Honeywell Linux Assessment

Generated by Doxygen 1.11.0

Chapter 1

Honeywell Linux Assessment Documentation

1.1 Introduction

The test consisted of creating an application for a temperature sensor connected to an embedded Linux development board.

DISCLAIMER:
As a candidate for Senior Advanced Embedded Engineer position I decided to carry this project to the physical implementation, meaning that all the hardware and software cited in here was tested in the loop by me (Mauricio Gutierrez).

1.1.1 Yocto with Poky for Raspberry Pi

Yocto is an open-source project that allows the creation of custom Linux systems for embedded devices. Poky is the reference build system in Yocto, providing the tools and metadata necessary to create Linux images. Using Yocto with Poky on a Raspberry Pi allows for the creation of a highly customized and optimized operating system for specific applications.

1.1.2 AHT10 Temperature Sensor

The AHT10 is a high-precision temperature and humidity sensor widely used in embedded applications. It offers a simple I2C interface for communication with microcontrollers and embedded systems. This project uses the AHT10 to measure ambient temperature and send this data over a TCP/IP network.

1.1.3 Client-Server Architectures in Embedded Systems

Client-server architectures are fundamental in data communication in embedded systems. In this project, an embedded server on the Raspberry Pi communicates with multiple clients to provide real-time temperature data. The server manages TCP connections and uses threads to handle client requests and data acquisition from the AHT10 sensor.

1.2 Hardware setup

1.2.1 Step 1: I2C Connections

AHT10 temperature sensor has a I2C interface for communication with only 4 pins:

- VIN (5v Power)
- GND (Ground)
- SCL (GPIO 3 Clock)
- SDA (GPIO 2 Data)

Which can be directly connected to IC2 pins on Raspberry Pi 3B+ (model used for this project):

1.3 Software setup

We decided to use Poky *meta-raspberrypi layer* with Yocto (dunfell release) to build our custom image for Raspberry Pi 3 B+ (BCM2837 processor).
Adding i2d_dev to /etc/modules yields:
modprobe -v -n -D i2c_dev
insmod /lib/modules/5.4.72-v8/drivers/i2c/i2c-dev.ko
Editing the file poky/build/conf/local.conf file to configure enabling I2C during boot.
Rebuiding Yocto:
Output found at
/build/tmp/deploy/images/raspberrypi3-64/bootfiles/config.txt
While I2C device is now present and correctly loaded at /dev/, we want the development tools to start programming by adding the following line in the Yocto configuration file.

We can see that now i2ctransfer is working by executing:
Handshaking:

- i2ctransfer -y 1 w3@0x38 0xE1 0x33 0x00 r6

Data request:

- i2ctransfer -y 1 w3@0x38 0xac 0x33 0x00 r6

Conditioning of the signal according to AHT10 datasheet:
After making threads for

- 1) data gathering,
- 2) data processing and
- 3) service requesting,

this is the console output:
On the left side of the console, we can see the silent server working at port 5000. On the right we can see the request to the server by using telnet as client. We double checked the correct behavior by using another external client application called Hercules:
For watchdog purposes, we need to add the package in Yocto and rebuild the image again with Systemd enbaled:

- DISTRO_FEATURES:append = " systemd"
- DISTRO_FEATURES_BACKFILL_CONSIDERED += "sysvinit"
- VIRTUAL-RUNTIME_init_manager = "systemd"
- VIRTUAL-RUNTIME_initscripts = "systemd-compat-units"
- INIT_MANAGER = "systemd"

A Systemd service was designed to automatically start the application after the user level (3) is reached. As per project requirements (5, 6), the application will be restarted after 20 seconds if it crashed, killed or closed, as shown below:

_
After reboot we can see the service is active and correctly working:
Finally, we configure the watchdog to monitor the "real-time" system. If the board load goes above the '24' parameter the system will automatically reboot.

- echo 'dtparam=watchdog=on' >> /boot/config.txt

If the system doesn't respond during over 15 seconds, the hardware watchdog signal will reboot the OS.

- vim /etc/watchdog.conf
- systemctl enable watchdog

After correctly configuring the board's watchdog we can enable the WD service via Systemd, as showed below:
We ran the famous fork bomb to test it:

- bash -c ':(){ :|:& };:'

After the fork bomb exploited the system frozen during 15 seconds proximately and the automatically restarted.
This covers the 6th and last requirement from the Honeywell Linux Assessment.

