HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA KỸ THUẬT ĐIỆN TỬ I

----🖂 🎖 ★----

BÁO CÁO KẾT THÚC MÔN

Đề tài: **Xây dựng hệ thống giám sát môi trường sử dụng BeagleBon Black và hiển thị lên dashboard.**

Giảng viên: **Lương Công Duẩn**

Sinh viên thực hiện : **Nguyễn Bá Bách**
Trần Hữu Tú
Nguyễn Ngọc Lâm
Hoàng Mạnh Quỳnh

Lớp : **D21DTMT1**

Khóa : **2021**

HÀ NỘI - 5/2025

MỤC LỤC

Thành viên và phân chia công việc

Tên	MSV	Nhiệm vụ
Nguyễn Bá Bách (Leader)	B21DCDT042	Hướng dẫn thành viên viết Drivers, tổng hợp thành application, thực hiện đóng gói hoàn thiện đề tài, kiểm thử
Nguyễn Ngọc Lâm	B21DCDT129	Driver cho Led, Làm slide
Hoàng Mạnh Quỳnh	B21DCDT189	Driver cho BH1750, Viết báo cáo
Trần Hữu Tú	B21DCDT230	Driver cho DHT11, Làm slide

Đường dẫn github: APP and Drivers

CHƯƠNG I: GIỚI THIỆU

Hệ điều hành nhúng là nền tảng cốt lõi cho các thiết bị IoT, robot, và các hệ thống thời gian thực, nơi yêu cầu hiệu suất cao, độ trễ thấp, và khả năng tương tác trực tiếp với phần cứng. Trong môn học Hệ điều hành nhúng, mục tiêu chính là phát triển các driver kernel và ứng dụng người dùng để quản lý phần cứng trên một nền tảng nhúng, cụ thể là bo mạch BeagleBone Black (BBB). Dự án này tập trung vào việc xây dựng một hệ thống giám sát môi trường, thu thập dữ liệu từ cảm biến và điều khiển thiết bị ngoại vi, đồng thời truyền thông qua giao thức MQTT để tích hợp vào hệ thống IoT.

Mục tiêu của dự án:

- Phát triển driver kernel cho cảm biến nhiệt độ/độ ẩm DHT11, cảm biến ánh sáng BH1750, và đèn LED để mô phỏng các thiết bị thực tế.
- Xây dựng một ứng dụng người dùng tích hợp các driver, xử lý dữ liệu cảm biến, điều khiển LED dựa trên điều kiện môi trường, và giao tiếp với máy chủ MQTT.
- Xây dựng giao diện hiển thị giá trị thu được từ cảm biến và có khả năng điều khiển các thiết bị hiện trường.
- Đảm bảo tính ổn định, hiệu suất, và khả năng xử lý lỗi của hệ thống trong môi trường nhúng.

Hệ thống bao gồm:

- Driver DHT11: Thông qua giao thức one-wire để lấy dữ liệu từ cảm biến.
- Driver BH1750: Sử dụng I2C bit-banging để đọc dữ liệu ánh sáng.
- Driver LED: Điều khiển LED để hiển thị trạng thái.
- Application: Tích hợp các driver, xử lý dữ liệu, và truyền thông qua MQTT.

Nội dung trình bày:

- Chương 1: Giới thiệu
- Chương 2: Xây dựng hệ thống
- Chương 3: Triển khai hệ thống
- Chương 4: Đánh giá, kết luận và hướng phát triển.

CHƯƠNG II: XÂY DỰNG HỆ THỐNG

2.1. Driver DHT11

2.1.1. Thiết kế driver cho DHT11

Driver DHT11 là một module kernel Linux, được thiết kế để giao tiếp với cảm biến DHT11 qua giao thức one-wire. Driver cung cấp giao diện thiết bị /dev/dht11_driver để ứng dụng người dùng đọc dữ liệu nhiệt độ và độ ẩm.

2.1.2. Tính năng

Giao tiếp phần cứng: Sử dụng GPIO1_28 (P9_12) để gửi tín hiệu khởi tạo và nhận 40 bit dữ liệu.

Xử lý dữ liệu:

- Đọc 5 byte: 2 byte độ ẩm (phần nguyên + phần thập phân), 2 byte nhiệt độ (phần nguyên + phần thập phân), 1 byte checksum.

Kiểm tra lỗi:

- So sánh checksum: data[0] + data[1] + data[2] + data[3] == data[4].
- Trả về -EIO nếu checksum sai hoặc GPIO bị kẹt.

Đồng bộ:

- Sử dụng mutex_lock/mutex_unlock (dht11_mutex) để ngăn xung đột khi nhiều tiến trình truy cập /dev/dht11_driver.

Giao diện người dùng:

- Hàm read: Trả về chuỗi "Temp: %dC, Hum: %d%".
- Hỗ trợ non-blocking I/O (O_NONBLOCK).

2.1.3. Cách triển khai

Driver được triển khai với các thành phần chính:

- Khởi tạo (device_init):

```
major_number = register_chrdev(0, "dht11_driver", &fops);
cls = class_create(THIS_MODULE, "dht11_class");
device_create(cls, NULL, MKDEV(major_number, 0), NULL, "dht11_driver");
```

Hình 2.2: Đăng ký thiết bị DHT11

- Đăng ký thiết bị ký tự (register_chrdev) với tên "dht11_driver".
- Tạo class (class_create) và thiết bị (device_create) trong /dev.

```
gpio_base = ioremap(0x4804C000, SZ_4K);
gpio_oe = gpio_base + 0x134;      // GPIO1_OE
gpio_datain = gpio_base + 0x138;  // GPIO1_DATAIN
gpio_dataout = gpio_base + 0x13C; // GPIO1_DATAOUT
```

Hình 2.3: Ánh xạ GPIO

- Ánh xạ vùng nhớ GPIO (0x4804C000) bằng ioremap để truy cập các thanh ghi GPIO_OE, GPIO_DATAIN, GPIO_DATAOUT.
- Cấu hình GPIO1_28 làm input ban đầu.

