - максимальное количество портов
► cons - указатель на консольное устройство

Структуры данных

```
static struct uart_driver atmel_uart = {
.owner = THIS_MODULE,
.driver_name = "atmel_serial"
,
.dev_name = ATMEL_DEVICENAME,
.major = SERIAL_ATMEL_MAJOR,
.minor = MINOR_START,
.nr = ATMEL_MAX_UART,
.cons = ATMEL_CONSOLE_DEVICE,
};
static struct platform_driver atmel_serial_driver = {
.probe = atmel_serial_probe,
.remove = atmel_serial_remove,
.suspend = atmel_serial_suspend,
.resume = atmel_serial_resume,
.driver = {
.name = "atmel_usart"
,
.owner = THIS_MODULE,
},
};
```

Структуры данных

```
static int __init atmel_serial_init(void)
{
	uart_register_driver(&atmel_uart);
	platform_driver_register(&atmel_serial_driver);
	return 0;
}
static void __exit atmel_serial_exit(void)
{
	platform_driver_unregister(&atmel_serial_driver);
	uart_unregister_driver(&atmel_uart);
}
```

Регистрация

```
static int atmel_serial_probe(struct platform_device *pdev)
{
        struct atmel_uart_port *port;
        port = &atmel_ports[pdev->id];
        port->backup_imr = 0;
        atmel_init_port(port, pdev);
        uart_add_one_port(&atmel_uart, &port->uart);
        platform_set_drvdata(pdev, port);
        return 0;
}
static int atmel_serial_remove(struct platform_device *pdev)
{
        struct uart_port *port = platform_get_drvdata(pdev);
        platform_set_drvdata(pdev, NULL);
        uart_remove_one_port(&atmel_uart, port);
        return 0;
}
```

uart_ops

Важные операции

- tx_empty(), указывает, является ли FIFO для передачи пустым или нет
- set_ctrl() и get_ctrl(), позволяют устанавливать и получать параметры управления модемом(RTS, DTR, LOOP и т.д.).
- start_tx() и stop_tx() для запуска и остановки передачи
- stop_rx() для остановки приема
- startup() и shutdown(), вызываемые при открытии/закрытии порта
- request_port() и release_port(), запрашивают/освобождают области ввода-вывода или памяти
- set_termios(), изменяют параметры порта

Смотрите подробное описание в документации/serial/driver

Реализация передачи

- Метод start_tx() должен начать передачу символов через последовательный порт
- Передаваемые символы хранятся в циклическом буфере, реализованном
- структурой struct uart_circ. Он содержит
- buf[], буфер символов
- tail - индекс следующего передаваемого символа. После передачи tail необходимо обновить, используя

tail = tail &(UART_XMIT_SIZE - 1)

- Служебные функции в uart_circuit
- uart_circ_empty(), сообщает, пуст ли циклический буфер
- uart_circ_chars_pending(), возвращает количество символов, оставшихся для передачи
- Из указателя uart_port к этой структуре можно получить доступ, используя

port->state->xmit

Смотрите подробное описание в документации/serial/driver

Передача при помощи пулинга

```
foo_uart_putc(struct uart_port *port, unsigned char c) {
        while(__raw_readl(port->membase + UART_REG1) & UART_TX_FULL)
        cpu_relax();
        __raw_writel(c, port->membase + UART_REG2);
}
foo_uart_start_tx(struct uart_port *port) {
        struct circ_buf *xmit = &port->state->xmit;
        while (!uart_circ_empty(xmit)) {
                foo_uart_putc(port, xmit->buf[xmit->tail]);
                xmit->tail = (xmit->tail + 1) & (UART_XMIT_SIZE - 1);
                port->icount.tx++;
        }
}
```

Смотрите подробное описание в документации/serial/driver

Прием

- При приеме, обычно в обработчике прерываний, драйвер должен увеличить значение port->icount.rx
- Вызвать uart_handle_break(), если был получен запрос BRK, и, если он возвращает значение TRUE, перейти к следующему символу
- Если произошла ошибка, увеличьте значение port->icount.parity, port->icount.frame, port->icount.overrun в зависимости от типа ошибки
- Вызовите uart_handle_sysrq_char() с полученным символом, и если он вернет значение TRUE, перейдите к следующему символу
- Вызовите uart_insert_char() с полученным символом и статусом
- Статус TRY_NORMAL означает, что все в порядке, или TTY_BREAK, TTY_PARITY, TTY_FRAME в случае ошибки
- Вызовите функцию tty_flip_buffer_push() для отправки данных на уровень TTY

Смотрите подробное описание в документации/serial/driver

Прием

- При приеме, обычно в обработчике прерываний, драйвер должен увеличить значение port->icount.rx
- Вызвать uart_handle_break(), если был получен запрос BRK, и, если он возвращает значение TRUE, перейти к следующему символу
- Если произошла ошибка, увеличьте значение port->icount.parity, port->icount.frame, port->icount.overrun в зависимости от типа ошибки
- Вызовите uart_handle_sysrq_char() с полученным символом, и если он вернет значение TRUE, перейдите к следующему символу
- Вызовите uart_insert_char() с полученным символом и статусом
- Статус TRY_NORMAL означает, что все в порядке, или TTY_BREAK, TTY_PARITY, TTY_FRAME в случае ошибки
- Вызовите функцию tty_flip_buffer_push() для отправки данных на уровень TTY

Смотрите подробное описание в документации/serial/driver

Прием

Часть работы по приему обработке выполняется Sysrq

► Sysrq - это специальные команды, которые могут быть отправлены ядру для перезагрузки, размонтирования файловых систем, сброса состояния задачи, создания задач реального времени и т.д.

► Эти команды реализованы на минимально возможном уровне, так что, даже если система заблокирована, вы можете восстановить ее.

► Через последовательный порт: отправьте символ BRK, отправьте символ команды Sysrq

► Смотрите Documentation/sysrq.txt

► В драйвере

► функция uart_handle_break() сохраняет текущее время + 5 секунд в переменной

► функция uart_handle_sysrq_char() проверит, не превышает ли текущее время сохраненного значения,
и если это так, запустит выполнение команды Sysrq

Смотрите подробное описание в документации/serial/driver

Прием

```
foo_receive_chars(struct uart_port *port) {
        int limit = 256;
        while (limit-- > 0) {
                status = __raw_readl(port->membase + REG_STATUS);
                ch = __raw_readl(port->membase + REG_DATA);
                flag = TTY_NORMAL;
                if (status & BREAK) {
                        port->icount.break++;
                        if (uart_handle_break(port))
                        continue;
                }
                else if (status & PARITY)
                        port->icount.parity++;
                else if (status & FRAME)
                        port->icount.frame++;
                else if (status & OVERRUN)
                    port->icount.overrun++;
        [...]
```

