Lời nói đầu

Đối với những người hay phải thuyết trình giảng viên các trường đại học, trung học chuyên nghiệp,... việc có một thiết bị hỗ trợ thuyết trình là một điều hết sức tiện lợi. Trên thị trường hiện nay có bán nhiều sản phẩm hỗ trợ việc trình chiếu slide. Ngoài việc có mẫu mã đẹp, thuận tiện trong việc sử dụng (giao tiếp USB), các sản phẩm này có một số nhược điểm như: giá khá đắt (tầm 50-60$, do sử dụng Bluetooth) và chỉ hỗ trợ trong trình chiếu slide (forward và backward slide)…

Vậy có giải pháp nào hợp túi tiền hơn mà lại hỗ trợ nhiều chức năng hơn cho người sử dụng? Người sử dụng không những có thể sử dụng trong trình chiếu slide, mà còn có thể dùng để tắt máy tính, bật một chương trình mà mình yêu thích nào đó? ...

Sau một thời gian tìm hiểu về tín hiệu hồng ngoại và giao tiếp USB, nhóm đã phát triển được một sản phẩm (bao gồm cả phần cứng+ phần mềm) thỏa mãn được yêu cầu trên. Điểm đặc biệt ở sản phẩm là: nó có thể hoạt động tốt với bất kì thiết bị điều khiển sử dụng hồng ngoại nào.

Nhóm xin cảm ơn thầy Bùi Quốc Anh, Bộ môn Kĩ thuật máy tính, Khoa CNTT, trường ĐH Bách Khoa Hà Nội, đã cho nhóm những lời chỉ bảo quí báu trong quá trình hoàn thành sản phẩm.

1 Các vấn đề liên quan tới hồng ngoại

Cách rẻ nhất để điều khiển từ xa một thiết bị (trong một khoảng cách cho trước) là sử dụng sóng hồng ngoại. Ngày nay, ta có thể thấy hàng loạt thiết bị sử dụng điều khiển từ xa dùng hồng ngoại trong các TV, đầu DVD.....

1.1 Hồng ngoại

Chúng ta không thể thấy hồng ngoại, vì bước sóng của nó là 950nm nằm ngoài khoảng nhìn thấy của mắt người. Đó cũng là một lí do khiến ta sử dụng hồng ngoại cho mục đích điều khiển từ xa: dùng nó nhưng lại không cần thấy nó. Một lí do khác nữa là các LED hồng ngoại rất dễ chế tạo và rất rẻ.

Có một điều không may là: trong tự nhiên, có rất nhiều nguồn phát sóng hồng ngoại, chẳng hạn như: mặt trời, cơ thể con người... Thực tế, hầy như mọi vật tỏa nhiệt đều phát ra hồng ngoại và là nguồn gây nhiễu.

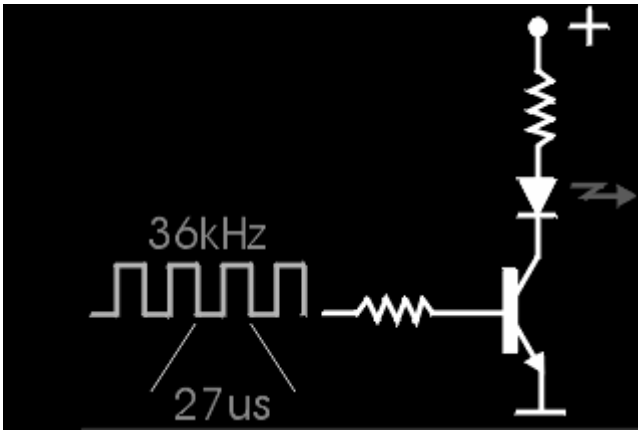
Điều chế tần số là cách để tín hiệu của chúng ta nằm ngoài tác động của nhiễu. Theo cách này, chúng ta làm cho nguồn hồng ngoại phát ra các tín hiệu ở tần số xác định. Và thiết bị thu sẽ được chỉnh tới tần số đó, vì thế có thể loại bỏ được nhiễu. Tần số tốt nhất nằm trong khoảng từ 30- 60 kHz, hay sử dụng nhất là 36 kHz.

1.2 Thiết bị phát hồng ngoại

Thiết bị phát thường là các thiết bị sử dụng pin, tiêu thụ ít năng lượng. Nhiều chip đã được thiết kế để sử dụng như là thiết bị phát hồng ngoại (IR transmitter). Khi không có phím nào được nhấn, chúng ở trong trạng thái 'ngủ', tiêu thụ rất ít năng lượng. Và sẽ được đánh thức để phát ra tín hiệu hồng ngoại khi có một phím được nhấn.

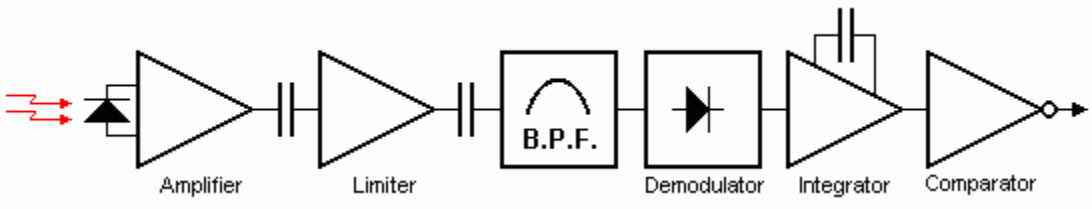
Bộ cộng hưởng thạch anh ít được sử dụng trong thiết bị cầm tay như thế này, vì chúng dễ bị vỡ, không có khả năng chịu sốc. Các bộ cộng hưởng gốm là sự thay thế hợp lý mặc dù chúng hoạt động ít chính xác hơn.

Dòng qua LED hồng ngoại có thể từ 100mA tới trên 1A, và có sự tỉ lệ thuận giữa khoảng cách điều khiển và cường độ dòng qua LED. Như vậy là: nếu muốn sử dụng ở khoảng cách xa thì tuổi thọ của Pin sẽ thấp.

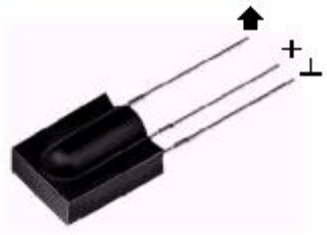


1.3 Thiết bị nhận hồng ngoại

Một mạch nguyên lý thường được sử dụng là:



Ta sẽ không đi sâu vào hoạt động của mạch này vì thực tế ra thì mọi thành phần trên đều được tích hợp vào một thiết bị điện tử duy nhất, có thể tìm thấy nhiều trên thị trường

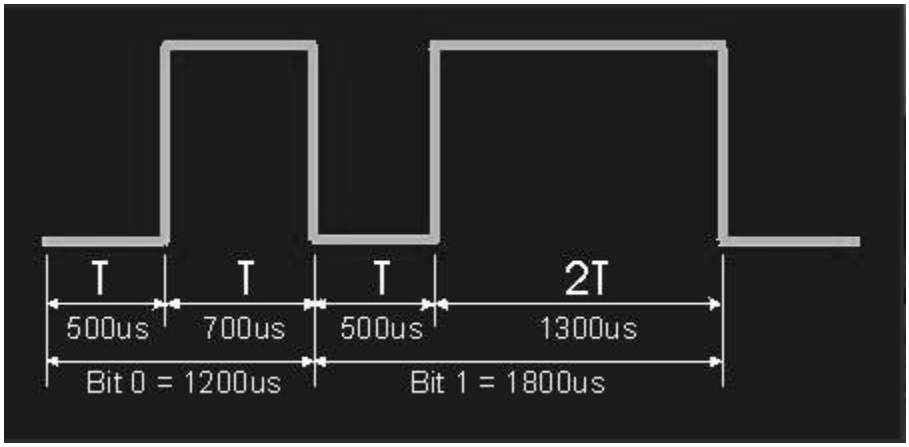


Mắt thu hồng ngoại như trên có 3 chân: chân ở giữa nối với VCC, chân gần nhất sẽ được nối với GND, chân còn lại là chân xuất tín hiệu nhận được