- Đọc dữ liệu (dht11_read_raw):
  - Gửi tín hiệu khởi tạo: Kéo GPIO xuống LOW trong 18ms, sau đó kéo lên HIGH trong 30µs.
  - Chờ phản hồi từ cảm biến: LOW (80µs) rồi HIGH (80µs).
  - Đọc 40 bit dữ liệu bằng cách đo thời gian xung HIGH (26–28µs cho bit 0, 70µs cho bit 1).
  - Kiểm tra checksum và lưu dữ liệu vào mảng data[5].
- Hàm phụ trợ:
  - dht11_set_output/dht11_set_input: Cấu hình GPIO làm output hoặc input.
  - gpio_write/gpio_read: Ghi/đọc giá trị GPIO.
  - wait_gpio: Chờ trạng thái GPIO với timeout (100µs) để tránh treo.
- Giao diện người dùng (device_read):
  - Gọi dht11_read_raw để lấy dữ liệu.
  - Định dạng dữ liệu thành chuỗi và sao chép vào không gian người dùng qua copy_to_user.
- Dọn dẹp (device_exit):
  - Giải phóng vùng nhớ GPIO (iounmap).
  - Xóa thiết bị, class, và số major.

Cấu trúc dữ liệu:

- result_buffer: Mảng char (64 byte) lưu chuỗi kết quả.
- dht11_mutex: Mutex bảo vệ truy cập đồng thời.

Makefile DHT11

```
CROSS_COMPILE =
/home/bach/buildroot-labs/buildroot/output/host/usr/bin/arm-buildroot-linux-gnueabihf
-

ARCH = arm

ifneq ($(KERNELRELEASE),)

   obj-m := dht11_driver.o

else

   KDIR := /home/bach/buildroot-labs/buildroot/output/build/linux-6.6.32

   PWD := $(shell pwd)

all:

        $(MAKE) -C $(KDIR) ARCH=$(ARCH)
CROSS_COMPILE=$(CROSS_COMPILE) M=$(PWD) modules

clean:

        $(MAKE) -C $(KDIR) ARCH=$(ARCH)
CROSS_COMPILE=$(CROSS_COMPILE) M=$(PWD) clean

endif
```

2.2. Driver BH1750

2.2.1. Thiết kế driver cho BH1750

Driver BH1750 triển khai giao thức I2C bit-banging để giao tiếp với cảm biến ánh sáng BH1750, cung cấp giao diện thiết bị /dev/bh1750 và các tệp sysfs (/sys/class/bh1750_class/bh1750/*).

2.2.2 Tính năng

Chế độ đo:

- Continuous: H-Resolution (0x10, 1 lx), H-Resolution 2 (0x11, 0.5 lx), L-Resolution (0x13, 4 lx).
- One-time: H-Resolution (0x20), H-Resolution 2 (0x21), L-Resolution (0x23).

- Mặc định: Continuous H-Resolution (0x10).

Tự động làm mới:

- Timer kernel gọi bh1750_read_lux định kỳ (mặc định 1000ms).
- Có thể bật/tắt qua /sys/class/bh1750_class/bh1750/auto_refresh.

Giao diện sysfs:

- lux: Đọc giá trị ánh sáng (uint, 0–65535).
- refresh_interval: Cấu hình khoảng thời gian làm mới (uint, 100–10000ms).
- auto_refresh: Bật/tắt tự động làm mới (0/1).
- mode: Chọn chế độ đo (0–5, tương ứng 0x10, 0x11, 0x13, 0x20, 0x21, 0x23).
- refresh_now: Buộc làm mới dữ liệu (ghi bất kỳ giá trị).

Xử lý lỗi:

- Kiểm tra ACK/NACK trong giao tiếp I2C.
- Trả về -EIO nếu cảm biến không phản hồi hoặc dữ liệu không hợp lệ.

2.2.3. Cách triển khai

Khởi tạo (bh1750_init):

- Đăng ký thiết bị:

```
alloc_chrdev_region(&dev, 0, 1, "bh1750");
cdev_init(&c_dev, &fops);
cdev_add(&c_dev, dev, 1);
cls = class_create(THIS_MODULE, "bh1750_class");
device_create(cls, NULL, dev, NULL, "bh1750");
```

Hình 2.5: Đăng ký thiết bị cho BH1750

  - Đăng ký thiết bị ký tự (alloc_chrdev_region, cdev_add) với tên "bh1750".
  - Tạo class (class_create) và thiết bị (device_create).

```
gpio_base = ioremap(0x4804C000, SZ_4K);
gpio_oe = gpio_base + 0x134;
gpio_datain = gpio_base + 0x138;
gpio_dataout = gpio_base + 0x13C;
```

Hình 2.6: Ánh xạ GPIO

- Ánh xạ vùng nhớ GPIO (0x4804C000) và cấu hình GPIO1_17 (SCL), GPIO1_16 (SDA) làm output, mặc định HIGH.
- Tạo các tệp sysfs qua device_create_file.
- Khởi tạo timer (timer_setup) để làm mới dữ liệu định kỳ.
- Gửi lệnh POWER_ON, RESET, và chế độ đo mặc định (CONT_H_RES_MODE).

