ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

ĐỒ ÁN TỐT NGHIỆP
CAPSTONE PROJECT

NGÀNH: KỸ THUẬT ĐIỀU KHIỂN VÀ TỰ ĐỘNG HÓA

ĐỀ TÀI:

NGHIÊN CỨU VÀ XÂY DỰNG HỆ ĐIỀU HÀNH LINUX NHÚNG TRÊN BEAGLEBONE BLACK.

Người hướng dẫn: **TS. GIÁP QUANG HUY**
KS. LƯU AN PHÚ

Sinh viên thực hiện:
LÊ THANH THƯỞNG – MSSV: 105200516 – LỚP: 20TDHCLC4
ĐẶNG NHẬT HUY – MSSV: 105200451 – LỚP: 20TDHCLC3

Đà Nẵng, 6/2025

TÓM TẮT

Tên đề tài: Nghiên cứu và xây dựng hệ điều hành Linux nhúng trên Beaglebone Black.

Sinh viên thực hiện	**Lê Thanh Thưởng**	**MSSV: 105200516**	**20TDHCLC4**
	Đặng Nhật Huy	**MSSV: 105200451**	**20TDHCLC3**

Trong bối cảnh công nghiệp 4.0, các hệ thống nhúng đóng vai trò quan trọng trong việc triển khai các giải pháp tự động hóa và IoT. Dự án này tập trung vào việc xây dựng một **hệ thống nhúng Linux tùy biến dựa trên nền tảng Beaglebone Black**, sử dụng Yocto Project để cấu hình, biên dịch và tối ưu hệ điều hành phù hợp với yêu cầu phần cứng và ứng dụng thực tế. Thông qua việc thiết lập môi trường phát triển, cấu hình pin-muxing, tích hợp driver thiết bị, và thêm các gói phần mềm cần thiết, hệ thống sẽ trở thành nền tảng vững chắc để triển khai các ứng dụng IoT tiên tiến.

Sau khi hoàn thiện hệ điều hành nhúng, hệ thống được ứng dụng vào mô hình vườn thông minh (Smart Garden), hỗ trợ giám sát và điều khiển môi trường trồng trọt một cách trực quan và hiệu quả. Hệ thống có khả năng thu thập các tham số môi trường như nhiệt độ, độ ẩm, cường độ ánh sáng thông qua các cảm biến tương ứng, sau đó hiển thị lên ThingsBoard (giao diện web) cũng như màn hình LCD cảm ứng gắn trực tiếp tại vườn. Ngoài ra, người dùng có thể theo dõi trạng thái thiết bị (đèn, máy bơm…) và điều khiển chúng từ xa qua internet hoặc trực tiếp từ màn hình LCD.

Hệ thống còn hỗ trợ các chế độ điều khiển tự động theo kịch bản, như hẹn giờ bật/tắt thiết bị dựa vào thời gian hoặc giá trị cảm biến, giúp tiết kiệm công sức vận hành. Giao diện khởi động có logo hiển thị trên màn hình LCD khi thiết bị được bật nguồn, tăng tính thân thiện và chuyên nghiệp.

Dự án không chỉ là một ứng dụng IoT nông nghiệp đơn thuần mà còn là một ví dụ điển hình về việc triển khai và tối ưu hệ điều hành nhúng Linux cho một nền tảng phần cứng cụ thể. Qua đó, nó góp phần mở rộng khả năng áp dụng hệ thống nhúng vào các mô hình tự động hóa thực tế trong nông nghiệp thông minh và các lĩnh vực liên quan.

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

TT	Họ tên sinh viên	Số thẻ SV	Lớp	Ngành
1	Lê Thanh Thưởng	105200516	20TDHCLC4	Kỹ thuật Điều khiển và Tự động hóa
2	Đặng Nhật Huy	105200451	20TDHCLC3	Kỹ thuật Điều khiển & Tự động hóa

1. ***Tên đề tài đồ án:***

   Nghiên cứu và xây dựng hệ điều hành Linux nhúng trên Beaglebone Black.

2. ***Đề tài thuộc diện:*** ☐*Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*

3. ***Các số liệu và dữ liệu ban đầu:***

   - Phần cứng: Sử dụng Beaglebone Black, màn hình LCD, các cảm biến nhiệt độ – độ ẩm, ánh sáng và các thiết bị điều khiển như đèn, máy bơm.
   - Phần mềm: Xây dựng chương trình nhúng hiển thị thông số lên LCD và điều khiển thiết bị. Phát triển giao diện giám sát, điều khiển từ xa qua ThingsBoard.
   - Cơ sở dữ liệu: Dữ liệu được lưu trữ và đồng bộ theo thời gian thực trên nền tảng ThingsBoard.

4. ***Nội dung các phần thuyết minh và tính toán:***

a. Phần chung:

TT	Họ tên sinh viên	Nội dung
1	Lê Thanh Thưởng	Tìm hiểu và lên ý tưởng cho đề tài: - Nghiên cứu về hệ điều hành Linux. - Nghiên cứu cách thiết kế và xây dựng một dự án nhúng Linux. - Nghiên cứu về Yocto và các môi trường biên dịch. - Nghiên cứu về các dự án nông trại xanh.
2	Đặng Nhật Huy	

b. Phần riêng:

TT	Họ tên sinh viên	Nội dung
	Lê Thanh Thưởng	- Xây dựng image cho BeagleBone Black bằng Yocto, bao gồm cài đặt môi trường, tải Poky, khởi tạo và cấu hình hệ thống build

		- Xây dựng SDK cho beagleboneblack và cài đặt các bộ SDK - Viết device tree cho các thiết bị phần cứng và cấu hình SSH để kết nối từ xa tới máy tính phát triển. - Phát triển chương trình hệ thống Smart Garden thu thập dữ liệu từ cảm biến, gửi và nhận dữ liệu qua socket và ThingsBoard, điều khiển thiết bị dựa trên dữ liệu cảm biến và tín hiệu điều khiển từ nền tảng ThingsBoard.
	Đặng Nhật Huy	- Đưa dữ liệu lên dashboard ThingsBoard, thực hiện hiển thị các giá trị lên Web, điều khiển thiết bị. - Thiết kế giao diện UI cho màn hình sử dụng QT Creator trên hệ điều hành Linux. - Phát triển chương trình hệ thống Smart Garden thu thập dữ liệu từ cảm biến, gửi và nhận dữ liệu qua socket và ThingsBoard, điều khiển thiết bị dựa trên dữ liệu cảm biến và tín hiệu điều khiển từ nền tảng ThingsBoard

5. Các bản vẽ,

a. Phần chung:

TT	Họ tên sinh viên	Nội dung
1	Lê Thanh Thưởng	- Lên ý tưởng thiết kế mạch layout và thi công mạch cho hệ thống giám sát. - Vẽ mạch in nguyên lý và layout PCB cho hệ thống.
2	Đặng Nhật Huy	

6. Họ tên người hướng dẫn:	*Phần/ Nội dung:*
TS. Giáp Quang Huy	
KS. Lưu An Phú	

7. Ngày giao nhiệm vụ đồ án: 30/1/2025

8. Ngày hoàn thành đồ án: 17/6/2025

Đà Nẵng, ngày tháng 6 năm 2025

Trưởng Bộ môn Tự động hóa **Người hướng dẫn**

TS. Giáp Quang Huy

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

PHIẾU KIỂM SOÁT TIẾN ĐỘ LÀM ĐỒ ÁN TỐT NGHIỆP

(Phiếu dành cho người hướng dẫn/sinh viên)

TT	Họ tên sinh viên	Số thẻ SV	Lớp	Ngành
1	Lê Thanh Thưởng	105200516	20TDHCLC4	Kỹ thuật Điều khiển và Tự động hóa
2	Đặng Nhật Huy	105200451	20TDHCLC3	Kỹ thuật Điều khiển & Tự động hóa

Tên đề tài ĐATN:

Nghiên cứu và xây dựng hệ điều hành Linux nhúng trên Beaglebone Black.

TT	Họ tên người HD	Đơn vị
1	TS. Giáp Quang Huy	Trường Đại học Bách Khoa & Đại học Đà Nẵng
2	KS. Lưu An Phú	Công ty TNHH Nghiên cứu và phát triển công nghệ lõi Linux Việt Nam

Tuần	**Ngày**	**Khối lượng**		**GVHD ký tên**
		đã thực hiện (%)	**tiếp tục thực hiện (%)**	
1				
2				
3				
4		Duyệt lần 1: Đánh giá khối lượng hoàn thành _____ % : Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
5				
6				
7				
8		Duyệt lần 2: Đánh giá khối lượng hoàn thành _____ % : Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		

9				
10				
11				
12		Duyệt lần 3: Đánh giá khối lượng hoàn thành _____ % : Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
13				
14				
15				

LỜI NÓI ĐẦU VÀ CẢM ƠN

Lời nói đầu nhóm xin trân trọng cảm ơn quý Thầy Cô trong Khoa Điện, Trường Đại học Bách khoa – Đại học Đà Nẵng đã tận tình giảng dạy, truyền đạt những kiến thức quý báu trong suốt bốn năm học, những kiến thức mà nhóm nhận được và tích luỹ trên giảng đường đại học là nền tảng để có thể thực hiện đề tài tốt nghiệp "Nghiên cứu và xây dựng hệ điều hành Linux nhúng trên Beaglebone Black", và cũng là hành trang để vững bước trong tương lai. Đề tài nghiên cứu được thực hiện dựa trên các kiến thức được học ở trong trường. Do khả năng của nhóm còn hạn chế nên không tránh khỏi những thiếu sót trong quá trình thực hiện và nghiên cứu, kính mong nhận sự góp ý kiến quý báu của Thầy Cô để đề tài được hoàn thiện hơn.

Nhóm xin chân thành gửi lời cảm ơn sâu sắc nhất tới Thầy Giáp Quang Huy và Anh Lưu An Phú đã giúp đỡ và chỉ dẫn tận tình để nhóm chúng tôi có thể hoàn thành đề tài một cách tốt nhất và đúng tiến độ. Trong thời gian được thầy hướng dẫn, nhóm không chỉ tiếp thu thêm nhiều kiến thức bổ ích mà còn học được tinh thần làm việc cũng như thái độ nghiên cứu đề tài nghiêm túc, hiệu quả, đây là những điều cần thiết cho chúng tôi trong quá trình học tập và công tác sau này.

Nhóm cũng xin chân thành cảm ơn đến gia đình, bạn bè đã luôn bên cạnh hỗ trợ và động viên để chúng tôi có thể nỗ lực học tập, nghiên cứu và thực hiện đề tài.

Cuối cùng, nhóm xin kính chúc quý thầy cô dồi dào sức khỏe để tiếp tục truyền đạt những kiến thức quý báu cho các thế hệ sinh viên.

Nhóm xin chân thành cảm ơn!

Đà nẵng, ngày tháng 6 năm 2025

Sinh viên thực hiện

Lê Thanh Thưởng Đặng Nhật Huy

LỜI CAM ĐOAN LIÊM CHÍNH HỌC THUẬT

Kính gửi: Hội đồng bảo vệ đồ án tốt nghiệp Khoa Điện, Trường Đại học Bách khoa - Đại học Đà Nẵng.

Nhóm sinh viên: Lê Thanh Thưởng, Đặng Nhật Huy, sinh viên Khoa Điện, Trường Đại học Bách khoa - Đại học Đà Nẵng.

Nhóm xin cam đoan trong quá trình làm đồ án tốt nghiệp thực hiện nghiêm túc các quy định về liêm chính học thuật:

- Không gian lận, bịa đặt, đạo văn, giúp người học khác vi phạm.
- Trung thực trong việc trình bày, thể hiện các hoạt động học thuật và kết quả từ hoạt động học thuật của bản thân.
- Không giả mạo hồ sơ học thuật.
- Không dùng các biện pháp bất hợp pháp hoặc trái quy định để tạo nên ưu thế cho bản thân.
- Chủ động tìm hiểu, tránh các hành vi vi phạm liêm chính học thuật, chủ động tìm hiểu và nghiêm túc thực hiện các quy định về luật sở hữu trí tuệ.
- Sử dụng sản phẩm học thuật của người khác phải có trích dẫn nguồn gốc rõ ràng.

Nhóm xin cam đoan số liệu và kết quả thực hiện trong đồ án này là trung thực và chưa hề được sử dụng để bảo vệ một học vị nào. Mọi sự giúp đỡ cho việc thực hiện đồ án này đã được cảm ơn và các thông tin trích dẫn trong đồ án đã được ghi nguồn gốc rõ ràng và được phép công bố.

Đà Nẵng, ngày tháng 6 năm 2025

Sinh viên thực hiện

Lê Thanh Thưởng Đặng Nhật Huy

MỤC LỤC

DANH SÁCH BẢNG BIỂU, HÌNH VẼ

DANH SÁCH CÁC CỤM TỪ VIẾT TẮT

Viết tắt	Nội dung
BBB	BeagleBoneBlack
GPIO	General Purpose Input Output
AC	Alternating Current
DC	Direct current
ADC	Analog to Digital Converter
WIFI	Wireless Fidelity
IoT	Internet of Things
CPU	Central Processing Unit
PWM	Pulse-Width Modulating
UART	Universal Asynchronous Receiver / Transmitter
SPI	Serial Peripheral Interface
I2C	Inter – Integrated Circuit
LCD	Liquid Crystal Display
NTP	Network Time Protocol
SSH	Secure Shell
SFTP	SSH File Transfer Protocol
PCB	Printed Circuit Board
SDK	Software Development Kit
MOSI	Master Out, Slave In
MISO	Master In, Slave Out
CS	Chip Select
SCL	Serial Clock Line
SDA	Serial Data Line
IRQ	Interrupt Request
DO	Data Out
DIN	Data In

MỞ ĐẦU

Ngày nay, chúng ta đang sống trong kỷ nguyên của cuộc cách mạng công nghiệp 4.0 với sự phát triển mạnh mẽ của công nghệ thông tin và kết nối không dây. Trong đó, mạng WiFi đóng vai trò quan trọng, giúp các thiết bị có thể kết nối và truyền dữ liệu mà không cần dây dẫn, tạo điều kiện thuận lợi để triển khai các hệ thống giám sát và điều khiển thông minh từ xa.

Smart Garden là một dự án ứng dụng công nghệ IoT nhằm xây dựng hệ thống giám sát và điều khiển môi trường cho khu vườn thông minh. Hệ thống cho phép theo dõi các thông số như nhiệt độ, độ ẩm, ánh sáng… theo thời gian thực, đồng thời điều khiển các thiết bị như đèn chiếu sáng, máy bơm thông qua kết nối internet.

Mục tiêu đề tài:

Mục tiêu của dự án là phát triển một hệ thống giám sát – điều khiển thông minh, hỗ trợ người dùng theo dõi và tác động lên điều kiện môi trường trồng trọt một cách tiện lợi và chính xác, từ đó tối ưu hóa hiệu quả chăm sóc cây trồng. Hệ thống hoạt động theo hai chế độ: tự động và thủ công. Ở chế độ tự động, hệ thống điều chỉnh thiết bị dựa trên dữ liệu cảm biến thu thập được. Ở chế độ thủ công, người dùng có thể điều khiển thiết bị trực tiếp thông qua giao diện Web. Đồng thời, các thông số môi trường và trạng thái thiết bị cũng được hiển thị trên màn hình LCD tại khu vực vườn.

Ngoài ra, hệ thống sử dụng nền tảng ThingsBoard để lưu trữ và hiển thị dữ liệu theo thời gian thực, hỗ trợ người dùng quản lý, theo dõi thông tin dễ dàng và hiệu quả

Báo cáo đồ án tốt nghiệp gồm 5 chương chính như sau:

CHƯƠNG 1: HỆ THỐNG NHÚNG LINUX TRÊN BEAGLEBONE BLACK ỨNG DỤNG TRONG SMART GARDEN

CHƯƠNG 2: TỔNG QUAN LINUX TRÊN HỆ THỐNG NHÚNG

CHƯƠNG 3: TỔNG QUAN LÝ THUYẾT THIẾT KẾ MỘT HỆ THỐNG LINUX NHÚNG TRÊN BEAGLEBONE BLACK

CHƯƠNG 4: THIẾT KẾ PHẦN CỨNG HỆ THỐNG

CHƯƠNG 5: THIẾT KẾ PHẦN MỀM HỆ THỐNG

CHƯƠNG 6: KẾT QUẢ VÀ ĐÁNH GIÁ HỆ THỐNG

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1.1 Giới thiệu chương

Linux nhúng là một phiên bản tinh gọn của hệ điều hành Linux, được thiết kế để hoạt động trên các thiết bị có tài nguyên hạn chế nhưng đòi hỏi hiệu năng và độ ổn định cao. Nhờ mã nguồn mở, khả năng tùy biến cao và cộng đồng phát triển mạnh, Linux nhúng trở thành lựa chọn ưu tiên cho các hệ thống điều khiển và giám sát tự động.

BeagleBone Black là một nền tảng phần cứng mạnh mẽ, hỗ trợ nhiều chuẩn giao tiếp và có khả năng chạy hệ điều hành Linux đầy đủ. Khi kết hợp với Linux nhúng, BeagleBone Black cho phép vừa khai thác sức mạnh xử lý của hệ thống, vừa giữ được khả năng điều khiển phần cứng chi tiết – rất phù hợp với các ứng dụng thời gian thực.

Trong xu thế phát triển của nông nghiệp thông minh và tự động hóa, việc sử dụng Linux nhúng trên BeagleBone Black mang lại giải pháp linh hoạt, tiết kiệm chi phí và dễ mở rộng. Đồng thời, hệ sinh thái phần mềm phong phú trên Linux giúp việc phát triển và triển khai ứng dụng trở nên nhanh chóng và hiệu quả hơn.

1.2 Tình trạng nghiên cứu của đề tài

Trong những năm gần đây, việc ứng dụng công nghệ nhúng vào nông nghiệp thông minh ngày càng phổ biến, tận dụng sức mạnh của IoT, cảm biến và nền tảng nhúng để giám sát và điều khiển môi trường canh tác. Ban đầu, các hệ thống thường dùng vi điều khiển như Arduino hay ESP32 vì chi phí thấp và dễ triển khai, nhưng những nền tảng này còn hạn chế về xử lý và khả năng mở rộng.

Để đáp ứng yêu cầu cao hơn như giao diện web, xử lý cục bộ và kết nối đám mây, xu hướng đã chuyển sang sử dụng các bo mạch như Raspberry Pi hoặc BeagleBone Black. Trong đó, BeagleBone Black nổi bật nhờ khả năng chạy Linux nhúng ổn định, bộ xử lý ARM Cortex-A8 mạnh mẽ, tích hợp PRU và truy cập GPIO với độ trễ thấp – rất phù hợp với các ứng dụng điều khiển ngoại vi thời gian thực.

Tại Việt Nam, nghiên cứu hệ thống nhúng Linux vẫn còn hạn chế, chủ yếu ở quy mô học thuật. Vì vậy, việc xây dựng hệ thống nhúng trên BeagleBone Black trong đề tài này không chỉ mang ý nghĩa thực tiễn mà còn góp phần cung cấp kinh nghiệm triển khai và tài liệu tham khảo cho cộng đồng kỹ thuật điều khiển và tự động hóa.

1.3 Sự cần thiết của đề tài

Trong bối cảnh chuyển đổi số đang lan rộng đến mọi lĩnh vực, việc đưa công nghệ vào nông nghiệp là xu hướng tất yếu để tăng năng suất, tối ưu nguồn lực và nâng cao chất lượng quản lý. Các hệ thống giám sát và điều khiển môi trường canh tác ngày càng đòi hỏi sự ổn định, khả năng mở rộng và tính linh hoạt trong kết nối – những yếu tố mà các nền tảng nhúng truyền thống khó đáp ứng.

Linux nhúng là giải pháp phù hợp, mang lại khả năng tùy biến cao, hỗ trợ đa nhiệm, giao tiếp mạng và xử lý dữ liệu cục bộ mạnh mẽ. BeagleBone Black, với cấu hình phần cứng mạnh và hỗ trợ tốt cho hệ điều hành Linux, là lựa chọn phù hợp để hiện thực hóa các hệ thống điều khiển thông minh trong nông nghiệp.

1.4 Mục đích nghiên cứu của đề tài

Đề tài nhằm nghiên cứu và triển khai hệ thống nhúng sử dụng Linux trên bo mạch BeagleBone Black. Thông qua việc cấu hình kernel, bootloader, tạo hệ thống tệp và phát triển phần mềm điều khiển, người thực hiện sẽ nắm được kiến thức nền tảng về hệ điều hành nhúng và triển khai thực tế.

Linux nhúng được chọn nhờ khả năng tùy biến, hỗ trợ đa nhiệm, bảo mật tốt và dễ tích hợp với các chuẩn phần cứng, phần mềm mã nguồn mở – phù hợp cho các hệ thống điều khiển có yêu cầu mở rộng.

Trên cơ sở đó, đề tài xây dựng hệ thống giám sát và điều khiển vườn thông minh với các mục tiêu:

- Thiết kế hệ thống có khả năng hoạt động ở hai chế độ: **tự động** (điều khiển theo cảm biến) và **thủ công** (điều khiển từ xa qua giao diện ThingsBoard).
- Thu thập và xử lý dữ liệu từ các cảm biến môi trường như nhiệt độ, độ ẩm, ánh sáng,...
- Điều khiển các thiết bị như hệ thống bơm, đèn chiếu sáng thông qua Linux.
- Giám sát từ xa qua mạng và hiển thị dữ liệu cục bộ trên LCD.
- Lưu trữ dữ liệu vào cơ sở dữ liệu để phục vụ phân tích, thống kê và hỗ trợ người dùng trong việc ra quyết định.

Qua đó, đề tài không chỉ góp phần hiện thực hóa các ứng dụng công nghệ cao trong nông nghiệp, mà còn giúp sinh viên tiếp cận sâu hơn với quy trình phát triển một hệ thống nhúng hoàn chỉnh trên nền Linux.

1.5 Đối tượng nghiên cứu và phạm vi nghiên cứu

1.5.1 Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là hệ thống nhúng Linux trên nền tảng BeagleBone Black, tập trung vào việc xây dựng, tùy biến và tối ưu hệ điều hành nhúng.

Các bước cấu hình kernel, bootloader, cross compiler và xây dựng root filesystem thủ công hoặc tự động hóa.

Ứng dụng của hệ thống nhúng này trong việc giám sát và điều khiển môi trường cho hệ thống Smart Garden.

1.5.2 Phạm vi nghiên cứu

Nghiên cứu quy trình build hệ điều hành Linux nhúng phù hợp với tài nguyên của BeagleBone Black. Cài đặt môi trường build trên Ubuntu 20.04, cấu hình và tạo image Linux tối ưu, bao gồm kernel, root filesystem và các gói cần thiết như SSH, giao diện dòng lệnh, driver cảm biến và điều khiển.

Triển khai image lên BeagleBone Black qua thẻ SD hoặc eMMC. Phát triển hệ thống thu thập dữ liệu môi trường (nhiệt độ, độ ẩm, ánh sáng) và điều khiển thiết bị trong vườn thông minh trên nền Linux nhúng.

Phạm vi đề tài không bao gồm các thuật toán điều khiển nâng cao hay AI, mà tập trung vào nền tảng phần mềm và tích hợp hệ thống.

1.5.3 Cách tiếp cận và phương pháp nghiên cứu:

Dự án được thực hiện theo hướng tiếp cận thực nghiệm và phát triển hệ thống nhúng. Trọng tâm là nghiên cứu, cấu hình và tối ưu hệ điều hành Linux nhúng trên BeagleBone Black, sau đó triển khai ứng dụng giám sát và điều khiển môi trường vườn thông minh dựa trên nền tảng đó.

❖ Phương pháp nghiên cứu sẽ bao gồm:

- Nghiên cứu tài liệu*:* Tìm hiểu về hệ thống nhúng, Linux nhúng, Yocto Project, BeagleBone Black và các cảm biến liên quan.
- Phân tích yêu cầu: Xác định yêu cầu phần cứng và phần mềm của hệ thống.
- Thiết kế và triển khai*:* Xây dựng hệ điều hành Linux tùy chỉnh, phát triển phần mềm điều khiển và giao diện người dùng.
- Kết nối phần cứng: Tích hợp cảm biến và thiết bị điều khiển với BeagleBone Black.
- Kiểm thử và đánh giá*:* Đánh giá hiệu năng và độ ổn định của hệ thống trong môi trường thực tế.

- Tối ưu và hoàn thiện*:* Tinh chỉnh cấu hình để nâng cao hiệu quả hoạt động của hệ thống.

1.6 Phương pháp đánh giá đề xuất

1.6.1 Hiệu quả

Phương pháp đánh giá hiệu quả của hệ thống sẽ tập trung vào các tiêu chí sau:

- Tối ưu tài nguyên: Sử dụng Yocto giúp giảm dung lượng hệ điều hành, rút ngắn thời gian khởi động và tiết kiệm RAM, CPU, bộ nhớ – phù hợp với giới hạn của BeagleBone Black.
- Hiệu quả điều khiển: Hệ thống hoạt động ổn định ở hai chế độ: tự động theo cảm biến và thủ công qua ThingsBoard.
- Giám sát & tương tác: Dữ liệu hiển thị rõ ràng qua LCD và web, đồng thời được lưu trữ phục vụ phân tích.
- Tính mở rộng: Dễ dàng tích hợp thêm thiết bị hoặc mở rộng ứng dụng mà không ảnh hưởng đến hệ thống hiện tại.

