TECH153 Greenhouse Controller

Version LabXX
2020-08-03 12:47:00 AM

Makefile:

```
GHC:GHC.o ghcontrol.o pisensehat.o
	gcc -g -o GHC GHC.o ghcontrol.o pisensehat.o -lwiringPi

GHC.o:GHC.c ghcontrol.h pisensehat.h
	gcc -g -c GHC.c

ghcontrol.o:ghcontrol.c ghcontrol.h pisensehat.h
	gcc -g -c ghcontrol.c

pisensehat.o:pisensehat.c pisensehat.h
	gcc -g -c pisensehat.c

clean:
	rm *.o
	touch *
```

Table of Contents

File Index

File List

Here is a list of all files with brief descriptions:

File Documentation

GHC.c File Reference

```
#include "ghcontrol.h"
```

Functions

- int **main** (void)

Detailed Description

main program structure and function call order for Gh controller

---

Function Documentation

int main (void)

```
 7 {
 8     struct setpoints sets = {0};
 9     struct readings creadings ={0};
10     struct controls ctrl = {0};
11     time_t now;
12     char test;
13     int logged;
14     alarmlimit_s alimits = { 0 };
15     alarm_s warn[NALARMS] = { 0 };
16 
17 
18     time(NULL);
19 
20     sets = GhSetTargets(sets);
21     alimits = GhSetAlarmsLimits();
22 
23     GhControllerInit();
24 
```

```
25      while(1)
26      {
27          now = time(NULL);
28
29          creadings = GhGetReadings();
30          logged = GhLogData("ghdata.txt",creadings);
31          ctrl = GhSetControls(sets,creadings);
32          GhDisplayAlarms(warn);
33          GhDisplayAll(creadings,sets);
34          GhDisplayReadings(creadings);
35          GhDisplayTargets(sets);
36          GhDisplayControls(ctrl);
37          GhDelay(GHUPDATE);
38
39      }
40
41      test =  fgetc(stdin);
42
43      if(test =='_')
44      {
45          return EXIT_SUCCESS;
46      }
47
48  }
```

---

ghcontrol.c File Reference

`#include "ghcontrol.h"`

Macros

- #define **ALARMNMSZ** 18

Functions

- int **GhGetRandom** (int range)
- void **GhDelay** (int milliseconds)
- void **GhControllerInit** (void)
- double **GhGetHumidity** (void)
- double **GhGetPressure** (void)
- double **GhGetTemperature** (void)
- struct **readings GhGetReadings** (void)
- struct **setpoints GhSetTargets** (struct **setpoints** spts)
- struct **controls GhSetControls** (struct **setpoints** target, struct **readings** rdata)
- int **GhLogData** (char *fname, struct **readings** ghdata)
- int **GhSaveSetpoints** (char *fname, struct **setpoints** spts)
- struct **setpoints GhRetrieveSetpoints** (char *fname)
- **alarmlimit_s GhSetAlarmsLimits** (void)
- void **GhSetAlarms** (**alarm_s** calarm[**NALARMS**], **alarmlimit_s** alarmpt, struct **readings** rdata)
- **alarm_s GhDisplayAlarms** (**alarm_s** alrm[**NALARMS**])
- void **GhDisplayControls** (struct **controls** ctrl)
- void **GhDisplayReadings** (struct **readings** rdata)
- void **GhDisplayTargets** (struct **setpoints** spts)
- void **GhDisplaySetpoints** (void)
- void **GhDisplayHeader** (const char *sname)
- void **GhDisplayAll** (struct **readings** rd, struct **setpoints** sd)

Detailed Description

function definitions and in depth structure for Gh main code

Macro Definition Documentation

#define ALARMNMSZ 18

Function Documentation

void GhControllerInit (void)

sets parameters by calling srand for number generation, sets targets, and diplays controllers name. initialization of the Ghcontroller

Version:

06 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

void	

Returns:

void

```
50 {
51     srand((unsigned)time(NULL));
52     GhDisplayHeader("Bronson Pearl");
53     #if SENSEHAT
54         ShInit();
55     #endif
56 }
```

void GhDelay (int *milliseconds*)

creates a timed delay for the clock

Version:

01 June 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

integer	delay measured in milliseconds

Returns:

void

```
28 {
29     long int wait;
30     clock_t  now,start;
31 
32     wait = milliseconds * (CLOCKS_PER_SEC/1000);
33     start = clock();
34     now = start;
35 
36     while( (now-start) < wait)
```

```
 37     {
 38         now = clock();
 39     }
 40 }
```

alarm_s GhDisplayAlarms (alarm_s *alrm*[NALARMS])

```
327 {
328     // Alarm Message Array
329 const char alarmnames[NALARMS][ALARMNMSZ] =
330         {"No Alarms","High Temperature","Low Temperature","High Humidity",
331           "Low Humidity","High Pressure","Low Pressure"};
332
333           for(int i = 1; i<7; i++)
334           {
335             if('->' != NOALARM)
336             {
337                 printf("%s %s", alarmnames['->'],ctime(&alrm[i].atime));
338             }
339           }
340
341 }
```