Смотрите подробное описание в документации/serial/driver

Прием

```
[...]
status &= port->read_status_mask;
if (status & BREAK)
    flag = TTY_BREAK;
else if (status & PARITY)
    flag = TTY_PARITY;
else if (status & FRAME)
    flag = TTY_FRAME;
if (uart_handle_sysrq_char(port, ch))
      continue;
      uart_insert_char(port, status, OVERRUN, ch, flag);
}
spin_unlock(& port->lock);
tty_flip_buffer_push(port->state->port.tty);
spin_lock(& port->lock);
}
```

Смотрите подробное описание в документации/serial/driver

Упрощенный serial

```
#include "am335x-boneblack.dts"
/* Метка, am335xx_pinmux, ссылается на существующий pin muxing определенный в
am33xx.dtsi */
&am33xx_pinmux {
    uart2_pins: uart2_pins { // Секция для определения муксинга для UART2
/*  Макрос AM33XX_IOPAD определен в omap.h. Он предоставляет абсолютный
физический адрес. Первый параметр получен из processor technical reference manual.    */
        pinctrl-single,pins = <
            AM33XX_IOPAD(0x954, PIN_OUTPUT_PULLDOWN | MUX_MODE1)
            AM33XX_IOPAD(0x950, PIN_INPUT_PULLUP | MUX_MODE1)
            >;
    };
};
/*  Метка ссылается на устройство, определенное в am33xx.dtsi . Поля указанные тут
переопределяют оригинальные значения.*/
&uart2 {
    compatible = "serial";                  // Идентификатор совместимых драйверов
    status = "okay";                        //  "okay" – разрешает использовать устройство
    pinctrl-names = "default";
    // Привязывает устройство UART2 к узлы, задающему пины
    pinctrl-0 = <&uart2_pins>;
};
```

Модификация dts

Модифицировать arch/arm/dts/ti/omap/Makefile

make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- **dtbs**

Макрос AM33XX_IOPAD

Макрос AM33XX_IOPAD определен в omap.h:

```
#define OMAP_IOPAD_OFFSET(pa, offset)   (((pa) & 0xffff) - (offset))

#define AM33XX_IOPAD(pa, val)     OMAP_IOPAD_OFFSET((pa), 0x0800) (val) (0)
```

https://elixir.bootlin.com/linux/v6.6.68/source/include/dt-bindings/pinctrl/omap.h

Работа с оборудованием

```
#include <linux/fs.h>
#include <linux/of.h>
#include <linux/io.h>
#include <linux/pm_runtime.h>
#include <linux/platform_device.h>
#include <linux/init.h>
#include <linux/uaccess.h>

static struct of_device_id hw_match_table[] = {
        {
            .compatible = "serial",
        },
};
static struct platform_driver hw_plat_driver = {
    .driver = {
        .name = "serial",
        .owner = THIS_MODULE,
        .of_match_table = hw_match_table},
    .probe = hw_probe,
    .remove = hw_remove
};
module_platform_driver(hw_plat_driver);
```

Работа с оборудованием

```
static int hw_probe(struct platform_device *pdev)
{
    struct resource *res;
    res = platform_get_resource(pdev, IORESOURCE_MEM, 0);


…
    host->base = ioremap(res->start, resource_size(res));
```

или

```
   dev->regs = devm_ioremap_resource(&pdev->dev, res);

   dev->irq = platform_get_irq(pdev, 0);
   int ret = devm_request_irq(&pdev->dev, dev->irq, irqHandler,
            0, "hw_serial", dev);
   …
}
```

Работа с оборудованием

```
static int hw_remove(struct platform_device *pdev) {

    …
    return 0;
}
```

Работа с оборудованием

```
static irqreturn_t irqHandler(int irq, void *d)
{
    struct hw_serial_dev *dev = d;
    do
    {
        char recv = reg_read(dev, UART_RX);
        write_circ_buff(recv, dev);
        wake_up(&dev->waitQ);
    }
    while (reg_read(dev, UART_LSR) & UART_LSR_DR);
    return IRQ_HANDLED;
}
```

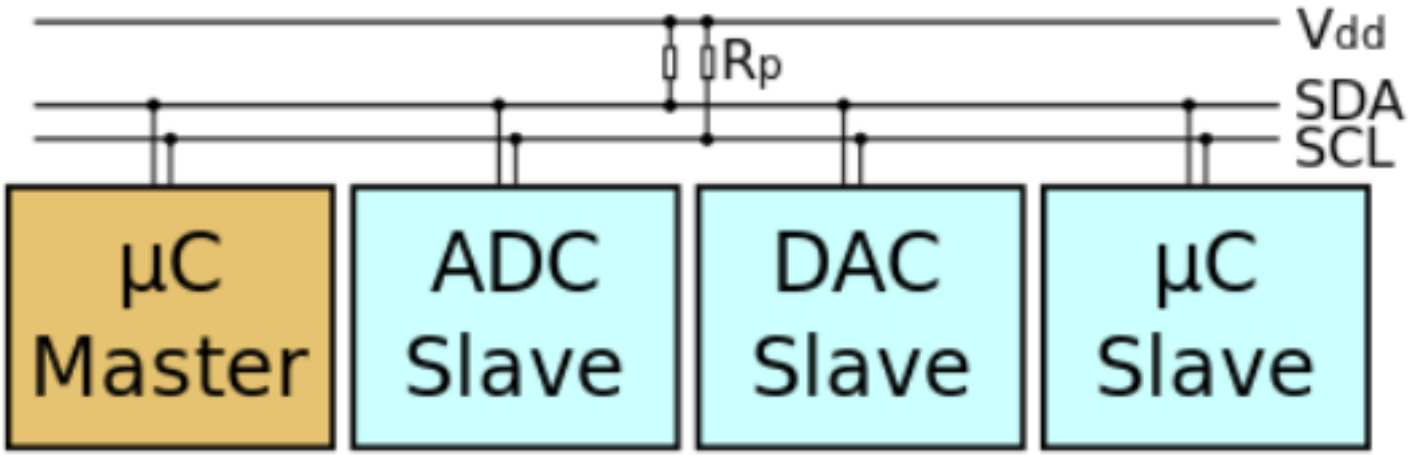
Шина i2c

Стандарт i2c разработан Philips Semiconductor в 1982 году, в настоящее время принадлежит NXP Semiconductors

- Синхронная последовательная шина с ведущим (maste) и несколькими ведомыми (slave) устройствами
- Такты генерирует master, ведомый лишь поддакивает при приеме байта
- На одной двупроводной шине может быть до 127 устройств.
- Полудуплексный протокол
- Сигнализация с открытым стоком
- В i2c используются только два сигнальных провода