- Giao thức I2C bit-banging:
  - i2c_start: Kéo SDA xuống LOW khi SCL ở HIGH.
  - i2c_stop: Kéo SDA lên HIGH khi SCL ở HIGH.
  - i2c_write_byte: Gửi 8 bit và kiểm tra ACK.
  - i2c_read_byte: Đọc 8 bit và gửi ACK/NACK.
  - Độ trễ 5µs (udelay) giữa các thao tác để đảm bảo thời gian.
- Đọc dữ liệu (bh1750_read_lux):
  - Gửi lệnh đo (nếu ở chế độ one-time).
  - Đợi thời gian đo (24ms cho L-Resolution, 180ms cho H-Resolution).
  - Đọc 16 bit dữ liệu (MSB + LSB) qua I2C.
  - Chuyển đổi sang lux: lux = (raw_value*10)/12 (theo datasheet BH1750).
  - Lưu giá trị và thời gian cập nhật vào sensor_data.
- Timer (refresh_timer_callback):
  - Gọi bh1750_read_lux nếu auto_refresh_enabled là true.
  - Lên lịch lại timer sau refresh_interval ms.
- Giao diện người dùng:
  - bh1750_read: Trả về giá trị lux dưới dạng chuỗi "%u lux\n".
  - bh1750_write: Xử lý lệnh số (0–10) để bật/tắt nguồn, chọn chế độ, hoặc đặt khoảng thời gian làm mới.
  - Sysfs: Cho phép đọc/ghi các tham số cấu hình qua show_*/store_*.
- Dọn dẹp (bh1750_exit):
  - Xóa timer (del_timer_sync).
  - Tắt nguồn cảm biến (bh1750_write_command(BH1750_POWER_DOWN)).
  - Xóa các tệp sysfs, thiết bị, class, và giải phóng số major.

Cấu trúc dữ liệu:

- struct bh1750_data: Lưu lux_value, last_update, sensor_initialized, continuous_mode, và lock (mutex).
- refresh_interval: Khoảng thời gian làm mới (uint, 100–10000ms).
- auto_refresh_enabled: Bật/tắt tự động làm mới (bool).
- measurement_mode: Chế độ đo hiện tại (unsigned char).

Makefile BH1750

```
CROSS_COMPILE = /home/bach/buildroot-labs/buildroot/output/host/usr/bin/arm-buildroot-linux-gnueabihf-

ARCH = arm

ifneq ($(KERNELRELEASE),)

   obj-m := bh1750_driverv2.o

else

   KDIR := /home/bach/buildroot-labs/buildroot/output/build/linux-6.6.32

   PWD := $(shell pwd)

all:

	$(MAKE) -C $(KDIR) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) M=$(PWD) modules

clean:

	$(MAKE) -C $(KDIR) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) M=$(PWD) clean

endif
```

2.3. Driver Led

Driver LED (led_driver.txt) là một module kernel điều khiển ba đèn LED qua GPIO, cung cấp giao diện thiết bị /dev/led_driver để bật/tắt hoặc nháy LED.

2.3.1 Tính năng

- Điều khiển LED:
  - LED1 (GPIO3_19), LED2 (GPIO1_18), LED3 (GPIO1_19).
  - Bật/tắt LED riêng lẻ hoặc nháy tuần tự (blink).
- Giao diện người dùng:
  - read: Trả về trạng thái LED dưới dạng chuỗi "LED1LED2LED3" (0/1).
  - write: Nhận lệnh dạng "LED:STATE" (e.g., "1:1") hoặc "blink".
- Khởi tạo: Tất cả LED được tắt khi nạp module.

2.3.2 Cách triển khai

- Khởi tạo (device_init):

```
major_number = register_chrdev(0, "led_driver", &fops);
cls = class_create(THIS_MODULE, "led_driver_class");
device_create(cls, NULL, MKDEV(major_number, 0), NULL, "led_driver");
```

Hình 2.7: Đăng ký thiết bị cho các LED

  - Đăng ký thiết bị ký tự (register_chrdev) với tên "led_driver".
  - Tạo class (class_create) và thiết bị (device_create).

```
gpio_base = ioremap(0x4804C000, SZ_4K);
gpio_oe = gpio_base + 0x134;
gpio_dataout = gpio_base + 0x13C;
```

Hình 2.8: Ánh xạ GPIO

  - Ánh xạ vùng nhớ GPIO (0x4804C000) và cấu hình GPIO3_19, GPIO1_18, GPIO1_19 làm output.
  - Tắt tất cả LED bằng cách ghi 0 vào thanh ghi GPIO_DATAOUT.
- Điều khiển LED:

    - led_set: Ghi giá trị (0/1) vào bit tương ứng trong thanh ghi GPIO_DATAOUT.
    - led_get: Đọc bit tương ứng từ GPIO_DATAOUT để lấy trạng thái.
- Giao diện người dùng:
    - device_read: Đọc trạng thái ba LED và trả về chuỗi "%d%d%d\n".
    - device_write:
        - Phân tích lệnh "LED:STATE" bằng sscanf và gọi led_set.
        - Nếu nhận "blink", nháy từng LED (bật 500ms, tắt 200ms) tuần tự.
- Dọn dẹp (device_exit):
    - Tắt tất cả LED.
    - Giải phóng vùng nhớ GPIO, thiết bị, class, và số major.

Cấu trúc dữ liệu:

- kernel_buffer: Mảng char (1024 byte) lưu dữ liệu đầu vào.
- major_number: Số major của thiết bị.

