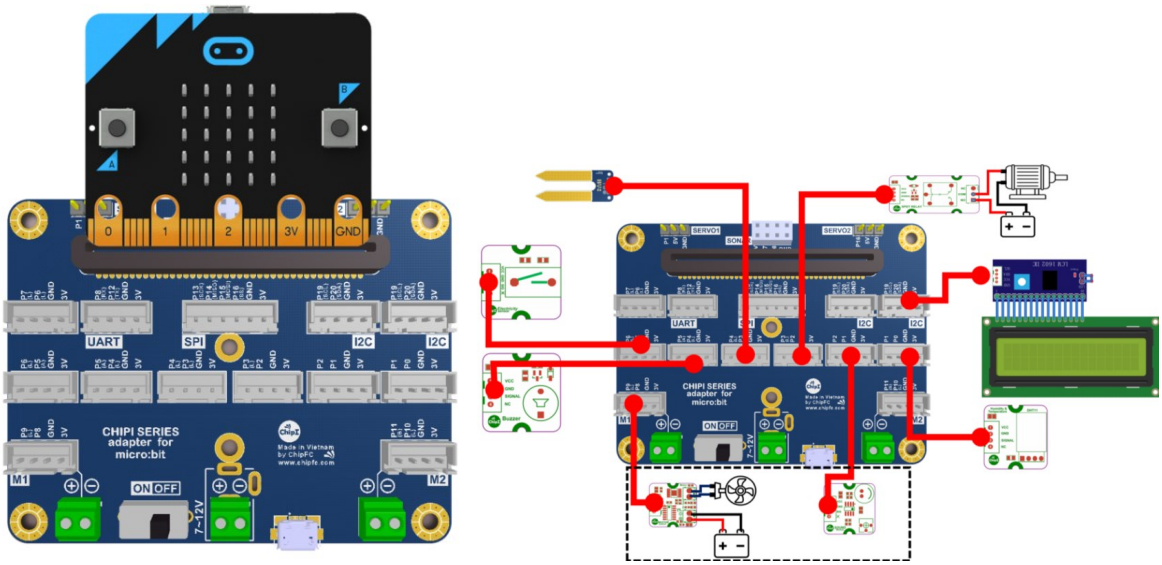
THE DARIU FOUNDATION

Investment In Empowerment



Hướng dẫn lập trình Micro:bit nâng cao cho học sinh

Lưu hành nội bộ

THE DARIU FOUNDATION
Investment In Empowerment

MỤC LỤC

GIỚI THIỆU

1. Tổng quan

Khoá học nâng cao ngôn ngữ lập trình MicroBit được tổ chức, tập huấn và triển khai bởi Tổ chức The Dariu Foundation, phối hợp cùng với các đối tác tài trợ.

Toàn bộ tài liệu học tập, tư liệu âm thanh hình ảnh, giáo trình giảng dạy được biên soạn bởi Tiến sĩ Lê Trọng Nhân – giảng viên trường Đại học Bách Khoa thành phố Hồ Chí Minh.

Với mong muốn đem lại những giờ học thú vị, kích thích tư duy sáng tạo cho các em học sinh, Quỹ Dariu kính mong quý thầy cô triển khai đầy đủ **8 tiết học** Lập trình MicroBit nâng cao đến với các em học sinh **dựa theo 8 bài giảng** như trên phần Mục Lục.

✓ Phân bổ tiết học và thời gian:

- **Tiết 1** (45 phút) – Bài 1: Chương trình đầu tiên với ChiPi
- **Tiết 2** (45 phút) – Bài 2: Điều khiển loa Buzzer
- **Tiết 3** (45 phút) – Bài 3: Nhận dữ liệu từ nút nhấn
- **Tiết 4** (45 phút) – Bài 4: Điều khiển nút nhấn trên MicroBit
- *(Bài tham khảo)* – Bài 5: Giới thiệu thư viện mở rộng cho MicroBit
- **Tiết 5** (45 phút) – Bài 6: Phát triển dự án Đèn giao thông
- **Tiết 6** (45 phút) – Bài 7: Tái cấu trúc chương trình đèn giao thông
- **Tiết 7** (45 phút) – Bài 8: Hiện thực trụ đèn phụ
- **Tiết 8** (45 phút) – Bài 9: Hiện thực trụ đèn phụ thứ 2 + **Kiểm tra Online trắc nghiệm**
- *(Bài tham khảo)* – Bài 10: Tích hợp thêm dịch vụ cho dự án Đèn Giao Thông

Hai bài tham khảo (bài 5 và bài 10) là tiết mở rộng, dành cho học sinh giỏi có nhu cầu tham khảo thêm các kiến thức nằm ngoài chương trình dạy chính.

Lưu ý: Chương trình dùng để lập trình MicroBit là Website Onine nên cần được đảm bảo mạng Internet cho các thiết bị học tập của học sinh. Hiện tại đã có bản Microbit Offline, thầy cô nên cài đặt sẵn trên máy để tiện cho việc học tập của học sinh tham gia khóa học.

2. Tài liệu học tập

Sách, Giáo trình và các tài liệu tham khảo chính:

[1] Trang tải phiên bản Microbit Offline: http://bit.ly/makecodeoffline

Hoặc thầy cô vào Microsoft Store → Search Make Code Microbit và tải về bản Offline

[2] Trang lập trình Online, cộng đồng chia sẻ ứng dụng, cũng như nơi tổng hợp các tài liệu tham khảo: https://makecode.microbit.org/

3. Mục tiêu

- ❖ Học sinh hiểu, nắm bắt được các nguyên lý tương tác giữa việc thiết lập bo mạch và các linh kiện hỗ trợ thông dụng bằng lập trình MicroBit, qua đó phát triển tư duy sáng tạo.
- ❖ Học sinh ứng dụng được cách tương tác giữa phần mềm lập trình đối với phần cứng bo mạch, bước đầu tiến đến kỷ nguyên Công nghệ 4.0.

Mọi thắc mắc, vui lòng liên hệ: - prt.ho01@dariu.org – *Trần Thanh Hải* (Giám đốc dự án)

- haphan@dariu.org – *Trần Hà Phan* (trợ lý dự án)

- **MỤC TIÊU**

Học sinh sẽ:

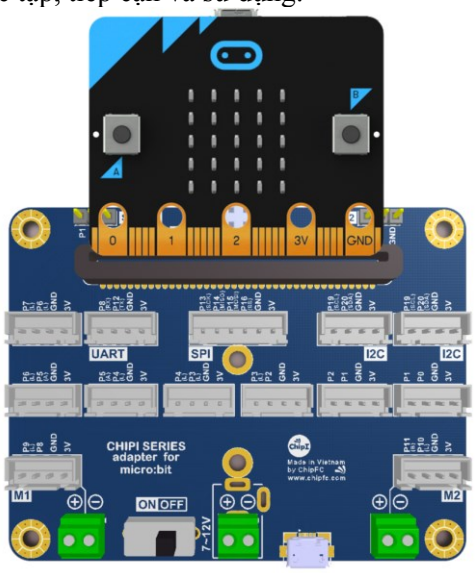
- Làm quen với hệ thống cảm biến ChiPi
- Kết nối được MicroBit và mạch ChiPi LED
- Hiện thực được chương trình đầu tiên với mạch ChiPi LED.
- Có thể lưu và mở lại chương trình cũ.

- **PHÂN BỔ THỜI LƯỢNG**

Bài 1: Chương trình đầu tiên với ChiPi

1 Giới thiệu về hệ thống ChiPi

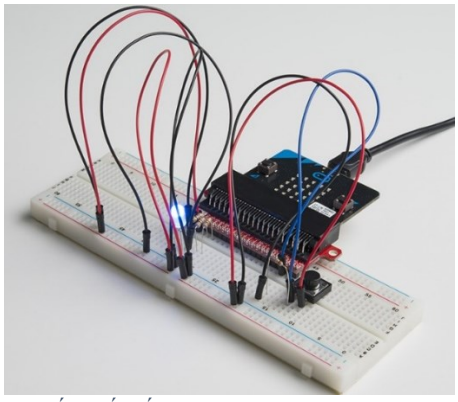
Chipi Series là một hệ thống các mô-đun điện tử, sẵn sàng và dễ dàng cho việc sử dụng. Tương tự như việc lắp ghép mô hình, mỗi mô-đun ChiPi sẽ là một mảnh ghép trong hệ thống điện tử của bạn. So với cách truyền thống, sử dụng breadboard và các linh kiện điện tử cơ bản để lắp ráp tạo nên một dự án điện tử, hệ thống ChiPi giúp đơn giản hóa và tiết kiệm thời gian đáng kể cho quá trình học tập, tiếp cận và sử dụng.



Hình 1: MicroBit ChiPi Base Shield: Kết nối các mô-đun khác với mạch MicroBit

Hệ thống Chipi bao gồm một MicroBit ChiPi Base Shield (Hình 1) và nhiều mô-đun cảm biến. Base Shield đóng vài trò là mạch trung gian, giúp kết nối các mô-đun Chipi với MicroBit. Trong khi mỗi mô-đun Chipi sẽ giải quyết một chức năng duy nhất, chẳng hạn như một nút nhấn đơn giản hoặc một bộ cảm biến nhịp tim phức tạp hơn. Các mô-đun được thiết kế hướng tới một chuẩn kết nối đồng nhất, giúp cho người dùng cảm thấy dễ dàng trong việc kết nối nhiều mô-đun khác nhau trong một ứng dụng, thay vì phải kết nối nhiều dây với các tiêu chuẩn khác nhau như Hình 2.

Lưu ý quan trọng: Màn hình của mạch MicroBit quay về phía các chân kết nối.



Hình 2: Hệ thống kết nối MicroBit thông thường với breadboard

Cuối cùng, một trong những nguyên nhân chính để hệ thống ChiPi cho MicroBit được xây dựng là do việc kết nối với các chân mở rộng trên mạch MicroBit rất khó để kết nối. Thông thường, người dùng chỉ có thể kết nối được với 3 chân là P0, P1 và P2 mà thôi. Mặc dù trên cộng đồng có khá nhiều dự án ấn tượng, tuy nhiên việc mở rộng thêm chức năng là rất khó khăn.

Bên cạnh ChiPi Base Shield, là rất nhiều các mô-đun cảm biến. Các mô-đun Chipi cảm biến sẽ trang bị cho hệ thống điện tử của bạn các "giác quan". Chúng giúp bạn đo đạc, phát hiện các yếu tố mọi trường hằng ngày như nhiệt độ, ánh sáng, độ ẩm... giúp bạn biểu đồ hóa, phân tích, đánh giá cũng như điều khiển các thiết bị liên quan một cách mềm dẻo phù hợp với từng điều kiện.

Trong giáo trình này, chúng tôi sẽ hướng dẫn sử dụng 4 mô-đun sau đây

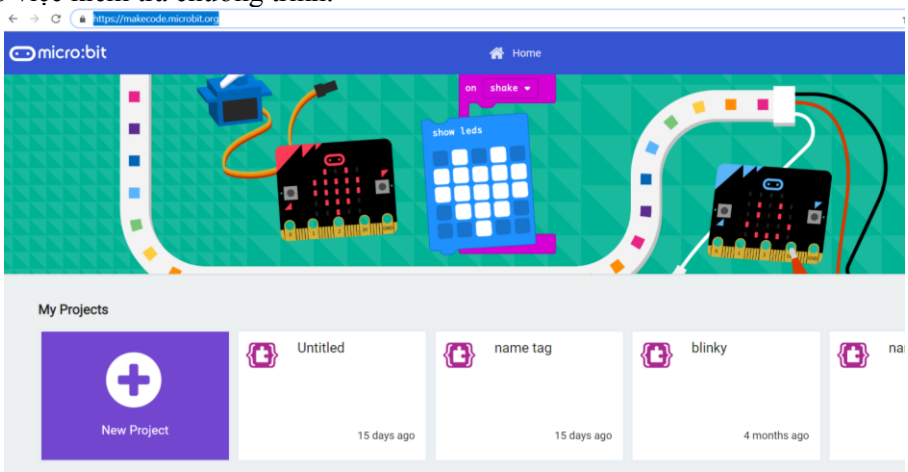
- LED: Đây là mô-đun cơ bản nhất trong hệ thống. Mô-đun này có thể điều khiển để hiện thị được 3 màu: Đỏ Xanh và Vàng.
- Buzzer: Loa báo động, có thể được dùng để phát ra âm thanh cảnh báo
- Button: Nút nhấn, đây cũng là một trong những mô-đun căn bản nhận tương tác từ người dùng.
- Light Sensor: Cảm biến ánh sáng. Khác với cảm biến nội bên trong MicroBit, cảm biến này hoạt động dựa trên nguyên lý của quang trở, nên sẽ chính xác hơn.

Bằng cách kết hợp các mô-đun này, chúng ta sẽ lập trình để minh họa một hệ thống đèn giao thông thông minh. Trong hướng dẫn này sẽ được chia làm 2 phần. Phần đầu sẽ là các hướng dẫn cơ bản để sử dụng các mô-đun này. Ở phần 2, các mô-đun này sẽ được kết hợp để làm nên một ứng dụng giao thông thông minh.

Trong bài hướng dẫn đầu tiên này, chúng ta sẽ làm việc với module đơn giản nhất, có tên là LED ChiPi.

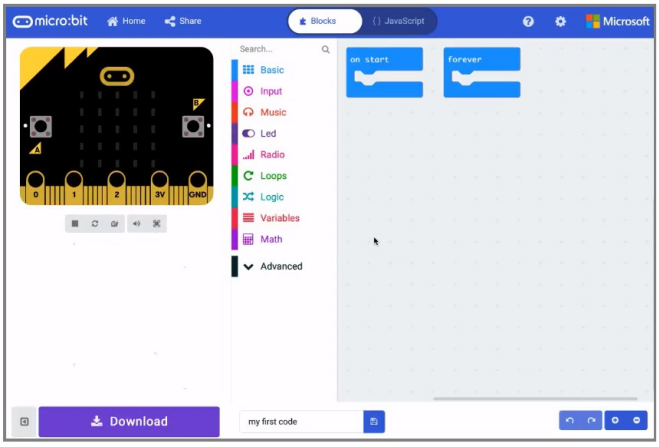
2 Mô trường lập trình cho MicroBit

Đầu tiên, phải kể đến chương trình lập trình trực tuyến MakeCode trên máy tính, bằng cách truy cập vào đường dẫn https://makecode.microbit.org/. Một lợi thế rất lớn mà MakeCode có được là việc mô phỏng chương trình trước khi nạp trực tiếp vào mạch MicroBit. Chức năng này sẽ tiết kiệm nhiều thời gian cho việc kiểm tra chương trình.



Hình 3: Môi trường lập trình trực tuyến trên máy tính

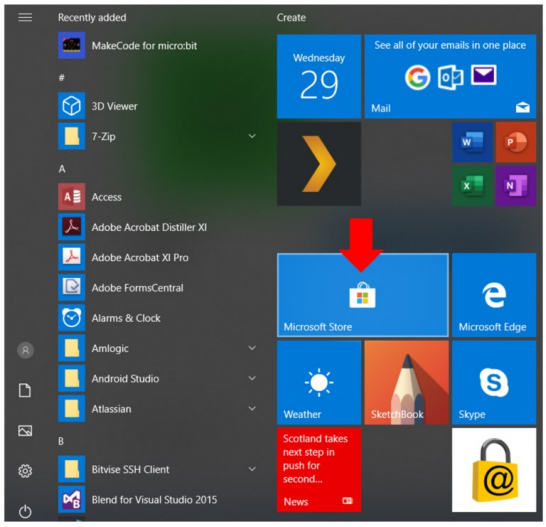
Chọn vào mục New Project ở Hình 3, giao diện sau đây sẽ hiện ra để chúng ta có thể bắt đầu lập trình.



Hình 4: Giao diện lập trình cơ bản

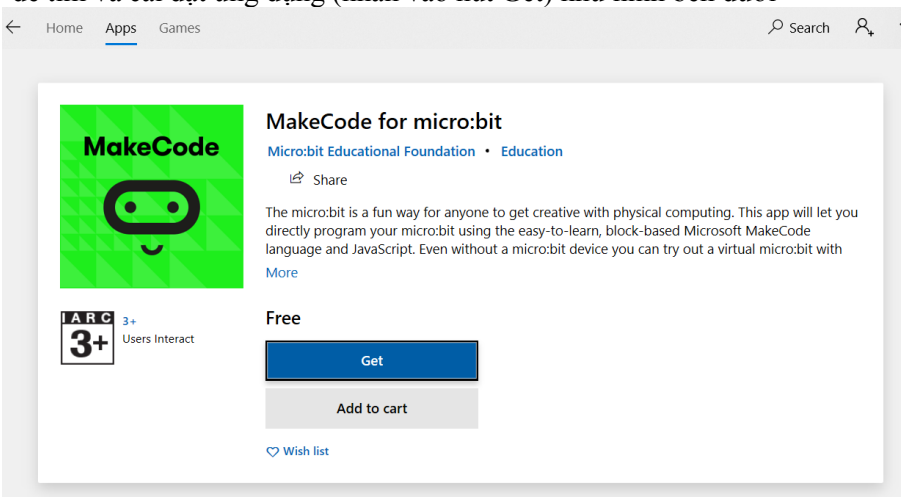
Ưu điểm thứ 2 của việc lập trình trên máy tính là việc nạp chương trình cho mạch MicroBit thực sự rất đơn giản. Chúng ta chỉ cần sao chép file hex vào mạch MicroBit như chép dữ liệu vào một USB bình thường là chương trình đã có thể chạy trực tiếp trên mạch MicroBit.

Chúng ta cũng có một môi trường lập trình ngoại tuyến trên máy tính được hỗ trợ sẵn trên Windows 10. Từ nút Start của Windows, chúng ta vào kho ứng dụng Microsoft Store (xem Hình 5)



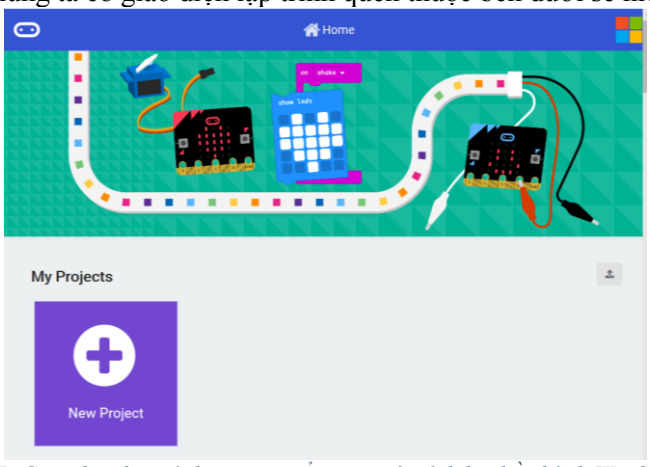
Hình 5: Kho ứng dụng Microsoft Store trên Windows 10

Bước tiếp theo, chúng ta có thể gõ vào ô tìm kiếm (Search) ở góc trên bên phải từ khóa "MakeCode" để tìm và cài đặt ứng dụng (nhấn vào nút Get) như hình bên dưới



Hình 6: Tìm kiếm và cài đặt ứng dụng MakeCode

Sau khi cài đặt xong, chúng ta có giao diện lập trình quen thuộc bên dưới sẽ hiện ra



Hình 7: Giao diện lập trình ngoại tuyến trên máy tính hệ điều hành Windows 10

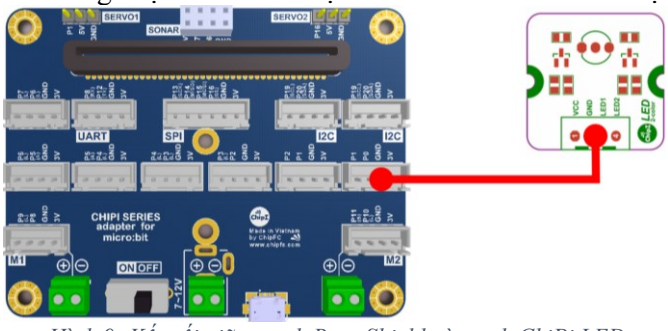
Trong thời gian gần đây, sử dụng điện thoại hoặc máy tính bảng để lập trình cho MicroBit đang trở nên phổ biến do tính tiện dụng của nó. Tuy nhiên trong giáo trình này, chúng tôi chỉ tập trung sử dụng môi trường lập trình MakeCode trên máy tính mà thôi.