1.6.2 Kế hoạch nghiên cứu

Đề tài này đề xuất một kế hoạch nghiên cứu gồm các bước sau:

Bước 1: Khảo sát và phân tích yêu cầu:
- Xác định yêu cầu hệ thống giám sát và điều khiển môi trường nông nghiệp.
- Đánh giá phần cứng phù hợp (BeagleBone Black, cảm biến, thiết bị chấp hành...).

Bước 2: Thiết kế hệ thống tổng thể:
- Xây dựng sơ đồ kiến trúc phần cứng và phần mềm.
- Lựa chọn hệ điều hành Linux phù hợp để tùy chỉnh cho thiết bị.

Bước 3: Cấu hình và xây dựng hệ điều hành:
- Thiết lập môi trường build.
- Cấu hình kernel, hệ thống file và các gói cần thiết.
- Build và flash hệ điều hành lên thiết bị.

Bước 4: Phát triển ứng dụng giám sát và điều khiển:
- Lập trình giao tiếp với các cảm biến, màn hình LCD.
- Xây dựng giao diện điều khiển từ xa qua ThingsBoard.
- Thiết lập chế độ hoạt động tự động và thủ công.

Bước 5: Tích hợp hệ thống và kiểm thử

- Tích hợp phần cứng và phần mềm.
- Kiểm thử chức năng từng phần và toàn hệ thống trong điều kiện thực tế.

Bước 6: Đánh giá và hoàn thiện.

- Đánh giá hiệu quả hoạt động theo các tiêu chí: tài nguyên, tính ổn định, khả năng giám sát, mở rộng. Ghi nhận hạn chế và đề xuất hướng cải tiến

1.7 Kết luận chương 1

Trong Chương 1, đề tài đã được giới thiệu tổng quan, làm rõ lý do lựa chọn nghiên cứu, mục tiêu, phạm vi và phương pháp tiếp cận trong quá trình phát triển hệ thống điều khiển và giám sát môi trường sử dụng thiết bị nhúng. Bên cạnh đó, chương cũng trình bày kế hoạch thực hiện cụ thể nhằm đảm bảo tính khả thi và hiệu quả của đề tài.

Chương tiếp theo sẽ tập trung trình bày tổng quan về hệ điều hành Linux trong lĩnh vực hệ thống nhúng. Nội dung bao gồm các đặc điểm nổi bật của Linux, lý do lựa chọn Linux cho hệ nhúng, cũng như các thành phần chính của hệ điều hành này trong môi trường nhúng. Đây là cơ sở lý thuyết quan trọng để xây dựng và triển khai hệ thống điều khiển thông minh trên nền tảng phần cứng như BeagleBone Black trong các chương tiếp theo.

CHƯƠNG 2: TỔNG QUAN LINUX TRÊN HỆ THỐNG NHÚNG

2.1 Khái niệm cơ bản

Linux là tên gọi thay thế cho nhân Linux, hệ thống Linux hoặc bản phân phối Linux. Chính xác, Linux chỉ phần hạt nhân do Linus Torvalds phát triển, có nhiệm vụ quản lý phần cứng và điều phối hoạt động của toàn hệ thống. Dù không phải là phần đầu tiên được thực thi (bootloader chạy trước), nhưng khi đã khởi động, nhân Linux sẽ duy trì điều khiển hệ thống cho đến khi tắt máy.

Trong thực tế, khi nói "sử dụng Linux", người ta thường ám chỉ cả hệ thống Linux đầy đủ, gồm nhân Linux và các thành phần bổ trợ như phần mềm GNU, thư viện C, các ứng dụng, thậm chí cả giao diện đồ họa và tính năng real-time.

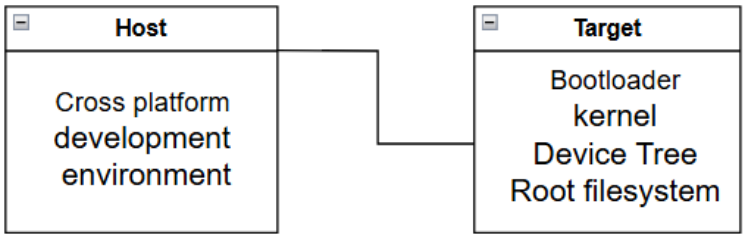
Linux có thể được xây dựng tùy chỉnh hoặc dựa trên các bản phân phối như Red Hat, Debian, Fedora,... Mỗi bản phân phối phục vụ mục đích khác nhau nhưng đều có điểm chung là cung cấp bộ công cụ để xây dựng hệ thống phù hợp với phần cứng và nhu cầu cụ thể của người dùng hoặc nhà phát triển.

2.2 Hệ điều hành Linux nhúng (Embbeded Linux)

Trong hệ thống nhúng, sử dụng một hệ điều hành Linux tùy chỉnh là giải pháp phổ biến nhằm đảm bảo tính linh hoạt, hiệu suất cao và tiết kiệm tài nguyên. Khác với các bản phân phối đầy đủ dành cho PC, Linux nhúng được thiết kế tối giản, chỉ bao gồm các thành phần cần thiết để phục vụ các chức năng như điều khiển thiết bị, thu thập dữ liệu hay giao tiếp ngoại vi.

Linux nhúng vẫn dựa trên nhân (kernel) Linux nhưng được cấu hình lại để phù hợp với phần cứng cụ thể, chẳng hạn như BeagleBone Black. Ngoài kernel, hệ thống còn bao gồm thư viện hệ thống, driver, tập tin cấu hình, tập tin thực thi và root filesystem. Các thành phần này có thể được tự xây dựng hoặc sử dụng từ các công cụ như Yocto Project hoặc Buildroot.

Một điểm quan trọng là mô hình phát triển tách biệt giữa **máy chủ phát triển (host)** và **thiết bị mục tiêu (target)**. Máy host dùng để cài đặt công cụ biên dịch chéo, tạo bootloader, đóng gói hệ thống. Máy target là thiết bị nhúng thực tế – ví dụ BeagleBone Black – nơi hệ điều hành được triển khai và chạy thử nghiệm.



Hình 2. 1 Mô hình tách biệt giữa máy chủ (host) và thiết bị mục tiêu (target)

Khác với các bản phân phối Linux phổ thông, hệ điều hành nhúng thường được biên dịch thủ công từ mã nguồn và cấu hình lại để phù hợp với phần cứng cụ thể, nhằm giảm kích thước kernel, rút ngắn thời gian khởi động và đảm bảo hiệu suất ổn định.

Ngoài kernel, hệ thống còn tích hợp các phần mềm nhẹ và thư viện tối ưu, phục vụ điều khiển, giám sát. Như vậy, một hệ điều hành Linux nhúng không chỉ là bản rút gọn mà là một nền tảng được thiết kế chuyên biệt cho ứng dụng cụ thể. Trong dự án này, việc xây dựng hệ điều hành nhúng cho BeagleBone Black giúp đáp ứng các yêu cầu điều khiển – giám sát trong môi trường nhúng một cách hiệu quả.

2.3 Phân loại hệ điều hành Linux nhúng:

2.3.1 Kích thước

Hệ điều hành Linux nhúng có thể được phân loại dựa trên mức độ tối giản hoặc đầy đủ của hệ thống:

- **Linux tối giản (Minimal Linux)** chỉ bao gồm các thành phần cơ bản như kernel, thư viện C, shell và tiện ích hệ thống nhỏ gọn (như BusyBox), phù hợp cho thiết bị có tài nguyên hạn chế. Ví dụ, trên BeagleBone Black (ARM Cortex-A8 1GHz, 512MB RAM, 4GB eMMC), hệ điều hành được build bằng Yocto Project với cấu hình tối giản, loại bỏ GUI và dịch vụ không cần thiết. Image sau biên dịch chỉ vài trăm MB, thích hợp cho eMMC hoặc thẻ microSD.
- **Linux tiêu chuẩn**: Bao gồm thêm các tiện ích quản lý gói, dịch vụ mạng, và một số daemon cơ bản.
- **Linux đầy đủ (Full Linux)**: Hệ điều hành tương đương như trên PC, có thể gồm cả giao diện người dùng đồ họa (GUI), trình duyệt, môi trường desktop...

2.3.2 Đáp ứng theo thời gian (real-time)

Tùy theo yêu cầu thời gian thực của hệ thống, Linux nhúng được phân loại:

- **Không thời gian thực (Non-RT)**: Dùng kernel mặc định, không cam kết thời gian phản hồi chính xác.
- **Thời gian thực mềm (Soft Real-Time)**: Dùng kernel PREEMPT, phản hồi nhanh, phù hợp điều khiển nông nghiệp như bật/tắt bơm theo cảm biến.
- **Thời gian thực cứng (Hard Real-Time)**: Dùng kernel vá RT-PREEMPT hoặc hệ điều hành chuyên dụng (như Xenomai), đảm bảo phản hồi chính xác trong thời gian nghiêm ngặt.

2.3.3 Khả năng kết nối

Một lợi thế lớn của Linux nhúng là khả năng kết nối linh hoạt với nhiều chuẩn như UART, SPI, I2C, CAN, Ethernet, Wi-Fi, Bluetooth.

Trên BeagleBone Black, hệ thống giao tiếp với cảm biến qua I2C/UART/GPIO, đồng thời có thể kết nối Ethernet hoặc Wi-Fi USB để truyền dữ liệu về trung tâm. Nhờ hỗ trợ tốt từ Linux, các giao thức như HTTP, MQTT dễ dàng tích hợp để phục vụ giám sát từ xa.

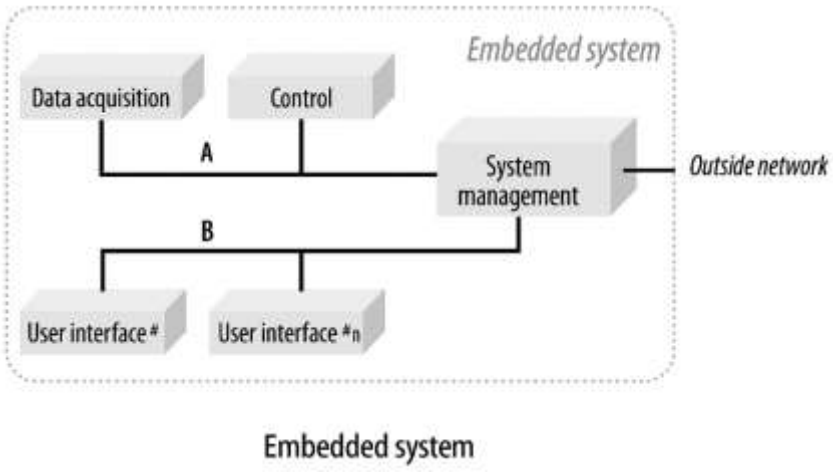
2.3.4 Khả năng tương tác với người dùng

Tùy mục đích, hệ thống nhúng có thể có hoặc không có giao tiếp người dùng. Một số dùng màn hình cảm ứng (như PDA), số khác chỉ có LED và nút nhấn, hoặc hoàn toàn tự động, không cần giao diện.

Hệ thống điều khiển vườn thông minh không yêu cầu tương tác trực tiếp, sử dụng giao diện dòng lệnh (CLI) qua UART hoặc SSH để cấu hình, kiểm tra. Dữ liệu được gửi lên dashboard từ xa để giám sát, giúp tiết kiệm tài nguyên và phù hợp triển khai ở môi trường ngoài trời.

2.4 Kiến trúc tổng quát của hệ thống Linux nhúng

Một hệ thống Linux nhúng tiêu biểu có thể được chia thành bốn thành phần chính: DAQ (Data Acquisition) – thu thập dữ liệu từ cảm biến; Control – xử lý và điều khiển thiết bị; System Management (SYSM) – quản lý hệ thống và kết nối mạng; và User Interface (UI) – giao tiếp với người dùng.



Hinh 2. 2 Kiến trúc của hệ thống Linux Nhúng

Các thành phần này được kết nối thông qua mạng nội bộ sử dụng giao thức TCP/IP. Trong đó, khối **DAQ** và **Control** thường chạy trên một đường truyền riêng để đảm bảo tốc độ và độ tin cậy. Khối **SYSM** nằm ở trung tâm, quản lý dữ liệu, giao tiếp với bên ngoài và kết nối các thành phần còn lại. Giao diện người dùng (UI) giúp quan sát, tương tác và điều khiển các tiến trình theo nhu cầu.

Kiến trúc này phù hợp cho các ứng dụng nhúng như giám sát công nghiệp, nhà thông minh hoặc hệ thống nông nghiệp tự động.

2.4.1 Khối Data Acquisition

Khối DAQ là thành phần đầu tiên trong chuỗi đo lường, thu thập tín hiệu điện từ các cảm biến sau khi được khuếch đại, lọc và cách ly. Dữ liệu sau đó được số hóa và lưu trữ tạm thời.

DAQ có thể phân tích dữ liệu tại chỗ hoặc gửi đến khối quản lý hệ thống (SYSM) để lưu trữ, hiển thị và xử lý. Nếu phát hiện bất thường, DAQ sẽ thông báo cho SYSM và nhận lệnh điều chỉnh.

DAQ khởi động từ bộ nhớ flash, dùng RAM, dễ cập nhật, không giao tiếp trực tiếp với người dùng mà kết nối với SYSM qua IP cố định để đảm bảo giám sát liên tục.

2.4.2 Khối Control

Khối Control là máy tính công nghiệp dạng nhúng, kết nối trực tiếp với phần cứng điều khiển. Nó phát tín hiệu điều khiển dựa trên dữ liệu từ DAQ và lệnh từ SYSM. Hệ thống hoạt động không cần giao tiếp người dùng, chạy Linux thời gian

thực để đảm bảo đáp ứng nhanh, kết nối mạng qua IP tĩnh. Sau khi thực hiện tác vụ, khối Control gửi trạng thái về SYSM để cập nhật.

2.4.3 Khối System Management (SYSM)

Khối SYSM là trung tâm điều phối và xử lý trong hệ thống nhúng, đóng vai trò kết nối DAQ, Control và giao tiếp với bên ngoài. Nó nhận dữ liệu từ DAQ, phân tích và gửi lệnh điều khiển đến Control, đồng thời truyền thông tin đến giao diện người dùng. SYSM chạy đa nhiệm, có IP tĩnh, hỗ trợ giao tiếp qua HTTP và SSH, cấp phát IP động cho UI, và hoạt động như một server nhúng để đảm bảo hệ thống vận hành ổn định, hiệu quả.

2.4.4 Khối User interface

UI là khối giao tiếp với người dùng, giúp quản lý tiến trình hệ thống. Đây thường là hệ thống nhúng nhỏ, đáp ứng thời gian thông thường, có khả năng kết nối mạng và nhận IP từ DHCP. UI boot từ flash hoặc qua mạng, chạy trên nền ARM/MIPS với kernel tiêu chuẩn, không cần giao diện đồ họa hay thư viện phức tạp.

2.5 Tổng quan về kernel của Linux/Unix

Kernel là thành phần cốt lõi của hệ điều hành, chịu trách nhiệm quản lý tài nguyên hệ thống như CPU, bộ nhớ và thiết bị ngoại vi. Nó cung cấp môi trường cho các ứng dụng hoạt động, đảm bảo sự ổn định và bảo mật khi nhiều tiến trình chạy song song.

2.5.1 Mô hình Process/Kernel

Hệ điều hành Linux hoạt động theo mô hình hai chế độ:

- **User Mode**: nơi các chương trình ứng dụng hoạt động.
- **Kernel Mode**: nơi kernel có quyền truy cập đầy đủ tới phần cứng.

Một tiến trình (process) thường chạy trong **User Mode** và chỉ chuyển sang **Kernel Mode** khi cần gọi các dịch vụ hệ thống (system call), ví dụ như truy cập file, cấp phát bộ nhớ, hoặc giao tiếp với thiết bị ngoại vi. Việc chuyển đổi này được CPU hỗ trợ bằng các lệnh đặc biệt để đảm bảo an toàn.

Kernel không phải là một tiến trình, mà là **trình quản lý tiến trình**, điều phối các tiến trình người dùng cũng như các **kernel thread** – là các tiến trình đặc quyền chạy ngầm để xử lý các tác vụ hệ thống.

Trên một hệ thống đơn nhân, tại một thời điểm chỉ có một tiến trình được thực thi và có thể chuyển đổi qua lại giữa User Mode và Kernel Mode.

2.5.2 Sự đồng bộ hóa và những vùng then chốt

Trong hệ thống nhúng Linux như trên BeagleBone Black, đặc biệt trong dự án nông nghiệp, nhiều tiến trình cùng hoạt động đồng thời có thể truy cập chung tài nguyên, dẫn đến tranh chấp dữ liệu (race condition). Để tránh tình trạng này, cần sử dụng cơ chế **đồng bộ hóa** nhằm bảo vệ các vùng tài nguyên quan trọng gọi là vùng then chốt (critical section).

Linux kernel cung cấp hai cơ chế đồng bộ hóa phổ biến là **mutex** và **semaphore**.

- **Mutex** cho phép chỉ một tiến trình truy cập vùng then chốt tại một thời điểm, ngăn chặn ghi đè dữ liệu.
- **Semaphore** điều phối truy cập tài nguyên khi có nhiều tiến trình cùng tranh chấp với số lượng giới hạn.

Trong Smart Garden, các tiến trình đọc dữ liệu cảm biến và điều khiển thiết bị cần được đồng bộ để đảm bảo dữ liệu chính xác và hoạt động ổn định. Việc sử dụng mutex hoặc semaphore giúp tránh xung đột truy cập, giữ cho hệ thống tin cậy và hoạt động liên tục.

2.6 Quản lý bộ nhớ trong Linux

2.6.1 Bộ nhớ ảo

Bộ nhớ ảo là một cơ chế quan trọng của các hệ điều hành hiện đại, đặc biệt là Linux. Nó tạo ra một lớp trừu tượng giữa yêu cầu bộ nhớ của chương trình và bộ nhớ vật lý thực tế. Nhờ đó, hệ thống có thể:

- Cho phép nhiều tiến trình thực thi đồng thời mà không can thiệp lẫn nhau.
- Hỗ trợ thực thi chương trình có dung lượng lớn hơn bộ nhớ RAM vật lý.
- Tải từng phần của chương trình vào RAM theo nhu cầu (on-demand).
- Tái định vị chương trình tại bất kỳ vị trí nào trong bộ nhớ vật lý.
- Hỗ trợ lập trình độc lập với kiến trúc bộ nhớ vật lý.

Không gian bộ nhớ ảo được CPU và MMU hỗ trợ thông qua cơ chế phân trang (paging), thường chia thành các page 4KB hoặc 8KB. Hệ thống sử dụng bảng trang (page table) để ánh xạ địa chỉ ảo sang địa chỉ vật lý. Việc ánh xạ này diễn ra tự động và nhanh chóng nhờ mạch cứng MMU tích hợp trong CPU.

2.6.2 Sử dụng bộ nhớ RAM

RAM trong hệ thống Linux được chia làm hai phần:

- Một phần nhỏ dùng cho kernel (mã nguồn và cấu trúc dữ liệu tĩnh).
- Phần còn lại được hệ thống quản lý linh hoạt qua bộ nhớ ảo để: Cấp phát bộ nhớ động cho kernel (bộ đệm, cấu trúc dữ liệu…). Hỗ trợ ánh xạ tập tin và vùng bộ nhớ dùng chung và Caching dữ liệu để tăng hiệu suất truy xuất thiết bị lưu trữ.

Trong điều kiện RAM giới hạn, hệ thống sử dụng thuật toán *page frame reclaiming* để giải phóng các trang không dùng đến.

2.7 Kết luận chương 2

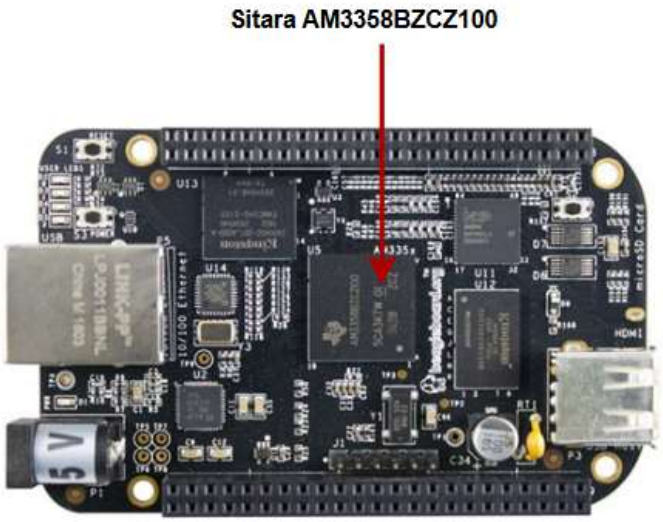
Chương 2 đã trình bày tổng quan về hệ điều hành Linux trong môi trường hệ thống nhúng, bao gồm các khái niệm cơ bản, đặc điểm của Linux nhúng cũng như phân loại và kiến trúc tổng thể của nó. Qua đó, người đọc có thể hiểu rõ vai trò của các khối chức năng trong hệ thống như Data Acquisition, Control, System Management và User Interface. Đồng thời, chương cũng cung cấp kiến thức nền tảng về nhân (kernel) của Linux, mô hình quản lý tiến trình, đồng bộ hóa, và cách thức quản lý bộ nhớ trong môi trường nhúng. Những kiến thức này là cơ sở quan trọng để bước sang chương tiếp theo – nơi sẽ đi sâu vào quá trình thiết kế và triển khai một hệ thống Linux nhúng thực tế trên nền tảng phần cứng BeagleBone Black.

CHƯƠNG 3: TỔNG QUAN LÝ THUYẾT THIẾT KẾ MỘT HỆ THỐNG LINUX NHÚNG TRÊN BEAGLEBONE BLACK

3.1 Phần cứng hỗ trợ

3.1.1 Kiến trúc CPU

Trong hệ thống được xây dựng, **BeagleBone Black** sử dụng **vi xử lý Sitara AM3358BZCZ100** của Texas Instruments. Đây là một vi xử lý dựa trên kiến trúc *ARM Cortex-A8 32-bit RISC*, hoạt động ở tần số *1GHz*, mang lại hiệu năng xử lý mạnh mẽ, phù hợp với các ứng dụng nhúng phức tạp và có yêu cầu tính toán cao, như trong các hệ thống IoT và giám sát môi trường.



Hình 3. 1 Kiến trúc CPU của Beaglebon Black

3.1.1.1 Thông số và đặc điểm nổi bật

- Vi xử lý: Sitara AM3358BZCZ100 – ARM Cortex-A8
- Tốc độ xử lý: 1GHz
- RAM: Tích hợp bộ nhớ 512MB DDR3, đủ đáp ứng cho hệ điều hành và các tiến trình xử lý dữ liệu song song.
- Lưu trữ: Hỗ trợ eMMC 4GB hoặc thẻ nhớ microSD, mở rộng dung lượng lưu trữ linh hoạt.
- Bộ nhớ cache: L1 (lệnh + dữ liệu) và L2 cache tích hợp giúp giảm độ trễ truy xuất và tăng tốc xử lý.

3.1.1.2 Khả năng hỗ trợ hệ điều hành và ứng dụng

AM3358 hỗ trợ hệ điều hành **Linux** cũng như các hệ điều hành thời gian thực (RTOS). Nhờ tích hợp **MMU (Memory Management Unit)**, hệ thống có thể thực thi các chức năng như:

- Quản lý bộ nhớ ảo, cách ly tiến trình,
- Chạy các hệ điều hành chuẩn như **Debian, Ubuntu, Yocto**, v.v.

Việc sử dụng Linux trên vi xử lý này mang lại khả năng tùy biến linh hoạt, hỗ trợ đa luồng, đa tác vụ và có kho phần mềm mã nguồn mở phong phú.

3.1.1.3 Ưu điểm

- Hiệu năng cao, tiêu thụ điện thấp – đặc trưng của kiến trúc ARM.
- Tích hợp nhiều tính năng trên một chip (SoC), giảm thiểu phần cứng bổ sung.
- Cộng đồng phát triển mạnh, dễ dàng tiếp cận tài liệu, driver và hỗ trợ kỹ thuật.
- Khả năng hoạt động như một máy tính nhúng, thực hiện xử lý dữ liệu phức tạp ngay tại biên (edge computing).