void GhDisplayAll (struct readings *rd*, struct setpoints *sd*)

```
398 {
399     int rv, sv, ach, avl;
400     fbpixel_s pxc={0};
401
402     ShClearMatrix();
403
404     rv = (NUMPTS * (((rd.temperature-LSTEMP) / (USTEMP-LSTEMP)) +0.05) -1);
405     sv = (NUMPTS * (((sd.temperature - LSTEMP) / (USTEMP-LSTEMP)) +0.05) -1);
406         pxc.red = 0x00;
407         pxc.green = 0xFF;
408         pxc.blue = 0x00;
409         ShSetVerticalBar(TBAR,pxc,rv);
410         pxc.red = 0xF0;
411         pxc.green = 0x0F;
412         pxc.blue = 0xF0;
413         ShSetPixel(TBAR,sv,pxc);
414
415     rv = (NUMPTS * (((rd.humidity-LSHUMID) / (USHUMID-LSHUMID)) +0.05) -1);
416     sv = (NUMPTS * (((sd.humidity - LSHUMID) / (USHUMID-LSHUMID)) +0.05) -1);
417         pxc.red = 0x00;
418         pxc.green = 0xFF;
419         pxc.blue = 0x00;
420         ShSetVerticalBar(HBAR,pxc,rv);
421         pxc.red = 0xF0;
422         pxc.green = 0x0F;
423         pxc.blue = 0xF0;
424         ShSetPixel(HBAR,sv,pxc);
425
426     rv = (NUMPTS * (((rd.pressure-LSPRESS) / (USPRESS-LSPRESS)) +0.05) -1);
427         pxc.red = 0x00;
428         pxc.green = 0xFF;
429         pxc.blue = 0x00;
430         ShSetVerticalBar(PBAR,pxc,rv);
431
432
433 }
```

void GhDisplayControls (struct controls *ctrl*)

displays the on and off status of humidifier and heater based on current temperatures compared to target temperatures

Version:

06 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

heater	pointer humidifier pointer

Returns:

void

```
  352 {
  353     fprintf(stdout,"\nControls: \theater status: %d\t humidifier status:
%d\n",ctrl.heater, ctrl.humidifier);
  354 }
```

void GhDisplayHeader (const char * *sname*)

prints GH controller title

Version:

06 MAY 2018

Author:

Paul Moggach
Bronson Pearl

Parameters:

sname	string with Operator's name

Returns:

void

```
  393 {
  394     fprintf(stdout, "%s's CENG153 Greenhouse Controller\n",sname);
  395 }
```

void GhDisplayReadings (struct readings *rdata*)

prints the readings of units measured: temp, humidity, wind pressure.

Version:

08 June 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

rtime	and array dreads passed to display current time and appropriate measurements

Returns:

void

```
  364 {
  365     fprintf(stdout,"\n%sReadings\tT:%5.1fC\t H:%5.1lf%%\t P:%
6.1lfPa",ctime(&rdata.rtime),rdata.temperature,rdata.humidity,rdata.pressure);
  366 }
```

void GhDisplaySetpoints (void)

```
  381 {
  382 }
```

void GhDisplayTargets (struct setpoints *spts*)

displays set targets for humdifier and heater

Version:

15 June 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

void	

Returns:

void

```
376 {
377     fprintf(stdout,"\nSetpoints:  T: %5.1fC\t H: %5.1f%%", STEMP, SHUMID);
378 }
```

double GhGetHumidity (void)

generates simulated measurement for humidity

Version:

06 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

void	

Returns:

double

```
66 {
67
68 #if SIMTEMPERATURE
69     return GhGetRandom(USHUMID-LSHUMID) + LSHUMID;
70 #else
71     return 25.0;
72 #endif
73 }
```

double GhGetPressure (void)

generates simulated measurement for wind pressure

Version:

06 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

void	

Returns:

double

```
83 {
84 #if SIMTEMPERATURE
85     return GhGetRandom(USPRESS-LSPRESS) + LSPRESS;
86 #else
87     return 25.0;
88 #endif
89 }
```

int GhGetRandom (int *range*)

generates a random number within a range

Version:

01 June 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

range	integer mod division to random number

Returns:

int

```
 16 {
 17     return rand()%range;
 18 }
```

struct readings GhGetReadings (void)

assigns simulated measurements to the appropraite array location

Version:

06 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

type	double readings[SENSORS]

Returns:

void

```
115 {
116     struct readings now = {0};
117     now.rtime = time(NULL);
118     now.temperature = GhGetTemperature();
119     now.humidity = GhGetHumidity();
120     now.pressure = GhGetPressure();
121     return now;
122 }
```

double GhGetTemperature (void)

generates simulated measurement for temperature

Version:

06 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

void	

Returns:

double

```
 99 {
100 #if SIMTEMPERATURE
101     return GhGetRandom(USTEMP-LSTEMP) + LSTEMP;
102 #else
```

```
  103     return 25.0;
  104 #endif
  105 }
```

int GhLogData (char * *fname*, struct readings *ghdata*)

Writes data to a file to log simulated weather readings

Version:

13 July 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

fname	string with file name, ghdata

Returns:

1

```
  184 {
  185     FILE *fp;
  186     char ltime[25];
  187 
  188     fp = fopen(fname,"a");
  189 
  190     if(fp==NULL)
  191     {
  192     return 0;
  193     }
  194 
  195     strcpy(ltime, ctime(&ghdata.rtime));
  196     ltime[3] = ',';
  197     ltime[7] = ',';
  198     ltime[10]= ',';
  199     ltime[19]= ',';
  200 
  201 
  202     fprintf(fp,"%.24s,%5.1lf,%5.1lf,%6.1lf\n", ltime, ghdata.temperature,
ghdata.humidity, ghdata.pressure);
  203 
  204     fclose(fp);
  205 
  206     return 1;
  207 }
```

struct setpoints GhRetrieveSetpoints (char * *fname*)

```
  241 {
  242     struct setpoints spts = {0.0};
  243 
  244     FILE *fp;
  245 
  246     fp = fopen(fname,"r");
  247 
  248     if(fp==NULL)
  249     {
  250     return spts;
  251     }
  252 
  253     fread(&spts, sizeof(struct setpoints),1,fp);
  254 
  255     fclose(fp);
  256 
  257     return spts;
  258 }
```

int GhSaveSetpoints (char * *fname*, struct setpoints *spts*)

```
217 {
218     FILE *fp;
219     fp = fopen(fname,"w");
220 
221     if(fp==NULL)
222     {
223     return 0;
224     }
225 
226     fwrite(&spts,sizeof(struct setpoints),1, fp);
227 
228     fclose(fp);
229 
230     return 1;
231 }
```

void GhSetAlarms (alarm_s *calarm*[NALARMS], alarmlimit_s *alarmpt*, struct readings *rdata*)

```
274 {
275 
276 int i;
277 for(i=0; i<7; i++)
278 {
279     calarm[i].code = NOALARM;
280 }
281 
282 if(rdata.temperature >= alarmpt.hight)
283 {
284     calarm[HTEMP].code = HTEMP;
285     calarm[HTEMP].atime = rdata.rtime;
286     calarm[HTEMP].value = rdata.temperature;
287 }
288 
289 if(rdata.temperature <= alarmpt.lowt)
290 {
291     calarm[LTEMP].code = LTEMP;
292     calarm[LTEMP].atime = rdata.rtime;
293     calarm[LTEMP].value = rdata.temperature;
294 }
295 
296 if(rdata.humidity >= alarmpt.highh)
297 {
298     calarm[HHUMID].code = HHUMID;
299     calarm[HHUMID].atime = rdata.rtime;
300     calarm[HHUMID].value = rdata.temperature;
301 }
302 
303 if(rdata.humidity <= alarmpt.lowh)
304 {
305     calarm[LHUMID].code = LHUMID;
306     calarm[LHUMID].atime = rdata.rtime;
307     calarm[LHUMID].value = rdata.temperature;
308 }
309 
310 if(rdata.pressure >= alarmpt.highp)
311 {
312     calarm[HPRESS].code = HPRESS;
313     calarm[HPRESS].atime = rdata.rtime;
314     calarm[HPRESS].value = rdata.temperature;
315 }
316 
317 if(rdata.pressure <= alarmpt.lowp)
318 {
319     calarm[LPRESS].code = LPRESS;
320     calarm[LPRESS].atime = rdata.rtime;
321     calarm[LPRESS].value = rdata.temperature;
322 }
323 
```

```
324 }
```

alarmlimit_s GhSetAlarmsLimits (void)

```
261 {
262     alarmlimit_s calarm;
263     calarm.hight = UPPERATEMP;
264     calarm.lowt = LOWERATEMP;
265     calarm.highh = UPPERAHUMID;
266     calarm.lowh = LOWERAHUMID;
267     calarm.highp = UPPERAPRESS;
268     calarm.lowp = LOWERAPRESS;
269
270     return calarm;
271 }
```

struct controls GhSetControls (struct setpoints *target*, struct readings *rdata*)

controls the heater and humidifier for green house

Version:

15 June 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

heater	pointer humidifier pointer and creadings array

Returns:

void

```
152 {
153     struct controls cset = {0};
154
155     if( rdata.temperature < STEMP )
156     {
157          cset.heater = ON;
158     }
159     else
160     {
161          cset.heater = OFF;
162     }
163
164     if( rdata.humidity < SHUMID)
165     {
166          cset.humidifier = ON;
167     }
168     else
169     {
170          cset.humidifier = OFF;
171     }
172
173     return cset;
174 }
```

struct setpoints GhSetTargets (struct setpoints *spts*)

sets contorl points for simulation

Version:

06 july 2020

Author:

Paul Moggach
Bronson Pearl

Parameters:

type	void

Returns:

cpoints

```
132 {
133     struct setpoints cpoints = GhRetrieveSetpoints("setpoints.dat");
134
135     if( (cpoints.temperature = 0) )
136     {
137     cpoints.temperature = STEMP;
138     cpoints.humidity = SHUMID;
139     GhSaveSetpoints("setpoints.dat", cpoints);
140     }
141     return cpoints;
142 }
```