– SDA: последовательные данные
– SCL: последовательные часы

Шина i2c



Шина i2c

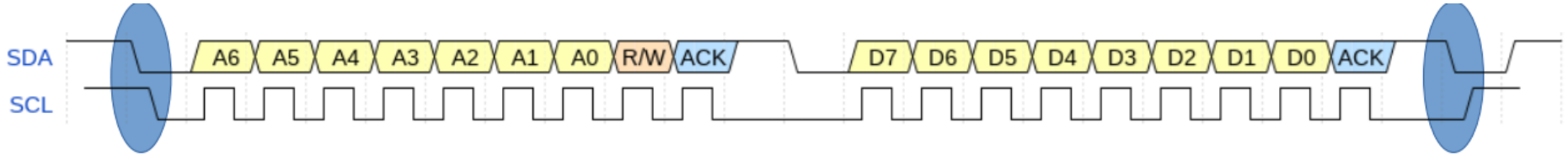
- Адрес на I2C обычно составляет 7 бит в соответствии с исходной спецификацией
- Тактовая частота I2C обычно составляет 100 кГц в соответствии с исходными спецификациями
- Более поздние версии спецификации ввели более быстрые режимы синхронизации и 10-битную адресацию:

– Версия 1 добавила быстрый режим 400 кГц и 10-битную адресацию
– Версия 2 добавила режим Hs 3,4 МГц
– Версия 3 добавила быстрый режим 1 МГц + и механизм идентификации
– Версия 4 добавила однонаправленный Ultra Fast mode 5 МГц

Примеры устройств на i2c

- Часы реального времени
- EEPROM
- Аналоговые преобразователи (АЦП, ЦАП)
- Датчики (температуры, давления, акселерометры)
- Микроконтроллеры
- Контроллеры тач скринов
- Расширители GPIO
- Адаптеры для мониторов и телевизоров
- и др.

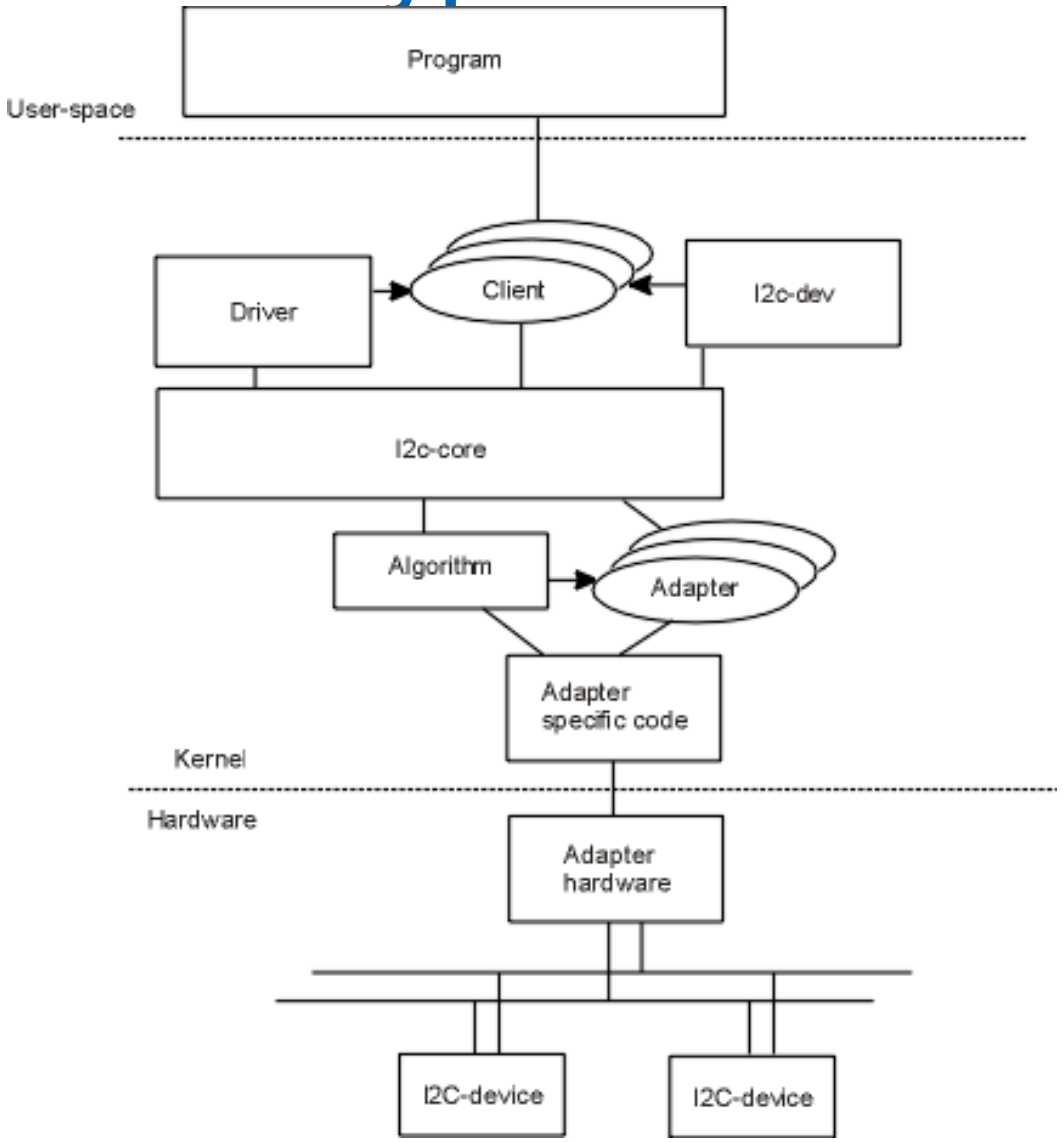
Протокол i2c



- Start - сигнал SDA понижается до сигнала SCLK, сигнализируя о начале передачи
- Addr - 7 bit адреса определяют слейв с которым будет коммуникация
- R/W - бит направления передачи данных (1 = read, 0 = write)
- Data - данные для чтения или записи на слейв. Может быть несколько байт
- ACK - Acknowledge bit. (0 = ack, 1 = nak)
- Stop - SDA повышается после SCL для сигнализации об окончании передачи

RISoft

Архитектура i2c в Linux



Утилиты для работы с i2c

- i2cdetect - Эта команда используется для обнаружения и составления списка всех шин I2C, доступных и известных Linux.
- i2cget - i2cget можно использовать для считывания ведомого устройства I2C.
- i2cset - i2cset может использоваться для записи данных по указанному адресу

ведомого устройства I2C.

- i2cdump - i2cdump может использоваться для сброса данных с ведомого устройства I2C.
- i2c_transfer - команда i2ctransfer очень полезна и может использоваться для чтения или записи нескольких байт в одной и той же команде.

Устсновка утилит:
sudo apt-get install i2c-tools

https://linuxhint.com/i2c-linux-utilities/

i2cdetect

i2cdetect - команда используется для обнаружения и перечисления всех шин I2C, доступных и известных Linux. Последовательно выставляет на I2C шину адреса устройств и при получении положительного ответа выводит в консоль номер адреса устройства на шине

Пример:

i2cdetect -l

```
i2c-1   i2c             0b234500.i2c-bus                        I2C adapter
i2c-2   i2c             0b234580.i2c-bus                        I2C adapter
i2c-0   i2c             0b234580.i2c-bus                        I2C adapter
i2c-5   i2c             0b234500.i2c-bus                        I2C adapter
```

i2cdetect -y 0

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: 30 -- -- -- -- -- 36 -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- 52 -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

I2get и i2cset

i2cget - i2cget можно использовать для считывания ведомого устройства I2C.

