Data Acquisition System – IoT

(Eight Analog and Eight digital interfacing)

Prototype

Start Date:

End Date: 29 November 2017

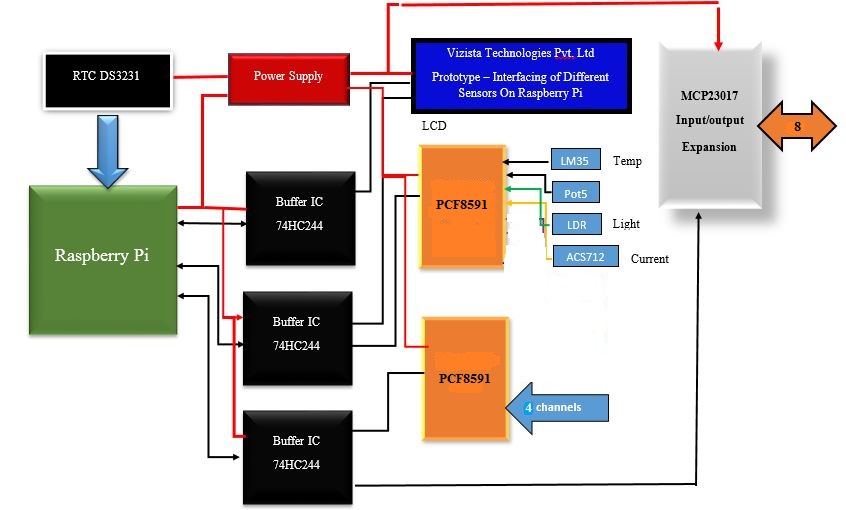
**Index**

|  |  |  |
| --- | --- | --- |
| Sr.No | Content | Page No |
| 1 | Introduction | 3 |
| 2 | Block diagram of system | 3 |
| 3 | Components Used | 3 |
| 4 | Settings for I2C/ SPI communication | 4 |
| 5 | Setting for RTC | 4 |
| 6 | Interfacing of LCD with raspberry pi | 5 |
| 7 | Interfacing of PCF8591 with raspberry pi | 5 |
| 8 | Calibration of different Sensors | 6 |
| 9 | Interfacing of MCP-23017 | 8 |
| 10 | Sqlite3 settings | 11 |
| 11 | Bulk update | 12 |

**Introduction**

The proposed project consist of eight Analog and eight digital interfacing with Raspberry Pi Model 3. The communication with ADC – PCF 8591 and MCP27031 ICs using I2C (inter interfacing Communication). The eight Analog sensors are temperature sensor – LM35, LDR, six potentiometer, Humidity Sensor. The respective values will get loaded in database and later they will get automatically updated on things speak.

**Block diagram**

****

**Figure 2.1: Block Diagram of Data Acquisition System**

Figure 2.1 shows the block Diagram of Prototype.

**Component Used**

* Buffer IC- 74HC244
* ADC Analog to Digital Converter – PCF8591
* Raspberry Pi Model 3
* Expansion IC- MCP23017 –Input /Output devices
* RTC Module – RTC-DS3231
* I2C Level converter
* DH11 – Temperature and Humidity sensor
* LM35
* Thermistor
* LDR
* POT (6)

