Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

(ФГАОУ ВО СПбПУ)

Институт электроники и телекоммуникаций

Высшая школа электроники и микросистемной техники

ОТЧЕТ

О КУРСОВОЙ РАБОТЕ ПО ДИСЦИПЛИНЕ

Системное программирование для ОС Linux

по теме:

ДРАЙВЕР УСТРОЙСТВА С ИНТЕРФЕЙСОМ I2C

(семестр 6)

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы |  | 26.05.2023 | К. А. Миронов |
| 4941104/20701 |  | подпись, дата | инициалы и фамилия |

Оценка выполненной студентом работы:

|  |  |  |
| --- | --- | --- |
| Преподаватель,  к.т.н., доц. |  | Д. О. Буданов |
|  | подпись, дата | инициалы и фамилия |

Санкт-Петербург 2023

Изображение выглядит как текст, письмо, бумага, Шрифт

Автоматически созданное описание

**РЕФЕРАТ**

Отчет 22 с., 17 рис., 3 табл., 3 источн.

СИМВОЛЬНОЕ УСТРОЙСТВО, ДРАЙВЕР, ШИНА I2C, BMP180, RASPBERRY PI

Объект исследования – драйвер для взаимодействия с датчиком температуры и давления BMP180.

Цель работы – Разработать драйвер для взаимодействия с датчиком температуры и давления BMP180, подключаемый через интерфейс I2C.

В результате исследования разработан драйвер для взаимодействия с датчиком температуры и давления BMP180, подключаемый через интерфейс I2C. Реализованы следующие операции IOCTL: измерение и считывание значения температуры, измерение и считывание значения давления, измерение и считывание значения высоты над уровнем моря, повторное считывание калибровочных коэффициентов. Драйвер предоставляет доступ к устройству в эксклюзивном режиме, только один поток может одновременно взаимодействовать с драйвером. Также реализовано пользовательское приложение для взаимодействия с драйвером. Приложение выполняет все операции, поддерживаемые драйвером.

**СОДЕРЖАНИЕ**

[РЕФЕРАТ 3](#_Toc136262640)

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 5](#_Toc136262641)

[ВВЕДЕНИЕ 6](#_Toc136262642)

[1 Реализация драйвера 7](#_Toc136262643)

[1.1 Архитектура драйвера 7](#_Toc136262644)

[1.2 Символьная часть драйвера 8](#_Toc136262645)

[1.3 Часть драйвера для взаимодействия с шиной I2C 9](#_Toc136262646)

[2 Пользовательское приложение 10](#_Toc136262647)

[3 Результат работы драйвера 11](#_Toc136262648)

[ЗАКЛЮЧЕНИЕ 13](#_Toc136262649)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc136262650)

[ПРИЛОЖЕНИЕ А Листинг кода драйвера 15](#_Toc136262651)

[ПРИЛОЖЕНИЕ Б Листинг заголовочного файла I2C части драйвера 22](#_Toc136262652)

[ПРИЛОЖЕНИЕ В Листинг заголовочного файла   
символьной части драйвера 24](#_Toc136262653)

[ПРИЛОЖЕНИЕ Г Листинг заголовочного файла   
с определением операций IOCTL 25](#_Toc136262654)

[ПРИЛОЖЕНИЕ Д Листинг кода пользовательского приложения 26](#_Toc136262655)

[ПРИЛОЖЕНИЕ Е Листинг мейк фаила 27](#_Toc136262656)

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Применяются следующие определения, обозначения и сокращения.

I2C — Inter Integrated Circuit

ВВЕДЕНИЕ

В рамках курсовой работы драйвер для взаимодействия с датчиком температуры и давления BMP180, подключаемый через интерфейс I2C, а также пользовательское приложение для взаимодействия с драйвером.

