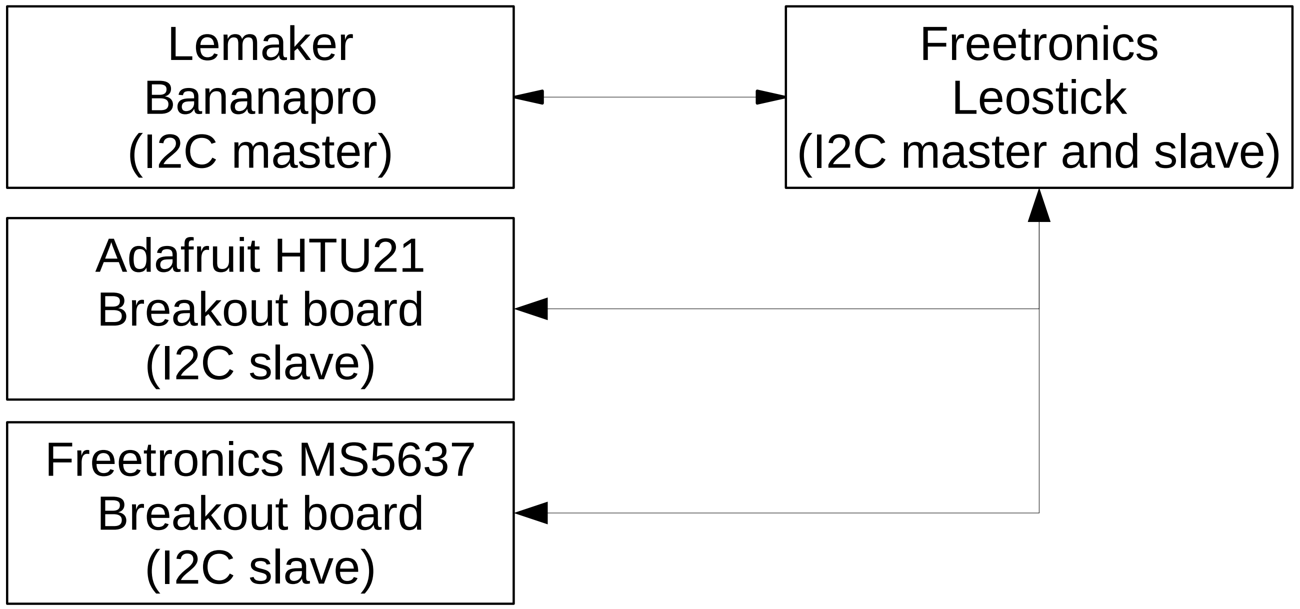
Embedded Linux and the Industrial I/O subsystem

This presentation will discuss accessing data from sensors using the Linux Kernel’s Industrial I/O subsystem on Arm SBCs (single board computers).