**Settings [1]**

*pi@raspberrypi:* *>ls /dev/\*i2c\**

*PI will respond to it as /dev/i2c-1*

*pi@raspberrypi:~/$ i2cdetect -y 1*

*0 1 2 3 4 5 6 7 8 9 a b c d e f*

*00: -- -- -- -- -- -- -- -- -- -- -- -- --*

*10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --*

*20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --*

*30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --*

*40: -- -- -- -- -- -- -- -- -- 49 -- 4b -- -- -- --*

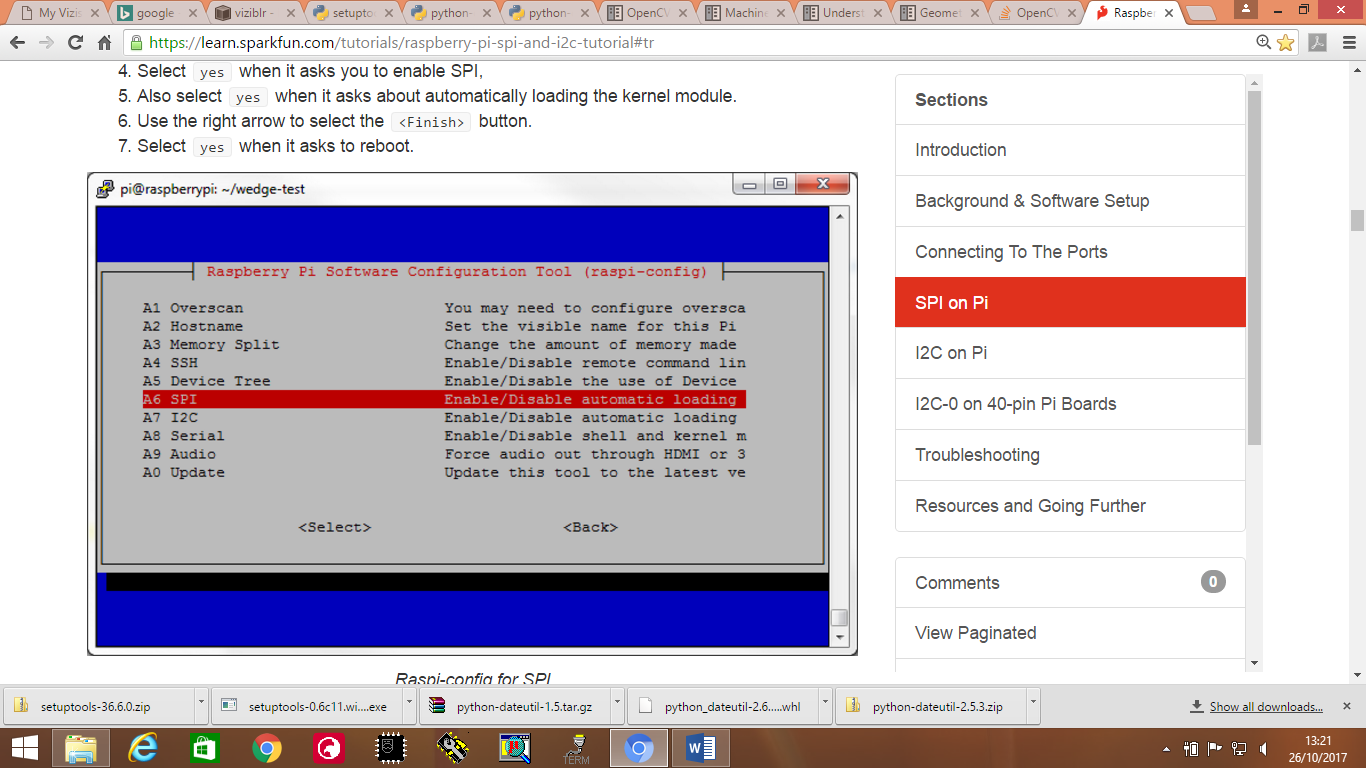
*50: 50 --- -- -- -- -- -- -- -- -- -- -- -- -- -- --*

*60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --*

*70: -- -- -- -- -- -- -- --*

Read and write the data.

Using the above syntax.



**Figure 3.1: Raspberry Pi Software Configuration (SPI**

**Setting of hardware clock- RTC Module [9]**

To set the time and date of your raspberry pi:

*sudo date –s “DD MM YY hh : mm : ss*

*sudo hwclock –w*

*This will write into the hardware of the raspberry pi.*

*sudo hwclock –r*

*This will read the current date and time.*

**LCD interfacing with raspberry pi 3**

**#include lcd.h:** Header files contains call the syntax to write on LCD.

Initialize the LCD first:

*Syntax : lcdInit (const int rows, const int cols, const int bits, const int rs, const int strb, const int D0, const int D1, const int D2, const int D3, const int D4, const int D5, const int D6, const int D7)*