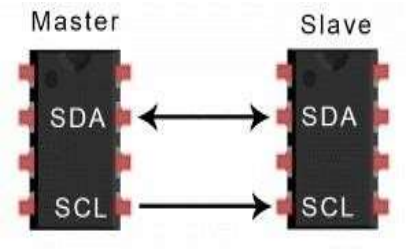
3.1.2 Bus và các chuẩn giao tiếp

Trong hệ thống Linux nhúng được triển khai trên nền tảng **BeagleBone Black**, các chuẩn giao tiếp ngoại vi đóng vai trò quan trọng trong việc kết nối bộ xử lý trung tâm với các cảm biến, thiết bị hiển thị và điều khiển. Các bus giao tiếp chính được sử dụng bao gồm:

3.1.2.1 Chuẩn giao tiếp I2C

I2C (Inter-Integrated Circuit) là giao thức truyền thông nối tiếp phổ biến, cho phép kết nối nhiều thiết bị Slave với một hoặc nhiều Master chỉ qua **hai đường dây**:

- **SDA (Serial Data):** truyền dữ liệu
- **SCL (Serial Clock):** cung cấp xung nhịp đồng bộ



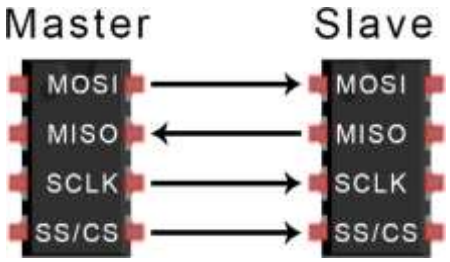
Hình 3. 2 Chuẩn giao tiếp I2C

I2C hoạt động theo cơ chế khởi động – truyền khung địa chỉ – khung dữ liệu – xác nhận (ACK/NACK) – và kết thúc. Mỗi Slave có một địa chỉ riêng, giúp Master xác định đúng thiết bị để giao tiếp. Giao thức này sử dụng bit đọc/ghi để phân biệt dữ liệu đang gửi hay yêu cầu nhận lại.

3.1.2.2 Chuẩn giao tiếp SPI

❖ **Tổng quan về giao tiếp SPI**

SPI (Serial Peripheral Interface) là giao thức giao tiếp nối tiếp đồng bộ, cho phép vi điều khiển (master) trao đổi dữ liệu tốc độ cao với các thiết bị ngoại vi (slave) như cảm biến, thẻ nhớ, hoặc module thu phát không dây và đặc biệt là màn hình.



Hình 3. 3 Chuẩn giao tiếp SPI.

- **MOSI (Master Out – Slave In)**: Master (BeagleBone Black) gửi dữ liệu đến thiết bị ngoại vi.
- **MISO (Master In – Slave Out)**: Thiết bị gửi dữ liệu về cho BeagleBone.
- **SCLK**: Xung nhịp do master tạo ra để đồng bộ truyền dữ liệu.
- **CS/SS (Chip Select/Slave Select)**: Chọn thiết bị slave để giao tiếp.

3.1.2.3 Chuẩn giao tiếp GPIO

GPIO (General Purpose Input/Output) là các chân đầu vào/ra số, được cấu hình linh hoạt để giao tiếp với thiết bị ngoại vi. Trên BeagleBone Black, GPIO đóng vai trò quan trọng trong việc đọc tín hiệu từ cảm biến (như công tắc mức nước, cảm biến mưa) hoặc điều khiển thiết bị (như relay, bơm nước, van điện).

GPIO hoạt động ở hai chế độ:

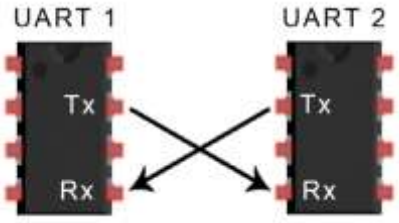
- **Input**: Đọc tín hiệu logic từ môi trường (0 hoặc 1).
- **Output**: Xuất tín hiệu để điều khiển thiết bị theo mức logic (0V hoặc 3.3V).

Trong dự án, GPIO được sử dụng để điều khiển hệ thống tưới tiêu và giám sát trạng thái môi trường, là cầu nối chính giữa BeagleBone Black và phần cứng vườn thông minh.

3.1.2.4 Chuẩn giao tiếp UART

❖ **Tổng quan lý thuyết**

UART là một giao thức truyền thông nối tiếp không đồng bộ, phổ biến trong các hệ thống nhúng. Khác với SPI hoặc I2C, UART **không cần tín hiệu xung nhịp (clock)** để đồng bộ, vì các thiết bị giao tiếp UART thỏa thuận trước một tốc độ truyền dữ liệu gọi là **baud rate** (ví dụ: 9600, 115200 bps).



Hình 3. 4 Chuẩn giao tiếp UART

Giao tiếp UART chỉ cần hai dây chính:

- **TX (Transmit)** – Dây truyền dữ liệu từ thiết bị gửi.
- **RX (Receive)** – Dây nhận dữ liệu từ thiết bị khác.

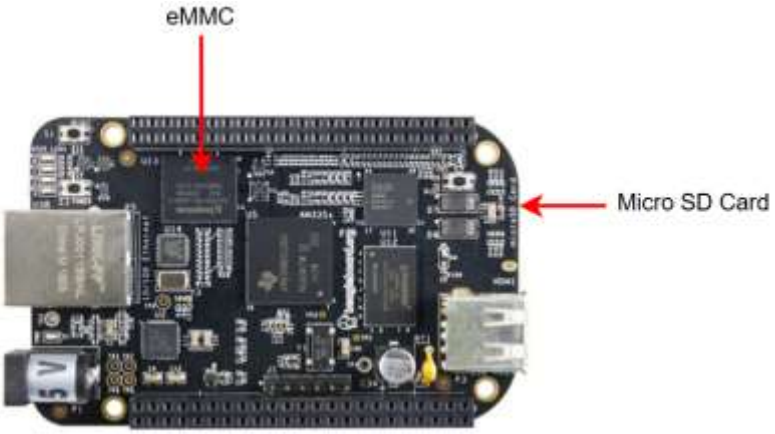
❖ **Nguyên lý hoạt động:**

UART truyền dữ liệu nối tiếp từng bit, bắt đầu bằng bit Start (0), 8 bit dữ liệu, tùy chọn bit parity và kết thúc bằng bit Stop (1).

- **Tốc độ truyền (baud rate):** Hai thiết bị phải cùng baud rate (ví dụ: 9600 bps) để tránh lỗi dữ liệu.
- **Không cần clock chung:** Chỉ cần TX và RX nhờ cơ chế lấy mẫu nội bộ.
- **Full-duplex:** Có thể gửi và nhận đồng thời nếu phần cứng hỗ trợ.

3.1.3 Thiết bị lưu trữ

BeagleBone Black (BBB) hỗ trợ hai loại thiết bị lưu trữ chính: **bộ nhớ eMMC** tích hợp sẵn và **thẻ microSD** rời. Cả hai phương án này đóng vai trò quan trọng trong việc lưu trữ hệ điều hành, ứng dụng người dùng, dữ liệu cảm biến cũng như các file cấu hình hệ thống.



Hình 3. 5 Thiết bị lưu trữ trên Beaglebone Black

3.1.3.1 eMMC

BeagleBone Black sử dụng eMMC 4GB hàn cố định trên bo mạch, kết nối với cổng MMC1 của vi xử lý AM335x. Thiết bị này hỗ trợ bus 8-bit và là bộ nhớ lưu trữ mặc định để khởi động hệ điều hành khi cấp nguồn.

- Tốc độ đọc/ghi nhanh hơn so với thẻ microSD thông thường.
- Độ ổn định và tuổi thọ cao trong môi trường công nghiệp.
- Cho phép bo mạch khởi động nhanh chóng mà không cần thiết bị rời.

eMMC là bộ nhớ cố định nên không thể thay thế dễ dàng khi hư hỏng

3.1.3.2 Thẻ nhớ microSD

BeagleBone Black cũng hỗ trợ một khe cắm thẻ microSD, kết nối qua **cổng MMC0**. Đây là lựa chọn phổ biến để:

- Khởi động hệ điều hành thay thế (boot từ thẻ).
- Cập nhật phần mềm hệ thống hoặc ghi đè lại nội dung eMMC.
- Lưu trữ dữ liệu người dùng, nhật ký cảm biến, hình ảnh, v.v.

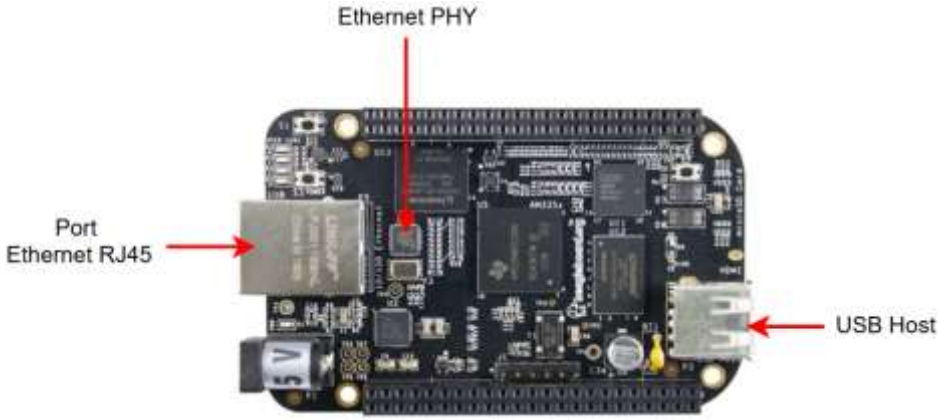
Khi chèn thẻ microSD chứa hệ điều hành hợp lệ và giữ nút **S2 (Boot Select)** trong khi cấp nguồn, bo mạch sẽ khởi động từ microSD thay vì eMMC.

3.1.4 Kết nối mạng

BeagleBone Black hỗ trợ kết nối mạng thông qua hai hình thức chính: **Ethernet tích hợp** và **WiFi qua USB**, giúp thiết bị dễ dàng giao tiếp và truyền dữ liệu đến các hệ thống điều khiển từ xa hoặc nền tảng IoT.

3.1.4.1Ethernet (RJ45 – 10/100 Mbps)

Bo mạch BeagleBone Black được tích hợp cổng **Ethernet 10/100 Mbps** sử dụng giao tiếp MAC bên trong SoC AM335x và PHY ngoài. Ethernet đóng vai trò là kênh truyền thông ổn định, có độ trễ thấp, phù hợp cho các ứng dụng điều khiển giám sát thời gian thực.



Hình 3. 6 Kết nối mạng trên Board Beaglebone Black

Ethernet được sử dụng để:

Kết nối thiết bị với máy chủ giám sát trung tâm (PC hoặc gateway), truyền dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) về máy chủ theo chu kỳ và nhận lệnh điều khiển từ người dùng thông qua mạng LAN.

Sử dụng Ethernet giúp đảm bảo tính ổn định và băng thông truyền tải trong môi trường cố định như hệ thống nông nghiệp thông minh, nhà kính, hoặc khu công nghiệp.

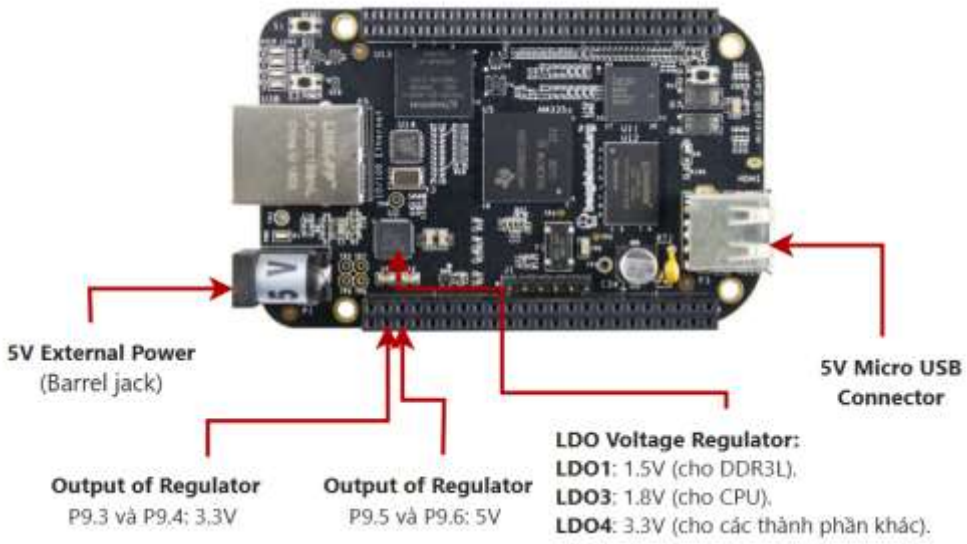
3.1.4.2 Kết nối WiFi qua USB

Ngoài Ethernet, BeagleBone Black có thể mở rộng kết nối không dây WiFi thông qua **USB WiFi Adapter**. Giao diện USB hỗ trợ cắm thiết bị WiFi chuẩn 802.11b/g/n phổ biến, giúp bo mạch có thể kết nối vào mạng WiFi như một thiết bị IoT.

Ưu điểm của kết nối WiFi trong dự án:

- Tính linh hoạt cao, không phụ thuộc vào hệ thống dây mạng cố định.
- Tích hợp với các dịch vụ cloud, MQTT broker, hoặc Node-RED dashboard để theo dõi dữ liệu từ xa qua trình duyệt.

3.1.5 Các nguồn ra/vào trên Board

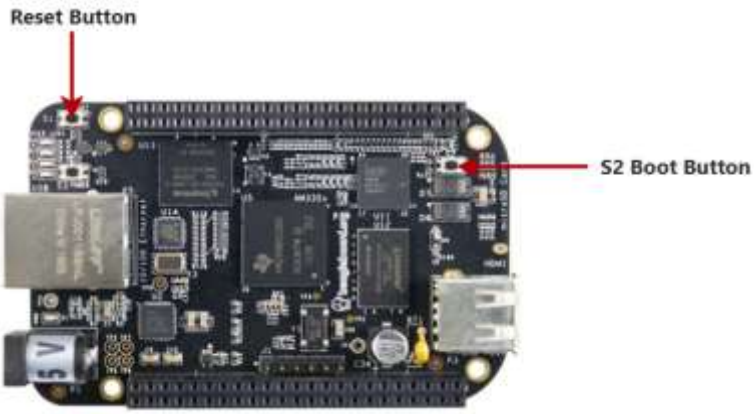


Hình 3. 7 Các nguồn ra/vào trên Bỏad Beaglebone Black

Do đa số phần lớn các module cảm biến sử dụng nguồn 3.3V nên board BeagleBone Black cũng cho ra nguồn 3.3V để dễ dàng kết nối. Để có được điện áp 3.3V thì có bộ chỉnh điện áp LDO (Low DropOut) để giữ điện áp ổn định ở 3.3V.

Việc cấp nguồn cho bo mạch BeagleBone Black (BBB) được thực hiện thông qua đầu nối nguồn DC (jack barrel) với nguồn adapter 5V – 2A để đảm bảo dòng điện ổn định cho toàn hệ thống, ngoài ra có thể cấp nguồn trực tiếp qua cổng micro USB khi sử dụng trong các ứng dụng phát triển hoặc thử nghiệm.

3.1.6 Nút nhấn và đèn Led trên Board BeagleBone Black



Hình 3. 8 Nút nhấn và đèn Led trên Board BeagleBone Black

Có một công tắc S2 cho phép chuyển đổi giữa các chế độ khởi động (boot):

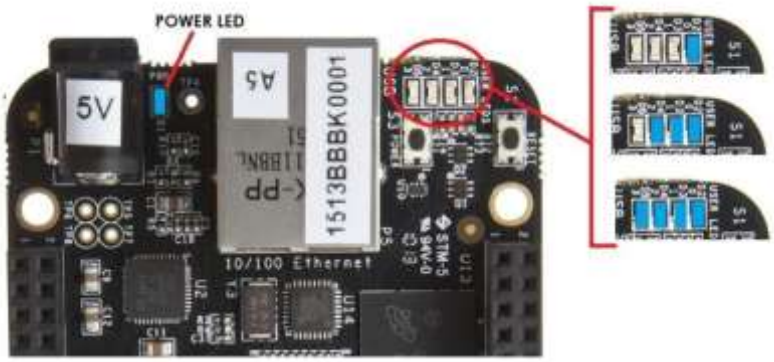
Có 4 chế độ khởi động:

- **Khởi động eMMC:** Là chế độ mặc định, cho phép bo mạch khởi động nhanh nhờ sử dụng ảnh hệ điều hành đã được flash sẵn, không cần thẻ nhớ microSD.
- **SD Boot:** Khởi động từ khe microSD. Có thể dùng để ghi đè eMMC, lập trình trong sản xuất hoặc cập nhật tại hiện trường.
- **Serial Boot:** Khởi động qua cổng nối tiếp để tải phần mềm trực tiếp. Cần cáp USB–Serial riêng để sử dụng.
- **USB Boot:** Hỗ trợ khởi động qua cổng USB.

Khi không nhấn nút S2, BeagleBone Black sẽ khởi động theo thứ tự: eMMC → microSD → cổng nối tiếp → USB.

Nếu giữ nút S2 khi tháo và cắm lại nguồn, bo mạch sẽ buộc khởi động từ USB, và nếu không tìm thấy hệ điều hành, sẽ chuyển sang cổng nối tiếp.

Nếu giữ nút S2 và cắm thẻ microSD có chứa hệ điều hành, bo mạch sẽ khởi động từ microSD thay vì eMMC.



Hình 3. 9 Nút nhấn và đèn Led trên Board BeagleBone Black.

Ngay khi cấp nguồn, bo mạch bắt đầu quá trình khởi động. Các đèn LED sẽ sáng theo trình tự trong vài giây, báo hiệu trạng thái khởi động. Trong quá trình khởi động hạt nhân Linux, đèn LED có thể nhấp nháy không ổn định.

Mặc dù 4 đèn LED người dùng có thể được lập trình lại, nhưng theo mặc định, chúng hiển thị các trạng thái sau:

USER0: Nhấp nháy như nhịp tim – chỉ báo hoạt động của hạt nhân Linux

USER1: Sáng khi đang truy cập thẻ nhớ microSD

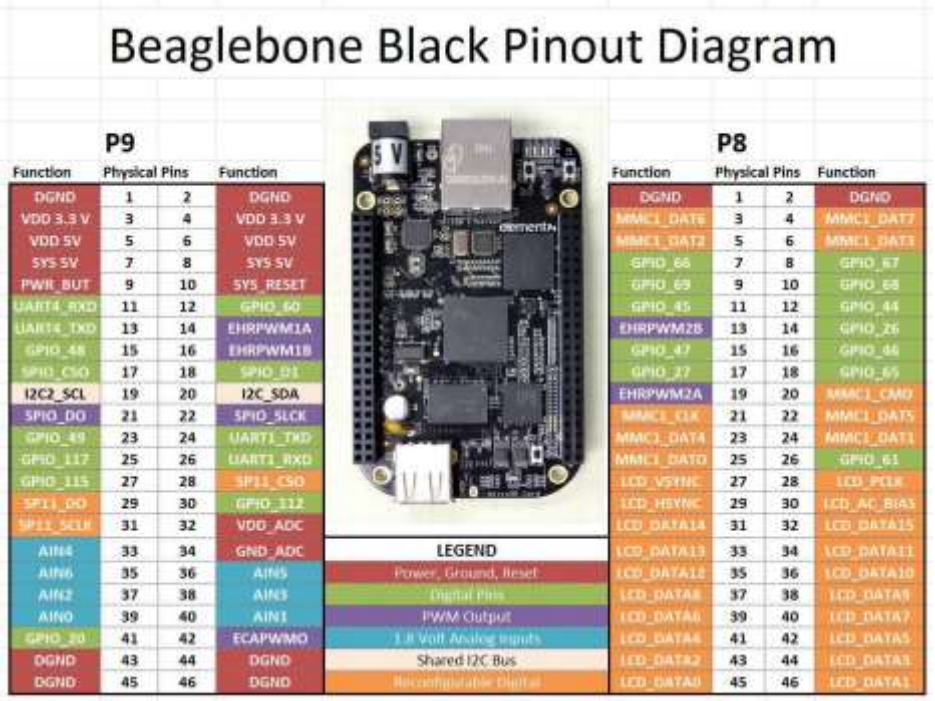
USER2: Chỉ báo hệ thống hoạt động (kernel không ở trạng thái nhàn rỗi)

USER3: Sáng khi truy cập bộ nhớ eMMC tích hợp.

Bảng 3. 1 Thông số kỹ thuật của Board BeagleBone Black.

Tính năng	Thông số
Bộ xử lý chính (CPU)	Texas Instruments Sitara AM3358BZCZ100,

	xung nhịp 1GHz, hiệu năng 2000 MIPS
Bộ xử lý đồ họa (GPU)	PowerVR SGX530, hỗ trợ 3D, 20 triệu đa giác/giây
Bộ nhớ RAM	512MB DDR3L, tốc độ bus 800MHz
Bộ nhớ lưu trữ	4GB eMMC, giao tiếp 8-bit
Bộ quản lý nguồn	PMIC TPS65217C kèm LDO bổ sung
Nguồn cấp	microUSB hoặc đầu cắm DC 5V
Kích thước PCB	3.4 x 2.1 inch (86mm x 53mm)
Giao diện mạng	Ethernet 10/100 Mbps, cổng RJ45
Giao diện USB	1 cổng USB Client miniUSB, 1 cổng USB Host Type-A
UART	UART0 truy cập qua header 6 chân TTL 3.3V
Khe cắm thẻ nhớ	Khe microSD, hỗ trợ thẻ dung lượng lớn
Xuất video	Cổng microHDMI, độ phân giải tối đa 1920x1080 @24Hz
Âm thanh	Âm thanh stereo thông qua HDMI
I/O mở rộng	Tối đa 69 GPIO, hỗ trợ I2C, SPI, PWM, UART, CAN, ADC (1.8V), LCD, v.v
LED chỉ thị	1 LED nguồn, 2 LED Ethernet, 4 LED người dùng.
Trọng lượng	~ 40g



Hình 3. 10 Sơ đồ chân Board BeagleBone Black

Bảng 3. 2 Chức năng các chân của module NODE MCU ESP8266.

Tên chức năng	Vị trí chân (P8/P9)	Chức năng
Power	P9_3, P9_4 (3.3V), P9_5, P9_6 (VDD_5V), P9_7, P9_8 (SYS_5V), P9_32 (VDD_ADC)	Cung cấp nguồn cho Board BBB
GND	P8_1, P8_2, P9_1, P9_2, P9_43,..	Nối đất
Control	P9_9 (PWR_BUT), P9_10 (SYS_RESETn)	Nút nguồn và chân reset hệ thống
ADC	P9_33–P9_40 (AIN0 đến AIN6)	Đầu vào analog 1.8V, độ phân giải 12-bit
I2C	I2C1: P9_17 (SCL), P9_18 (SDA) I2C2: P9_19 (SCL), P9_20 (SDA)	Giao tiếp cảm biến hoặc module theo chuẩn I2C
GPIO	P8_3 - P8_46 P9_11 - P9_31,..	Các chân vào/ra số (GPIO), hỗ trợ nhiều chức năng phụ
SPI	SPI0: P9_17 (CS0), P9_18 (MOSI), P9_21 (MISO), P9_22 (SCLK); SPI1: P9_28–P9_31	Giao tiếp SPI với thiết bị ngoại vi
UART	UART0: P9_24 (TX), P9_26 (RX); UART1: P9_24–P9_26; UART2: P9_21–P9_22...	TX: Chân truyền dữ liệu RX: Chân nhận dữ liệu Giao tiếp nối tiếp (serial)
SD Card (eMMC)	SD3 (SDD3), SD2 (SDD2)	Bộ nhớ trong, chứa hệ điều hành
SD Card (microSD)	MMC1 (SD card): sử dụng khi boot từ thẻ	Giao tiếp với thẻ nhớ ngoài
PWM	P9_14, P9_16, P8_13, P8_19, P8_34, P8_36, P8_45, P8_46,..	Tín hiệu PWM điều khiển motor, LED.
Ethernet	P9_23–P9_46 (chiếm các chân MII0/MII1)	Giao tiếp mạng Ethernet
Clock / Timer	P9_41A (CLKOUT2), P8_7–P8_10 (Timer)	Xung hệ thống, điều khiển ngoại vi

Mỗi chân trên header P9 & P8 của BeagleBone Black có thể đảm nhận nhiều chức năng khác nhau, được định nghĩa thông qua các chế độ (mode0 đến mode7). Trong đó, luôn có một chế độ mặc định sẽ được kích hoạt khi hệ thống khởi động.

Chúng ta hoàn toàn có thể thay đổi chế độ này bằng cách cấu hình lại pinmux trong device tree hoặc thông qua script ở người dùng.

3.2 Các thiết bị ngoại vi

3.2.1 Thiết bị khối cảm biến

3.2.1.1 Cảm biến nhiệt độ, độ ẩm SHT30

Cảm biến SHT30 được sử dụng để đo nhiệt độ và độ ẩm môi trường. Cảm biến này giao tiếp qua giao thức I2C, cho phép truyền dữ liệu nhanh và ổn định. Với độ chính xác cao, SHT30 giúp hệ thống kiểm soát môi trường chính xác hơn. Kích thước nhỏ gọn và tiêu thụ điện năng thấp phù hợp cho các ứng dụng IoT nông nghiệp. Dữ liệu từ cảm biến được dùng để điều khiển thiết bị như quạt, máy bơm.

Hình 3. 11 Cảm biến nhiệt độ, độ ẩm SHT30

Bảng 3. 3 Thông số kỹ thuật thiết bị cảm biến SHT30.