---

ghcontrol.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "pisensehat.h"
```

Data Structures

- struct **readings**
- struct **setpoints**
- struct **controls**
- struct **alarmlimits**
- struct **alarms**

Macros

- #define **ON** 1
- #define **OFF** 0
- #define **STEMP** 25.0
- #define **SHUMID** 55.0
- #define **SIMULATE** 1
- #define **USTEMP** 50
- #define **LSTEMP** -10
- #define **USHUMID** 100
- #define **LSHUMID** 0
- #define **USPRESS** 1016
- #define **LSPRESS** 975
- #define **GHUPDATE** 2000
- #define **SENSORS** 3
- #define **TEMPERATURE** 0
- #define **HUMIDITY** 1
- #define **PRESSURE** 2
- #define **SIMTEMPERATURE** 1
- #define **SIMHUMIDITY** 1
- #define **SIMPRESSURE** 1

- #define **NUMBARS** 8
- #define **NUMPTS** 8.0
- #define **TBAR** 7
- #define **HBAR** 5
- #define **PBAR** 3
- #define **SENSEHAT** 1
- #define **NALARMS** 7
- #define **UPPERATEMP** 30
- #define **LOWERATEMP** 10
- #define **LOWERAHUMID** 25
- #define **UPPERAHUMID** 70
- #define **LOWERAPRESS** 985
- #define **UPPERAPRESS** 1016

Typedefs

- typedef struct **alarmlimits alarmlimit_s**
- typedef struct **alarms alarm_s**

Enumerations

- enum **alarm_e** { **NOALARM**, **HTEMP**, **LTEMP**, **HHUMID**, **LHUMID**, **HPRESS**, **LPRESS** }

Detailed Description

GH control constants, structures, function prototypes for Gh control code

---

Data Structure Documentation

struct readings

Data Fields:

double	humidity	
double	pressure	
time_t	rtime	
double	temperature	

struct setpoints

Data Fields:

double	humidity	
double	temperature	

struct controls

Data Fields:

int	heater	
int	humidifier	

struct alarmlimits

Data Fields:

double	highh	
double	highp	
double	hight	
double	lowh	
double	lowp	
double	lowt	

struct alarms

Data Fields:

time_t	atime	
alarm_e	code	
double	value	

Macro Definition Documentation

#define GHUPDATE 2000

#define HBAR 5

#define HUMIDITY 1

#define LOWERAHUMID 25

#define LOWERAPRESS 985

#define LOWERATEMP 10

#define LSHUMID 0

#define LSPRESS 975

#define LSTEMP -10

#define NALARMS 7

#define NUMBARS 8

#define NUMPTS 8.0

#define OFF 0

#define ON 1

#define PBAR 3

#define PRESSURE 2

#define SENSEHAT 1

#define SENSORS 3

#define SHUMID 55.0

#define SIMHUMIDITY 1

#define SIMPRESSURE 1

#define SIMTEMPERATURE 1

#define SIMULATE 1

#define STEMP 25.0

#define TBAR 7

#define TEMPERATURE 0

#define UPPERAHUMID 70

#define UPPERAPRESS 1016

#define UPPERATEMP 30

#define USHUMID 100

#define USPRESS 1016

#define USTEMP 50

Typedef Documentation

typedef struct alarms alarm_s

typedef struct alarmlimits alarmlimit_s

Enumeration Type Documentation

enum alarm_e

Enumerator:

NOALARM	
HTEMP	
LTEMP	
HHUMID	
LHUMID	
HPRESS	
LPRESS	

```
52 { NOALARM,HTEMP,LTEMP,HHUMID,LHUMID,HPRESS,LPRESS } alarm_e;
```

---

pisensehat.c File Reference

```
#include "pisensehat.h"
```

Functions

- int **ShInit** (void)

- int **ShExit** (void)
- void **ShClearMatrix** (void)
- uint8_t **ShSetPixel** (int x, int y, **fbpixel_s** px)
- int **ShSetVerticalBar** (int bar, **fbpixel_s** px, uint8_t value)
- **lps25hData_s ShGetLPS25HData** (void)
- **ht221sData_s ShGetHT221SData** (void)

Variables

- static int **fbfd**
- static uint16_t * **map**
- static int **HTS221fd**
- static int **LPS25Hfd**
- int **numReadings** =0

Detailed Description

RPi Sensehat functions

Version:

2020-05-03

Function Documentation

void ShClearMatrix (void)

Clears Sensehat 8X8 RGB LED display

Author:

Paul Moggach
Kristian Medri

Version:

2020-05-03

Parameters:

void	

Returns:

void

```
108 {
109 #if EMULATOR
110     if (numReadings >=12){
111         numReadings=0;
112         printf("12 readings is about the limit for the emulator\n"
113              "the way that the current code is written since\n"
114              "it spawns too many threads and using Py_Finalize\n"
115              "causes a decref segmentation fault. In addition,\n"
116              "it doesn't respond to Ctrl-C thus exiting gracefully.\n");
117      /* Note that if you want to exit sooner you can stop the ghc process
118     by using Ctrl-Z, find the PID of ghc by using the command ps, and
119     use kill -9 PID# to end the process. */
120         exit(EXIT_FAILURE);
121     }
122     else{
123         //printf("numReadings= %d\n",numReadings);
124         numReadings++;
125     }
126         PyRun_SimpleString(
```

