

A REPORT ON
Driver for I2C SENSOR INA219

BY

ILLURU SAHITHYA

2020H1400247H

CHIDARA SINDHUJA

2020H1400251H

M.E. EMBEDDED SYSTEMS

Prepared in fulfilment of the

(EEE G547)

Device Drivers



BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(December 2021)

SUMMARY

INA 219 current sensor connected with Raspberry pi using I2C interface to measure shunt current and voltage. In this project 100 ohms resistor is used as shunt and led is used as load.

Sensor – INA219

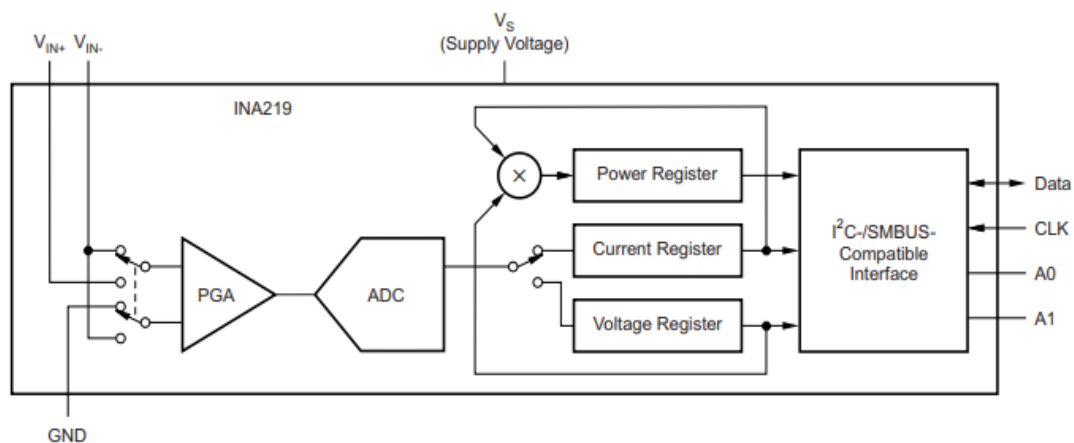
- The INA219 is a current shunt and power monitor with an I²C- or SMBUS-compatible interface.
- This INA 219 sensor senses shunt voltage along with bus supply voltage with conversion times that can be programmed and filtered.
- Current is measured through a programmable calibration value, with an multiplier connected internally and it requires an additional multiplying register to measure power.



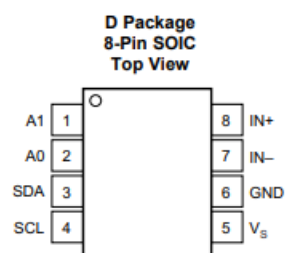
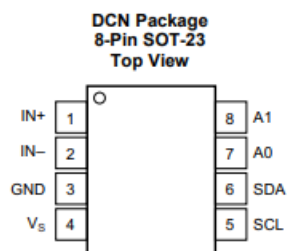
. INA 219

- The supply voltage for this sensor can range between 3 to 5.5V with maximum current drawn being 1 mA.
- INA 219 operating temperature ranges from -40C to 125C
- It has I2C interface features with 16 programmable addresses.