1.4 Software compilation method

For this project (assessment) we designed, developed and built 2 applications:

- A HAL application that is able to call from user space the I2C kernel functions, and
- A TCP multithreading server that gets sensor data, transform data to a lecture and attend incoming requests for temperature service subscribers.

Since Raspberry Pi 3B+ board has enough computational resources, we decided to do not crosscompile but compile the sources directly in the board.
In this regard, the command to compile is:
for HAL application:

- *gcc hw_aht10_get_temp.c - o hw_aht10_get_temp*

for TCP server application:

- *gcc -pthread hw_aht10_temperature_server.c hw_aht10_func.c -o hw_aht10_temperature_server*

1.4.1 Running the app

Both applications can run with parameter, however for the HAL application the parameters are madatory:

- Usage: ./hw_aht10_get_temp device_name device_address
- e.g.: ./hw_aht10_get_temp /dev/i2c-1 0x38

This allows the possibility the change/update the sensor without changing the source code.
On the other hand, the TCP server application can run with or without parameters.

- Usage: ./hw_aht10_temperature_server device_name device_address port
- e.g. (default) ./hw_aht10_temperature_server /dev/i2c-1 0x38 5000

This means that the default values are the I2C descriptor 1, the default addres for AHT10 sensor and TCP Port 5000.
NOTE: As we previously showed, is not necessary to manually run the server since it is automatically started and monitored via software (systemd service) and hardware (watchdog).

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 SensorInfo Struct Reference

`#include <hw_aht10_func.h>`

Data Fields

- char dev_name [MAX_SIZE]
- int dev_address
- char data [6]

4.1.1 Detailed Description

Definition at line 6 of file hw_aht10_func.h.

4.1.2 Field Documentation

4.1.2.1 data

`char data[6]`
Definition at line 9 of file hw_aht10_func.h.

4.1.2.2 dev_address

`int dev_address`
Definition at line 8 of file hw_aht10_func.h.

4.1.2.3 dev_name

`char dev_name[MAX_SIZE]`
Definition at line 7 of file hw_aht10_func.h.
The documentation for this struct was generated from the following file:

- hw_aht10_func.h

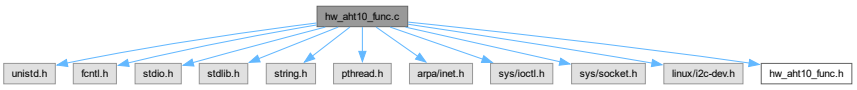
Chapter 5

File Documentation

5.1 hw_aht10_func.c File Reference

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <linux/i2c-dev.h>
#include "hw_aht10_func.h"
```

Include dependency graph for hw_aht10_func.c:



Functions

- int validate_inputs (int argc, char **argv)
- char * get_data_from_sensor (char *device_name, int dev_addr)
- float compute_temperature_celsius (char *data)
- int start_listener (unsigned short port)
- void * read_sensor (void *arg)
- void * compute_temperature (void *arg)
- void send_message (int client_socket)
- void * connection_handler (void *socket_desc)

Variables

- pthread_mutex_t mutex
- pthread_cond_t cond
- int data_ready = e_gathering
- float temperature

5.1.1 Function Documentation

5.1.1.1 compute_temperature()

```
void * compute_temperature (
            void * arg)
```

Definition at line 152 of file hw_aht10_func.c.

5.1.1.2 compute_temperature_celsius()

```
float compute_temperature_celsius (
            char * data)
```

Definition at line 103 of file hw_aht10_func.c.

5.1.1.3 connection_handler()

```
void * connection_handler (
            void * socket_desc)
```

Definition at line 175 of file hw_aht10_func.c.

5.1.1.4 get_data_from_sensor()

```
char * get_data_from_sensor (
            char * device_name,
            int dev_addr)
```

Definition at line 41 of file hw_aht10_func.c.

5.1.1.5 read_sensor()

```
void * read_sensor (
            void * arg)
```

Definition at line 136 of file hw_aht10_func.c.

5.1.1.6 send_message()

```
void send_message (
            int client_socket)
```

Definition at line 168 of file hw_aht10_func.c.

5.1.1.7 start_listener()