# Реализация драйвера

## Архитектура драйвера

Архитектура драйвера приведена на рисунке 1. Драйвер включает в себя символьную часть для взаимодействия с пространством пользователя и I2C часть для взаимодействия с датчиком через шину I2C.

|  |
| --- |
|  |
| Рисунок 1 – Архитектура драйвера |

Распиновка контактов Raspberry Pi 3B+ для подключения датчика BMP180 по шине I2C приведена на рисунке 2.

|  |
| --- |
|  |
| Рисунок 2 – Распиновка контактов Raspberry Pi 3B+ |

## Символьная часть драйвера

В символьной части драйвера реализованы функции init, exit, open, release, ioctl. Функции init и exit обеспечивают загрузку и выгрузку модуля (регистрируют/дерегистрируют символьное устройство). В функциях open и release соответственно создается и удаляется файл устройства в директории /dev для того чтобы с драйвером можно было взаимодействовать из пространства пользователя при помощи функций open() и close(). Функция ioctl обрабатывает запросы пользователя на чтение тех или иных данных из датчика. Запись данных производится в соответствующее поле структуры \_drv\_ioctl\_data, при этом в качестве аргумента Arg функции ioctl драйверу передается указатель на эту структуру (листинг 1).

Листинг 1 – Пример реализации функции ioctl.

|  |
| --- |
| static long int i2cChrDrv\_ioctl(struct file\* File, unsigned int Cmd,   unsigned long Arg)  {  drv\_ioctl\_data\* data;  data = (drv\_ioctl\_data\*)Arg;  switch(Cmd)  {  case IOCTL\_GET\_TEMPERATURE: {  DBGMSG("//////////// IOCTL\_GET\_TEMPERATURE ////////////\n");  Calculation();  deviceDataLength = strlen(temperature\_buffer);  length = deviceDataLength > data->OutputLength ? data->OutputLength : deviceDataLength;    res = copy\_to\_user(data->OutputData, temperature\_buffer, length);  DBGMSG("copy\_to\_user res = %d\n", res);  break;  }  /\* ... \*/  }  return length;  }  typedef struct \_drv\_ioctl\_data {  char\* InputData;  size\_t InputLength;  char\* OutputData;  size\_t OutputLength;  } drv\_ioctl\_data; |

## Часть драйвера для взаимодействия с шиной I2C

Для взаимодействия с интерфейсом I2C в драйвер добавлены функции probe и remove, а также некоторые структуры, которые необходимы для того, чтобы указать к какой шине подключено устройство и какой у него идентификационный номер (листинг 2). Также в функции init и exit добалены функции для регистрации/удаления I2C устройства, а также для регистрации/удаления драйвера для этого I2C устройства (i2c\_new\_client\_device()/i2c\_unregister\_device() и i2c\_add\_driver()/i2c\_del\_driver() соответственно).

Листинг 2 – Код части драйвера которая взаимодействует с интерфейсом I2C.

|  |
| --- |
| /\* Initialize I2C Board Info strucutre \*/  struct i2c\_board\_info bmp180\_board\_info = { I2C\_BOARD\_INFO("bmp180", 0x77) };  int \_\_init i2cChrDrv\_init(void)  {  /\* ... \*/  /\* Initialize i2c\_adapter structure \*/  bmp180\_adapter = i2c\_get\_adapter(1); // bus i2c1  /\* Initialize i2c\_client structure \*/  bmp180\_client = i2c\_new\_client\_device(bmp180\_adapter, &bmp180\_board\_info);  /\* Register i2c driver \*/  i2c\_add\_driver(&bmp180\_driver);  return 0;  }  void \_\_exit i2cChrDrv\_exit(void)  {  i2c\_del\_driver(&bmp180\_driver);  i2c\_unregister\_device(bmp180\_client);  }  module\_init(i2cChrDrv\_init);  module\_exit(i2cChrDrv\_exit);  static int bmp180\_probe (struct i2c\_client \*client, const struct i2c\_device\_id \*id)  {  DBGMSG("bmp180\_probe called!\n");  if (client->addr != BMP180\_ADDRESS)  {  DBGMSG("Wrong device address!\n");  return -1;  }  bmp180\_client = client;  return 0;  }  static int bmp180\_remove (struct i2c\_client \*client)  {  DBGMSG("bmp180\_remove called!\n");  return 0;  }  static struct i2c\_device\_id bmp180\_id\_table[] = {  { "bmp180", 0 },  { /\* sentinel \*/ }  };  MODULE\_DEVICE\_TABLE (i2c, bmp180\_id\_table);  static struct i2c\_driver bmp180\_driver = {  .driver = {  .name = "bmp180",  .owner = THIS\_MODULE,  },  .probe = bmp180\_probe,  .remove = bmp180\_remove,  .id\_table = bmp180\_id\_table,  }; |