```
127         "from sense_emu import SenseHat\n"
128         "sense=SenseHat()\n"
129         "sense.clear()\n"
130         );
131 #else
132         memset(map, 0, FILESIZE);
133 #endif
134 }
```

int ShExit (void)

Closes Down the Sensehat

Author:

Paul Moggach
Kristian Medri

Version:

2020-05-01

Parameters:

void	

Returns:

exit status

```
82 {
83 #if EMULATOR
84     Py_Finalize();
85 #else
86     ShClearMatrix();
87     /* un-map and close */
88     if (munmap(map, FILESIZE) == -1)
89     {
90         perror("Error un-mmapping the file");
91         return EXIT_FAILURE;
92     }
93     close(fbfd);
94     close(HTS221fd);
95     close(LPS25Hfd);
96 #endif
97     return EXIT_SUCCESS;
98 }
```

ht221sData_s ShGetHT221SData (void)

Gets HT221S Sensehat sensor data

Author:

Paul Moggach
Kristian Medri

Version:

2020-05-03

Parameters:

void	

Returns:

ht221sData_s temperature and humidity data

```
286 {
287     ht221sData_s rd = {0};
288 #if EMULATOR
289     PyRun_SimpleString(
290         "from sense_emu import SenseHat\n"
291         "#from time import time,ctime\n"
292         "#print('Today is '+ctime(time))\n"
293         "sense=SenseHat()\n"
```

```
  294         "temp=sense.temp\n"
  295         "humid=sense.humidity\n"
  296         "#print(temp)\n"
  297         "#print(humid)\n"
  298         "f=open(\"tempfileforpython.txt\",\"w\")\n"
  299         "f.write(repr(temp))\n"
  300         "f.close()\n"
  301         "f=open(\"humifileforpython.txt\",\"w\")\n"
  302         "f.write(repr(humid))\n"
  303         "f.close()\n"
  304         );
  305     double reading=0;
  306     FILE *fp;
  307     fp=fopen("tempfileforpython.txt","r");
  308     fscanf(fp, "%lf", &reading);
  309     fclose(fp);
  310     rd.temperature = reading;
  311     //fprintf(stdout, "%lf\n", reading);
  312     fp=fopen("humifileforpython.txt","r");
  313     fscanf(fp, "%lf", &reading);
  314     fclose(fp);
  315     //fprintf(stdout, "%lf\n", reading);
  316     rd.humidity = reading;
  317 #else
  318     int status;
  319     uint8_t t0_out_l,t0_out_h,t1_out_l,t1_out_h;
  320     uint8_t t0_degC_x8,t1_degC_x8,t1_t0_msb;
  321     int16_t T0_OUT,T1_OUT;
  322     uint16_t T0_DegC_x8,T1_DegC_x8;
  323     double T0_DegC,T1_DegC;
  324     double t_gradient_m,t_intercept_c;
  325     uint8_t t_out_l,t_out_h;
  326     int16_t T_OUT;
  327     uint8_t
h0_out_l,h0_out_h,h1_out_l,h1_out_h,h0_rh_x2,h1_rh_x2,h_t_out_l,h_t_out_h;
  328     int16_t H0_T0_OUT,H1_T0_OUT,H_T_OUT;
  329     double H0_rH,H1_rH,h_gradient_m,h_intercept_c;
  330
  331     // Power down the device (clean start)
  332     wiringPiI2CWriteReg8(HTS221fd, CTRL_REG1, 0x00);
  333     // Turn on the humidity sensor analog front end in single shot mode
  334     wiringPiI2CWriteReg8(HTS221fd, CTRL_REG1, 0x84);
  335     // Run one-shot measurement (temperature and humidity). The set bit will be reset
by the
  336     // sensor itself after execution (self-clearing bit)
  337     wiringPiI2CWriteReg8(HTS221fd, CTRL_REG2, 0x01);
  338
  339     // Wait until the measurement is completed
  340     do
  341     {
  342         usleep(HTS221DELAY);    // 25 ms
  343         status = wiringPiI2CReadReg8(HTS221fd, CTRL_REG2);
  344     }
  345     while (status != 0);
  346
  347     // Read calibration temperature LSB (ADC) data
  348     // (temperature calibration x-data for two points)
  349     t0_out_l = wiringPiI2CReadReg8(HTS221fd, T0_OUT_L);
  350     t0_out_h = wiringPiI2CReadReg8(HTS221fd, T0_OUT_H);
  351     t1_out_l = wiringPiI2CReadReg8(HTS221fd, T1_OUT_L);
  352     t1_out_h = wiringPiI2CReadReg8(HTS221fd, T1_OUT_H);
  353
  354    // Read calibration relative humidity LSB (ADC) data
  355     // (humidity calibration x-data for two points)
  356     h0_out_l = wiringPiI2CReadReg8(HTS221fd, H0_T0_OUT_L);
  357     h0_out_h = wiringPiI2CReadReg8(HTS221fd, H0_T0_OUT_H);
  358     h1_out_l = wiringPiI2CReadReg8(HTS221fd, H1_T0_OUT_L);
  359     h1_out_h = wiringPiI2CReadReg8(HTS221fd, H1_T0_OUT_H);
  360
  361     // Read calibration temperature (°C) data
  362     // (temperature calibration y-data for two points)
```