Пример (прочесть по смещению 0x0 с I2C слейва с адресом 0x50 on на шине 0**:**

```
i2cget -y 0 0x50 0
0x23
```

i2cset - i2cset может использоваться для записи данных по указанному внутреннему адресу

Пример (Записать 0x12 по смещению 0x2 для слейва 0x68)

```
i2cset -y 1 0x68 0x2 0x12
```

RISoft

Работа с i2c из программы на C

Обычно устройства i2c управляются драйвером ядра. Однако,возможно получить доступ ко всем устройствам на адаптере из пользовательского пространства через
интерфейс /dev. Для этого вам необходимо загрузить модуль i2c-dev.

Каждому зарегистрированному адаптеру i2c присваивается номер, считая от 0. Вы можете просмотреть /sys/class/i2c-dev/ или "i2cdetect -l", чтобы увидеть, какой номер соответствует какому адаптеру.

Файлы устройств I2C - это символьные файлы устройств с major номером устройства 89

Они должны называться "i2c-%d" (i2c-0, i2c-1, ..., i2c-10, ...).

https://www.kernel.org/doc/Documentation/i2c/dev-interface

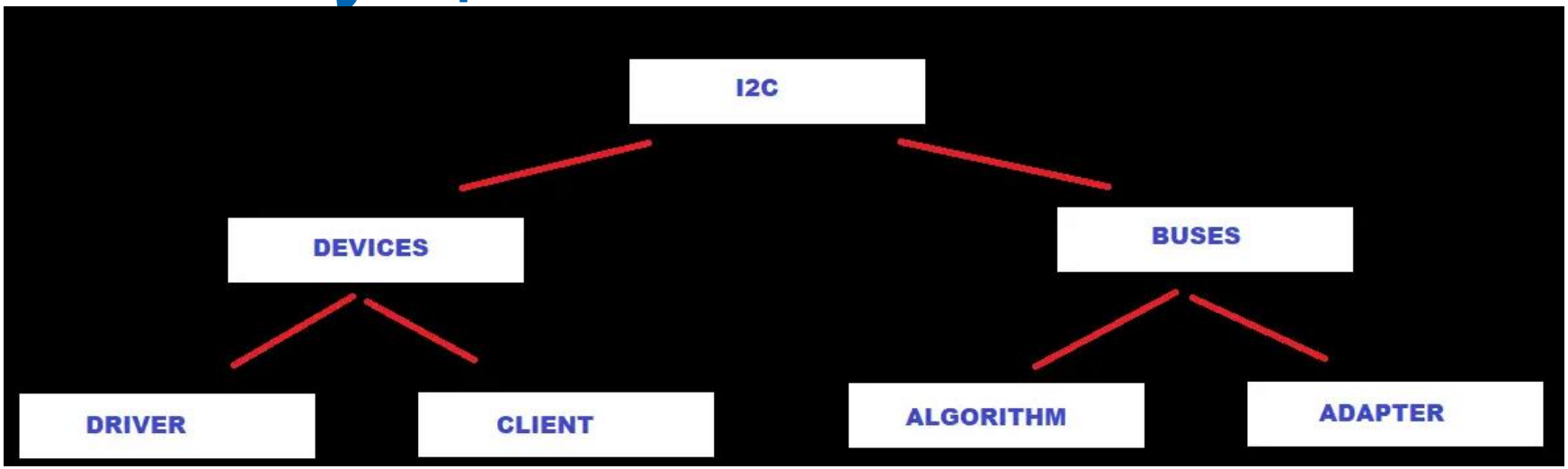
Работа с i2c из программы на C

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>

int main(int argc, char **argv) {
        int fd;
        fd = open("/dev/i2c-0", O_RDWR);
        struct i2c_rdwr_ioctl_data data;
        struct i2c_msg messages[2];
        unsigned char write_buf[1] = {0xD0}, read_buf[1] = {0x00};
        unsigned char write[200];

        // .addr - Адрес устройства (датчика) ,                    .flags - операция чтения или записи (0 - w, 1 - r)
        // .len - кол-во передаваемых/принимаемых сообщений .buf - буфер на чтение или запись
        messages[0].addr = 0x50;
        messages[0].flags = 0;
        messages[0].len = 1;
        messages[0].buf = write_buf;
        messages[1].addr = 0x50;
        messages[1].flags = 1;
        messages[1].len = 1;
        messages[1].buf = read_buf;
        data.msgs = messages;
        data.nmsgs = 2;
        if (ioctl(fd, I2C_RDWR, &data) < 0)
                printf("Cant send data!\n");
        else
                printf("ID = 0x%x\n", read_buf[0]);
         return 0;
}
```

Сущности i2c в Linux



- Bus

- Algorithm - Драйвер алгоритма содержит общий код, который может быть использован для класса адаптеров I2C
- Adapter - Драйвер адаптера либозависит от одного драйвера алгоритма, либо включает в себя его собственную реализацию

- Device

- Driver - Драйвер содержит общий код для доступа к некоторому типу устройства. Каждое обнаруженное устройство получает свои собственные данные в структуре Client.
- Client - Клиент представляет собой микросхему (подчиненное устройство) на I2C.

I2C Bus Driver

I2C Bus Driver определяет и выделяет приватную структуру данных (должна содержать struct i2c_adapter)

- Заполняет структуру algorithm struct
  - .master_xfer() – функция для выполнения передачи
  - .functionality() – функция для получения функциональности шины.
- Заполнить структуру adaptor struct
  - i2c_set_adapdata()
  - .algo – указатель на struct algorithm
  - .algo_data – указатель на приватную структуру
- Добавит адаптер
  - i2c_add_adapter() return 0;

Драйвер в dts

```
&i2c2 {
    mpu6050: accelerometer@68 {
        compatible = "foompu6050";
//          compatible = "invensense,mpu6050";

        reg = <0x68>;
        status="okay";
    };
};
```

I2C Bus Driver

I2C Bus Driver определяет и выделяет приватную структуру данных (должна содержать struct i2c_adapter)

- Заполняет структуру algorithm struct
  - .master_xfer() – функция для выполнения передачи
  - .functionality() – функция для получения функциональности шины.
- Заполнить структуру adaptor struct
  - i2c_set_adapdata()
  - .algo – указатель на struct algorithm
  - .algo_data – указатель на приватную структуру
- Добавит адаптер
  - i2c_add_adapter() return 0;

I2C Device Driver

```
static const struct i2c_device_id foo_mpu_id[] = {
        { SLAVE_DEVICE_NAME, 0 },
        { }
};
MODULE_DEVICE_TABLE(i2c, foo_mpu_id);
static struct i2c_driver foo_mpu_driver = {
        .driver = {
            .name   = SLAVE_DEVICE_NAME,
            .owner  = THIS_MODULE,
        },
        .probe          = foo_mpu_probe,
        .remove         = foo_mpu_remove,
        .id_table       = foo_mpu_id,
};
static struct i2c_board_info foo_mpu_board_info = {
        I2C_BOARD_INFO(SLAVE_DEVICE_NAME, SSD1306_SLAVE_ADDR)
    };

int  init_module(void) {
   return i2c_add_driver(&foo_mpu_driver);
}
void  cleanup_module(void) {
   i2c_del_driver(&foo_mpu_driver);
}
```

I2C Device Driver