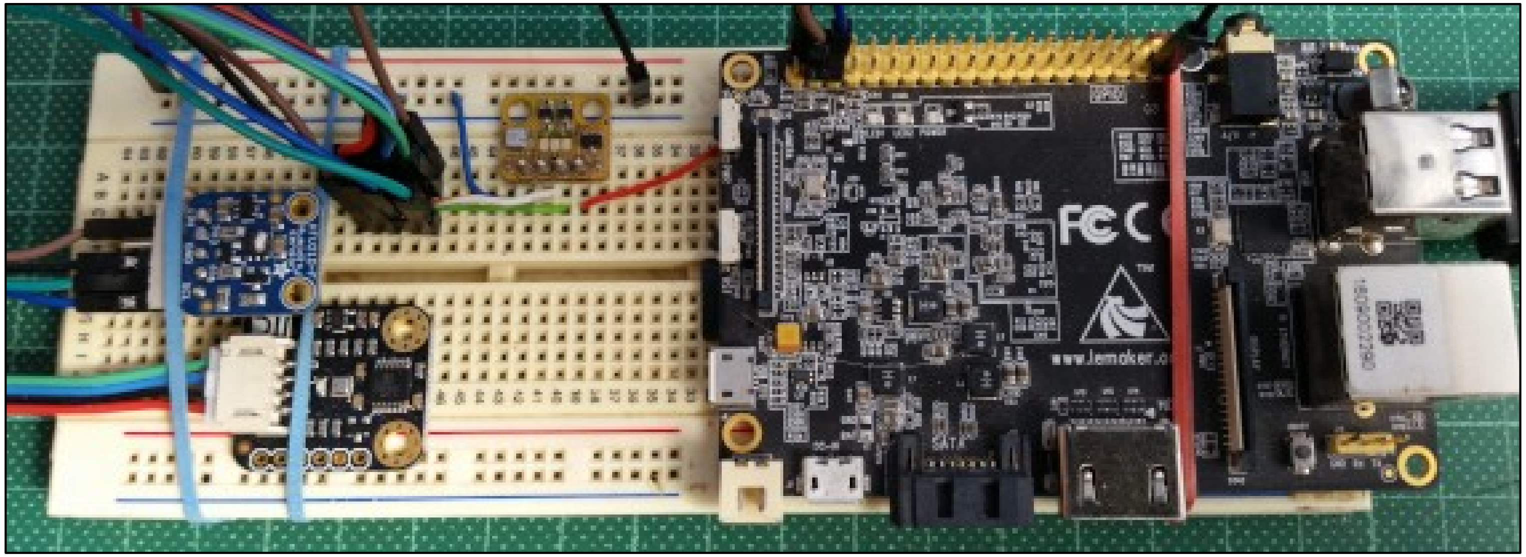
Melbourne Linux Users Group, aka MLUG
Presented by: Rick Miles
February 24 2020

My current weather station is set up with A Bananapro polling data via I2C from a Leostick, an arduino clone, which in turn polls two breakout boards via I2C.



I was never completely satisfied with this setup but have used it for years not realising that the Linux kernel provided the **Industrial I/O subsystem** which allows for simple direct interface with **I2C** sensors.

- The Industrial I/O (IIO) subsystem provides easy implementation of drivers for sensors.
- It is a subsystem for accessing Analog to Digital Converters (ADCs) such as temperature sensors, voltage sensors and light sensors.
- It can also be used to access Digital to Analog Converters (DACs)
- IIO devices can accessed on either the I2C or SPI bus



There are three breakout boards with sensor chips connected to this Bananapro with Slackwarearm installed. From left to right:

- an Adafruit humidity and temperature sensor with a `htu21` chip
- a Gravity air pressure, temperature and humidity sensor with a `bme280` chip
- a Freetronics air pressure and temperature sensor with a `ms5637` chip

The htu21, bme280 and ms5637 are compiled as IIO devices. IIO device driver source is found in the kernel source in the directory /usr/src/linux/drivers/iio with the source grouped in directories by category. For example the htu21 is a humidity sensor that can also provide air temperature data.

```
/usr/src/linux/drivers/iio/humidity/htu21.c
```

To compile the htu21 module:

```
Device Drivers  --->
 Industrial I/O support  --->
   Humidity sensors  --->
   <M> Measurement Specialties HTU21 humidity & temperature sensor
```

I2c devices are not detected by the kernel and loaded at boot but with i2c-tools installed we can use i2cdetect to find their hexidecimal addresses on the i2c bus on /dev/i2c1.

```
root@bpro8: i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 77
```

In order for an I2c device module to be loaded at boot the device is described in a subnode of the DTB. Slackware boots with the mainline kernel DTB for the Bananapro.

- A DTB (device tree blob) is a database that represents the hardware components and devices on a given board that are loaded at boot. A DTB is compiled from a DTS (device tree source) file.

- There are also DTBOs (device tree blob overlay). DTBOs are used for loading devices not described in the DTB. Overlays will be discussed later in this presentation.

- A DTB or DTBO is compiled from a DTS file using the DTC (device tree compiler). DTC is included in the kernel source but it can also be installed as a stand alone program to compile and decompile DTB’s outside the kernel source tree.

- DTC is architecture agnostic. You can compile a DTB for an Arm SOC on an X86 machine.

