Device Driver for interfacing DHT12 (Digital Humidity and Temperature Sensor) with Raspberry Pi using I2C protocol

DONE BY:   
 Anjana Sharma(2018H1400143G),

S.Linga Abhishek(2018h1400140G)

# 1. **SUMMARY OF THE PROJECT**

The aim of this project is to write an I2C driver to interface Raspberry Pi with DHT12 Sensor. The following sections describe the specifications of DHT12, the I2C protocol implemented using the sensor and the format of the data received from the sensor. The subsequent sections describe the hardware design (schematic), kernel space driver and its build process and the user space application.

When the kernel space program is compiled and the kernel object file is inserted into the kernel using insmod, the driver is instantiated in the init function and it then waits for a DHT12 sensor to be detected. The detection process is explained in detail, in a later section. Once it has been detected, it creates a device file (device is registered) for the detected device in the “/dev” directory.  
This device file can then be used for reading the sensed values from the sensor by associating a “function operations” structure with the device file, which specifies the kernel read function to be executed upon a read request.

The user application makes use of the device file created using our driver to read values from the DHT12 sensor and display them after implementing CRC and formatting the data properly.

## **DHT12**

This section provides an introduction to the DHT12 digital humidity and temperature sensor. It is an upgraded version of DHT11, fully downward compatible, more precise and adds an I2C interface. DHT12 provides a calibrated digital output record of temperature and humidity. It has an application-specific digital temperature and humidity sensor module and semiconductor, which ensure high reliability and excellent long-term stability.

Two kinds of communication modes are possible with DHT12, namely, single wire communication mode (which is fully compatible with DHT11) and the I2C protocol communication. Standard bus interface makes it simple and quick for system integration. With super small size and low power consumption, it is suitable for a wide variety of applications. I2C Communication uses a standard communication sequence, wherein the user can directly communicate on the bus with no additional wiring. It is simple to use.

## **SCOPE OF APPLICATIONS**

Listed are some of the major areas of application where DHT12 may be used.

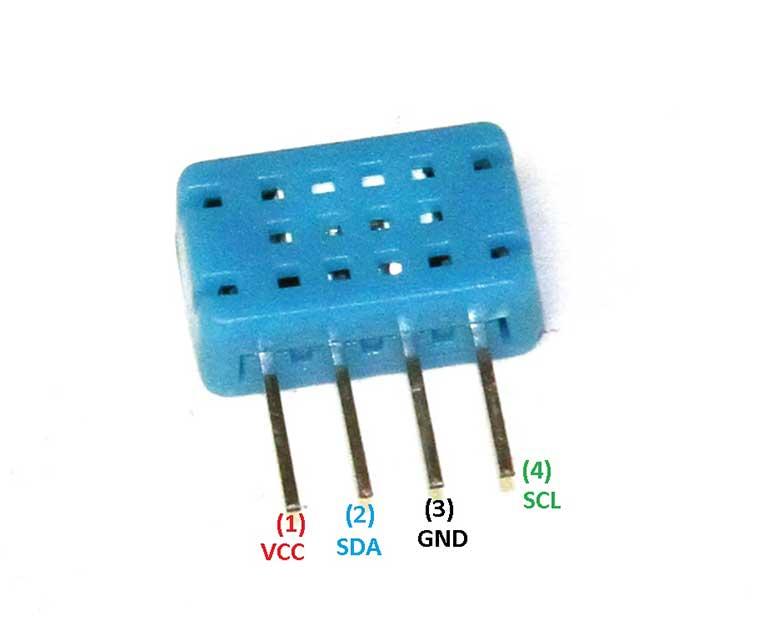
Heating, ventilation and air conditioning, Dehumidifier, test and inspection equipment,

consumer products, automotive, automation, data recorders, weather stations, home

appliances, control, Medical and other relative humidity control.