1.4 Mã tín hiệu của một số thiết bị điều khiển từ xa

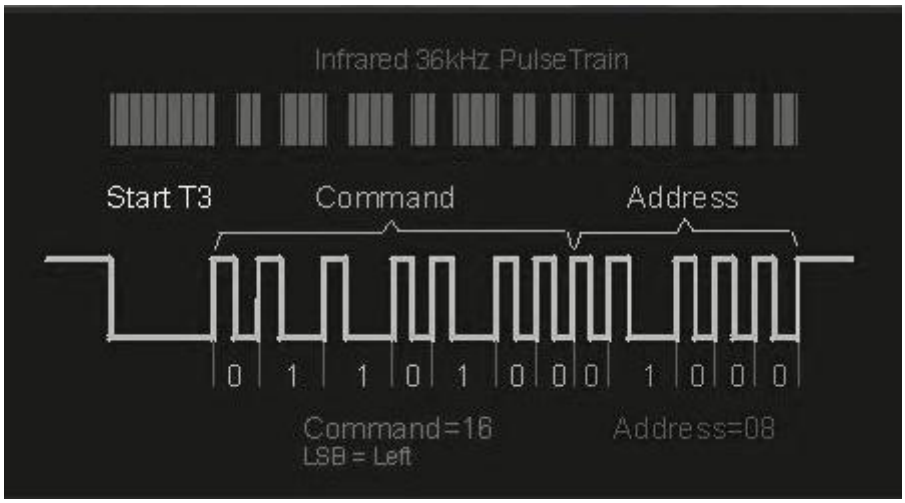
Hầu hết các thiết bị điều khiển từ xa sử dụng cách mã hóa các bit 0 và 1 dựa trên 'chiều dài bit'. Và về khuôn dạng tín hiệu thì không có một qui định chung nào. Mỗi hãng có một qui định khác nhau

Đối với điều khiển SONY RM-Y123: các bit được truyền là tập hợp của bit các bit 0 và 1 với qui định về chiều dài như hình vẽ dưới đây.



Chú ý rằng: phần cao trong hình trên có nghĩa là tín hiệu hồng ngoại đang được truyền bởi sóng mang với tần số 36 kHz; phần thấp ứng với việc không có gì được truyền. Và thực tế, dạng tín hiệu đầu ra của mắt thu hồng ngoại luôn được đảo ngược.

Sau đây là khuôn dạng của một mẫu tín hiệu:

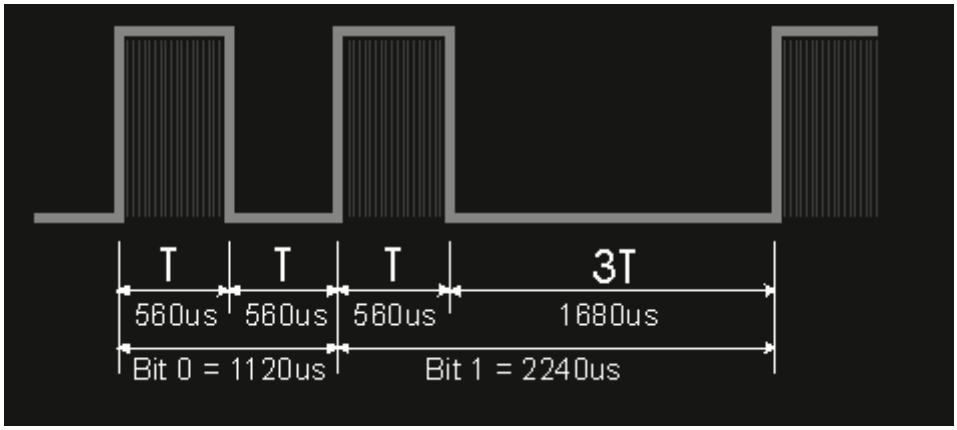


Đầu tiên là phần Header (có thể xem là START bit), với chiều dài 1800us. Theo sau là 12 bit (đã được điều chế như trên), trong đó: 7 bit đầu tiên dành cho phần mã lệnh, 5 bit còn lại dành cho địa chỉ của thiết bị. Như vậy có thể có 2^5=32 địa chỉ và 128 mã lệnh. Các địa chỉ dùng để phân biệt các loại điều khiển khác nhau.

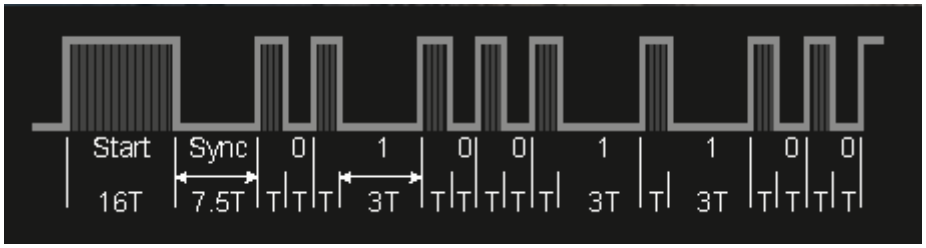
Nếu như có một nút được giữ thì toàn bộ thông tin sẽ được gửi lại cách nhau 25ms.

Điều khiển JVC:

Qui ước về chiều dài bit:

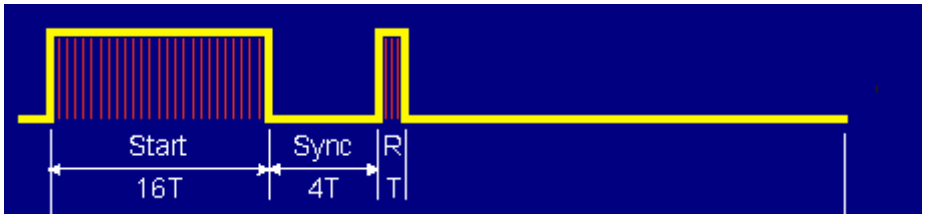


Qui ước về khuôn dạng gói tin:



Phần SYNC dùng cho mục đích đồng bộ trong quá trình nhận.

Nếu như bạn giữ một phím nào đó, thay vì gửi lại toàn bộ thông tin theo kiểu điều khiển SONY, điều khiển JVC chỉ gửi các gói tin REPEAT cách nhau 110ms, theo khuôn dạng



1.5 Một cách lấy mã tín hiệu hồng ngoại đơn giản

Bằng việc tham khảo các tài liệu trên internet, nhóm đã gặp nhiều thuật toán để phát hiện và nhận diện các mã tín hiệu hồng ngoại. Điểm đáng nói ở đây là: các thuật toán chỉ áp dụng cho một loại điều khiển nhất định. Điều đó có nghĩa là: nếu bạn sử dụng thuật toán đọc cho điều khiển SONY cho điều khiển JVC thì kết quả đọc được có thể không đáng tin, thay đổi theo thời gian.

Tự bản thân nhóm đã phát triển một thuật toán khác, có thể áp dụng cho mọi loại điều khiển có trên thị trường hiện nay. Thực tế, nhóm mới chỉ có điều kiện kiểm tra trển các điều khiển SONY và JVC. Thuật toán này dựa trên các nhận xét quan trọng sau:

+ Với tín hiệu xuất ra từ mắt thu hồng ngoại:

+ Phần bắt đầu của tín hiệu ở mức logic 0 trong khoảng thời gian > START_THRESHOLD.

+ Phần kết thúc // 1 // > STOP_THRESHOLD.

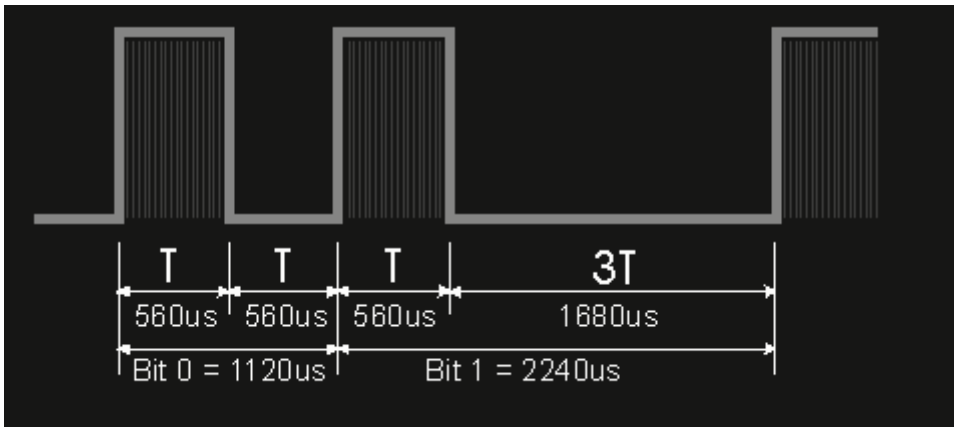
Dựa trên các thống kê về mã tín hiệu, ta có các giá trị:

START_THRESHOLD=800us.

STOP_THRESHOLD=2500us.