*Example: fd= lcdInit(2,16,8,LCD\_RS, LCD\_EN, D0,D1,D2,D3,D4,D5,D6,D7 )*

*lcdPrintf(fd, “Hellow World”)*

*lcdPosition(fd, row, col)*

**Interfacing of PCF8591 with raspberry pi: [8]**

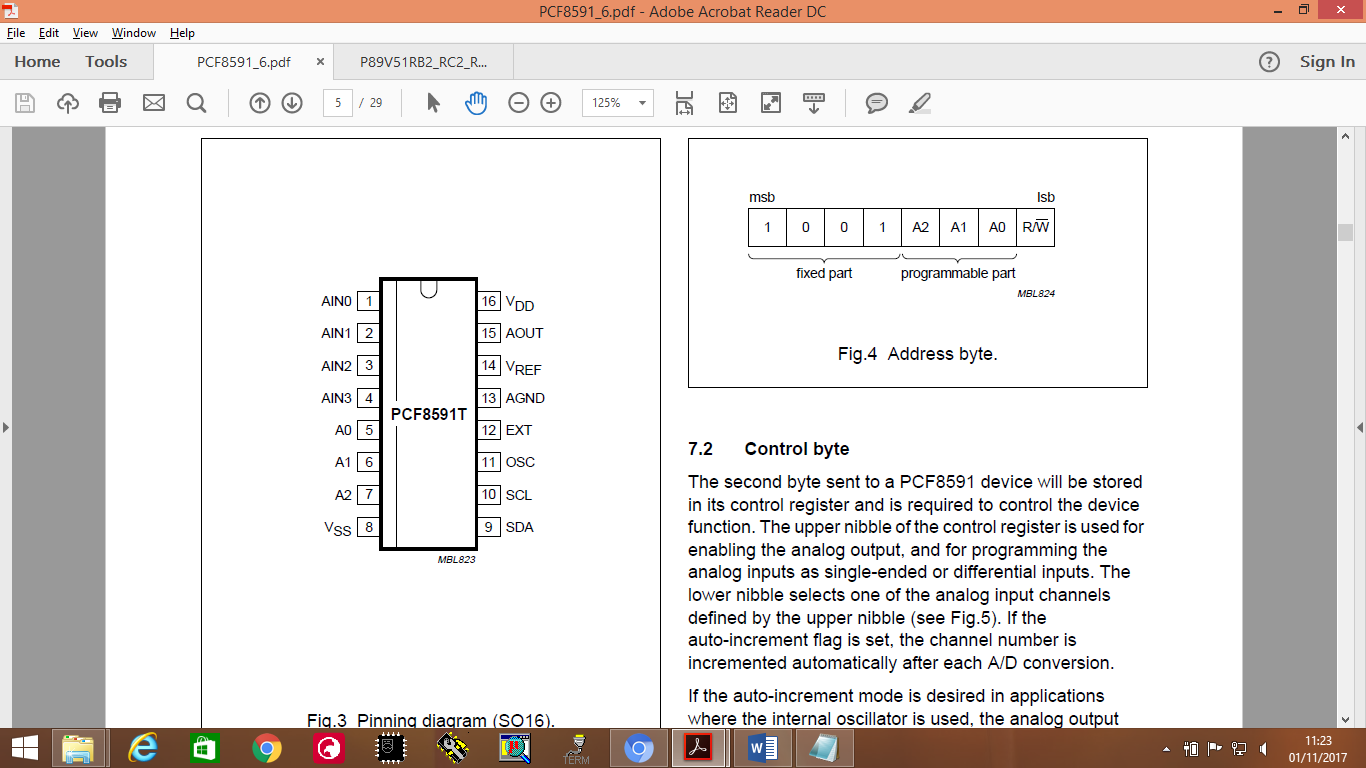
*Header file: #include <pcf8591.h>*

*Initialization: #include ( int pinBase, int i2cAddress)*

The pinBase can be any number you like above 64 and the i2cAddress is the address of the device in the I2C bus – 0x48 is the default but they can change if you have multiple devices. Use the **i2cdetect** command (**gpio i2cd**) to probe your I2C bus to work out the right address to use.

* **Auto incremental channel selection**
* **The analog voltage is varied between Vss and Vdd**
* **8 bit successive approximation ADC**

**Address register:**



**Figure 7.1: Address resistor of PCF8591 useful for register programming**

**Example 7.1: PCF8591 interfacing with raspberry pi 3 [10]**

*#include <wiringPi.h>*

*#include <pcf8591.h>*

*#include <stdio.h>*

*#define Address 0x48*

*#define BASE 64*

*#define A0 BASE+0*

*#define A1 BASE+1*

*#define A2 BASE+2*

*#define A3 BASE+3*

*int main(void)*

*{*

*int value;*

*wiringPiSetup();*

*pcf8591Setup(BASE, Address);*

*while(1)*

*{*

*value = analogRead(A0);*

*printf("Analoge: %dmv\n",value\*3300/255);*

*delay(1000);*

*}*

*}*

We might work with 2 I2C ICs at same time just change the base address as shown in example 7.2

Example 7.2: #define BASE\_1 64

#define BASE\_2 74

Please refer reference section [10] for more details regarding the PCF8591 interfacing with raspberry pi.

Syntax for DAC – Digital to analog Converter:

*analogWrite(A0,value);*

*printf("AOUT: %d\n",value++);*

**Calibration of different sensors**

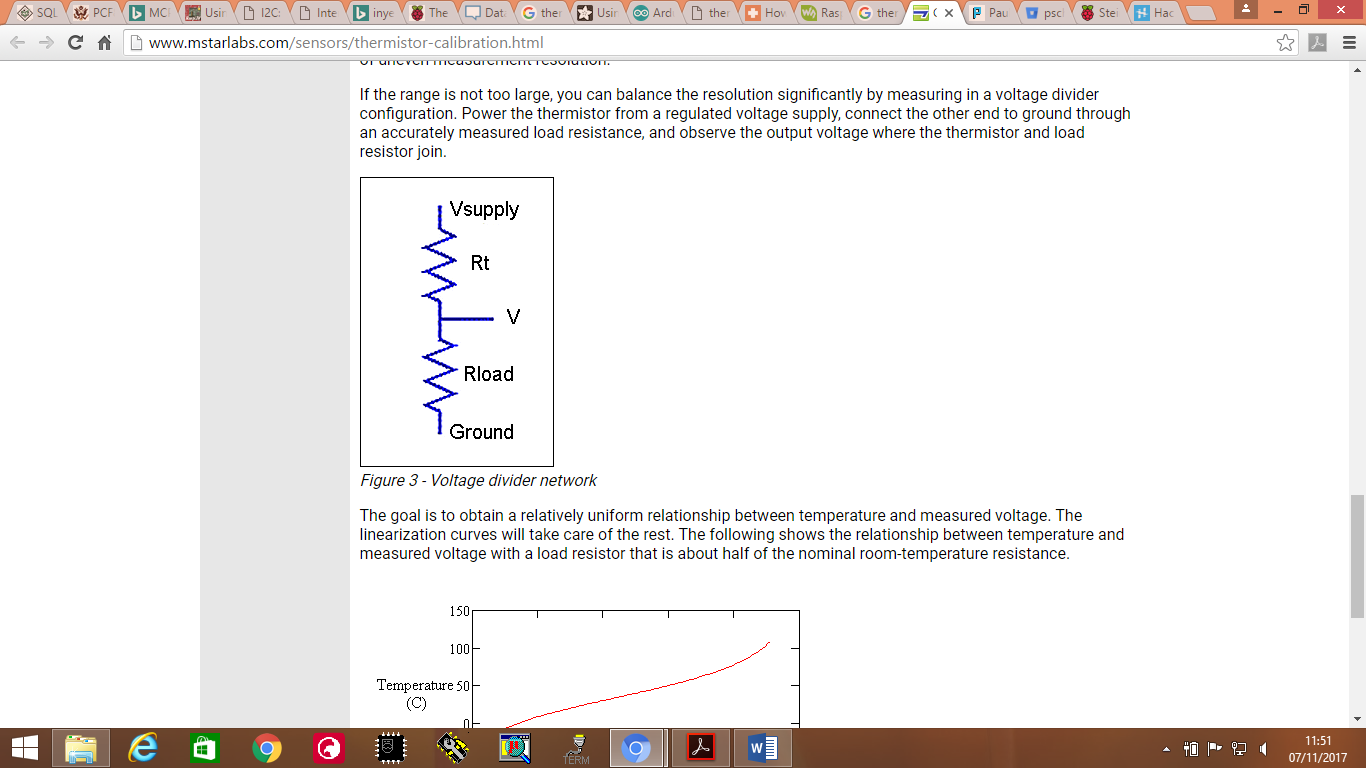
* **LM35 :** Convert the ADC raw value to voltage using the below formula

raw\_value = analogRead(A0)

Voltage= (raw\_value \* voltage\_value)/ (2^n -1)

Temp= voltage /10;

**Thermistor [3]:** Thermistors have the advantage of a very high sensitivity to temperature changes, but the disadvantage of an aggressively nonlinear characteristic. Here is a characteristic curve showing the resistance of a typical negative temperature coefficient thermistor device over a temperature range from 0 to 100 degrees C.



* **Vout = analog\_value\* (voltage / (2^n -1))**
* **Using voltage divider rule:**
* **Rt = (1000 \* Vout)/ (3.3- vout)**
* **X= ln(Rt);**
* **T= 1/( ka + kb\*x + kc\*(x^3))**

**ka = 1.283\*10-3**

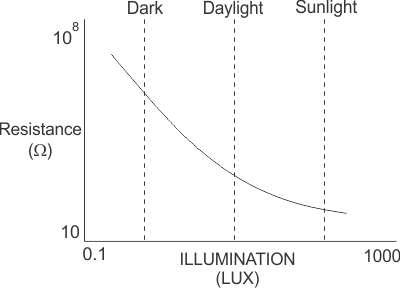
**kb = 2.362 \*10^(-4)**

**kc = 9.285\* 10^(-8)**

If in case you are working with math library, one should add –lm in command line. This will remove the error while building the code.

**LDR (Light Dependent Register) [11]:** As the light intensity increase the resistance decreases.

When light strikes on the surface of LDR. If the light energy is greater than the electron band gap the electrons get excited to conduction level there by increasing the current and decreasing the resistance. Figure 8.1 is significant enough for dis proportionality.

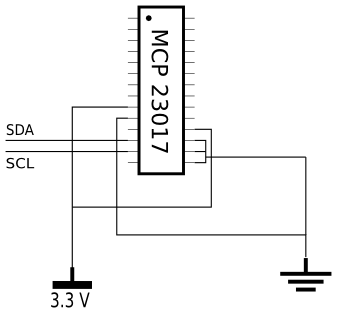
  
**Figure 8.1: intensity and resistance graph for LDR.**

Calibration is not needed as such until you want to display the LUX value. We will receive the values between 0 -255. Since the PCF 8591 is 8 bit resolution. For more accuracy increase the resolution.