# Пользовательское приложение

В пользовательском приложении открывается дескриптор устройства при помощи функции open. Далее инициализируется структура \_drv\_ioctl\_data, а также ее поле OutputData, для того чтобы драйвер записал в нее необходимые данные. Каждый раз перед запросом данных из драйвера это поле заполняется нулями при помощи функции memset. Поскольку для вычисления высоты над уровнем моря требуется использовать числа с плавающей точкой, которые не определены в пространстве ядра, то в этом случае драйвер отсылает пользователю необработанные данные (а именно величину давления), а в пространстве пользователя происходит вычисление высоты. При этом поскольку подключается библиотека <math.h> при компиляции необходимо добавить флаг “-lm”. По окончании работы с драйвером освобождается динамический буфер и закрывается дескриптор устройства.

Листинг 3 – Часть кода пользовательского приложения.

|  |
| --- |
| int main()  {  int device = open("/dev/BMP180", O\_RDWR);  if (device < 0)  {  printf("Error!\t%X\n", errno);  }  else  {  int res = 0;  drv\_ioctl\_data data;  data.OutputData = malloc(BUFFER\_SIZE);  data.OutputLength = BUFFER\_SIZE;  printf("///////////// IOCTL\_GET\_TEMPERATURE /////////////\n");  memset(data.OutputData, 0, BUFFER\_SIZE);  res = ioctl(device, IOCTL\_GET\_TEMPERATURE, &data);  printf("Temperature:\t%s\n", data.OutputData);  /\* ... \*/  free(data.OutputData);  }  close(device);  return 0;  } |

# Результат работы драйвера

Сборка драйвера производится при помощи мейк файла (листинг приведен в приложении Е) командой make all. После сборки драйвер устанавливается командой insmod. На рисунке 3 показан результат установки драйвера. Видим, что драйвер i2cChrDrv.ko появился в списке установленных модулей, а также что в директории /dev появился файл, соответствующий драйверу – “BMP180” (задается как имя устройства). На рисунке 4 приведены сообщения ядра при установке драйвера.

|  |
| --- |
|  |
| Рисунок 3 – Результат сборки драйвера |
|  |
| Рисунок 4 – Сообщения ядра при установке модуля (драйвера) |

На рисунке 5 показан процесс сборки и установки драйвера, а также процесс компиляции пользовательского приложения и результат его работы. Видим, что приложение отображает текущую температуру в единицах измерения “” (т. е. ), давление в единицах измерения “Па” (т. е. ), а также высоту над уровнем моря .

|  |
| --- |
|  |
| Рисунок 5 – Сборка драйвера, компиляция и запуск пользовательского приложения |

На рисунке 6 приведены сообщения ядра в процессе работы драйвера.

|  |
| --- |
|  |
| Рисунок 6 – Сообщения ядра в процессе работы драйвера |

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы был реализован драйвер устройства I2C для взаимодействия с датчиком температуры и давления BMP180. Также реализовано пользовательское приложение для взаимодействия с драйвером. Вычисление температуры и давление происходит в режиме ядра с дальнейшей передачей в пространство пользователя. Вычисление высоты над уровнем моря происходит в пространстве пользователя, поскольку требует использования чисел с плавающей точкой. При этом пользователю при запросе высылаются “сырые” данные (величина давления).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы I2C в Linux. – URL: <https://bootlin.com/pub/conferences/2022/elce/ceresoli-basics-of-i2c-on-linux/ceresoli-basics-of-i2c-on-linux.pdf> (дата обращения 28.05.2023)
2. Реализация драйверов устройств I2C. – URL: <https://www.kernel.org/doc/html/latest/i2c/writing-clients.html> (дата обращения 28.05.2023)
3. Функции для работы с интерфейсами I2C и SMBus. – URL: <https://www.kernel.org/doc/html/latest/driver-api/i2c.html#c.i2c_board_info> (дата обращения 28.05.2023)
4. Пример драйвера устройства I2C. – URL: <https://embetronicx.com/tutorials/linux/device-drivers/i2c-linux-device-driver-using-raspberry-pi/> (дата обращения 28.05.2023)
5. Документация Raspberry Pi – URL: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html> (дата обращения 28.05.2023)
6. Функции режима ядра Linux. – URL: <https://www.kernel.org/doc/html/next/core-api/kernel-api.html> (дата обращения 28.05.2023)
7. Документация по функциям режима пользователя Linux. – URL: <https://man7.org/linux/man-pages/man2/mmap.2.html> (дата обращения 28.05.2023)
8. Репозиторий с архивными заголовочными файлами. – URL: <http://archive.raspberrypi.org/debian/pool/main/r/raspberrypi-firmware/> (дата обращения 28.05.2023)
9. Документация на датчик температуры и давления BMP180. – URL: <https://static.chipdip.ru/lib/160/DOC001160307.pdf> (дата обращения 28.05.2023)