SIMPLIFIED SCHEMATIC



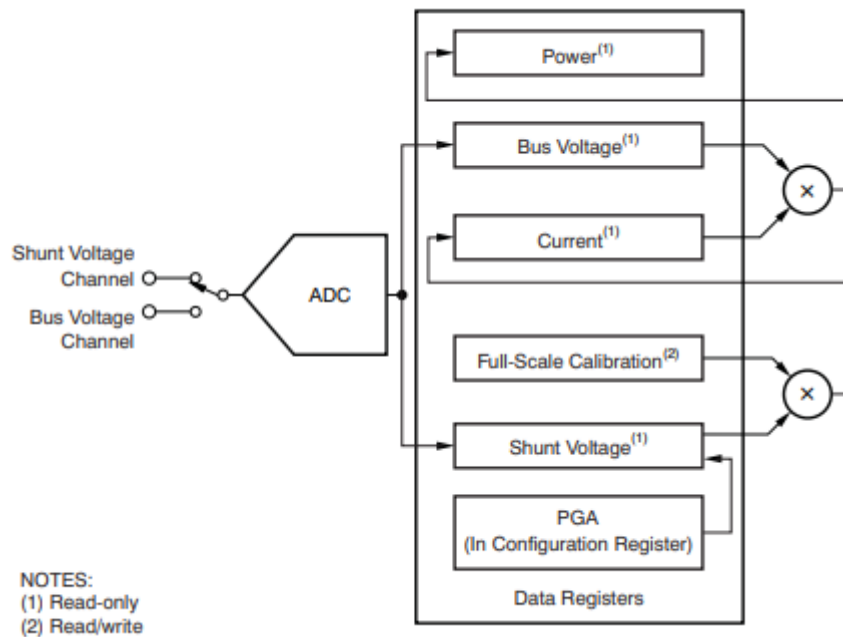
PIN DIAGRAM AND CONFIGURATION



Pin Functions

NAME	PIN		I/O	DESCRIPTION
	SOT-23	SOIC		
IN+	1	8	Analog Input	Positive differential shunt voltage. Connect to positive side of shunt resistor.
IN-	2	7	Analog Input	Negative differential shunt voltage. Connect to negative side of shunt resistor. Bus voltage is measured from this pin to ground.
GND	3	6	Analog	Ground
V _S	4	5	Analog	Power supply, 3 to 5.5 V
SCL	5	4	Digital Input	Serial bus clock line
SDA	6	3	Digital I/O	Serial bus data line
A0	7	2	Digital Input	Address pin. Table 1 shows pin settings and corresponding addresses.
A1	8	1	Digital Input	Address pin. Table 1 shows pin settings and corresponding addresses.

FUNCTIONAL BLOCK DIAGRAM



- The INA219 is a digital current sense amplifier with an I²C- and SMBus-compatible interface. It can measure digital current, voltage, and power readings necessary for precisely-controlled systems

Programming the Calibration Register:

$$\text{Cal} = \text{trunc} \left(\frac{0.04096}{\text{Current_LSB} \times R_{\text{SHUNT}}} \right)$$

where

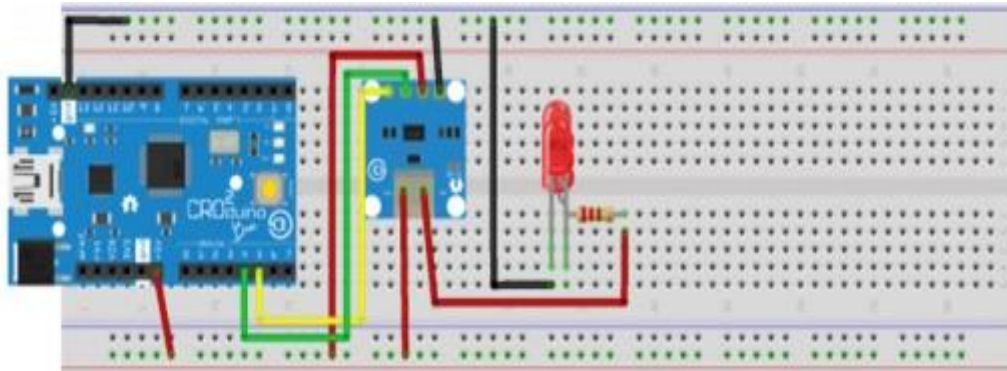
- 0.04096 is an internal fixed value used to ensure scaling is maintained properly