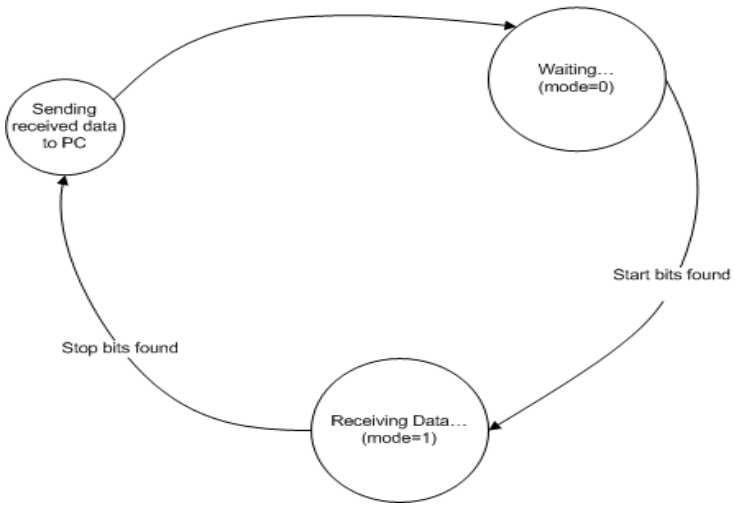
+ Ta không cần đọc chính xác mã của tín hiệu đó là gì. Mục đích chỉ là: thu được mã khác nhau cho các tín hiệu hồng ngoại khác nhau.

+ Mọi mã tní hiệu đều được tạo nên từ các mã của bit 0 và bit 1 (tất nhiên là trừ phần START và SYNC như ở điều khiển JVC chẳng hạn). Và các mã của bit 0 và bit 1 có thể phân biệt rõ ràng với nhau theo độ dài của mã đó.



Chẳng hạn, bit 0 ở trên 'dài' 1120us, bit 1 'dài' 2240us.

Dưới đây là lưu đồ:



+ Ban đầu ở trạng thái đợi (Waiting mode), nếu phát hiện bit start thì chuyển sang trạng thái nhận dữ liệu.

+ Ở trạng thái nhận dữ liệu, nếu phát hiện bit stop thì chuyển dữ liệu lên PC xử lý và chuyển về trạng thái đợi. (Xem chi tiết về thuật toán trong phần tiếp theo của báo cáo)

2 Các vấn đề về giao tiếp USB

2.1 Giới thiệu về USB

Mục này chỉ nhằm giới thiệu những nét cơ bản nhất về USB, nếu muốn hiểu kĩ thì bạn nên xem trong quyển sách "USB Complete" của Jan Axelson.

2.1.1 Giới thiệu chung

Universal Serial Bus (USB) là một chuẩn được phát triển bởi Compaq, Intel, Microsoft và NEC, sau đó là Hewlett- Packard, Lucent và Philips. Các công ty này thành lập USB Implementers Forum, Inc là một công ty phi lợi nhuận công bố những chi tiết kỹ thuật và những phát triển sau này của USB.

Mục đích của chuẩn USB là để tìm phương pháp kết hợp các phương thức kết nối đến máy PC đang được sử dụng. Chúng ta đã có cổng nối tiếp, cổng song song, các kết nối bàn phím và chuột, cổng Joystick, cổng midi vv… Và không có chuẩn nào thỏa mãn những yêu cầu cắm là chạy. Ngoài ra nhiều trong số những cổng đó được sử dụng như là một tài nguyên của PC, chẳng hạn như ngắt phần cứng, và các kênh DMA.

Vì vậy, các cổng USB đã được phát triển như là một phương tiện mới để kết nối một số lượng lớn các thiết bị với máy PC, và cuối cùng để thay thế các cổng thừa. Nó không cần một ngắt nhất định hoặc tài nguyên DMA, và cũng có thể 'hot- pluggable'. Quan trọng là không cần một user-knowledge đặc biệt nào để cài đặt một thiết bị mới, và tất cả các thiết bị đều có để phân biệt được bởi các thiết bị khác.

Rõ ràng là xây dựng một hệ thống rất thân thiện với người sử dụng như vậy sẽ có ý nghĩa rất nhiều về sau cho các nhà phát triển.

2.1.2 Tốc độ truyền dữ liệu

Loại	**Tốc độ**
Low Speed	1.5 Mbit/s
Full Speed	12 Mbit/s
High Speed	480 Mbit/s

Trên đây là các tốc độ dữ liệu của USB. Đây là tốc độ đồng hồ của hệ thống.

- Low Speed

Nó được dành cho các thiết bị trao đổi dữ liệu chậm như chuột. Các loại cáp cho low speed thường mỏng và mềm hơn cáp cho Full và High Speed.

- Full Speed

Nó được dùng cho tất cả các thiết bị khác

- High Speed

Các phần high speed bổ sung các đặc điểm kỹ thuật trong USB 2.0 ví dụ như Firewire

2.1.3 Kiến trúc

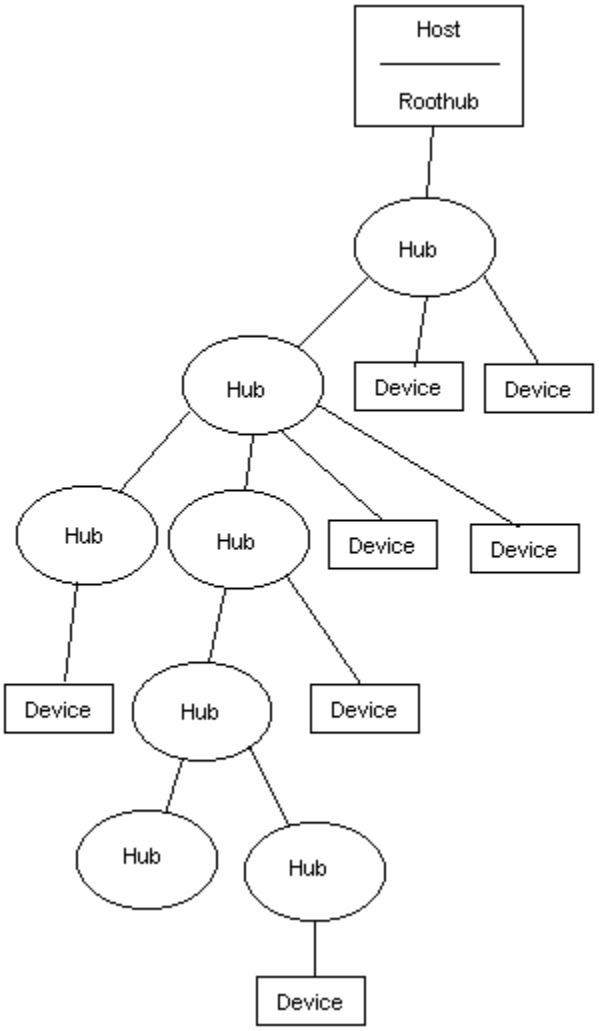
USB được dựa trên hình dưới được gọi là 'kiến trúc hình sao' trong đó có một host controller, và lên đến 127 thiết bị slave. Các host controller được kết nối với một hub, tích hợp trong máy PC, cho phép một số lượng lớn các điểm kết nối (thường được gọi tắt là cổng). Một hub hơn nữa có thể được cắm vào từng điểm kết nối vv… tuy nhiên nó có số lượng hạn chế.

Như đã nêu ở trên chúng ta có thể kết nối tối đa là 127 thiết bị (bao gồm cả Hubs). Bởi vì trường địa chỉ trong một gói là 7 bit dài, và địa chỉ 0 không thể được sử dụng vì nó là dấu hiệu nhận biết đặc biệt. (Trong hầu hết các hệ thống, bus có thể chạy vượt băng thông, hoặc các tài nguyên, lâu dài trước khi các thiết bị đã được đạt đến 127)

Một thiết bị có thể được cắm vào một hub, và một hub có thể được cắm vào một hub khác vv... Tuy nhiên, số lượng tối đa cho phép là sáu lớp.

Độ dài của cáp bị hạn chế là 5 mét. Điều này có nghĩa là một thiết bị không thể xa quá 30m so với PC, và thậm chí để đạt được giới hạn đó thì sẽ cần đến 5 Hub bên ngoài, trong đó có ít nhất là 2 cái sẽ cần phải được tự cấp nguồn.

Do đó, kết nối USB dành cho các thiết bị ở gần máy PC. Đối với các ứng dụng đòi hỏi khoảng cách tương đối xa tới máy PC, ta cần một hình thức kết nối khác, chẳng hạn như Ethernet.



Kiến trúc hình sao

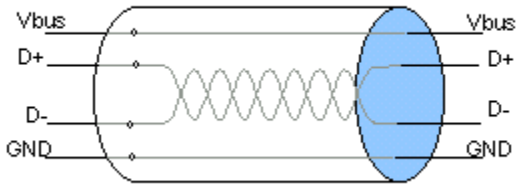
Hub 4 cổng

2.1.4 Cấu trúc vật lý

2.1.4.1 Cáp

Các loại cáp USB đã được thiết kế để đảm bảo chính xác các kết nối. Bằng cách kết nối khác nhau có giữa host và thiết bị, nó là không thể kết nối hai host hoặc hai thiết bị với nhau.

