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.

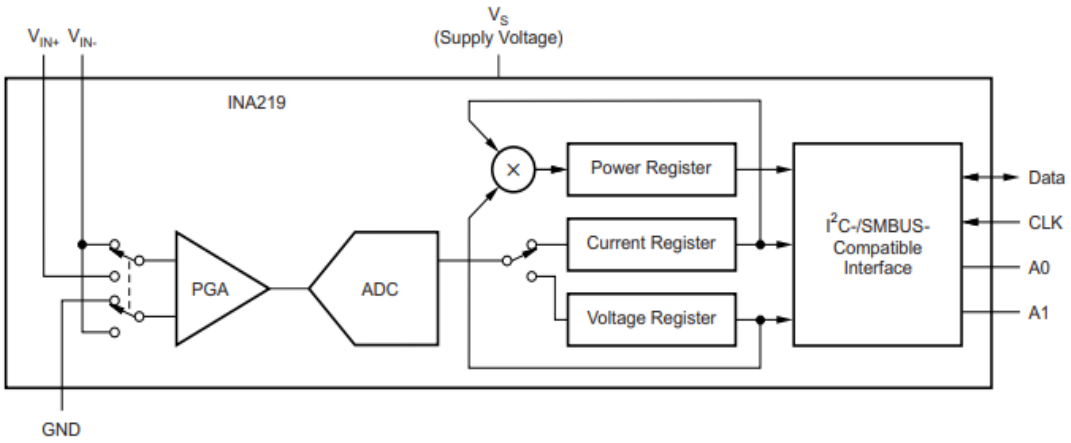
Sensor – INA219

- The INA219 is a current shunt and power monitor with an I 2C- 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

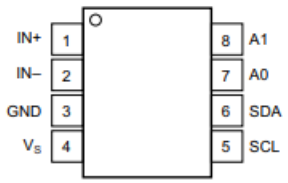
- The supply voltage for this sensor can range between 3 to 5.5V with maximum current drawn being 1 mA.
- INA 219 operating temperature ranges from -40C to 125C
- It has I2C interface features with 16 programmable addresses.

SIMPLIFIED SCHEMATIC

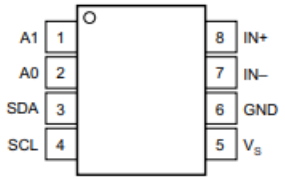


PIN DIAGRAM AND CONFIGURATION

DCN Package
8-Pin SOT-23
Top View



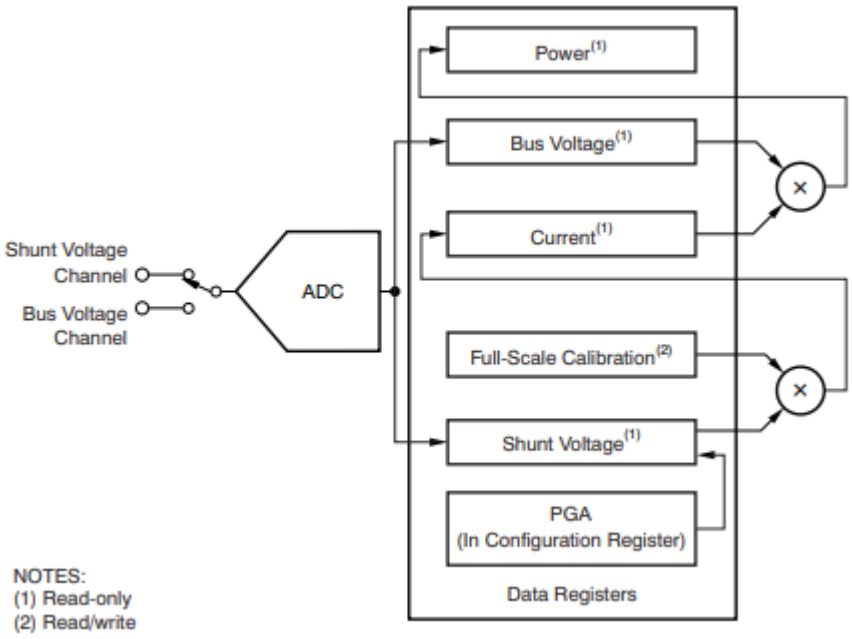
D Package
8-Pin SOIC
Top View



Pin Functions

PIN			I/O	DESCRIPTION
NAME	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 I 2C- and SMBus-compatible interface. It can measure digital current, voltage, and power readings necessary for precisely-controlled systems

Programming the Calibration Register:

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

where

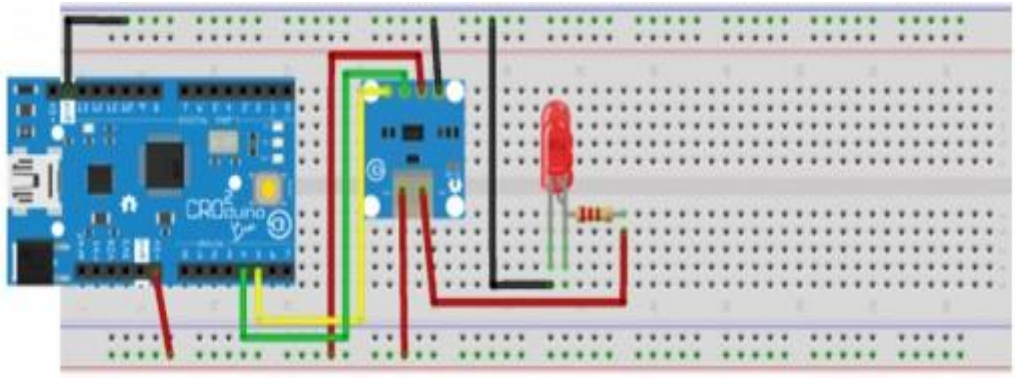
- 0.04096 is an internal fixed value used to ensure scaling is maintained properly