Device tree bindings are the specifications for how a device is to be described. They are found in usr/src/linux/Documentation/devicetree/bindings. Here is the device tree binding for the htu21.

```
*HTU21 - Measurement-Specialties htu21 temperature & humidity
sensor and humidity part of MS8607 sensor

Required properties:

    - compatible: should be "meas,htu21" or "meas,ms8607-humidity"
    - reg: I2C address of the sensor

Example:

htu21@40 {
    compatible = "meas,htu21";
    reg = <0x40>;
};
```

Slackwarearm does not use DTB overlays. All devices are described in the mainline sun7i-a20-bananpro.dts which is in the directory /usr/src/linux/arch/arm/boot/dts. I will add three subnodes for the three sensors to the subnode &12c2 which is /dev/i2c1 on the Bananapro.

```
<snipped>
&i2c0 {
        status = "okay";
        axp209: pmic@34 {
                compatible = "x-powers,axp209";
                reg = <0x34>;
                interrupt-parent = <&nmi_intc>;
                interrupts = <0 IRQ_TYPE_LEVEL_LOW>;
                interrupt-controller;
                #interrupt-cells = <1>;
        };
};
&i2c2 {
        status = "okay";
};
<snipped>
```

```
<snipped>
&i2c2 {
       status = "okay";

        pressure@77 {
        compatible = "bosch,bme280";
       reg = <0x77>;
       };

       htu21@40 {
       compatible = "meas,htu21";
       reg = < 0x40 >;
       };

       ms5637@76 {
       compatible = "meas,ms5637";
       reg = < 0x76 >;
       };
};
<snipped>
```

After saving sun7i-a20-bananpro.dts I cd to the linux source root directory, compile the new sun7i-a20-bananpro.dts, copy it to /boot/dtb and reboot.

```
root@bpro8: make sun7i-a20-bananapro.dtb
  DTC     arch/arm/boot/dts/sun7i-a20-bananapro.dtb
root@bpro8: cp arch/arm/boot/dts/sun7i-a20-bananapro.dtb /boot/dtb
root@bpro8: reboot
```

After the reboot I can run i2cdetect -y 1 again and confirm that the three sensors have now been loaded as devices

```
root@bpro8: i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- UU UU
```

Upon boot a pseudo /sys/bus/iio directory is created and the devices can be accessed from the /sys/bus/iio/devices directory.

I have highlighted the hexidecimal address of each sensor . Note each sensor has an IIO device number, e.g. 0077/iio:device2

```
sys/bus/iio
|-- devices
|   |-- iio:device0
../../../devices/platform/soc/1c2b400.i2c/i2c-1/1-0076/iio:device0

../../../devices/platform/soc/1c2b400.i2c/i2c-1/1-0040/iio:device1

../../../devices/platform/soc/1c2b400.i2c/i2c-1/1-0077/iio:device2
```

Real time data from the bme280 sensor can be accessed from files in /sys/bus/iio/devices/iio\:device2.

```
root@bpro8: tree /sys/bus/iio/devices/iio\:device2
/sys/bus/iio/devices/iio:device2
|-- dev
|-- in_humidityrelative_input
|-- in_humidityrelative_oversampling_ratio
|-- in_pressure_input
|-- in_pressure_oversampling_ratio
|-- in_pressure_oversampling_ratio_available
|-- in_temp_input
|-- in_temp_oversampling_ratio
|-- in_temp_oversampling_ratio_available
<snipped>
```

```
root@bpro8: cat /sys/bus/iio/devices/iio\:device2/in_humidityrelative_input
55944
root@bpro8: cat /sys/bus/iio/devices/iio\:device2/in_pressure_input
99.660562500
root@bpro8: cat /sys/bus/iio/devices/iio\:device2/in_temp_input
24410
```

To understand the values returned from the bme280 we can check out the Industrial IO ABI in /usr/src/linux/Documentation/ABI/testing/sysfs-bus-iio.

```
What:           /sys/bus/iio/devices/iio:deviceX/in_temp_input
<snipped>
Description:
                Scaled temperature measurement in milli degrees Celsius.


What:           /sys/bus/iio/devices/iio:deviceX/in_pressure_input
<snipped>
Description:
                Scaled pressure measurement from channel Y, in kilopascal.


What:           /sys/bus/iio/devices/iio:deviceX/in_humidityrelative_input
<snipped>
Description:
                Scaled humidity measurement in milli percent.
```

The Industrial I/O ABI ensures that if it is an IIO device, it must provide data in a file with a given filename and in a given scale.