```
363     t0_degC_x8 = wiringPiI2CReadReg8(HTS221fd, T0_degC_x8);
364     t1_degC_x8 = wiringPiI2CReadReg8(HTS221fd, T1_degC_x8);
365     t1_t0_msb = wiringPiI2CReadReg8(HTS221fd, T1_T0_MSB);
366
367    // Read relative humidity (% rH) data
368     // (humidity calibration y-data for two points)
369     h0_rh_x2 = wiringPiI2CReadReg8(HTS221fd, H0_rH_x2);
370     h1_rh_x2 = wiringPiI2CReadReg8(HTS221fd, H1_rH_x2);
371
372     // make 16 bit values (bit shift)
373     // (temperature calibration x-values)
374     T0_OUT = t0_out_h << 8 | t0_out_l;
375     T1_OUT = t1_out_h << 8 | t1_out_l;
376
377     // make 16 and 10 bit values (bit mask and bit shift)
378     T0_DegC_x8 = (t1_t0_msb & 3) << 8 | t0_degC_x8;
379     T1_DegC_x8 = ((t1_t0_msb & 12) >> 2) << 8 | t1_degC_x8;
380
381     // Calculate calibration values
382     // (temperature calibration y-values)
383     T0_DegC = T0_DegC_x8 / 8.0;
384     T1_DegC = T1_DegC_x8 / 8.0;
385
386     // Solve the linear equasions 'y = mx + c' to give the
387     // calibration straight line graphs for temperature and humidity
388     t_gradient_m = (T1_DegC - T0_DegC) / (T1_OUT - T0_OUT);
389     t_intercept_c = T1_DegC - (t_gradient_m * T1_OUT);
390
391     // Read the ambient temperature measurement (2 bytes to read)
392     t_out_l = wiringPiI2CReadReg8(HTS221fd, TEMP_OUT_L);
393     t_out_h = wiringPiI2CReadReg8(HTS221fd, TEMP_OUT_H);
394
395     // make 16 bit value
396     T_OUT = t_out_h << 8 | t_out_l;
397
398     // make 16 bit values (bit shift)
399     // (humidity calibration x-values)
400     H0_T0_OUT = h0_out_h << 8 | h0_out_l;
401     H1_T0_OUT = h1_out_h << 8 | h1_out_l;
402
403     // Humidity calibration values
404     // (humidity calibration y-values)
405     H0_rH = h0_rh_x2 / 2.0;
406     H1_rH = h1_rh_x2 / 2.0;
407     h_gradient_m = (H1_rH - H0_rH) / (H1_T0_OUT - H0_T0_OUT);
408     h_intercept_c = H1_rH - (h_gradient_m * H1_T0_OUT);
409
410     // Read the ambient humidity measurement (2 bytes to read)
411     h_t_out_l = wiringPiI2CReadReg8(HTS221fd, H_T_OUT_L);
412     h_t_out_h = wiringPiI2CReadReg8(HTS221fd, H_T_OUT_H);
413
414     // make 16 bit value
415     H_T_OUT = h_t_out_h << 8 | h_t_out_l;
416
417     // Power down the device
418     wiringPiI2CWriteReg8(HTS221fd, CTRL_REG1, 0x00);
419
420     // Calculate and return ambient temperature
421     rd.temperature = (t_gradient_m * T_OUT) + t_intercept_c;
422     rd.humidity = (h_gradient_m * H_T_OUT) + h_intercept_c;
423 #endif
424     return rd;
425 }
```

lps25hData_s ShGetLPS25HData (void)

Gets LPS25H Sensehat sensor information

Author:

Paul Moggach

Kristian Medri

Version:

2020-05-01

Parameters:

void	

Returns:

lps25hData_s pressure and temperature data