**Interfacing of MPC23017 (Expansion IC)[8] [5]**

**Setup**

We'll start by connecting the chip. **Connect pins 9 and 18 to 3v3, and pin 10 to ground. For now, connect pins 15 through 17 to ground as well.** (They configure the I2C address of the chip, which I'll talk about later.) Now to be able to talk to the chip, **connect pin 12 to the Pi's pin 5 (SCL) and pin 13 to the Pi's pin 3 (SDA).** Here's my first ever attempt at a circuit diagram showing it, although I haven't included the Pi:

[](https://camo.githubusercontent.com/c80be3d0c9e3146cd09e15b7ddf07dbb8b7d40b8/687474703a2f2f692e696d6775722e636f6d2f4575524e722e706e67)

**Figure 9.1: Pin configuration of MCP23017**

At this point, run sudo i2cdetect -y 0 and you should get the following output:

0 1 2 3 4 5 6 7 8 9 a b c d e f

00: -- -- -- -- -- -- -- -- -- -- -- -- --

10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

70: -- -- -- -- -- -- -- --

This says there's an I2C device with address 0x20 that you can talk to.

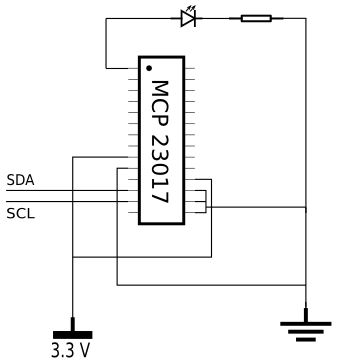
To actually talk to it, use the programs i2cget and i2cdump for reading, and i2cset for writing. The chip has **21 registers at 22 addresses (one register has two addresses), and reading and writing these allows you to read and write to the 16 GPIO pins.** (The register addresses have nothing to do with the device address.)

### Writing

Let's start by activating an LED on pin 1. The 16 GPIO pins are divided into banks 'A' and 'B' of eight pins each, and somewhat counterintuitively, **pin 1 is GPB0 (pin 0 in bank B). We need to set it up for output; the register we need is IODIRB at address 0x01, which controls the direction of each pin in bank B.** (They can all be set individually, but they're all done from the same register.)

To get the current value of this register, **run i2cget -y 0 0x20 0x01**. Here 0 is the bus again, 0x20 is the device address and 0x01 is the register address. It will probably print 0xff. This should be read as eight bits rather than as a single byte; the least significant bit corresponds to GPB0, and so on. A 1 bit indicates that pin is configured for input, and a 0 indicates output, so we need to turn off bit 1. Run i2cset -y 0 0x20 0x01 0xfe, which means "assign value 0xfe to register 0x01 of device 0x20 on bus 0". Of course, you could use 0x00 or something else instead of 0xfe, as long as the final bit is 0.

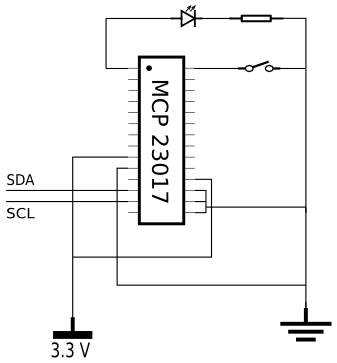
Now to turn it on. (Connect the LED first, if you haven't already. Its + terminal connects to pin 1 on the MCP, its - terminal connects to ground via a suitable resistor.) The register we use for this is GPIOB at address 0x13. Write a 1 to bit 1 of this register and the LED should turn on: i2cset -y 0 0x20 0x13 0x01. Write a 0 again to turn it off: i2cset -y 0 0x20 0x13 0x00.

[](https://camo.githubusercontent.com/1367f2714eb7f543e312cec368b7e95f9aaa9490/687474703a2f2f692e696d6775722e636f6d2f76737a56422e706e67)

**Figure 9.2: Writing on MCP-23017 and glowing the LED**

### Reading

Now we'll read the state of a button, which we'll put on pin 28. This is GPA7, i.e. pin 7 on bank A. So connect a button with one terminal connected to pin 28 and the other connected to ground.

[](https://camo.githubusercontent.com/9e6241026ff3f3abbf210810019263e9c9e2866a/687474703a2f2f692e696d6775722e636f6d2f615377674c2e706e67)

**Figure 9.3: Reading the MCP 23017**

The direction register for bank A is IODIRA at address 0x00. Like IODIRB, it's probably already set up as 0xff, but if it doesn't already have the most significant bit set (which is true iff its value is less than 0x80), you'll need to write to it.

Now you can read the value of the button as the most significant bit from GPIOA, address 0x12. Except that there's no power being supplied to that pin or to the button, so you'll just read 0x00.