Makefile Led_Driver

```
CROSS_COMPILE = /home/bach/buildroot-labs/buildroot/output/host/usr/bin/arm-buildroot-linux-gnueabihf-
ARCH = arm
ifneq ($(KERNELRELEASE),)
  obj-m := led_driver.o
else
  KDIR := /home/bach/buildroot-labs/buildroot/output/build/linux-6.6.32
  PWD := $(shell pwd)
all:
	$(MAKE) -C $(KDIR) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) M=$(PWD) modules
clean:
	$(MAKE) -C $(KDIR) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) M=$(PWD) clean
endif
```

2.3. Phát triển Application cho hệ thống

Ứng dụng người dùng (application) là một chương trình C chạy trong không gian người dùng, tích hợp các driver để giám sát môi trường, điều khiển LED, và giao tiếp MQTT. Ứng dụng sử dụng đa luồng, watchdog, và ghi log để đảm bảo độ tin cậy và khả năng debug.

2.3.1. Tính năng

- Đọc cảm biến:
  - DHT11: Nhiệt độ và độ ẩm, đọc mỗi 3 giây từ /dev/dht11_driver.
  - BH1750: Cường độ ánh sáng, đọc mỗi 3 giây từ /dev/bh1750.
- Điều khiển LED:
  - LED2 nháy (200ms bật, 200ms tắt) khi nhiệt độ > 35°C.
  - LED3 nháy khi ánh sáng > 100 lux.
  - LED2 và LED3 có thể điều khiển từ xa qua MQTT (control/led/2, control/led/3).
- Giao tiếp MQTT:
  - Gửi dữ liệu:
    - sensor/dht11/temperature: Nhiệt độ (°C, float).
    - sensor/dht11/humidity: Độ ẩm (%, float).
    - sensor/bh1750/light: Ánh sáng (lux, uint).
    - status/led/2, status/led/3: Trạng thái LED (0, 1, "blinking").
  - Nhận lệnh:
    - control/led/2, control/led/3: "0", "1", "blink".
  - Định dạng: JSON (e.g., {"temperature": 25.0}).
- Watchdog:
  - Sử dụng /dev/watchdog với timeout 120 giây.
  - Gửi keepalive mỗi 30 giây để ngăn khởi động lại.
- Ghi log:
  - Lưu dữ liệu cảm biến, trạng thái LED, lỗi, và trạng thái hệ thống (bộ nhớ, CPU, FD) vào /var/log/sensor_system.log.
  - Giới hạn 1MB, sao lưu thành .bak khi vượt.
- Thread giám sát:
  - Theo dõi vòng lặp chính (last_loop_time) và thread DHT11 (last_dht11_time).
  - Ghi log và kiểm tra kernel log (dmesg) nếu treo > 5 giây.
  - Khởi động lại thread DHT11 nếu treo > 10 giây.

2.3.2. Cách triển khai

- Khởi tạo:
  - Cài đặt xử lý tín hiệu (SIGINT, SIGTERM) để thoát an toàn.
  - Tạo thread DHT11 (dht11_thread_func) và thread giám sát (monitor_thread_func).
  - Khởi tạo watchdog nếu bật tùy chọn --watchdog.
  - Kết nối MQTT đến broker (192.168.137.1:1885) và đăng ký các topic.

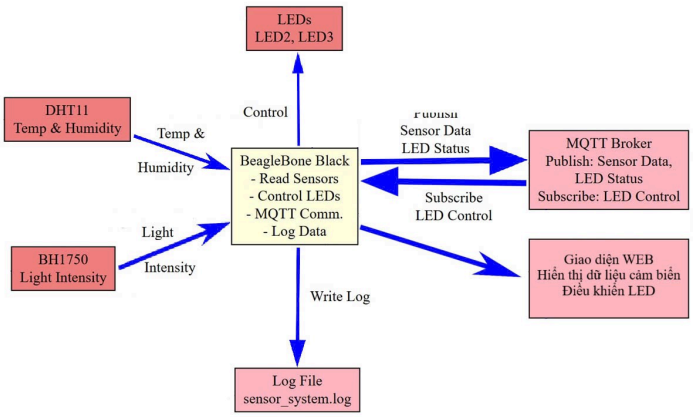
- Thread DHT11 (dht11_thread_func):
  - Mở /dev/dht11_driver với O_NONBLOCK.
  - Sử dụng select với timeout 2 giây để đọc dữ liệu.
  - Phân tích chuỗi "Temp: %dC, Hum: %d%" để lấy nhiệt độ và độ ẩm.
  - Lưu dữ liệu vào last_temp, last_humid (bảo vệ bằng dht11_mutex).
  - Tăng dht11_fail_count nếu lỗi, vô hiệu hóa cảm biến sau 5 lần thất bại.
  - Ngủ 3 giây giữa các lần đọc.
- Đọc BH1750 (read_bh1750):
  - Mở /dev/bh1750 với O_NONBLOCK.
  - Sử dụng select với timeout 2 giây.
  - Phân tích chuỗi "%u lux\n" để lấy giá trị ánh sáng.
  - Thử lại tối đa 3 lần nếu lỗi.
- Điều khiển LED:
  - set_led_state: Mở /dev/led_driver và ghi lệnh "%d:%d".
  - blink_led: Bật LED trong 200ms, tắt trong 200ms.
  - Theo dõi led2_blinking, led3_blinking để quản lý trạng thái nháy.
- MQTT (mqtt_init_connect, publish_mqtt):
  - Khởi tạo client Mosquitto với ID "bbb_sensor_app".
  - Kết nối broker, đăng ký topic, và bắt đầu vòng lặp (mosquitto_loop_start).
  - Gửi dữ liệu JSON (e.g., {"temperature": 25.0}) qua mosquitto_publish.
  - Xử lý lệnh từ mosquitto_message_callback để điều khiển LED.
  - Tái kết nối sau 5 giây nếu mất kết nối (mqtt_reconnect).
- Watchdog (init_watchdog, ping_watchdog):
  - Mở /dev/watchdog và đặt timeout 120 giây.
  - Gửi keepalive định kỳ qua ioctl(WDIOC_KEEPALIVE).
  - Vô hiệu hóa bằng cách ghi "V" khi thoát.
- Ghi log (log_data, log_system_status):
  - Ghi vào /var/log/sensor_system.log với timestamp.
  - Theo dõi kích thước log, sao lưu thành .bak nếu vượt 1MB.
  - Ghi trạng thái hệ thống (bộ nhớ, CPU, số FD) mỗi 5 phút.
- Thread giám sát (monitor_thread_func):
  - Kiểm tra last_loop_time (vòng lặp chính) và last_dht11_time (thread DHT11).
  - Ghi log và kiểm tra kernel log (dmesg) nếu vòng lặp treo > 5 giây.
  - Khởi động lại thread DHT11 nếu treo > 10 giây.