3 Chương trình đầu tiên

Trong hướng dẫn này, chúng ta sẽ lập trình điều khiển mô-đun ChiPi LED. Đây là một mô-đun đèn tích hợp có 2 màu cơ bản là đỏ và xanh lá. Tuy nhiên nếu cả 2 đèn này cùng sáng, chúng ta có màu vàng. Chúng ta hãy lưu ý vị trí của 2 chân LED1 và LED2 (xem Hình 8), 2 chân tín hiệu dùng để điều khiển đèn sáng hay tắt.

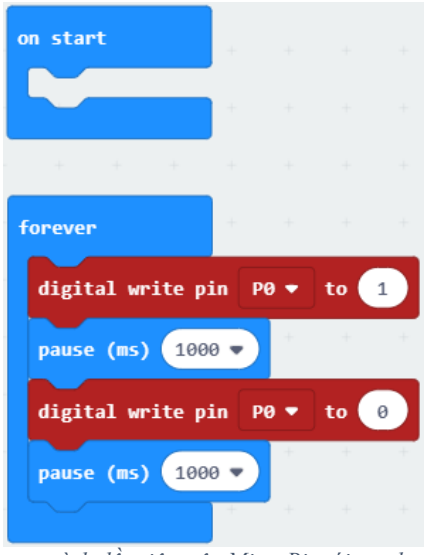
Hình 8: Mô-đun ChiPi LED

Tiếp theo, chúng ta sẽ kết nối module này với mô-đun Base Shield. Trong ví dụ này, chúng ta sẽ kết nối mô-đun LED với chân P1-P0-GND-VCC, như minh họa ở hình bên dưới. Dựa vào màu của dây kết nối, chúng ta sẽ dễ dàng nhận ra LED0 được nối với P0 còn LED1 được nối với chân P1.



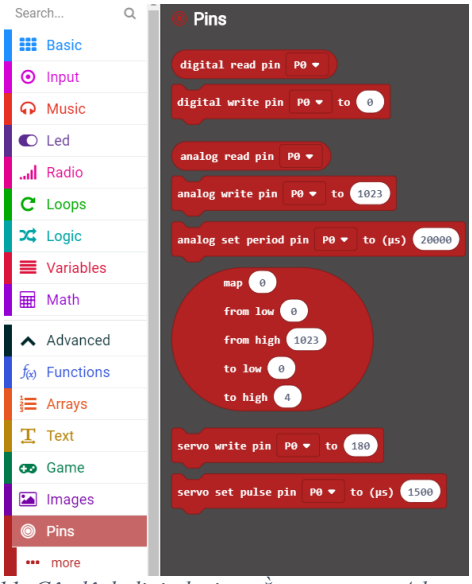
Hình 9: Kết nối giữa mạch Base Shield và mạch ChiPi LED

Chúng ta có thể hiện thực một chương trình nhỏ, điều khiển chân P0 (được nối với LED0) để xem kết quả thực thi của chương trình sẽ như thế nào. Một chương trình gợi ý như sau:



Hình 10: Chương trình đầu tiên trên MicroBit với mạch mở rộng ChiPi

Trong chương trình trên, chúng ta đã sử dụng câu lệnh digital write để xuất tín hiệu ra chân P0, đang được nối với chân LED0 của mô-đun LED ChiPi. Câu lệnh này nằm trong mục Advance, Pins, như trình bày ở Hình 11.

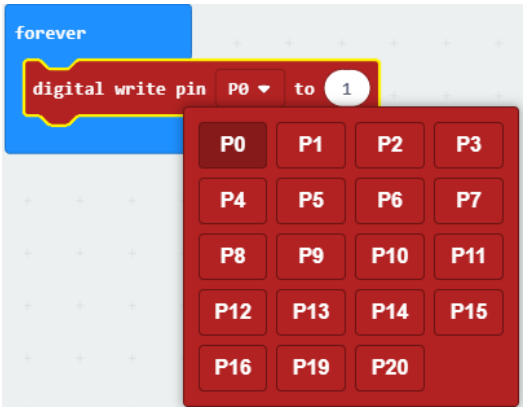


Hình 11: Câu lệnh digital wite, nằm trung mục Advance/Pins

Sau khi hiện thực xong chương trình trên máy tính, chúng ta cần tải chương trình này xuống board mạch MicroBit để xem kết quả. Chúng ta có thể thấy rằng, với chương trình trên, LED đang chớp tắt mỗi giây với màu đỏ.

Với câu lệnh digital write sử dụng ở trên, chúng ta có 2 tùy chọn như sau:

- Chọn chân: Bằng cách nhấn vào phím mũi tên, chúng ta sẽ có rất nhiều chân để kết nối với các thiết bị bên ngoài, như trình bày ở Hình 12.



Hình 12: Lựa chọn chân xuất tín hiệu trong câu lệnh digital write

- Chọn dữ liệu: Ở tùy chọn thứ 2 này, chúng ta chỉ có 2 trạng thái là 0 hoặc 1. Khi xuất ra giá trị 0, sẽ tương đương với điện áp 0V và 1 sẽ tương đương với mức điện áp 3.3V. Do đó, khi chân P0 đang nối với 1 bóng đèn, xuất giá trị 1 đèn sẽ sáng và giá trị 0 đèn sẽ tắt.

4 Bài tập

1. Hiệu chỉnh lại chương trình, để LED chớp tắt mỗi giây với màu xanh.
   Gợi ý: Thay đổi chân P0 thành P1

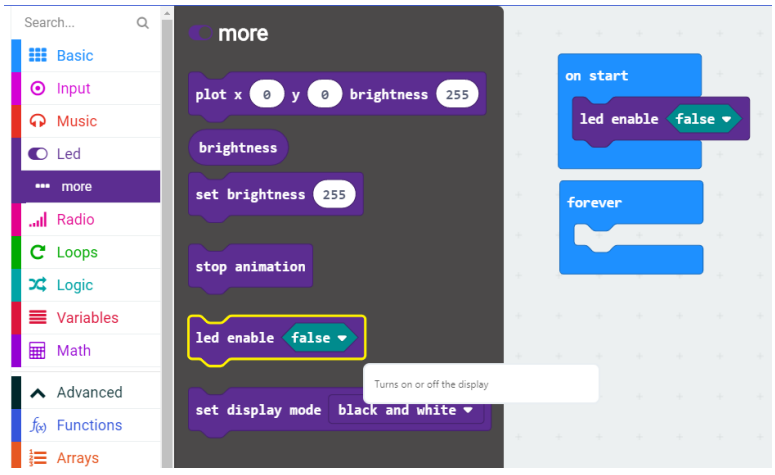
Hình 13: Chương trình điều khiển LED xanh

2. Viết chương trình chuyển màu LED sau mỗi giây theo trình tự: Đỏ - Xanh – Vàng
   Gợi ý: Xuất kết hợp đồng thời 2 chân P0 và P1 để điều khiển màu cho LED. Ví dụ để có màu đỏ: P0 = 1 và P1 = 0, để có màu vàng thì P0 = 1 và P1 = 1

Hình 14: Chương trình điều khiển 3 màu của LED

3. Giáo viên cho học sinh thay đổi chân kết nối với mạch MicroBit và sửa lại chương trình cho phù hợp.
   Trong yêu cầu này, có thể một số chân sẽ không hoạt động đúng với chức năng lập trình của chúng ta. Lý do là các chân này liên quan đến việc điều khiển 25 bóng đèn có sẵn trên mạch MicroBit. Trong trường hợp này, chúng ta khắc phục bằng cách dùng câu lệnh 

, để tắt chức năng của 25 bóng đèn và sử dụng toàn bộ tài nguyên của MicroBit để điều khiển các thiết bị mới. Câu lệnh này được nằm trong mục Led, và chọn tiếp vào mục more, như minh họa ở Hình 15. Chúng ta sẽ đặt câu lệnh này trong khối lệnh on start ở đầu chương trình.



Hình 15: Tắt chức năng của 25 bóng đèn

Bài 2: Điều khiển loa Buzzer

- **MỤC TIÊU**

Học sinh sẽ:

- Làm quen với hệ thống cảm biến ChiPi.
- Hiểu được nguyên lý điều khiển loa ChiPi Buzzer.
- Hiện thực được chương trình phát nhạc với MicroBit và ChiPi Buzzer.
- Phân biệt được 2 dạng xuất dữ liệu: digital và analog.

- **PHÂN BỔ THỜI LƯỢNG**

1 Giới thiệu về loa Buzzer (05 phút)

2 Điều khiển Buzzer cơ bản (15 phút)

3 Điều khiển âm lượng của Buzzer (15 phút)

4 Chơi nhạc với ChiPi Buzzer (10 phút)

1 Giới thiệu về loa Buzzer

ChiPi - Buzzer là một mô-đun có một loa bíp thuộc hệ thống Chipi của ChipFC. Đầu ra là rào cắm 4 chân tương thích với Chipi Base Shield. Với ChiPi Buzzer, chúng ta có thể dễ dàng tạo cho mình một ứng dụng báo động bằng âm thanh hoặc thậm chí là lập trình để phát ra một bản nhạc mong muốn.

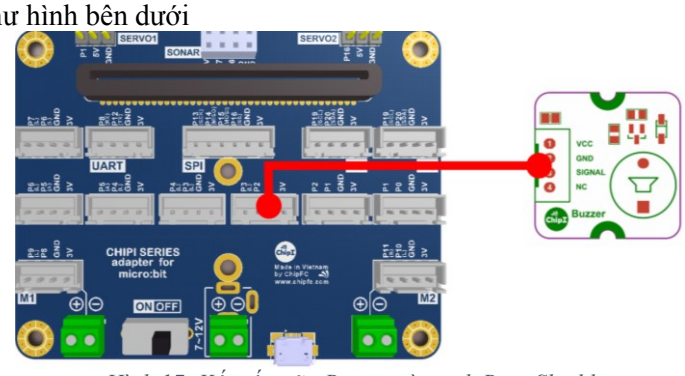
Hình 16: Mô-đun ChiPi Buzzer

Hãy lưu ý vào các chân kết nối của mạch ChiPi Buzzer, ngoài 2 chân nguồn và đất (GND và VCC), chỉ có 1 chân tín hiệu, có kí hiệu là SIGNAL. Chân còn lại (NC) không có kết nối với mạch Arduino. NC là chữ viết tắt của Not Connected, tiếng Anh nghĩa là không kết nối, một thuật ngữ rất phổ biến trong kết nối giữa các mạch điện tử.

Việc điều khiển loa rất đơn giản, giống như việc cho một đèn LED sáng lên ở bài trước. Loa sẽ kêu khi tín hiệu xuất ra là 1 và sẽ không kêu nếu tín hiệu xuất ra là 0.

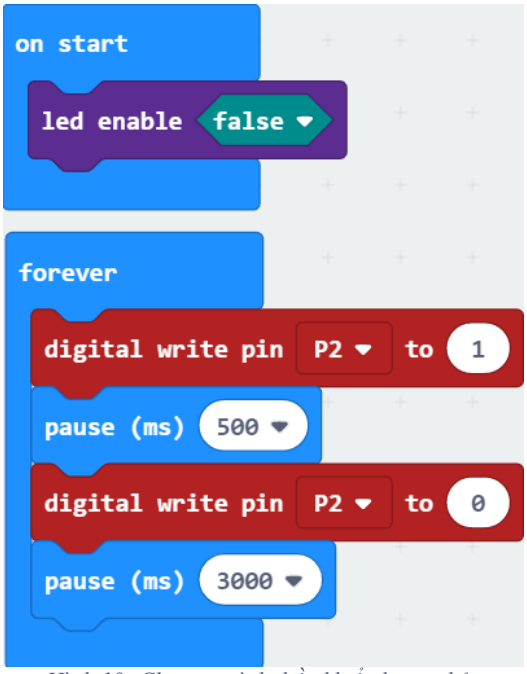
2 Điều khiển Buzzer cơ bản

Theo như nguyên lý giải thích ở trên, chúng ta sẽ xây dựng một chương trình đơn giản để điều khiển Buzzer. Trong ví dụ này, Buzzer sẽ được nối vào khe cắm VCC – GND – P2 – P3, như hình bên dưới



Hình 17: Kết nối giữa Buzzer và mạch Base Shield

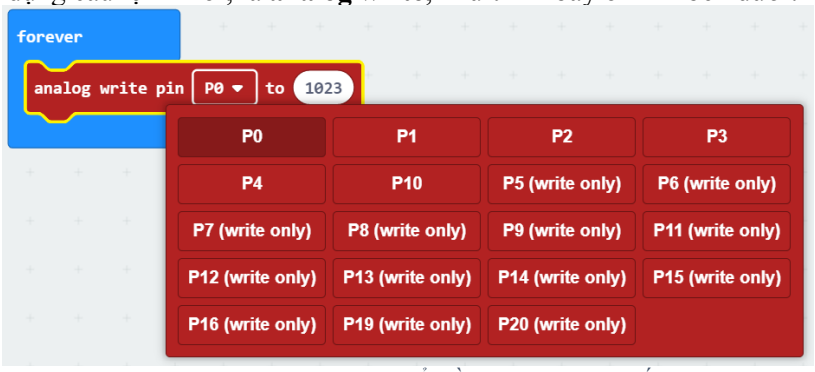
Dựa vào màu của dây kết nối, chúng ta sẽ thấy chân tín hiệu điều khiển loa Buzzer đang nối với chân P2 của MicroBit. Chương trình đơn giản sau đây sẽ cho loa phát ra tiếng kêu trong nửa giây, và không kêu trong 3 giây.



Hình 18: Chương trình điều khiển loa cơ bản

3 Điều khiển âm lượng của Buzzer

Như đã trình bày ở trên, sử dụng câu lệnh set digital chỉ đơn giản là phát ra âm thanh với âm lượng tối đa (dữ liệu 1) và tắt âm thanh (dữ liệu 0 – âm lượng tối thiểu). Trong một số trường hợp, chúng ta cần điều chỉnh âm lượng để giúp ứng dụng thân thiện hơn với người sử dụng. Trong trường hợp này, chúng ta phải sử dụng câu lệnh mới, là **analog write**, như trình bày ở hình bên dưới:



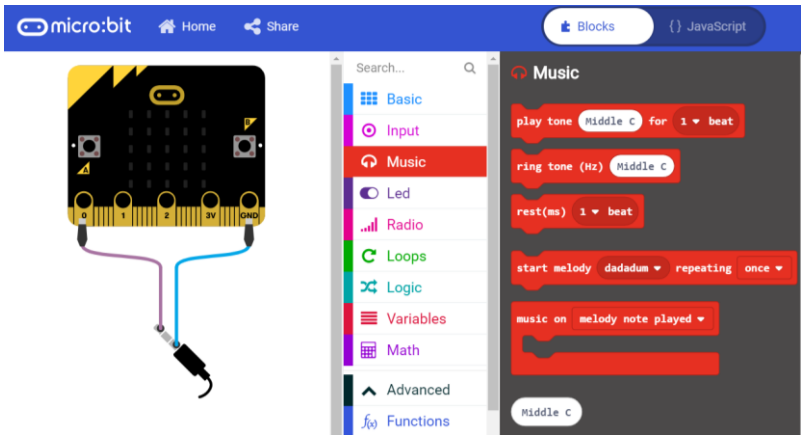
Hình 19: Câu lệnh set pwm để điều chỉnh điện áp xuất ra

Câu lệnh analog write có 2 thông số. Thông số đầu tiên là tên của chân kết nối với Buzzer, ở hướng dẫn này chân P2. Tham số thứ 2 là một danh sách chọn, với các giá trị từ 0 đến 1023. Chúng ta có thể chọn trực tiếp từ danh sách này hoặc đánh 1 con số mà chúng ta mong muốn, miễn sao nó nằm trong đoạn [0 – 1023]. Trong chương trình gợi ý ở Hình 20, âm lượng của loa được giảm chỉ còn 500 đơn vị, bằng cách thay thế câu lệnh digital write bằng analog write. Bạn hãy thử nạp chương trình này vào và rõ ràng âm lượng từ loa sẽ nhỏ hơn nhiều so với chương trình đầu tiên (chương trình ở Hình 18).

Hình 20: Điều chỉnh âm lương bằng câu lệnh analog write

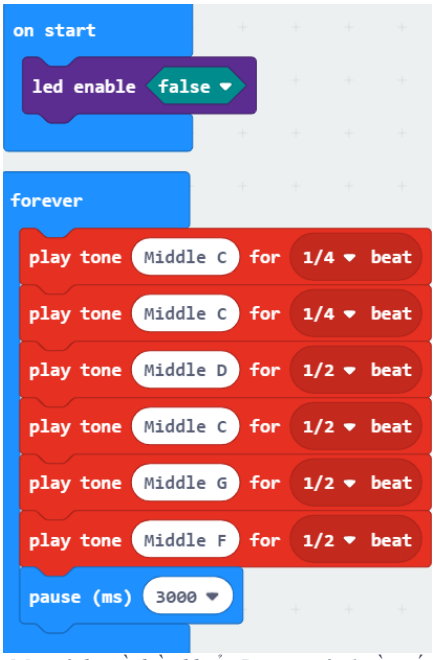
Chi tiết hơn, câu lệnh digital write chỉ xuất là được 2 trạng thái là HIGH và LOW, tương ứng với 2 mức điện áp, tối đa 3V và tối thiểu 0V. Do chỉ có 2 trạng thái như vậy, nên câu lệnh set digital còn được gọi là câu lệnh xuất **dữ liệu đầu ra ở dạng số** (digital). Ngược lại, khi dùng câu lệnh analog write, đầu ra có thể **tinh chỉnh mức điện áp đầu ra**. Cụ thể, điện áp đầu ra từ 0V đến 3V sẽ được chia đều thành 1024 giá trị: giá trị 0 tương ứng với 0V và giá trị 1023 tương ứng với 3V. Khi bạn muốn xuất đầu ra ở mức điện áp 1.5V, giá trị cần sử dụng trong câu lệnh analog write sẽ là 512. Do đầu ra có nhiều trạng thái, và mức điện áp giữa 2 trạng thái liền kề rất gần nhau, nên câu lệnh này còn được gọi là xuất dữ liệu đầu ra ở dạng analog, tạm dịch là **dữ liệu tương tự**. Với câu lệnh analog write này, chúng ta có thể điều khiển âm lượng của loa, điều khiển được độ sáng của bóng đèn, và trong các hướng dẫn tiếp theo, chúng ta có thể điều khiển được tốc độ của động cơ.