```
  210 {
  211     lps25hData_s rd = {0};
  212 #if EMULATOR
  213     PyRun_SimpleString(
  214         "from sense_emu import SenseHat\n"
  215         "sense=SenseHat()\n"
  216         "temp=sense.pressure\n"
  217         "f=open(\"tempfileforpython.txt\",\"w\")\n"
  218         "f.write(repr(temp))\n"
  219         "f.close()\n"
  220         );
  221     double reading=0;
  222     FILE *fp;
  223     fp=fopen("tempfileforpython.txt","r");
  224     fscanf(fp, "%lf", &reading);
  225     fclose(fp);
  226     rd.pressure = reading;
  227     rd.temperature = 5; //placeholder, use the temperature from the ht221s
  228 #else
  229     uint8_t temp_out_l = 0, temp_out_h = 0;
  230     int16_t temp_out = 0;
  231     uint8_t press_out_xl = 0;
  232     uint8_t press_out_l = 0;
  233     uint8_t press_out_h = 0;
  234     int32_t press_out = 0;
  235     uint8_t status = 0;
  236 
  237     // Power down the device (clean start)
  238     wiringPiI2CWriteReg8(LPS25Hfd, CTRL_REG1, 0x00);
  239 
  240     // Turn on the humidity sensor analog front end in single shot mode
  241     wiringPiI2CWriteReg8(LPS25Hfd, CTRL_REG1, 0x84);
  242 
  243     // Run one-shot measurement (temperature and humidity). The set bit will be reset
by the
  244     // sensor itself after execution (self-clearing bit)
  245     wiringPiI2CWriteReg8(LPS25Hfd, CTRL_REG2, 0x01);
  246 
  247     // Wait until the measurement is completed
  248     do
  249     {
  250         usleep(HTS221DELAY);    // 25 ms
  251         status = wiringPiI2CReadReg8(LPS25Hfd, CTRL_REG2);
  252     }
  253     while (status != 0);
  254 
  255     /* Read the temperature measurement (2 bytes to read) */
  256     temp_out_l = wiringPiI2CReadReg8(LPS25Hfd, TEMP_OUT_L);
  257     temp_out_h = wiringPiI2CReadReg8(LPS25Hfd, TEMP_OUT_H);
  258 
  259     /* Read the pressure measurement (3 bytes to read) */
  260     press_out_xl = wiringPiI2CReadReg8(LPS25Hfd, PRESS_OUT_XL);
  261     press_out_l = wiringPiI2CReadReg8(LPS25Hfd, PRESS_OUT_L);
  262     press_out_h = wiringPiI2CReadReg8(LPS25Hfd, PRESS_OUT_H);
  263 
  264     /* make 16 and 24 bit values (using bit shift) */
  265     temp_out = temp_out_h << 8 | temp_out_l;
  266     press_out = press_out_h << 16 | press_out_l << 8 | press_out_xl;
  267 
  268     /* calculate output values */
```

```
269     rd.temperature = 42.5 + (temp_out / 480.0);
270     rd.pressure = press_out / 4096.0;
271 
272     // Power down the device
273     wiringPiI2CWriteReg8(LPS25Hfd, CTRL_REG1, 0x00);
274 #endif
275     return rd;
276 }
```

int ShInit (void)

Initialize Sensehat

Author:

Paul Moggach
Kristian Medri

Version:

2020-05-01

Parameters:

void	

Returns:

exit status

```
22 {
23 #if EMULATOR
24     Py_Initialize();
25 #else
26     wiringPiSetup();
27     struct fb_fix_screeninfo fix_info;
28 
29     // Frame Buffer Initialization for 8X8 LED Matrix
30     /* open the led frame buffer device */
31     fbfd = open(FILEPATH, O_RDWR);
32     if (fbfd == -1)
33     {
34         perror("Error (call to 'open')");
35         exit(EXIT_FAILURE);
36     }
37 
38     /* read fixed screen info for the open device */
39     if (ioctl(fbfd, FBIOGET_FSCREENINFO, &fix_info) == -1)
40     {
41         perror("Error (call to 'ioctl')");
42         close(fbfd);
43         exit(EXIT_FAILURE);
44     }
45 
46     /* now check the correct device has been found */
47     if (strcmp(fix_info.id, "RPi-Sense FB") != 0)
48     {
49         printf("%s\n", "Error: RPi-Sense FB not found");
50         close(fbfd);
51         exit(EXIT_FAILURE);
52     }
53 
54     /* map the led frame buffer device into memory */
55     map = mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fbfd, 0);
56     if (map == MAP_FAILED)
57     {
58         close(fbfd);
59         perror("Error mmapping the file");
60         exit(EXIT_FAILURE);
61     }
62 
63     // Sensor Initialization
64     HTS221fd = wiringPiI2CSetup(HTS221I2CADDRESS);
65     LPS25Hfd = wiringPiI2CSetup(LPS25HI2CADDRESS);
```

```
 66
 67     // Power down the device (clean start)
 68     wiringPiI2CWriteReg8(HTS221fd, CTRL_REG1, 0x00);
 69     wiringPiI2CWriteReg8(LPS25Hfd, CTRL_REG1, 0x00);
 70 #endif
 71     return EXIT_SUCCESS;
 72 }
```

uint8_t ShSetPixel (int *x*, int *y*, fbpixel_s *px*)

Sets a pixel on the Sensehat display

Author:

Paul Moggach
Kristian Medri

Version:

2020-05-01

Parameters:

x	an integer position value
y	an integer position value
fbpixel_s	pixel colour data

Returns:

uint8_t exit status

```
146 {
147 #if EMULATOR
148     char ltime [120];
149     sprintf(ltime,
150         "from sense_emu import SenseHat\n"
151         "sense=SenseHat()\n"
152         "sense.set_pixel(%d,%d,%d,%d,%d)\n"
153         ,x,y,px.red,px.green,px.blue);
154     PyRun_SimpleString(ltime);
155     return EXIT_SUCCESS;
156 #else
157     int i;
158
159     if (x >= 0 && x < 8 && y >= 0 && y < 8)
160     {
161         i = (y*8)+x; // offset into array
162         map[i] = (px.red << 11) | (px.green << 5) | (px.blue);
163         return EXIT_SUCCESS;
164     }
165 #endif
166     return EXIT_FAILURE;
167 }
```

int ShSetVerticalBar (int *bar*, fbpixel_s *px*, uint8_t *value*)

Sets a vertical bar on the Sensehat display

Author:

Paul Moggach
Kristian Medri

Version:

2020-05-01

Parameters:

int	bar to light
fbpixel_s	pixel colour data
uint8_t	value how many pixels to light in bar

Returns:

exit status