USB cần một cáp bọc vỏ chứa 4 dây. Hai trong số này, D + và D-, tạo thành một cặp dây xoắn chịu trách nhiệm về truyền tín hiệu dữ liệu dưới dạng vi sai (mã hóa NRZ-I), và một số tín hiệu trạng thái kết thúc (để cho loại low speed thì dây dữ liệu có thể không cần xoắn với nhau).



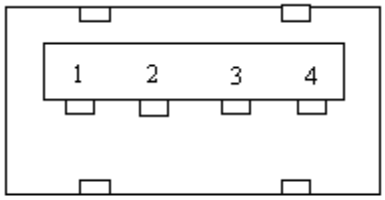
Cấu tạo cáp USB

Chân A, Chân B, và Chân Mini B

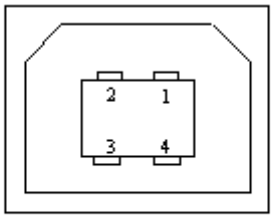
Các tín hiệu trên hai dây trên được tham chiếu đến dây GND (thứ ba).

Thứ tư được gọi là dây VBUS, và mang 5V cung cấp, và có thể được sử dụng như nguồn cho một thiết bị

2.1.4.2 Connector



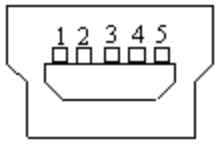
"A" Receptacle



"B" Receptacle

Contact Number	**Signal Name**	**Typical Cable Colour**
1	VBUS	Red
2	D-	White
3	D+	Green

4	GND	Black
Shell	Shield	Drain Wire



"Mini-B" Receptacle

Contact Number	**Signal Name**	**Typical Cable Colour**
1	VBUS	Red
2	D-	White
3	D+	Green
4	ID	No connnection
5	GND	Black
Shell	Shield	Drain Wire

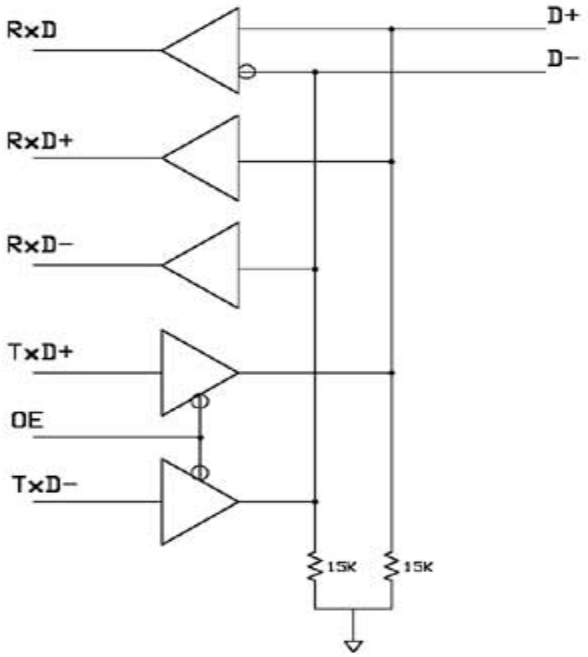
2.1.4.3 Nguồn điện cung cấp

- Một lợi điểm quan trọng của thiết bị USB là không cần nguồn cấp điện bên ngoài mà được cấp từ host
- Thiết bị USB có 3 chế độ nguồn:
    1. Low power: dòng cung cấp 100mA, điện áp trong khoảng 4.4V - 5.25V
    2. High power: dòng cung cấp tối đa 500mA, điện áp bus từ 4.75V-5.25V
    3. Self power: thiết bị được cấp nguồn từ bên ngoài

2.1.5 Luồng dữ liệu

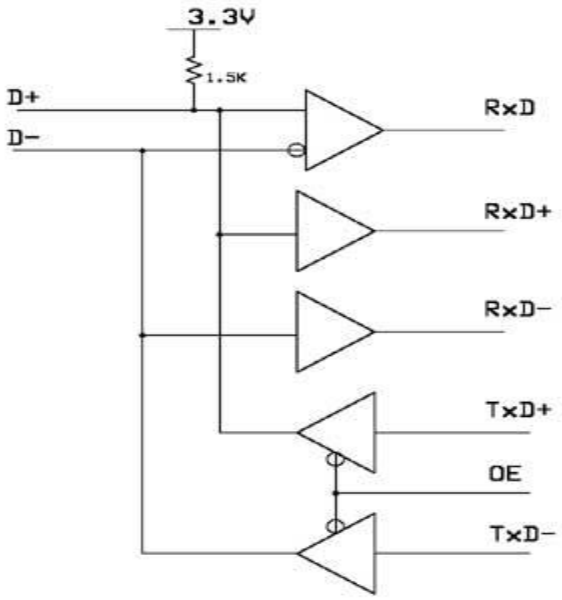
2.1.5.1 Gửi nhận

Đến mỗi đầu cuối của dữ liệu, liên kết giữa các host và thiết bị là một mạch transceiver. Các transceivers là giống nhau, khác nhau chủ yếu ở các điện trở liên kết. Một thiết bị không thể bắt đầu một việc gửi nhận, mà phải chờ để được yêu cầu chuyển dữ liệu của các host. Bằng upstream, ta có thể coi nó gần giống host. Upstream có hai điện trở pull down 15K.



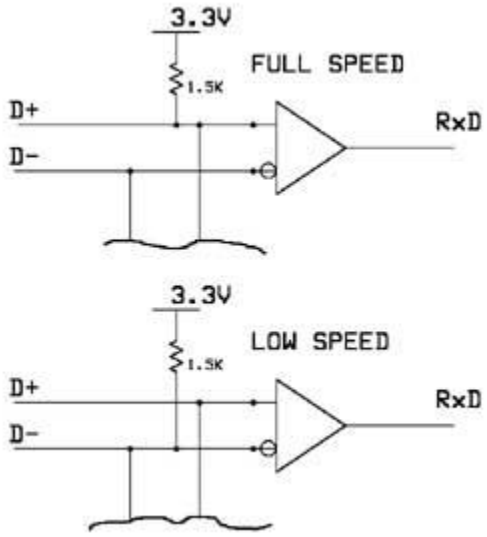
Gửi nhận theo kiểu upstream

Tương đương downstream transceiver được hiển thị bên phải đây. Khi nhận, các receiver riêng biệt trên mỗi dòng có khả năng phát hiện một tín hiệu kết thúc, được gọi là điều kiện Single Ended Zero (SE0), khi phát hiện hai dòng thấp. Ngoài ra còn có một phần nhận đáng tin cậy để tiếp nhận dữ liệu



Gửi nhận theo kiểu Downstream (Full Speed)

2.1.5.2 Xác định tốc độ hoạt động



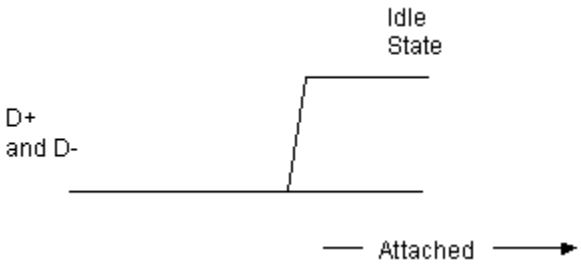
Một điện trở 1.5k ohm đẩy một trong 2 dây lên 3.3v cung cấp bởi VBus. Điện trở này được nối vào D- khi cho một thiết bị low speed, và vào D+ khi cho một thiết bị full speed (trường hợp high speed được nối tương tự như full speed)

2.1.5.3 Trạng thái dây

2.1.5.3.1 Nhàn rỗi

Khi không có thiết bị nào được cắm vào, các host sẽ nhìn thấy cả hai dòng dữ liệu thấp, như là điện trở 15 kohm kéo mỗi dòng dữ liệu xuống thấp.

2.1.5.3.2 Sử dụng



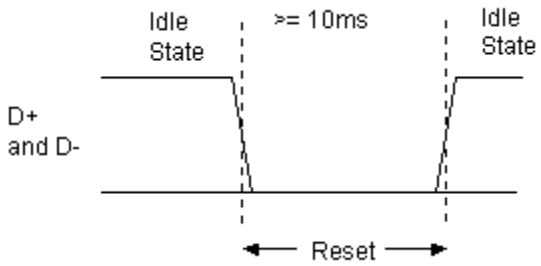
Khi các thiết bị được cắm vào host, các host sẽ thấy, hoặc D+ hay D- có tín hiệu '1 ', và sẽ biết rằng một thiết bị đã được cắm vào.