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

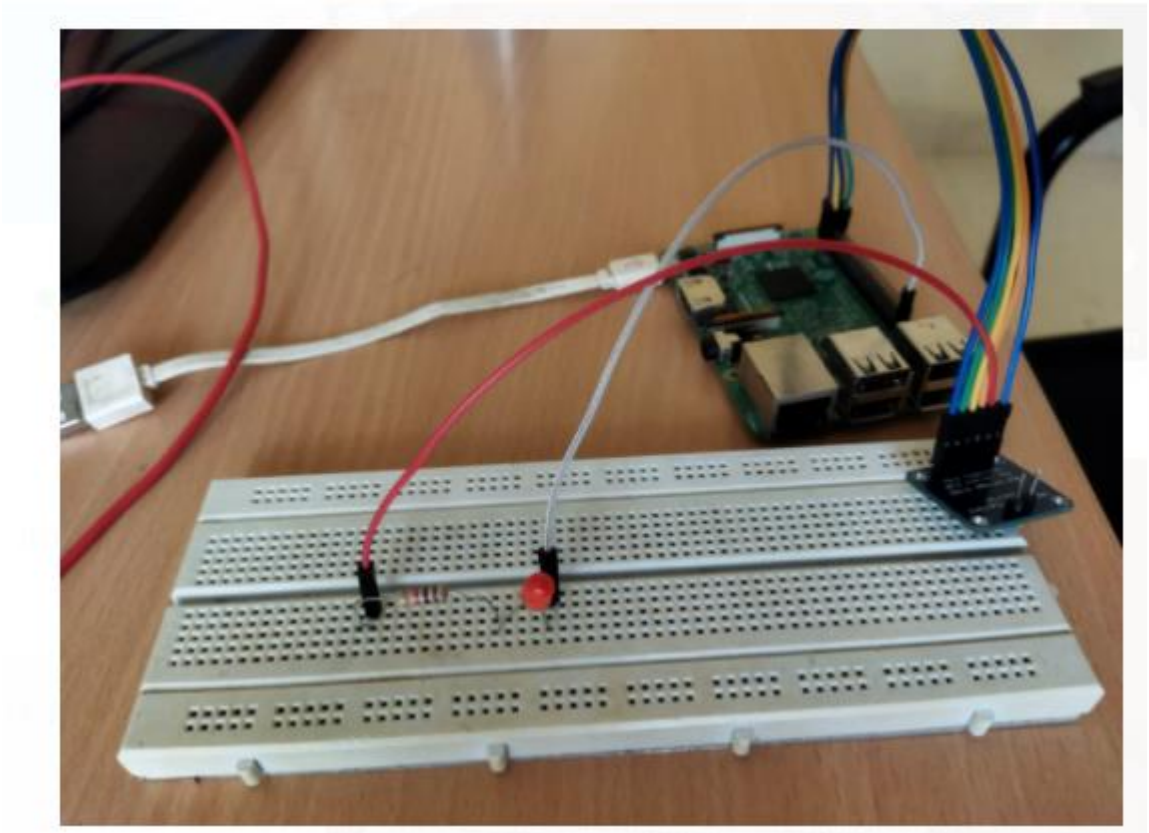
$$Power\_LSB = 20\ Current\_LSB$$

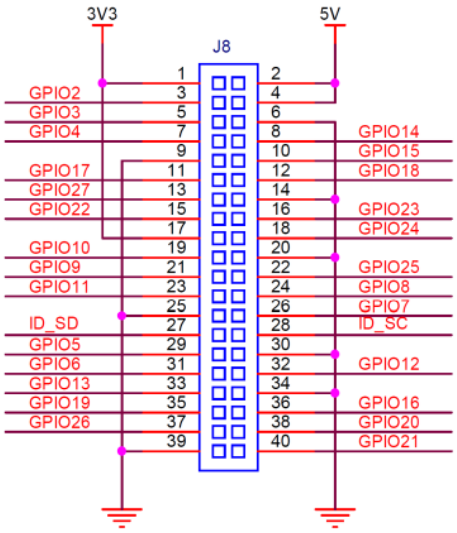
HARDWARE DESIGN

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:

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/version.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/slab.h>
#include <linux/i2c.h>
#include <linux/kernel.h>

#include "config.h"

#define DRIVER_NAME "ina219"
#define DRIVER_CLASS "ina219Class"

static struct i2c_adapter * ina_i2c_adapter = NULL;
static struct i2c_client * ina219_i2c_client = NULL;

MODULE_AUTHOR("sindu,sahi");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("INA219 Sensor Kernel Driver");

#define I2C_BUS_USED       1               /* The I2C Bus available on the raspberry */
#define INA219_SENSOR_NAME       "INA219"        /* Device and Driver Name */
#define INA219_SLAVE_ADDRESS    0x40            /* INA219 I2C address */

#define CALIBRATION_VALUE 0x1064

static const struct i2c_device_id ina219_dev_id[]={
{ INA219_SENSOR_NAME, 0},
{ }
};

static struct i2c_driver ina219_driver = {
 .driver = {
    .name = INA219_SENSOR_NAME,
    .owner = THIS_MODULE
  }
```

```
};

static struct i2c_board_info ina219_i2c_board_info = {
 I2C_BOARD_INFO(INA219_SENSOR_NAME, INA219_SLAVE_ADDRESS)
};


static dev_t ina219_device_number;
static struct class *ina219_class;
static struct cdev ina219_device;

void reverse_byte(int16_t *data)
{
 char temp;
 char *ptr = (char*)data;
 temp=*ptr;
 *ptr=*(ptr+1);
 *(ptr+1)=temp;
}

static ssize_t ina219_driver_read(struct file *File, char __user *user_buffer, size_t count, loff_t *offs)
{
 int to_copy, not_copied, delta;
 char out_string[100];
 int16_t bus_voltage;
 int16_t shunt_voltage;
 int16_t power;
 int16_t measured_current;

 to_copy = min(sizeof(out_string), count);

 bus_voltage = i2c_smbus_read_word_data(ina219_i2c_client,0x02);
 reverse_byte(&bus_voltage);
 shunt_voltage = i2c_smbus_read_word_data(ina219_i2c_client,0x01);
 reverse_byte(&shunt_voltage);
 power = i2c_smbus_read_word_data(ina219_i2c_client,0x03);
 reverse_byte(&power);
 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  {
```

```
{
 printk("Driver Close\n");
 return 0;
}

static struct file_operations fops  = {
 .owner = THIS_MODULE,
 .open = ina219_driver_open,
 .release = ina219_driver_close,
 .unlocked_ioctl = ioctl_dev,
 .read = ina219_driver_read,
};

static int __init ina219Init(void)
{
 int ret =-1;

 int16_t id;
 printk("Driver Init\n");

 if ( alloc_chrdev_region(&ina219_device_number, 0, 1, DRIVER_NAME) < 0)
 {
  printk("Device Nr. could not be allocated!\n");
 }

 printk("Driver - Device Nr %d was registered\n", ina219_device_number);

        /* Create Device Class */
 if ((ina219_class = class_create(THIS_MODULE, DRIVER_CLASS)) == NULL)
 {
  printk("Device Class can not be created!\n");
  goto ClassError;
 }

        /* Create Device file */
 if (device_create(ina219_class, NULL, ina219_device_number, NULL, DRIVER_NAME) == NULL)
 {
  printk("Can not create device file!\n");
```

```
 {
  printk("Can not create device file!\n");
  goto FileError;
 }

        /* Initialize Device file */
 cdev_init(&ina219_device, &fops);