Knowing the scale in which the data is returned allows for simple conversion to understandable measurements

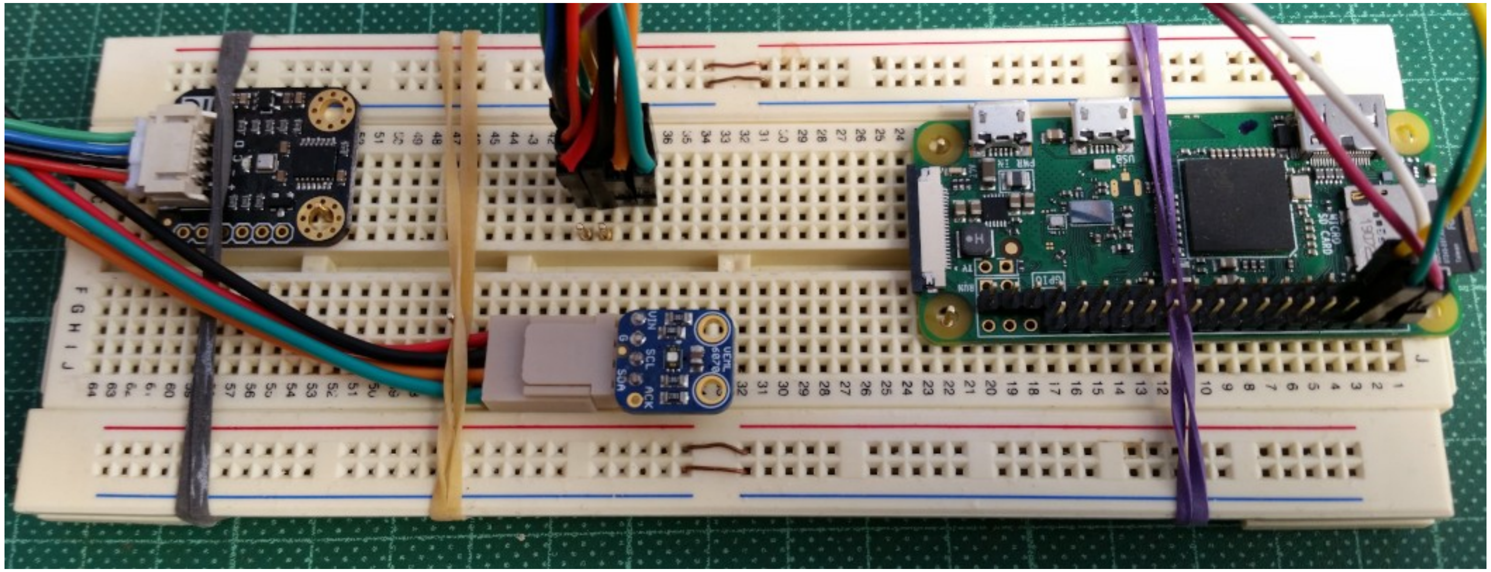
```
in_humidityrelative_input = 55944*1000 = 55.944% relative humidity
in_pressure_input = 99.660562500*10 = 996.605625 local hPa
in_temp_input = 24410/1000 = 24.41 degrees Celsius
```

It is obvious that a Bash script could easily manage accessing data from an IIO sensor and doing something with it but before I move on to DTB overlays here is a short C program that returns a temperature from the bme280.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *in_file;
    //  Get temperature //
    float air_temp;
    in_file = fopen("/sys/bus/iio/devices/iio:device2/in_temp_input", "r");
    fscanf(in_file,"%f", &air_temp);
    air_temp = air_temp / 1000;;
    printf("\t\nThe temperature is: %.1\n", air_temp);
    fclose(in_file);
}
```

```
root@bpro8: ./get-temp

The temperature is: 24.2C
```



There are two breakout boards with sensor chips connected to this Raspberry Pi Zero W with Raspbian Buster installed. From left to right:

- a Gravity air pressure, temperature and humidity sensor board with a bme280 chip
- An Adafruit uv sensor board with a veml6070 chip

The Raspberry Pi Zero W boots with the non-mainline bcm2708-rpi-zero-w.dtb which will boot a minimal system. The file /boot/config.txt is read at boot and any overlay listed in or uncommented in /boot/config.txt will be appended to bcm2708-rpi-zero-w.dtb.

For example to enable I2C or SPI the corresponding line below would be uncommented