Thông số kỹ thuật	Giá trị
Dải đo nhiệt độ	-40°C đến +125°C
Sai số nhiệt độ	± 0.3°C (ở 25°C)
Dải đo độ ẩm	0% RH đến 100% RH
Sai số độ ẩm	± 3% RH (ở 25°C, 20–80% RH)
Giao tiếp	I2C
Dòng tiêu thụ trung bình	~2 µA (chế độ đo định kỳ)

3.3.1.2 Cảm biến ánh sáng BH1750

Cảm biến BH1750 được sử dụng để đo cường độ ánh sáng theo đơn vị lux, cảm biến có ADC nội và bộ tiền xử lý nên giá trị được về ra là giá trị trực tiếp cường độ ánh sáng lux mà không cần phải qua bất kỳ xử lý hay tính toán nào. Cảm biến BH1750 giao tiếp với MCU thông qua giao thức I2C.

Hình 3. 12 Cảm biến ánh sáng BH1750

Bảng 3. 4 Thông số kỹ thuật thiết bị cảm biến ánh sáng BH1750.

Thông số kỹ thuật	Giá trị
Nguồn	3~5VDC
Điện áp giao tiếp:	TTL 3.3~5VDC
Chuẩn giao tiếp	I2C
Khoảng đo	1 -> 65535 lux
Kích cỡ	21*16*3.3mm

3.2.2 Thiết bị chuyển đổi

3.2.2.1 Mạch chuyển đổi USB to TTL CP2102

CP2102 USB to TTL là một mạch chuyển đổi tín hiệu từ cổng USB của máy tính sang tín hiệu UART (TTL level) để giao tiếp với các vi điều khiển.

Mạch sử dụng chip CP2102 của hãng Silicon Labs, có độ ổn định cao, hỗ trợ tốc độ truyền dữ liệu lên tới 1 Mbps. Nó được dùng phổ biến để nạp chương trình, giao tiếp Serial, hoặc gỡ lỗi (debug) thiết bị nhúng qua UART.

Hình 3. 13 Mạch chuyển đổi USB to TTL CP2102

Bảng 3. 5 Thông số kỹ thuật module thu phát nRF24L01.

Thông số kỹ thuật	Giá trị
Chip chính	CP2102 – Silicon Labs
Giao tiếp	USB 2.0 ↔ TTL UART (TX/RX)
Tốc độ baud rate	300 bps – 1 Mbps (thường dùng 9600, 115200 bps)

Điện áp logic	3.3V hoặc 5V
Driver hỗ trợ	Windows, Linux, macOS

3.2.3 Thiết bị khối điều khiển

3.2.3.1 Máy bơm

Nhằm mục đích bơm nước vào để cung cấp hơi ẩm khi điều điệu môi trường khô khan, thiếu hơi ẩm.

Hình 3. 14 Bơm nước mini.

Bảng 3. 6. Thông số kỹ thuật máy bơm nước mini.

Thông số kỹ thuật	**Giá trị**
Điện áp hoạt động	3V - 6V DC
Dòng điện	100mA - 200mA
Lưu lượng bơm	80 - 120 L/giờ
Độ cao hút tối đa	0.5 - 1 mét

3.2.3.2 Bóng đèn

Bảng 3. 7 Thông số kỹ thuật bóng đèn sợi đốt

Thông số kỹ thuật	
Điện áp hoạt động	220VAC
Đường kính bóng	50mm
Dòng điện	4.5A
Công suất	60W

Hình 3. 15 Bóng đèn 220VAC.

3.2.4 Thiết bị khối hiển thị

3.2.4.1 LCD ILI9341

Màn hình cảm ứng LCD TFT Touch Screen 2.8 inch ILI9341 SPI Interface được sử dụng trong các ứng dụng điều khiển cảm ứng và hiển thị, màn hình sử dụng giao tiếp SPI nên rất dễ giao tiếp, giúp bạn xây dựng giao diện điều kiển cảm ứng.

Hình 3. 16 Màn hình LCD ILI9341.

Bảng 3. 8 Thông số kỹ thuật LCD ILI9341.

Thông số kỹ thuật	**Giá trị**
Điện áp sử dụng	3.3~5VDC
Điện áp giao tiếp	ILI9341 giao tiếp SPI.
Cỡ màn hình	2.8 inch
Độ phân giải	240 x 320 pixels
IC Driver cảm ứng	XPT2046 giao tiếp SPI
Kích thước hiển thị	46(W) X 65(H)mm
Kích thước toàn màn hình	50 mm

3.2.5 Thiết bị khối nguồn

3.2.5.1 Adapter 5V-2A

BeagleBone Black cần nguồn ổn định để hoạt động liên tục. Dự án sử dụng adapter 5V–2A cấp nguồn qua cổng DC jack 5.5mm x 2.1mm.

Hình 3. 17 Adapter 5V-2A

Bảng 3. 9 Thông số kỹ thuật biến áp.

Thông số kỹ thuật	Giá trị

Điện áp đầu ra	5V DC ±5%
Dòng điện cực đại	2A
Cỡ màn hình	2.8 inch
Công suất tối đa	10W
Đầu vào	100–240V AC, 50/60Hz
Đầu cắm ra	Jack tròn 5.5mm x 2.1mm
Loại adapter	Adapter nguồn rời dây, cách ly an toàn

3.3 Công cụ phát triển hệ thống

3.3.1 Yocto

Yocto Project là một framework mã nguồn mở hỗ trợ xây dựng hệ điều hành Linux tùy chỉnh cho các thiết bị nhúng có tài nguyên hạn chế như Raspberry Pi, BeagleBone Black, NXP i.MX... Việc tạo hệ điều hành nhúng thủ công rất phức tạp và dễ sai sót: từ cấu hình kernel, bootloader, thiết lập cross-compiler, thu thập và biên dịch mã nguồn, đến tạo root filesystem và image để flash.

Yocto giúp tự động hóa toàn bộ quá trình này, cho phép tùy chỉnh kernel, gói phần mềm, script khởi động; hỗ trợ tái sử dụng thông qua hệ thống layer; dễ cập nhật, bảo trì và tích hợp CI/CD nếu cần.

3.3.2 Tổng quan về Bootloader của Linux/Yocto:

Bootloader là phần mềm đầu tiên được khởi chạy khi thiết bị nhúng được cấp nguồn hoặc reset. Nó chịu trách nhiệm khởi tạo phần cứng cơ bản và chuẩn bị môi trường để hệ điều hành (kernel) được tải và chạy.

3.3.2.1 Mô hình hoạt động của Bootloader

Bootloader hoạt động ở chế độ đặc quyền (tương tự Kernel Mode trong hệ điều hành), có khả năng truy cập trực tiếp và điều khiển phần cứng. Khi thiết bị bật nguồn, bootloader thực thi các bước sau:

- Khởi tạo RAM, lưu trữ và ngoại vi cơ bản
- Tải kernel từ bộ nhớ lưu trữ (flash, SD card, eMMC...) vào RAM.
- Thiết lập các tham số khởi động và truyền quyền điều khiển cho kernel.
- Cung cấp giao diện tương tác (thường qua cổng serial) cho phép cấu hình và bảo trì hệ thống.

3.3.2.2 Quản lý quá trình khởi động và cấu hình

Bootloader quản lý chuỗi các bước khởi động (boot sequence) bằng cách:

- Đọc cấu hình và biến môi trường lưu trong bộ nhớ không thay đổi.
- Xác định thiết bị lưu trữ chứa kernel.
- Thực hiện các kiểm tra phần cứng cơ bản để đảm bảo hệ thống sẵn sàng.
- Hỗ trợ các chế độ boot linh hoạt: boot từ bộ nhớ trong, boot qua mạng, boot từ thiết bị ngoại vi...

Trong các dự án Yocto, bootloader có thể được tùy biến cấu hình thông qua các file cấu hình và recipe, giúp người phát triển dễ dàng điều chỉnh các tham số khởi động phù hợp với phần cứng mục tiêu.

3.3.2.3 Xử lý lỗi và nâng cấp phần mềm

Bootloader cũng chịu trách nhiệm xử lý các tình huống lỗi trong quá trình khởi động, ví dụ như kernel không hợp lệ, bộ nhớ lỗi hoặc thiếu kernel.

Ngoài ra, bootloader thường tích hợp các cơ chế cập nhật firmware an toàn (firmware upgrade) qua mạng hoặc qua thiết bị lưu trữ ngoài, giúp duy trì và nâng cấp phần mềm hệ thống hiệu quả.

3.3.3 Tổng quan về Kernel của Linux/Yocto:

Kernel là thành phần trung tâm của hệ điều hành nhúng Linux mà Yocto sẽ build cho thiết bị BeagleBone Black. Kernel đảm nhận vai trò quản lý tài nguyên phần cứng, cung cấp môi trường cho các tiến trình hoạt động, và là cầu nối giữa phần cứng với phần mềm ứng dụng.

3.3.3.1 Kernel và quản lý phần cứng

Kernel chạy ở chế độ đặc quyền (Kernel Mode), có quyền truy cập trực tiếp tới CPU, bộ nhớ, và các thiết bị ngoại vi của BeagleBone Black.

Kernel chịu trách nhiệm điều khiển các thiết bị thông qua driver, quản lý thiết bị ngoại vi và xử lý các tín hiệu ngắt (interrupts) để đáp ứng kịp thời các sự kiện từ phần cứng.

3.3.3.2 Quản lý tiến trình và đa nhiệm

Kernel cho phép BeagleBone Black chạy đa nhiệm, nghĩa là nhiều tiến trình có thể hoạt động cùng lúc nhờ cơ chế lập lịch CPU (scheduling).

Kernel lưu trữ trạng thái tiến trình và chuyển đổi ngữ cảnh, giúp hệ thống vận hành mượt mà, đảm bảo tiến trình dừng có thể tiếp tục chính xác khi được chạy lại.

Hệ thống đa nhiệm này rất quan trọng với các ứng dụng nhúng phức tạp cần xử lý nhiều tác vụ song song như cảm biến, giao tiếp mạng và xử lý dữ liệu.

3.3.3.3 Xử lý ngắt và đồng bộ hóa

Kernel xử lý các tín hiệu ngắt phần cứng, ví dụ như tín hiệu từ cảm biến hoặc thiết bị ngoại vi trên BeagleBone Black, đảm bảo phản hồi nhanh và chính xác.

Để tránh xung đột dữ liệu khi nhiều tiến trình hoặc driver truy cập chung tài nguyên, kernel sử dụng cơ chế đồng bộ hóa như mutex, spinlock, đảm bảo tính nguyên tử cho các vùng quan trọng.

3.3.3.4 Tùy biến kernel

Trong Linux, kernel có thể được tùy biến bằng cách:

- Chọn phiên bản phù hợp với kiến trúc phần cứng (như khi build bằng Yocto cho BeagleBone Black).
- Cấu hình qua file .config hoặc menuconfig để bật/tắt tính năng, driver thiết bị, tối ưu hiệu năng và dung lượng.
- Cross-compile tự động cho thiết bị mục tiêu.
- Kết hợp với Device Tree để mô tả và nhận diện đúng phần cứng trên board.

3.3.4 Device Tree của Linux/Yocto:

3.3.4.1 Khái quát lý thuyết về Device Tree

Trong Linux, Device Tree là cấu trúc dữ liệu mô tả phần cứng để kernel nhận diện và khởi tạo thiết bị. Thay vì mã hóa cố định trong kernel, thông tin phần cứng được lưu trong file Device Tree Blob (.dtb) và nạp khi khởi động.

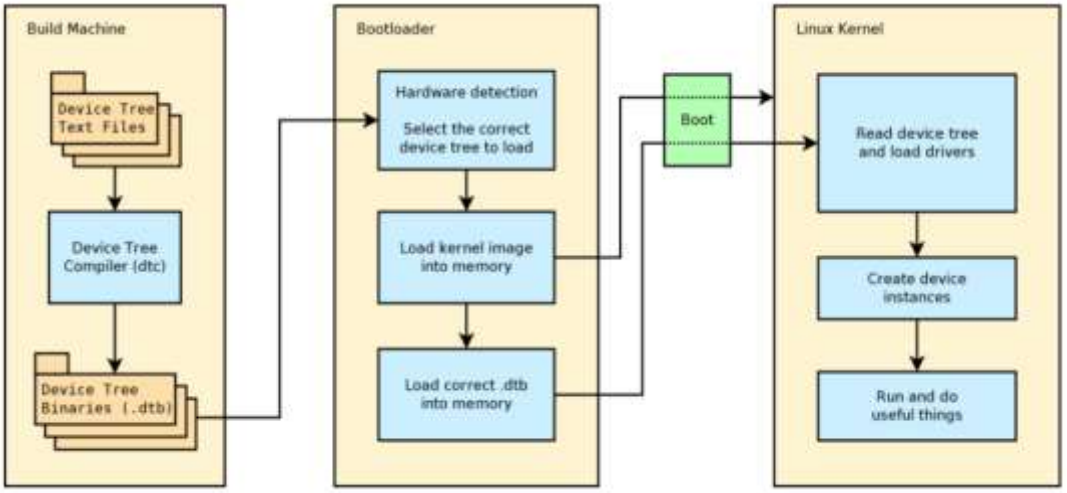
Mỗi hệ thống nhúng có file Device Tree riêng, mô tả CPU, RAM, các cổng UART, I2C, SPI, GPIO... giúp kernel ánh xạ địa chỉ và kích hoạt driver phù hợp.

3.3.4.2 Vai trò của Device Tree trong dự án

Trong dự án này, hệ điều hành được build từ Yocto để chạy trên BeagleBone Black (BBB) – một bo mạch ARM với cấu hình phần cứng cố định. Device Tree đóng vai trò quan trọng trong việc mô tả phần cứng (CPU TI AM335x, RAM 512MB, GPIO, I2C, SPI, ADC,...), giúp kernel ánh xạ và khởi tạo thiết bị chính xác.

Khi cần mở rộng phần cứng (thêm cảm biến, module...), chỉ cần chỉnh sửa Device Tree mà không phải thay đổi mã nguồn kernel. Yocto sẽ tự động biên dịch các file `.dts` thành `.dtb`, đóng gói cùng kernel và rootfs để bootloader nạp khi khởi động. Việc này giúp chuẩn hóa, tăng tính ổn định và rút ngắn thời gian phát triển.

3.3.5 Sơ đồ tổng quát quy trình khởi động hệ thống Linux/Yocto



Hình 3. 18 Sơ đồ tổng quát quy trình khởi động hệ thống Linux/Yocto

3.3.5.1 Build Machine

Đây là máy phát triển nơi build hệ điều hành bằng Yocto.

- Device Tree Files (.dts, .dtsi): Mô tả phần cứng như CPU, RAM, GPIO, I2C, SPI, cảm biến,...
- Device Tree Compiler (dtc): Biên dịch .dts thành .dtb để kernel sử dụng.
- Output: File .dtb cùng kernel và rootfs được đóng gói trong image (.wic, .ext4,...).

3.3.5.2 Bootloader (U-Boot)

- Phát hiện phần cứng và chọn đúng file .dtb (VD: am335x-boneblack.dtb).
- Nạp kernel (zImage/uImage) và .dtb vào RAM.
- Chuẩn bị sẵn sàng cho kernel khởi động (bootz/bootm).

3.3.5.3 Linux Kernel

Sau khi kernel và Device Tree được nạp vào RAM, bootloader thực hiện lệnh khởi động (như bootz hoặc bootm), gán địa chỉ .dtb, truyền các tham số khởi động (bootargs) và chuyển quyền điều khiển cho kernel. Từ đây, bootloader kết thúc vai trò.

Kernel đọc file .dtb để nhận diện phần cứng và nạp các driver tương ứng (như I2C, UART,...), sau đó tạo các thiết bị tại /dev/. Cuối cùng, kernel khởi tạo tiến trình init, mount rootfs và chuyển sang user space để chạy ứng dụng.

3.3.6 Môi trường phát triển tích hợp

Môi trường phát triển tích hợp (IDE) đóng vai trò quan trọng trong việc xây dựng và debug hệ thống nhúng trên BeagleBone Black. Hai công cụ chính được đề xuất là Visual Studio Code (VSCode) và MobaXterm, hỗ trợ tối ưu cho quy trình phát triển từ xa và quản lý thiết bị.

3.3.6.1 Visual Studio Code

Visual Studio Code (VS Code) là trình soạn thảo mã nguồn mở, nhẹ và đa nền tảng, hỗ trợ nhiều ngôn ngữ như C/C++, Python,... cùng hệ sinh thái extension phong phú. VS Code cho phép kết nối SSH đến BeagleBone Black qua plugin Remote-SSH, hỗ trợ chỉnh sửa và debug từ xa bằng GDB kết hợp với cross-compiler từ Yocto.

3.3.6.2 MobaXterm

MobaXterm là công cụ giả lập terminal đa năng cho Windows, tích hợp SSH, SFTP, X11,... giúp kết nối từ xa đến môi trường build Yocto trên Linux. Ngoài ra, MobaXterm hỗ trợ truyền file qua SFTP và quản lý thiết bị BeagleBone Black từ xa sau khi flash image, như chạy script, kiểm tra trạng thái hệ thống, hoặc gửi dữ liệu lên ThingsBoard.

3.3.7 Giao thức truyền thông & quản lý dữ liệu

3.3.7.1 SFTP (Secure File Transfer Protocol)

SFTP là giao thức truyền tệp an toàn dựa trên nền tảng SSH, đảm bảo bảo mật, toàn vẹn dữ liệu và xác thực người dùng. SFTP cho phép upload/download tệp, tạo, đổi tên, xóa thư mục và cấp quyền truy cập từ xa.

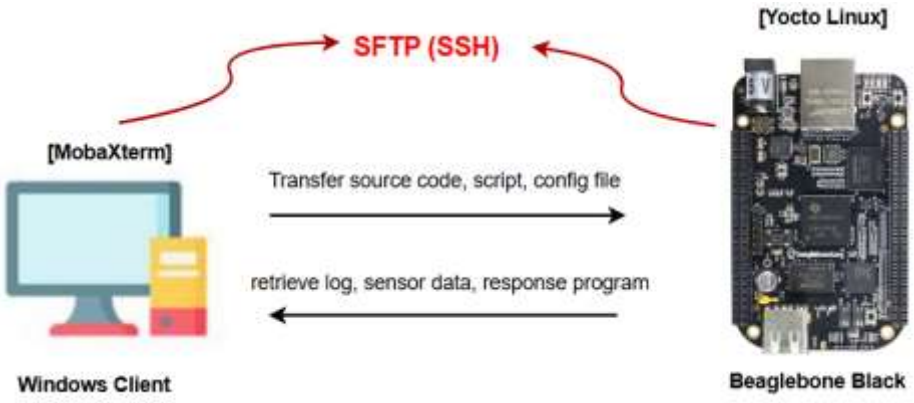
Nguyên lý hoạt động:

Client kết nối tới server qua SSH. Sau khi xác thực (mật khẩu hoặc SSH key), kênh SFTP được thiết lập, mọi thao tác truyền file đều được mã hóa. Kết nối duy trì để thực hiện nhiều lệnh liên tục.

Ứng dụng trong dự án:

- Chuyển mã nguồn, script, file cấu hình từ Windows sang BeagleBone Black.
- Lấy log, dữ liệu cảm biến từ thiết bị về máy tính.

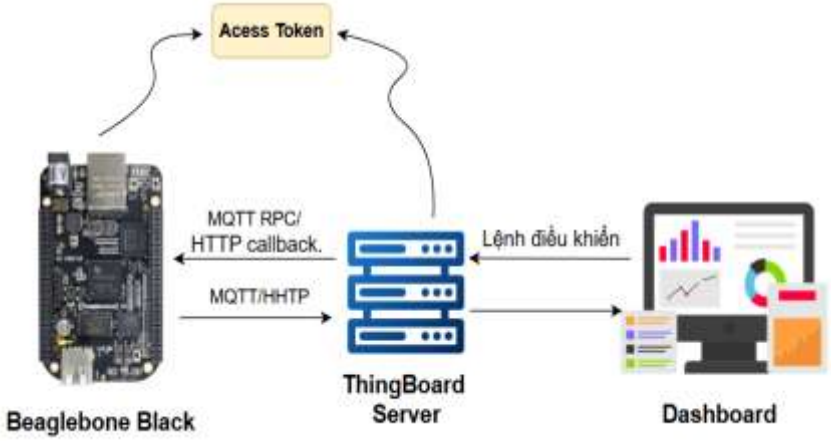
- Sử dụng qua **MobaXterm**, hỗ trợ giao diện kéo/thả hoặc dòng lệnh.



Hình 3. 19 Giao thức SFTP

3.3.7.2 ThingsBoard

ThingsBoard là nền tảng mã nguồn mở cho hệ thống IoT, hỗ trợ thu thập, xử lý, hiển thị và điều khiển dữ liệu thiết bị thông minh. Với giao thức phổ biến như MQTT, HTTP, CoAP, ThingsBoard cho phép truyền dữ liệu thời gian thựcvà điều khiển từ xa qua giao diện Dashboard trực quan.



Hình 3. 20 Quá trình gửi dữ liệu lên ThingsBoard

3.4 Kết luận chương 3

Chương 3 đã đi sâu vào việc thiết kế một hệ thống Linux nhúng trên nền tảng BeagleBone Black, bắt đầu từ phân tích phần cứng hỗ trợ như kiến trúc CPU, các chuẩn giao tiếp (I2C, SPI, UART, GPIO), thiết bị lưu trữ, kết nối mạng, và các cổng I/O. Tiếp theo là tích hợp các thiết bị ngoại vi như cảm biến, bộ điều khiển, thiết bị hiển thị và nguồn cấp. Phần cuối chương trình bày các công cụ phát triển hệ thống bao gồm Yocto, Bootloader, Kernel, Device Tree và môi trường phát triển tích hợp. Toàn

bộ chương đã cung cấp một cái nhìn toàn diện và thực tiễn để triển khai một hệ thống Linux nhúng hoàn chỉnh, là bước đệm quan trọng cho việc xây dựng, cấu hình và vận hành hệ thống trên BeagleBone Black trong các ứng dụng thực tế.

CHƯƠNG 4: THIẾT KẾ PHẦN CỨNG HỆ THỐNG

4.1 Giới thiệu chương

Chương này trình bày quá trình thiết kế và cấu hình phần cứng hệ thống nhúng sử dụng BeagleBone Black. Các thiết bị ngoại vi như cảm biến, màn hình, bộ điều khiển sẽ được kết nối và cấu hình tương thích thông qua Device Tree và kernel driver. Mục tiêu là đảm bảo phần cứng hoạt động ổn định để phục vụ cho phần mềm điều khiển ở các chương sau.

4.2 Thiết kế phần cứng

4.2.1 Yêu cầu đặt ra

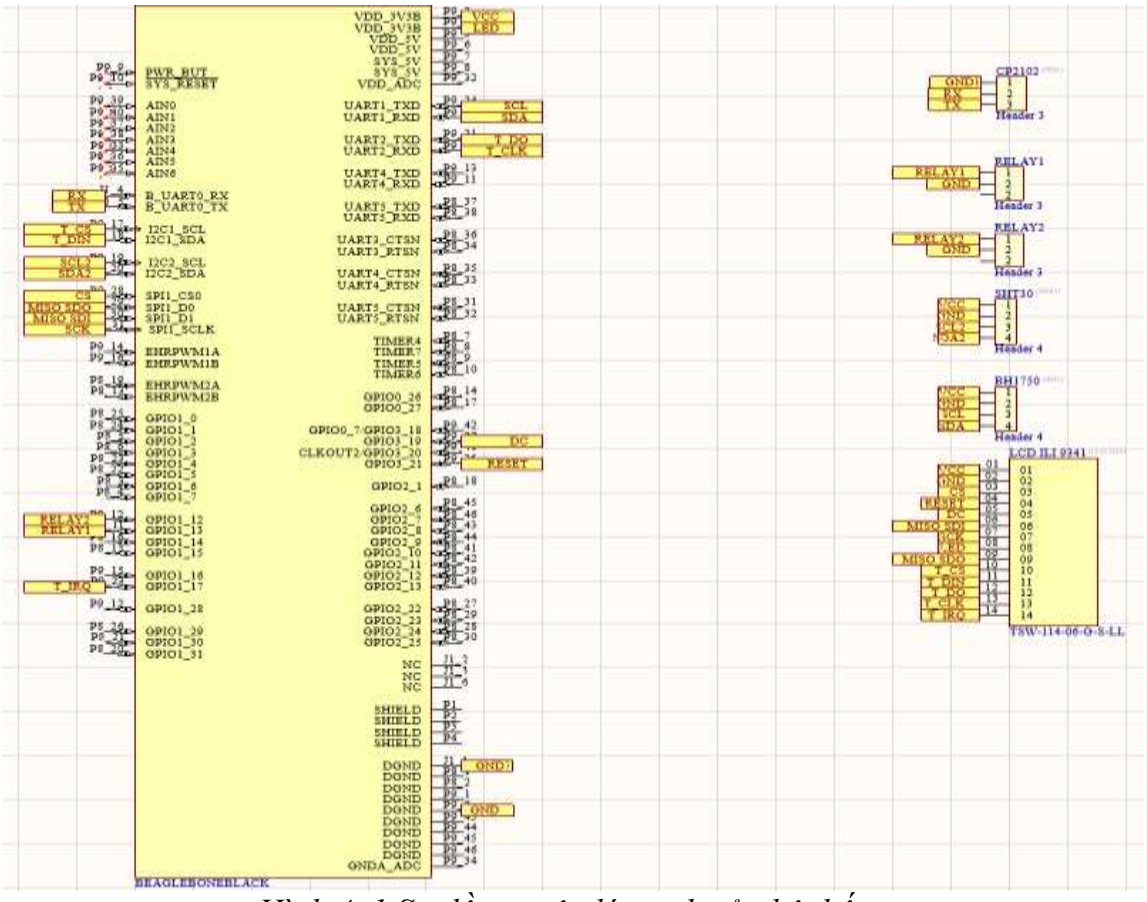
Để đảm bảo hệ thống nhúng Linux trên BeagleBone Black hoạt động ổn định và tương thích với các thiết bị ngoại vi trong mô hình Smart Garden, phần cứng cần đáp ứng các yêu cầu sau:

- Kết nối đầy đủ với cảm biến SHT30, BH1750, độ ẩm đất và điều khiển quạt, đèn, bơm qua relay; LCD ILI9341 dùng giao tiếp SPI.
- Tương thích điện áp 3.3V, cấp nguồn 5V ổn định.
- Tối ưu sử dụng GPIO/I2C/SPI, tránh xung đột pinmux, cấu hình được trong kernel.
- Dễ mở rộng và debug, hỗ trợ UART (CP2102), sơ đồ kết nối rõ ràng.
- Phù hợp với Linux, thiết bị có driver hoặc có thể port trong Yocto, khai báo đúng trong Device Tree.