```
int start_listener (
            unsigned short port)
```

Definition at line 111 of file hw_aht10_func.c.

5.1.1.8 validate_inputs()

```
int validate_inputs (
            int argc,
            char ** argv)
```

Definition at line 19 of file hw_aht10_func.c.

5.1.2 Variable Documentation

5.1.2.1 cond

```
pthread_cond_t cond
```

Definition at line 15 of file hw_aht10_func.c.

5.1.2.2 data_ready

`int data_ready = e_gathering`
Definition at line 16 of file hw_aht10_func.c.

5.1.2.3 mutex

`pthread_mutex_t mutex`
Definition at line 14 of file hw_aht10_func.c.

5.1.2.4 temperature

`float temperature`
Definition at line 17 of file hw_aht10_func.c.

5.2 hw_aht10_func.c

Go to the documentation of this file.

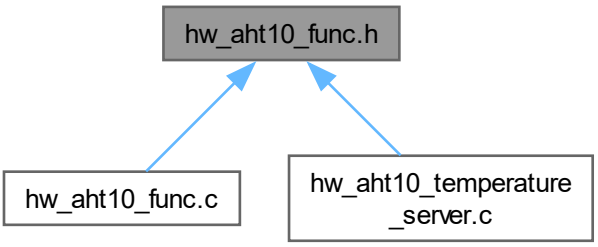
```
00001 #include <unistd.h>
00002 #include <fcntl.h>
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <string.h>
00006 #include <pthread.h>
00007 #include <arpa/inet.h>
00008 #include <sys/ioctl.h>
00009 #include <sys/socket.h>
00010 #include <linux/i2c-dev.h>
00011 #include "hw_aht10_func.h"
00012
00013 /* Shared data and sync for threads*/
00014 pthread_mutex_t mutex;
00015 pthread_cond_t cond;
00016 int data_ready = e_gathering;
00017 float temperature;
00018
00019 int validate_inputs(int argc, char **argv) {
00020     char *cp, *program_name = argv[0];
00021
00022     // skip over program name
00023     --argc;
00024     ++argv;
00025
00026     if (argc < 3) {
00027         fprintf(stderr,"\n WARNING %s: not (all) arguments specified", program_name);
00028         fprintf(stderr,"\n -------> starting server with default value!\n\n", program_name);
00029         return 0;
00030     }
00031
00032     cp = *argv;
00033     if (*cp == 0) {
00034         fprintf(stderr,"\n WARNING %s: starting server with default values!\n", program_name);
00035         return 0;
00036     }
00037
00038     return 1;
00039 }
00040
00041 char * get_data_from_sensor(char *device_name, int dev_addr) {
00042     static unsigned char data[6] = {0};
00043     int i2c_handler, length, comp_temp;
00044     float temperature_celsius;
00045
00046     /* Get I2C handler */
00047     if ((i2c_handler = open(device_name, O_RDWR)) < 0)
00048     {
00049         /* Error getting the file descriptor from device */
00050         printf("Unable to open I2C device");
00051         return data;
00052     }
00053     /* Calling kernel layer from user space to connect with I2C device */
00054     if (ioctl(i2c_handler, I2C_SLAVE, dev_addr) < 0)
00055     {
00056         /* Bus access failed */
00057         printf("Unable to connect to low-level device I2C.\n");
00058         return data;
00059     }
00060
00061     /* Handshake (initialization command) according AHT10 sensor datasheet */
```

