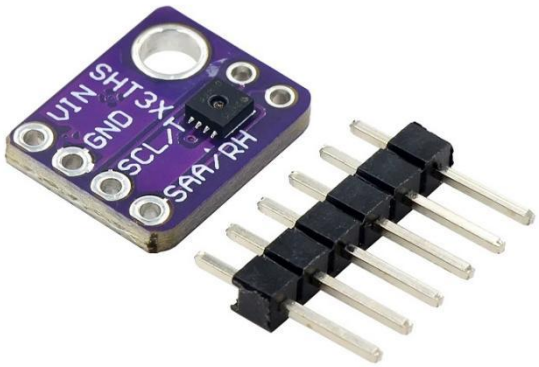
1. Tổng quan về cảm biến GY-SHT30-D

Cảm Biến Nhiệt Độ Độ Ẩm Không Khí GY-SHT30-D Temperature Humidity Sensor sử dụng cảm biến SHT30, SHT30 rất dễ bắt gặp trong các thiết bị nhà thông minh, cảm biến thuộc dòng trung cấp của hãng Sensirion với độ chính xác cao, sai số của độ ẩm là 2% và của nhiệt độ là 0.2℃.

Thông số kỹ thuật:

- Điện áp: 2.15 ~ 5.5VDC.
- Dải đo độ ẩm: 0 - 100% RH.
- Sai số độ ẩm: ± 2% RH.
- Dải đo nhiệt độ: -40 ~ 125℃.
- Sai số nhiệt độ: ± 0.2℃ (tại 0 ~ 90℃)
- Tín hiệu ngõ ra: I2C
- Kích thước: 11 x 13mm



2. Porting GY-SHT30-D cho BeagleBone Black(BBB)

Cảm biến GY-SHT30-D sử dụng giao thức I2C để gửi nhận dữ liệu nên việc đầu tiên trong quá trình porting là xác định bus I2C và chân pins của BBB sẽ sử dụng

2.1. Xác định bus I2C và chân (pins) của BBB sẽ sử dụng

Mục tiêu của bước này là kiểm tra các bus I2C và các chân còn trống có thể sử dụng. Để làm được điều này, ta cần xem bảng pin-muxing và sơ đồ mạch (schematic) của BBB.

Dựa vào tài liệu pin-muxing table ta thấy rằng hai chân P9_24 và P9_26 có thể dùng cho bus I2C1 nếu được cấu hình ở mode 3

P9 Pin	Offset	mode0	mode1	mode2	mode3	mode4
P9_1	GND					
P9_2	GND					
P9_3	3.3V					
P9_4	3.3V					
P9_5	VDD_5V					
P9_6	VDD_5V					
P9_7	SYS_5V					
P9_8	SYS_5V					
P9_9	PWR_BUT					
P9_10		Reset_Out				
P9_11	0x870	gpmc_wait0	gmii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd
P9_12	0x878	gpmc_be1n	gmii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir
P9_13	0x874	gpmc_wpn	gmii2_rxerr	gpmc_csn5	rmii2_rxerr	mmc2_sdcd
P9_14	0x848	gpmc_a2	gmii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18
P9_15	0x840	gpmc_a0	gmii2_txen	rgmii2_tctl	rmii2_txen	gpmc_a16
P9_16	0x84C	gpmc_a3	gmii2_txd2	rgmii2_td2	mmc2_dat2	gpmc_a19
P9_17	0x95C	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci	pr1_uart0_txd
P9_18	0x958	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone_input	pr1_uart0_rxd
P9_19	0x97C	uart1_rtsn	timer5	dcan0_rx	I2C2_SCL	spi1_cs1
P9_20	0x978	uart1_ctsn	timer6	dcan0_tx	I2C2_SDA	spi1_cs0
P9_21	0x954	spi0_d0	uart2_txd	I2C2_SCL	ehrpwm0B	pr1_uart0_rts_n
P9_22	0x950	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A	pr1_uart0_cts_n
P9_23	0x844	gpmc_a1	gmii2_rxdv	rgmii2_rctl	mmc2_dat0	gpmc_a17
P9_24	0x984	uart1_txd	mmc2_sdwp	dcan1_rx	I2C1_SCL	
P9_25	0x9AC	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4
P9_26	0x980	uart1_rxd	mmc1_sdwp	dcan1_tx	I2C1_SDA	
P9_27	0x9A4	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU2
P9_28	0x99C	mcasp0_ahclkr	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out