```
static int foo_mpu_probe(struct i2c_client *client,
                         const struct i2c_device_id *id)
{
    pr_info("OLED Probed!!!\n");

    return 0;
}
/*
** This function getting called when the slave has been removed
** Note : This will be called only once when we unload the driver.
*/

static int foo_mpu_remove(struct i2c_client *client)
{

    pr_info("OLED Removed!!!\n");
    return 0;
}
```

Plain I2C API

```
int i2c_master_send(struct i2c_client *client,char *buf,int count);
int i2c_master_recv(struct i2c_client *client,char *buf, int count);
```

Эти процедуры считывают и записывают несколько байт с/на клиент. Клиент содержит адрес I2C, поэтому не обязательно его указывать. Второй параметр содержит байты для чтения/записи, третий - количество байт для чтения/записи (должно быть меньше длины буфера, также должно быть меньше 64 кб, поскольку msg.len равен u16.) Возвращается фактическое количество прочитанных/записанных байт.

int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msg, int num);

Функция отправляет серию сообщений. Каждое сообщение может быть для чтения или записи, и они могут быть смешаны любым способом.
Транзакции объединяются: между транзакциями не выдается условие остановки.
Структура i2c_msg содержит для каждого сообщения адрес клиента, количество байт сообщения и сами данные сообщения

include/linux/i2c.h

При использовании device tree

```
i2c0: i2c@01c2ac00 {
        compatible = "allwinner,sun7i-a20-i2c",
                     "allwinner,sun4i-a10-i2c";
        reg = <0x01c2ac00 0x400>;
        interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&apb1_gates 0>;
        status = "disabled";
        #address-cells = <1>;
        #size-cells = <0>;
};
```

Пример определения I2C контроллера в файле sun7i-a20.dtsi

```
&i2c0 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c0_pins_a>;
        status = "okay";

        axp209: pmic@34 {
                compatible = "x-powers,axp209";
                reg = <0x34>;
                interrupt-parent = <&nmi_intc>;
                interrupts = <0 IRQ_TYPE_LEVEL_LOW>;

                interrupt-controller;
                #interrupt-cells = <1>;
        };
};
```

Пример определения I2C устройства в файле sun7i-a20-olinuxino-micro.dts

http://nyx.skku.ac.kr/wp-content/uploads/2019/04/ESW05-I2C-1.pdf

I2c на beegle bone

В соответствии с техническим справочным руководством AM335X на Beaglebone Black установлены три шины I2C, адреса их памяти следующие:

i2c-0: 0x44E0_B000
i2c-1: 0x4802_A000
i2c-2: 0x4819_C000

```
debian@beaglebone:~$ i2cdetect -l
i2c-1   i2c         OMAP I2C adapter                I2C adapter
i2c-2   i2c         OMAP I2C adapter                I2C adapter
i2c-0   i2c         OMAP I2C adapter                I2C adapter
```

https://www.teachmemicro.com/beaglebone-black-i2c-tutorial/

I2c на beegle bone

2 I2C ports

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
I2C1_SCL	17	18	I2C1_SDA
I2C2_SCL	19	20	I2C2_SDA
I2C2_SCL	21	22	I2C2_SDA
GPIO_49	23	24	I2C1_SCL
GPIO_117	25	26	I2C1_SDA
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

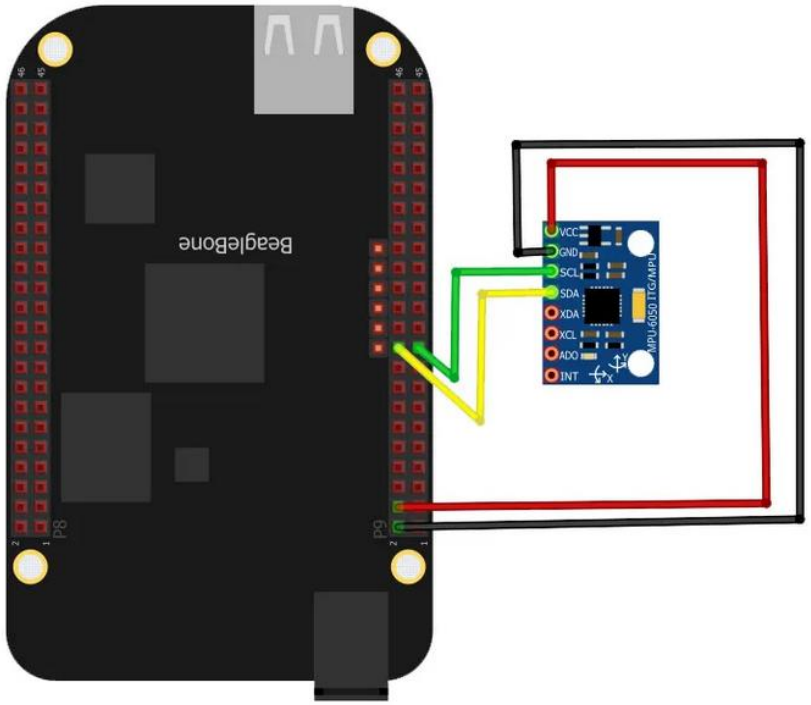
I2c на beegle bone

To detect the devices connected to i2c-2:

i2cdetect -r 2

https://www.teachmemicro.com/beaglebone-black-i2c-tutorial/

MPU6050 на beegle bone



https://www.teachmemicro.com/beaglebone-black-i2c-tutorial/

MPU6050 на beegle bone

```
debian@beaglebone:~$ i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

MPU6050 на beegle bone

Данные акселерометра и гироскопа имеют ширину 16 бит, поэтому данные с каждой оси используются в двух регистрах:

Register	Address
ACCEL_XOUT_H	0x3B
ACCEL_XOUT_L	0x3C
ACCEL_YOUT_H	0x3D
ACCEL_YOUT_L	0x3E
ACCEL_ZOUT_H	0x3F
ACCEL_ZOUT_L	0x40
GYRO_XOUT_H	0x43
GYRO_XOUT_L	0x44
GYRO_YOUT_H	0x45
GYRO_YOUT_L	0x46
GYRO_ZOUT_H	0x47
GYRO_ZOUT_L	0x48

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6500-Register-Map2.pdf

MPU6050 на beegle bone

Мы можем проверить содержимое этих регистров через i2cdump:

```
i2cdump 2 0x68
No size specified (using byte-data access)
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2, address 0x68, mode byte
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 89 7f 02 02 24 30 01 66 02 c3 02 0c 28 47 4d 8e    ????$0?f????(GM?
10: 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    Z...............
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00 00    ...........@....
70: 00 00 00 00 00 68 00 00 00 00 00 00 00 00 00 00    .....h..........
80: 89 7f 02 02 24 30 01 66 02 c3 02 0c 28 47 4d 8e    ????$0?f????(GM?
90: 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    Z...............
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
e0: 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00 00    ...........@....
f0: 00 00 00 00 00 68 00 00 00 00 00 00 00 00 00 00    .....h..........
```

Здесь вы увидите, что регистры в таблице выше имеют нулевые значения. Это потому, что нам нужно разбудить MPU6050, прежде чем он выдаст данные.