4 Chơi nhạc với ChiPi Buzzer



Hình 21: Các câu lệnh trong mục Music

Dựa trên trên nguyên lý của câu lệnh analog write, bằng cách xuất ở đầu ra lúc có tín hiệu, lúc không có tín hiệu với 1 chu kì xác định, chúng ta có thể tạo ra một nốt nhạc. Rất may mắn cho chúng ta, là việc xuất dữ liệu như vậy đã được hỗ trợ sẵn bởi câu lệnh play tone, câu lệnh đầu tiên trong mục Music (xem Hình 21). Chúng ta có thể hiện thực đoạn chương trình sao để xem kết quả chạy trên ChiPi Buzzer.



Hình 22: Một ví dụ về điều khiển Buzzer với 1 tần số nhất định

Đây chính là đoạn nhạc mở đầu của bài Happy Birthday, với **Middle C** là nốt Đô chuẩn, **Middle D** là nốt Rê, **Middle G** là nốt Sol và **Middle F** là nốt Pha. Câu lệnh play tone này có tới 2 thông số: đầu tiên là nốt nhạc cần xuất ra loa, tiếp theo là thời gian ngân của nốt đó tính theo đơn vị giây.

Chú ý: để sử dụng câu lệnh play tone này, loa phải được nối vào chính xác chân P0, như mô phỏng của MakeCode ở Hình 21. Do đó, chúng ta cần phải kết nối giữa mạch ChiPi Buzzer với cổng VCC-GND-P0-P1 nếu muốn sử dụng câu lệnh play tone.

Như vậy, ngoài việc chỉ phát ra một tiếng báo động khá đơn điệu, chúng ta hoàn toàn có thể thay thế nó bằng một bài nhạc, làm tăng tính thân thiện của ứng dụng đối với người dùng. Các bạn có thể tiếp tục hoàn thiện chương trình để phát một bản nhạc hoàn chỉnh.

Tuy nhiên, MakeCode cũng hỗ trợ cho chúng ta những bản nhạc có sẵn, bằng cách sử dụng câu lệnh start melody, như ví dụ ở chương trình dưới đây:

Hình 23: Chương trình phát nhạc với các bài hỗ trợ sẵn trên MakeCode

Lưu ý: Không được đặt câu lệnh start melody trong khối lệnh forever. Trong trường hợp này bài nhạc sẽ được lặp lại ngay lập tức và chúng ta chỉ nghe được một đoạn nhỏ mà thôi.

Bài 3: Nhận dữ liệu từ nút nhấn

- **MỤC TIÊU**

Học sinh sẽ:

- Làm quen với các thiết bị nhận dữ liệu.
- Lấy được dữ liệu từ nút nhấn đơn giản.
- Lập trình chống nhiễu cho nút nhất.
- Xây dựng được ứng dụng sử dụng nút nhấn.

- **PHÂN BỔ THỜI LƯỢNG**

1 Giới thiệu về nút nhấn (05 phút)

2 Chương trình nút nhấn đơn giản (10 phút)

3 Chương trình nút nhấn nâng cao (15 phút)

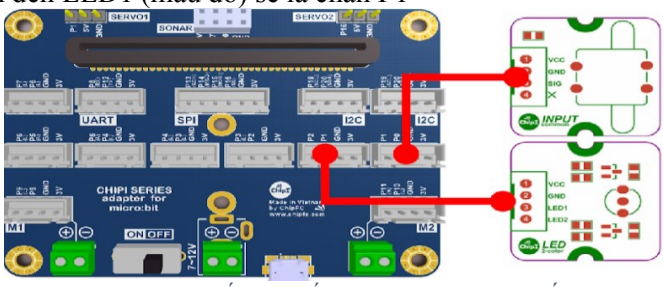
4 Nút nhấn cảm ứng (15 phút)

1 Giới thiệu về nút nhấn

Nút nhấn là một thiết bị đầu vào cơ bản nhất. Sử dụng nút nhấn, chúng ta có thể nhận tương tác từ phía người dùng. Mô-đun nút nhấn được thiết kế hoàn toàn tương thích với Base Shield ChiPi, như hình ảnh bên dưới.

Hình 1: Mô-đun nút nhấn

Khi kết nối mô-đun này vào mạch Base Shield, hãy chú ý chân của Arduino sẽ kết nối với chân IN (hoặc SIG) của nút nhấn. Như minh họa ở Hình 2, nút nhấn được nối vào chân P0 còn đèn LED1 (màu đỏ) sẽ là chân P1



Hình 2: Kết nối nút nhấn và đèn LED vào hệ thống

2 Chương trình nút nhấn đơn giản

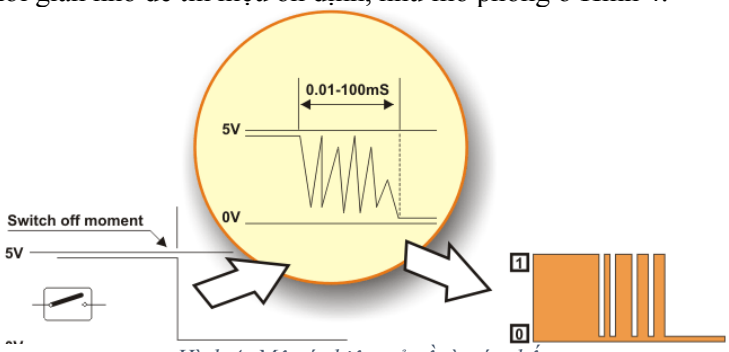
Trong phần này, chúng ta sẽ hiện thực một đoạn chương trình ngắn với chức năng như sau: Khi nút nhấn được nhấn, đèn sẽ sáng. Ở trạng thái bình thường, đèn sẽ tắt. Chương trình gợi ý sẽ như sau:

Hình 3: Chương trình nút nhấn đơn giản

Ở chương trình trên, chúng ta cần phải xài câu lệnh set pull pin P0 to up trong phần on start. Câu lệnh này có chức năng cấu hình 1 chân là trạng thái nhập dữ liệu, nằm trong mục Pins, chọn tiếp more. Để nhận dữ liệu bằng câu lệnh digital read, chúng ta bắt buộc phải có câu lệnh này. Cuối cùng, trong chương trình trên, chúng ta dùng câu lệnh điều kiện if để kiểm tra xem nút nhấn nối với 1 chân nào đó có được nhấn hay không và bật tắt đèn màu đỏ một cách tương ứng: Nút nhấn thì đèn sáng, ngược lại đèn sẽ tắt.

3 Chương trình nút nhấn nâng cao

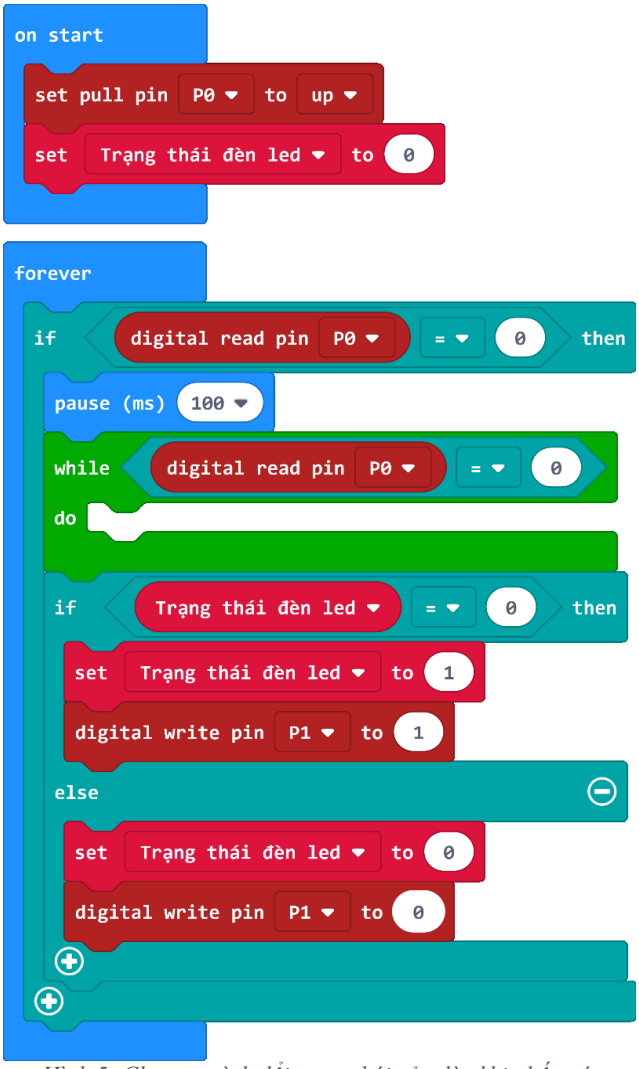
Ở phần này, chúng ta sẽ cải thiện chương trình ở bên trên. Mỗi lần nhấn nút sẽ đổi trạng thái của đèn LED như sau: nếu đèn đang tắt thì sáng, ngược lại đèn đang sáng sẽ tắt đi.
Để hiện thực được yêu cầu như trên, chúng ta sẽ tập trung vào phần đầu tiên của chương trình ở Hình 3, là phần nút nhấn được nhấn. Tiếp theo, chúng ta sẽ phải chờ cho đến khi nút nhấn được thả ra thì mới điều khiển bóng đèn. Tuy nhiên trước khi kiểm tra nút nhấn có được thả ra hay chưa, chúng ta cần phải đợi 1 khoảng thời gian nhỏ để tín hiệu ổn định, như mô phỏng ở Hình 4.



Hình 4: Một tín hiệu trả về từ nút nhấn

Do những rung động về cơ khí, thời gian từ lúc nút được nhấn cho đến khi tín hiệu ổn định có thể kéo dài đến 100ms. Cuối cùng, chúng ta cần 1 biến, chúng ta có thể đặt tên biến là tiếng Việt, chẳng hạn như **Trạng thái đèn LED**, để lưu trạng thái của đèn LED.

Chương trình gợi ý của chúng ta như sau.



Hình 5: Chương trình đổi trạng thái của đèn khi nhấn nút

Do trong MakeCode không có câu lệnh đợi cho đến khi, chúng ta phải dùng câu lệnh **while do** để thực hiện chức năng tương đương. **Tức là khi nút đang được nhấn, chúng ta không làm gì cả**. Khi nút được trả ra, các câu lệnh phía sau sẽ được thực hiện. Điều này có nghĩa là, đợi cho đến khi nút nhấn được thả thì thực hiện các câu lệnh tiếp theo.

4 Nút nhấn cảm ứng

Thay vì sử dụng nút nhấn cơ, như hướng dẫn ở trên, chúng ta có một dạng nút nhấn cảm ứng ChiPi Touch. Mô-đun này được thiết kế có chuẩn đầu ra như một nút nhấn bình thường. Hình ảnh của mô-đun này được trình bày ở hình bên dưới.

Hình 6: Nút nhấn cảm ứng ChiPi Touch

Chúng ta chỉ cần thay thế nút nhấn cơ bằng nút nhấn cảm ứng này thì chương trình vẫn chạy bình thường. Đây là một ưu điểm của hệ thống ChiPi, các mô-đun có độ tương thích cao và dễ dàng thay thế cho nhau.

Bài 4: Cảm biến cường độ ánh sáng

- **MỤC TIÊU**

Học sinh sẽ:

- Làm quen với các thiết bị nhận dữ liệu.
- Hiểu được khái niệm và dữ liệu tương tự (dữ liệu analog).
- Lấy được dữ liệu từ cảm biến ánh sáng.
- Xây dựng được ứng dụng sử dụng cảm biến ánh sáng.

- **PHÂN BỔ THỜI LƯỢNG**

1 Giới thiệu về cảm biến ánh sáng (10 phút)

2 Chương trình đọc dữ liệu từ cảm biến (15 phút)

3 Phát triển ứng dụng điều khiển đèn (20 phút)

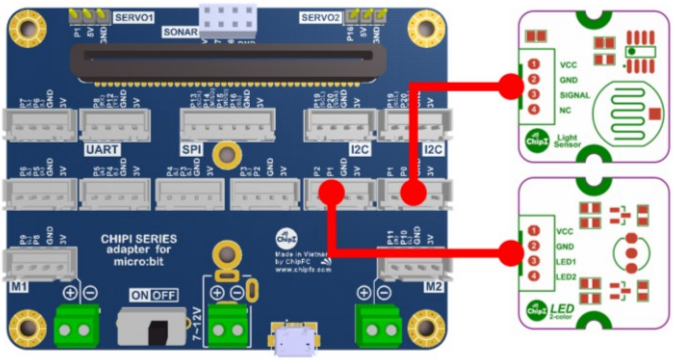
1 Giới thiệu về cảm biến ánh sáng

Cảm biến ánh sáng trong hướng dẫn này được thiết kế dựa trên nguyên lý của quang trở, một loại điện trở có trở kháng thay đổi khi cường độ ánh sáng chiếu vào nó thay đổi. Dựa vào nguyên lý này, điện áp đầu ra của cảm biến sẽ thay đổi khi cường độ ánh sáng thay đổi.

Hình 24: Mô-đun cảm biến ánh sáng

Mặc dù mạch microbit đã tích hợp sẵn cảm biến về cường độ ánh sáng, việc triển khai nó vào ứng dụng thực tế là không khả thi. Lý do là mạch MicroBit sử dụng 25 bóng đèn là chính cảm biến ánh sáng. Vì các bóng đèn này được thiết kế tích hợp trên mạch MicroBit, nên việc triển khai vào ứng dụng thực tế sẽ khó khăn cho việc lắp đặt. Thêm nữa, công nghệ cảm biến trên MicroBit rất đắt tiền so với công nghệ sử dụng quang trở. Do đó, cảm biến dựa trên quang trở thích hợp để triển khai cho các ứng dụng thực tế hơn.

Khác với nút nhấn, dữ liệu trả về chỉ có 2 trạng thái là LOW và HIGH, dữ liệu trả về từ cảm biến ánh sáng, cũng như nhiều loại cảm biến khác, sẽ có 1024 trạng thái khác nhau. Cũng vì lý do này, mà dữ liệu trả về từ nút nhấn được gọi là dữ liệu dạng digital, còn dữ liệu từ cảm biến ánh sáng là dữ liệu analog. Trong bài này, chúng ta sẽ sử dụng cảm biến ánh sáng này, để mô phỏng 1 ứng dụng bật tắt đèn tự động như sau: Khi trời tối, đèn sẽ tự động bật sáng, ngược lại đèn sẽ tắt để tiết kiệm điện. Chúng ta cũng phải xem xét trường hợp cảm biến bị nhiễu bởi ánh đèn ô tô hắt vào lúc trời tối.

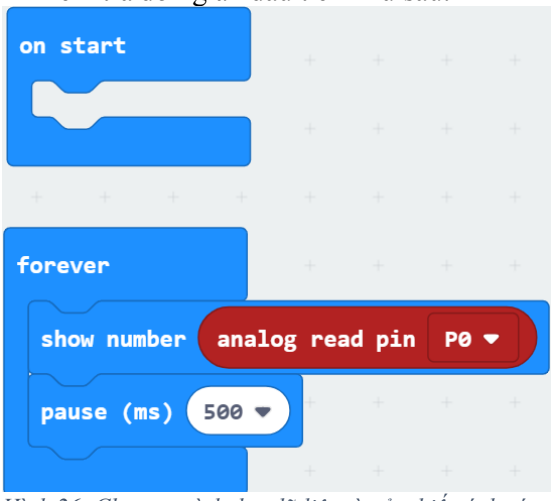


Hình 25: Kết nối cảm biến ánh sáng và đèn LED vào hệ thống

Một gợi ý cho việc kết nối các phần cứng cho yêu cầu của bày này được trình bày ở Hình 25. Cảm biến ánh sáng được nối vào chân P0 còn đèn điều khiển được nói vào chân P1 và P2.

2 Chương trình đọc dữ liệu từ cảm biến

Như đã trình bày ở trên, dữ liệu trả về từ cảm biến là dạng analog, nên chúng ta sẽ dùng câu lệnh analog read pin. Chương trình kiểm tra đơn giản đầu tiên như sau:



Hình 26: Chương trình đọc dữ liệu từ cảm biến ánh sáng

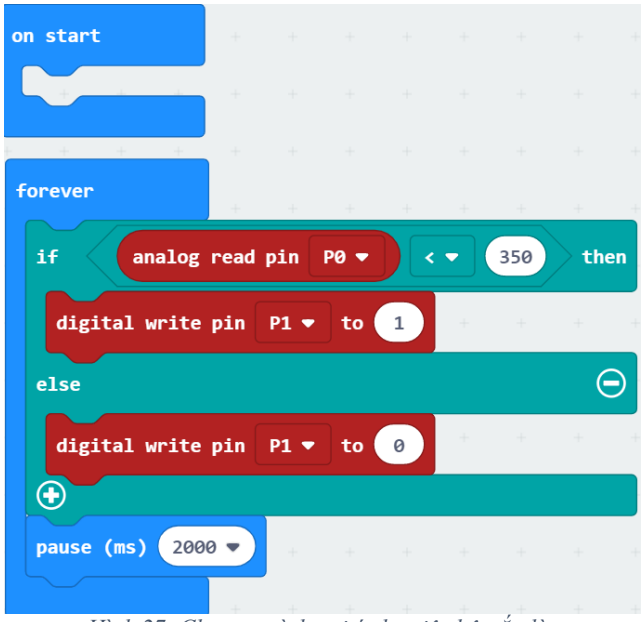
Đầu tiên, chúng ta sẽ xuất kết quả cảm biến ra màn hình 25 đèn led, nên **câu lệnh *led enable false* sẽ không được dùng**. Thêm nữa, việc xuất ra màn hình liên tục cũng không thực sự cần thiết, nên chúng ta sẽ đợi nửa giây mới xuất 1 lần.

Chúng ta có thể kiểm tra sự thay đổi của cường độ ánh sáng bằng cách dùng ngón tay bịt quang trở lại. Lúc này, giá trị hiển thị sẽ giảm đi rất nhiều. Bên cạnh đó, chúng ta cũng cần thống kê giá trị trả về của cảm biến khi trời sáng và khi trời tối, để chuẩn bị dữ liệu cho phần lập trình tiếp theo.

Bài tập trên lớp: Giáo viên cho học sinh đo 10 lần giá trị của cảm biến khi trời sáng và 10 lần giá trị khi trời tối. Sau đó tính ra giá trị trung bình cho 2 trạng thái này.