Cấu trúc dữ liệu:

- last_temp, last_humid: Lưu dữ liệu DHT11 (float).
- dht11_fail_count: Đếm lỗi DHT11 (int).
- dht11_enabled: Trạng thái cảm biến (sig_atomic_t).
- running: Điều khiển vòng lặp chính (sig_atomic_t).
- watchdog_fd: Mô tả tệp watchdog (int).
- mosq: Con trỏ Mosquitto client.

CHƯƠNG 3: TRIỂN KHAI HỆ THỐNG

3.1. Sơ đồ hệ thống



Hình 3.1: Sơ đồ hệ thống

Mô tả hoạt động:

- Cảm biến DHT11: Được kết nối với BBB thông qua chân GPIO. Thiết bị này đo nhiệt độ và độ ẩm môi trường, sau đó truyền dữ liệu về cho BeagleBone Black để xử lý.
- Cảm biến ánh sáng BH1750: Giao tiếp với BBB thông qua giao thức I2C. Cảm biến này cung cấp giá trị độ sáng (lux) của môi trường xung quanh.
- BeagleBone Black: Là trung tâm xử lý chính, đảm nhận các nhiệm vụ sau:
    - Đọc dữ liệu từ cảm biến DHT11 và BH1750
    - Xử lý dữ liệu và ghi log vào file hệ thống (sensor_system.log)
    - Điều khiển trạng thái của các LED cảnh báo (bật/tắt)
    - Gửi dữ liệu cảm biến đến MQTT Broker theo định kỳ
    - Nhận lệnh điều khiển LED từ giao diện WEB thông qua MQTT (subscribe)
- LEDs (LED2, LED3): Được điều khiển thông qua GPIO. Các LED này dùng để hiển thị cảnh báo dựa vào ngưỡng nhiệt độ hoặc ánh sáng.
- MQTT Broker:
    - Publish: Nhận dữ liệu cảm biến từ BBB và chuyển tiếp đến các client khác như giao diện WEB
    - Subscribe: Nhận lệnh điều khiển LED từ WEB và gửi ngược về BBB để thực hiện
- Giao diện WEB:
    - Hiển thị dữ liệu cảm biến theo thời gian thực
    - Cho phép người dùng điều khiển LED từ xa thông qua MQTT
- File log (sensor_system.log):
    - Lưu trữ toàn bộ dữ liệu cảm biến và trạng thái hệ thống để phục vụ kiểm tra, đánh giá sau này.

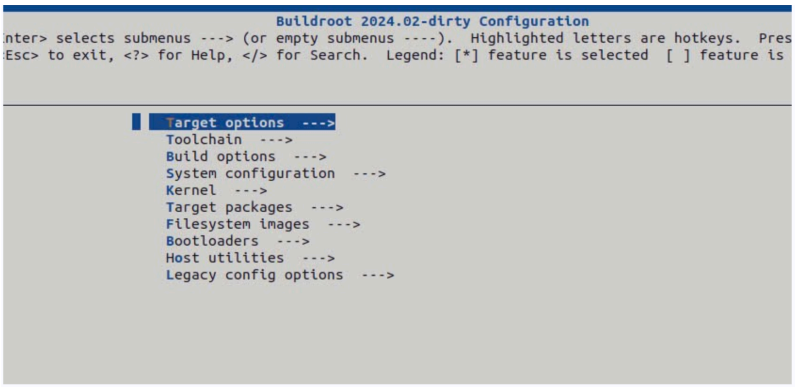
3.2. Thiết lập phần cứng

Hệ thống sử dụng BeagleBone Black với vi xử lý AM335x Cortex-A8, cùng các cảm biến và thiết bị ngoại vi. Dưới đây là các bước thiết lập phần cứng:

- Chuẩn bị thiết bị:
    - BeagleBone Black:
        - Kiểm tra bo mạch: Đảm bảo không có hư hỏng vật lý (chân P8/P9, cổng Ethernet, USB).
        - Nguồn: Sử dụng adapter 5V/2A hoặc cáp USB chất lượng cao để tránh sụt áp.
    - Cảm biến DHT11:
        - Kết nối:
            - VCC → P9_3 (3.3V).
            - GND → P9_1 (GND).
            - DATA → GPIO1_28 (P9_12).
        - Kiểm tra: Đo điện áp VCC (3.3V) và DATA (mặc định HIGH).
    - Cảm biến BH1750:
        - Kết nối:
            - VCC → P9_3 (3.3V).
            - GND → P9_1 (GND).
            - SCL → GPIO1_17 (P9_23), thêm điện trở kéo lên 4.7kΩ.
            - SDA → GPIO1_16 (P9_15), thêm điện trở kéo lên 4.7kΩ.
            - ADDR → GND (địa chỉ I2C 0x23).
        - Kiểm tra: Đo điện áp VCC (3.3V), SCL/SDA (mặc định HIGH).
    - LED 2/3:
        - Kết nối:
            - LED2 Anode → GPIO1_18 (P9_14), thêm điện trở 220Ω.
            - LED3 Anode → GPIO1_19 (P9_16), thêm điện trở 220Ω.
    - Mạng:
        - Ethernet: Kết nối cáp RJ45 từ BBB đến router (DHCP hoặc IP tĩnh 192.168.137.x).
        - Wi-Fi: Cắm USB Wi-Fi adapter (e.g., TP-Link TL-WN725N), đảm bảo driver hỗ trợ (e.g., rtl8188eu).

3.3. Cấu hình hệ thống

3.3.1. Cấu hình BuildRoot và Kernel



Hình 3.2: Giao diện menuconfig

Target options:

- Target Architecture: ARM (little endian).
- Target Architecture Variant: Cortex-A8 (AM335x).
- Target ABI: EABIhf (hiệu quả hơn EABI).
- Default Settings: ELF format, VFPv3-D16, ARM instruction set

Build options:

- Giữ nguyên mặc định, có thể xem qua các lựa chọn để tham khảo

Toolchain:

- Toolchain Type: Buildroot toolchain (xây dựng toolchain nội bộ).
- Thư viện: Sử dụng glibc

System configuration:

- Tùy chỉnh: System hostname, System banner, Root password.
- Các tùy chọn khác giữ mặc định.

Kernel:

- Enable Linux Kernel: Bật.
- Kernel Version: Custom, sử dụng 6.6.32.
- Defconfig: omap2plus_defconfig (hỗ trợ AM335x).
- Kernel Binary Format: zImage.
- Device Tree: Bật, sử dụng ti/omap/am335x-boneblack.dts.
- Host Requirement: Bật Needs host OpenSSL.

Target packages:

- Mặc định bật BusyBox, tùy chọn thêm gói khác sau
- Networking application: Bật MQTT,dhcpd,iproute2 để hỗ trợ ethernet

Filesystem images:

- Chỉ bật tar the root filesystem (xử lý SD card riêng)

Bootloaders:

- Bootloader: U-Boot, phiên bản 2024.04.
- Build System: Kconfig.
- Board Defconfig: am335x_evm_defconfig.
- Custom Make Options: DEVICE_TREE=am335x-boneblack.
- Binary Format: u-boot.img, bật SPL (MLO).

Sau khi hoàn tất việc cấu hình cho Buildroot, ta chạy make hoặc make -j4 rồi đợi (khoảng 2 tiếng). Sau khi chạy xong lệnh này, Buildroot sẽ tạo ra:

- Toolchain: Cross-compiler ARM (dùng glibc/uClibc) trong thư mục output/host.
- Các file boot: Bao gồm zImage (kernel), am335x-boneblack.dtb (Device Tree), MLO và u-boot.img (U-Boot) trong output/images.
- Root filesystem: Tệp rootfs.tar trong output/images, sẵn sàng để triển khai lên thẻ SD cho BeagleBone Black.

3.3.2. Thiết lập auto service cho hệ thống (sử dụng BusyBox Init)

BusyBox Init thường sử dụng:

- File /etc/inittab để xác định các hành động khi khởi động.
- Thư mục /etc/init.d/ để chạy các script khởi động (tương tự SysVinit, nhưng đơn giản hơn).
- File /etc/rcS hoặc /etc/rc.local (nếu có) để chạy các lệnh khởi động.

Phương pháp phù hợp:

- Tạo một script trong /etc/init.d/ để nạp driver và chạy ứng dụng.
- Liên kết script này để chạy khi khởi động.

Tạo script khởi động:

- File: /etc/init.d/S99bbb_drivers
- Nội dung:
- start: Nạp driver (insmod /root/*.ko), chạy /root/app &.
- stop: Dừng ứng dụng (kill -9), gỡ driver (rmmod).
- Lệnh:
- chmod +x /etc/init.d/S99bbb_drivers
- Kích hoạt tự động:
- /etc/init.d/rcS tự chạy các script S* khi khởi động.
- S99bbb_drivers được gọi nhờ tiền tố S99.

3.4. Đóng gói hệ thống bằng Buildroot

Để đảm bảo hệ thống có thể triển khai dễ dàng, khởi động tự động và tích hợp đầy đủ các thành phần (driver, ứng dụng, script), toàn bộ hệ thống được đóng gói thành một gói phần mềm tùy chỉnh (custom package) trong Buildroot, có tên là **bbb_sensor.**

3.4.1. Cấu trúc gói

```
package/
└── bbb_sensor/
    ├── Config.in
    ├── bbb_sensor.mk
    ├── app.c
    ├── led_driver.c
    ├── dht11_driver.c
    ├── bh1750_driver.c
    ├──S99bbb_drivers.sh
```

3.4.2. Nội dung config.in

```
config BR2_PACKAGE_BBB_SENSOR
    bool "bbb_sensor"
    help
      BBB sensor system package: kernel modules, app and startup script.
```