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

RISoft

MPU6050 на beegle bone

Чтобы активировать MPU6050, нам нужно записать ноль в регистр PWR_MGMT_1, который находится по адресу 0x6B. Вот как это сделать:

```
i2cset 2 0x68 0x6B 0
WARNING! This program can confuse your I2C
bus, cause data loss and worse!
I will write to device file /dev/i2c-2, chip
address 0x68, data address
0x6b, data 0x00, mode byte.
Continue? [Y/n] y
```

Здесь вы увидите, что регистры в таблице выше имеют нулевые значения. Это потому, что нам нужно разбудить MPU6050, прежде чем он выдаст данные.

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

MPU6050 на beegle bone

Чтение данных после активации:

```
i2cdump 2 0x68
No size specified (using byte-data access)
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2, address 0x68, mode byte
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 89 7f 02 02 24 30 01 66 02 c3 02 0c 28 47 4d 8e    ????$0?f????(GM?
10: 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    Z...............
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
30: 00 00 00 00 00 00 00 00 00 00 01 f2 00 0b 5c 40    ..........??.?\@
40: 58 f4 e0 fe 3e fd 60 00 14 00 00 00 00 00 00 00    X???>?`.?.......
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e6    ...............?
70: 00 00 00 00 00 68 00 00 00 00 00 00 00 00 00 00    .....h..........
80: 89 7f 02 02 24 30 01 66 02 c3 02 0c 28 47 4d 8e    ????$0?f????(GM?
90: 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    Z...............
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
b0: 00 00 00 00 00 00 00 00 00 00 01 f1 a4 0b ec 3f    ..........??????
c0: 24 f3 e0 01 d4 ff df 00 f1 00 00 00 00 00 00 00    $????.?.?.......
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 e7    ..............??
f0: 00 00 00 00 00 68 00 00 00 00 00 00 00 00 00 00    .....h..........
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

MPU6050 на beegle bone

Чтение одного регистра после активации (0x30 is the high byte of the MPU6050's acceleration in the x-axis.):

```
i2cget 2 0x68 0x3B
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will read from device file /dev/i2c-2, chip address 0x68, data address
0x3b, using read byte data.
Continue? [Y/n] y
0xee
```

Чтение одного регистра после активации (0x30 is the low byte of the MPU6050's acceleration in the x-axis.):

```
i2cget 2 0x68 0x3C
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will read from device file /dev/i2c-2, chip address 0x68, data address
0x3c, using read byte data.
Continue? [Y/n] y
0x94
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

MPU6050 на beegle bone

What we've read from the MPU6050 so far is raw data and needs to be converted to make sense. The acceleration data is measured in g (acceleration due to gravity) units while the gyroscope data is measured in degrees per second. The MPU6050 uses 2g and 250 deg/sec as default sensitivities.

Since each axes is 16 bytes and is signed, the raw values will range from -32,768 to 32, 767. This means acceleration in axes X ,Y and Z should have values within these ranges that represent the effect of gravity. For the default sensitivity which is 2g, the scaling factor should be around 16, 384 (32, 768/2).

In the same way, the X, Y and Z gyro data should have raw data readings from +250 to -250. According to the datasheet, the best scaling factor to use is 131.

However, even with scaling factors you may not get the exact acceleration and rotational speed since there will always be noise and sensor reading errors

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

Dts для MPU6050

```
&i2c2 {
    mpu6050: accelerometer@68 {
        compatible = "foompu6050";
//          compatible = "invensense,mpu6050";

        reg = <0x68>;
        status="okay";
    };
};
```

Модуль

```
static const struct i2c_device_id foo_mpu_id[] = {
    { SLAVE_DEVICE_NAME, 0 },
    { }
};
MODULE_DEVICE_TABLE(i2c, foo_mpu_id);
/*
** I2C driver Structure that has to be added to linux
*/
static struct i2c_driver foo_mpu_driver = {
    .driver = {
        .name   = SLAVE_DEVICE_NAME,
        .owner  = THIS_MODULE,
    },
    .probe          = foo_mpu_probe,
    .remove         = foo_mpu_remove,
    .id_table       = foo_mpu_id,
};
/*
** I2C Board Info strucutre
*/
static struct i2c_board_info foo_mpu_board_info = {
    I2C_BOARD_INFO(SLAVE_DEVICE_NAME, SSD1306_SLAVE_ADDR)
};

//module_i2c_driver(foo_mpu_driver);

int  init_module(void) {
        return i2c_add_driver(&foo_mpu_driver);
}

void  cleanup_module(void) {
        i2c_del_driver(&foo_mpu_driver);
```

Модуль

```
static int foo_mpu_probe(struct i2c_client *client, const struct i2c_device_id *id) {
    int status;

    // Check for platform i2c device capabilities here
    if (!i2c_check_functionality(client->adapter,I2C_FUNC_SMBUS_BYTE_DATA)) {
        dev_dbg(&client->dev, "i2c_check_functionality failed (0x%x)\n", client->addr);
        status = -EIO;
        goto exit;
    }

    dev_info(&client->dev, "chip probed - adding to client store\n");
    // adding to list here
    i2c_cpld_add_client(client);
    /* Register sysfs hooks */
    status = sysfs_create_group(&client->dev.kobj, &i2c_cpld_attr_grp);
    if (status) {
        printk(KERN_INFO "Cannot create sysfs\n");
    }
    return 0;
exit:
    return status;
}
```

Модуль

```
static DEVICE_ATTR(qsfp_lpmode, S_IRUGO,get_lpmode,NULL);
static DEVICE_ATTR(qsfp_reset,  S_IRUGO,get_reset,NULL);

static struct attribute *i2c_cpld_attrs[] = {
    &dev_attr_qsfp_lpmode.attr,
    &dev_attr_qsfp_reset.attr,
    NULL,
};

static struct attribute_group i2c_cpld_attr_grp = {
    .attrs = i2c_cpld_attrs,
};
```

Модуль

```
int i2c_master_recv ( const struct i2c_client * клиент, const char * buf,
int count);
```

Где,
client – дескриптор подчиненного устройства
buf – Данные, которые будут считываться с подчиненного устройства count – Количество байт для чтения должно быть меньше 64 кб, поскольку длина сообщения в msg равна u16

Он возвращает отрицательное значение errno, или же количество считываемых байт.

```
int i2c_master_send ( const struct i2c_client * клиент, const char * buf,
int count);
```

Где,
client – дескриптор подчиненного устройства
buf – Данные, которые будут записаны на подчиненное устройство
count – Количество байт для записи, должно быть меньше 64 кб, поскольку длина сообщения равна u16
Возвращает отрицательное значение errno, или количество записанных байт.