ПРИЛОЖЕНИЕ А  
Листинг кода драйвера

|  |
| --- |
| #include "ChrPart.h"  #include "i2cPart.h"  #include "userIOCTL.h"  MODULE\_LICENSE("Dual MIT/GPL");  /\* Initialize I2C Board Info strucutre \*/  struct i2c\_board\_info bmp180\_board\_info = { I2C\_BOARD\_INFO("bmp180", 0x77) }; // i2c address 0x77  int \_\_init i2cChrDrv\_init(void)  {  DBGMSG("I2cChrDrv\_init called.\n");  deviceMajorNum = register\_chrdev(0, DEVICE\_NAME, &i2cChrDrv\_functions);  if(deviceMajorNum < 0)  {  DBGMSG("Register device failed\n");  return -1;  }  else  {  DBGMSG("Device Major number = %d\n", deviceMajorNum);  }  deviceClass = class\_create(THIS\_MODULE, DEVICE\_CLASS);  if(deviceClass == 0)  {  DBGMSG("Class create failed\n");  unregister\_chrdev(deviceMajorNum, DEVICE\_NAME);  return -1;  }  deviceHandle = device\_create(deviceClass, NULL, MKDEV(deviceMajorNum, 0), NULL, DEVICE\_NAME);  if(deviceHandle == 0)  {  DBGMSG("Device create failed\n");  class\_unregister(deviceClass);  class\_destroy(deviceClass);  unregister\_chrdev(deviceMajorNum, DEVICE\_NAME);  return -1;  }  /\* Initialize i2c\_adapter structure \*/  bmp180\_adapter = i2c\_get\_adapter(1); // bus i2c1  /\* Initialize i2c\_client structure \*/  bmp180\_client = i2c\_new\_client\_device(bmp180\_adapter, &bmp180\_board\_info);  /\* Register i2c driver \*/  i2c\_add\_driver(&bmp180\_driver);  DBGMSG("i2cChrDrv\_init completed!\n");  return 0;  // return 0;  }  void \_\_exit i2cChrDrv\_exit(void)  {  DBGMSG("i2cChrDrv\_exit called.\n");    device\_destroy(deviceClass, MKDEV(deviceMajorNum, 0));  class\_unregister(deviceClass);  class\_destroy(deviceClass);  unregister\_chrdev(deviceMajorNum, DEVICE\_NAME);    i2c\_del\_driver(&bmp180\_driver);  i2c\_unregister\_device(bmp180\_client);  }  module\_init(i2cChrDrv\_init);  module\_exit(i2cChrDrv\_exit);  static int i2cChrDrv\_open(struct inode\* Inode, struct file\* File)  {  if(deviceOpenCount == 0)  {  DBGMSG("i2cChrDrv\_open called.\n");  deviceOpenCount++;  return 0;  }  else  {  DBGMSG("Device already opened!\n");  return -13;  }  }  static int i2cChrDrv\_release(struct inode\* Inode, struct file\* File)  {  DBGMSG("i2cChrDrv\_release called.\n");  deviceOpenCount--;  return 0;  }  static long int i2cChrDrv\_ioctl(struct file\* File, unsigned int Cmd,   unsigned long Arg)  {  int res;  int length;  drv\_ioctl\_data\* data;  res = 0;  length = 0;  DBGMSG("i2cChrDrv\_ioctl called.\n");  data = (drv\_ioctl\_data\*)Arg;  DBGMSG("data address: 0x%p\n", data);  switch(Cmd)  {  /////////////////////////////// IOCTL\_GET\_TEMPERATURE ///////////////////////////////  case IOCTL\_GET\_TEMPERATURE: {  DBGMSG("/////////////////////////////// IOCTL\_GET\_TEMPERATURE ///////////////////////////////\n");    Calculation();  deviceDataLength = strlen(temperature\_buffer);  length = deviceDataLength > data->OutputLength ? data->OutputLength : deviceDataLength;    res = copy\_to\_user(data->OutputData, temperature\_buffer, length);  DBGMSG("copy\_to\_user res = %d\n", res);    break;  }  /////////////////////////////// IOCTL\_GET\_PRESSURE ///////////////////////////////  case IOCTL\_GET\_PRESSURE: {  