3 Phát triển ứng dụng điều khiển đèn

Trong phần này, chúng ta sẽ hiện thực ứng dụng tự động bật đèn dựa vào điều kiện ánh sáng. Ứng dụng này minh họa cho việc điều khiển đèn đường tự động: Đèn sẽ sáng khi trời tối và sẽ tắt khi trời sáng. Giả sử, dữ liệu nhận về từ cảm biến khi "trời tối" là 200 đơn vị, còn "trời sáng" là 500 đơn vị. Chúng ta sẽ chọn 1 ngưỡng ở giữa 200 và 500, chẳng hạn như 350, để quyết định bật tắt đèn. Chương trình đơn giản bước đầu sẽ như sau:

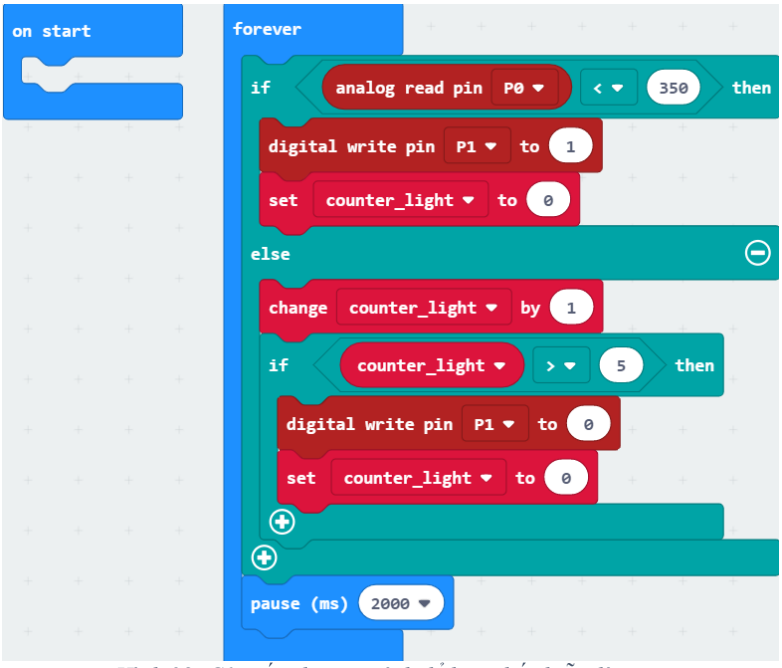


Hình 27: Chương trình gợi ý cho việc bật tắt đèn

Chúng ta không cần phải kiểm tra điều kiện một cách liên tục, nên câu lệnh đợi 2s được dùng ở cuối chương trình. Tuy nhiên, điều này cũng sẽ làm giảm độ nhạy của hệ thống. Trong trường hợp xấu nhất, khi che cảm biến lại, 2 giây sau đèn mới tắt. Tuy nhiên đối với một ứng dụng thực tế, đây là điều bình thường.

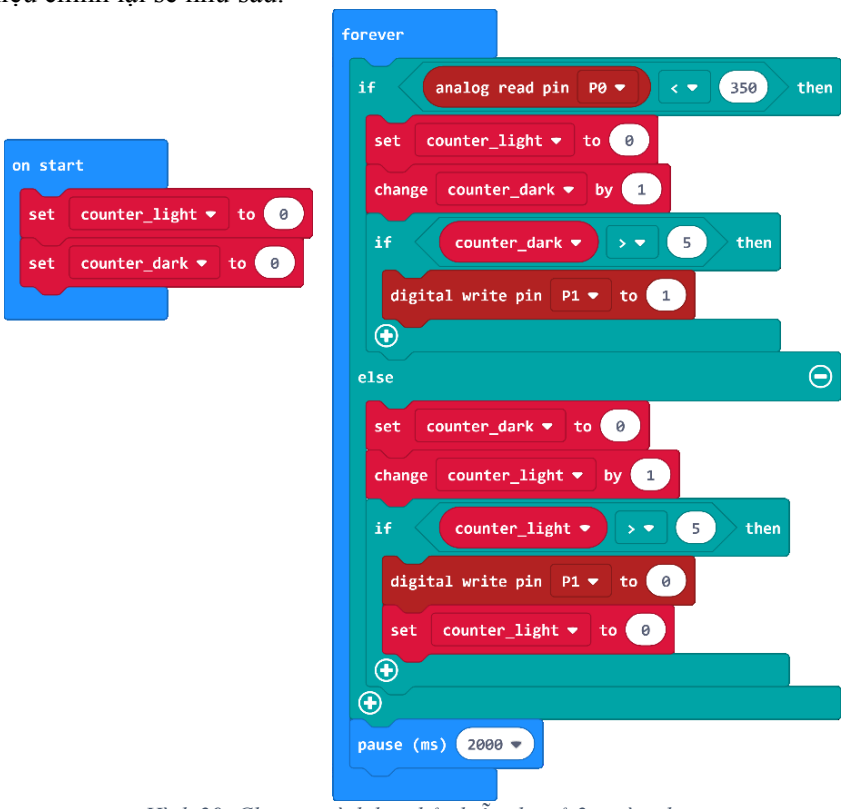
Với chương trình ở trên, khi có ánh đèn xe ô tô hắt vào tại thời điểm hệ thống kiểm tra cảm biến ánh sáng, thì nó sẽ tắt đèn. Điều này sẽ không hợp lý khi triển khai ứng dụng này trong điều kiện thực tế. Do đó, chúng ta sẽ phải kiểm tra điều này trước khi quyết định tắt đèn. Bằng cách khai báo thêm 1 biến số (có tên là **counter_light**), với giá trị tăng lên 1 đơn vị. Và khi biến này lớn hơn 1 giá trị định trước, ví dụ là 5 đơn vị, thì mới quyết định bật đèn. Việc này sẽ tiếp tục làm giảm độ nhạy của hệ thống, cụ thể là 10s, nhưng điều này sẽ giải quyết được hiện tượng ánh sáng nhiễu từ phía đèn ô tô giao thông.

Chương trình gợi ý cho chức năng cải tiến này được trình bày Hình 28. Cần lưu ý là phải gán biến này về 0 khi trời tối.



Hình 28: Cải tiến chương trình để hạn chế nhiễu đèn ô tô

Cũng tương tự hiện tượng nhiễu đèn ô tô, trước khi bật đèn, chúng ta cũng phải đảm bảo là trời thật sự đã tối. Một số trường hợp có thể dẫn ra nhiễu là khi một vật lạ bay vướng vào cảm biến chẳng hạn. Tương tự như trường hợp trên, chúng ta cũng sẽ khai báo một biến số, có tên là counter_dark. Chương trình hiệu chỉnh lại sẽ như sau.



Hình 29: Chương trình loại bỏ nhiễu cho cả 2 trường hợp

Giáo viên có thể cho học sinh làm trường hợp thứ 2 như 1 bài tập về nhà.

Bài 5: Giới thiệu thư viện mở rộng cho MicroBit

(Bài tham khảo)

- **MỤC TIÊU**

Học sinh sẽ:

- Làm quen với NPNBot – Robot dựa trên MicroBit
- Sử dụng được thư viện hỗ trợ cho MicroBit
- Viết được chương trình cơ bản trên NPNBot
- Nạp được chương trình và cho Robot vận hành cơ bản
- Làm quen với bộ thư viện cho nhà thông minh

- **PHÂN BỔ THỜI LƯỢNG**

1 Giới thiệu lập trình NPNBot (05 phút)

2 Chương trình điều khiển động cơ (05 phút)

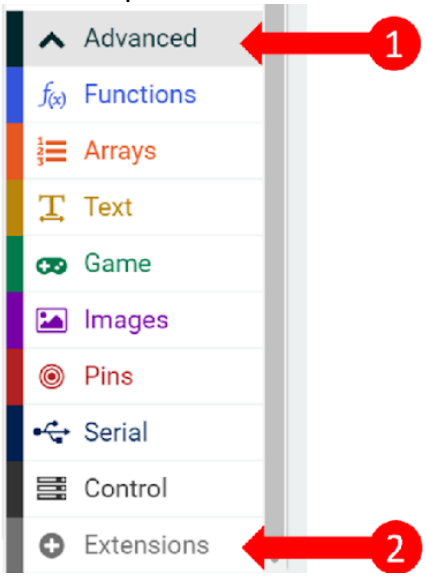
3 Nạp chương trình cho Robot (10 phút)

4 Phát triển dự án Robot (10 phút)

5 Thư viện cho Nhà thông minh (15 phút)

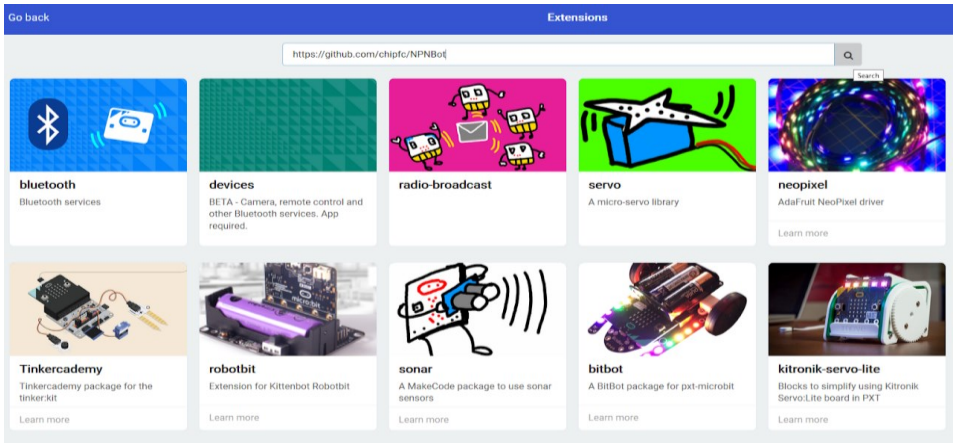
1 Giới thiệu lập trình NPNBot

Để thuận tiện cho việc tiếp cận nhanh với Robot, chúng tôi hỗ trợ cho người dùng bộ thư viện chuyên biệt cho Robot này, cũng có tên là NPNBot. Để thêm thư viện này, Trước tiên, từ cửa sổ lập trình, chúng ta chọn và Advanced, sau đó chọn tiếp Extensions, như minh họa ở Hình 1.

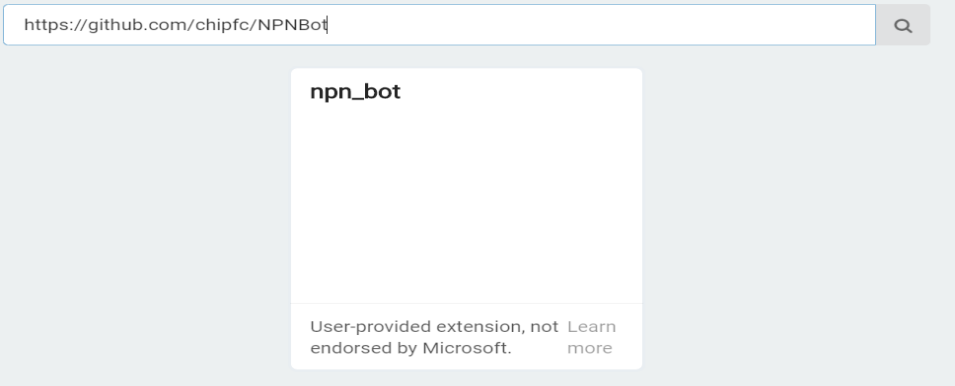


Hình 1: Thêm thư viện mở rộng

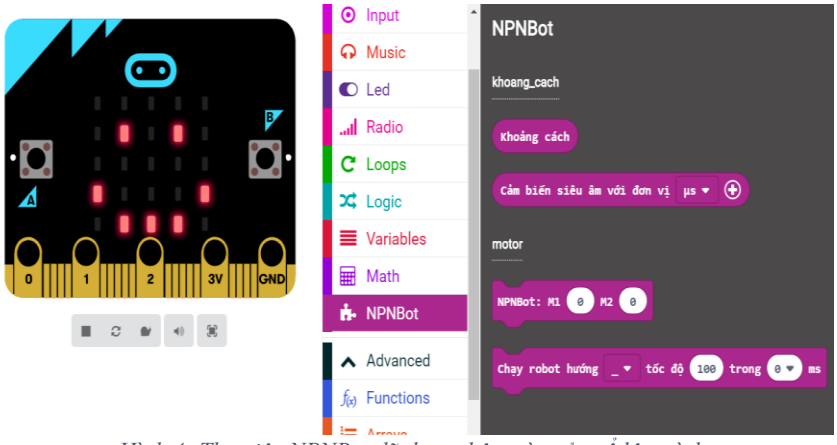
Giao diện như Hình 2 sẽ hiện ra, để chúng ta có thể thêm thư viện. Để dùng với NPNBot, chúng ta phải nhập chính xác vào ô tìm kiếm đường liên kết https://github.com/chipfc/NPNBot. Sau đó nhấn vào nút Tìm kiếm. Kết quả của việc tìm kiếm sẽ hiện ra như Hình 3. Nhấn vào đó, thư viện NPNBot sẽ được tự động thêm vào cửa sổ lập trình, như kết quả ở Hình 4.



Hình 2: Giao diện thêm thư viện mở rộng trên MicroBit

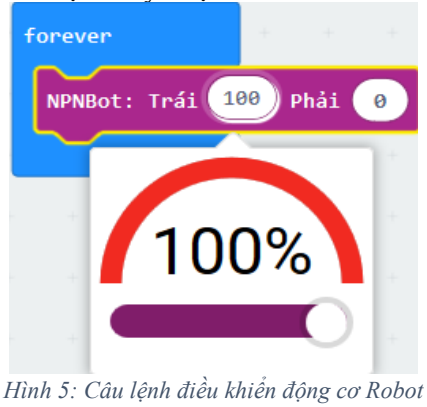


Hình 3: Kết quả tìm kiếm thư viện NPNBot



Hình 4: Thư viện NPNBot đã được thêm vào cửa sổ lập trình

Trong chương trình đầu tiên này, chúng ta chỉ cần quan tâm tới câu lệnh dùng để điều khiển tốc độ của 2 động cơ trên Robot. Câu lệnh này được trình ở Hình 5.



Hình 5: Câu lệnh điều khiển động cơ Robot

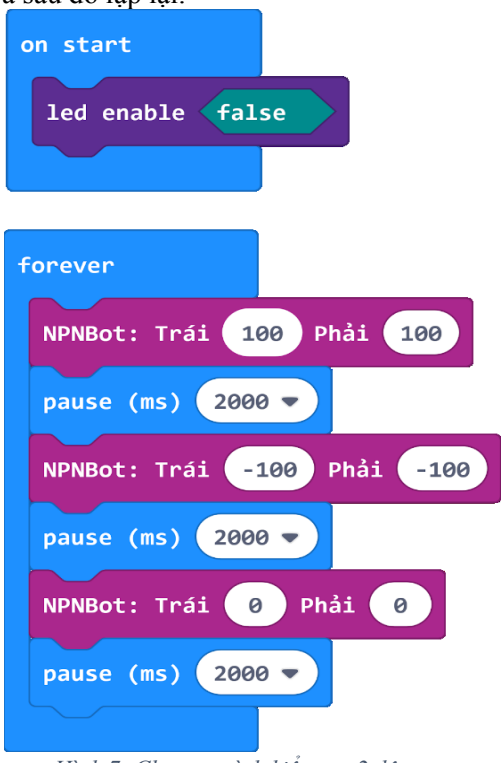
Câu lệnh này cần 2 tham số đưa vào là tốc độ bánh trái và tốc độ bánh phải tương ứng với động cơ M1 và M2. Bạn có thể nhập số hoặc kéo thanh trượt để chỉnh tốc độ. Khi tốc độ là số dương, động cơ sẽ quay tới và ngược lại, động cơ sẽ quay lui với số âm.

2 Chương trình điều khiển động cơ

Để đảm bảo việc xuất dữ liệu ra các chân mở rộng của MicroBit chính xác, chúng ta cần phải tắt chức năng hiển thị trên màn hình 25 bóng đèn của MicroBit. Việc này có thể được thực hiện bằng câu lệnh led enable false, nằm trong mục more của khối lệnh Led, như minh họa ở Hình 6.

Hình 6: Tắt màn hình hiển thị 25 LED của MicroBit

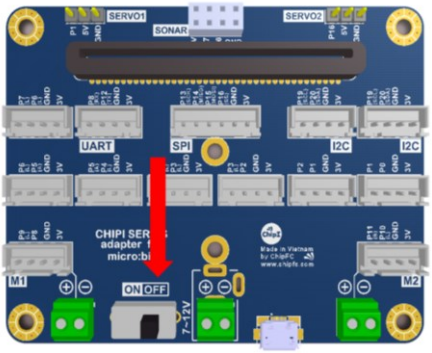
Tiếp theo, chúng ta có thể hiện thực một chương trình nhỏ để kiểm tra 2 động cơ, như minh họa ở Hình 7. Chương trình này sẽ chạy 2 bánh tới trong 2 giây, sau đó chạy 2 bánh lui trong 2 giây, sau đó dừng lại trong 2 giây và sau đó lặp lại.



Hình 7: Chương trình kiểm tra 2 động cơ

3 Nạp chương trình cho Robot

Trước khi kết nối dây USB từ máy tính và mạch MicroBit trên Robot, cần đảm bảo nguồn chính từ PIN của Robot đã được tắt (công tắc nguồn nằm ở vị trí OFF).



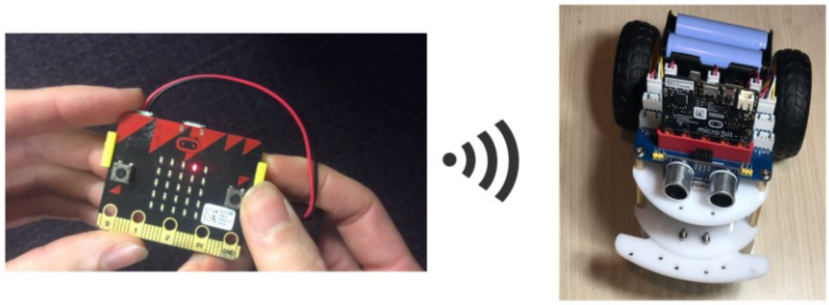
Hình 8: Tắt công tắt nguồn từ USB để nạp chương trình

Khi chương trình đã nạp xong, chúng ta có thể rút dây kết nối USB, bật công tắc nguồn để thử chương trình.

Việc tắt nguồn từ PIN là rất quan trọng, đặc biệt là nếu chúng ta nạp chương trình từ điện thoại. Lúc này chúng ta phải chuyển mạch MicroBit sang chế độ Bluetooth. Trong chế độ này, mạch sẽ xuất xung liên tục ra các chân mở rộng và ảnh hưởng rất lớn đến độ bền của động cơ. Do đó, để an toàn, **chúng ta phải luôn tắt nguồn khi nạp. Sau khi nạp thành công mới mở nguồn điện**.