## **DHT12 HIGHLIGHTS**

Fully interchangeable, low cost, long term stability, relative humidity and temperature measurement, digital output, precise calibration, power consumption is very low and communication protocol can be freely chosen between the standard single-wire digital interface and the standard I2C Bus digital interface.



## **SPECIFICATION**

### 1. Power supply pins (VDD & GND）

The supply voltage range is between 2.7V - 5.5V.

### 2. Serial data (SDA）

SDA PIN is the data pin, used for reading and writing data whilst making use of the I2C protocol.

### 3.GND

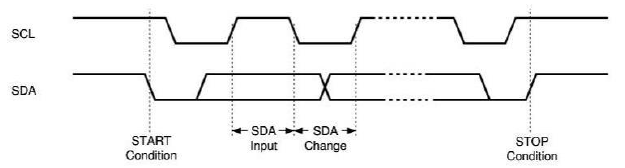
Represents the pin to be grounded.

### 4.Serial clock input (SCL)

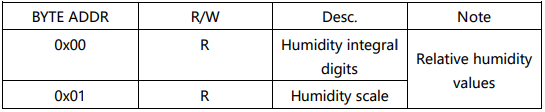
The SCL Pin is used to select the type of protocol to be used for communication. If SCL remains low it indicates that the user selects a single wire protocol for communication, otherwise I2C Communication.

## **DHT12 Sensor I2C Communication protocol**

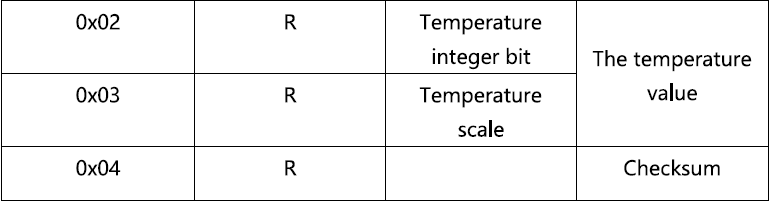
DHT12 communicates in full accordance with I2C Standard protocol and the sensor can be linked directly to I2C Bus. I2C device address in the I2C bus 0x5c. The Communication rate cannot be higher than 400KHZ. Communication must be in strict accordance with the following distribution specification, otherwise, the sensor does not work.



Data can be read from the following Byte addresses when reading from the sensor.



Humidity and temperature scale are nothing but the decimal part of the sensed values.



**Example: receiving 40 Data for:**

## 00111000 00001000 00011010 00000110

Humidity integral bits Humidity scale Temperature integer bits Temperature scale

## 01100000

Check sum

00111000+00001000+00011010+00000110=01100000 (Check sum)

Data from 0x04 is for Cyclic Redundancy Check. (CRC).

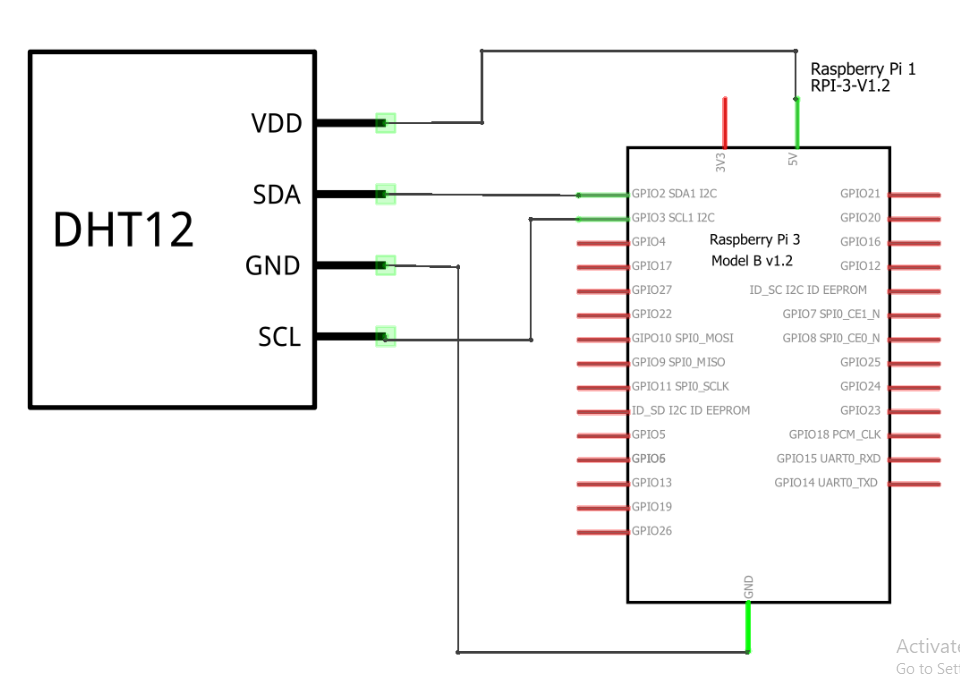
Humidity: 00111000 (Binary) => 56 (Decimal)