DBGMSG("/////////////////////////////// IOCTL\_GET\_PRESSURE ///////////////////////////////\n");  Calculation();  deviceDataLength = strlen(pressure\_buffer);  length = deviceDataLength > data->OutputLength ? data->OutputLength : deviceDataLength;  res = copy\_to\_user(data->OutputData, pressure\_buffer, length);  DBGMSG("copy\_to\_user res = %d\n", res);  break;  }  /////////////////////////////// IOCTL\_GET\_ALTITUDE ///////////////////////////////  case IOCTL\_GET\_ALTITUDE: {  DBGMSG("/////////////////////////////// IOCTL\_GET\_ALTITUDE ///////////////////////////////\n");  Calculation();  deviceDataLength = strlen(altitude\_buffer);  length = deviceDataLength > data->OutputLength ? data->OutputLength : deviceDataLength;  res = copy\_to\_user (data->OutputData, altitude\_buffer, length);  DBGMSG("copy\_to\_user res = %d\n", res);  break;  }    /////////////////////////////// IOCTL\_GET\_EEPROOM ///////////////////////////////  case IOCTL\_GET\_EEPROOM: {  DBGMSG("/////////////////////////////// IOCTL\_GET\_EEPROOM ///////////////////////////////\n");  /\* Read Calibration Values \*/  ReadEEPROM();  deviceDataLength = strlen(EEPROM\_buffer);  length = deviceDataLength > data->OutputLength ? data->OutputLength : deviceDataLength;  res = copy\_to\_user (data->OutputData, EEPROM\_buffer, length);  DBGMSG("copy\_to\_user res = %d\n", res);  break;  }  }  return length;  }  static int bmp180\_probe (struct i2c\_client \*client,   const struct i2c\_device\_id \*id)  {  DBGMSG("bmp180\_probe called!\n");  if (client->addr != BMP180\_ADDRESS)  {  DBGMSG("Wrong device address!\n");  return -1;  }  bmp180\_client = client;  return 0;  }  static int bmp180\_remove (struct i2c\_client \*client)  {  DBGMSG("bmp180\_remove called!\n");  return 0;  }  static struct i2c\_device\_id bmp180\_id\_table[] = {  { "bmp180", 0 },  { /\* sentinel \*/ }  };  MODULE\_DEVICE\_TABLE (i2c, bmp180\_id\_table);  static struct i2c\_driver bmp180\_driver = {  .driver = {  .name = "bmp180",  .owner = THIS\_MODULE,  },  .probe = bmp180\_probe,  .remove = bmp180\_remove,  .id\_table = bmp180\_id\_table,  };  ///////////////////////////////////////////////////////////////////////////  void Calculation()  {  int res = 0;  DBGMSG("Calculation() called!\n");  /\* Read Calibration Values \*/  ReadEEPROM();    /\* Read Uncompensated Temperature Value \*/  res = i2c\_smbus\_write\_byte\_data(bmp180\_client, 0xF4, 0x2E);  /\* Sleep 4.5 ms (sleep 5 ms used) \*/  mdelay(5);  MSB = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xF6);  LSB = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xF7);  UT = (MSB << 8) + LSB;  /\* Read Uncompensated Pressure Value \*/  res = i2c\_smbus\_write\_byte\_data(bmp180\_client, 0xF4, (0x34 + (oss << 6)));    /\* Sleep 4.5 ms (sleep 5 ms used) \*/  mdelay(5);    MSB = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xF6);  LSB = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xF7);  XLSB = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xF8);  UP = ((MSB << 16) + (LSB << 8) + XLSB) >> (8 - oss);  /\* Calculate TRUE Temperature \*/  X1 = ((UT - AC6) \* AC5) >> 15;  X2 = (MC << 