4 Phát triển dự án Robot

Với sự hỗ trợ của thư viện lập trình NPNBot, chúng ta có thể hiện thực một dự án điều khiển không dây cho Robot, như minh họa ở hình bên dưới:

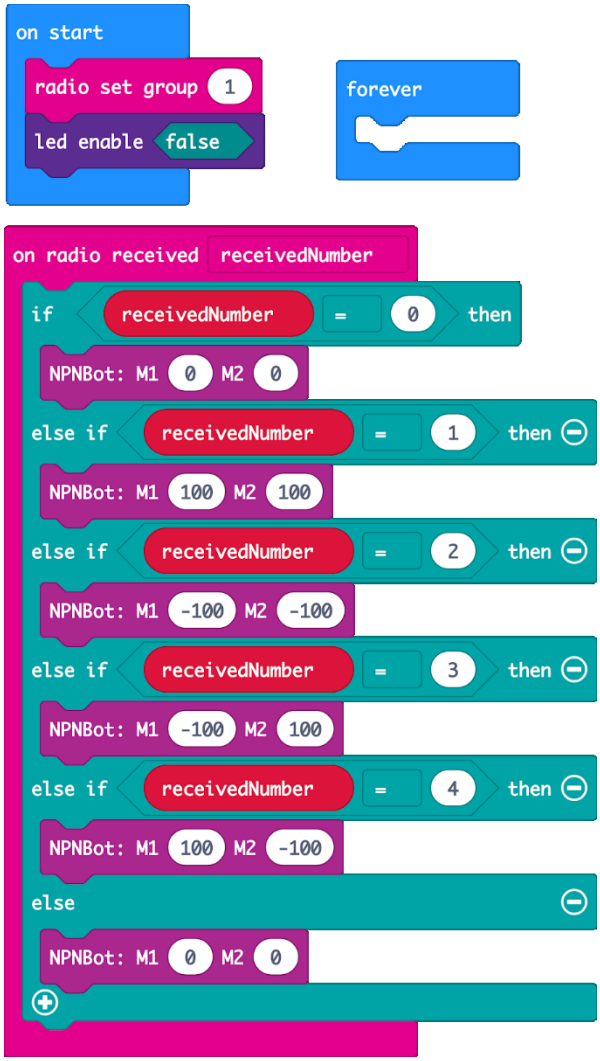


Hình 9: Mô hình điều khiển không dây giữa mạch MicroBit và Robot

Chúng ta sẽ quy ước các "thông điệp" gửi từ mạch MicroBit tới Robot là các con số, mỗi con số tương ứng với 1 chức năng, được mô tả như sau:

- 0: Dừng lại
- 1: Đi tới
- 2: Đi lùi
- 3: Rẽ trái
- 4: Rẽ phải

Với những quy ước như trên, chương trình dành cho Robot sẽ như hình bên dưới:



Hình 300: Chương trình cho NPNBot: Xử lý trong hàm nhận dữ liệu on radio received number

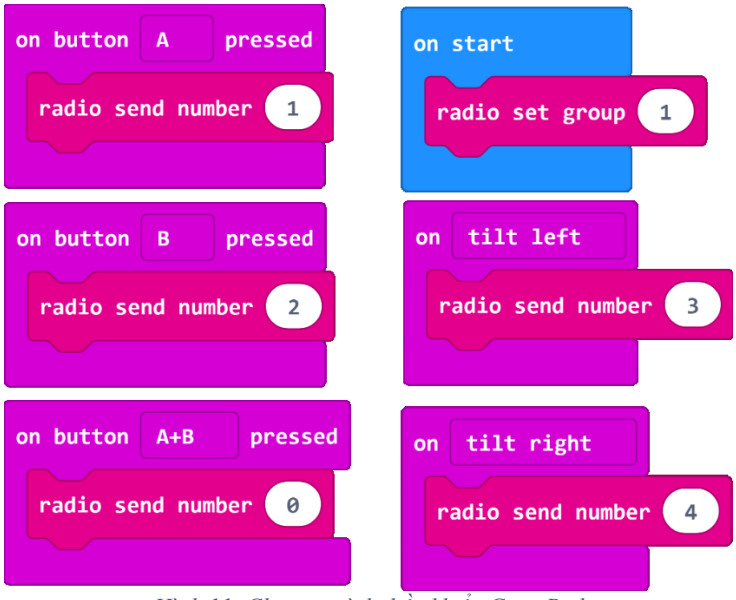
Có 3 điều cần lưu ý về chương trình này:

- Chỉnh nhóm cho việc giao tiếp không dây: Chỉ những mạch có cùng nhóm mới có thể gửi nhận dữ liệu. Việc chỉnh nhóm bằng câu lệnh set radio group chỉ cần được thực hiện 1 lần, nên chúng ta sẽ để nó trong khối lệnh on start.
- Thực thi việc di chuyển trong hàm on radio received number: Chúng ta đang gửi nhận dữ liệu số, nên phải xử lý trong hàm nhận số mà thôi.

Đến đây, chúng ta có thể nạp chương trình cho Robot. Cần lưu ý là phải tắt nguồn điện từ PIN trước khi nạp chương trình.

Nguyên lý của chương trình bên GamePad rất đơn giản, làm sao để gửi được 5 con số, tương ứng với 5 hành vi khác nhau từ người điều khiển là được. Chính vì vậy, đây là chương trình có thể được sáng tạo ra rất nhiều phiên bản từ phía người dùng, tùy vào việc chúng ta định nghĩa khi nào truyền số 0, khi nào là đi tới (số 1).

Tuy nhiên, như quy ước của việc giao tiếp không dây, chúng ta cũng phải chỉnh nhóm cho chương trình GamePad trong khối lệnh on start giống như nhóm của Robot.

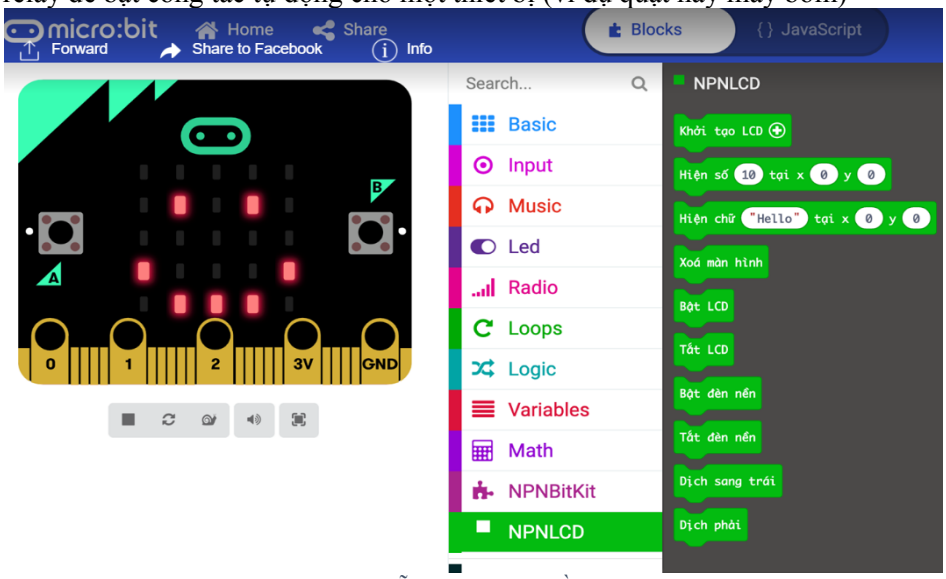


Hình 11: Chương trình điều khiển GamePad

Với chương trình gợi ý ở Hình, nhấn nút A sẽ đi tới, nút B đi lùi, nhấn A và B cùng lúc Robot sẽ dừng. Để rẽ trái và rẽ phải, người dùng sẽ nghiêng mạch MicroBit qua trái hoặc qua phải tương ứng. Rõ ràng, việc điều khiển này là không thực sự dễ dàng và thuận lợi. Người dùng sẽ phải tự định nghĩa một cách điều khiển thuận tiện nhất cho mình.

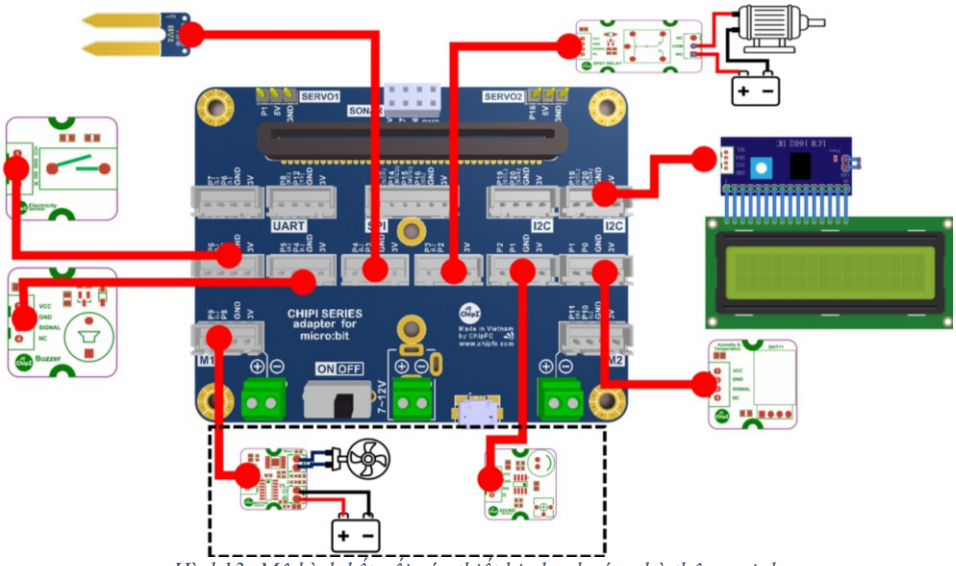
5 Thư viện cho Nhà thông minh

Mạch mở rộng cho mạch MicroBit có thể được sử dụng để phát triển dự án nhà thông minh. Thư viện lập trình này đang được phát triển ở đường dẫn https://github.com/chipfc/NPNBitKit. Sau khi thêm thư viện này vào, chúng ta có thêm đến 2 khối lệnh mới: Khối NPNLCD, hỗ trợ việc xuất dữ liệu ra một màn hình LCD và NPNBitKit để truy xuất các mô đun cao cấp khác như cảm biến nhiệt độ, độ ẩm, điều khiển relay để bật công tắc tự động cho một thiết bị (ví dụ quạt hay máy bơm)



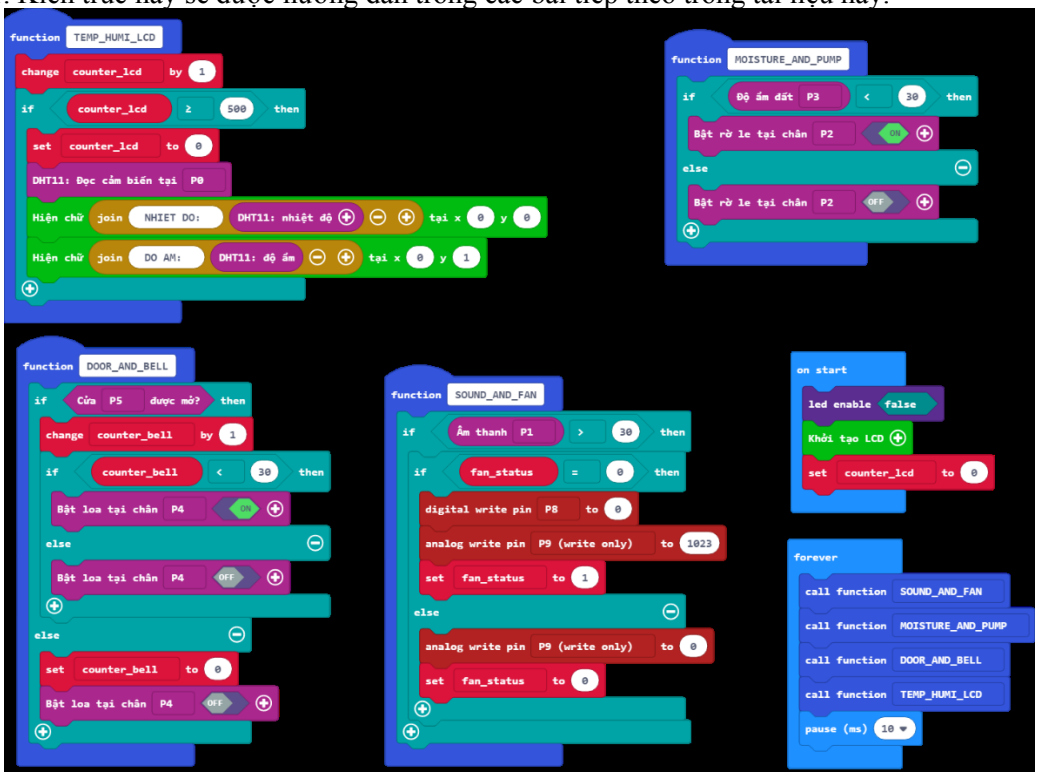
Hình 12: Các câu lệnh hỗ trợ cho LCD, nằm trong mục NPNLCD

Một hình ảnh về hệ thống nhà thông minh, với rất nhiều thiết bị được kết nối với mạch mở rộng MicroBit, được trình bày như hình bên dưới:



Hình13: Mô hình kết nối các thiết bị cho dự án nhà thông minh

Một hình ảnh về chương trình cho hệ thống này, được minh họa ở hình bên dưới. Khi hệ thống có nhiều thiết bị, việc tổ chức chương trình là rất quan trọng. Cụ thể, không có câu lệnh lặp nào được sử dụng, cũng như chỉ có đúng duy nhất 1 câu lệnh pause trong chương trình, nằm ở cuối khối lệnh forever. Kiến trúc này sẽ được hướng dẫn trong các bài tiếp theo trong tài liệu này.



Hình 14: Chương trình cho nhà thông minh: Không có câu lệnh lặp với lệnh đợi

- **MỤC TIÊU**

Học sinh sẽ:

- Nhận biết được các yêu cầu cơ bản của dự án
- Phát triển dự án đèn thông minh cho trụ đèn chủ
- Điều khiển được màu sắc của bóng đèn
- Có thể tạo và gọi một hàm trong MicroBit
- Hiện thực được chương trình mô phỏng đèn giao thông

- **PHÂN BỔ THỜI LƯỢNG**

1 Giới thiệu về dự án đèn giao thông

(05 phút)

2 Hiện thực chương trình

(15 phút)

3 Hiện thực đèn xanh và đèn vàng

(20 phút)

Bài 6: Phát triển dự án Đèn giao thông

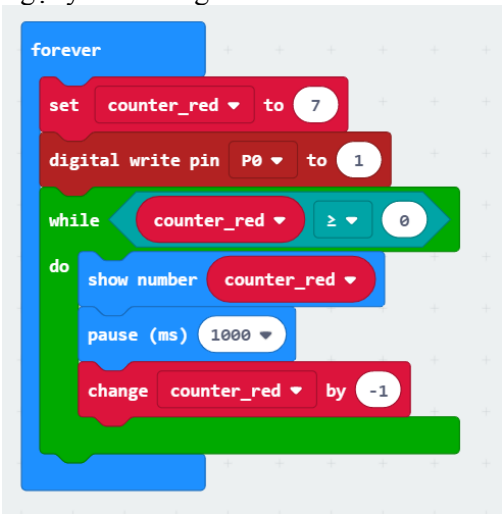
1 Giới thiệu về dự án đèn giao thông

Từ bài này, chúng ta sẽ kết hợp các module sẵn có để làm dự án đèn giao thông. Đèn giao thông sẽ có 1 trụ điều khiển chính và 3 trụ còn lại sẽ nhận tín hiệu điều khiển từ trụ chính để hiển thị các thông tin cần thiết. Việc giao tiếp giữa các trụ đèn sẽ được dựa trên sóng radio được hỗ trợ sẵn trên mạch microbit. Trong bài này, chúng ta sẽ làm chương trình chạy trên mạch trung tâm trước. Các chức năng cơ bản sẽ như sau:

- Hiển thị đếm ngược trên màn hình hiển thị 25 LED của MicroBit
- Điều khiển đèn hiển thị với màu tương ứng
- Thời gian đèn đỏ là 7s, đèn xanh là 5s và đèn vàng là 2s (thời gian đèn đỏ bằng tổng đèn xanh và đèn vàng)

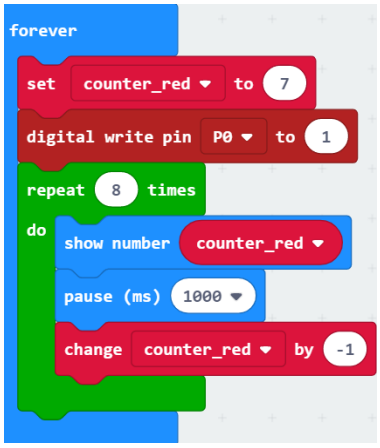
2 Hiện thực chương trình

Đầu tiên, chúng ta hãy hiện thực chức năng cho đèn đỏ trước. Yêu cầu ở đây là module LED Chipi sẽ sáng màu đỏ, còn trên màn hình sẽ đếm lùi từ 7 về 0 và sau đó lặp lại. Giáo viên có thể cho yêu cầu này như 1 bài tập trên lớp với học sinh.
Có rất nhiều cách để hiện thực yêu cầu này, chúng ta có thể khai báo 1 biến, có tên là **counter_red**, đặt giá trị ban đầu của nó là 7. Sau đó dùng 1 vòng lặp while để đến lùi cho đến khi nó bằng 0. Tất nhiên, chúng ta cũng không quên xuất tín hiệu ra chân điều khiển module LED Chipi, trong trường hợp này là chân P0. Một đáp án gợi ý cho chúng ta sẽ như sau:



Hình 31: Chương trình điều khiển đèn màu đỏ

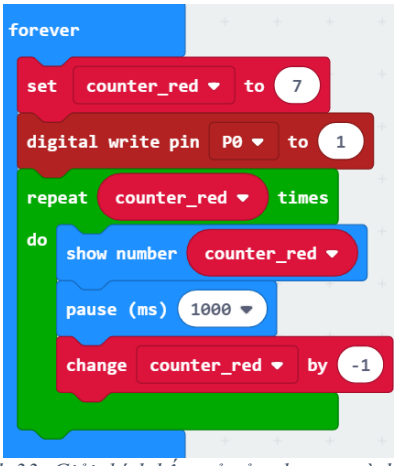
Trong chương trình này, chúng ta không có câu lệnh nào cần thiết trong khối on start, nên khối này được lược bỏ.
Bài tập trên lớp: Học sinh hãy hiện thực lại chương trình theo yêu cầu ở trên, nhưng thay vì dùng câu lệnh while, hãy dùng câu lệnh repeat. Đáp án gợi ý sẽ như sau:



Hình 32: Đáp án gợi ý cho câu lệnh repeat

Lưu ý là chúng ta phải lặp lại 8 lần, để giá trị hiển thị ra màn hình sẽ có số 0.

Bài tập trên lớp: Học sinh hãy hiện thực chương trình sau đây, xem thử kết quả thực thi có đúng với yêu cầu hay không và giải thích tại sao.



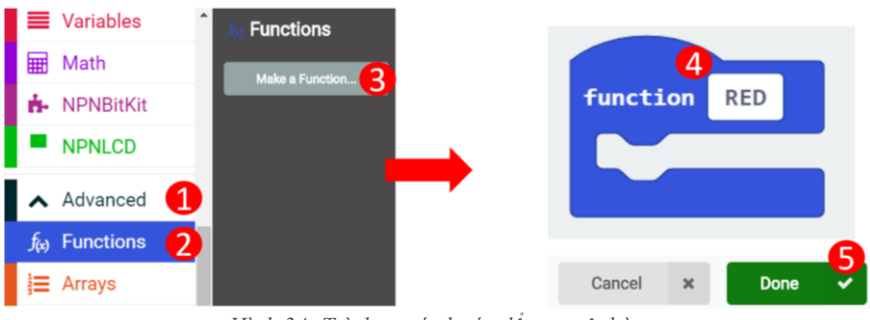
Hình 33: Giải thích kết quả của chương trình này

Thoạt nhìn, chương trình này cũng có vẻ tương tự như chương trình trên. Tuy nhiên, nó chỉ hiện thị ra màn hình kết quả đếm lùi từ 7 đến 4 mà thôi. Lý do là trong câu lệnh repeat, chương trình ngầm tạo ra 1 biến **i**, có giá trị **ban đầu là 0 và sẽ tăng dần sau mỗi lần lặp**. Khi giá trị của i lớn hơn counter_red, vòng lặp sẽ kết thúc. Mỗi vòng lặp I tăng lên 1 đơn vị còn counter_red lại giảm đi 1, nên số lần lặp giảm đi 1 nửa.

Qua ví dụ này, chúng ta có thể thấy rằng về ý nghĩa, repeat và while có thể thay thế cho nhau. Nhưng thực tế, repeat phù hợp với số lần lặp cố định, là 1 con số thay vì là 1 biến số. Còn vòng while, sẽ thích hợp hơn cho các điều kiện lặp, liên quan đến giá trị của biến số.