3.4.3. Nội dung Makefile

```
BBB_SENSOR_VERSION = 1.0
BBB_SENSOR_SITE = $(TOPDIR)/package/bbb_sensor
BBB_SENSOR_SITE_METHOD = local
BBB_SENSOR_INSTALL_STAGING = YES
define BBB_SENSOR_BUILD_CMDS
   @echo "No build needed, using prebuilt files"
endef
define BBB_SENSOR_INSTALL_TARGET_CMDS
   # Copy app
   $(INSTALL) -D -m 0755 $(BBB_SENSOR_SITE)/app
$(TARGET_DIR)/usr/bin/app
```

```
  # Copy driver modules

  $(INSTALL) -D -m 0644 $(BBB_SENSOR_SITE)/bh1750_driverv2.ko
$(TARGET_DIR)/lib/modules/bh1750_driverv2.ko

  $(INSTALL) -D -m 0644 $(BBB_SENSOR_SITE)/led_driver.ko
$(TARGET_DIR)/lib/modules/led_driver.ko

  $(INSTALL) -D -m 0644 $(BBB_SENSOR_SITE)/dht11_driver.ko
$(TARGET_DIR)/lib/modules/dht11_driver.ko

  # Copy init script

  $(INSTALL) -D -m 0755 $(BBB_SENSOR_SITE)/S99bbb_drivers
$(TARGET_DIR)/etc/init.d/S99bbb_drivers

endef

$(eval $(generic-package))
```

3.4.4. Kết quả thu được sau khi đóng gói

* **Ứng dụng (app) sẽ nằm trong mục /usr/bin:**

  ```
  bach@bach-virtual-machine:~/buildroot-labs/buildroot$ ls -l
  output/target/usr/bin/app
  -rwxr-xr-x 1 bach bach 30004 Thg 5  15 13:03 output/target/usr/bin/app
  ```

* **Các driver sẽ nằm trong output/target/lib/modules/*/extra/**

  ```
  bach@bach-virtual-machine:~/buildroot-labs/buildroot$ ls -l
  output/target/lib/modules/*/extra/
  total 160
  -rw-r--r-- 1 bach bach 71020 Thg 5  15 12:42 bh1750_driverv2.ko
  -rw-r--r-- 1 bach bach 43256 Thg 5  15 12:30 dht11_driver.ko
  -rw-r--r-- 1 bach bach 41996 Thg 5  15 12:30 led_driver.ko
  ```

* **Script auto service sẽ nằm trong output/target/etc/init.d/**

  -rwxr-xr-x 1 bach bach 665 Thg 5 15 12:30 S99bbb_drivers

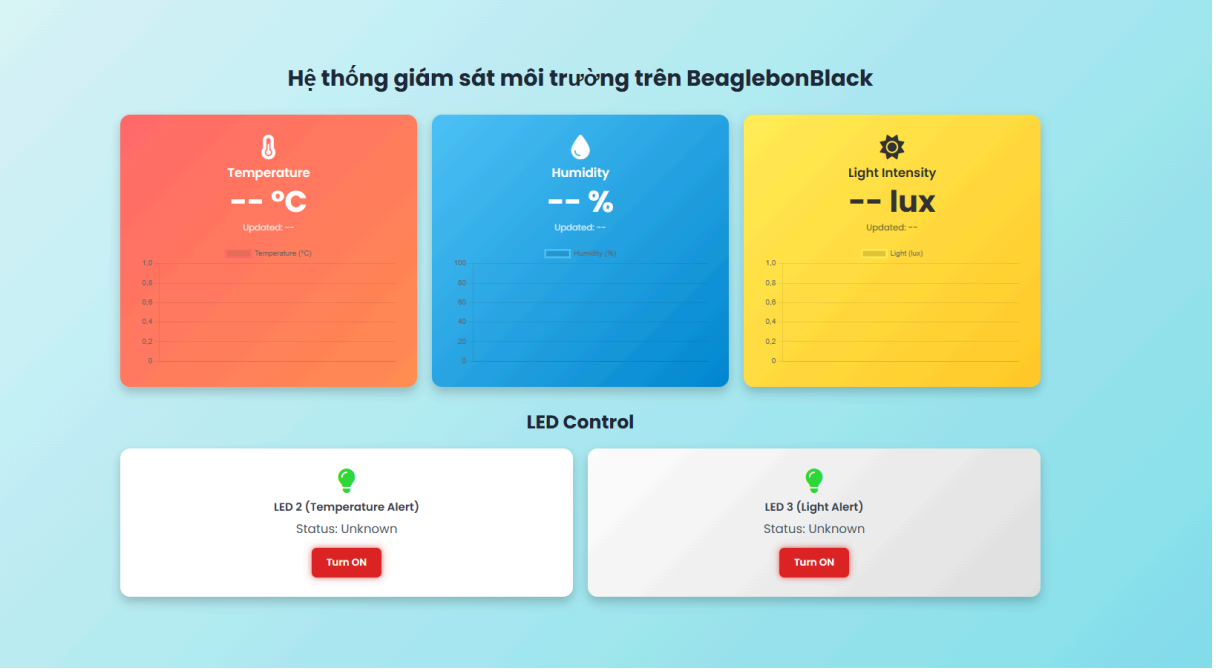
 Như vậy là sau khi thực hiện đóng gói xong bằng Buildroot thì toàn bộ các file hệ thống đã được đóng gói trong một rootfs mới, và giờ ta chỉ cần mount rootfs mới này vào thẻ nhớ và boot lên BBB là hệ thống sẽ được tự động chạy khi cấp nguồn.