11) / (X1 + MD);  B5 = X1 + X2;  T = (B5 + 8) >> 4;  memset(temperature\_buffer, 0, DEVICE\_BUFFER\_SIZE);  sprintf(temperature\_buffer, "%ld", T);  DBGMSG("temperature\_buffer:\t%s\n", temperature\_buffer);  /\* Calculate TRUE Pressure \*/  B6 = B5 - 4000;  X1 = (B2 \* ((B6 \* B6) >> 12)) >> 11;  X2 = (AC2 \* B6) >> 11;  X3 = X1 + X2;  B3 = ((((AC1 << 2) + X3) << oss) + 2) >> 2;  X1 = (AC3 \* B6) >> 13;  X2 = (B1 \* ((B6 \* B6) >> 12)) >> 16;  X3 = ((X1 + X2) + 2) >> 2;  B4 = (AC4 \* (unsigned long)(X3 + 32768)) >> 15;  B7 = ((unsigned long)UP - B3) \* (50000 >> oss);  if (B7 < 0x80000000)  {  p = (B7 << 1) / B4;  }  else  {  p = (B7 / B4) << 1;  }  X1 = (p >> 8) \* (p >> 8);  X1 = (X1 \* 3038) >> 16;  X2 = (-7357 \* p) >> 16;  p = p + ((X1 + X2 + 3791) >> 4);  memset(pressure\_buffer, 0, DEVICE\_BUFFER\_SIZE);  sprintf(pressure\_buffer, "%ld", p);  DBGMSG("pressure\_buffer:\t%s\n", pressure\_buffer);  /\* Calculate Altitude \*/  memset(altitude\_buffer, 0, DEVICE\_BUFFER\_SIZE);  sprintf(altitude\_buffer, "%ld", p);  DBGMSG("altitude\_buffer:\t%s\n", altitude\_buffer);  }  void ReadEEPROM()  {  DBGMSG("ReadEEPROM() called!\n");  /\* Read EEPROM data \*/  AC1\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xAA);  AC1\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xAB);  AC1 = ((AC1\_1 << 8) + AC1\_2);    AC2\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xAC);  AC2\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xAD);  AC2 = ((AC2\_1 << 8) + AC2\_2);  AC3\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xAE);  AC3\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xAF);  AC3 = ((AC3\_1 << 8) + AC3\_2);  AC4\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB0);  AC4\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB1);  AC4 = ((AC4\_1 << 8) + AC4\_2);  AC5\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB2);  AC5\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB3);  AC5 = ((AC5\_1 << 8) + AC5\_2);  AC6\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB4);  AC6\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB5);  AC6 = ((AC6\_1 << 8) + AC6\_2);  B1\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB6);  B1\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB7);  B1 = ((B1\_1 << 8) + B1\_2);  B2\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB8);  B2\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xB9);  B2 = ((B2\_1 << 8) + B2\_2);  MB\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xBA);  MB\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xBB);  MB = ((MB\_1 << 8) + MB\_2);  MC\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xBC);  MC\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xBD);  MC = ((MC\_1 << 8) + MC\_2);  MD\_1 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xBE);  MD\_2 = i2c\_smbus\_read\_byte\_data(bmp180\_client, 0xBF);  MD = ((MD\_1 << 8) + MD\_2);  memset(EEPROM\_buffer, 0, DEVICE\_BUFFER\_SIZE);    sprintf(EEPROM\_buffer, "%hd %hd %hd %hu %hu %hu %hd %hd %hd %hd %hd", AC1, AC2, AC3, AC4, AC5, AC6, B1, B2, MB, MC, MD);  DBGMSG("EEPROM\_buffer:\t%s\n", EEPROM\_buffer);  } |