00001000(Binary) => 8 (Decimal)

Therefore, Humidity is 56.8%RH.

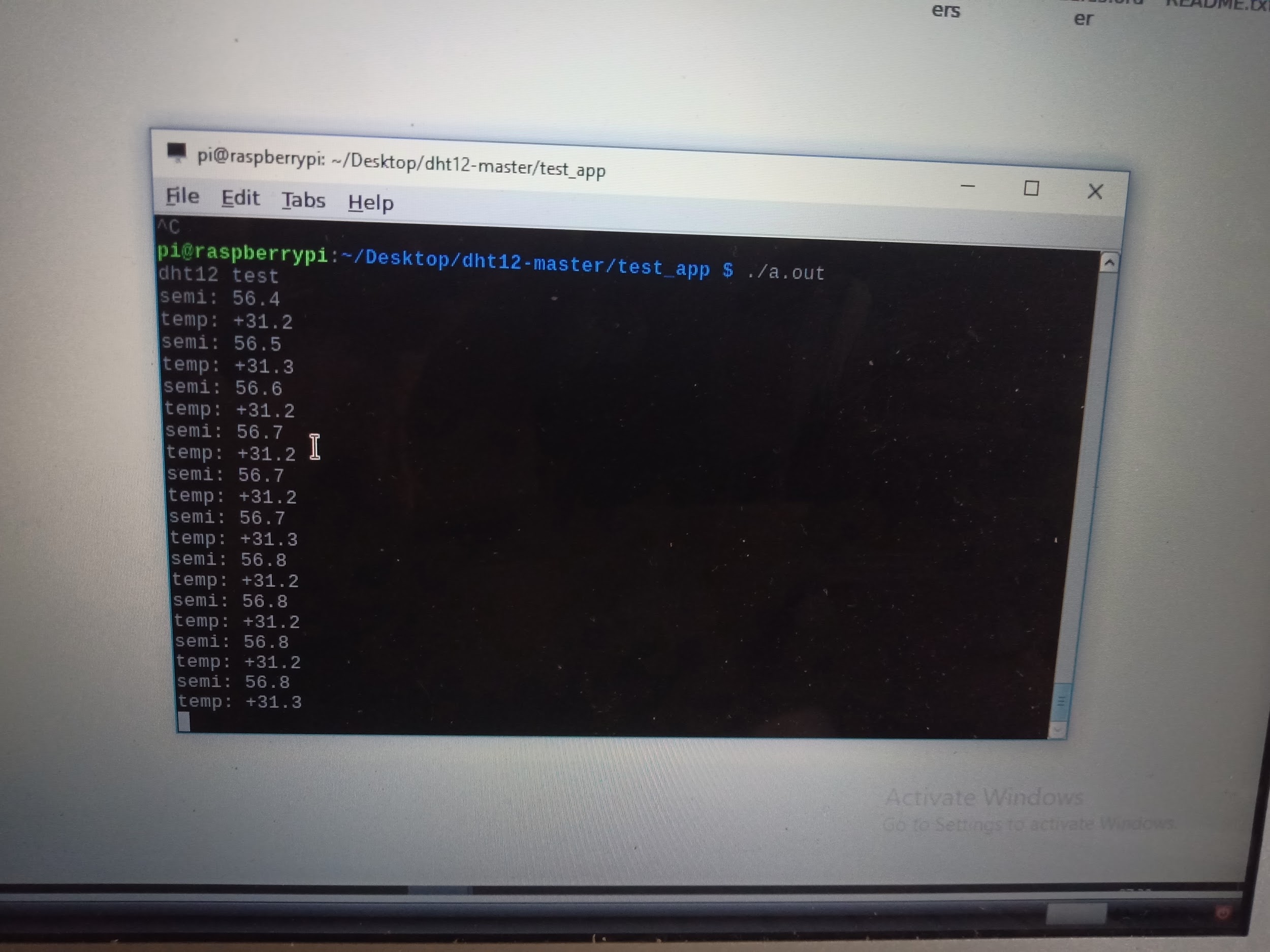
# **3. HARDWARE DESIGN (SCHEMATIC)**

This section describes the hardware schematic for connecting a DHT12 sensor to Raspberry Pi and also, photograph of our implementation is attached.

The schematic is depicted below. 

Vcc is connected to 5V power supply, Ground pin is grounded and SDA & SCL Pins are connected to GPIO SDA1 I2C and GPIO SCL1 I2C pins respectively.

## 



# **4.** **KERNEL SPACE DRIVER & BUILD PROCESS**

### **DRIVER**

The I2C device driver for interfacing DHT12 was written in C Programming language. The various functions used for achieving this and the flow of control is described in this section. Firstly, in the init function the driver is instantiated using the [**i2c\_add\_driver**](https://elixir.bootlin.com/linux/latest/ident/i2c_add_driver)(driver)macro.

### **ID TABLE**

This driver has the device name and the id table. The ID table lists out all the devices that the current driver can handle. [**MODULE\_DEVICE\_TABLE**](https://elixir.bootlin.com/linux/latest/ident/MODULE_DEVICE_TABLE)(type, name)is used to associated the i2c driver with the list of devices to be handled.

### **PROBE**

Once the driver has been created, it then waits for a client\_object to be instantiated. It matches its device name with the device name of the instantiated client and if it matches, it calls the “probe” function. The probe function then proceeds to create