4.3.2 Thiết kế mạch cho hộp điều khiển

Khi cấp nguồn 5V–2A, BeagleBone Black khởi động hệ điều hành Linux (build bằng Yocto) và hiển thị splash screen trên LCD ILI9341. Sau đó, BBB thu thập dữ liệu cảm biến (SHT30, BH1750, độ ẩm đất) qua I2C, hiển thị lên LCD và gửi lên ThingsBoard qua MQTT/HTTP. Dựa vào dữ liệu hoặc lệnh điều khiển, hệ thống kích rơ-le qua GPIO để bật/tắt bơm, đèn LED từ xa.

Sơ đồ nguyên lý hệ thống được trình bày trong hình 4.1, thiết kế bằng Altium Designer.



Hình 4. 1 Sơ đồ nguyên lý mạch của hệ thống

4.3.2.1 Porting SHT30 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

4.3.2.1.1 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

Hình 4. 2 Cấu hình bus I2C cho các chân của Beaglebone Black với SHT30

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 SHT30.

❖ Sơ đồ kết nối SHT30 với Beaglebone Black

Bảng 4. 1 Bảng sơ đồ kết nối GY-SHT30-D với BBB

Beaglebone Black	BH1750
3.3V	VCC
GND	GND
P9_24	SCL
P9_26	SDA

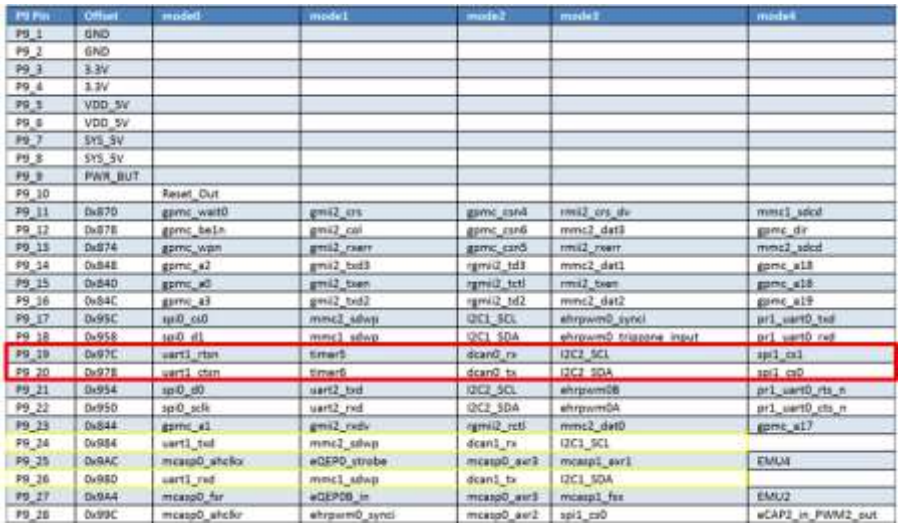
4.3.2.2 Porting BH1750 cho Beaglebone Black (BBB)

Cảm biến BH1750 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

4.3.2.2.1 Xác định bus I2C và chân 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_19 và P9_20 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

Hình 4. 3 Cấu hình bus I2C cho các chân của Beaglebone Black với BH1750

Dựa vào tài liệu schematic ta thấy rằng chân P9_19 và P9_20 không được kết nối vào bất kỳ thiết bị nào. Vì vậy ta có thể sử dụng chân P9_19 và P9_20 để kết nối với BH1750.

❖ Sơ đồ kết nối BH1750 với Beaglebone Black

Bảng 4. 1 Bảng sơ đồ kết nối BH1750 với Beaglebone Black

Beaglebone Black	BH1750
3.3V	VCC
GND	GND
P9_19	SCL
P9_20	SDA

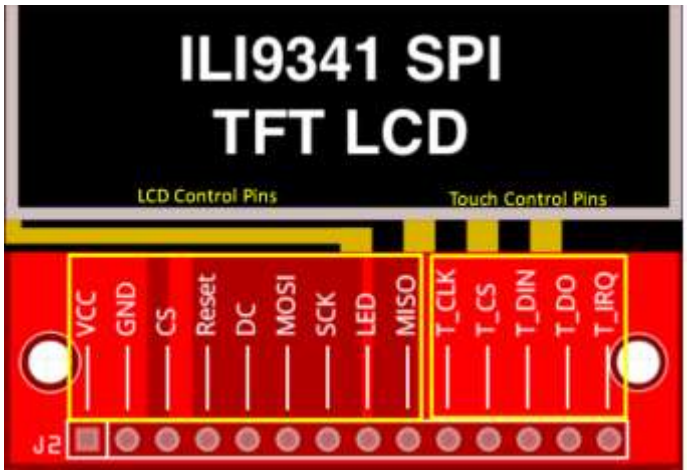
4.3.2.3 Porting LCD ILI9341 cho BeagleBone Black(BBB)

Phần giao diện người dùng (UI) của dự án sẽ được phát triển bằng framework QT. Để màn hình LCD ILI9341 hoạt động được với QT, chúng ta cần porting cho LCD hỗ trợ FrameBuffer của Linux. Ngoài ra, LCD ILI9341 sử dụng giao thức SPI để gửi nhận dữ liệu nên việc đầu tiên trong quá trình porting và xác định bus SPI và các chân của BBB sẽ sử dụng.

4.3.2.3.1 Xác định bus SPI và các chân của BBB sẽ sử dụng

Mục tiêu của bước này là chọn ra bus SPI và chân có thể sử dụng. Để làm được điều này, chúng ta cần sử dụng tài liệu sơ đồ chân của LCD ILI9431, pinmuxing table và schematic của BBB.

Dựa vào tài liệu sơ đồ chân của LCD ILI9341, để điều khiển được LCD, ngoài 4 chân SPI (SCK, MISO, MOSI, CS), chúng ta còn cần thêm 2 chân GPIO: Reset và DC



Hình 4. 4 Sơ đồ chân của LCD ILI9341

Dựa vào pinmuxing table, chúng ta thấy rằng 4 chân P9_28, P9_29, P9_30, và P9_31 có thể sử dụng cho bus SPI1 nếu cấu hình ở mode 3, và 2 chân P9_25, và P9_27 có thể sử dụng cho GPIO nếu cấu hình ở mode 7. Tham chiếu đến schematic, ta cũng thấy rằng những chân này không được kết nối vào bất kỳ thiết bị nào.

❖ Sơ đồ kết nối chân LCD ILI9341 và Beaglebone Black

Bảng 4. 2 Bảng sơ đồ kết nối chân LCD ILI9341 và Beaglebone Black

BBB	LCD ILI9341
VCC 3.3v	VCC
GND	GND
P9_28	CS
P9_25	Reset
P9_27	DC
P9_30	MOSI
P9_31	SCK
VCC 3.3v	LED
P9_29	MISO

4.3.2.3.2 Xác định bus SPI và các chân của BBB sẽ sử dụng để Touchscreen

Giao diện người dùng (UI) cần hỗ trợ cả hiển thị và tương tác cảm ứng. Để LCD ILI9341 hỗ trợ cảm ứng, cần port driver touch sử dụng giao tiếp SPI. Ngoài 4 chân SPI, cần thêm 1 chân GPIO cho tín hiệu ngắt (T_IRQ).

Các chân P9_17, P9_18, P9_21, P9_22 có thể dùng cho SPI0 (mode 0), và P9_23 cấu hình ở mode 7 để làm chân ngắt T_IRQ..

❖ Sơ đồ kết nối chân giữa touch và Beaglebone Black

Bảng 4. 3 Bảng sơ đồ kết nối chân giữa touch và Beaglebone Black

Beaglebone Black	Touch ILI9341
P9_17	T_CS
P9_18	T_DIN
P9_21	T_DO
P9_22	T_CLK
P9_23	T_IRQ

4.3.2.4 Porting Relay cho Beaglebone Black

Relay điều khiển thiết bị sử dụng giao tiếp GPIO để nhận tín hiệu điều khiển từ BeagleBone Black

4.3.2.4.1 Xác định chân GPIO điều khiển Relay

Xác định và cấu hình hai chân GPIO trên BBB để điều khiển relay 1 (bóng đèn) và relay 2 (máy bơm).

Theo schematic của BeagleBone Black: P8_11 (GPIO1_13) và P8_12 (GPIO1_12) không được kết nối tới bất kỳ thiết bị nào khác, vì vậy hoàn toàn có thể sử dụng để điều khiển relay.

❖ Sơ đồ kết nối relay với BeagleBone Black

Bảng 4. 4 Bảng sơ đồ kết nối relay với BeagleBone Black

Relay Module	Beaglebone Black
IN1	P8_11
IN2	P8_12
GND	GND
VCC	3.3 V

4.3.2.5 Layout PCB của hệ thống được thiết kế bằng phần mềm Altium Designer.

❖ **Các thành phần trên PCB**

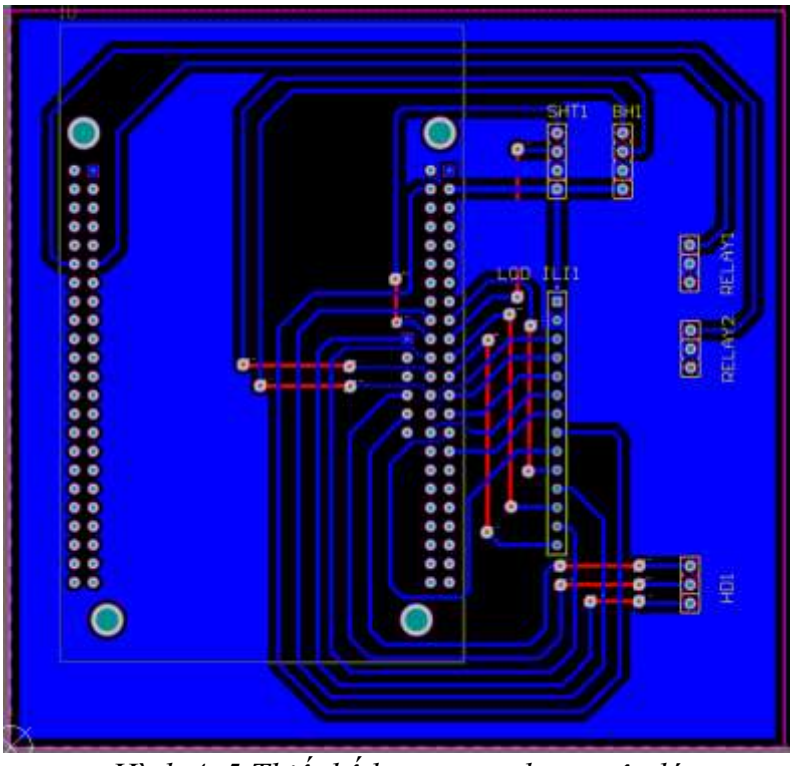
Vi xử lý BeagleBone Black được đặt ở vị trí trung tâm trên PCB nhằm thuận tiện cho việc kết nối với các thiết bị ngoại vi xung quanh như màn hình hiển thị, cảm biến và mạch điều khiển.

Màn hình LCD sử dụng giao tiếp SPI được kết nối thông qua header tiêu chuẩn, được bố trí ở vị trí bên ngoài để dễ dàng quan sát thông tin như độ ẩm đất, nhiệt độ, ánh sáng,...

Mạch điều khiển relay được đặt ở rìa PCB để dễ dàng kết nối với thiết bị như đèn sưởi, máy bơm,...

Nguồn cấp được bố trí ở góc mạch, cung cấp điện cho BeagleBone Black, màn hình LCD và các khối điều khiển.

Dưới đây là sơ đồ mạch nguyên lý PCB của hệ thống, được thiết kế trên phần mềm Altium Designer, thể hiện rõ các kết nối và bố trí linh kiện hợp lý cho ứng dụng Smart Garden.



Hình 4. 5 Thiết kế layout mạch nguyên lý

4.4 Kết luận chương 4

Chương 4 đã hoàn thành việc thiết kế phần cứng cho hệ thống Smart Garden dựa trên BeagleBone Black (BBB). Cụ thể gồm: (1) Xây dựng sơ đồ khối tổng quan cho toàn hệ thống; (2) Tích hợp các module chức năng như cảm biến SHT30 (đo nhiệt độ/độ ẩm), BH1750 (đo ánh sáng), màn hình ILI9341 (giao tiếp SPI) và relay điều khiển qua GPIO; (3) Cấu hình chính xác các bus giao tiếp I2C/SPI và xác định rõ các chân kết nối vật lý trên BBB.

Ngoài ra, sơ đồ nguyên lý và layout PCB cũng đã được thiết kế hoàn chỉnh bằng phần mềm Altium Designer, đảm bảo tính khoa học trong bố trí linh kiện, đường mạch hợp lý và sẵn sàng cho chế tạo. Kết quả đạt được là một hệ thống phần cứng hoàn chỉnh, ổn định, sẵn sàng cho bước triển khai phần mềm điều khiển và kiểm thử thực tế.

CHƯƠNG 5: THIẾT KẾ PHẦN MỀM HỆ THỐNG

5.1 Giới thiệu chương

Trong chương này, nội dung trình bày tập trung vào quá trình thiết kế phần mềm hệ thống cho nền tảng Linux nhúng trên BeagleBone Black, phục vụ việc giám sát và điều khiển môi trường trồng trọt thông minh (Smart Garden). Các nội dung bao gồm: xây dựng firmware hệ điều hành bằng Yocto, cấu hình kernel, Device Tree, lập trình điều khiển thiết bị và truyền dữ liệu lên nền tảng giám sát từ xa ThingsBoard. Phần mềm được thiết kế để đảm bảo tính tự động hóa, linh hoạt và khả năng mở rộng.

5.2 Thiết kế phần mềm

5.2.1 Mục tiêu

Phần mềm được xây dựng nhằm hiện thực hóa khả năng:

- Giám sát thời gian thực các thông số môi trường như nhiệt độ, độ ẩm không khí, ánh sáng, độ ẩm đất.
- Điều khiển thông minh các thiết bị trong hệ thống như đèn chiếu sáng, máy bơm nước, quạt làm mát.
- Kết nối và truyền dữ liệu lên cloud thông qua nền tảng IoT ThingsBoard.
- Hiển thị giao diện người dùng cục bộ thông qua màn hình LCD SPI.

5.2.2 Yêu cầu đặt ra

Hiển thị giá trị cảm biến và trạng thái thiết bị lên màn hình LCD. Phần mềm hệ thống cần đảm bảo các yêu cầu sau:

- Kết nối mạng: Kết nối Internet qua Ethernet/WiFi USB, giao tiếp ThingsBoard bằng HTTP REST API, tự động reconnect khi mất mạng.
- Xử lý cảm biến: Đọc dữ liệu định kỳ từ cảm biến SHT30, BH1750 qua I2C, lưu tạm và gửi lên ThingsBoard.
- Điều khiển thiết bị: Điều khiển bơm, đèn qua GPIO/relay, tự động theo ngưỡng hoặc theo lệnh từ dashboard.
- Hiển thị LCD: Hiển thị thông số môi trường, trạng thái thiết bị trên ILI9341, có splash screen và chuyển giao diện.
- Phần mềm viết bằng C, chạy như service khởi động cùng hệ thống, tối ưu cho Yocto trên BeagleBone Black.

5.3 Chuẩn bị và cấu hình môi trường Yocto

5.3.1 Yêu cầu hệ thống tối thiểu

Để tiến hành biên dịch hệ điều hành nhúng cho BeagleBone Black (BBB) bằng Yocto, hệ thống máy tính dùng để build cần đảm bảo các yêu cầu sau:

- Ổ cứng: Tối thiểu 100 GB dung lượng trống (nên sử dụng ổ SSD để tăng tốc độ build).

- Thực hiện các commands sau để tạo swap 16GB:

```
sudo dd if=/dev/zero of=/swapfile bs=1M count=16384
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile

sudo swapon --show
```

5.3.2 Cài đặt công cụ cần thiết và tải source Poky

- Hệ điều hành: Ubuntu 20.04 LTS để chạy Yocto

- Cài đặt các gói phần mềm cần thiết để chuẩn bị cho môi trường build:

```
sudo apt update
sudo apt install gawk wget git-core diffstat unzip texinfo gcc-multilib \
  build-essential chrpath socat cpio python3 python3-pip python3-pexpect \
  xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa \

  libsdl1.2-dev pylint xterm
```

- Tạo thư mục chứa Yocto và tải source Poky:

```
mkdir -p yocto
cd yocto
git clone git://git.yoctoproject.org/poky

git checkout -t origin/kirkstone -b my-kirkstone
```

➢ Chúng ta sử dụng branch **kirkstone** vì đây là phiên bản LTS (Long Term Support) ổn định, được cập nhật lâu dài, có nhiều tài liệu và hỗ trợ tốt.

5.3.3 Khởi tạo môi trường build và cấu hình

Chạy lệnh sau để tạo môi trường build:

```
$ source oe-init-build-env
```

Sau khi chạy lệnh ở trên Yocto sẽ in ra màn hình với các sự lựa chọn phổ biến để build image.

```
### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-full-cmdline
    core-image-sato
    core-image-weston
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

Other commonly useful commands are:
 - 'devtool' and 'recipetool' handle common recipe tasks
 - 'bitbake-layers' handles common layer tasks
 - 'oe-pkgdata-util' handles common target package tasks
```

Hình 5. 1 Sau khi khởi tạo môi trường build

❖ Đồng thời thiết lập và tạo ra thư mục *build* cùng với các file cấu hình cần thiết

- Chỉnh sửa file *conf/local.conf:*

```
MACHINE ?= "beaglebone-yocto"
BB_NUMBER_THREADS = "12"
PARALLEL_MAKE = "-j12"
IMAGE_INSTALL:append = " openssh"
```

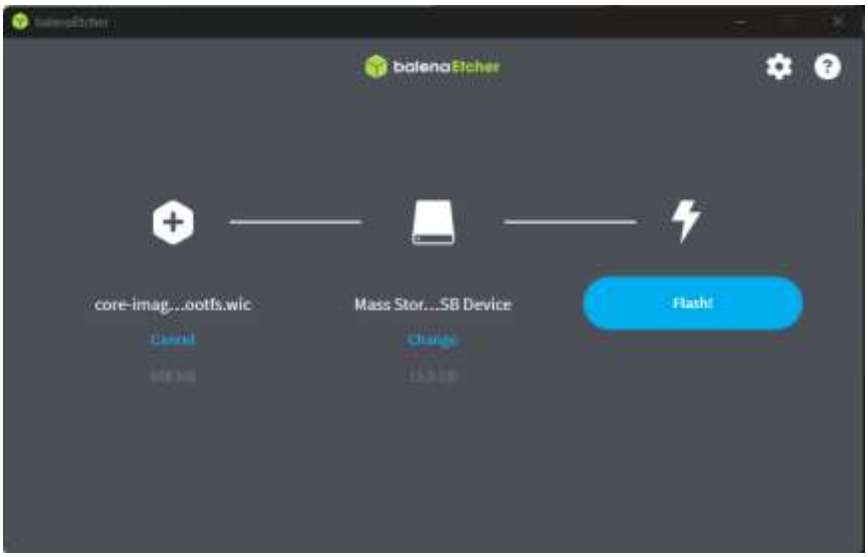
➢ Chúng ta sẽ build image **core-image-full-cmdline**, một bản image có giao diện dòng lệnh đầy đủ, không có GUI, phù hợp với hệ thống nhúng:

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

5.3.4 Kiểm tra kết quả build và flash image vào thẻ nhớ

Sau khi build xong output file sẽ ở trong folder tmp/deploy/images/beaglebone-yocto, gồm có:

- .ext4, .wic: Hệ thống file
- .dtb: Device Tree
- .uImage, .zImage: Kernel
- .tar.bz2, .hddimg,...



Hình 5. 2 Flash image vào thẻ nhớ

Sử dụng file core-image-full-cmdline-beaglebone-yocto.wic với balenaEtcher để flash image vào SD card.

5.4 Tích hợp cảm biến nhiệt độ – độ ẩm SHT30

5.4.1 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.

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
```

Trong cấu hình pinctrl, mỗi thiết lập được liên kết với một chức năng cụ thể và thể hiện trong file pinctrl-handles. Dựa vào nội dung file này, có thể kiểm tra trạng thái cấu hình của các chân. Hiện tại, hai chân P9_24 và P9_26 trên BeagleBone Black vẫn chưa được cấu hình.

Để thực hiện cấu hình pinctrl cho chân P9_24 và P9_26 chúng ta truy cập vào '*build/tmp/work-shared/beaglebone-yocto/kernelsource/arch/arm/boot/dts/am335x-boneblack.dts*'và khai báo các thiết lập cho pinctrl và kích hoạt i2c1, đồng thời định nghĩa node cho cảm biến SHT30 với địa chỉ 0x44.

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.

- Thực hiện rebuild lại kernel bằng command sau:

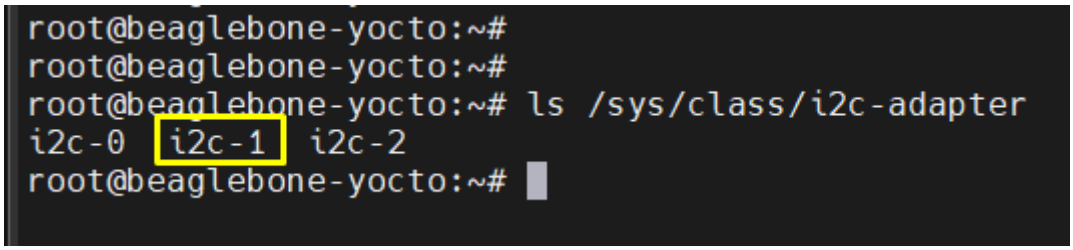
```
$ bitbake -f -c compile linux-yocto
$ bitbake -c deploy linux-yocto
$ bitbake core-image-full-cmdline
```

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

5.4.2 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



Hình 5. 3 Bus I2C 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.

5.4.3 Kiểm tra address của SHT30 và viết chương trình demo

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

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

Hình 5. 4 Kiểm tra Address của SHT30

5.5 Tích hợp cảm biến ánh sáng BH1750

5.5.1 Cấu hình pinctrl trong device tree

Tương tự như cảm biến độ ẩm nhiệt độ:

Mục tiêu của bước này là cấu hình pinctrl cho chân P9_19 và P9_20 trong device tree hoạt động ở mode 3.

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'

Nếu chưa có, thêm đoạn cấu hình sau vào: *build/tmp/work-shared/beaglebone-yocto/kernel-source/arch/arm/boot/dts/am335x-boneblack.dts và* khai báo các thiết lập cho pinctrl và kích hoạt i2c2, đồng thời định nghĩa node cho cảm biến BH1750 với địa chỉ 0x23

- Thực hiện rebuild lại kernel bằng command sau:

```
$ bitbake -f -c compile linux-yocto
$ bitbake -c deploy linux-yocto
$ bitbake core-image-full-cmdline
```

5.5.2 Kiểm tra cấu hình

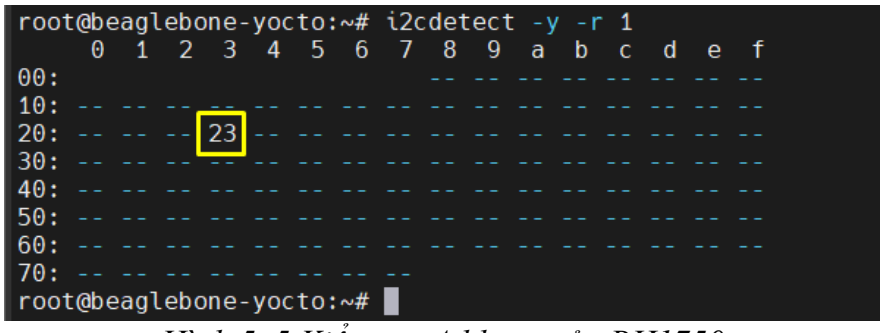
Tương tự như cảm biến SHT30:

Mục tiêu của bước này là kiểm tra xem bus I2C2 đã được enable và cấu hình pinctrl cho chân P9_19 và P9_20 đã chính xác chưa.

5.5.3 Kiểm tra address của BH1750 và viết chương trình demo

Mục tiêu của bước này kiểm tra được address của BH1750 và lấy address đó để viết chương trình demo đọc được dữ liệu lux từ BH1750 và in ra màn hình console.

Chúng ta có thể kiểm tra address của BH1750 bằng command 'i2cdetect -y -r '. 0x23 được trả về sau command chính là address của cảm biến.



Hình 5. 5 Kiểm tra Address của BH1750

Khi cảm biến BH1750 được cấp nguồn, nó sẽ mặc định chuyển sang chế độ Power Down. Trong chế độ này, cảm biến không hoạt động và sẽ chờ lệnh điều khiển

tiếp theo. Để đọc được giá trị lux, chúng ta cần đưa cảm biến vào chế độ đo (measurement mode) bằng cách gửi lệnh đo thích hợp.

5.6 Tích hợp màn hình LCD ILI9341 với BeagleBone Black

5.6.1 Cấu hình pinctrl cho màn hình LCD ILI9341 trong device tree

Mục tiêu của bước này là cấu hình 4 chân SPI1: P9_28, P9_29, P9_30, và P9_31 hoạt động ở mode 3, và 2 chân GPIO: P9_25, và P9_27 hoạt động ở mode 7

Kiểm tra xem 6 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'*

Dựa vào file pinctrl-handles, có thể xác định các node cấu hình đang được áp dụng trên BeagleBone Black. Hiện tại, các chân SPI1 (P9_28, P9_29, P9_30, P9_31) và GPIO (P9_25, P9_27) đang được cấu hình bởi node mcasp0_pins trong file am335x-boneblack-hdmi.dtsi bởi node mcasp0_pins nằm trong am335x-boneblack-hdmi.dtsi.

Vì vậy chúng ta sẽ cần disable node mcasp0 là node chức năng đang sử dụng node mcasp0_pins nằm trong am335x-boneblack-hdmi.dtsi.