```
# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on
```

Raspbian ships with 186 overlays located in the directory `/boot/overlays`. While an overlay for the bme280 or veml6070 could be written as a DTS, compiled as a DTB and copied into /boot/overlays. However, the Raspbian overlay `i2c-sensors.dtbo` enables I2C and contains fragments for both devices.

Enabling I2C and loading those two devices simply involves appending either one or two lines to /boot/config.txt

```
dtoverlay=i2c-sensor,bme280,veml6070
```

```
dtoverlay=i2c-sensor,bme280
dtoverlay=i2c-sensor,veml6070
```

Fragments describe functional changes to the device tree. This is an overlay fragment describing the bme280 device. Note that this fragment contains the same subnode that was added to the sun7i-a2-bananapro.dtb.

```
fragment@0 {
                target = < 0xffffffff >;

                __dormant__ {
                        #address-cells = < 0x01 >;
                        #size-cells = < 0x00 >;
                        status = "okay";

                        bme280@77 {
                                compatible = "bosch,bme280";
                                reg = < 0x77 >;
                        };
                };
        };
```

After boot we can see that the bme280 and veml6070 have been loaded as devices on the I2C bus. The veml6070 loads at two addresses but only 0x38 is used to access data.

```
root@rpi0-7: i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- UU UU -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- UU
```

```
/sys/bus/iio
|-- devices
../../../devices/platform/soc/20804000.i2c/i2c-1/1-0038/iio:device0

../../../devices/platform/soc/20804000.i2c/i2c-1/1-0077/iio:device1
```

There are two files listed for the veml6070 that will return values.

```
root@rpi0-7: tree /sys/bus/iio/devices/iio:device0
/sys/bus/iio/devices/iio:device0
|-- dev
|-- in_intensity_uv_raw
|-- in_uvindex_input
|-- name
<snipped>
```

```
root@rpi0-7:  cat /sys/bus/iio/devices/iio:device0/in_intensity_uv_raw
1094

root@rpi0-7: cat /sys/bus/iio/devices/iio:device0/in_uvindex_input
5
```

We can find what the values returned from the veml6070 represent in the sys-bus-iio ABI.

```
What:        /sys/.../iio:deviceX/in_intensity_uv_raw
<snipped>
Description:
        Unit-less light intensity. <snipped>. Modifier
        uv indicates that measurements contains ultraviolet light
        components. <snipped>
.
What:        /sys/.../iio:deviceX/in_uvindex_input
<snipped>
Description:
        UV light intensity index measuring the human skin's response to
        different wavelength of sunlight weighted according to the
        standardised CIE Erythemal Action Spectrum. UV index values range
        from 0 (low) to >=11 (extreme).
```

The the C program on the following slide provides an example of how data from the veml6070 device can be used.

```
root@rpi0-7: ./get-uv

The raw UV level is: 1094, The UV Index is: Moderate
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE *in_file;
  int uv_index;
  int uv_raw;
  // Get raw uv
  in_file = fopen("/sys/bus/iio/devices/iio:device0/in_intensity_uv_raw","r");
  fscanf(in_file,"%d", &uv_raw);
  fclose(in_file);
  printf("The raw UV level is: %d, The UV Index is: ", uv_raw);
  // Get UV index //
  in_file = fopen("/sys/bus/iio/devices/iio:device0/in_uvindex_input", "r");
  fscanf(in_file,"%d", &uv_index);
  fclose(in_file);
  if (uv_index <= 2) printf("LOW");
  if (uv_index > 2 && uv_index <=5) printf("Moderate");
  if (uv_index > 5 && uv_index <=7) printf("High");
  if (uv_index > 7 && uv_index <= 10) printf("Very High");
  if (uv_index >= 11) printf("Extreme");
 }
```

The Industrial I/O Linux subsystem offers a unified framework to communicate (read and write) with drivers covering many different types of embedded sensors and a few actuators. It also offers a standard interface to user space applications manipulating sensors through sysfs and devfs. [1]

The aim is to fill the gap between the somewhat similar hwmon and input subsystems. Hwmon is directed at low sample rate sensors used to monitor and control the system itself, like fan speed control or temperature measurement. Input is, as its name suggests, focused on human interaction input devices (keyboard, mouse, touchscreen) [2].

This presentation's focus was to provide some practical examples of using IIO devices.

If you are interested there is much more information about the Industrial I/O subsystem available on the net such as the two links footnoted below.

[1] *https://wiki.st.com/stm32mpu/wiki/IIO_overview*
[2] *https://www.kernel.org/doc/html/v4.14/driver-api/iio/intro.html*

Any questions?

Appendix 1

I have attached a C program on the following slides that I will adapt for my next weather station setup. It prints a line of data in .csv format that can be parsed for specific values when updating my local weather webpage