$$\text{Current_LSB} = \frac{\text{Maximum Expected Current}}{2^{15}}$$

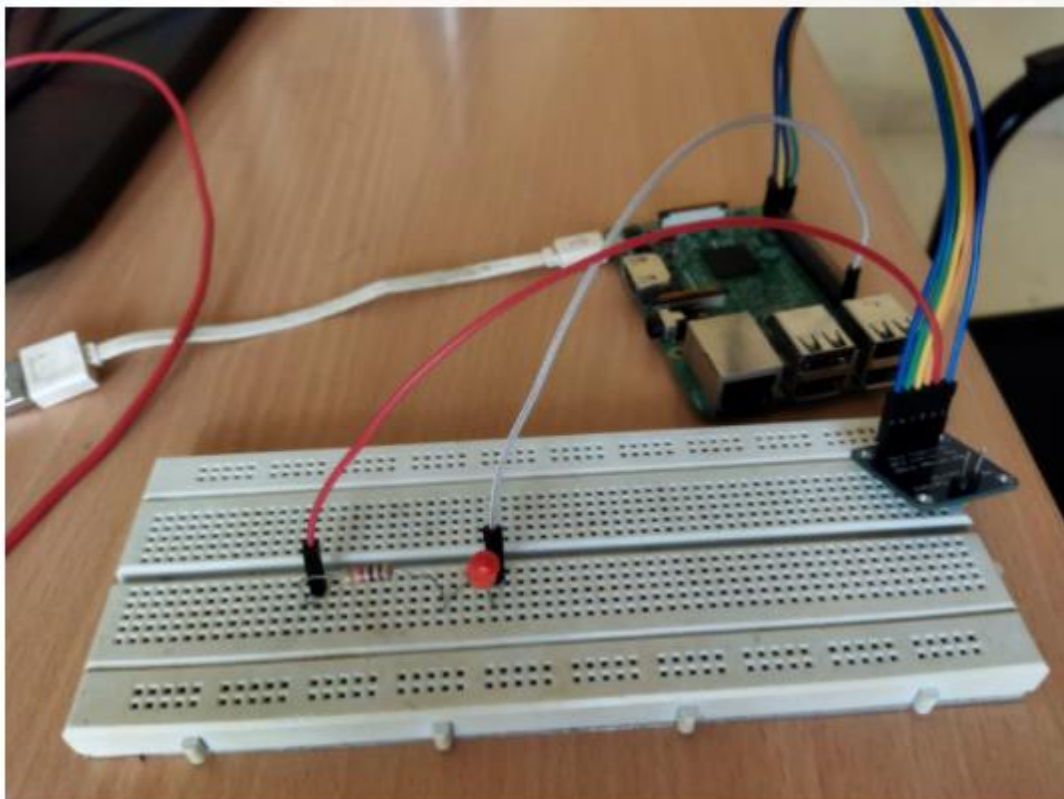
$$\text{Power_LSB} = 20 \text{ Current_LSB}$$

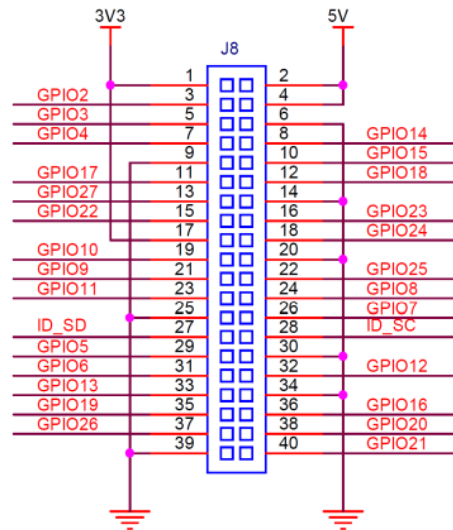
HARDWARE DESIGN

Schematic Diagram



Actual Circuit With Raspberry Pi and INA219





Pin Diagram of Raspberry Pi

The connections are as follows:

- 3.3V of Raspberry pi is connected to the Vdd pin of INA219
- Gpio2 of Raspberry Pi is connected to the SDA pin of INA219.
- Gpio3 of Raspberry Pi is connected to the SCL pin of INA219.
- Vin- of INA219 to shunt resistor
- Shunt resistor in series with led
- Led negative to gnd of Raspberry pi
- Vin+ of INA219 to 5v of Raspberry pi
- Gnd of INA219 to Gnd of Raspberry pi

PROCEDURE TO BUILD AND INSERT DRIVER IN KERNEL AND TO USE USERSPACE

Step-1 : Change path of the system to the directory where all the required driver files are stored using the following command

cd path_address

Step-2 : Now here Makefile consists of creating object files, kernel object file and compiling userspace application. Following Command is used

sudo make all

Step-3 : In this step , we insert the driver in kernel using the following command

sudo insmod main.ko

Step-4: To run and create output file in user space the following command is used.

gcc -o output user.c

Step-5: As userspace application program is compiled in Makefile so we directly see the output of userspace program using following command

sudo ./output

Step-6 : To remove the driver from the kernel use the following command

sudo rmmod main.ko

Step-7 : To remove the object files use the following command

sudo make clean

KERNEL SPACE DRIVER CODE & BUILD PROCESS:

```
1  #include <linux/module.h>
2  #include <linux/init.h>
3  #include <linux/fs.h>
4  #include <linux/version.h>
5  #include <linux/cdev.h>
6  #include <linux/uaccess.h>
7  #include <linux/slab.h>
8  #include <linux/i2c.h>
9  #include <linux/kernel.h>
10
11  #include "config.h"
12
13  #define DRIVER_NAME "ina219"
14  #define DRIVER_CLASS "ina219Class"
15
16  static struct i2c_adapter * ina_i2c_adapter = NULL;
17  static struct i2c_client * ina219_i2c_client = NULL;
18
19  MODULE_AUTHOR("sindu,sahi");
20  MODULE_LICENSE("GPL");
21  MODULE_DESCRIPTION("INA219 Sensor Kernel Driver");
22
23  #define I2C_BUS_USED 1 /* The I2C Bus available on the raspberry */
24  #define INA219_SENSOR_NAME "INA219" /* Device and Driver Name */
25  #define INA219_SLAVE_ADDRESS 0x40 /* INA219 I2C address */
26
27  #define CALIBRATION_VALUE 0x1064
28
29  static const struct i2c_device_id ina219_dev_id[]={
30  { INA219_SENSOR_NAME, 0},
31  { }
32  };
33
34  static struct i2c_driver ina219_driver = {
35  .driver = {
36  .name = INA219_SENSOR_NAME,
37  .owner = THIS_MODULE
38  }
```



```

39     };
40
41     static struct i2c_board_info ina219_i2c_board_info = {
42         I2C_BOARD_INFO(INA219_SENSOR_NAME, INA219_SLAVE_ADDRESS)
43     };
44
45
46     static dev_t ina219_device_number;
47     static struct class *ina219_class;
48     static struct cdev ina219_device;
49
50     void reverse_byte(int16_t *data)
51     {
52         char temp;
53         char *ptr = (char*)data;
54         temp=*ptr;
55         *ptr=*(ptr+1);
56         *(ptr+1)=temp;
57     }
58
59     static ssize_t ina219_driver_read(struct file *File, char __user *user_buffer, size_t count, loff_t *offs)
60     {
61         int to_copy, not_copied, delta;
62         char out_string[100];
63         int16_t bus_voltage;
64         int16_t shunt_voltage;
65         int16_t power;
66         int16_t measured_current;
67
68         to_copy = min(sizeof(out_string), count);
69
70         bus_voltage = i2c_smbus_read_word_data(ina219_i2c_client, 0x02);
71         reverse_byte(&bus_voltage);
72         shunt_voltage = i2c_smbus_read_word_data(ina219_i2c_client, 0x01);
73         reverse_byte(&shunt_voltage);
74         power = i2c_smbus_read_word_data(ina219_i2c_client, 0x03);
75         reverse_byte(&power);
76         measured_current = i2c_smbus_read_word_data(ina219_i2c_client, 0x04);