Tiếp tục, dựa vào tài liệu schematic ta thấy rằng chân P9_24 và P9_26 không được kết nối vào bất kỳ thiết bị nào. Vì vậy ta có thể sử dụng chân P9_24 và P9_26 để kết nối với GY-SHT30-D

Sơ đồ kết nối GY-SHT30-D với BBB

BBB	GY-SHT30-D
3.3V	VCC
GND	GND
P9_24	SCL
P9_26	SDA

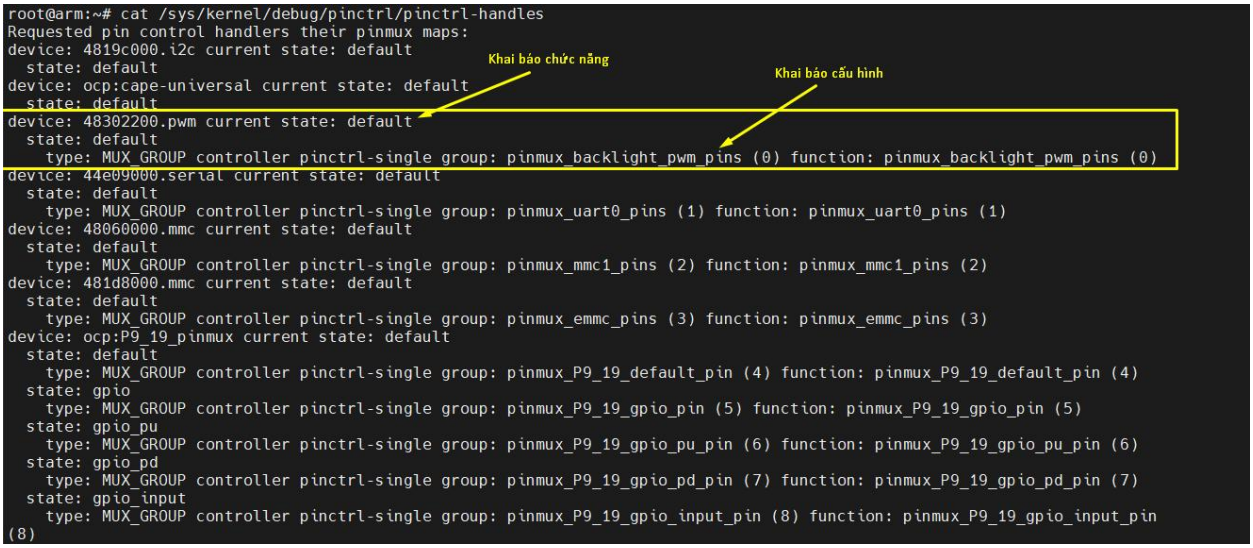
Tiếp theo, ta cần cấu hình pinctrl cho chân này trong device tree

2.2. Cấu hình pinctrl trong device tree

Mục tiêu của bước này là cấu hình pinctrl cho chân P9_24 và P9_26 trong device tree hoạt động ở mode 3.