```
root@rpi0-7: ./get-w-data
24.6,61.0,16.6,25.7,1018.9,1049,Medium
```

Feel free to use it and/or adapt it to suit your needs. Note that you may have problems on a Raspberry Pi. Device numbers might change from one boot to the next. This seems to be a problem that other folks have also had with Raspberry Pi’s. This wasn’t a problem on my Bananapro running Slackware. If you run the program and get a segmentation fault your device numbers have most likely changed.

```
// get-w-data rm20200215
// Tested with bme280 and veml6070 sensors.
// Prints weather data in .csv format.
// Compile with 'gcc -lm' to link the math library.
// The variable 'elv_adjst' is the difference in hPa between my
// home and MSL (mean sea level) hPa.
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
const float elv_adjst = 8.77;
int main()
{
    FILE *in_file;
    //  Get temperature, bme280 is device1
    float air_temp;
    in_file = fopen("/sys/bus/iio/devices/iio:device1/in_temp_input", "r");
    fscanf(in_file,"%f", &air_temp);
    air_temp = air_temp / 1000;;
    fclose(in_file);

    // Get relative humidity, bme280 is device1
    float rel_hum;
    in_file = fopen("/sys/bus/iio/devices/iio:device1/in_humidityrelative_input", "r");
     fscanf(in_file,"%f", &rel_hum);
    rel_hum = rel_hum / 1000;
    fclose(in_file);
```

```
 // Get air pressure and convert to MSL hPa, bme280 is device1
float air_prssr;
in_file =        fopen("/sys/bus/iio/devices/iio:device1/in_pressure_input", "r");
fscanf(in_file,"%f", &air_prssr);
air_prssr = air_prssr * 10 + elv_adjst;
fclose(in_file);

 // Calculate dew point
float dew_pt = powf((rel_hum/100), (1.0/8.0))
*(0.9*air_temp +112)
+(0.1*air_temp)-112;

 // Calculate heat index.
float ht_ndx =-8.784695 +
1.61139411 * air_temp +
2.33854900 * rel_hum +
-0.14611605 * air_temp*rel_hum +
-0.01230809 * powf(air_temp, 2) +
-0.01642482 * powf(rel_hum, 2) +
0.00221173 * powf(air_temp, 2) * rel_hum +
0.00072546 * air_temp * pow(rel_hum, 2) +
-0.00000358 * pow(air_temp, 2) * pow(rel_hum, 2);
```

```
    // Get raw uv, good for graphing, veml6070 is device0
    int uv_raw;
    in_file = fopen("/sys/bus/iio/devices/iio:device0/in_intensity_uv_raw", "r");
    fscanf(in_file,"%d", &uv_raw);
    fclose(in_file);

    // Get UV index as number, veml6070 is device0
    int uv_index;
    in_file = fopen("/sys/bus/iio/devices/iio:device0/in_uvindex_input", "r");
    fscanf(in_file,"%d", &uv_index);
    fclose(in_file);

    // Print data in .csv format
    printf("%0.1f,%0.1f,%0.1f,%0.1f,%0.1f,%d,",
    air_temp, rel_hum, dew_pt, ht_ndx, air_prssr, uv_raw);

    // Get UV index and append to the line of data
    if (uv_index <= 2) printf("LOW");
    if (uv_index > 2 && uv_index <=5) printf("Moderate");
    if (uv_index > 5 && uv_index <=7) printf("High");
    if (uv_index > 7 && uv_index <= 10) printf("Very High");
    if (uv_index >= 11) printf("Extreme");

return 0;
}
```