```
97 &mcasp0 {
98         #sound-dai-cells = <0>;
99         pinctrl-names = "default";
100        pinctrl-0 = <&mcasp0_pins>;
101 //     status = "okay";
102        status = "disable";
103        op-mode = <0>;  /* MCASP_IIS_MODE */
104        tdm-slots = <2>;
105        serial-dir = <  /* 0: INACTIVE, 1: TX, 2: RX */
106                        0 0 1 0
107                >;
108        tx-num-evt = <32>;
109        rx-num-evt = <32>;
110 };
```

Hình 5. 6 Disable node mcasp0

Tiếp tục, để thực hiện cấu hình pinctrl chúng ta truy cập vào file: *'build/tmp/work-shared/beaglebone-yocto/kernel-source/arch/arm/boot/dts/am335x-boneblack.dts' và khai báo cấu hình cho các bus SPI của màn hình LCD ILI9341.*

5.6.2 Porting LCD ILI9341 để hỗ trợ FrameBuffer

Mục tiêu của bước này là enable tinydrm_ili9341 driver và thêm node tương ứng vào device tree để LCD ILI9341 có thể hoạt động với FrameBuffer.

Trước tiên, chúng ta thêm node vào device tree bằng cách truy cập vào file *'build/tmp/work-shared/beaglebone-yocto/kernel-source/arch/arm/boot/dts/am335x-boneblack.dts'*.

Và thêm đoạn code sau vào node &spi1:

```
&spi1 {
        #address-cells = <1>;
        #size-cells = <0>;

        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_spi1_pins>;
        lcd@0{
                compatible = "adafruit,yx240qv29";
                reg = <0>;
                pinctrl-names = "default";
                pinctrl-0 = <&bb_lcd_pins>;
                spi-max-frequency = <24000000>;
                rotation = <0>;
                dc-gpios    = <&gpio3 19 0>;   // lcd dc    P9.27 gpio3[19]
                reset-gpios = <&gpio3 21 0>;   // lcd reset P9.25 gpio3[21]
                status = "okay";
        };
};
```

Trong đó:

- spi-max-frequency = <24000000> : Tốc độ truyền dữ liệu SPI, ở đây là 24MHz. Tốc độ cao giúp hiển thị nhanh hơn
- rotatation = <90> : Xoay màn hình theo góc 90 độ để hiển thị đúng hướng.
- reset-gpios và dc-gpios :Khai báo các chân GPIO dùng để điều khiển

Tiếp theo, sử dụng kernel menuconfig để enable *tinydrm_ili9341 driver:*

```
$ bitbake -c menuconfig -f virtual/kernel
```

- Lúc này trên màn hình console sẽ mở ra giao diện tương tác:

Hình 5. 7 Giao diện tương tác để enable tinydrm_ili9341 drive

❖ Thực hiện rebuild lại kernel và image bằng command sau:

```
$ bitbake -f -c compile linux-yocto
$ bitbake -c deploy linux-yocto
$ bitbake core-image-full-cmdline
```

➢ Tiếp theo, cần flash image vào sdcard và kiểm tra hoạt động của LCD

5.6.3 Kiểm tra hoạt động của LCD

- Mục tiêu của bước này là kiểm tra hoạt động của LCD ILI9341

Trong quá trình boot, nếu node lcd@0 trong device tree tương thích với tinydrm_ili9341driver thì trên LCD ILI9341 cũng sẽ hiển thị console

fif 

Hình 5. 8 Hiển thị trên màn hình LCD ILI9341

5.6.4 Cấu hình Pinctrl trong Device Tree cho Porting Touchscreen trên BBB

Cần kiểm tra các chân P9_17, P9_18, P9_21, P9_22, P9_23 xem đã được cấu hình trong device tree chưa. Nếu chưa, có thể sử dụng trực tiếp mà không cần disable node nào.

Để cấu hình các chân này thông qua pinctrl, truy cập file: build/tmp/work-shared/beaglebone-yocto/kernel-source/arch/arm/boot/dts/am335x-boneblack.dts

Sau đó, viết code khai báo cấu hình pin tương ứng vào phần phù hợp trong device tree. Các chân sẽ được ánh xạ chế độ (mode) và thiết lập đầu vào/ra phù hợp để phục vụ giao tiếp SPI, GPIO hoặc mục đích cụ thể của dự án

5.6.5 Porting Touch ILI9341 để hỗ trợ Input TouchScreen

Mục tiêu của bước này là enable TOUCHSCREEN_ADS7846 driver và thêm node tương ứng vào device tree để touch có thể hoạt động với Input TouchScreen.

Trước tiên, chúng ta thêm node vào device tree bằng cách truy cập vào file *'build/tmp/work-shared/beaglebone-yocto/kernel-source/arch/arm/boot/dts/am335x-boneblack.dts' và thêm các khai báo cấu hình bus Spi0 của board cho màn hình.*

Tiếp theo, sử dụng kernel menuconfig để enable touchscreen_ads7846 driver bằng command sau:

```
$ bitbake -c menuconfig -f virtual/kernel
```

Dùng phím mũi tên để di chuyển và nhấn Enter để chọn. Làm theo các bước sau để enable touchscreen_ads7846 driver

```
-> Device Drivers
     -> Input device support
           -> Generic input layer (needed for keyboard, mouse, ...) (INPUT [=y])
                 -> Touchscreens (INPUT_TOUCHSCREEN [=y])
                       -> ADS7846/TSC2046/AD7873 and AD(S)7843 based
touchscreens (TOUCHSCREEN_ADS7846 = [y])
```

Tiếp theo, thêm '*evtest'* package để có thể sử dụng command line *'evtest'* cho việc debug. Sau đó, rebuild lại image để update thay đổi.

```
$ bitbake -f -c compile linux-yocto
$ bitbake -c deploy linux-yocto
$ bitbake core-image-full-cmdline
```

Tiếp theo, flash image vào SDcard và kiểm tra hoạt động của touch

5.7 Thiết kế và phát triển ứng dụng với Qt trên BeagleBone Black

5.7.1 Giới thiệu

Trong hệ thống giám sát môi trường ứng dụng BeagleBone Black, Qt được lựa chọn làm framework phát triển giao diện đồ họa. Qt hỗ trợ phát triển ứng dụng trực quan, dễ sử dụng, đồng thời tương thích tốt với hệ thống Linux nhúng được build bởi Yocto.

5.7.2 Mục tiêu sử dụng Qt

- Tạo giao diện hiển thị các thông số môi trường như nhiệt độ, độ ẩm, ánh sáng.
- Cho phép người dùng tương tác trực tiếp trên màn hình cảm ứng.
- Phát triển ứng dụng trên PC bằng QtCreator, cross-compile và triển khai trực tiếp lên BeagleBone Black qua SSH.

5.7.3 Xây dựng Qt SDK với Yocto

Để tạo SDK đầy đủ hỗ trợ Qt, ta dùng một recipe được cung cấp bởi Yocto là '*meta-toolchain-qt5'*. Recipe này sẽ:

- Gồm đầy đủ Qt libraries, Qt tools (qmake, moc, uic…)
- Gồm cross-compiler tương thích
- Gồm file cấu hình toolchain (sẵn sàng dùng cho Qt Creator)
- Tự động thực hiện populate_sdk

❖ Lệnh build Qt SDK như sau:

```
bitbake meta-toolchain-qt5
```

➢ Sau khi build xong, chúng ta sẽ tìm thấy Qt SDK trong thư mục '*tmp/deploy/sdk'*.

5.7.4 Cài đặt Qt SDK (Installing Qt SDK)

Sau khi đã build thành công SDK Qt với lệnh '*bitbake meta-toolchain-qt5'*, một trình cài đặt SDK (.sh) sẽ được tạo ra tại:

```
tmp/deploy/sdk/poky-glibc-x86_64-meta-toolchain-qt5-cortexa8hf-neon-
beaglebone-yocto-toolchain-4.0.26.sh
```

Trình cài đặt sẽ hỏi nơi bạn muốn cài SDK. Nếu thư mục mặc định /opt/poky/4.0.26 phù hợp, chỉ cần nhấn Enter hai lần để tiếp tục.

Nếu không, bạn có thể nhập một đường dẫn tùy chọn, ví dụ:

```
$ <home>/workspace/bbb/qt-sdk
```

Sau đó nhấn Enter hai lần để hoàn tất.

Trước khi biên dịch bất kỳ ứng dụng nào bằng SDK, bạn cần thiết lập môi trường bằng cách chạy:

```
$ source /opt/poky/4.0.26/environment-setup-cortexa8hf-neon-poky-linux-gnueabi
```

5.7.5 Using Qt SDK with Qt Creator

Đầu tiên, chúng ta cần cài đặt Qt Creator bằng command sau:

```
sudo apt update -y
sudo apt install -y qtcreator qtbase5-dev qt5-qmake cmake -y
```

Sau đó export Qt SDK và chạy Qt Creator bằng command sau, mục tiêu việc này là để Qt Creator có thể sử dụng các biến môi trường sau khi export Qt SDK