Trước khi cấu hình, chúng ta cần kiểm tra xem 2 chân này đã được cấu hình chưa bằng cách đọc nội dung của file `pinctrl-handles` bằng command

‘cat /sys/kernel/debug/pinctrl/pinctrl-handles’




```
root@arm:~# cat /sys/kernel/debug/pinctrl/pinctrl-handles
Requested pin control handlers their pinmux maps:
device: 4819c000.i2c current state: default
  state: default
device: ocp:cape-universal current state: default
  state: default
device: 48302200.pwm current state: default
  state: default
    type: MUX_GROUP controller pinctrl-single group: pinmux_backlight_pwm_pins (0) function: pinmux_backlight_pwm_pins (0)
device: 44e09000.serial current state: default
  state: default
    type: MUX_GROUP controller pinctrl-single group: pinmux_uart0_pins (1) function: pinmux_uart0_pins (1)
device: 48060000.mmc current state: default
  state: default
    type: MUX_GROUP controller pinctrl-single group: pinmux_mmc1_pins (2) function: pinmux_mmc1_pins (2)
device: 481d8000.mmc current state: default
  state: default
    type: MUX_GROUP controller pinctrl-single group: pinmux_emmc_pins (3) function: pinmux_emmc_pins (3)
device: ocp:P9_19_pinmux current state: default
  state: default
    type: MUX_GROUP controller pinctrl-single group: pinmux_P9_19_default_pin (4) function: pinmux_P9_19_default_pin (4)
  state: gpio
    type: MUX_GROUP controller pinctrl-single group: pinmux_P9_19_gpio_pin (5) function: pinmux_P9_19_gpio_pin (5)
  state: gpio_pu
    type: MUX_GROUP controller pinctrl-single group: pinmux_P9_19_gpio_pu_pin (6) function: pinmux_P9_19_gpio_pu_pin (6)
  state: gpio_pd
    type: MUX_GROUP controller pinctrl-single group: pinmux_P9_19_gpio_pd_pin (7) function: pinmux_P9_19_gpio_pd_pin (7)
  state: gpio_input
    type: MUX_GROUP controller pinctrl-single group: pinmux_P9_19_gpio_input_pin (8) function: pinmux_P9_19_gpio_input_pin
(8)
```

Trong cấu hình pinctrl, mỗi khai báo cấu hình sẽ được liên kết với một khái báo chức năng và điều này có thể được quan sát khi đọc nội dung của file pinctrl-handles. Do đó, các khái báo cấu hình xuất hiện trong file này cho thấy chúng đang được áp dụng trên BeagleBone Black (BBB). Dựa trên các thông tin đó, ta có thể đối chiếu với device tree để kiểm tra trạng thái cấu hình của các chân P9_24 và P9_26. Trong trường hợp hiện tại, hai chân này vẫn chưa được cấu hình.

Để thực hiện cấu hình pinctrl cho chân P9_24 và P9_26 chúng ta truy cập vào file ‘build/tmp/work-shared/beaglebone-yocto/kernel-source/arch/arm/boot/dts/am335x-boneblack.dts’và thêm đoạn code sau:

```
&am33xx_pinmux {
        i2c1_pins: pinmux_i2c1_pins {
                pinctrl-single,pins = <
                        AM33XX_PADCONF(AM335X_PIN_UART1_RXD, PIN_INPUT_PULLUP,
MUX_MODE3)     /* uart1_rxd.i2c1_sda */
                        AM33XX_PADCONF(AM335X_PIN_UART1_TXD, PIN_INPUT_PULLUP,
MUX_MODE3)     /* uart1_txd.i2c1_scl */
                >;
        };
};
```

```
&i2c1 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;
        status = "okay";
};
```

Thực hiện rebuild lại kernel bằng command sau:

```
$ bitbake -f -c compile linux-yocto
$ bitbake -c deploy linux-yocto
```

Tiếp tục, thêm i2c-tools package để có thể sử dụng command line 'i2cdetect' cho việc debug. Và rebuild lại image để update thay đổi.