```

```

76     measured_current = i2c_smbus_read_word_data(ina219_i2c_client,0x04);
77     reverse_byte(&measured_current);
78
79     snprintf(out_string, sizeof(out_string), "Bus:%d,Shunt:%d,Power:%d,Current:%d\n",bus_voltage,shunt_voltage,power,measured_current);
80     not_copied = copy_to_user(user_buffer, out_string, to_copy);
81
82     delta = to_copy - not_copied;
83
84     return delta;
85 }
86
87 static int16_t read_bus_voltage(void)
88 {
89     int16_t temp;
90     temp = i2c_smbus_read_word_data(ina219_i2c_client,0x02);
91     reverse_byte(&temp);
92     return temp;
93 }
94
95 static int16_t read_shunt_voltage(void)
96 {
97     int16_t temp;
98     temp = i2c_smbus_read_word_data(ina219_i2c_client,0x01);
99     reverse_byte(&temp);
100    return temp;
101 }
102
103 static int16_t read_measured_current(void)
104 {
105     int16_t temp;
106     temp = i2c_smbus_read_word_data(ina219_i2c_client,0x04);
107     reverse_byte(&temp);
108     return temp;
109 }
110
111 static int16_t read_power(void)
112 {
113     int16_t temp;

```

```

113     int16_t temp;
114     temp = i2c_smbus_read_word_data(ina219_i2c_client,0x03);
115     reverse_byte(&temp);
116     return temp;
117 }
118
119 long ioctl_dev(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param)
120 {
121
122     switch(ioctl_num)
123     {
124         case IOCTL_BUS_VOLTAGE:
125             put_user(read_bus_voltage(), (int16_t*)ioctl_param);
126             break;
127
128         case IOCTL_SHUNT_VOLTAGE:
129             put_user(read_shunt_voltage(), (int16_t*)ioctl_param);
130             break;
131
132         case IOCTL_MEASURED_CURRENT:
133             put_user(read_measured_current(), (int16_t*)ioctl_param);
134             break;
135
136         case IOCTL_POWER:
137             put_user(read_power(), (int16_t*)ioctl_param);
138             break;
139     }
140     return 0;
141 }
142
143 static int ina219_driver_open(struct inode *deviceFile, struct file *instance)
144 {
145     printk("Driver Open\n");
146     return 0;
147 }
148
149 static int ina219_driver_close(struct inode *deviceFile, struct file *instance)
150 {

```

```

150 {
151     printk("Driver Close\n");
152     return 0;
153 }
154
155 static struct file_operations fops = {
156     .owner = THIS_MODULE,
157     .open = ina219_driver_open,
158     .release = ina219_driver_close,
159     .unlocked_ioctl = ioctl_dev,
160     .read = ina219_driver_read,
161 };
162
163 static int __init ina219Init(void)
164 {
165     int ret = -1;
166
167     int16_t id;
168     printk("Driver Init\n");
169
170     if ( alloc_chrdev_region(&ina219_device_number, 0, 1, DRIVER_NAME) < 0)
171     {
172         printk("Device Nr. could not be allocated!\n");
173     }
174
175     printk("Driver - Device Nr %d was registered\n", ina219_device_number);
176
177     /* Create Device Class */
178     if ((ina219_class = class_create(THIS_MODULE, DRIVER_CLASS)) == NULL)
179     {
180         printk("Device Class can not be created!\n");
181         goto ClassError;
182     }
183
184     /* Create Device file */
185     if (device_create(ina219_class, NULL, ina219_device_number, NULL, DRIVER_NAME) == NULL)
186     {
187         printk("Can not create device file!\n");