```
$ source /opt/poky/4.0.26/environment-setup-cortexa8hf-neon-poky-linux-gnueabi
$ qt-creator
```

Bây giờ, chúng ta sẽ cấu hình Qt Creator có thể truy cập được vào BBB bằng SSH. Trong QtCreator, chúng ta mở hộp thoại Tools | Devices | Devices và nhấn nút Add. Chúng ta chọn tùy chọn Generic Linux Device từ hộp thoại Device Configuration Selection và nhấn nút Start Wizard. Ở bước đầu tiên của trình hướng dẫn (wizard) là Connection, chúng ta nhập BBB làm tên cấu hình, 192.168.2.9 làm địa chỉ IP của thiết bị đích và root làm tên người dùng để đăng nhập vào thiết bị.

5.7.6 Thiết lập Qt Creator

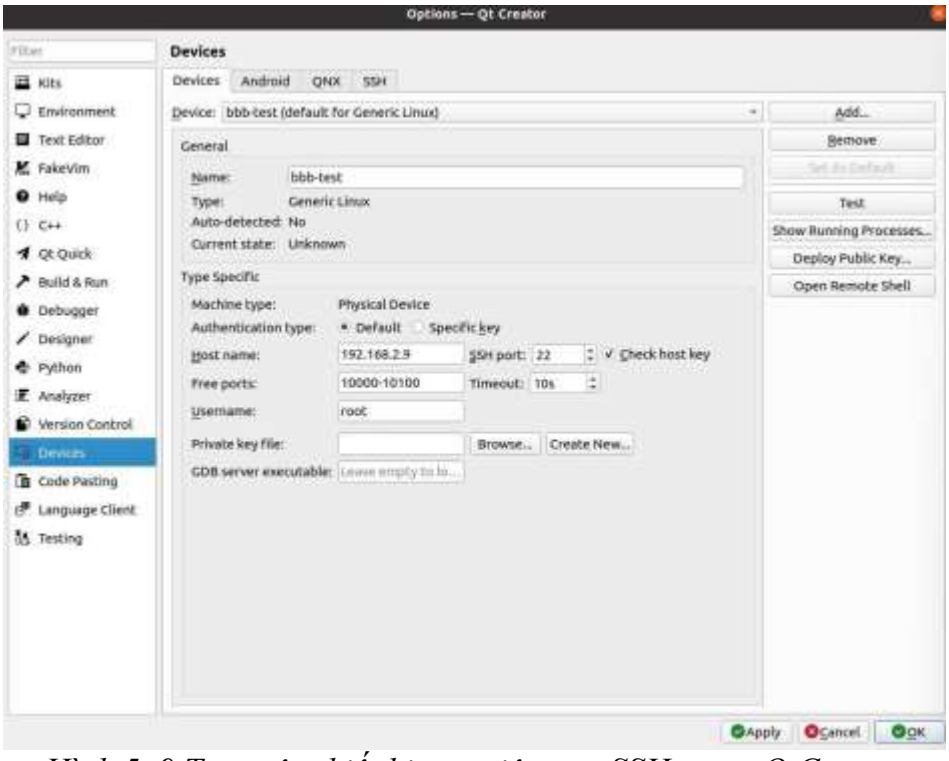
5.7.6.1 Truy cập thiết bị mục tiêu qua SSH trong QtCreator

Để QtCreator có thể triển khai và chạy ứng dụng trên BeagleBone Black, cần thiết lập kết nối SSH giữa PC phát triển và thiết bị.

QtCreator sử dụng giao thức SSH (sftp hoặc rsync) để sao chép file và thực thi ứng dụng từ xa. Cách đơn giản nhất là đảm bảo thiết bị và PC phát triển nằm trong cùng một mạng con (subnet). Trong QtCreator:

- Vào Tools → Devices → Nhấn Add
- Tên thiết bị → Địa chỉ IP của thiết bị (192.168.2.9) → Tên đăng nhập (thường là *root*)

Sau khi hoàn tất, QtCreator có thể tự động sao chép, biên dịch và chạy ứng dụng trên thiết bị từ xa thông qua SSH.



Hình 5. 9 Truy cập thiết bị mục tiêu qua SSH trong QtCreator

Chúng ta nhấn nút Next để chuyển sang bước thứ hai của trình hướng dẫn là Key Deployment. Bởi vì f/w của chúng ta không sử dụng password nên chúng ta tiếp tục nhấn Next

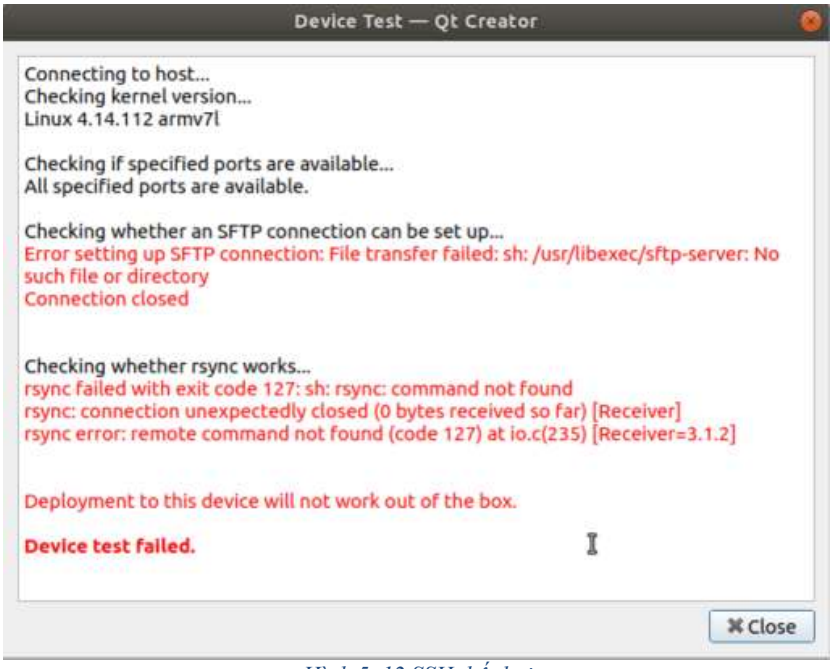
Hình 5. 10 Thiết lập Key Deployment

Đóng hộp thoại bật lên và nhấn nút *Next* theo trên hộp thoại bên dưới. QtCreator hiển thị bước hướng dẫn cuối cùng.

Hoàn tất thiết lập SSH bằng cách nhấn nút Finish . Nếu thiết lập SSH đúng, chúng ta sẽ thấy hộp thoại sau.

Hình 5. 11 SSH thành công

Nếu có gì sai sót trong quá trình thiết lập, chúng ta sẽ thấy hộp thoại báo lỗi.

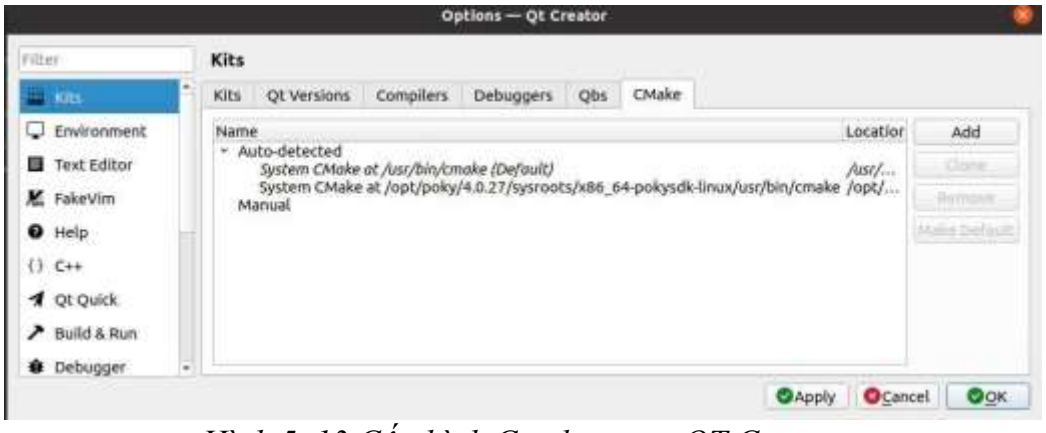


Hình 5. 12 SSH thất bại

5.7.6.2 Cấu hình CMake trong QtCreator từ máy chủ PC

QtCreator mặc định sử dụng CMake tích hợp từ Qt SDK. Tuy nhiên, nếu bạn đã cài đặt một phiên bản CMake khác trên hệ thống (như /usr/local/bin/cmake), bạn có thể tùy chỉnh lại đường dẫn sử dụng:

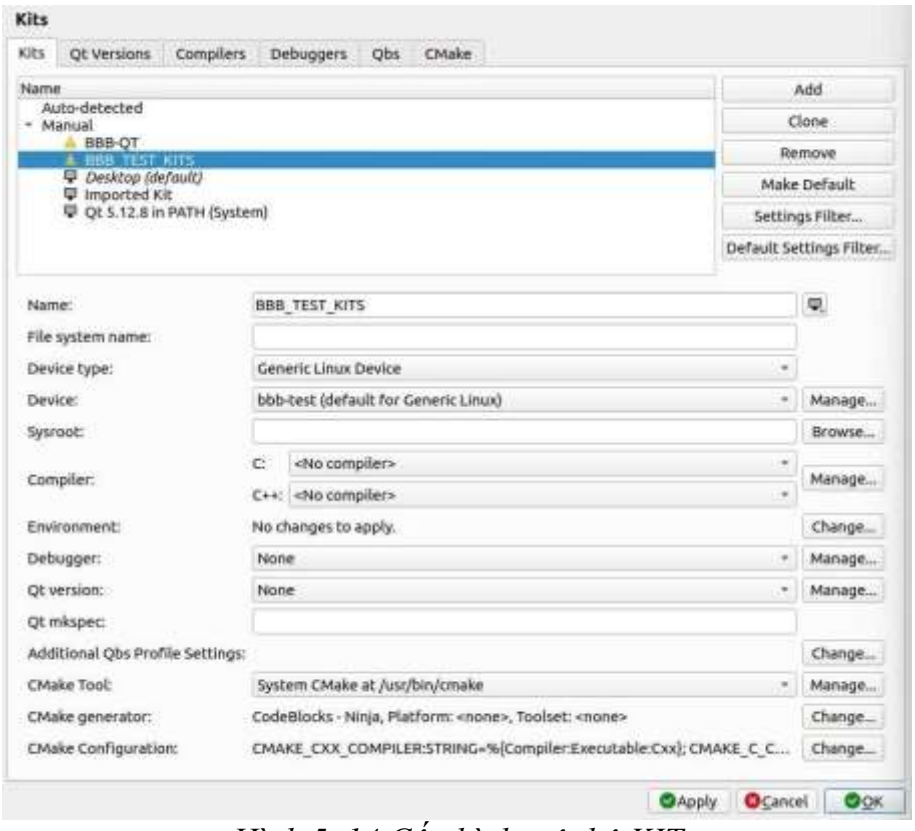
- Mở Tools → Options → Kits → CMake
- Nhấn Add, sau đó chọn đúng đường dẫn đến file thực thi của CMake bạn muốn dùng.



Hình 5. 13 Cấu hình Cmake trong QT Creator

5.7.6.3 Cấu hình một bộ KIT

Chúng ta chuyển sang trang tab Tools | Options | Kits | Kits để định nghĩa một bộ từ Qt Version, Compilers và CMake. Biểu mẫu đã điền trông như sau.



Hình 5. 14 Cấu hình một bộ KIT

- Nhập tên cho bộ công cụ: BBB_TEST_KITS.
- Device type = Generic Linux Device
- Device = bbb-test
- CMake Tool: System Cmake at /usr/bin/cmake

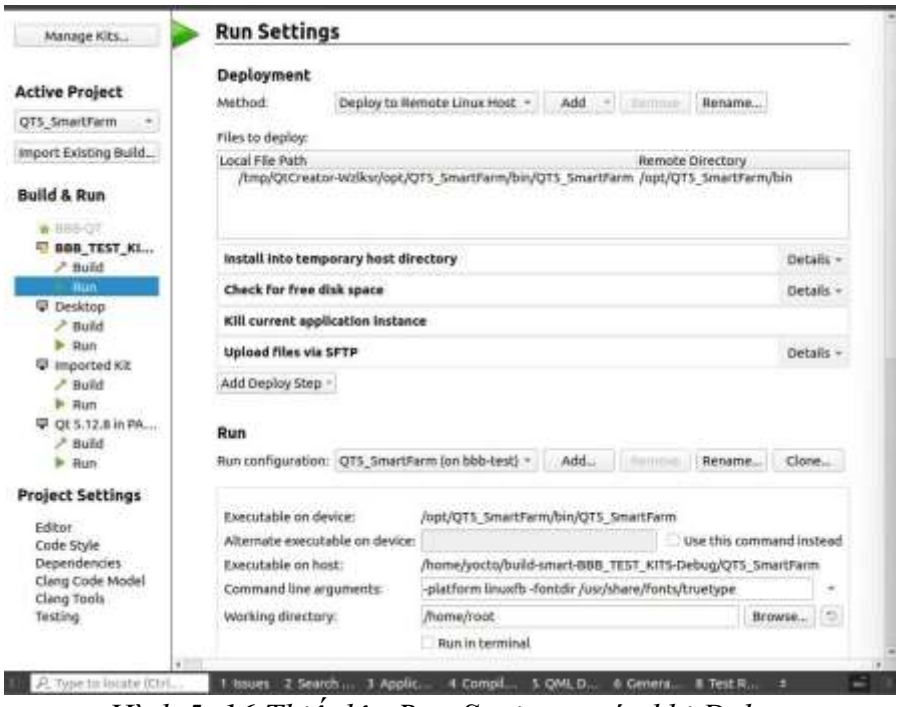
- Debugger = None
- Không thiết lập Sysroot , Compiler , Qt version và Qt mkspec , vì chúng được xử lý bởi tệp toolchain CMake.

5.7.6.4 Chạy ứng dụng

Chúng ta chuyển sang bộ công cụ *Beaglebone Black* bằng cách mở trình chuyển đổi cấu hình ở phía dưới thanh công cụ bên trái và chọn BBB_TEST_KITS làm bộ công cụ . *Debug* làm cấu hình dựng và *Custom Executable (trên Beaglebone Black)* làm cấu hình chạy.

Hình 5. 15 Chạy Debug ứng dụng

QtCreator sử dụng giao thức SFTP để tự động sao chép file thực thi từ máy phát triển sang thư mục đích trên thiết bị. Quá trình chạy cho phép chỉ định tham số dòng lệnh như *-platform linuxfb* để cấu hình giao diện hiển thị. Thư mục làm việc mặc định trên thiết bị là */home/root*. Cấu hình này hỗ trợ triển khai, thử nghiệm và debug ứng dụng Qt nhúng một cách tiện lợi.



Hình 5. 16 Thiết lập Run Setting trước khi Debug

5.7.6.5 Thiết kế Ui cho màn hình ILI9341

Hình 5. 17 Thiết kế Ui cho màn hình ILI9341

Hình ảnh thể hiện bố cục giao diện người dùng (UI) của ứng dụng Smart Farm với các thành phần như nhãn hiển thị nhiệt độ, độ ẩm, ánh sáng và hai nút điều khiển "Light" và "Pump".

5.8 Thiết kế ThingsBoard

5.8.1 Thiết lập và cấu hình ThingsBoard

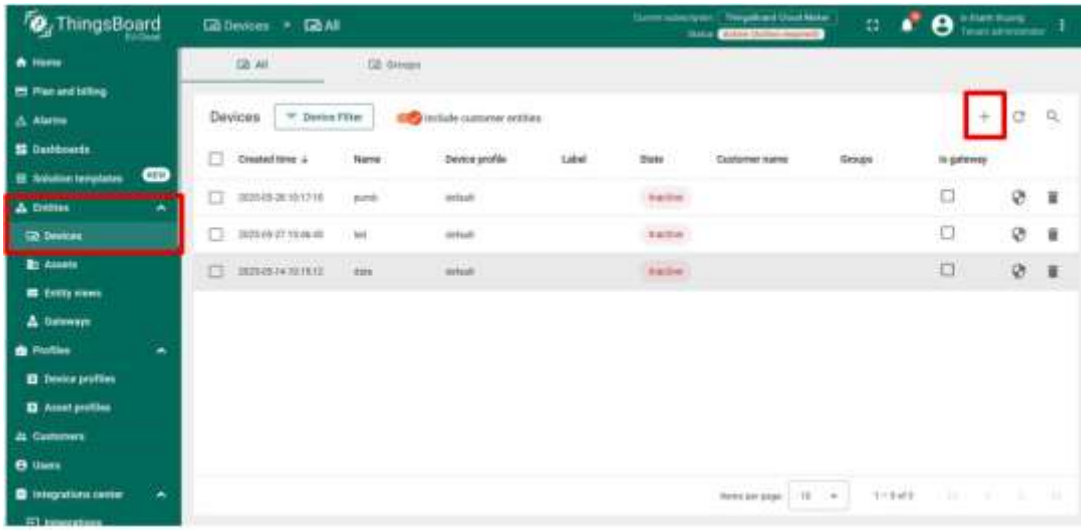
Các bước tạo một project ThingsBoard

Trước tiên truy cập vào địa chỉ https://thingsboard.io/ giao diện hiển thị như hình 4.10 phía dưới.

Hình 5. 18 Trang chủ ThingsBoard

Tiếp theo ta tạo thiết bị trên ThingsBoard:

- Chọn Entities → Devices và Add thiết bị

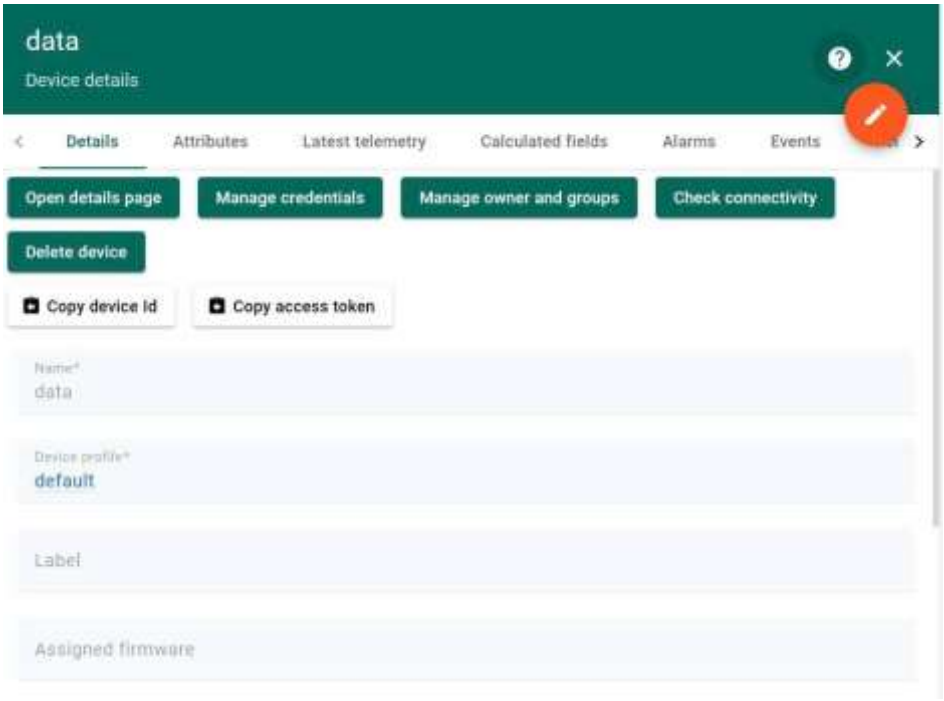


Hình 5. 19 Thiết lập Devices trên ThingsBoard

Điền các thông tin của thiết bị: Tên thiết bị, Mô tả và cấu hình Access Token

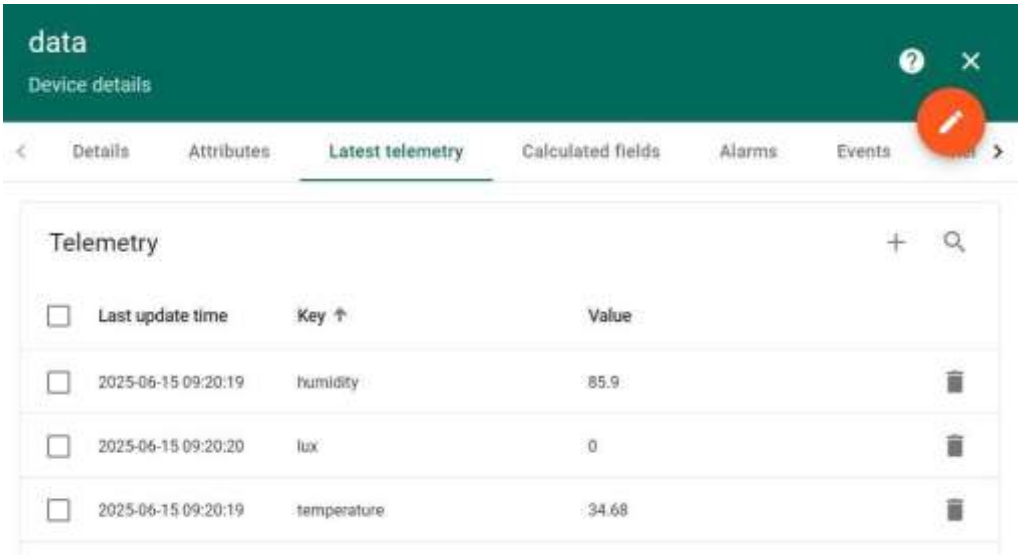
Để thiết bị IoT có thể truyền dữ liệu lên nền tảng ThingsBoard, cần thực hiện xác thực thiết bị. ThingsBoard sử dụng Access Token để nhận diện và xác thực từng thiết bị

- Nhấn nút **"Copy access token"** trong mục **Device details** để sao chép mã token



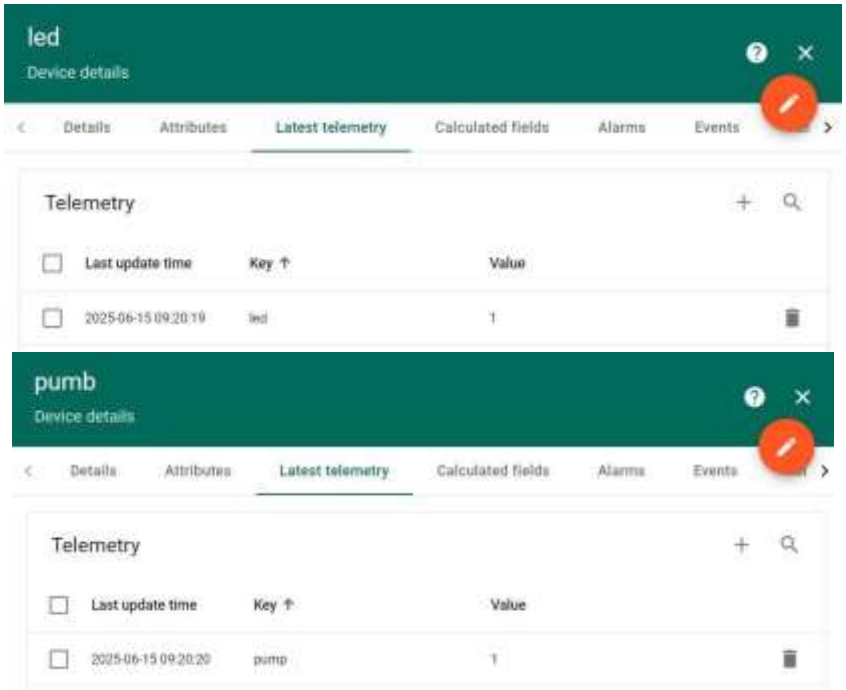
Hình 5. 20 Cấu hình Detail Device data

Giá trị gửi lên có thể được tìm kiếm tại Lastest Telemetry (Entities → Devices → data → Lastest Telemetry)



Hình 5. 21 Cấu hình Lastest Telemetry Device data

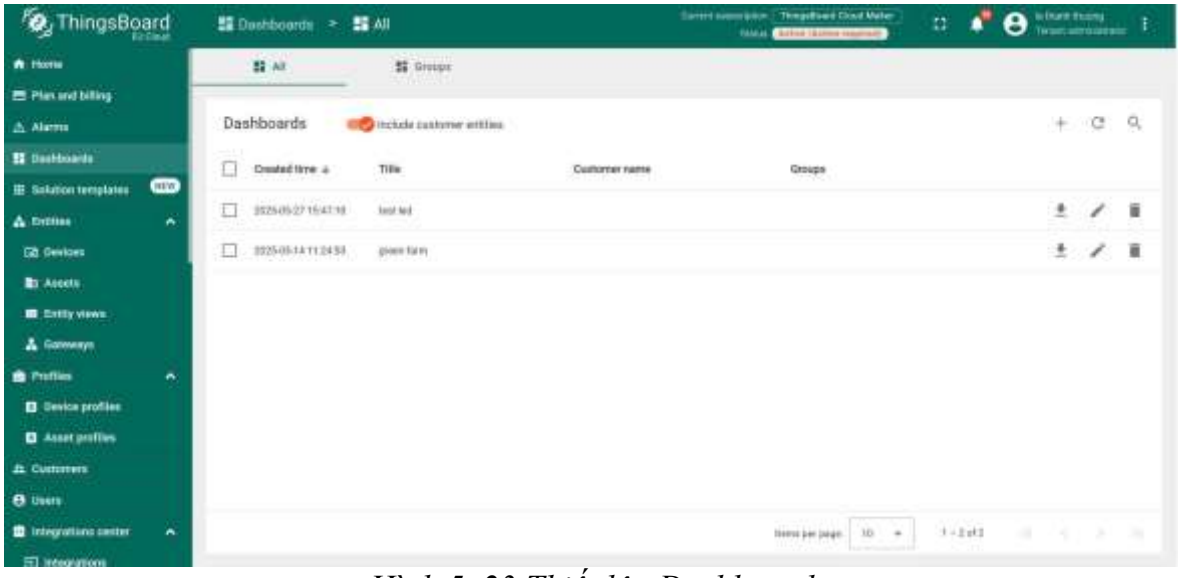
Tương tự cho Devices "led" và "pump" sẽ gửi giá trị lên tại Lastest Telemetry (Entities → Devices → led/pump → Lastest Telemetry) để bật tắt led và pump



Hình 5. 22 Cấu hình Lastest Telemetry Device led/pumb

5.8.2 Thiết lập Dashboard

- Chọn Dashboards "→ Add "green farm"



Hình 5. 23 Thiết lập Dashboard

Khi tạo widget trong Dashboard, ThingsBoard yêu cầu xác định **thiết bị dữ liệu đầu vào** thông qua cơ chế gọi là **Entity alias**

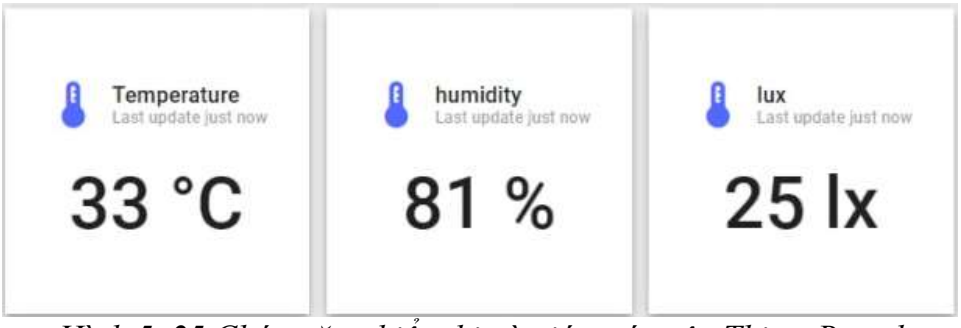
- Trong chế độ chỉnh sửa dashboard, nhấn nút **"Entity Aliases"**.

- Nhấn **Add alias**

Hình 5. 24 Thêm Alias trong dashboard

5.8.2.1 Hiển thị và giám sát

Chức năng hiện thị thông số môi trường cho phép người dùng có thể xem thông tin thông số môi tường trong hệ thống, bao gồm nhiệt độ, độ ẩm không khí.

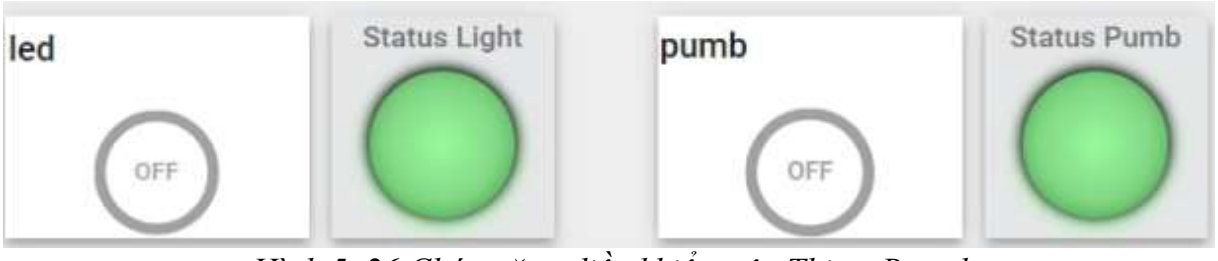


Hình 5. 25 Chức năng hiển thị và giám sát trên ThingsBoard

5.8.2.2 Chức năng điều khiển

Người dùng có thể kiểm soát và điều chỉnh các thiết bị theo ý muốn để đáp ứng yêu cầu cụ thể.

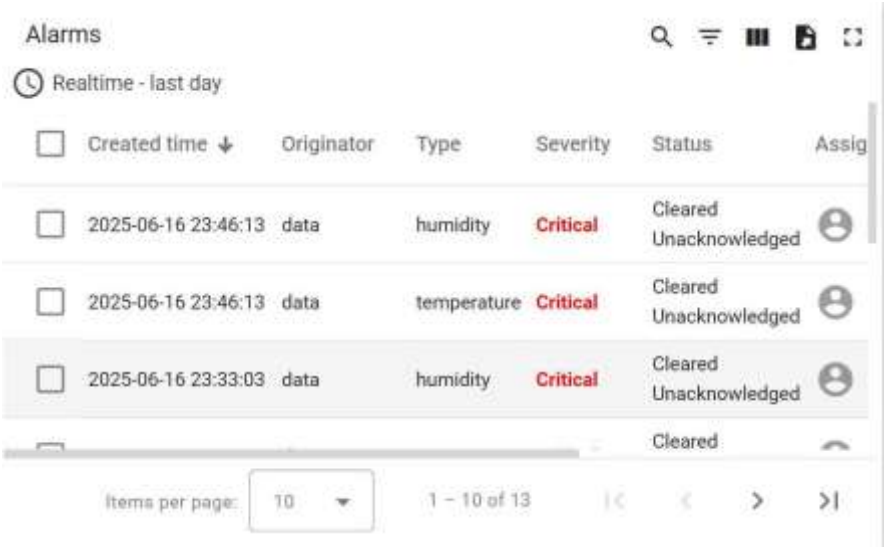
Thông qua giao diện trực quan người dùng có thể đánh giá nhanh chóng các thông số môi trường.



Hình 5. 26 Chức năng điều khiển trên ThingsBoard

5.8.2.3 Chức năng cảnh báo

Chức năng cảnh báo trên ThingsBoard giúp giám sát các giá trị môi trường như nhiệt độ, độ ẩm, ánh sáng theo thời gian thực.



Hình 5. 27 Chức năng cảnh báo trên ThingsBoard

5.9 Kết luận chương 5

Trong chương này, hệ thống phần mềm đã được thiết kế và triển khai toàn diện để đáp ứng các yêu cầu của mô hình Smart Garden. Cụ thể, quá trình tích hợp các cảm biến (SHT30, BH1750), cấu hình hiển thị LCD ILI9341, và phát triển giao diện với Qt trên nền BeagleBone Black đã được thực hiện chi tiết. Đồng thời, việc thiết lập hệ thống ThingsBoard để giám sát, điều khiển và cảnh báo đã giúp hệ thống có khả năng quản lý từ xa hiệu quả. Kết quả đạt được từ chương này là nền tảng quan trọng cho việc vận hành ổn định, mở rộng và ứng dụng thực tiễn của hệ thống trong các mô hình nông nghiệp thông minh.

CHƯƠNG 6: KẾT QUẢ VÀ ĐÁNH GIÁ HỆ THỐNG

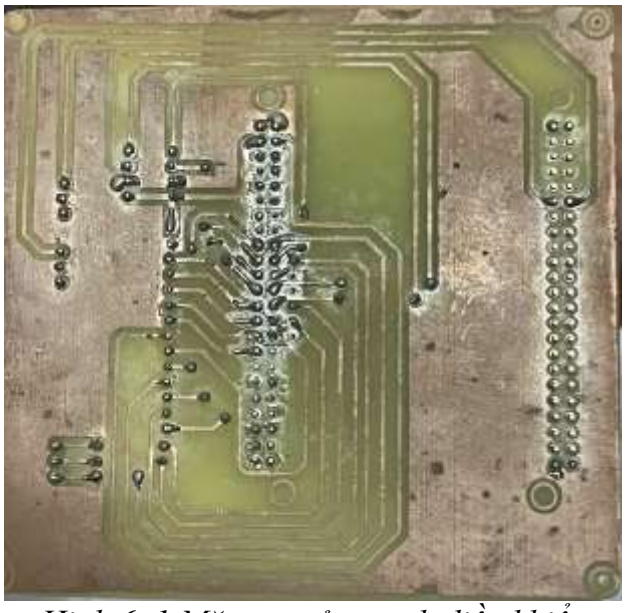
6.1 Giới thiệu chương

Chương này sẽ trình bày chi tiết về các kết quả đạt được sau khi thiết kế và triển khai hệ thống. Tiếp đến sẽ đi sâu vào các kết quả của phần cứng và phần mềm, bao gồm quá trình lắp ráp, kiểm tra, triển khai và thực nghiệm. Cuối cùng, chương sẽ đánh giá tổng thể hệ thống dựa trên hiệu suất, tính ổn định. Phần này sẽ cung cấp cái nhìn toàn diện về hiệu quả và tiềm năng của hệ thống.

6.2 Kết quả hệ thống

6.2.1 Kết quả phần cứng

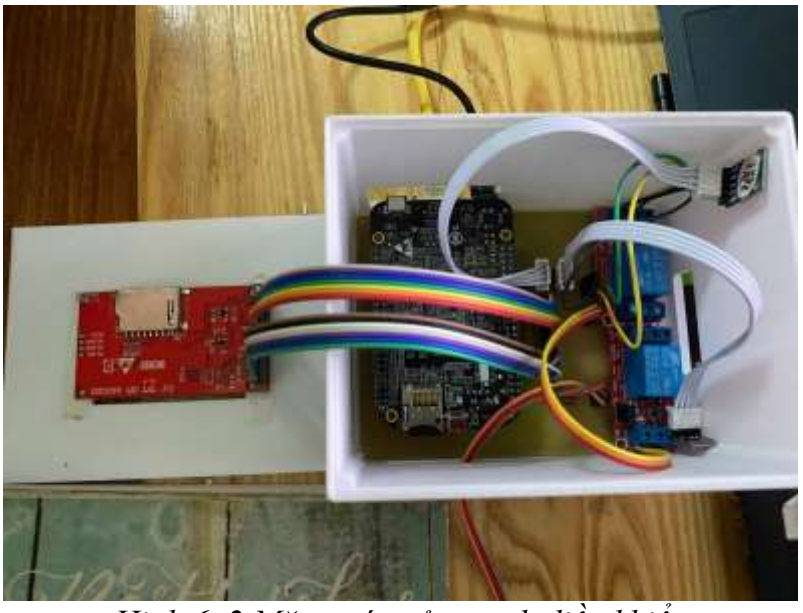
6.2.1.1 Mạch sau khi thi công



Hinh 6. 1 Mặt sau của mạch điều khiển

Sử dụng đồng hồ vạn năng để kiểm tra các đường mạch trên board mạch đảm bảo không có đứt gãy hoặc ngắn mạch. Sau khi hàn các linh kiện, tiếp tục kiểm tra kết nối giữa các điểm hàn. Gắn các cảm biến nhiệt độ, độ ẩm và ánh sáng vào vị trí thích hợp và nối dây cảm biến đến các chân tương ứng trên hộp điều khiển.

Quá trình lắp ráp và kiểm tra hình 6.2 cho thấy các linh kiện được kết nối chính xác, mạch chính được lắp ráp hoàn chình với các linh kiện, các cảm biến hoạt động tốt và dữ liệu được thu thập chính xác. Hệ thống phần cứng đã sẵn sàng cho việc triển khai thực tế.



Hình 6. 2 Mặt trước của mạch điều khiển

6.1.1.2 Đóng gói mạch thực tế

Sau khi thi công hoàn thành nhóm tiến hành đóng gói bộ điều khiển, mạch được đặt trong một hộp nhựa hình hộp chữ nhật chống nước kích thước 130cm x 100cm x 120cm

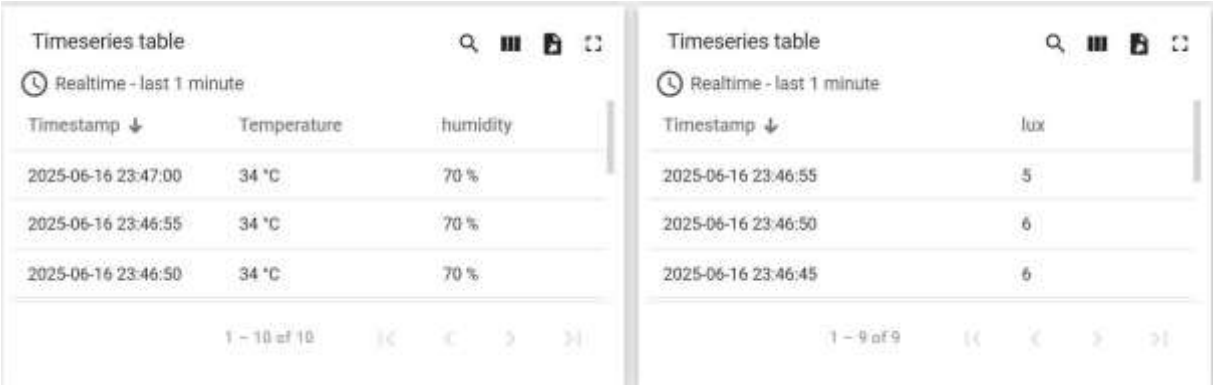
Board mạch chính được cố định, dễ thao tác tháo lắp. Mạch chính được đặt trong hộp bảo vệ với mục đích chống bụi và chống nước, đảm bảo hoạt động ổn định.

Hình 6. 3 Mạch sau khi đóng gói

6.2.2 Kết quả phần mềm

6.2.2.1 Kết quả ThingsBoard Realtime Database

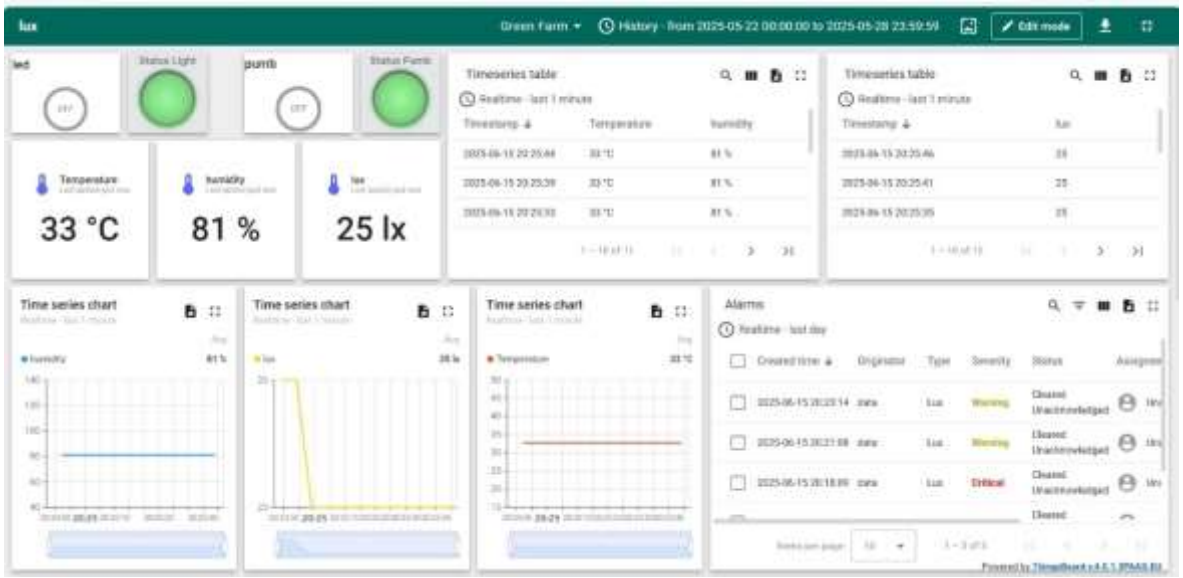
Hình ảnh 6.4 cho thấy được dữ liệu thu thập được từ hộp điều khiển chính và trạng thái điều khiển được gửi lên ThingsBoard Realtime Databse chính xác.



Hình 6. 4 Thu thập dữ liệu từ database trên ThingsBoard

6.2.2.2 Kết quả giao diện Dashboard

Giao diện dashboard hình 6.5 nơi có thể theo dõi các thông số môi trường và điều khiển các thiết bị từ xa



Hình 6. 5 Giao diện Dashboard của hệ thống

Giao diện website được thiết kế để người dùng có các chức năng sau:

+ Giám sát thông số môi trường như nhiệt độ, độ ẩm và ánh sáng theo thời gian

+ Điều khiển từ xa các thiết bị Bật/tắt quạt, đèn sưởi và quạt hơi nước.

+ Xem biểu đồ dữ liệu hiển thị biến động của các thông số môi trường theo thời gian. Kết quả thực nghiệm.

6.2.2 Mô hình hệ thống hoàn chỉnh

Hinh 6. 6 Mô hình hệ thống hoàn chỉnh

Kết quả của mô hình hệ thống thực tế hình 6.6 được thiết kế bằng bìa mô hình với các thiết bị điện như đèn, bơm được lắp đặt bên trong mô hình giúp việc thử nghiệm các thiết bị điện nhằm mô phỏng chức năng kiểm soát nhiệt độ và độ ẩm trong môi trường.

6.3 Đánh giá tổng thể

Sau quá trình nghiên cứu và thi công đồ án với đề tài "Thiết kế và xây dựng hệ thống nhúng Linux ứng dụng trong Smart Garden " nhóm đã đạt được kết quả như sau:

- Phần cứng của hệ thống đã được lắp ráp và đóng gói hoàn chỉnh. Quá trình lắp ráp đảm bảo các linh kiện được gắn chắc chắn và kết nối đúng theo sơ đồ mạch.
- Phần mềm của hệ thống đã phát triển và triển khai thành công, đáp ứng đầy đủ các yêu cầu đề ra. Dữ liệu từ các cảm biến và thiết bị điều khiển được thu thập và xử lý chính xác. Giao diện ThingsBoard và QT Creator dễ dàng sử dụng, cung cấp đầy đủ thông tin và có chức năng điều khiển từ xa giúp người dùng quản lý hệ thống hiệu quả.

Qua quá trình thực hiện đề tài, đề tài có khả năng ứng dụng thực tiễn cao, có thể giám sát trực tiếp trên ThingsBoard và màn hình LCD. Đồng thời điều khiển và giám

sát được các yếu tố môi trường đưa ra những chỉ số phục vụ cho người dùng. Có thể điều khiển, khống chế các yếu tố của môi trường tác động lên đời sống của cây trồng như nhiệt độ, độ ẩm, ánh sáng.

- Hoàn thành thiết kế phần cứng
- Hoàn thành thiết kế phần mềm
- Hoàn thành thu thập dữ liệu không dây
- Hoàn thành kết nối dữ liệu giữa thiết bị và ThingsBoard
- Hoàn thành thiết kế giao diện UI trên màn hình LCD
- Có thể giám sát các chỉ số thay đổi, điều khiển đèn, máy bơm và xem trực tiếp qua ThingsBoard và màn hình theo thời gian thực.

6.4 Kết luận chương 6

Chương này đã trình bày chi tiết các kết quả đạt được sau quá trình thiết kế và triển khai hệ thống Smart Garden. Các kết quả bao gồm cả phần cứng và phần mềm, từ khâu lắp ráp, kiểm thử, triển khai đến thực nghiệm thực tế. Hệ thống đã được đánh giá toàn diện về hiệu suất, độ ổn định và khả năng mở rộng. Kết quả cho thấy hệ thống góp phần nâng cao hiệu quả giám sát và điều khiển môi trường trồng trọt, giúp tối ưu hóa điều kiện phát triển của cây trồng và mang lại lợi ích thiết thực cho người làm nông nghiệp.

KẾT LUẬN

Sau khoảng thời gian thực hiện đồ án, dù không phải là thời gian dài nhưng chúng tôi đã tích lũy được một lượng kiến thức đáng kể. Trong quá trình thực hiện, chúng tôi nhận đã được sự hướng dẫn và hỗ trợ từ Thầy Giáp Quang Huy và anh Lưu An Phú, giúp chúng tôi hiểu rõ hướng đi và quy trình hoàn thiện đồ án. Chúng tôi đã nghiên cứu về các đặc điểm và tính chất của từng linh kiện bao gồm cảm biến và vi điều khiển, và kết hợp chúng thành một hệ thống hoàn chỉnh. Đồng thời, chúng tôi đã mở rộng kiến thức bằng việc học ngôn ngữ C/C++ để phát triển phần mềm cho hệ thống.

Trong đề tài lần này, nhóm đã thực hiện được những thành tựu quan trọng như:

- Nắm vững kiến thức lập trình cho vi điều khiển và cách truyền nhận dữ liệu với ThingsBoard
- Thiết kế thành công giao diện Dashboard sử dụng ngôn ngữ C và ngôn ngữ lập trình C++ để thiết kế giao diện UI trên QT Creator.
- Xây dựng thành công cơ sở dữ liệu Realtime.
- Phát triển kỹ năng vẽ mạch và thi công mạch.

Trong quá trình thực hiện Đồ Án Tốt Nghiệp, chúng tôi đã học cách làm việc độc lập và phân chia công việc cho dự án lớn thành những phần nhỏ hơn, tuân thủ tiến độ đã đề ra. Chúng tôi đã có các cuộc thảo luận với giáo viên hướng dẫn và xây dựng kế hoạch chi tiết cho từng nhiệm vụ.

Hướng phát triển

Sau khi hoàn thành đề tài và đạt được những kết quả nhất định, nhóm chúng tôi đặt ra hướng phát triển cho hệ thống để có được kết quả tốt hơn, tăng hiệu suất và có thể áp dụng vào thực tế:

- Tiếp tục nghiên cứu và phát triển phần cứng để hệ thống đạt được hiệu suất cao nhất, tiết kiệm chi phí sản xuất.
- Mở rộng thêm số lượng thiết bị điều khiển, khả năng tích hợp nhiều hệ thống khác.
- Phát triển thêm về ứng dụng trên app điện thoại, thêm chức năng cảnh báo đến người trồng trọt.

TÀI LIỆU THAM KHẢO

[1]. V. Sreekrishnan, *Essential Linux Device Drivers*, Prentice Hall, 2008.

[2]. M. Kerrisk, *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*, No Starch Press, 2010

[3]. Corbet, J., Rubini, A., & Kroah-Hartman, G. (2005). *Linux device drivers* (3rd ed.). O'Reilly Media.

[4]. R. J. Streif, *Embedded Linux Systems with the Yocto Project*, Prentice Hall, 2016

[5]. The Yocto Project. "Yocto Project Documentation." [Online]. Available: https://docs.yoctoproject.org/

[6]. R. J. Streif, *Embedded Linux Systems with the Yocto Project*, 2nd ed. [Online]. Available: https://www.oreilly.com/library/view/embedded-linux-systems/9780133443240/

[7]. Qt Company. "Qt Creator Manual." [Online]. Available: https://doc.qt.io/qtcreator/index.html

[8]. ThingsBoard. "ThingsBoard Documentation (Community Edition)." [Online]. Available: https://thingsboard.io/docs/

[9]. ThingsBoard. "ThingsBoard on GitHub." [Online]. Available: https://github.com/thingsboard/thingsboard

[10]. BeagleBoard.org. "BeagleBone Black System Reference Manual." [Online]. Available: https://docs.beagleboard.org/latest/boards/beaglebone-black/index.html

[11]. BeagleBoard.org. "BeagleBoard Official Website." [Online]. Available: https://beagleboard.org/

[12]. M. Kerrisk, *The Linux Programming Interface*. [Online]. Available: https://man7.org/tlpi

[13]. J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*, 3rd ed. [Online]. Available: https://lwn.net/Kernel/LDD3/

PHỤ LỤC

❖ Chương trình cảm biến

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <curl/curl.h>
#include <pthread.h>

#define BH1750_ADDR     0x23
#define I2C_DEVICE      "/dev/i2c-2"
#define LUX_AVG_SAMPLES 20
#define THINGSBOARD_URL "https://eu.thingsboard.cloud"
#define THINGSBOARD_TOKEN "6ajheVfXVI5L9OxRB8P3"
#define THINGSBOARD_TOKEN1 "0PIVNYnXhPVwN3knlLti"

// Global variables (file scope)
static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
static volatile int lux = 0;
static int anh_sang = 0;
static int led_status;

// Thread to read illuminance data from BH1750 sensor
void *read_lux(void *arg) {
    int fd = open(I2C_DEVICE, O_RDWR);
    if(fd < 0) {
        perror("Failed to open I2C device");
        return NULL;
    }

    if (ioctl(fd, I2C_SLAVE, BH1750_ADDR) < 0) {
        perror("Failed to set I2C slave address");
        close(fd);
        return NULL;
    }

    while(1) {
        int sum = 0;
        for(int i = 0; i < LUX_AVG_SAMPLES; ++i) {
            char cmd = 0x20; // Continuous H-resolution mode
```

```
            if(write(fd, &cmd, 1) != 1) {
                perror("I2C write failed");
                continue;
            }

            usleep(180000); // Wait 180ms for measurement

            char buf[2];
            if(read(fd, buf, 2) != 2) {
                perror("I2C read failed");
                continue;
            }

            pthread_mutex_lock(&lock);
            lux = (buf[0] << 8) | buf[1];
            sum += lux;
            pthread_mutex_unlock(&lock);
            usleep(200000);
        }

        pthread_mutex_lock(&lock);
        anh_sang = sum / LUX_AVG_SAMPLES;
        pthread_mutex_unlock(&lock);

        printf("Average illuminance: %d lux\n", anh_sang);
        sleep(10);
    }

    close(fd);
    return NULL;
}

// Callback to receive CURL response into a buffer string
static size_t write_callback(void *contents, size_t size, size_t nmemb,
void *userp) {
    size_t total_size = size * nmemb;
    strncat((char *)userp, contents, total_size); // Append response data
to buffer
    return total_size;
}

// Thread to listen for RPC requests from ThingsBoard
void *rpc_listener(void *arg) {
    CURL *curl = curl_easy_init();
    if (!curl) return NULL;

    char url[256];
```

```
    snprintf(url, sizeof(url), "%s/api/v1/%s/rpc", THINGSBOARD_URL,
THINGSBOARD_TOKEN1);

    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "Content-Type: application/json");

    while (1) {
        char response[512] = {0}; // Buffer for server response

        curl_easy_setopt(curl, CURLOPT_URL, url);
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
        curl_easy_setopt(curl, CURLOPT_TIMEOUT, 60L);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, response);
        curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, "GET");
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L);

        CURLcode res = curl_easy_perform(curl);
        if (res == CURLE_OK) {
            if (strstr(response, "\"method\":\"setState\"")) {
                pthread_mutex_lock(&lock);
                if (strstr(response, "true")) {
                    led_status = 1;
                    printf("LED ON (RPC)\n");
                } else if (strstr(response, "false")) {
                    led_status = 0;
                    printf("LED OFF (RPC)\n");
                }
                pthread_mutex_unlock(&lock);
            }
        } else {
            fprintf(stderr, "CURL RPC error: %s\n",
curl_easy_strerror(res));
        }

        sleep(10);
    }

    curl_slist_free_all(headers);
    curl_easy_cleanup(curl);
    return NULL;
}

// Dummy callback for unused CURL response
static size_t dummy_callback(void *buffer, size_t size, size_t nmemb, void
*userp) {
```

```
    return size * nmemb;
}

// Thread to send illuminance data to ThingsBoard
void *gui_data_thingsboard(void *arg) {
    CURL *curl = curl_easy_init();
    if(!curl) {
        fprintf(stderr, "Failed to initialize CURL\n");
        return NULL;
    }

    char url[256];
    snprintf(url, sizeof(url), "%s/api/v1/%s/telemetry",
            THINGSBOARD_URL, THINGSBOARD_TOKEN);

    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "Content-Type: application/json");

    while(1) {
        char payload[64];
        int current_lux;

        pthread_mutex_lock(&lock);
        current_lux = anh_sang;
        pthread_mutex_unlock(&lock);

        snprintf(payload, sizeof(payload), "{\"lux\":%d}", current_lux);

        curl_easy_setopt(curl, CURLOPT_URL, url);
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, payload);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, dummy_callback);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L);

        CURLcode res = curl_easy_perform(curl);
        if(res != CURLE_OK) {
            fprintf(stderr, "CURL error: %s\n", curl_easy_strerror(res));
        } else {
            printf("Sent to ThingsBoard: %s\n", payload);
        }

        sleep(5);
    }

    curl_slist_free_all(headers);
    curl_easy_cleanup(curl);
```

```
    return NULL;
}

// Thread to provide local sensor data via TCP socket
void *gui_data(void *arg) {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0) {
        perror("Socket creation failed");
        return NULL;
    }

    int opt = 1;
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    struct sockaddr_in addr = {
        .sin_family = AF_INET,
        .sin_port = htons(5000),
        .sin_addr.s_addr = INADDR_ANY
    };

    if(bind(sockfd, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("Bind failed");
        close(sockfd);
        return NULL;
    }

    listen(sockfd, 5);

    while(1) {
        int client_fd = accept(sockfd, NULL, NULL);
        if(client_fd < 0) {
            perror("Accept failed");
            continue;
        }

        char buffer[128];
        pthread_mutex_lock(&lock);
        snprintf(buffer, sizeof(buffer), "%d,%d\n", anh_sang, led_status);
        pthread_mutex_unlock(&lock);

        write(client_fd, buffer, strlen(buffer));
        close(client_fd);

        sleep(10);
    }

    close(sockfd);
```

```
    return NULL;
}

int main() {
    pthread_t threads[4];

    // Create all threads
    if(pthread_create(&threads[0], NULL, read_lux, NULL)) {
        perror("Failed to create sensor thread");
        return 1;
    }
    if(pthread_create(&threads[1], NULL, rpc_listener, NULL)){
        perror("Failed to create ThingsBoard RPC thread");
        return 1;
    }

    if(pthread_create(&threads[2], NULL, gui_data_thingsboard, NULL)) {
        perror("Failed to create ThingsBoard telemetry thread");
        return 1;
    }

    if(pthread_create(&threads[3], NULL, gui_data, NULL)) {
        perror("Failed to create socket thread");
        return 1;
    }

    // Join all threads (program will run indefinitely)
    for(int i = 0; i < 4; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&lock);
    return 0;
}
```