You can fix this by putting a [pull-up resistor](http://en.wikipedia.org/wiki/Pull-up_resistor) into your circuit; a 10 kΩ resistor between 3v3 and pin 28 will do the trick. But the MCP has pull-up resistors built-in, they just aren't enabled by default. To enable it for GPA7, we use register GPPUA at address 0x0c, and turn on the MSB: i2cset -y 0 0x20 0x0c 0x80. (Only two of the Pi's GPIO pins have pull-up resistors, so even if you don't need the extra GPIOs, the MCP might make your circuit simpler.)

(Update: it appears that in fact all of the Pi's GPIO pins have pull-up resistors, and most have pull-down resistors as well. The gpio program from [WiringPi](https://projects.drogon.net/raspberry-pi/wiringpi/) can enable and disable these.)

After this, i2cget -y 0 0x20 0x12 should return 0x00 if the putton is pressed, and 0x80 if the button is released. I found that for a short time after releasing the button, I would read 0xc0, indicating that GPA6 was also returning a 1 bit. I assume this is just due to electrical interference or something; pin 27 isn't connected to either power or ground, so its value is unreliable. (When I enabled its pull-up resistor, or connected it to ground, it read the expected value every time.)

**Programming**

*void digitalIO()*

*{*

*fd\_1 = wiringPiI2CSetup(0x20);*

*sw= wiringPiI2CReadReg8(fd\_1, 0x13);*

*printf("switch number = %d", sw)*

*swtich(sw)*

*{*

*case 0x07:*

*printf("swtich 1 pressed\n");*

*break;*

*case 0x0b:*

*printf("swtich 2 pressed\n");*

*break;*

*case 0x0d:*

*printf("swtich 3 pressed\n");*

*break;*

*case 0x0e:*

*printf("swtich 4 pressed\n");*

*break;*

*default :*

*printf("swtich 1 pressed\n");*

*break;*

*}*

*}*

**Sqlite3 Settings:**

*$ sudo apt-get install sqlite3*

*This will install the sqlite lib*

*Now check wheather it has successfully installed*

*$ sqlite mydatabase.db*

*It will response as SQLite Version 3.8.7.1 date time*

**Example for creating the database in sqlite3**

*#include <stdio.h>*

*#include <sqlite3.h>*

*int main(int argc, char\* argv[]) {*

*sqlite3 \*db;*

*char \*zErrMsg = 0;*

*int rc;*

*rc = sqlite3\_open("test.db", &db);*

*if( rc ) {*

*fprintf(stderr, "Can't open database: %s\n", sqlite3\_errmsg(db));*

*return(0);*

*} else {*

*fprintf(stderr, "Opened database successfully\n");*

*}*

*sqlite3\_close(db);*

*}*

*Compile using the following Set build commands*

*Gcc-Wall –c “%f”-1 wriningPi –l curl –l wiringPiDev –lsqilte3 -1m –std = c99 –ljson*

*static int callback(void \*NotUsed, int argc, char \*\*argv, char \*\*azColName) {*

*int i;*

*for(i = 0; i<argc; i++) {*

*printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");*

*}*

*printf("\n");*

*return 0;*

*}*

*/\*Create SQL statement \*/*

*sql = "CREATE TABLE SENSORS(" \*

*"ID INTEGER PRIMARY KEY AUTOINCREMENT," \*

*"date\_time TEXT NOT NULL," \*

*"temperature float NOT NULL," \*

*"voltage float NOT NULL," \*

*"speed float NOT NULL," \*

*"current float NOT NULL," \*

*"ldr float NOT NULL," \*

*"humidity float NOT NULL,"*

*"AQI INT NOT NULL);";*

*/\* Execute SQL statement \*/*

*rc = sqlite3\_exec(db, sql, callback, 0, &zErrMsg);*

*if( rc != SQLITE\_OK ){*

*fprintf(stderr, "SQL error: %s\n", zErrMsg);*

*sqlite3\_free(zErrMsg);*

*} else {*

*fprintf(stdout, "Table created successfully\n");*

*}*

*sqlite3\_close(db);*

*return 0;*

**Splitting the date and time [6]**

*#include <stdio.h>*

*#include <time.h>*

*int main () {*

*time\_t rawtime;*

*struct tm \*info;*

*char buffer[80];*

*time( &rawtime );*

*info = localtime( &rawtime );*

*strftime(buffer,80,"%x - %I:%M%p", info);*

*printf("Formatted date & time : |%s|\n", buffer );*

*return(0);*

*}*

***Example for the database creation***

*void database()*

*{*

*time(&rawtime);*

*info= localtime(&rawtime);*

*if (rc)*

*{*

*fprintf(stderr,"Can't open database: %s\n",sqlite3\_errmsg(db));*

*}*

*else*

*{*

*fprintf(stdout,"opened successfully");*

*}*

*strftime(buf\_hr,10,"%X", info)*

*strftime(buf\_YY,10,"%Y", info)*

*strftime(buf\_MM,10,"%m", info)*

*strftime(buf\_DD,10,"%d", info)*

*strcat(buf\_YY,"-");*

*strcat(buf\_YY,buf\_MM);*

*strcat(buf\_YY,"-");*

*strcat(buf\_YY,buf\_DD);*

*strcpy (collect, "INSERT INTO SENSORS(date\_time,temperature\_lm35, temp\_termistor, LDR, Current, Vtg,AQI)"\*