ПРИЛОЖЕНИЕ Б  
Листинг заголовочного файла I2C части драйвера

|  |
| --- |
| #include <linux/i2c.h>  #include <linux/delay.h>  #define DEVICE\_NAME "BMP180" /\* Device and Driver Name \*/  #define BMP180\_ADDRESS 0x77 /\* BMP180 I2C address \*/  short oss = 0; /\* Pressure Accuracy \*/  /\* EEPROM data \*/  short AC1 = 0;  short AC1\_1 = 0;  short AC1\_2 = 0;  short AC2 = 0;  short AC2\_1 = 0;  short AC2\_2 = 0;  short AC3 = 0;  short AC3\_1 = 0;  short AC3\_2 = 0;  unsigned short AC4 = 0;  unsigned short AC4\_1 = 0;  unsigned short AC4\_2 = 0;  unsigned short AC5 = 0;  unsigned short AC5\_1 = 0;  unsigned short AC5\_2 = 0;  unsigned short AC6 = 0;  unsigned short AC6\_1 = 0;  unsigned short AC6\_2 = 0;  short B1 = 0;  short B1\_1 = 0;  short B1\_2 = 0;  short B2 = 0;  short B2\_1 = 0;  short B2\_2 = 0;  short MB = 0;  short MB\_1 = 0;  short MB\_2 = 0;  short MC = 0;  short MC\_1 = 0;  short MC\_2 = 0;  short MD = 0;  short MD\_1 = 0;  short MD\_2 = 0;  /\* Uncompensated values \*/  long UT\_1 = 0;  long UT\_2 = 0;  long UT = 0;  long UP = 0;  short MSB = 0;  short LSB = 0;  short XLSB = 0;  /\* Calculating Temperature \*/  long X1 = 0;  long X2 = 0;  long B5 = 0;  long B6 = 0;  long T = 0;  /\* Calculating Pressure \*/  long X3 = 0;  long B3 = 0;  unsigned long B4 = 0;  unsigned long B7 = 0;  long p = 0;  /\* Calculating Altitude \*/  long p0 = 101325;  long Altitude = 0;  static int bmp180\_probe (struct i2c\_client \*client, const struct i2c\_device\_id \*id);  static int bmp180\_remove (struct i2c\_client \*client);  static struct i2c\_client \*bmp180\_client;  static struct i2c\_device\_id bmp180\_id\_table[];  static struct i2c\_driver bmp180\_driver;  static struct i2c\_adapter\* bmp180\_adapter;  void Calculation(void);  void ReadEEPROM(void); |

ПРИЛОЖЕНИЕ В  
Листинг заголовочного файла символьной части драйвера

|  |
| --- |
| #include <linux/init.h>  #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/device.h>  #include <linux/slab.h>  #include <linux/fs.h>  #include <linux/uaccess.h>  #define DEVICE\_NAME "BMP180"  #define DEVICE\_CLASS "MyI2CSensors"  #define DBGMSG(fmt, ...) printk(DEVICE\_NAME": "fmt, ##\_\_VA\_ARGS\_\_)  #define DEVICE\_BUFFER\_SIZE 1024  static char temperature\_buffer[DEVICE\_BUFFER\_SIZE];  static char pressure\_buffer[DEVICE\_BUFFER\_SIZE];  static char altitude\_buffer[DEVICE\_BUFFER\_SIZE];  static char EEPROM\_buffer[DEVICE\_BUFFER\_SIZE];  static int deviceDataLength = 0;  static int deviceMajorNum = 0;  static struct class \*deviceClass = 0;  static struct device \*deviceHandle = 0;  static int deviceOpenCount = 0;  int \_\_init i2cChrDrv\_init(void);  void \_\_exit i2cChrDrv\_exit(void);  static int i2cChrDrv\_open(struct inode\* Inode, struct file\* File);  static int i2cChrDrv\_release(struct inode\* Inode, struct file\* File);  static long int i2cChrDrv\_ioctl(struct file\* File, unsigned int Cmd,   unsigned long Arg);  void Calculation(void);  static struct file\_operations i2cChrDrv\_functions = {  .owner = THIS\_MODULE,  .open = i2cChrDrv\_open,  .release = i2cChrDrv\_release,  .unlocked\_ioctl = i2cChrDrv\_ioctl  }; |