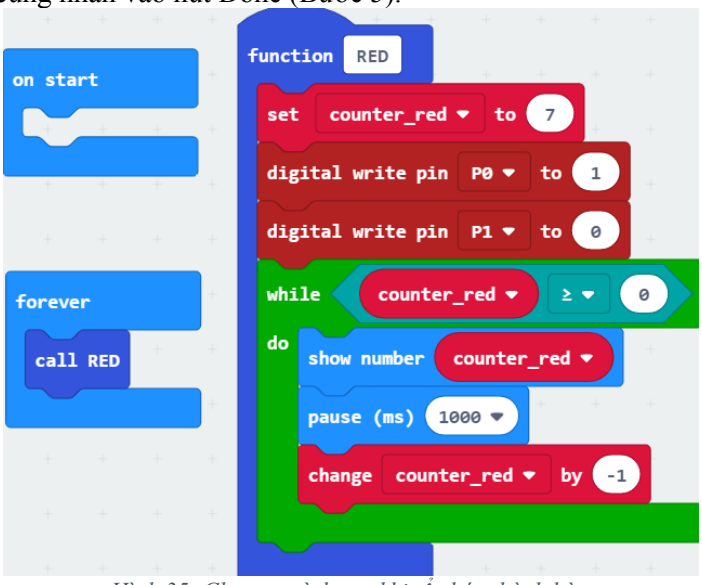
3 Hiện thực đèn xanh và đèn vàng

Việc hiện thực chức năng cho đèn xanh và đèn vàng cũng sẽ tương tự như đèn đỏ. Do vậy, để tránh chương trình trở nên dài dòng, trước khi hiện thực đèn xanh và đèn vàng, chúng ta sẽ tạo ra hàm xử lý cho đèn đỏ, rồi gọi lại trong khối forever. Trình tự 5 bước để tạo một hàm được mô tả như hình bên dưới:



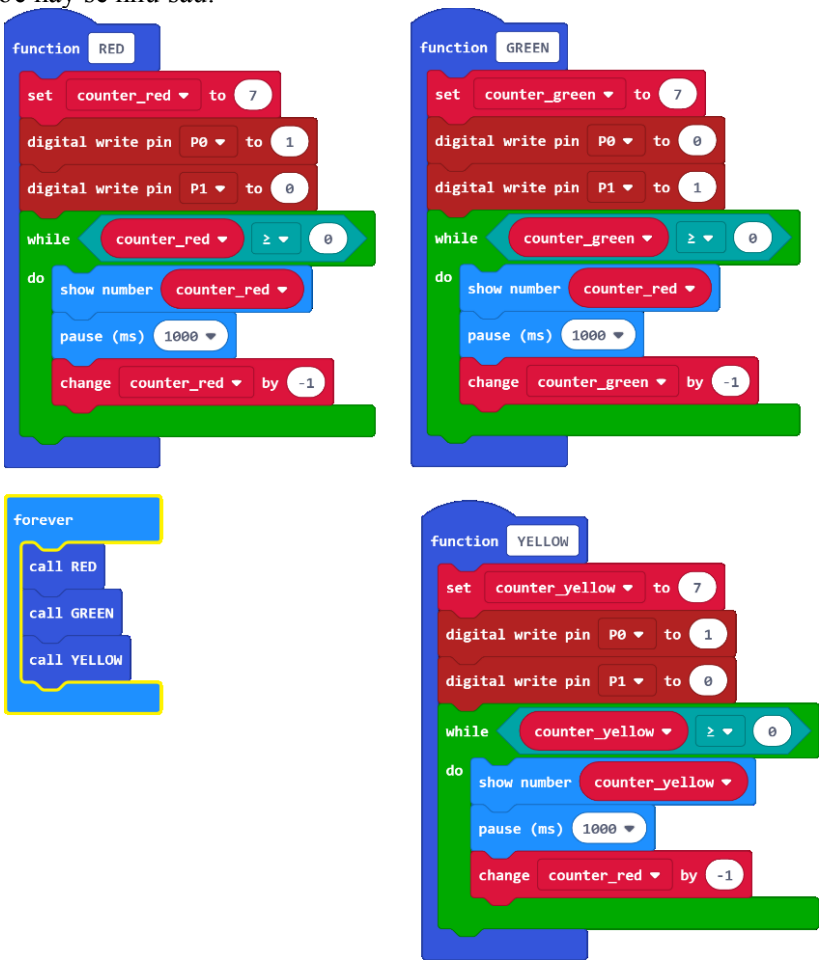
Hình 34: Trình tự các bước để tạo một hàm

Để tạo 1 hàm, chúng ta sẽ vào mục Advanced (Bước 1), chọn Functions (Bước 2), chọn tiếp Make a Function (Bước 3). Một cửa sổ mới sẽ hiện ra để chúng ta đặt tên hàm (Bước 4), trong ví dụ này là RED, và cuối cùng nhấn vào nút Done (Bước 5).



Hình 35: Chương trình sau khi tổ chức thành hàm

Tương tự như vậy, chúng ta sẽ hiện thực đủ chức năng 3 đèn cho hệ thống, bằng cách tạo ra thêm các biến dành cho đèn xanh (counter_green) và đèn vàng (counter_yellow). Thực ra, chúng ta có thể dùng lại biến counter_red, nhưng như vậy sẽ làm cho chương trình không dễ đọc. Chương trình của chúng ta đến bước này sẽ như sau:



Hình 36: Chương trình đèn giao thông đơn giản cho mạch chủ

Bài 7: Tái cấu trúc chương trình đèn giao thông

- **MỤC TIÊU**

Học sinh sẽ:

- Nắm được nhược điểm của kiến trúc cũ
- Tái cấu trúc được chương trình mới tối ưu hơn
- Hiểu được luồng thực thi của chương trình
- Tích hợp được khả năng gửi dữ liệu không dây giữa các trụ đèn

- **PHÂN BỔ THỜI LƯỢNG**

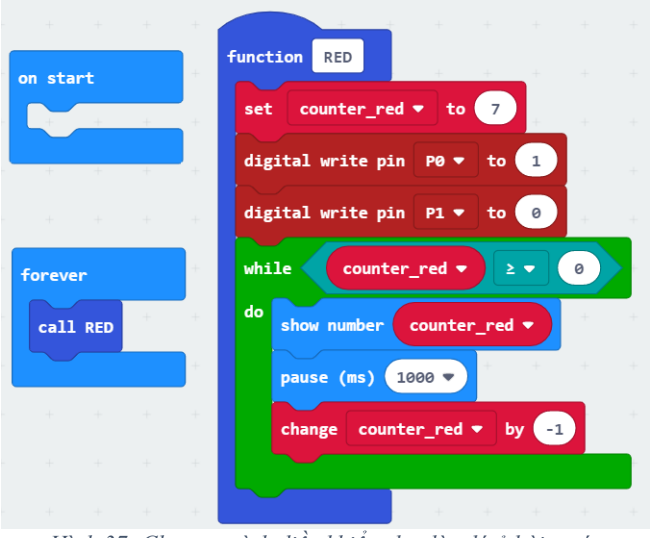
1 Hạn chế của chương trình hiện tại (15 phút)

2 Hiện thực đèn xanh và đèn đỏ (15 phút)

3 Gửi tín hiệu cho các trụ đèn khác (15 phút)

1 Hạn chế của chương trình hiện tại

Đầu tiên, chúng ta hãy xem lại đoạn chương trình khi hiện thực chỉ đèn đỏ mà thôi, chương trình của chúng ta như sau:



Hình 37: Chương trình điều khiển cho đèn đỏ ở bài trước

Trong chương trình này, chúng ta sử dụng câu lệnh lặp while để hiển thị chức năng đếm lùi ra màn hình. Toàn bộ các câu lệnh trong vòng lặp này được thực hiện trong khoảng 7 giây (mỗi lần lặp mất 1 giây do chúng ta có câu lệnh đợi 1 giây).
Như vậy, khi đèn đỏ đang đếm lùi, chúng ta sẽ không thể làm thêm tác vụ nào khác với mạch MicroBit. Giả sử như chúng ta muốn kiểm tra xem người đi bộ, có nhấn nút để xin đi qua đường hay không, thì trong chương trình này, điều đó là gần như không khả thi. Chỉ có khi chúng ta nhấn đè nút nhấn trên 7 giây, thì chương trình mới có thể nhận thấy được tín hiệu đó, sau khi nó đã kết thúc vòng lặp while.

Hiện tượng này trong chương trình được gọi là block (có nghĩa là mắc kẹt), và nó thực sự là điều không tốt trong việc lập trình, đặc biệt đối với các hệ thống mạch điện tương tác trực tiếp với người dùng. Để khắc phục hiện tượng đó, chúng ta cần phải tuân thủ những quy ước như sau:

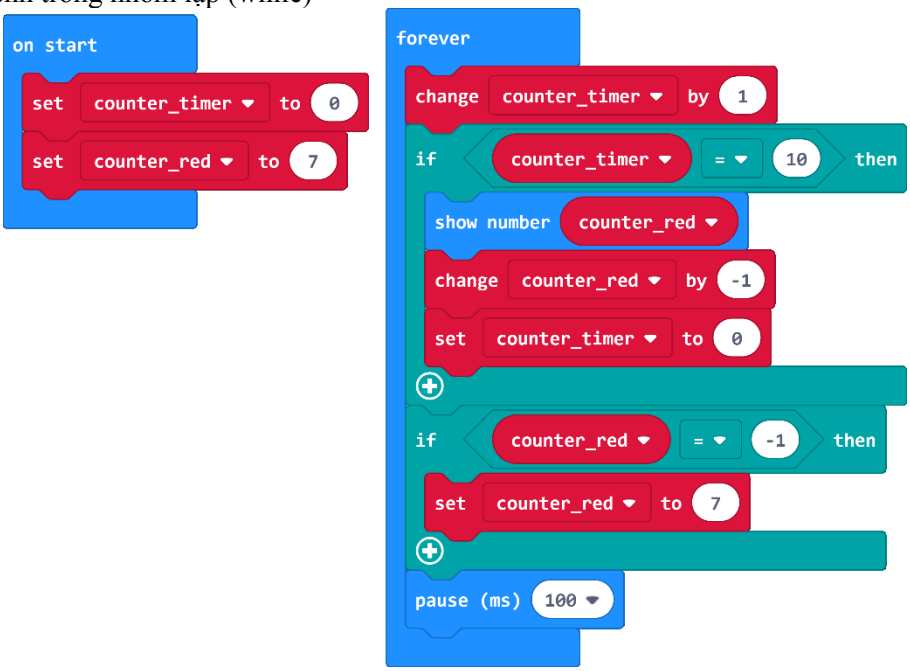
- Không được sử dụng bất kì câu lệnh lặp nào nữa, chủ yếu là các câu lệnh while và repeat
- Không được sử dụng bất kì câu lệnh pause nào trong chương trình, ngoại trừ duy nhất 1 câu lệnh pause(100) ở cuối vòng forever

Với yêu cầu như trên, chương trình của chúng ta sẽ có cấu trúc ban đầu như sau:

Hình 38: Cấu trúc chương trình mới

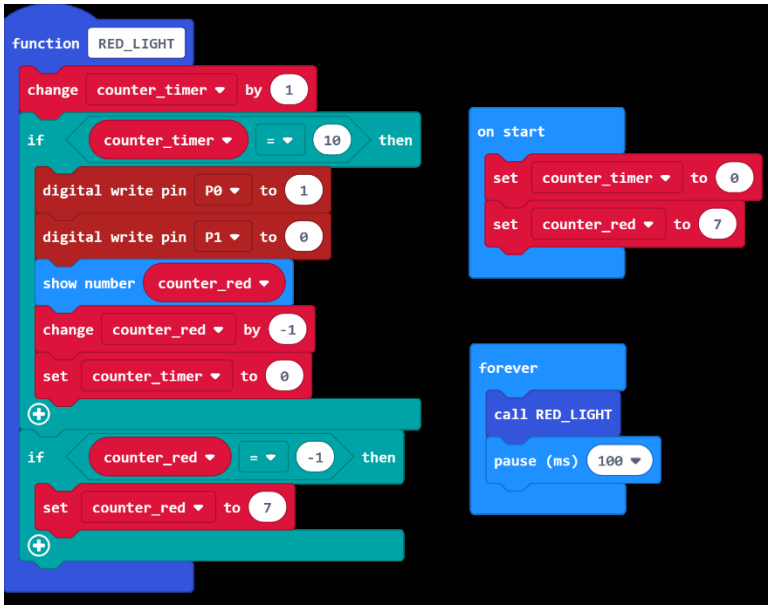
Bài tập trên lớp: Hãy hiện thực lại chức năng của đèn đỏ, nhưng không được sử dụng câu lệnh while và câu lệnh pause.
Gợi ý: Chúng ta sẽ tạo thêm 1 biến counter để đếm, khi biến này có giá trị là 10, tức là đã qua 10 lần của 100ms, nghĩa là 1 giây.

Dưới đây là một chương trình để điều khiển đèn màu đỏ nhưng không sử dụng câu lệnh đợi (pause) và lệnh trong nhóm lặp (while)



Hình 39: Chương trình cho đèn màu đỏ

Như vậy, để có thể tạo một hiệu ứng thời gian trong chương trình, chúng ta phải khai báo thêm biến và khéo léo kiểm tra điều kiện của nó. Ví dụ, trong câu lệnh if đầu tiên, chúng ta xét điều kiện **counter_timer = 10**, để thiết lập 1 hiệu ứng đợi trong 1 giây. Ở câu lệnh if thứ 2, đó là khi hết chu kì của đèn đỏ. Do chúng ta muốn giá trị 0 sẽ hiện ra trong 1s, nên điều kiện ở đây là **counter_red = -1**. Cuối cùng, chúng ta sẽ đóng gói chương trình điều khiển đèn đỏ trong 1 hàm, đặt tên là hàm là RED_LIGHT, như hình bên dưới:

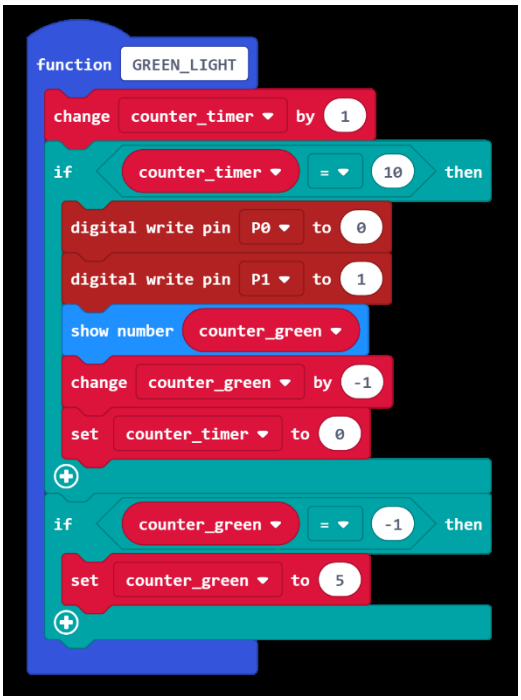


Hình 40: Chương trình cho đèn đỏ được rút gọn trong hàm RED_LIGHT

Lưu ý: Thông thường khi điều kiện if là đúng, chúng ta sẽ thực hiện 1 số tác vụ và gán lại giá trị cho các biến đếm, để bắt đầu lại 1 chu kì mới.

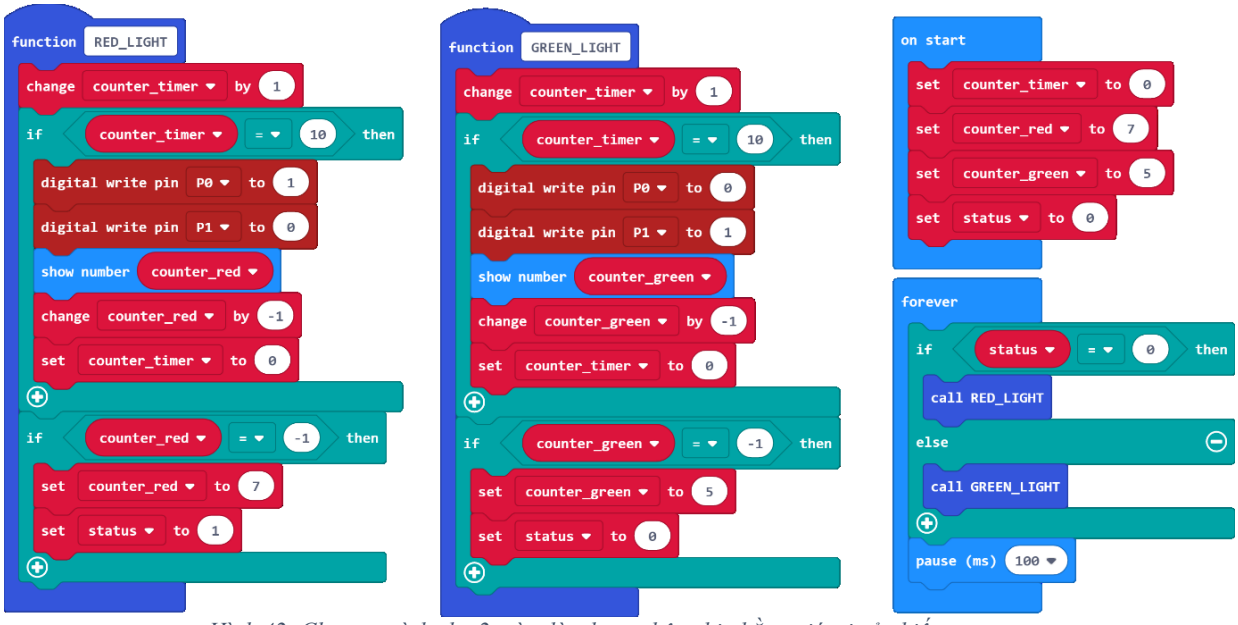
2 Hiện thực đèn xanh và đèn đỏ

Chúng ta sẽ làm tương tự với trường hợp đèn xanh, cũng tạo ra 1 biến số counter_green và hiện thực hàm GREEN_LIGHT như sau:



Hình 41: Chương trình cho đèn xanh

Tuy nhiên, chúng ta sẽ không thể đơn giản gọi chương trình điều khiển đèn màu xanh trong khối forever được. Vì các chương trình con này không có khả năng đợi nhau như câu lệnh while trong bài học trước. Nên để chương trình màu đỏ chạy trước, rồi mới tới chương trình đèn màu xanh, chúng ta sẽ phải tạo thêm 1 biến số nữa, gọi là status. Biến này bằng 0 là chương trình màu đó sẽ được chạy, còn khi bằng 1, thì chương trình màu xanh sẽ được phép chạy. Chương trình của chúng ta sẽ như sau:

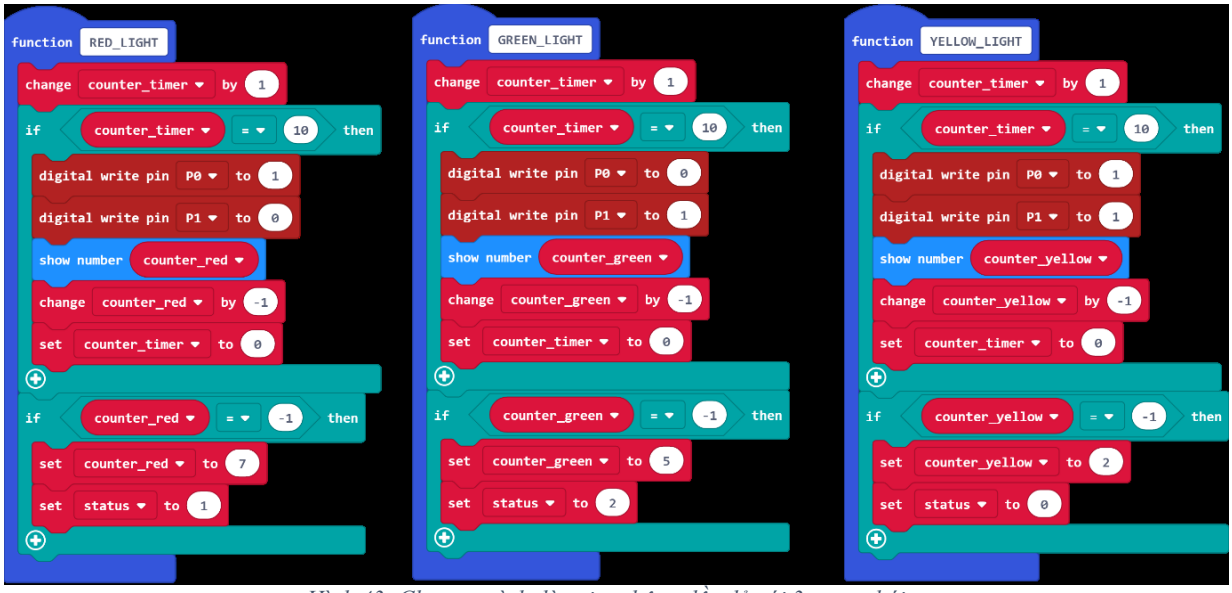


Hình 42: Chương trình cho 2 màu đèn được phân chia bằng giá trị của biến status

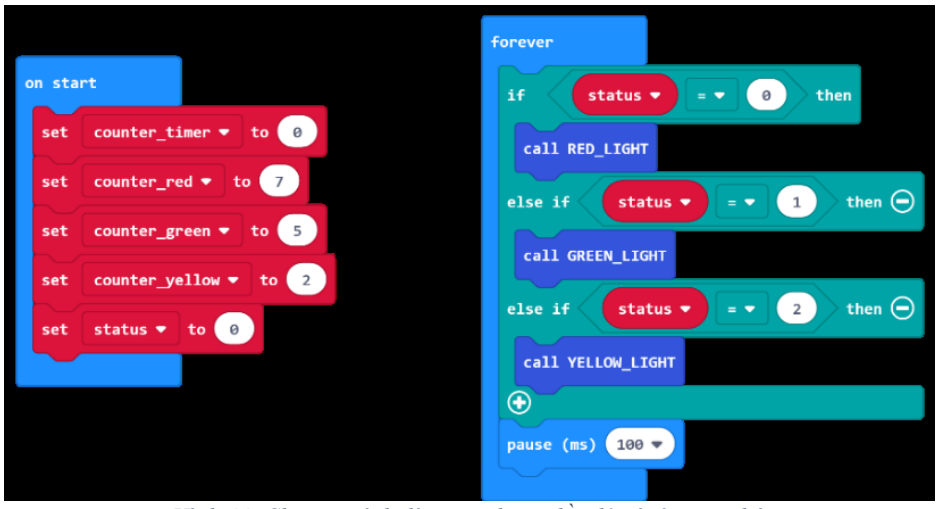
Hãy lưu ý việc gán giá trị của biến **status** ở phía cuối mỗi hàm RED_LIGHT và GREEN_LIGHT. Chương trình sẽ chuyển sang trạng thái mới khi trạng thái cũ đã thực hiện xong chức năng của nó.

Bài tập trên lớp: Hãy thử thay đổi giá trị của status trong khối on start bằng 1 và giải thích kết quả của chương trình.

Đáp án: Đèn màu xanh sẽ chạy trước, rồi mới tới đèn đỏ và sau đó chương trình sẽ được lặp lại. Với ví dụ này, chúng ta có thể nói status là điểm bắt đầu của 1 trạng thái. Tùy vào trạng thái nào mà chương trình tương ứng sẽ được chạy. Việc chuyển sang trạng thái khác, tùy thuộc vào từng ứng dụng. Hoàn toàn tương tự, chúng ta sẽ hiện thực đầy đủ chương trình với đèn màu vàng, như sau:



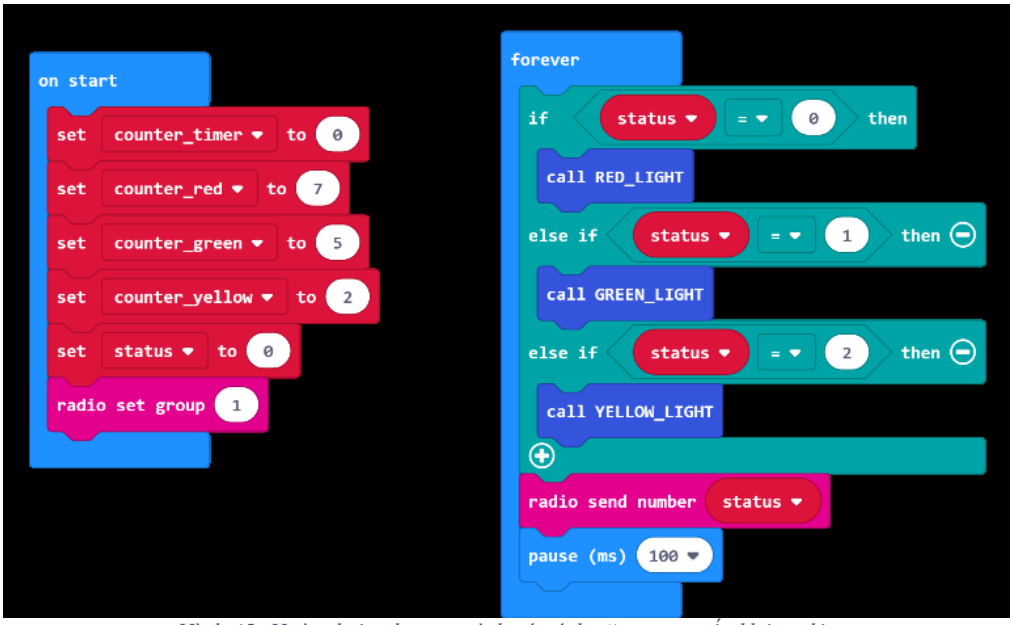
Hình 43: Chương trình đèn giao thông đầy đủ với 3 trạng thái



Hình 44: Chương trình đèn giao thông đầy đủ với 3 trạng thái

3 Gửi tín hiệu cho các trụ đèn khác

Với tính năng này, chúng ta chỉ đơn giản là gửi đi trạng thái status của hệ thống trong hàm **forever**. Các trụ đèn khi nhận được giá trị này, sẽ điều chỉnh màu đèn tương ứng. Đây là lợi thế rất lớn của việc thiết kế chương trình theo trạng thái. Việc quản lý chương trình và giao tiếp sẽ thuận lợi hơn rất nhiều. Chúng ta cũng cần lưu ý là chỉnh nhóm trong khối on start. Chương trình hiệu chỉ ở khối on start và forever sẽ như sau:



Hình 45: Hoàn thiện chương trình với tính năng giao tiếp không dây

- **MỤC TIÊU**

Học sinh sẽ:

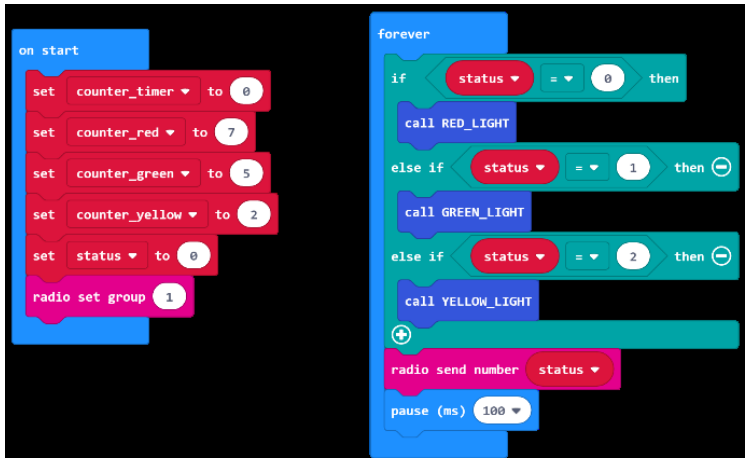
- Nắm được nhược điểm của kiến trúc cũ
- Tái cấu trúc được chương trình mới tối ưu hơn
- Hiểu được luồng thực thi của chương trình
- Tích hợp được khả năng gửi dữ liệu không dây giữa các trụ đèn

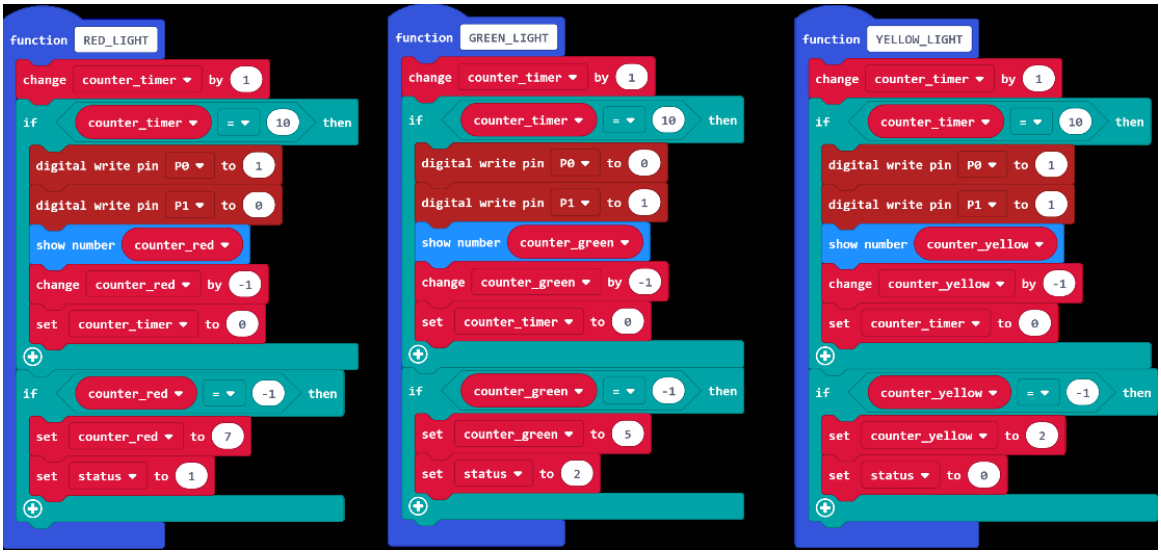
- **PHÂN BỔ THỜI LƯỢNG**

Bài 8: Hiện thực trụ đèn phụ

1 Sơ lược về chương trình mạch chủ

Đầu tiên, chúng ta hãy xem lại các câu lệnh quan trọng của chương trình ở mạch chủ. Chương trình của mạch chủ hiện tại như sau:





Hình 46: Chương trình trụ đèn chủ

Hãy lưu ý rằng, chương trình của chúng ta đang chạy dựa vào trạng thái, hay nói cách khác là giá trị của biến **status**. Các câu lệnh gán giá trị cho biến này nằm ở các khối lệnh quan trọng sau đây:

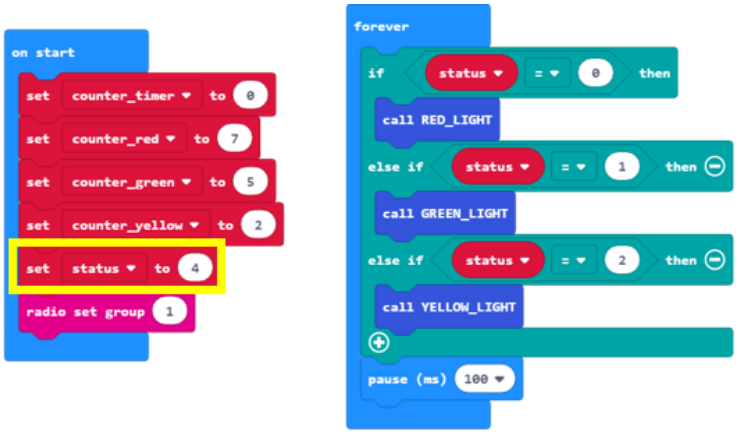
- **on start**: Gán giá trị khởi động cho biết status, ở đây là 0, tức là chạy đèn đỏ đầu tiên. Nếu chúng ta gán là 1, chương trình sẽ chạy đèn xanh đầu tiên, bằng 2 sẽ chạy đèn vàng. Vì lý do nào đó, nếu chúng ta gán cho nó giá trị khác, chẳng hạn là 3 thì chương trình sẽ không chạy gì cả.
- **Hàm RED_LIGHT**: khi chạy xong chu kì của đèn đó, biến này được gán lại giá trị là 1, để chuyển sang chu kì của đèn xanh.
- **Hàm GREEN_LIGHT**: Tương tự như đèn đó, sau khi chạy xong chu kì của đèn xanh, nó được gán bằng 2, để bắt đầu 1 chu kì mới của đèn vàng.
- **Hàm YELLOW_LIGHT**: Cuối cùng, khi xong đèn vàng, nó được gán trở lại bằng 0, để quay lại đèn đỏ.

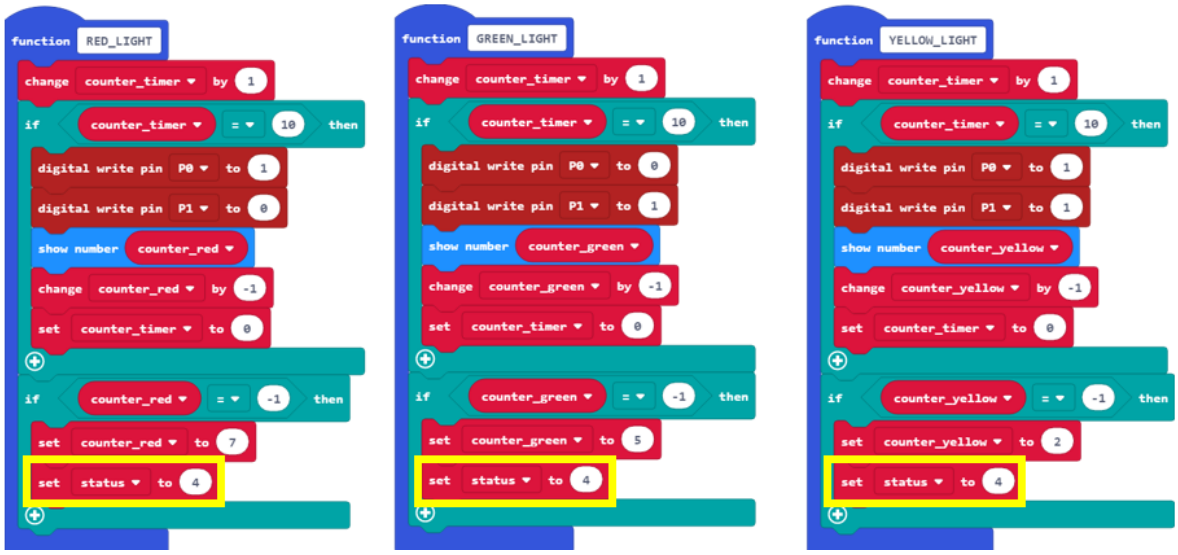
Bây giờ, chúng ta sẽ hiện thực một trụ đèn phụ, đặt đối diện trụ đèn chính trên một ngã tư. Tức là, trụ đèn này cần hiển thị trạng thái giống hệt trụ đèn chủ.

2 Hiện thực trụ đèn phụ thứ nhất

Trước tiên, đây là trụ đèn phụ, nên khi mới bật nguồn lên, nó không cần phải sáng 1 đèn nào cả. Chúng ta có thể sửa lại câu lệnh trong hàm on start, để đặt giá trị ban đầu của nó là 4, với quy ước rằng trụ đèn không cần phải làm gì khi trạng thái là 4.

Tiếp theo, khi đèn phụ đã sáng hết thời gian cho đèn màu đỏ, nó cũng không cần phải chuyển trạng thái sang đèn xanh, do nó phải đồng bộ hiển thị và cả thời gian với trụ đèn chủ. Chúng ta cũng sửa lại câu lệnh trong 3 làm RED_LIGHT, GREEN_LIGHT và YELLOW_LIGHT lại. Chương trình của chúng ta sẽ như sau:



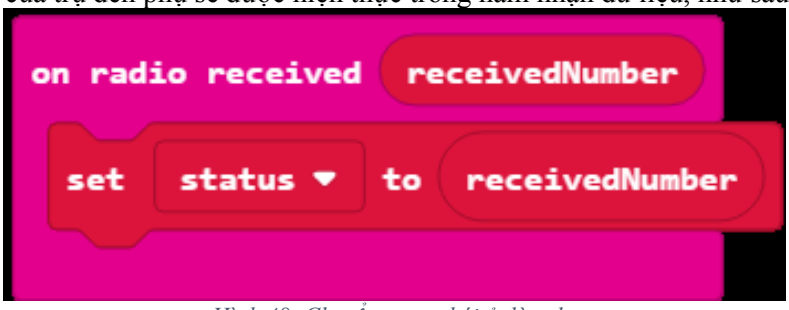


Hình 47: Thay đổi các câu lệnh chuyển trạng thái cho đèn phụ thứ nhất

Lưu ý: Đèn phụ nhận lệnh từ đèn chủ, nên nó phải có cùng nhóm Radio với đèn chính. Thêm nữa, câu lệnh gửi dữ liệu không dây trong khối forever ở đèn phụ cần phải bỏ đi, vì nó chỉ hoạt động ở chế độ nhận dữ liệu mà thôi.

3 Chuyển trạng thái ở trụ đèn phụ

Với chương trình như trên, khi nạp vào đèn phụ sẽ không có hiện tượng gì cả, do nó được gán trạng thái ban đầu là 4, và chúng ta không gọi thực thi 1 hàm nào với giá trị trạng thái nào là 4 cả. Việc chuyển trạng thái của trụ đèn phụ sẽ được hiện thực trong hàm nhận dữ liệu, như sau:



Hình 48: Chuyển trạng thái ở đèn phụ

Với khối lệnh thêm này, hãy thử bật đèn phụ trước. Chúng ta sẽ thấy đèn phụ sẽ không hiện thị gì cả. Cho đến khi chúng ta bật đèn chủ lên, đèn phụ nhận được trạng thái từ đèn chính và sẽ chạy theo cùng trạng thái với đèn chính.

4 Xử lý lỗi mất đồng bộ giữa 2 trụ đèn

Lỗi mất đồng bộ này sẽ xảy ra khi việc gửi dữ liệu không dây từ đèn chủ qua đèn phụ không thành công. Và như vậy, phải 100ms sau đèn phụ mới nhận được dữ liệu. Mặc dù 100ms là rất nhỏ, tuy nhiên việc hiển thị ra 25 đèn mất tới nửa giây, nên sự mất đồng bộ này sẽ rất dễ nhận ra. Do đó, chúng ta cần phải đồng bộ lại toàn bộ hệ thống, khi nó chuyển sang trạng thái mới, tức là giá trị receiveNumber khác với giá trị status. Chúng ta sẽ thực thi lại các câu lệnh như trong khối lệnh on start. Lúc này, chương trình trong khối nhận dữ liệu không dây sẽ như sau:

Hình 49: Chương trình hoàn chỉnh, với các câu lệnh mới để đồng bộ

Hãy lưu ý các câu lệnh trong khối lệnh nhận dữ liệu không dây. Chúng ta sẽ kiểm tra giá trị vừa nhận (chứa trong biến receivedNumber), liệu nó có thay đổi so với trạng thái hiện tại hay không (lưu trong biến status). Nếu hai giá trị này khác nhau, chúng ta sẽ đồng bộ lại toàn bộ trạng thái của hệ thống bằng cách đặt lại giá trị của các biến như ban đầu. Cuối cùng, chúng ta gán giá trị nhận được cho biến trạng thái.