Tín hiệu '1' ở D- sẽ cho biết là một thiết bị low speed, và ở D+ là full speed hoặc high speed.

2.1.5.3.3 Chờ

Trạng thái của các dòng dữ liệu khi một dây được lên cao, dây còn lại là thấp, được gọi là trạng thái chờ. Đây là trạng thái của các dòng trước và sau khi một gói được gửi.

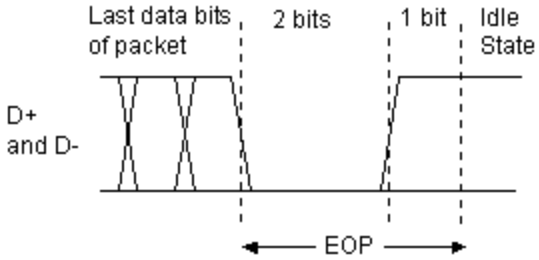
2.1.5.3.4 Reset



Khi host muốn bắt đầu giao tiếp với một thiết bị, nó sẽ bắt đầu bằng cách reset thiết bị về trạng thái chưa được thiết lập cấu hình.

Thiết bị nhận được tín hiệu reset ít nhất sau 2.5 micro giây. Reset không nên nhầm lẫn với reset của vi điều khiển, nó là một giao thức USB để đảm bảo USB bắt đầu một phiên làm việc mới.

2.1.5.3.5 Tín hiệu End of Packet



EOP là một trạng thái SEO cho 2 lần bit.

2.1.5.4 Các kiểu truyền

Có 4 kiểu truyền USB:

- Control transfer:
  - ✓ truyền 2 hướng
  - ✓ hỗ trợ cài đặt, truyền thông tin giữa host và function
  - ✓ Gồm 3 giai đoạn: setup, data, status
  - ✓ Mục đích truyền thông tin khi thiết bị bắt đầu kết nối với host
- Isochronous transfer:
  - ✓ truyền 1 hướng hoặc 2 hướng
  - ✓ mục đích truyền dữ liệu với tốc độ cố định, bỏ qua lỗi
  - ✓ Ví dụ: truyền voice qua USB

- Interrupt transfer:
  - ✓ Chỉ truyền 1 hướng đi vào host
  - ✓ Mục đích truyền dữ liệu nhỏ, không liên tục
  - ✓ Ví dụ: thiết bị sử dụng interrupt như keyboard, mouse
- Bulk transfer:
  - ✓ truyền 1 hướng hoặc 2 hướng
  - ✓ mục đích truyền dữ liệu lớn, chính xác, không khắc khe về thời gian
  - ✓ Ví dụ: scanner, printer, USB disk

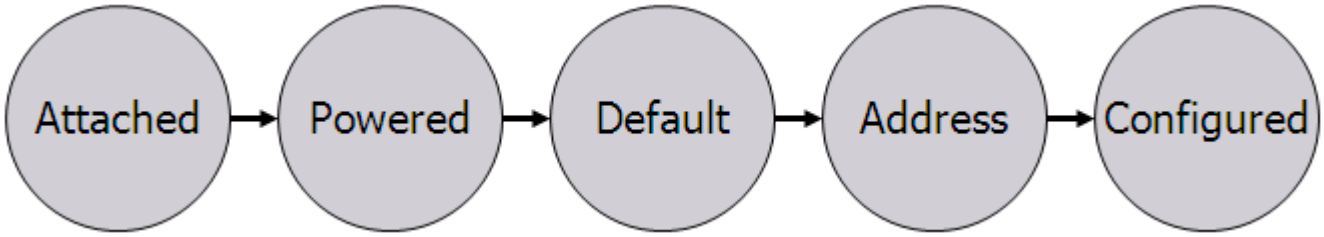
2.1.5.5 Các loại gói dữ liệu

- Việc truyền dữ liệu cổng USB luôn được khởi động từ host
- Dữ liệu truyền đi gồm 5 loại gói (packet):
  - ✓ Token packet: báo mode truyền
  - ✓ Start of Frame packet: chỉ thị bắt đầu 1 khung mới
  - ✓ Data packet: chứa dữ liệu được truyền đi
  - ✓ Handshake packet: dùng để xác nhận dữ liệu và báo lỗi
  - ✓ Special packet: dùng để báo tốc độ mà host muốn truyền

Giao tiếp USB dựa trên giao thức riêng, mỗi gói tin đều có khuôn dạng riêng. Vì lí do phần này quá phức tạp, và trong quá trình làm thì nhóm cũng không ....phải can thiệp đến mức giao thức, cho nên, phần này chỉ trình bày đến đây.

2.1.5.6 Các bước kết nối với thiết bị qua cổng USB

- Khi một thiết bị USB được kết nối vào bus, host sẽ thực hiện 1 tiến trình để xác nhận và quản lý thiết bị, tiến trình này là đặc tính "Plug and Play" của USB
- Tiến trình này gồm các bước sau:
  - Attached: Khi có kết nối thiết bị chủ sẽ nhận biết nhờ sự thay đổi tín hiệu D+ và D-
  - Powered: host gửi lệnh port enable and reset, đồng thời cấp dòng 100mA cho thiết bị, lúc này thiết bị được khởi động
  - Default: sau khi reset, thiết bị ở trạng thái mặc định (default) với địa chỉ 0 và endpoint 0
  - Address: host gán 1 địa chỉ cho thiết bị để quản lý
  - Configured: host gửi lệnh GET DESCRIPTOR đến thiết bị để yêu cầu định cấu hình thiết bị, sau đó host gán các giá trị cấu hình thiết bị qua setup packet. Lúc này thiết bị đã sẵn sàng để sử dụng



2.1.6 Vấn đề liên quan tới Device Driver

Một trong những trở ngại chính cho việc phát triển các thiết bị sử dụng giao tiếp USB là vấn đề về device driver. Windows có sẵn nhiều driver mà chúng ta có thể sử dụng để xây dựng ứng dụng của mình. Sử

dụng các lớp USB là một cách để định nghĩa cho một nhóm thiết bị có tác vụ giống nhau. Chẳng hạn như:

- ✓ Human Interface Devices (HID)
- ✓ Audio Class
- ✓ Communications Device Class (CDC)
- ✓ Imaging Class
- ✓ Mass Storage Class

Lập trình driver cho thiết bị là một vấn đề khó, chưa có thể làm ngay. Chính vì thế nhóm quyết định sử dụng HID (Human Interface Device).

2.1.6.1 Các đặc điểm chung của HID

+ Mọi dữ liệu được trao đổi nằm trong các cấu trúc gọi là các report (báo cáo). Host gửi và nhận dữ liệu bằng việc gửi và yêu cầu các report theo control transfer hoặc interrupt transfer. Khuôn dạng của báo cáo có tính linh động và có thể định nghĩa bất kì kiểu dữ liệu nào.

+ Giao tiếp HID phải có 1 endpoint IN cho việc gửi Input Report . Nhiều nhất là 1 endpoint IN và 1 endpoint OUT.

+ IN endpoint cho phép thiết bị HID gửi thông tin tới host trong thời gian bất kì. Ví dụ, không có cách nào để máy tính biết khi nào người sử dụng ấn một phím trên bàn phím. Vì thế driver của host sử dụng các interrupt transaction (giao dịch ngắt) để 'hỏi' định kì thiết bị xem có dữ liệu mới hay không.

2.1.6.2 Các yêu cầu về phần cứng

Mọi thiết bị HID sử dụng endpoint IN cho việc gửi dữ liệu tới host. Endpoint OUT là tùy chọn.

Transfer Type	Source of Data	Typical Data
Control	Device (IN transfer)	Data that doesn't have critical timing requirements.
	Host (OUT transfer)	Data that doesn't have critical timing requirements, or any data if there is no OUT interrupt pipe.
Interrupt	Device (IN transfer)	Periodic or low-latency data.
	Host (OUT transfer)	Periodic or low-latency data.

Thiết bị của chúng ta sử dụng Interrupt Transfer. Các interrupt endpoint cung cấp khả năng trao đổi dữ liệu khi chúng ta cần nhận dữ liệu nhanh chóng hoặc có tính chất định kì.

2.1.6.3 Các yêu cầu về Firmware

Firmware của thiết bị cũng phải thỏa mãn các yêu cầu về lớp. Các descriptor của thiết bị phải chứa một interface descriptor (mô tả giao diện) để chỉ rõ lớp HID, HID descriptor, và IN endpoint descriptor. Firmware cũng phải chứa report descriptor, chứa các thông tin về nội dung của một HID report.