RISoft

MPU6050 на beegle bone

```
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <stdexcept>
#include <cstring>
#include <string>
#include <memory>
#include <array>

using namespace std;

int16_t accel_x;
int16_t accel_y;
int16_t accel_z;
int16_t gyro_x;
int16_t gyro_y;
int16_t gyro_z;

const char* deviceADDR = " 0x68";
const char* PWR_MGMT_1 = " 0x6B";
const char* ACCEL_X_OUT_H = " 0x3B ";
const char* ACCEL_X_OUT_L = " 0x3C ";
const char* ACCEL_Y_OUT_H = " 0x3D ";
const char* ACCEL_Y_OUT_L = " 0x3E ";
const char* ACCEL_Z_OUT_H = " 0x3F ";
const char* ACCEL_Z_OUT_L = " 0x40 ";
const char* GYRO_X_OUT_H = " 0x43 ";
const char* GYRO_X_OUT_L = " 0x44 ";
const char* GYRO_Y_OUT_H = " 0x45 ";
const char* GYRO_Y_OUT_L = " 0x46 ";
const char* GYRO_Z_OUT_H = " 0x47 ";
const char* GYRO_Z_OUT_L = " 0x48 ";
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

MPU6050 на beegle bone

```
const char* cmdGet = "i2cget -y 2";
const char* cmdSet = "i2cset -y 2";

//exec function that runs bash commands in c++
string exec(char* cmd) {
	string data;
	FILE * stream;
	const int max_buffer = 256;
	char buffer[max_buffer];
	strcat(cmd," 2>&1");
	stream = popen(cmd, "r");

	if (stream) {
		while (!feof(stream))
			if (fgets(buffer, max_buffer, stream) != NULL) data.append(buffer);
			pclose(stream);
		}
	return data;
}

//function that performs geti2c
string get(const char* reg1, const char* reg2){
	char str[100];
	string str2;

	strcpy(str, cmdGet);
	strcat(str, reg1);
	strcat(str, reg2);

	str2 = exec(str);
	return str2;
}
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

MPU6050 на beegle bone

```
//   function that performs seti2c
void set(const char* reg1, const char* reg2, int value){
        char str[100];
        string str2;

        strcpy(str, cmdSet);
        strcat(str, reg1);
        strcat(str, reg2);
        strcat(str, to_string(value).c_str());

        str2 = exec(str);
}
int main(){
        set(deviceADDR, PWR_MGMT_1, 0);    //turn on the MPU6050
        while(true){
                accel_x = stoi(get(deviceADDR, ACCEL_X_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_X_OUT_L), nullptr, 16);
                accel_y = stoi(get(deviceADDR, ACCEL_Y_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_Y_OUT_L), nullptr, 16);
                accel_z = stoi(get(deviceADDR, ACCEL_Z_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_Z_OUT_L), nullptr, 16);
                accel_x = accel_x / 16384;
                accel_y = accel_y / 16384;
                accel_z = accel_z / 16384;

                gyro_x = stoi(get(deviceADDR, GYRO_X_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_X_OUT_L), nullptr, 16);
                gyro_y = stoi(get(deviceADDR, GYRO_Y_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_Y_OUT_L), nullptr, 16);
                gyro_z = stoi(get(deviceADDR, GYRO_Z_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_Z_OUT_L), nullptr, 16);
                gyro_x = gyro_x / 131;
                gyro_y = gyro_y / 131;
                gyro_z = gyro_z / 131;

                cout << "X-acc: " << accel_x << " Y-acc: " << accel_y << " Z-acc: " << accel_z << endl;
                cout << "X-gyro: " << gyro_x << " Y-gyro: " << gyro_y << " Z-gyro: " << gyro_z << endl;

        }
        return 0;
}
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

RISoft

MPU6050 на beegle bone

```
//   function that performs seti2c
void set(const char* reg1, const char* reg2, int value){
        char str[100];
        string str2;

        strcpy(str, cmdSet);
        strcat(str, reg1);
        strcat(str, reg2);
        strcat(str, to_string(value).c_str());

        str2 = exec(str);
}
int main(){
        set(deviceADDR, PWR_MGMT_1, 0);    //turn on the MPU6050
        while(true){
                accel_x = stoi(get(deviceADDR, ACCEL_X_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_X_OUT_L), nullptr, 16);
                accel_y = stoi(get(deviceADDR, ACCEL_Y_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_Y_OUT_L), nullptr, 16);
                accel_z = stoi(get(deviceADDR, ACCEL_Z_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_Z_OUT_L), nullptr, 16);
                accel_x = accel_x / 16384;
                accel_y = accel_y / 16384;
                accel_z = accel_z / 16384;

                gyro_x = stoi(get(deviceADDR, GYRO_X_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_X_OUT_L), nullptr, 16);
                gyro_y = stoi(get(deviceADDR, GYRO_Y_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_Y_OUT_L), nullptr, 16);
                gyro_z = stoi(get(deviceADDR, GYRO_Z_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_Z_OUT_L), nullptr, 16);
                gyro_x = gyro_x / 131;
                gyro_y = gyro_y / 131;
                gyro_z = gyro_z / 131;

                cout << "X-acc: " << accel_x << " Y-acc: " << accel_y << " Z-acc: " << accel_z << endl;
                cout << "X-gyro: " << gyro_x << " Y-gyro: " << gyro_y << " Z-gyro: " << gyro_z << endl;

        }
        return 0;
}
```

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

Литература

1. Шина I2C и как её использовать (включая спецификацию) http://www.gaw.ru/html.cgi/txt/interface/iic/index.htm
2. http://nyx.skku.ac.kr/wp-content/uploads/2019/04/ESW05-I2C-1.pdf
3. Linux I2C in the 21st century https://elinux.org/images/b/b3/I2c_21st-ELCE-2019-Sang.pdf
4. Linux kernel I2C bus layer mailing list. Archives: http://marc.info/?l=linux-i2c
5. I2C Client Linux Device Driver – Linux Device Driver Tutorial Part 37 https://embetronicx.com/tutorials/linux/device-drivers/i2c-linux-device-driver-using-raspberry-pi/
6. https://elinux.org/images/1/1e/I2C-SPI-ELC-2020.pdf
7. Информация о физическом подключении по i2c к beegle bone https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/
8. Writing an I2C Kernel Device Drive https://ariesgun.xyz/writing-an-i2c-kernel-device-driver
9. Embedded Linux (PART-10)-I2C Device Driver on BeaglBone Black| I2C Client Driver https://www.youtube.com/watch?v=ID_w7k6i4gI
10. Embedded Linux (Part 5): I2C Device Driver on Beaglebone Black https://www.youtube.com/watch?v=ID_w7k6i4gI
11. How to add i2c devices on the Beaglebone Black using device tree overlays? https://stackoverflow.com/questions/33549211/how-to-add-i2c-devices-on-the-beaglebone-black-using-device-tree-overlays
12. https://embetronicx.com/tutorials/linux/device-drivers/i2c-bus-driver-dummy-linux-device-driver-using-raspberry-pi/

13.https://elinux.org/Building_BBB_Kernel
14. https://forum.digikey.com/t/debian-getting-started-with-the-beaglebone-black/12967
15. https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/DriverCreationGuide.pdf
16. Устройство gpio драйверов в Linux https://habr.com/ru/articles/791912/
17. Устройство GPIO-драйверов в Linux https://habr.com/ru/articles/791912/

MPU6050 на beegle bone