```
00062     data[0] = 0xE1;
00063     data[1] = 0x08;
00064     data[2] = 0x00;
00065     length = 3;
00066     /* Send initialization command to I2C device */
00067     if (write(i2c_handler, data, length) != length)
00068     {
00069         /* Unable to handshake with i2c device */
00070         printf("Unable to handshake the i2c bus.\n");
00071     }
00072
00073     /* Wait 20ms before proceeding with the next write command */
00074     sleep(0.02);
00075
00076     /* Trigger measurement (request command) acc. AHT10 sensor datasheet */
00077     data[0] = 0xAC;
00078     data[1] = 0x33;
00079     data[2] = 0x00;
00080     length = 3;
00081     /* Send trigger measurement command to I2C device */
00082     if (write(i2c_handler, data, length) != length)
00083     {
00084         /* Unable to request data from I2C device */
00085         printf("Unable to request data from the I2C bus.\n");
00086     }
00087
00088     /* Wait 20ms before proceeding with the next read command */
00089     sleep(0.02);
00090
00091     /* Read the 6 bytes answer from I2C */
00092     length = 6;
00093     if (read(i2c_handler, data, length) != length)
00094     {
00095         /* Unable to get data from I2C device */
00096         printf("Unable to read output temp. from the i2c AHT10 sensor.\n");
00097     }
00098     else {
00099         return data;
00100     }
00101 }
00102
00103 float compute_temperature_celsius(char *data) {
00104     int comp_temp;
00105     float temperature_celsius;
00106     comp_temp = ((data[3] & 0x0F) << 16) | (data[4] << 8) | data[5];
00107     temperature_celsius = ((comp_temp*200.0)/1048576) -50.0;
00108     return temperature_celsius;
00109 }
00110
00111 int start_listener(unsigned short port) {
00112     int socket_desc;
00113     struct sockaddr_in server;
00114
00115     socket_desc = socket(AF_INET , SOCK_STREAM , 0);
00116     if (socket_desc == -1)
00117     {
00118         printf("Unable to create socket.");
00119     }
00120
00121     server.sin_family = AF_INET;
00122     server.sin_addr.s_addr = INADDR_ANY;
00123     server.sin_port = htons( port );
00124
00125     if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
00126     {
00127         perror("Unable to bind socket.");
00128         return 1;
00129     }
00130
00131     listen(socket_desc , MAX_SIZE);
00132
00133     return socket_desc;
00134 }
00135
00136 void *read_sensor(void *arg) {
00137     SensorInfo *sensor_info = (SensorInfo *)arg;
00138
00139     char *data = get_data_from_sensor(sensor_info->dev_name, sensor_info->dev_address);
00140
00141     pthread_mutex_lock(&mutex);
00142     for (int i = 0; i < 6; ++i) {
00143         sensor_info->data[i] = data[i];
00144     }
00145     data_ready = e_computing;
00146     pthread_cond_signal(&cond);
00147     pthread_mutex_unlock(&mutex);
00148
```

```
00149     return NULL;
00150 }
00151
00152 void *compute_temperature(void *arg) {
00153     SensorInfo *sensor_info = (SensorInfo *)arg;
00154
00155     pthread_mutex_lock(&mutex);
00156     while( data_ready != e_computing ) {
00157         pthread_cond_wait(&cond, &mutex);
00158     }
00159
00160     temperature = compute_temperature_celsius(sensor_info->data);
00161
00162     data_ready = e_ready;
00163     pthread_cond_signal(&cond);
00164     pthread_mutex_unlock(&mutex);
00165     return NULL;
00166 }
00167
00168 void send_message(int client_socket) {
00169     char message[MAX_SIZE];
00170
00171     sprintf(message, "Honeywell ATH10 Temperature Server\n Current temperature: %.2f°C\n",
    temperature);
00172     write(client_socket , message , strlen(message));
00173 }
00174
00175 void *connection_handler(void *socket_desc) {
00176     int client_socket = *(int*)socket_desc;
00177
00178     pthread_mutex_lock(&mutex);
00179     while (data_ready != e_ready) {
00180         pthread_cond_wait(&cond, &mutex);
00181     }
00182     send_message(client_socket);
00183     data_ready = e_gathering;
00184
00185     pthread_cond_signal(&cond);
00186     pthread_mutex_unlock(&mutex);
00187
00188     return NULL;
00189 }
```

5.3 hw_aht10_func.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct SensorInfo

Macros

- #define MAX_SIZE 256

Enumerations

- enum threads_states { e_gathering , e_computing , e_ready }

Functions

- int validate_inputs (int, char **)
- int start_listener (unsigned short)
- char * get_data_from_sensor (char *, int)
- float compute_temperature_celsius (char *)
- void send_message (int)
- void * read_sensor (void *)
- void * compute_temperature (void *)
- void * connection_handler (void *)

Variables

- SensorInfo sensor_info
- pthread_mutex_t mutex
- pthread_cond_t cond

5.3.1 Macro Definition Documentation

5.3.1.1 MAX_SIZE

```
#define MAX_SIZE 256
```

Definition at line 4 of file hw_aht10_func.h.

5.3.2 Enumeration Type Documentation

5.3.2.1 threads_states