Các report descriptor xác định cỡ và nội dung của dữ liệu trong một report của thiết bị, và cũng có thể kèm theo thông tin về việc xử lý dữ liệu thu được như thế nào. Chẳng hạn như: gửi dữ liệu đó đến hàm API phát sinh sự kiện bàn phím. Các giá trị trong descriptor định nghĩa các report như là Input, Output hoặc Feature report. Host nhận dữ liệu trong các Input reports và gửi dữ liệu trong các Output reports. Một feature report có thể được truyền theo 2 hướng.

Các HID descriptor cung cấp thông tin cho host biết cần làm những gì để giao tiếp với thiết bị của chúng ta.

(xem trong file usb_desc_hid 8-byte.h, phần Firmware để biết thêm thông tin)

3 Xây dựng phần cứng RemotePC Controller

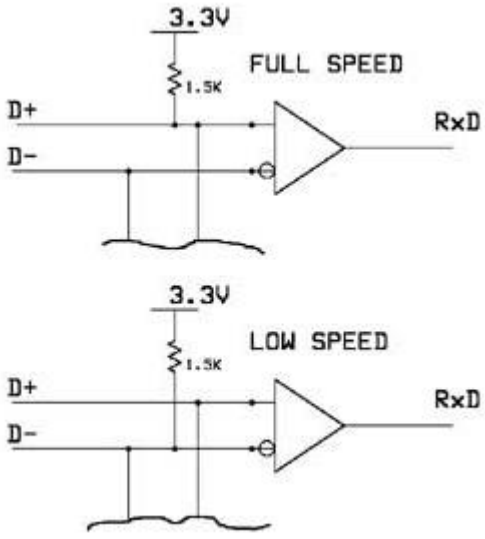
(sử dụng PIC 18F4550, trình dịch PCWH compiler version 4.033 để biên dịch firmware)

3.1 PIC 18F4550 và giao tiếp USB

Phần này không có tham vọng nêu toàn bộ những đặc điểm về PIC18F4550. Thông tin này có thể tra khảo trong datasheet của MicroChip.

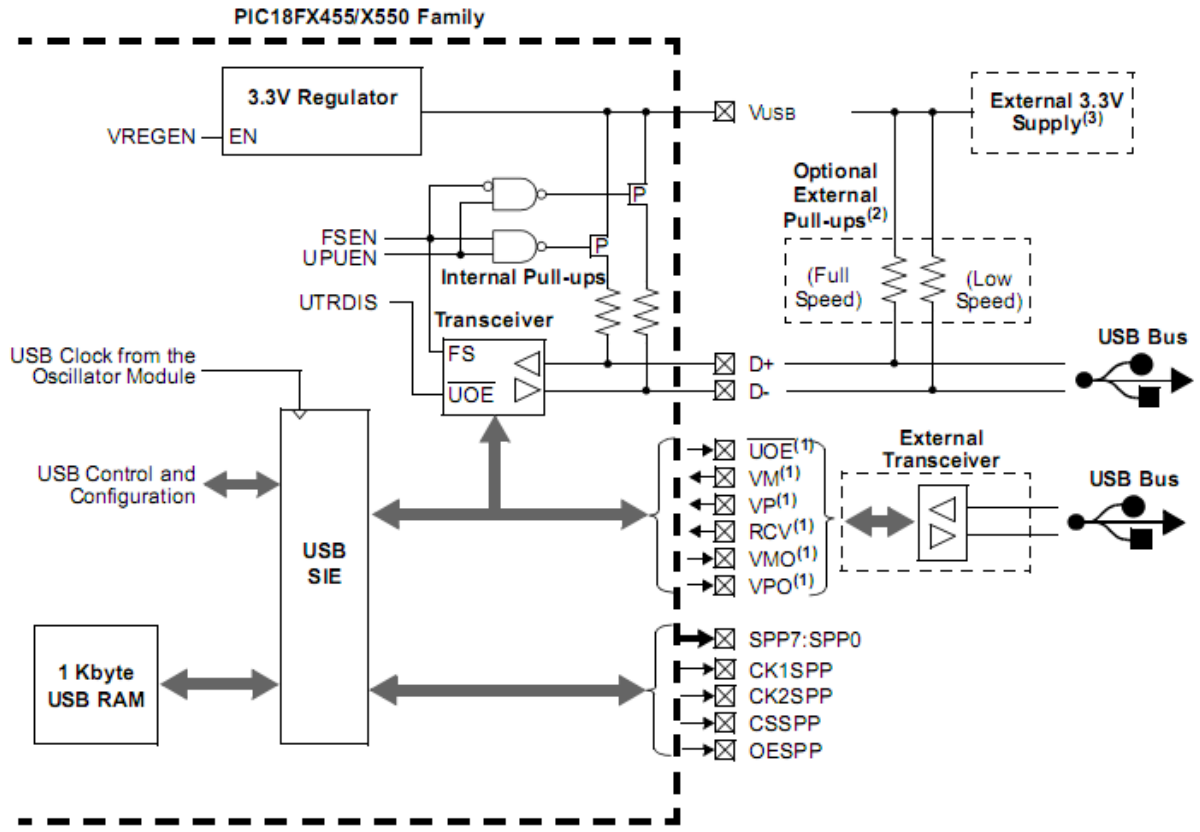
Mục tiêu ở đây là làm sao cho PIC 18F4550 hoạt động được với giao tiếp USB. Cấu hình các thanh ghi sẽ được thực hiện qua các tùy chọn biên dịch động của trình dịch PCWH.

+ Xác định tốc độ cho thiết bị: như ta đã biết, để định tốc độ cho thiết bị sử dụng giao tiếp USB, ta phải sử dụng một điện trở kéo ở phía D+ hoặc D- như trong hình dưới đây:



PIC18F4550 có hỗ trợ điện trở kéo trong (Internal Pull-ups resistors). Ta chỉ định thiết bị hoạt động ở chế độ Full Speed qua khai báo

```
#define USB_USE_FULL_SPEED   TRUE
```



+ Xác định thông tin về End Point: có tất cả 16 End Point (2 chiều) trong PIC 18F4550 và mỗi endpoint được điều khiển bởi một thanh ghi khác nhau.

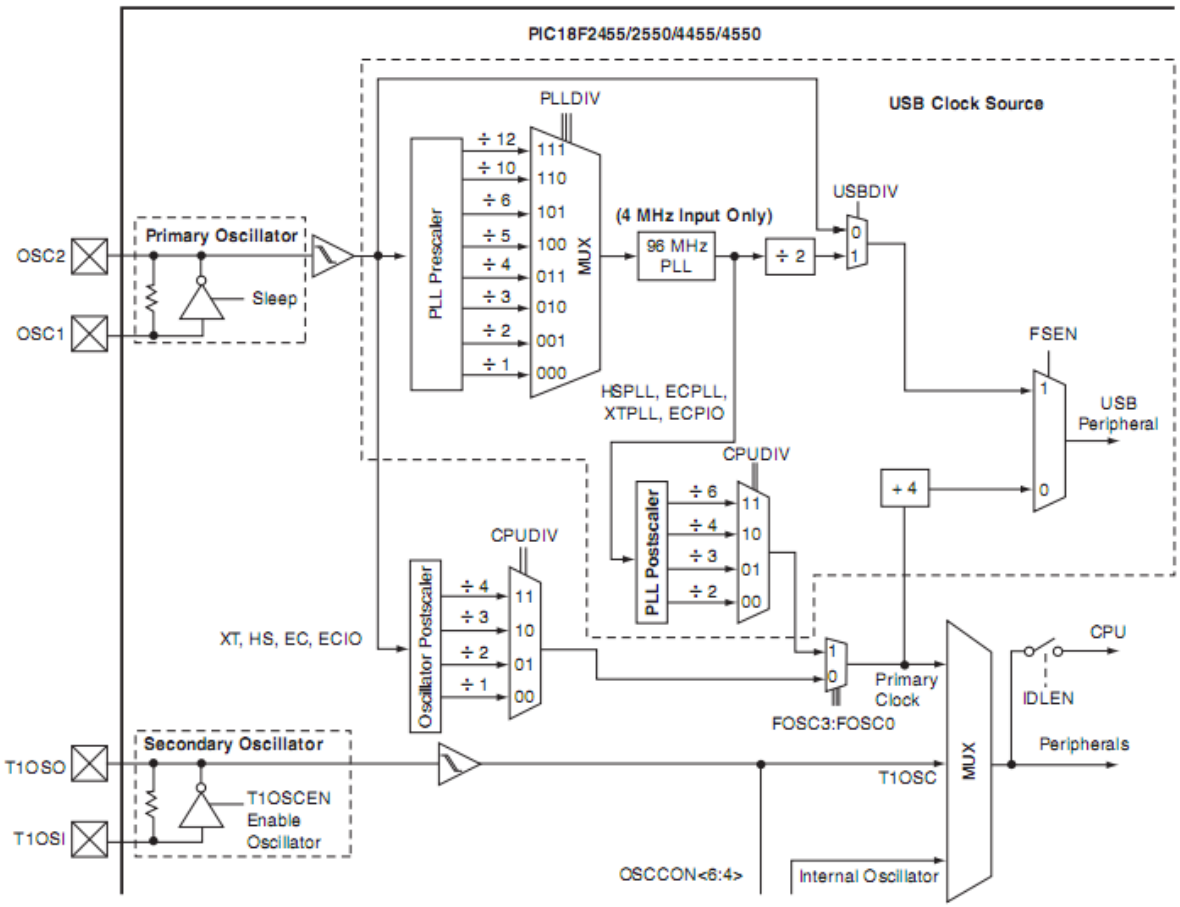
Trong PCWH, dùng các định nghĩa sau để chỉ định EP1 là end point vào, EP2 là end point ra cho PIC.

```
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT   //turn on EP1 for IN bulk/interrupt transfers
#define USB_EP1_TX_SIZE    64  //allocate 64 bytes in the hardware for transmission
#define USB_EP2_RX_ENABLE  USB_ENABLE_INTERRUPT   //turn on EP2 for OUT bulk/interrupt transfers
#define USB_EP2_RX_SIZE    64  //allocate 64 bytes in the hardware for reception
```