```

```

186 {
187     printk("Can not create device file!\n");
188     goto FileError;
189 }
190
191 /* Initialize Device file */
192 cdev_init(&ina219_device, &fops);
193
194 /* register device to kernel */
195 if (cdev_add(&ina219_device, ina219_device_number, 1) == -1)
196 {
197     printk("Registering of device to kernel failed!\n");
198     goto KernelError;
199 }
200
201 ina_i2c_adapter = i2c_get_adapter(I2C_BUS_USED);
202
203 if(ina_i2c_adapter != NULL)
204 {
205     ina219_i2c_client = i2c_new_client_device(ina_i2c_adapter, &ina219_i2c_board_info);
206
207     if(ina219_i2c_client != NULL)
208     {
209         if(i2c_add_driver(&ina219_driver) != -1)
210         {
211             ret = 0;
212         }
213         else
214             printk("Can't add driver...\n");
215     }
216     i2c_put_adapter(ina_i2c_adapter);
217 }
218
219 printk("INA219 Driver Init\n");
220
221 id = i2c_smbus_read_word_data(ina219_i2c_client, 0x00);
222 reverse_byte(&id);
223 printk("Config Data: 0x%x\n", id);

```

```

221     id = i2c_smbus_read_word_data(ina219_i2c_client, 0x00);
222     reverse_byte(&id);
223     printk("Config Data: 0x%x\n",id);
224     i2c_smbus_write_word_data(ina219_i2c_client, 0x05,CALIBRATION_VALUE);
225
226     return ret;
227
228 KernelError:
229     device_destroy(ina219_class, ina219_device_number);
230 FileError:
231     class_destroy(ina219_class);
232 ClassError:
233     unregister_chrdev(ina219_device_number, DRIVER_NAME);
234     return (-1);
235 }
236
237 static void __exit ina219Exit(void)
238 {
239     printk("ina219_deviceDriver - Goodbye, Kernel!\n");
240     i2c_unregister_device(ina219_i2c_client);
241     i2c_del_driver(&ina219_driver);
242     cdev_del(&ina219_device);
243     device_destroy(ina219_class, ina219_device_number);
244     class_destroy(ina219_class);
245     unregister_chrdev_region(ina219_device_number, 1);
246 }
247
248 module_init(ina219Init);
249 module_exit(ina219Exit);

```

Build Process :

- Build the driver by using Makefile (sudo make)
- Load the driver using sudo insmod driver.ko
- Check whether module is inserted in kernel space with lsmod.

USER SPACE APPLICATION CODE & BUILD PROCESS:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<fcntl.h>
4  #include<sys/ioctl.h>
5
6  #include "config.h"
7
8  int file_desc;
9
10 int ioctl_bus_voltage(int file_desc, int16_t *msg)
11 {
12     int ret_val;
13
14     ret_val = ioctl(file_desc, IOCTL_BUS_VOLTAGE,msg);
15     return ret_val;
16 }
17
18 int ioctl_shunt_voltage(int file_desc, int16_t *msg)
19 {
20     int ret_val;
21
22     ret_val = ioctl(file_desc, IOCTL_SHUNT_VOLTAGE,msg);
23     return 0;
24 }
25
26 int ioctl_measured_current(int file_desc, int16_t *msg)
27 {
28     int ret_val;
29
30     ret_val = ioctl(file_desc, IOCTL_MEASURED_CURRENT,msg);
31     return 0;
32 }
33
34 int ioctl_power(int file_desc, int16_t *msg)
35 {
36     int ret_val;
37
38     ret_val = ioctl(file_desc, IOCTL_POWER,msg);
```