```
enum threads_states
```

Enumerator

e_gathering	
e_computing	
e_ready	

Definition at line 16 of file hw_aht10_func.h.

5.3.3 Function Documentation

5.3.3.1 compute_temperature()

```
void * compute_temperature (
            void * arg)
```

Definition at line 152 of file hw_aht10_func.c.

5.3.3.2 compute_temperature_celsius()

```
float compute_temperature_celsius (
            char * data)
```

Definition at line 103 of file hw_aht10_func.c.

5.3.3.3 connection_handler()

```
void * connection_handler (
            void * socket_desc)
```

Definition at line 175 of file hw_aht10_func.c.

5.3.3.4 get_data_from_sensor()

```
char * get_data_from_sensor (
            char * device_name,
            int dev_addr)
```

Definition at line 41 of file hw_aht10_func.c.

5.3.3.5 read_sensor()

```
void * read_sensor (
            void * arg)
```

Definition at line 136 of file hw_aht10_func.c.

5.3.3.6 send_message()

```
void send_message (
            int client_socket)
```

Definition at line 168 of file hw_aht10_func.c.

5.3.3.7 start_listener()

```
int start_listener (
            unsigned short port)
```

Definition at line 111 of file hw_aht10_func.c.

5.3.3.8 validate_inputs()

```
int validate_inputs (
            int argc,
            char ** argv)
```

Definition at line 19 of file hw_aht10_func.c.

5.3.4 Variable Documentation

5.3.4.1 cond

```
pthread_cond_t cond  [extern]
```

Condition variable for thread synchronization (prevents race conditions)
Definition at line 15 of file hw_aht10_func.c.

5.3.4.2 mutex

```
pthread_mutex_t mutex  [extern]
```

Mutex for thread synchronization (prevents deadlocks)
Definition at line 14 of file hw_aht10_func.c.

5.3.4.3 sensor_info

```
SensorInfo sensor_info  [extern]
```

Sensor information structure
Definition at line 12 of file hw_aht10_temperature_server.c.

5.4 hw_aht10_func.h

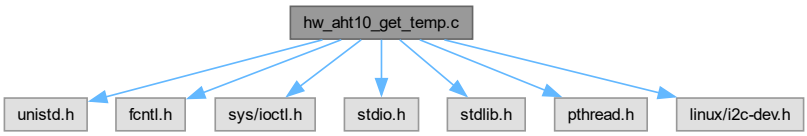
Go to the documentation of this file.

```
00001 #ifndef HONEYWELL_ASSESSMENT_HW_AHT10_FUNC_H
00002 #define HONEYWELL_ASSESSMENT_HW_AHT10_FUNC_H
00003
00004 #define MAX_SIZE 256
00005
00006 typedef struct {
00007     char dev_name[MAX_SIZE];
00008     int dev_address;
00009     char data[6];
00010 } SensorInfo;
00011
00012 extern SensorInfo sensor_info;
00013 extern pthread_mutex_t mutex;
00014 extern pthread_cond_t cond;
00015
00016 typedef enum  {e_gathering, e_computing, e_ready} threads_states;
00017
00018 int validate_inputs(int, char **);
00019 int start_listener(unsigned short);
00020 char * get_data_from_sensor(char *, int);
00021 float compute_temperature_celsius(char *);
00022 void send_message(int);
00023 void *read_sensor(void *);
00024 void *compute_temperature(void *);
00025 void *connection_handler(void *);
00026
00027 #endif //HONEYWELL_ASSESSMENT_HW_AHT10_FUNC_H
```

5.5 hw_aht10_get_temp.c File Reference

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <linux/i2c-dev.h>
```

Include dependency graph for hw_aht10_get_temp.c:



Functions

- int validate_inputs (int, char **)
- float get_temperature_celsius (char *, int)
- int main (int argc, char **argv)

5.5.1 Function Documentation

5.5.1.1 get_temperature_celsius()

```
float get_temperature_celsius (
            char * device_name,
            int dev_addr)
```

Definition at line 47 of file hw_aht10_get_temp.c.

5.5.1.2 main()

```
int main (
            int argc,
            char ** argv)
```

Definition at line 12 of file hw_aht10_get_temp.c.