+ Xác định thông tin về HID:

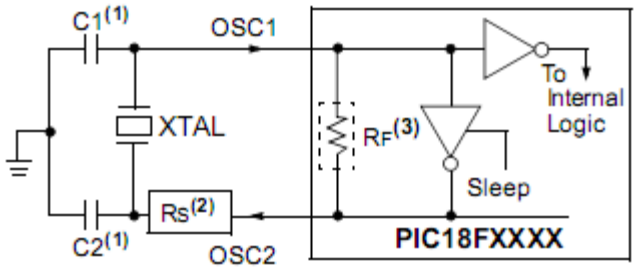
```
#define USB_HID_DEVICE  TRUE //Tells the CCS PIC USB firmware to include HID handling code.
```

+ Xác định thông tin về bộ tạo dao động (Oscillator): Khi PIC 18F4550 được sử dụng với giao tiếp USB, nó phải có clock là 6 MHz (cho Low-speed) và 48 MHz (cho Full-speed). Do PIC 18F4550 chỉ hỗ trợ oscillator trong với tần số lớn nhất là 8 MHz va thiết bị của ta hoạt động ở chế độ Full speed nên ta sử dụng external oscillator.



Hình trên mô tả một phần của mạch tạo xung trong PIC 18F4550. Mạch gồm PLL (Phase Locked Loop) Prescaler với độ chia từ 1- 12; PLL PostScaler với độ chia từ 2-6; PLL chỉ chấp nhận đầu vào 4MHz và tạo ra 96MHz ở mạch ra.

Ta sử dụng thạch anh 20Mhz nối vào các chân OSC1 và OSC2:



(tra trong DataSheet, các giá trị C1 và C2 đều là 15pF)

Phải thiết lập PreScaler=5 (vì 20/5=4), PostScaler =2 (vì 96/2=48) và sử dụng PLL là nguồn phát xung.

Thêm các định nghĩa sau trong PCWH:

```
#FUSES PLL5                    //Divide By 5(20MHz oscillator input)
#FUSES CPUDIV1                 //System Clock by 1
#FUSES USBDIV                  //USB clock source comes from PLL divide by 2
#use delay(clock=48000000)     // operates at the frequency of 48Mhz
```

+ Ngoài ra, ta có sử dụng timer 2 (8 bits) để gây ngắt sau 20us. PCWH có cung cấp sẵn hàm để thiết lập timer 2

setup_timer_2(Prescale,Period,Postscale)

Thời gian gây ngắt được tính theo công thức:

Timer= (4 x ClockPeriod) x Prescale x Postscale x Period = 20 us

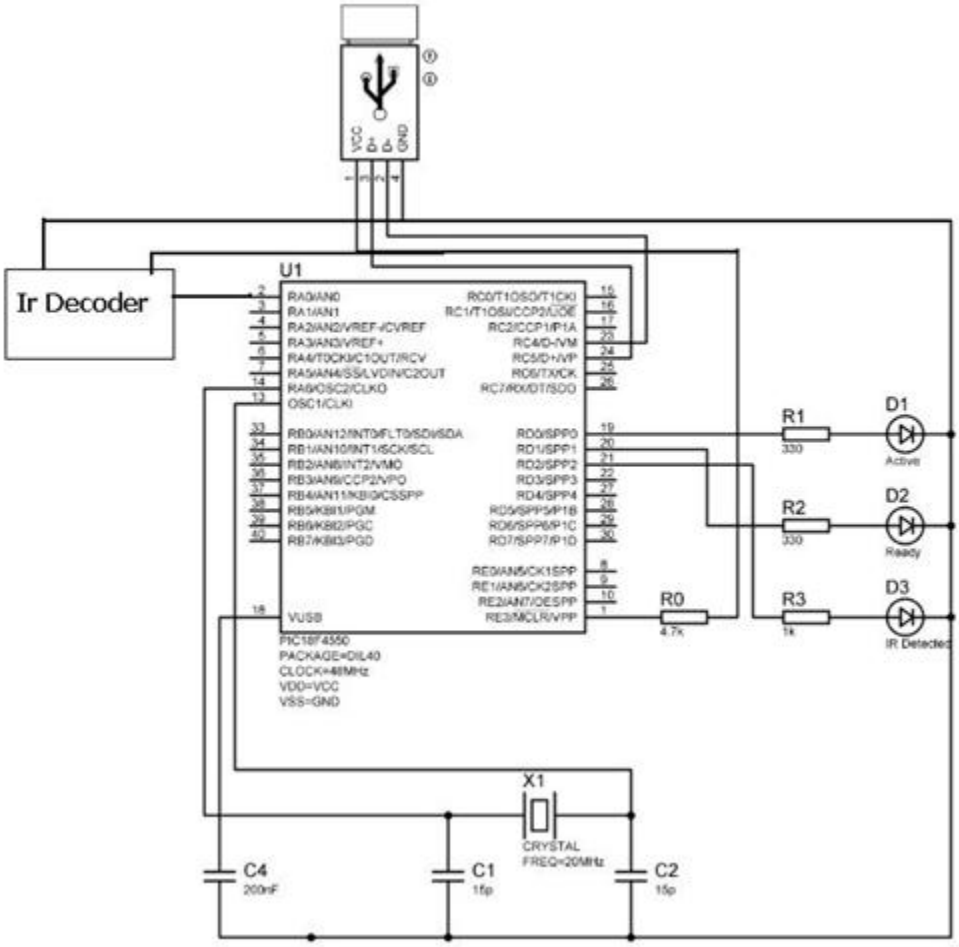
Thay ClockPeriod=1/(48 MHz)= 1/(48e6 Hz), ta dễ có một bộ giá trị chấp nhận được là:

Prescale=1

Period=240

Postscale=1

3.2 Sơ đồ mạch RemotePC Controller



3.3 Firmware

Nhắc lại: Firmware được biên dịch trên PCWH Compiler version 4.033, có dựa trên việc tham khảo một chương trình mẫu đi kèm PCWH (ex_usb_mouse.c)