*"VALUES(" );*

*sprintf(data\_time,"'%sT%sZ'", buf\_YY,buf\_hr)*

*sprintf(data\_tem\_lm35,"%0.2f", temp\_lm35)*

*sprintf(data\_temp\_termistor,"%0.2f", temp\_termistor)*

*sprintf(data\_LDR,"%d", ldr)*

*sprintf(data\_current,"%0.2f", current)*

*sprintf(data\_voltage,"%0.2f", voltage)*

*sprintf(data\_AQI,"%d", AQI)*

*strcat(collect,data\_time)*

*strcat(collect,data\_tem\_lm35)*

*strcat(collect,data\_temp\_termistor)*

*strcat(collect,data\_LDR)*

*strcat(collect,data\_current)*

*strcat(collect,data\_voltage)*

*strcat(collect,data\_AQI)*

*sql= collect;*

*rc= sqlite3\_exec(db, sql, callback,0,&zErrMsg)*

*if(rc != SQLITE\_OK){*

*fprintf("sqlerror= %s\n", zErrMsg);*

*sqlite3\_free(zErrMsg);*

*}*

*sqlite3\_close(db);*

*The splitting of the time and date is according to the Json format specified by the thingspeak library*

*}*

**Bulk update using JSON format [15]**

**JSON**

| **Title** | **HTTP POST Request to Bulk-Update a Channel Feed Using a JSON Object** |
| --- | --- |
| **URL** | https://api.thingspeak.com/channels/CHANNEL\_ID/bulk\_update.json |
| **HTTP Method** | POST |
| **HTTP Header** | Content-Type: application/json |
| **Data Parameters** | {"write\_api\_key":"WRITE\_API\_KEY","updates":[{"created\_at":"DATETIME\_STAMP","field1":"FIELD1\_VALUE","field2":"FIELD2\_VALUE","field3":"FIELD3\_VALUE","field4":"FIELD4\_VALUE","field5":"FIELD5\_VALUE","field6":"FIELD6\_VALUE","field7":"FIELD7\_VALUE","field8":"FIELD8\_VALUE"},{"created\_at":"DATETIME\_STAMP","field1":"FIELD1\_VALUE","field2": "FIELD2\_VALUE","field3":"FIELD3\_VALUE","field4":"FIELD4\_VALUE","field5":"FIELD5\_VALUE","field6":"FIELD6\_VALUE", "field7":"FIELD7\_VALUE","field8":"FIELD8\_VALUE"},{"created\_at":"DATETIME\_STAMP","field1":"FIELD1\_VALUE","field2":"FIELD2\_VALUE","field3":"FIELD3\_VALUE","field4":"FIELD4\_VALUE","field5":"FIELD5\_VALUE","field6":"FIELD6\_VALUE","field7":"FIELD7\_VALUE","field8":"FIELD8\_VALUE"}]} |
| **Success Response** | 202: Accepted |
| **Error Codes** | 400: Bad request 401: Invalid API key or channel mismatch 413: Request too large 429: Too many requests |
| **Sample HTTP Call** | POST /channels/12/bulk\_update.json HTTP/1.1 Host: api.thingspeak.com Content-Type: application/json  {  "write\_api\_key":"XXXXXXXXXXXXXXXXX",  "updates":[{"created\_at":"2017-01-30 10:26:21 -0500","field1":1500,"field2":150,"field3":100,"field4":100,"field5":100,"field6":100,"field7":100,"field8":100},  {"created\_at":"2017-01-30 10:26:2 -0500","field1":100,"field2":150,"field3":100,"field4":100,"field5":100,"field6":100,"field7":100,"field8":100},  {"created\_at":"2017-01-30 10:26:2 -0500","field1":100,"field2":150,"field3":100,"field4":100,"field5":100,"field6":100,"field7":100,"field8":100},  {"created\_at":"2017-01-30 10:26:2 -0500","field1":100,"field2":150,"field3":100,"field4":100,"field5":100,"field6":100,"field7":100,"field8":100}]  } |
| **Notes** | Replace CHANNEL\_ID with your channel ID and WRITE\_API\_KEY with your channel write API key. The DATETIME\_STAMP must follow ISO 8601, EPOCH, or MYSQL formats. |