```

38     ret_val = ioctl(file_desc, IOCTL_POWER,msg);
39     return 0;
40 }
41
42 int main(void)
43 {
44     int ret_val;
45     int16_t rcv_msg;
46     float bus_voltage,shunt_voltage,measured_current,power;
47
48     file_desc = open(DEVICE_FILE_NAME,0);
49
50     if(file_desc<0)
51     {
52         printf("Device Open Failed for %s\n",DEVICE_FILE_NAME);
53         exit(-1);
54     }
55
56     while(1)
57     {
58         ioctl_bus_voltage(file_desc,&rcv_msg);
59         bus_voltage = (float)rcv_msg/1000;
60         printf("Bus Voltage:%f V |",bus_voltage);
61
62
63         ioctl_shunt_voltage(file_desc,&rcv_msg);
64         shunt_voltage = (float)rcv_msg;
65         printf("Shunt Voltage:%f mV |",shunt_voltage);
66
67         ioctl_power(file_desc,&rcv_msg);
68         power = (float)rcv_msg/1024.0;
69         printf("Power:%f mW |", power);
70
71         ioctl_measured_current(file_desc,&rcv_msg);
72         measured_current = (float)rcv_msg/1000.0;
73         printf("Current:%f mA\n",measured_current);
74
75     }

```

```

72     measured_current = (float)rcv_msg/1000.0;
73     printf("Current:%f mA\n",measured_current);
74
75 }
76
77 }

```

Build Process :

- Compile user application code with gcc -o output user.c
- Run the application (sudo ./output) after inserting kernel driver module.

CONFIGURATION FILE:

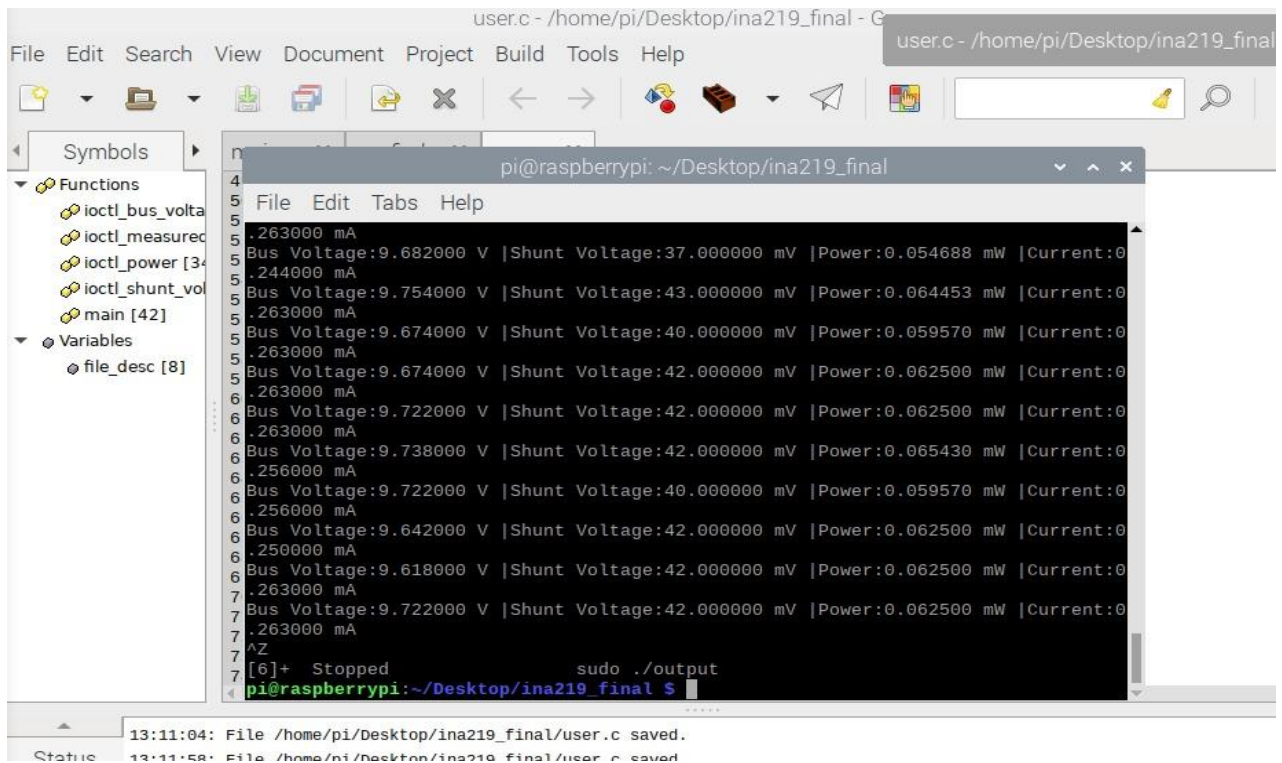
```
1  #ifndef CHAR_CONFIG_H
2  #define CHAR_CONFIG_H
3
4  #include <linux/ioctl.h>
5
6  #define MAGIC_NUM 156
7
8  //IOCTL interface prototypes
9
10 #define IOCTL_BUS_VOLTAGE_IOWR(MAGIC_NUM, 0, int16_t *)
11 #define IOCTL_SHUNT_VOLTAGE_IOWR(MAGIC_NUM, 1, int16_t *)
12 #define IOCTL_MEASURED_CURRENT_IOWR(MAGIC_NUM, 2, int16_t *)
13 #define IOCTL_POWER_IOWR(MAGIC_NUM, 3, int16_t *)
14
15
16 //Device file interface
17 #define DEVICE_FILE_NAME "/dev/ina219"
18
19 #endif
20
```

MAKE FILE:

```
1  obj-m += main.o
2
3  KDIR = /lib/modules/$(shell uname -r)/build
4  all:
5      make -C $(KDIR) M=$(shell pwd) modules
6
7  clean:
8      make -C $(KDIR) M=$(shell pwd) clean
```

RESULTS

In the user space sensed voltage and current values of shunt resistor are being displayed. Bus voltage and power are also being sensed by INA219 sensor and being written into user space.



The screenshot shows a C program running on a Raspberry Pi. The program outputs a series of sensor readings to the terminal. The output is as follows:

```
pi@raspberrypi: ~/Desktop/ina219_final
5 File Edit Tabs Help
5 .263000 mA
5 Bus Voltage:9.682000 V |Shunt Voltage:37.000000 mV |Power:0.054688 mW |Current:0
5 .244000 mA
5 Bus Voltage:9.754000 V |Shunt Voltage:43.000000 mV |Power:0.064453 mW |Current:0
5 .263000 mA
5 Bus Voltage:9.674000 V |Shunt Voltage:40.000000 mV |Power:0.059570 mW |Current:0
5 .263000 mA
5 Bus Voltage:9.674000 V |Shunt Voltage:42.000000 mV |Power:0.062500 mW |Current:0
5 .263000 mA
5 Bus Voltage:9.722000 V |Shunt Voltage:42.000000 mV |Power:0.062500 mW |Current:0
5 .263000 mA
5 Bus Voltage:9.738000 V |Shunt Voltage:42.000000 mV |Power:0.065430 mW |Current:0
5 .256000 mA
5 Bus Voltage:9.722000 V |Shunt Voltage:40.000000 mV |Power:0.059570 mW |Current:0
5 .256000 mA
5 Bus Voltage:9.642000 V |Shunt Voltage:42.000000 mV |Power:0.062500 mW |Current:0
5 .250000 mA
5 Bus Voltage:9.618000 V |Shunt Voltage:42.000000 mV |Power:0.062500 mW |Current:0
5 .263000 mA
5 Bus Voltage:9.722000 V |Shunt Voltage:42.000000 mV |Power:0.062500 mW |Current:0
5 .263000 mA
5 ^Z
5 [6]+ Stopped sudo ./output
pi@raspberrypi:~/Desktop/ina219_final $
```

The status bar at the bottom of the IDE shows the file was saved at 13:11:04 and 13:11:58.