Đến bước này, chúng ta có thể bật nguồn hay mạch điện để xem việc thực thi chương trình 2 bên. Với những cải tiến như trên, chương trình 2 bên thực sự chạy rất giống nhau về mặt hiển thị.

5 Bài tập về nhà

Học sinh hãy hiện thực chức năng cho trụ đèn đối diện: Trạng thái của đèn sẽ ngược lại với 2 trụ đèn này: Khi trụ đèn chủ đỏ, thì đèn phụ sẽ xanh, rồi tự chuyển sang vàng, khi trụ đèn chủ vàng, thì đèn phụ vẫn không đổi. Khi đèn chủ xanh, thì đèn phụ đỏ.

Bài 9: Hiện thực trụ đèn phụ thứ 2

- **MỤC TIÊU**

Học sinh sẽ:

- Nắm được nhược điểm của kiến trúc cũ
- Tái cấu trúc được chương trình mới tối ưu hơn
- Hiểu được luồng thực thi của chương trình
- Tích hợp được khả năng gửi dữ liệu không dây giữa các trụ đèn

- **PHÂN BỔ THỜI LƯỢNG**

1 Sơ lược về chương trình mạch phụ bài trước (15 phút)

2 Hiện thực trụ đèn phụ thứ hai (15 phút)

3 Chuyển trạng thái từ xanh sang vàng (15 phút)

1 Sơ lược về chương trình mạch phụ bài trước

Ở trụ phụ thứ nhất, vấn đề đơn giản hơn, do nó hiển thị giống với trụ chủ. Còn ở trụ phụ thứ 2 và thứ 3, nó sẽ hiển thị gần như ngược lại với trụ chính. Chúng ta sẽ thống kê lại chức năng của trụ này như sau:

- Khi trạng thái là 0 (Trụ chính đang đỏ): Trụ phụ sẽ hiển thị đèn xanh
- Khi trạng thái là 1 (Trụ chính đang xanh): Trụ phụ sẽ hiển thị đèn đỏ
- Khi trạng thái là 2 (trụ chính đang vàng): Trụ phụ không làm gì cả, lúc này trụ phụ đang trong chu kì của đèn màu đỏ, nó vẫn tiếp tục hiển thị màu đỏ.
- Trụ phụ khi hoạt động hết chu kì xanh, sẽ tự động chuyển sang vàng mà không chờ lệnh từ trụ chính

Với 4 chức năng kể trên, 3 chức năng đầu chúng ta sẽ hiện thực trong khối lệnh forever, còn chức năng cuối cùng, chúng ta sẽ thay đổi câu lệnh chuyển trạng thái ở hàm GREEN_LIGHT.

Dưới đây là chương trình của trụ phụ, đặt đối diện với trụ chính:

Hình 50: Chương trình trụ đèn chủ

2 Hiện thực trụ đèn phụ thứ hai

Như đã mô tả ở trên, chúng ta cần gọi hàm cho đúng với các giá trị trạng thái trong khối lệnh forever. Chương trình cho trụ đèn phụ này sẽ như sau:

Hình 51: Chương trình xử lý trạng thái cho trụ đèn phụ

Lưu ý rằng, chúng ta sẽ không xử lý gì cho trường hợp nhận giá trị là 2, ví lúc này chúng ta đang hiển thị màu đỏ. Tuy nhiên, trụ đèn phụ này cũng phải hiển thị màu vàng. Do đó, chúng ta sẽ định nghĩa thêm 1 trạng thái mới cho nó, trạng thái số 3 để hiển thị màu vàng. Khi hiển thị xong màu xanh, hệ thống sẽ tự động chuyển sang trạng thái 3, để hiển thị màu vàng.

3 Chuyển trạng thái từ xanh sang vàng

Như đã mô tả ở trên, khác với trụ phụ thứ nhất, sau khi kết thúc màu xanh nó sẽ đợi tín hiệu từ trụ chủ, trụ thứ 2 sẽ tự động chuyển sang màu vàng. Do đó, ở cuối hàm GREEN_LIGHT, chúng ta sẽ có 1 câu lệnh để thay đổi status thành 3, là giá trị để hệ thống chuyển sang vàng.
Hình ảnh toàn bộ chương trình cho trụ đèn phụ này sẽ như sau:

Hình 52: Tự chuyển sang đèn vàng sau khi hết đèn xanh

Lợi thế của việc tổ chức chương trình theo kiến trúc này là việc thay đổi rất nhanh: chỉ cần thay đổi trạng thái của biến status ở vị trí thích hợp.

Hình 53: Toàn bộ chương trình cho trụ đèn phụ 2

- **MỤC TIÊU**

Học sinh sẽ:

- Hiểu được luồng thực thi của chương trình
- Kết nối được phần cứng với với dự án hiện có
- Tích hợp được thêm chức năng mới
- Hiểu được khái niệm "Lập trình độc lập"

- **PHÂN BỔ THỜI LƯỢNG**

1 Nguyên lý cơ bản (20 phút)

2 Hiện thực tính năng xin qua đường (25 phút)

Bài 10: Tích hợp thêm dịch vụ cho dự án Đèn Giao Thông

(Bài tham khảo)

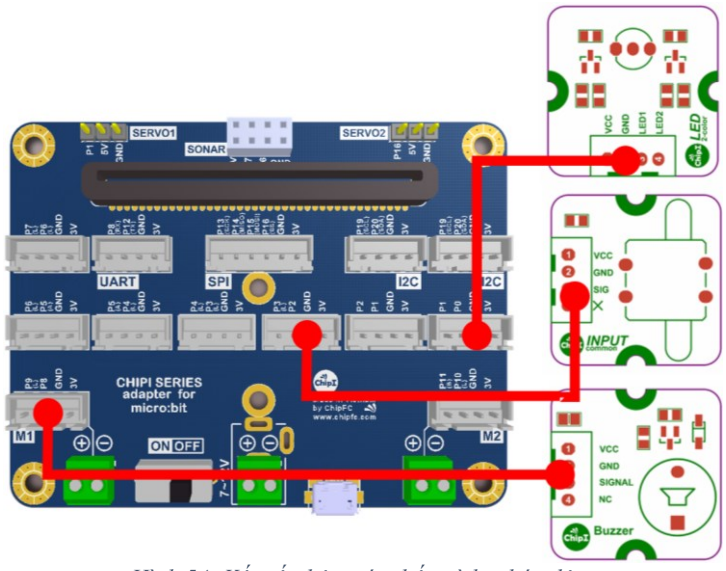
1 Nguyên lý cơ bản

Với những hướng dẫn ở bài trước, chúng ta cần phải tuân thủ 2 nguyên tắc lập trình trên phần cứng, như sau:

- Không sử dụng các câu lệnh lặp, sử dụng thêm biến để tạo hiện ứng đợi
- Không sử dụng câu lệnh pause, trừ câu lệnh duy nhất trong khối lệnh forever

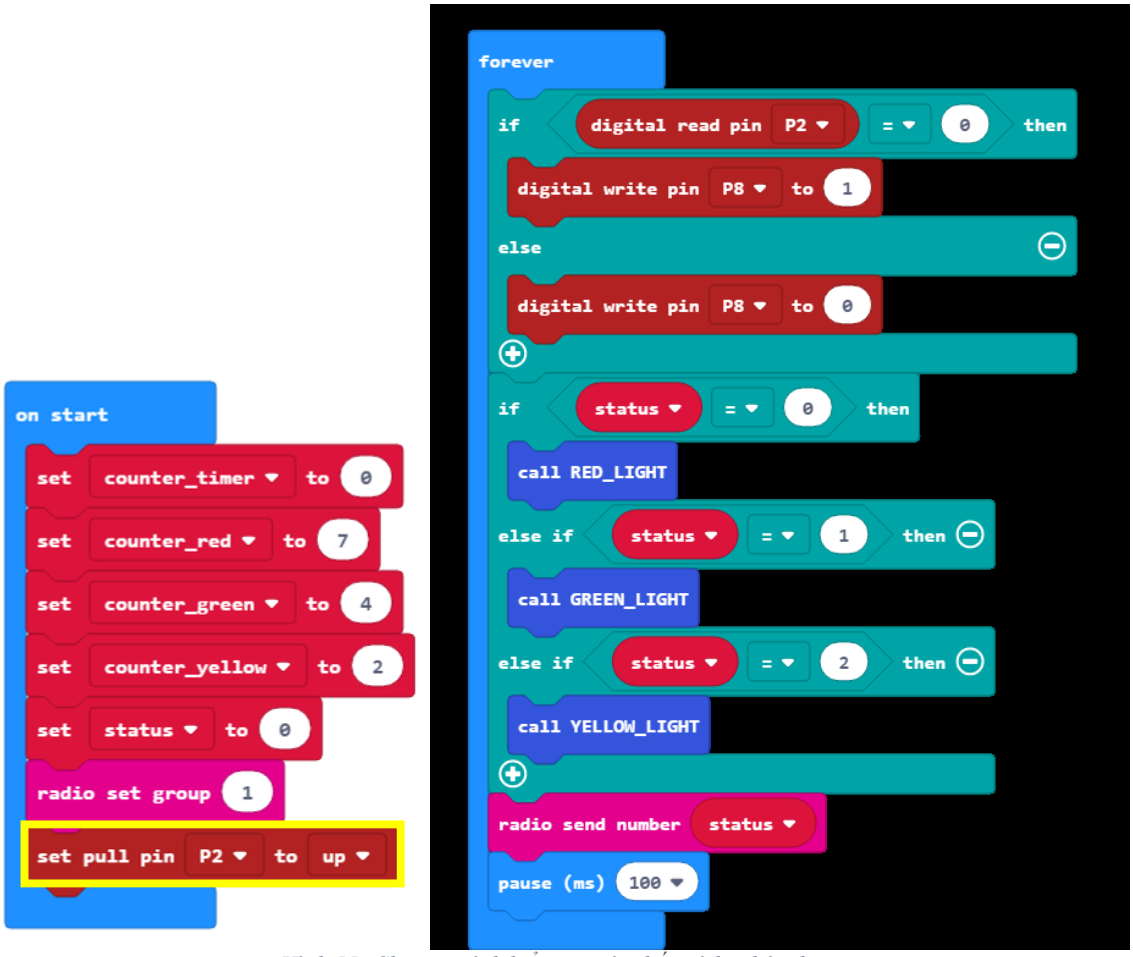
Sau khi tuân thủ 2 nguyên tắc này, chúng ta có thể dễ dàng tích hợp tính năng mới, hoàn toàn không ảnh hưởng đến các tính năng đã hiện thực. Chúng ta chỉ cần hiện thực thêm các câu lệnh trong vòng forever là được.

Chúng ta có thể thử kết nối nút nhấn vào chân P2 và loa báo động vào chân P8, như hình bên dưới. Hãy lưu ý vị trí kết nối của các thiết bị, do mô đun LED đã sử dụng hết 2 chân P0 và P1, nên các phần cứng mới, chúng ta phải nối vào các chân phần cứng khác, không đụng độ với phần cứng đã xài.



Hình 54: Kết nối thêm nút nhấn và loa báo động

Chúng ta sẽ hiện thực một chương trình nhỏ để kiểm tra việc kết nối 2 thiết bị mới: Khi nhấn nút loa sẽ kêu, khi không nhấn thì loa không kêu. **Lưu ý rằng, để một chân có thể nhận tín hiệu, chúng ta phải cấu hình nó là pull up trong on start**. Chúng ta sẽ hiện thực chương trình này trong khối forever, như sau:



Hình 55: Chương trình kiểm tra nút nhấn và loa báo động

Với chương trình ở trên, đôi khi nhấn nút chúng ta sẽ thấy loa không kêu và đôi khi đã thả nút nhấn, loa vẫn kêu. Lý do là mặc dù chúng ta không còn sử dụng câu lệnh lặp nào, việc xuất dữ liệu ra màn hình 25 LED bị trễ khoảng nửa giây. Chúng ta có thể thấy, bất kì câu lệnh nào phải đợi sẽ ảnh hưởng rất nhiều đến những hệ thống có tương tác từ bên ngoài. Độ trễ của chương trình sẽ làm giảm rất nhiều hiệu suất của hệ thống.

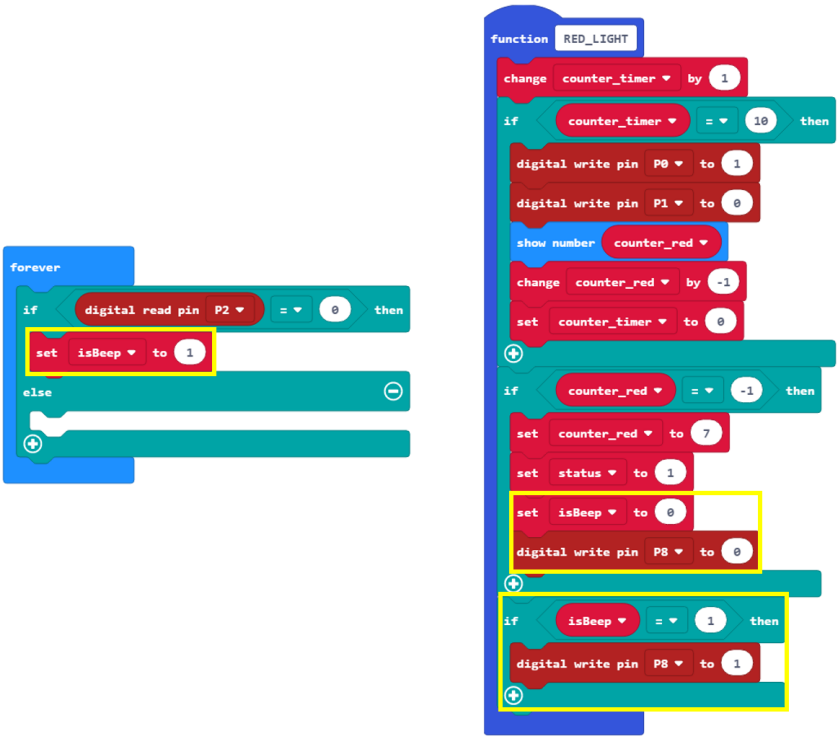
Để khắc phục điều này, một cách nhanh nhất, chúng ta sẽ tạo thêm 1 luồng xử lý song song, bằng cách sử dụng kết hợp khối lệnh run in background và vòng lặp vô tận, như hình bên dưới:

Hình 56: Hiệu chỉnh chương trình: Thêm một luồng thực thi song song

2 Hiện thực tính năng xin qua đường

Sau khi đã kiểm tra dữ liệu đầu vào và đầu ra, chúng ta có thể hiện thực một dịch vụ mới, dịch vụ xin qua đường của người đi bộ với chức năng như sau: Khi người dùng nhấn nút, đến khi đèn đỏ, loa bắt đầu báo hiệu, khi hết đèn đỏ, loa sẽ dừng kêu.

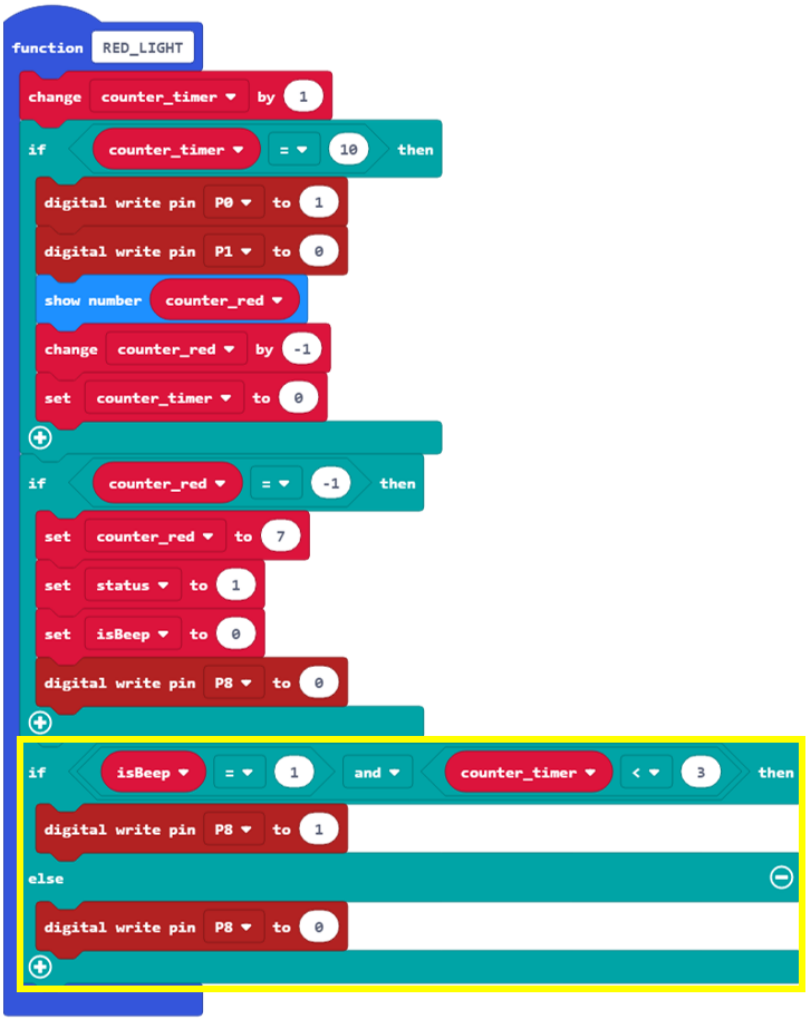
Hướng dẫn: Tạo một biến số, tên là isBeep, có giá trị ban đầu là 0. Khi nhấn nút nhấn thì gán nó lên 1. Điều chỉnh lại hàm RED_LIGHT để thêm tính năng xuất dữ liệu ra loa.



Hình 57: Chương trình tích hợp thêm loa báo động cho người đi bộ

Chúng ta sẽ bắt đầu báo hiệu bằng nhóm lệnh cuối trong khối RED_LIGHT. Khi hết chu kì của đèn đỏ, chúng ta gán lại giá trị isBeep về 0 và tắt chân tín hiệu P8. Hãy lưu ý là biến isBeep được gán bằng 1 khi nhấn nút, trong khối lệnh forever.

Việc làm cho ứng dụng trở nên thân thiện hơn đòi hỏi sự phân tích và hiện thực, chủ yếu nằm ở đoạn chương trình cuối hàm RED_LIGHT. Ví dụ, chúng ta chỉ xuất ra tiếng beep mỗi 300ms, và lặp lại điều này sau mỗi giây, thay vì cứ kêu liên tục suốt thời gian của đèn đỏ. Chúng ta có thể tận dụng biến counter_timer để hiện thực chức năng này. Chương trình với yêu cầu này sẽ như sau:



Hình 58: Loa kêu mỗi giây một lần, mỗi lần 300ms

Rất nhiều tính năng mới có thể được hiện thực, với độ phức tạp tăng dần:

- Khi đang đèn đỏ, mà người dùng nhấn nút xin qua đường, thì loa sẽ không kêu.
- Khi loa đang kêu mỗi giây 01 lần, còn 02 giây nữa hết đèn đỏ, thì kêu với tốc độ nhanh hơn hoặc kêu liên tục không ngừng.

Giáo viên có thể chủ động đặt thêm các yêu cầu khác và xem như là bài tập mở rộng cho học sinh.

☺ Hết ☺
Chúc khỏe và thành công