the device using the function [**misc\_register**](https://elixir.bootlin.com/linux/latest/ident/misc_register)(struct [**miscdevice**](https://elixir.bootlin.com/linux/latest/ident/miscdevice) \*[**misc**](https://elixir.bootlin.com/linux/latest/ident/misc)).

### **FILE OPERATIONS**

F\_OPS function associated with the created device specifies the kernel read function, which reads 5 bytes from the 0x00 address of the DHT12 device file using the [**i2c\_master\_recv**](https://elixir.bootlin.com/linux/latest/ident/i2c_master_recv)(const struct [**i2c\_client**](https://elixir.bootlin.com/linux/latest/ident/i2c_client) \*[**client**](https://elixir.bootlin.com/linux/latest/ident/client),char \*buf, int count)function and copies to user application using the copy\_to \_user() function.

### **CLIENT OBJECT INSTANTIATION**

Here, client object is instantiated slightly differently in our project. Since only one device with known i2c address needs to be added, we can instantiate the client object by writing into the “/sys/bus/i2c/devices/i2c-1/new\_device” the “device name” (DHT12) and its I2C address (0x5c). This can be done using echo command.

EXIT funtion then does all the cleanup and deregister operations.

# 5. USER SPACE APPLICATION

The user space application can be executed to perform the read function from the created device file. It also performs CRC check once it receives the 5 bytes from DHT12 Sensor.

It opens the “/dev/DHT12” device file to perform the read operation.