5.5.1.3 validate_inputs()

```
int validate_inputs (
            int argc,
            char ** argv)
```

Definition at line 26 of file hw_aht10_get_temp.c.

5.6 hw_aht10_get_temp.c

Go to the documentation of this file.

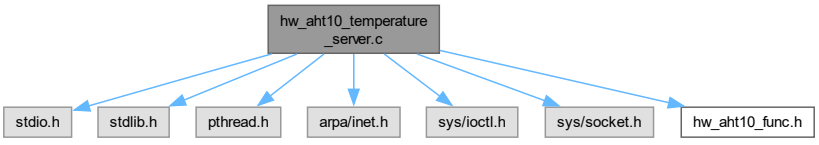
```
00001 #include <unistd.h>
00002 #include <fcntl.h>
00003 #include <sys/ioctl.h>
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <pthread.h>
00007 #include <linux/i2c-dev.h>
00008
00009 int validate_inputs(int, char **);
00010 float get_temperature_celsius(char *, int);
00011
00012 int main(int argc, char **argv) {
00013     /*char *device_name = (char*)"/dev/i2c-1";
00014     int dev_addr = 0x38;*/
00015     if( validate_inputs(argc, argv) == 1) {
00016         char *device_name = argv[1];
00017         int dev_addr = (int)strtol(argv[2], NULL, 16);
00018     printf("%0.2f\n", get_temperature_celsius(device_name, dev_addr));
00019     }
00020     else {
00021         printf("\n Usage: %s device_name device address\n\n", argv[0]);
00022     }
00023     return 0;
00024 }
00025
00026 int validate_inputs(int argc, char **argv) {
00027     char *cp, *program_name = argv[0];
00028
00029     // skip over program name
00030     --argc;
00031     ++argv;
00032
00033     if (argc < 2) {
00034         fprintf(stderr,"\n %s: not (all) arguments specified\n", program_name);
00035         return -1;
00036     }
00037
00038     cp = *argv;
00039     if (*cp == 0) {
00040         fprintf(stderr,"\n %s: argument an empty string\n", program_name);
00041         return -1;
00042     }
00043
00044     return 1;
00045 }
00046
00047 float get_temperature_celsius(char *device_name, int dev_addr) {
00048     unsigned char data[6] = {0};
00049     int i2c_handler, length, comp_temp;
00050     float temperature_celsius;
00051
00052     /* Get I2C handler */
00053     if ((i2c_handler = open(device_name, O_RDWR)) < 0)
00054     {
00055         /* Error getting the file descriptor from device */
00056         printf("Unable to open I2C device");
00057         return -1;
00058     }
00059     /* Calling kernel layer from user space to connect with I2C device */
00060     if (ioctl(i2c_handler, I2C_SLAVE, dev_addr) < 0)
00061     {
00062         /* Bus access failed */
```

```
00063         printf("Unable to connect to low-level device I2C.\n");
00064         return -1;
00065     }
00066 
00067     /* Handshake (initialization command) according AHT10 sensor datasheet */
00068     data[0] = 0xE1;
00069     data[1] = 0x08;
00070     data[2] = 0x00;
00071     length = 3;
00072     /* Send initialization command to I2C device */
00073     if (write(i2c_handler, data, length) != length)
00074     {
00075         /* Unable to handshake with i2c device */
00076         printf("Unable to handshake the i2c bus.\n");
00077     }
00078 
00079     /* Wait 20ms before proceeding with the next write command */
00080     sleep(0.02);
00081 
00082     /* Trigger measurement (request command) acc. AHT10 sensor datasheet */
00083     data[0] = 0xAC;
00084     data[1] = 0x33;
00085     data[2] = 0x00;
00086     length = 3;
00087     /* Send trigger measurement command to I2C device */
00088     if (write(i2c_handler, data, length) != length)
00089     {
00090         /* Unable to request data from I2C device */
00091         printf("Unable to request data from the I2C bus.\n");
00092     }
00093 
00094     /* Wait 20ms before proceeding with the next read command */
00095     sleep(0.02);
00096 
00097     /* Read the 6 bytes answer from I2C */
00098     length = 6;
00099     if (read(i2c_handler, data, length) != length)
00100     {
00101         /* Unable to get data from I2C device */
00102         printf("Unable to read output temp. from the i2c AHT10 sensor.\n");
00103     }
00104     else
00105     {
00106         comp_temp = ((data[3] & 0x0F) << 16) | (data[4] << 8) | data[5];
00107         temperature_celsius = ((comp_temp*200.0)/1048576) -50.0;
00108     }
00109     return temperature_celsius;
00110 }
```