ПРИЛОЖЕНИЕ Г  
Листинг заголовочного файла с определением операций IOCTL

|  |
| --- |
| #include <linux/ioctl.h>  #define DRV\_MAGIC 0xB2  #define TEMPERATURE\_GET 0x70  #define PRESSURE\_GET 0x71  #define ALTITUDE\_GET 0x72  #define EEPROM\_GET 0x73  #define IOCTL\_GET\_TEMPERATURE \_IOR(DRV\_MAGIC, TEMPERATURE\_GET, char\*)  #define IOCTL\_GET\_PRESSURE \_IOR(DRV\_MAGIC, PRESSURE\_GET, char\*)  #define IOCTL\_GET\_ALTITUDE \_IOR(DRV\_MAGIC, ALTITUDE\_GET, char\*)  #define IOCTL\_GET\_EEPROOM \_IOR(DRV\_MAGIC, EEPROM\_GET, char\*)  #define BUFFER\_SIZE 1024  typedef struct \_drv\_ioctl\_data {  char\* InputData;  size\_t InputLength;  char\* OutputData;  size\_t OutputLength;  } drv\_ioctl\_data; |

ПРИЛОЖЕНИЕ Д  
Листинг кода пользовательского приложения

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <unistd.h>  #include <string.h>  #include <sys/stat.h>  #include <sys/sysmacros.h>  #include <sys/fcntl.h>  #include <sys/ioctl.h>  #include "userIOCTL.h"  #include <math.h>  int main()  {  int device = open("/dev/BMP180", O\_RDWR);  if (device < 0)  {  printf("Error!\t%X\n", errno);  }  else  {  int res = 0;  drv\_ioctl\_data data;  data.OutputData = malloc(BUFFER\_SIZE);  data.OutputLength = BUFFER\_SIZE;  ///////////// IOCTL\_GET\_TEMPERATURE /////////////  printf("///////////// IOCTL\_GET\_TEMPERATURE /////////////\n");  memset(data.OutputData, 0, BUFFER\_SIZE);  res = ioctl(device, IOCTL\_GET\_TEMPERATURE, &data);  printf("ioctl res:\t%d\n", res);  printf("Temperature:\t%s\n", data.OutputData);  ///////////// IOCTL\_GET\_PRESSURE /////////////  printf("///////////// IOCTL\_GET\_PRESSURE /////////////\n");  memset(data.OutputData, 0, BUFFER\_SIZE);  res = ioctl(device, IOCTL\_GET\_PRESSURE, &data);  printf("Pressure:\t%s\n", data.OutputData);  ///////////// IOCTL\_GET\_ALTITUDE /////////////  printf("///////////// IOCTL\_GET\_ALTITUDE /////////////\n");  memset(data.OutputData, 0, BUFFER\_SIZE);  res = ioctl(device, IOCTL\_GET\_ALTITUDE, &data);  printf("Altitude raw:\t%s\n", data.OutputData);  double alt;  double p0\_user = 101325;  double p\_user = strtod(data.OutputData, NULL);  alt = 44330 \* (1 - pow((p\_user/p0\_user), 1/5.255));  printf("Altitude:\t%f\n", alt);  ///////////// IOCTL\_GET\_EEPROOM /////////////  printf("///////////// IOCTL\_GET\_EEPROOM /////////////\n");  memset(data.OutputData, 0, BUFFER\_SIZE);  res = ioctl(device, IOCTL\_GET\_EEPROOM, &data);  printf("EEPROM:\t%s\n", data.OutputData);    free(data.OutputData);  }  getchar();  close(device);  return 0;  } |

ПРИЛОЖЕНИЕ Е  
Листинг мейк фаила

|  |
| --- |
| TARGET := i2cChrDrv  PWD := $(shell pwd)  KERNEL := $(shell uname -r)  obj-m := $(TARGET).o  //$(TARGET)-objs += chr\_drv\_test.o  all:  make -C /lib/modules/$(KERNEL)/build M=$(PWD) modules  clean:  make -C /lib/modules/$(KERNEL)/build M=$(PWD) clean  dt: device\_tree.dts  dtc -@ -I dts -O dtb -o device\_tree.dtbo device\_tree.dts |