❖ Chương trình cho thiết bị

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
```

```c
#include <gpiod.h>
#include <pthread.h>
#include <curl/curl.h>
#include <time.h>

#define CONSUMER "Consumer"
#define LUX_THRESHOLD 100
#define THINGSBOARD_URL "https://eu.thingsboard.cloud"
#define THINGSBOARD_TOKEN1 "0PIVNYnXhPVwN3knlLti"

char *chipname = "gpiochip0";
unsigned int line_num = 13; // GPIO0_13 = P8_11
int ret,ret1;
int lux,led_status;
int current_lux = 0;
int current_led = 0;
volatile static int gpio_state;
struct gpiod_chip *chip;
struct gpiod_line *line;
static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
int is_night;

void *is_night_time_thread(void *arg) {
    printf("Thread is_night_time_thread started\n");
    fflush(stdout);

    while (1) {
        time_t rawtime;
        struct tm *timeinfo;

        if (time(&rawtime) == -1) {
            perror(" time() failed");
            sleep(60);
            continue;
        }

        timeinfo = localtime(&rawtime);
        if (!timeinfo) {
            perror("localtime() failed");
            sleep(60);
            continue;
        }

        int hour = timeinfo->tm_hour;
        int minute = timeinfo->tm_min;

        printf("Current time: %02d:%02d\n", hour, minute);
```

```
        fflush(stdout);

        // Determine if it's night time (6 PM to 6 AM)
        is_night = (hour >= 18 || hour < 6) ? 1 : 0;

        sleep(60); // Check every minute
    }

    return NULL;
}

int *export_gpio() {
    chip = gpiod_chip_open_by_name(chipname);
    if (!chip) {
        perror("Open chip failed");
        return -1;
    }

    line = gpiod_chip_get_line(chip, line_num);
    if (!line) {
        perror("Get line failed");
        gpiod_chip_close(chip);
        return -1;
    }
    int ret = gpiod_line_request_output(line, CONSUMER, 0);
    if (ret < 0) {
        perror("Request line as output failed");
        gpiod_chip_close(chip);
        return -1;
    }
    return NULL;
}

void *recive_data(void *arg){

    while(1){
        int sockfd = -1;
        struct sockaddr_in server_addr;
        char recv_buffer[1024];

        memset(recv_buffer, 0, sizeof(recv_buffer));
        memset(&server_addr, 0, sizeof(server_addr));

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(5000);
```

```
        if (connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) == 0) {
            read(sockfd, recv_buffer, sizeof(recv_buffer) - 1);
            printf("Received lux: %s\n", recv_buffer);
            if (sscanf(recv_buffer, "%d,%d", &lux, &led_status) == 2) {
                printf("Lux: %d, LED Status: %d\n", lux, led_status);
            }
            close(sockfd);
        }
        else{
            perror("Connection failed");
            close(sockfd);
            sleep(1);
        }
    }
}

static size_t dummy_callback(void *buffer, size_t size, size_t nmemb, void
*userp) {
    return size * nmemb; // Handle response if needed
}

void *gui_data(void *arg) {
    CURL *curl = curl_easy_init();
    if(!curl) {
        fprintf(stderr, "Failed to initialize CURL\n");
        return NULL;
    }

    char url[256];
    snprintf(url, sizeof(url), "%s/api/v1/%s/telemetry",
            THINGSBOARD_URL, THINGSBOARD_TOKEN1);

    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "Content-Type: application/json");

    while(1) {
        char payload[64];
        int led_state;

        pthread_mutex_lock(&lock);
        led_state = gpio_state;
        pthread_mutex_unlock(&lock);

        snprintf(payload, sizeof(payload), "{\"led\":%d}", gpio_state);

        curl_easy_setopt(curl, CURLOPT_URL, url);
```

```
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, payload);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, dummy_callback);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L);

        CURLcode res = curl_easy_perform(curl);
        if(res != CURLE_OK) {
            fprintf(stderr, "CURL error: %s\n", curl_easy_strerror(res));
        } else {
            printf("Sent to ThingsBoard: %s\n", payload);
        }

        sleep(5);
    }

    curl_slist_free_all(headers);
    curl_easy_cleanup(curl);
    return NULL;
}

void *control_led(void *arg) {
    while (1) {
        pthread_mutex_lock(&lock);
        current_lux = lux;
        current_led = led_status;
        pthread_mutex_unlock(&lock);

        // Determine target LED status based on received data and lux
threshold
        int target_status = (current_led == 1 || current_lux <
LUX_THRESHOLD) ? 1 : 0;

        ret = gpiod_line_set_value(line, target_status);
        if (ret < 0) {
            perror("Set line output failed");
        } else {
            printf("Light %s\n", target_status ? "ON" : "OFF");
        }

        sleep(1); // Reduce wait time for faster reaction
    }

    // GPIO release moved outside loop (will never reach here)
    gpiod_line_release(line);
    gpiod_chip_close(chip);
    return NULL;
```

```
}

void *read_data_gpio(void* arg) {
    int last_state = -1;

    while (1) {
        gpio_state = gpiod_line_get_value(line);
        if (gpio_state < 0) {
            perror("GPIO read error");
            sleep(1);
            continue;
        }

        if (gpio_state != last_state) {
            printf("LED state changed: %d\n", gpio_state);
            last_state = gpio_state;
        }

        usleep(200000); // 200ms is reasonable for quick response without
spamming logs
    }
}

int main() {

    if (export_gpio() != 0){
        perror("GPIO export failed");
        return 0;
    }

    pthread_t threads[5];

    // Initialize threads
    if(pthread_create(&threads[0], NULL, recive_data, NULL)) {
        perror("Failed to create sensor thread");
        return 1;
    }
    if(pthread_create(&threads[1], NULL, control_led, NULL)){
        perror("Failed to create LED control thread");
        return 1;
    }
    if(pthread_create(&threads[2], NULL, read_data_gpio, NULL)){
        perror("Failed to create GPIO read thread");
        return 1;
    }
    if(pthread_create(&threads[3], NULL, gui_data, NULL)){
        perror("Failed to create ThingsBoard send thread");
```

```
        return 1;
    }
    if(pthread_create(&threads[4], NULL, is_night_time_thread, NULL)){
        perror("Failed to create night time check thread");
        return 1;
    }


    // Wait for all threads (in practice, never ends)
    for(int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&lock);
    gpiod_line_release(line);
    gpiod_chip_close(chip);
    return 0;
}
```

Đề tài tập trung nghiên cứu, thiết kế và xây dựng hệ thống nhúng Linux tùy chỉnh dựa trên nền tảng BeagleBone Black để triển khai trong mô hình vườn thông minh (Smart Garden). Hệ thống sử dụng Yocto Project để cấu hình và biên dịch hệ điều hành phù hợp với phần cứng, tích hợp các cảm biến môi trường và giao tiếp với nền tảng ThingsBoard nhằm giám sát, điều khiển thiết bị từ xa hoặc trực tiếp. Mục tiêu là tạo ra một hệ thống giám sát – điều khiển môi trường trồng trọt linh hoạt, hiệu quả, tiết kiệm chi phí, đồng thời giúp sinh viên nắm vững quy trình phát triển một hệ điều hành nhúng hoàn chỉnh.

Nội dung nghiên cứu bao gồm:
- Xây dựng hệ điều hành Linux nhúng tùy biến cho BeagleBone Black.
- Sử dụng Yocto Project để tạo image hệ điều hành tối ưu.
- Tích hợp cảm biến nhiệt độ, độ ẩm, ánh sáng.
- Hiển thị và điều khiển qua ThingsBoard (Web) và LCD cảm ứng.
- Hỗ trợ hai chế độ điều khiển: tự động (dựa vào cảm biến) và thủ công (qua giao diện).
- Lưu trữ dữ liệu theo thời gian thực phục vụ phân tích.
- Ứng dụng trong mô hình Smart Garden – giám sát và chăm sóc cây trồng thông minh.
- Góp phần nghiên cứu, triển khai hệ thống nhúng Linux trong thực tế.

The project focuses on researching, designing, and developing a customized embedded Linux system based on the BeagleBone Black platform, to be implemented in a Smart Garden model. The system utilizes the Yocto Project to configure and compile the operating system tailored to the hardware, integrates environmental sensors, and communicates with the ThingsBoard platform to monitor and control devices either remotely or locally. The goal is to create a flexible, efficient, and cost-effective environmental monitoring and control system for crop cultivation, while enabling students to gain hands-on experience in developing a complete embedded Linux operating system.

Research contents include:
- Building a customized embedded Linux operating system for BeagleBone Black.
- Using the Yocto Project to generate an optimized Linux image.
- Integrating temperature, humidity, and light sensors.
- Displaying and controlling through ThingsBoard (Web) and touchscreen LCD.
- Supporting two control modes: automatic (sensor-based) and manual (via interface).
- Storing real-time data for analysis and monitoring.
- Applying the system to a Smart Garden model for intelligent crop management.
- Contributing to research and practical deployment of embedded Linux systems