5.7 hw_aht10_temperature_server.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include "hw_aht10_func.h"
```

Include dependency graph for hw_aht10_temperature_server.c:



Functions

- int main (int argc, char **argv)

In this section the TCP server binds connections with clients using threads for:
.

Variables

- pthread_mutex_t mutex
- pthread_cond_t cond
- SensorInfo sensor_info

5.7.1 Function Documentation

5.7.1.1 main()

```
int main (
            int argc,
            char ** argv)
```

In this section the TCP server binds connections with clients using threads for:
.

- 1) Getting data from AHT10 sensor
- 2) Computing the current temperature, and
- 3) Attending new temperature requests via TCP.

Parameters

argc	Argument count
argv	Argument vector

Returns

Exit status

TCP port is 5000 by default
Default values for I2C hardware:

- The device name by default is the file descriptor we found at /dev/i2c-1 and,
- The device address for the AHT10 sensor is 0x38.

NOTE: These values can be modified according to the connected sensor and the board when executing the program via the parameter *argv*.
Main loop:

- Once a client connection is accepted, three threads are created as mentioned above including the mutex and cond sections for **avoiding race conditions and/or deadlocks**.

Definition at line 26 of file hw_aht10_temperature_server.c.

5.7.2 Variable Documentation

5.7.2.1 cond

```
pthread_cond_t cond
```

Condition variable for thread synchronization (prevents race conditions)
Definition at line 11 of file hw_aht10_temperature_server.c.

5.7.2.2 mutex

```
pthread_mutex_t mutex
```

Mutex for thread synchronization (prevents deadlocks)
Definition at line 10 of file hw_aht10_temperature_server.c.

5.7.2.3 sensor_info

```
SensorInfo sensor_info
```

Sensor information structure
Definition at line 12 of file hw_aht10_temperature_server.c.

5.8 hw_aht10_temperature_server.c

Go to the documentation of this file.

```
00001
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <pthread.h>
00005 #include <arpa/inet.h>
00006 #include <sys/ioctl.h>
00007 #include <sys/socket.h>
00008 #include "hw_aht10_func.h"
00009
00010 pthread_mutex_t mutex;
00011 pthread_cond_t cond;
00012 SensorInfo sensor_info;
00026 int main(int argc, char **argv) {
00027     int socket_desc, client_sock;
00029     unsigned short server_port = 5000;
00030     struct sockaddr_in client_addr;
00031     socklen_t client_len;
00032
00041     sprintf(sensor_info.dev_name,"/dev/i2c-1");
00042     sensor_info.dev_address = 0x38;
00043
00044     if (validate_inputs(argc, argv) == 1) {
00045         sprintf(sensor_info.dev_name,"%s", argv[1]);
00046         sensor_info.dev_address = (int)strtol(argv[2], NULL, 16);
00047         server_port = (int)strtol(argv[3], NULL, 10);
00048     }
00049
00050     if ((socket_desc = start_listener(server_port)) < 0) {
00051         printf("Unable to open socket at %d", server_port);
00052         exit(-1);
00053     }
00060     while ((client_sock = accept(socket_desc,
00061                                  (struct sockaddr *)&client_addr,
00062                                  (socklen_t*)&client_len))) {
00063         pthread_t thread1, thread2, thread3;
00064
00065         pthread_mutex_init(&mutex, NULL);
00066         pthread_cond_init(&cond, NULL);
00067
00068         pthread_create(&thread1, NULL, read_sensor, &sensor_info);
00069         pthread_create(&thread2, NULL, compute_temperature, &sensor_info);
00070         pthread_create(&thread3, NULL, connection_handler, &client_sock);
00071
00072         pthread_join(thread1, NULL);
00073         pthread_join(thread2, NULL);
00074         pthread_join(thread3, NULL);
00075
00076         pthread_mutex_destroy(&mutex);
00077         pthread_cond_destroy(&cond);
00078     }
00079     return 0;
00080 }
```

5.9 mainpage.dox File Reference

Index