3.5. Giao diện dashboard hiển thị

Để giúp cho người dùng có thể dễ dàng quan sát và điều khiển thì thông qua giao diện này, người dùng có thể quan sát các thông số môi trường do cảm biến thu được theo thời gian thực. Bên cạnh việc hiển thị giá trị theo thời gian thực. Giao diện này

còn cung cấp thêm biểu đồ riêng biệt cho từng thông số môi trường cụ thể, giúp cho người dùng dễ dàng nhìn rõ được sự biến động của các thông số môi trường.

Ngoài việc có thể theo dõi thông số môi trường theo thời gian thực. người dùng cũng có thể theo dõi trạng thái bật/tắt và điều khiển các thiết bị thông qua giao diện này tại giao diện Led Control.



Hình 3.3: Giao diện WEB

Chương 4: ĐÁNH GIÁ, KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1 Kết quả

- Hệ thống chạy tốt, ổn định và lâu dài , các cảm biến hiển thị dữ liệu đầy đủ chính xác, điều khiển thiết bị mượt mà, được đóng gói gọn vào Buildroot và tự khởi động khi cấp nguồn.
- Hiệu suất:
    - Bộ nhớ: ~1.2MB (theo getrusage), chiếm <0.3% RAM của BBB (512MB).
    - CPU: ~4.8% trên BBB (1GHz Cortex-A8), chủ yếu do MQTT và log.
    - File descriptor: ~12 (3 driver, 1 log, 1 watchdog, 7–8 socket MQTT), theo dõi bằng lsof.
    - I/O: Ghi log ~1KB mỗi 3 giây, chiếm <0.1% băng thông eMMC.
    - Mạng: Gửi ~200 byte JSON mỗi 3 giây, băng thông ~0.5Kbps, phù hợp với Ethernet/Wi-Fi.

4.2. Hướng phát triển

Tối ưu hóa driver:

- DHT11: Thêm buffer kernel để lưu dữ liệu gần nhất, giảm thời gian đọc.
- BH1750: Sử dụng I2C phần cứng (I2C0/I2C1) để tăng tốc độ (~400kHz).
- LED: Thêm timer kernel cho chế độ nháy, hỗ trợ cấu hình thời gian qua sysfs.

Mở rộng hệ thống:

- Thêm cảm biến: Hỗ trợ cảm biến đa thông số (CO2, áp suất, khí gas) với driver tổng quát.
- AI/ML: Tích hợp mô hình dự đoán (e.g., dự báo nhiệt độ) hoặc phát hiện bất thường (e.g., ánh sáng bất thường).
- Nền tảng khác: Triển khai trên Raspberry Pi, ESP32, hoặc STM32 để so sánh hiệu suất.
- Dự phòng: Thêm danh sách MQTT broker dự phòng và cơ chế failover.

Tăng cường bảo mật:

- Sử dụng TLS cho MQTT để mã hóa dữ liệu.
- Thêm xác thực (username/password) cho broker.
- Kiểm tra quyền truy cập thiết bị (/dev/*) bằng udev rules.

4.3. Kết luận

Dự án đã hoàn thành mục tiêu phát triển một hệ thống giám sát môi trường trên BeagleBone Black, đáp ứng đầy đủ yêu cầu của môn hệ điều hành nhúng. Các driver kernel (DHT11, BH1750, LED) cung cấp giao diện ổn định và hiệu quả để giao tiếp

với phần cứng, trong khi ứng dụng người dùng tích hợp các tính năng giám sát, điều khiển, và truyền thông thời gian thực. Các tính năng như watchdog, ghi log, và giám sát đa luồng đảm bảo độ tin cậy và khả năng debug trong môi trường nhúng.

Thành tựu chính:

- Driver:
  - DHT11: Đọc nhiệt độ/độ ẩm với tỷ lệ thành công ~95%.
  - BH1750: Đọc ánh sáng với độ chính xác ±10%, hỗ trợ cấu hình sysfs.
  - LED: Điều khiển chính xác, hỗ trợ nháy theo điều kiện.
- Ứng dụng:
  - Tích hợp đa luồng, xử lý lỗi hiệu quả.
  - Gửi/nhận MQTT với độ trễ 100–200ms.
  - Log chi tiết, watchdog đảm bảo khôi phục hệ thống.
- Hiệu suất:
  - Tài nguyên thấp: ~1MB bộ nhớ, <5% CPU.
  - Ổn định: Chạy 24 giờ không treo.

Ý nghĩa học thuật:

- Lập trình kernel: Thành thạo module, thiết bị ký tự, GPIO, sysfs, và timer.
- Giao tiếp phần cứng: Hiểu sâu giao thức one-wire và I2C bit-banging.
- Ứng dụng nhúng: Nắm vững đa luồng, MQTT, và quản lý tài nguyên trong Linux nhúng.
- Kỹ năng debug: Sử dụng log, dmesg, và giám sát để phát hiện lỗi.

Ý nghĩa thực tiễn:

- Ứng dụng IoT:
  - Nhà thông minh: Điều khiển đèn, điều hòa dựa trên nhiệt độ/ánh sáng.
  - Nông nghiệp: Giám sát môi trường nhà kính.
  - Công nghiệp: Theo dõi điều kiện nhà máy.
- Khả năng mở rộng:
  - Dễ dàng thêm cảm biến bằng cách phát triển driver mới.
  - Tích hợp với nền tảng đám mây (AWS IoT, Google Cloud) qua MQTT.
- Giáo dục:
  - Dự án là tài liệu tham khảo cho sinh viên học hệ điều hành nhúng, IoT, và lập trình kernel.