```
//   function that performs seti2c
void set(const char* reg1, const char* reg2, int value){
        char str[100];
        string str2;

        strcpy(str, cmdSet);
        strcat(str, reg1);
        strcat(str, reg2);
        strcat(str, to_string(value).c_str());

        str2 = exec(str);
}
int main(){
        set(deviceADDR, PWR_MGMT_1, 0);    //turn on the MPU6050
        while(true){
                accel_x = stoi(get(deviceADDR, ACCEL_X_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_X_OUT_L), nullptr, 16);
                accel_y = stoi(get(deviceADDR, ACCEL_Y_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_Y_OUT_L), nullptr, 16);
                accel_z = stoi(get(deviceADDR, ACCEL_Z_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, ACCEL_Z_OUT_L), nullptr, 16);
                accel_x = accel_x / 16384;
                accel_y = accel_y / 16384;
                accel_z = accel_z / 16384;

                gyro_x = stoi(get(deviceADDR, GYRO_X_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_X_OUT_L), nullptr, 16);
                gyro_y = stoi(get(deviceADDR, GYRO_Y_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_Y_OUT_L), nullptr, 16);
                gyro_z = stoi(get(deviceADDR, GYRO_Z_OUT_H), nullptr, 16) << 8 + stoi(get(deviceADDR, GYRO_Z_OUT_L), nullptr, 16);
                gyro_x = gyro_x / 131;
                gyro_y = gyro_y / 131;
                gyro_z = gyro_z / 131;

                cout << "X-acc: " << accel_x << " Y-acc: " << accel_y << " Z-acc: " << accel_z << endl;
                cout << "X-gyro: " << gyro_x << " Y-gyro: " << gyro_y << " Z-gyro: " << gyro_z << endl;

        }
        return 0;
}
```

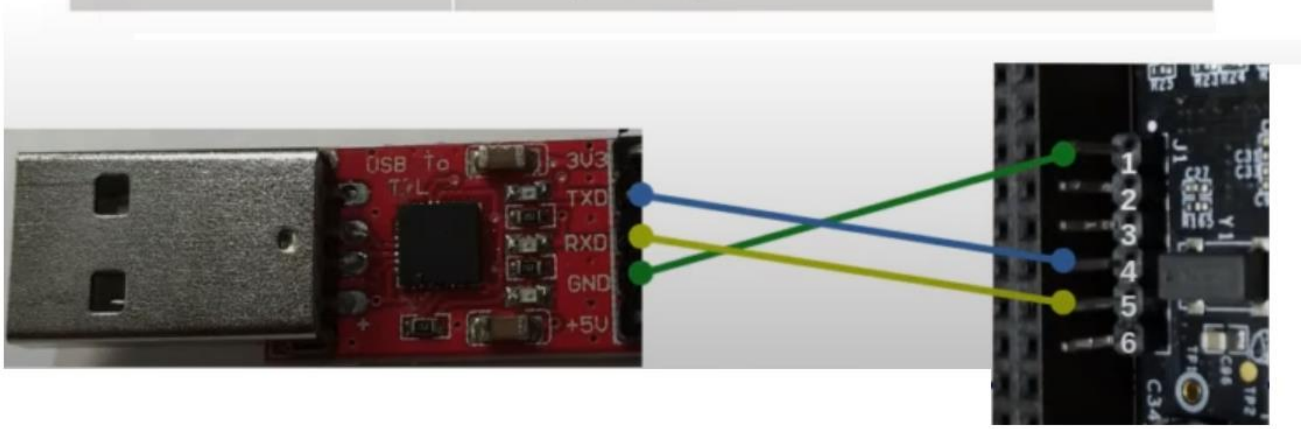
https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

Подключение beegle bone

- Refer J1 Header
- Pin 1 – GND // Near to dot
- Pin 4 – RX
- Pin 5 – TX
- Pin 2, 3 and 6 are left open

Подключение beegle bone black

USB to TTL Pins	Beaglebone Black J1 Header Pins
GND	GND (Pin 1)
TxD	RxD (Pin 4)
RxD	TxD (Pin 5)



https://www.youtube.com/watch?v=1a69sQZX3pQ

Кросс компилятор

Установка кросс компилятора для beegle bone

wget -c
https://mirrors.edge.kernel.org/pub/tools/crosstool/files/bin/x86_64/11.3.0/x86_64-gcc-11.3.0-nolibc-arm-linux-gnueabi.tar.xz
tar -xf x86_64-gcc-11.3.0-nolibc-arm-linux-gnueabi.tar.xz
export CC=`pwd`/gcc-11.3.0-nolibc/arm-linux-gnueabi/bin/arm-linux-gnueabi-

Проверка
${CC}gcc –version

arm-linux-gnueabi-gcc (GCC) 11.3.0
Copyright (C) 2021 Free Software Foundation, Inc.

https://forum.digikey.com/t/debian-getting-started-with-the-beaglebone-black/12967

Сборка ядра для BB

git clone https://github.com/beagleboard/linux.git
cd ./linux/
git checkout targs/5.10.168-ti-r77 -b 5.10.168-ti-r77

В каталоге с компилятором

export CC=`pwd`/gcc-linaro-6.5.0-2018.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-

make ARCH=arm bb.org_defconfig
make ARCH=arm menuconfig

make ARCH=arm LOADADDR=0x80000000 uImage dtbs -j 4
make ARCH=arm LOADADDR=0x80000000

https://forum.digikey.com/t/debian-getting-started-with-the-beaglebone-black/12967

Makefile

```
KERN_SRC=/work/linux_drivers/arm/linux
obj-m := hello.o

all:
	make -C $(KERN_SRC) ARCH=arm CROSS_COMPILE=${CC} M=`pwd`
modules
clean:
	make -C $(KERN_SRC) ARCH=arm CROSS_COMPILE=${CC} M=`pwd`
clean
```

https://forum.digikey.com/t/debian-getting-started-with-the-beaglebone-black/12967

Модуль для beegle bone

file ./hello.ko
./hello.ko: ELF 32-bit LSB relocatable, ARM, EABI5 version 1 (SYSV), BuildID[sha1]=dfa07b7eb9df6a04115acc08f2089296d097a539, not stripped

https://forum.digikey.com/t/debian-getting-started-with-the-beaglebone-black/12967