```
179 {
180     int i;
181     if (value>7){
182         value=7;
183     }
184     if (bar >= 0 && bar < 8 && value >= 0 && value < 8)
185     {
186         for(i=0; i<= value; i++)
187         {
188             ShSetPixel(bar,i,px);
189         }
190         px.red = 0x00;
191         px.green = 0x00;
192         px.blue = 0x00;
193         for(i=value+1; i< 8;i++)
194         {
195             ShSetPixel(bar,i,px);
196         }
197         return EXIT_SUCCESS;
198     }
199     return EXIT_FAILURE;
200 }
```

Variable Documentation

int fbfd [static]

int HTS221fd [static]

int LPS25Hfd [static]

uint16_t* map [static]

int numReadings =0

---

pisensehat.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <string.h>
#include <linux/fb.h>
#include <sys/ioctl.h>
#include <poll.h>
#include <dirent.h>
```

```
#include <linux/input.h>
#include <time.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
```

Data Structures

- struct **fbpixel**
- struct **lps25hData**
- struct **ht221sData**

Macros

- #define **EMULATOR** 0
- #define **LPS25HI2CADDRESS** 0x5c
- #define **PRESS_OUT_XL** 0x28
- #define **PRESS_OUT_L** 0x29
- #define **PRESS_OUT_H** 0x2A
- #define **HTS221I2CADDRESS** 0x5F
- #define **HTS221DELAY** 25000
- #define **WHO_AM_I** 0x0F
- #define **CTRL_REG1** 0x20
- #define **CTRL_REG2** 0x21
- #define **T0_OUT_L** 0x3C
- #define **T0_OUT_H** 0x3D
- #define **T1_OUT_L** 0x3E
- #define **T1_OUT_H** 0x3F
- #define **T0_degC_x8** 0x32
- #define **T1_degC_x8** 0x33
- #define **T1_T0_MSB** 0x35
- #define **TEMP_OUT_L** 0x2A
- #define **TEMP_OUT_H** 0x2B
- #define **H0_T0_OUT_L** 0x36
- #define **H0_T0_OUT_H** 0x37
- #define **H1_T0_OUT_L** 0x3A
- #define **H1_T0_OUT_H** 0x3B
- #define **H0_rH_x2** 0x30
- #define **H1_rH_x2** 0x31
- #define **H_T_OUT_L** 0x28
- #define **H_T_OUT_H** 0x29
- #define **FILEPATH** "/dev/fb1"
- #define **NUM_WORDS** 64
- #define **FILESIZE** (**NUM_WORDS** * sizeof(uint16_t))
- #define **RGB565_RED** 0xF800
- #define **RGB565_GREEN** 0x07E0
- #define **RGB565_BLUE** 0x001F

Typedefs

- typedef struct **fbpixel fbpixel_s**
- typedef struct **lps25hData lps25hData_s**
- typedef struct **ht221sData ht221sData_s**

Detailed Description

RPi Sensehat constants, structures, function prototypes

Version:

2020-05-03

Data Structure Documentation

struct fbpixel

Data Fields:

uint8_t	blue	
uint8_t	green	
uint8_t	red	

struct lps25hData

Data Fields:

double	pressure	
double	temperature	

struct ht221sData

Data Fields:

double	humidity	
double	temperature	

Macro Definition Documentation

#define CTRL_REG1 0x20

#define CTRL_REG2 0x21

#define EMULATOR 0

#define FILEPATH "/dev/fb1"

#define FILESIZE (NUM_WORDS * sizeof(uint16_t))

#define H0_rH_x2 0x30

#define H0_T0_OUT_H 0x37

#define H0_T0_OUT_L 0x36

#define H1_rH_x2 0x31

#define H1_T0_OUT_H 0x3B

#define H1_T0_OUT_L 0x3A

#define H_T_OUT_H 0x29

#define H_T_OUT_L 0x28

#define HTS221DELAY 25000

#define HTS221I2CADDRESS 0x5F

#define LPS25HI2CADDRESS 0x5c

#define NUM_WORDS 64

#define PRESS_OUT_H 0x2A

#define PRESS_OUT_L 0x29

#define PRESS_OUT_XL 0x28

#define RGB565_BLUE 0x001F

#define RGB565_GREEN 0x07E0

#define RGB565_RED 0xF800

#define T0_degC_x8 0x32

#define T0_OUT_H 0x3D

#define T0_OUT_L 0x3C

#define T1_degC_x8 0x33

#define T1_OUT_H 0x3F

#define T1_OUT_L 0x3E

#define T1_T0_MSB 0x35

#define TEMP_OUT_H 0x2B

#define TEMP_OUT_L 0x2A

#define WHO_AM_I 0x0F

Typedef Documentation

typedef struct fbpixel fbpixel_s

typedef struct ht221sData ht221sData_s

typedef struct lps25hData lps25hData_s

Index