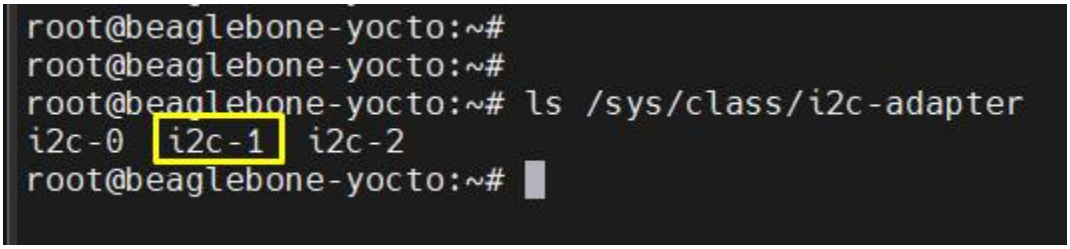
```
$ bitbake core-image-full-cmdline
```

Cuối cùng, chúng ta sẽ cần flash image vào sdcard và kiểm tra cấu hình

2.3. Kiểm tra cấu hình

Mục tiêu của bước này là kiểm tra xem bus I2C1 đã được enable và cấu hình pinctrl cho chân P9_24 và P9_26 đã chính xác chưa.

Thực hiện command 'ls /sys/class/i2c-adapter' để kiểm tra bus I2C1 đã được enable chưa. Nếu xuất hiện i2c-1, nghĩa là bus I2C1 đã được enable



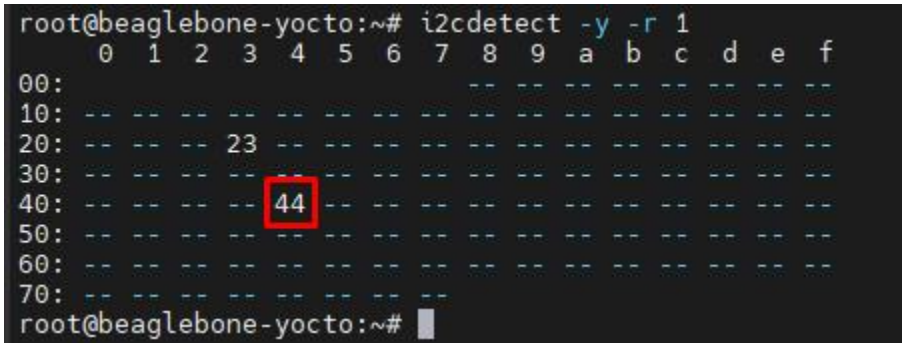
Thực hiện command 'cat /sys/kernel/debug/pinctrl/pinctrl-handles' để kiểm tra pinctrl của các chân đã cấu hình chưa. Nếu xuất hiện pinmux_i2c1_pins, nghĩa là được cấu hình thành công.

Tiếp theo, chúng ta sẽ kiểm tra address của GY-SHT30-D và viết chương trình demo

2.4. Kiểm tra address của GY-SHT30-D và viết chương trình demo

Mục tiêu của bước này kiểm tra được address của GY-SHT30-D và lấy address đó để viết chương trình demo đọc được dữ liệu lux từ GY-SHT30-D và in ra màn hình console.

Chúng ta có thể kiểm tra address của GY-SHT30-D bằng command 'i2cdetect -y -r 1'. 0x44 được trả về sau command chính là address của cảm biến.



Để đọc được dữ liệu về nhiệt độ và độ ẩm chúng ta cần thực hiện các bước sau:

1. Gửi lệnh đo: Gửi lệnh(0x24, 0x00) đo một lần tới cảm biến.
2. Chờ dữ liệu: Cảm biến cần vài mili giây để hoàn tất đo.
3. Đọc dữ liệu: 6 byte trả về gồm:

    2 byte nhiệt độ + CRC

    2 byte độ ẩm + CRC
4. Kiểm tra CRC: Tính CRC của nhiệt độ và độ ẩm, rồi so sánh với CRC được trả về
5. Xử lý dữ liệu:

    Temperature (°C) = -45 + 175 * (St / 65535)

    Humidity (%) = 100 * (Srh / 65535)

    Trong đó:

    St: Giá trị nhiệt độ raw 16-bit

    Srh: Giá trị độ ẩm raw 16-bit

Sau đây là chương trình demo để đọc dữ liệu về nhiệt độ và độ ẩm

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <linux/i2c-dev.h> /* header chứa defines và hàm để tương tác với i2c chardev */
#include <sys/ioctl.h>
#include <unistd.h>
#include <string.h>

#define SHT30_ADDR      0x44 /* I2C address of the SHT30 sensor */
```

```c
/* Hàm tính CRC8 */
unsigned char crc8(const unsigned char *data, size_t len)
{
    unsigned char crc = 0xFF;
    for (size_t i = 0; i < len; i++) {
        crc ^= data[i];
        for (int j = 0; j < 8; j++) {
            if (crc & 0x80) {
                crc = (crc << 1) ^ 0x31;
            } else {
                crc <<= 1;
            }
        }
    }

    return crc;
}

int main(int argc, char *argv[])
{
    char *devicePath = "/dev/i2c-1";
    int ret = 0;

       /* Open device file /dev/i2c-1 tương ứng với bus I2C1 */
    int fd = open(devicePath, O_RDWR);
    if (fd < 0) {
        perror("Failed to open the I2C device");
        return -1;
    }

       /* Set địa chỉ 0x23 */
    if (ioctl(fd, I2C_SLAVE, SHT30_ADDR) < 0) {
        perror("Failed to set the I2C address");
```

```
    ret = -1;
    goto cleanup;
}


for (int i = 0; i < 10; ++i)
{
   /* Gửi lệnh đến cảm biến */
    static char oneShotCmd[2] = {0x24, 0x00};
    if (write(fd, oneShotCmd, sizeof(oneShotCmd)) != sizeof(oneShotCmd)) {
        perror("Failed to write the command");
        ret = -1;
        goto cleanup;
    }
    usleep(50000); /* Đợi cho cảm biến đo hoàn tất */

   /* Đọc 6 bytes dữ liệu  */
    char data[6];
    if (read(fd, data, sizeof(data)) != sizeof(data)) {
        perror("Failed to read the data");
        ret = -1;
        goto cleanup;
    }

    /* Kiểm tra CRC */
    unsigned char crc1 = data[2];
    unsigned char crc2 = data[5];
    unsigned char crc1Calculated = crc8(data, 2);
    unsigned char crc2Calculated = crc8(data + 3, 2);
    if (crc1 != crc1Calculated || crc2 != crc2Calculated) {
        fprintf(stderr, "CRC check failed\n");
        continue;
    }
```

```
            /* Chuyển đổi dữ liệu thành nhiệt độ và độ ẩm */
            unsigned int rawTemp = (data[0] << 8) | data[1];
            unsigned int rawHum = (data[3] << 8) | data[4];
            float temperature = -45 + 175 * (float)rawTemp / 65535;
            float humidity = 100 * (float)rawHum / 65535;

            printf("Temperature: %.2f ° C\n", temperature);
            printf("Humidity: %.2f %%\n", humidity);
        }

    cleanup:
        close(fd);
        return ret;
    }
```

Bây giờ, chúng ta sẽ thực hiện cross compile chương trình bằng những command sau:

```
$ source /opt/poky/4.0.26/environment-setup-cortexa8hf-neon-poky-linux-gnueabi
$ ${CC} -o sht30 sht30.c
```

Khi chạy chương trình trên BBB, bạn sẽ thấy kết quả hiển thị ra màn hình

```
root@beaglebone-yocto:~# ./sht30
Temperature: 28.84 °C
Humidity: 82.93 %
Temperature: 28.84 °C
Humidity: 82.95 %
Temperature: 28.84 °C
Humidity: 82.96 %
Temperature: 28.84 °C
Humidity: 82.95 %
Temperature: 28.86 °C
Humidity: 82.93 %
Temperature: 28.87 °C
Humidity: 82.93 %
```