**Example to post the data using Postman**

{

"write\_api\_key":"LCY7MZHUVC20AZ62",

"updates":[{"created\_at":"2017-11-28T10:05:01Z","field1":27,"field2":3.75,"field3":200,"field4":4,"field5":60,"field6":60,"field7":355},

{"created\_at":"2017-11-21T10:10:01Z","field1":33,"field2":5,"field3":200,"field4":13,"field5":145,"field6":70,"field7":366},

{"created\_at":"2017-11-21T10:15:01Z","field1":27,"field2":0,"field3":255,"field4":12.4,"field5":20,"field6":30,"field7":455},

{"created\_at":"2017-11-21T10:20:20Z","field1":31,"field2":2.4,"field3":8,"field4":8.23,"field5":34,"field6":47,"field7":95}]

}

**{**

**"write\_api\_key": "LCY7MZHUVC20AZ62",**

**"updates": [{**

**"created\_at": "2017-11-28T10:05:01Z",**

**"field1": 27,**

**"field2": 3.75,**

**"field3": 200,**

**"field4": 4,**

**"field5": 60,**

**"field6": 60,**

**"field7": 355**

**},**

**{**

**"created\_at": "2017-11-21T10:10:01Z",**

**"field1": 33,**

**"field2": 5,**

**"field3": 200,**

**"field4": 13,**

**"field5": 145,**

**"field6": 70,**

**"field7": 366**

**},**

**{**

**"created\_at": "2017-11-21T10:15:01Z",**

**"field1": 27,**

**"field2": 0,**

**"field3": 255,**

**"field4": 12.4,**

**"field5": 20,**

**"field6": 30,**

**"field7": 455**

**},**

**{**

**"created\_at": "2017-11-21T10:20:20Z",**

**"field1": 31,**

**"field2": 2.4,**

**"field3": 8,**

**"field4": 8.23,**

**"field5": 34,**

**"field6": 47,**

**"field7": 95**

**}**

**]**

**}**

**Response**

**{**

**"success": true**

**}**

**For bulk update using Curl for TCP/IP [16]**

**Example for Curl and JSON for bulk update:**

**D:\Pranali\bulk\_update.txt**

**Appendix I**

**Reference**

1] SPI and I2C protocol:[https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial#tr](https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial" \l "tr)

2] Tutorial series for interfacing pcf8951 with raspberry pi, 1 November 2017

[https://www.waveshare.com/wiki/Raspberry\_Pi\_T utorial\_Series:\_PCF8591\_AD/DA](https://www.waveshare.com/wiki/Raspberry_Pi_Tutorial_Series:_PCF8591_AD/DA)

3] Thermistor calibration formulas: <http://www.mstarlabs.com/sensors/thermistor-calibration.html> , 06 November 2017

4] Integration of dh11 with raspberry pi: <http://www.rpiblog.com/2012/11/interfacing-temperature-and-humidity.html> , 14 November 2017

5] Interfacing of MCP23017 with raspberry pi: <https://gist.github.com/ChickenProp/3183960> , 13 November 2017

6] How to retrieve the time from time.h: <http://www.tutorialspoint.com/c_standard_library/c_function_strftime.htm> , 17 November 2017

7] Select command from sqlite3: <http://beginner-sql-tutorial.com/sql-select-top.htm> , 28 Nov. 17

8] Wiring pi reference for I2C and SPI communication of different ICs: <http://wiringpi.com/the-gpio-utility/i2c-mcp23008-and-mcp23017-extensions/> (accessed at 23 Nov. 17)

9] Setting the hardware clock DS3231: <https://www.raspberrypi-spy.co.uk/2015/05/adding-a-ds3231-real-time-clock-to-the-raspberry-pi/> (accessed on 23 Nov. 17)

10] Wiki share for interfacing of PCF8591 with raspberry pi using different library <https://www.waveshare.com/wiki/Raspberry_Pi_Tutorial_Series:_PCF8591_AD/DA>

(Accessed on 22 Nov. 17)

11] Working principle behind the LDR : <https://www.electrical4u.com/light-dependent-resistor-ldr-working-principle-of-ldr/> (accessed on 23 Jan. 18)

12] Setting of sqlite3: <http://www.raspberrywebserver.com/sql-databases/> , (accessed on 23 Jan. 18)

13] Tutorial for sqlite3: <https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm> , (accessed on 23 Jan. 18)

14] Examples of sqlite3: <http://zetcode.com/db/sqlitec/> , (accessed on 23 Jan. 18)

15] Bulk update on things speak site: <https://in.mathworks.com/help/thingspeak/bulk-update-a-channel-feed-1.html?requestedDomain=true> (accessed on 27 Nov. 17)

16] curltest for bulk update using post method: <https://gist.github.com/leprechau/e6b8fef41a153218e1f4> , (accessed on 27 Nov. 17)