        /* register device to kernel */
 if (cdev_add(&ina219_device, ina219_device_number, 1) == -1)
 {
  printk("Registering of device to kernel failed!\n");
  goto KernelError;
 }

 ina_i2c_adapter = i2c_get_adapter(I2C_BUS_USED);

 if(ina_i2c_adapter != NULL)
 {
  ina219_i2c_client = i2c_new_client_device(ina_i2c_adapter, &ina219_i2c_board_info);

  if(ina219_i2c_client != NULL)
  {
   if(i2c_add_driver(&ina219_driver) != -1)
   {
    ret = 0;
   }
   else
    printk("Can't add driver...\n");
  }
  i2c_put_adapter(ina_i2c_adapter);
 }

 printk("INA219 Driver Init\n");

 id = i2c_smbus_read_word_data(ina219_i2c_client, 0x00);
 reverse_byte(&id);
 printk("Config Data: 0x%x\n",id);
```

```
    id = i2c_smbus_read_word_data(ina219_i2c_client, 0x00);
    reverse_byte(&id);
    printk("Config Data: 0x%x\n",id);
    i2c_smbus_write_word_data(ina219_i2c_client, 0x05,CALIBRATION_VALUE);

    return ret;

KernelError:
        device_destroy(ina219_class, ina219_device_number);
FileError:
        class_destroy(ina219_class);
ClassError:
        unregister_chrdev(ina219_device_number, DRIVER_NAME);
        return (-1);
}

static void __exit ina219Exit(void)
{
    printk("ina219_deviceDriver - Goodbye, Kernel!\n");
    i2c_unregister_device(ina219_i2c_client);
    i2c_del_driver(&ina219_driver);
    cdev_del(&ina219_device);
    device_destroy(ina219_class, ina219_device_number);
    class_destroy(ina219_class);
    unregister_chrdev_region(ina219_device_number, 1);
}

module_init(ina219Init);
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 inserter in kernel space with lsmod.

USER SPACE APPLICATION CODE & BUILD PROCESS:

```c
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/ioctl.h>

#include "config.h"

int file_desc;

int ioctl_bus_voltage(int file_desc, int16_t *msg)
{
 int ret_val;

 ret_val = ioctl(file_desc, IOCTL_BUS_VOLTAGE,msg);
 return ret_val;
}

int ioctl_shunt_voltage(int file_desc, int16_t *msg)
{
 int ret_val;

 ret_val = ioctl(file_desc, IOCTL_SHUNT_VOLTAGE,msg);
 return 0;
}

int ioctl_measured_current(int file_desc, int16_t *msg)
{
 int ret_val;

 ret_val = ioctl(file_desc, IOCTL_MEASURED_CURRENT,msg);
 return 0;
}

int ioctl_power(int file_desc, int16_t *msg)
{
 int ret_val;

 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 recv_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,&recv_msg);
59     bus_voltage = (float)recv_msg/1000;
60     printf("Bus Voltage:%f V |",bus_voltage);
61
62
63     ioctl_shunt_voltage(file_desc,&recv_msg);
64     shunt_voltage = (float)recv_msg;
65     printf("Shunt Voltage:%f mV |",shunt_voltage);
66
67     ioctl_power(file_desc,&recv_msg);
68     power = (float)recv_msg/1024.0;
69     printf("Power:%f mW |", power);
70
71     ioctl_measured_current(file_desc,&recv_msg);
72     measured_current = (float)recv_msg/1000.0;
73     printf("Current:%f mA\n",measured_current);
74
75    }
```

```
72     measured_current = (float)recv_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:

```
#ifndef CHAR_CONFIG_H
#define CHAR_CONFIG_H

#include <linux/ioctl.h>

#define MAGIC_NUM 156

//IOCTL interface prototypes

#define IOCTL_BUS_VOLTAGE _IOWR(MAGIC_NUM, 0, int16_t *)
#define IOCTL_SHUNT_VOLTAGE _IOWR(MAGIC_NUM, 1, int16_t *)
#define IOCTL_MEASURED_CURRENT _IOWR(MAGIC_NUM, 2, int16_t *)
#define IOCTL_POWER _IOWR(MAGIC_NUM, 3, int16_t *)


//Device file interface
#define DEVICE_FILE_NAME "/dev/ina219"

#endif
```

MAKE FILE:

```
obj-m += main.o

KDIR = /lib/modules/$(shell uname -r)/build
all:
    make -C $(KDIR)  M=$(shell pwd) modules

clean:
    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.