Firmware.h

```
#include <18F4550.h>
#FUSES HSPLL
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES BROWNOUT                 //Reset when brownout detected
#FUSES BORV20                   //Brownout reset at 2.0V
#FUSES NOPUT                    //No Power Up Timer
#FUSES STVREN                   //Stack full/underflow will cause reset
```

```
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOLVP                    //No Low Voltage Programming on B3(PIC16) or B5(PIC18)
#FUSES PLL5                    //Divide By 5(20MHz oscillator input)
#FUSES CPUDIV1                  //System Clock by 1
#FUSES USBDIV                  //USB clock source comes from PLL divide by 2
#FUSES VREGEN                   //USB voltage regulator enabled



#use delay(clock=48000000)      // operates at the frequency of 48Mhz
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
//                              Define some stuff for using USB interface

#build(reset=0x1, interrupt=0x8)          // Necessary for Bootloader
#ORG 0x0F00,0x0FFF {}                    // Necessary for Bootloader



// CCS Library dynamic defines

#define USB_HID_DEVICE  TRUE //Tells the CCS PIC USB firmware to include HID handling code.

#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT   //turn on EP1 for IN bulk/interrupt transfers
#define USB_EP1_TX_SIZE    64  //allocate 64 bytes in the hardware for transmission
#define USB_EP2_RX_ENABLE  USB_ENABLE_INTERRUPT   //turn on EP2 for OUT bulk/interrupt
transfers
#define USB_EP2_RX_SIZE    64  //allocate 64 bytes in the hardware for reception

#define USB_USE_FULL_SPEED   TRUE

// CCS USB Libraries
#include <pic18_usb.h>   //Microchip 18Fxx5x hardware layer for usb.c
#include "usb_desc_hid 8-byte.h" // This header contains an HID configuration for our device
#include "usb.c" // a sample program provided by CCS compiler
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
//                                  OUR OWN CONSTANT DEFINITIONS

#define IR_CONNECTED_PIN PIN_A0
#define LED_DEVICE_ACTIVE PIN_D0 // Our device's connected to a computer
#define LED_DEVICE_READY PIN_D1 // Our program's running on the computer
#define LED_SIGNAL_DETECTED PIN_D2 // Ir Signal detected

// define a basic command set to interchange information between the device and HugeBrain Assistant App
#define COMMAND_APP_READY 0    // HugeBrain Assistant App's Initialization's successfull
#define COMMAND_SET_TIMER_INTERVAL 1  // The App sets up a value for the timer 2's interval
#define COMMAND_APP_SHUTDOWN 2 // The Apps's shutdown

// define end points for the USB interface
// IN or OUT is based on the host's perspective
#define END_POINT_IN 01 // related to the 153rd line in usb_desc_hid 8-byte.h
#define END_POINT_OUT 02 // related to the 165th line in usb_desc_hid 8-byte.h

// define some mode used by the Timer 2's Interrupt Service Routine (ISR)
#define MODE_WAITING 01 // Waiting for the start token
#define MODE_RECEIVING 02 // Receiving data

//Start token: a series of Zero-valued signals detected
#define DEFAULT_START_THRESHOLD 400

// Stop THRESHOLD: technically, a stop THRESHOLD represents the minimum time between 2 consecutive the IrDecoder's frames
#define DEFAULT_STOP_THRESHOLD 1250
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                  VARIABLE DEFINITIONS

int8 InData[8];// Array to store received data get from the host
//int8 OutData[8];// Array to send hash values of IrDecoder code to the host
int8 OUT_SIZE=64;
int8 OutData[64];
int8 OutIndex=0;

int8 IsSendingDataToHost;// If this variable's not zero, so the ISR will ignore any signal received.

int16 STOP_THRESHOLD=DEFAULT_STOP_THRESHOLD;
int16 START_THRESHOLD=DEFAULT_START_THRESHOLD;

// define some variable to use exclusively by the Timer 2 's Interrup Serivce Routine (ISR)
int8 Mode=MODE_WAITING ;// Represent 2 modes: MODE_WAITING or MODE_RECEIVING
char Received=0, LastReceived=0;
int16 HashValue;
int16 Counter ;
int8 Serving=0, WaitToSync=0;

// define variables for adjusting the timer 2's interval
// By default, the timer 2 is set up to trigger the ISR every QuantumTime(us)
//int8 QuantumTime=20;
int8 TimerPrescale=T2_DIV_BY_1, TimerPeriod=240, TimerPostscale=1;
////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Firmware.c

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                          HugeBrain Assistant's Firmware  v1.0
//                              Programmed by: Le Duc Anh, Ho Hai Dang, Lai Minh Huy, Vu Hanh Hop
//                              Team         : HugeBrain (Tin 2 K50)
//                              Homepage     : HugeBrain.groups.google.com
```

```
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Operation Description:
//                    1. WAITING MODE
//                       Our device's connected to a PC through a USB interface. When it's enumerated, the
//                       HugeBrain Assistant App (running on the PC) will send a configuration to our device.
//                       If everything's OK, then turns on the Red LED, switch to the RECEIVING MODE.
//
//                     2.RECEIVING MODE
//                       From now on, every 20us our device will get the signal at the PIN_A0
//                              + Check to see if it's the Start signal
//                              + If true, afterward our device will get the total signal to send to HugeBrain Assistant App.
//                                 Then the App will take an appropriate action
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define __USB_PIC_PERIF__ 1
#include "Firmware.h"
#int_TIMER2
void  TIMER2_ISR(void) // this service routine will be called every 100us
{
  if(Serving)
    return;
  Serving=1;

  Received= input(IR_CONNECTED_PIN);// Get the current status of the pin connected with the IrDecoder

  if(Mode== MODE_WAITING){

    if(Received== 0){
      Counter++;
      if(Counter>= START_THRESHOLD) // Start THRESHOLD detected
      {
        Mode=MODE_RECEIVING; // so the next state will be the receiving mode
        output_bit(LED_SIGNAL_DETECTED,1);// inform that we've detected a new signal
        WaitToSync=1;
```

```
        // (re)initialize variables
        OutIndex=2;


      }
    }
    else
      Counter=0;
}
else{ // so we are in the receiving mode.
    if(Received==LastReceived){
      if(!WaitToSync){
          Counter++;
        if(Counter>=STOP_THRESHOLD){// Stop THRESHOLD detected
          Mode=MODE_WAITING;
          Counter=0;// reset Counter
          OutData[0]=OutIndex;//size of array
          OutData[1]=0;
          IsSendingDataToHost=1;
          return;
        }
      }
    }
    else{

      if(!WaitToSync){
          if(LastReceived==0){
            //Start a new bit
            //Delta=(Counter0<<4) ^ Counter1;
            if(OutIndex+1 < OUT_SIZE){
              Counter>>1;// devide Counter by 2
              OutData[OutIndex++]=Counter;
            }
            Counter=1;
          }
          else
            Counter++;
```

```
      }
      WaitToSync=0;
    }
  }
  LastReceived=Received;

  Serving=0;
}

void Init(){
  // CCS generated code
  setup_adc_ports(NO_ANALOGS|VSS_VDD);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_SS_DISABLED);
  setup_wdt(WDT_OFF);
  setup_timer_0(RTCC_INTERNAL);
  setup_timer_1(T1_DISABLED);
  // setup timer 2
  setup_timer_2(TimerPrescale,TimerPeriod,TimerPostscale);
  setup_timer_3(T3_DISABLED);
  setup_comparator(NC_NC_NC_NC);
  enable_interrupts(GLOBAL);

  // Our initialization
  IsSendingDataToHost=0;// nothing to send :)
  Mode=MODE_WAITING;
  Received=0;
  LastReceived=0;
  //Counter=0;

  Serving=0;
  WaitToSync=0;
  // Initialize USB interface
  usb_init();
  delay_ms(100);
```

```
  // inform that our device is in active state
  output_bit(LED_DEVICE_ACTIVE,1);
}
void main()
{
  Init();

  while(1){ // an endless ...loop excepting the case of power loss
       usb_task();
       if (usb_enumerated()) {// if our device's enumerated, so we can communicate with the OS

         if(IsSendingDataToHost!=0){ // Need to send data to host
            // disable the timer 2 interrupt
            disable_interrupts(INT_TIMER2);

            usb_put_packet(END_POINT_IN, OutData, 64, USB_DTS_TOGGLE);
            delay_ms(10);
            // Turn off the led
            output_bit(LED_SIGNAL_DETECTED,0);
            IsSendingDataToHost=0; // nothing to send

            Delay_ms(100);

            Serving=0;

            //Reset the timer 2 counter
            set_timer2(0);

            //Enable the timer 2 interrupt
            enable_interrupts(INT_TIMER2);

         }

         if (usb_kbhit(END_POINT_OUT)) {// check to see if we receive any data
            usb_get_packet(END_POINT_OUT,InData,8);
```

```
            switch(InData[0]){

              case COMMAND_APP_READY:
                  // HugeBrain App initialized successfully
                  // The we turn on the Active LED, IR Ready LED
                  output_bit(LED_DEVICE_READY,1);
                  Serving=0;
                  //Reset the timer 2 counter
                  set_timer2(0);
                  //Enable the timer 2 interrupt
                  enable_interrupts(INT_TIMER2);
                  break;
              case COMMAND_APP_SHUTDOWN:
                  // HugeBrain App's terminated
                  // The we turn off the Ready Led
                  output_bit(LED_DEVICE_READY,0);

                  // disable the timer 2 interrupt
                  disable_interrupts(INT_TIMER2);
                  break;
            }
          }
        }
    }
}
```

4 Xây dựng phần mềm RemotePC Assistant

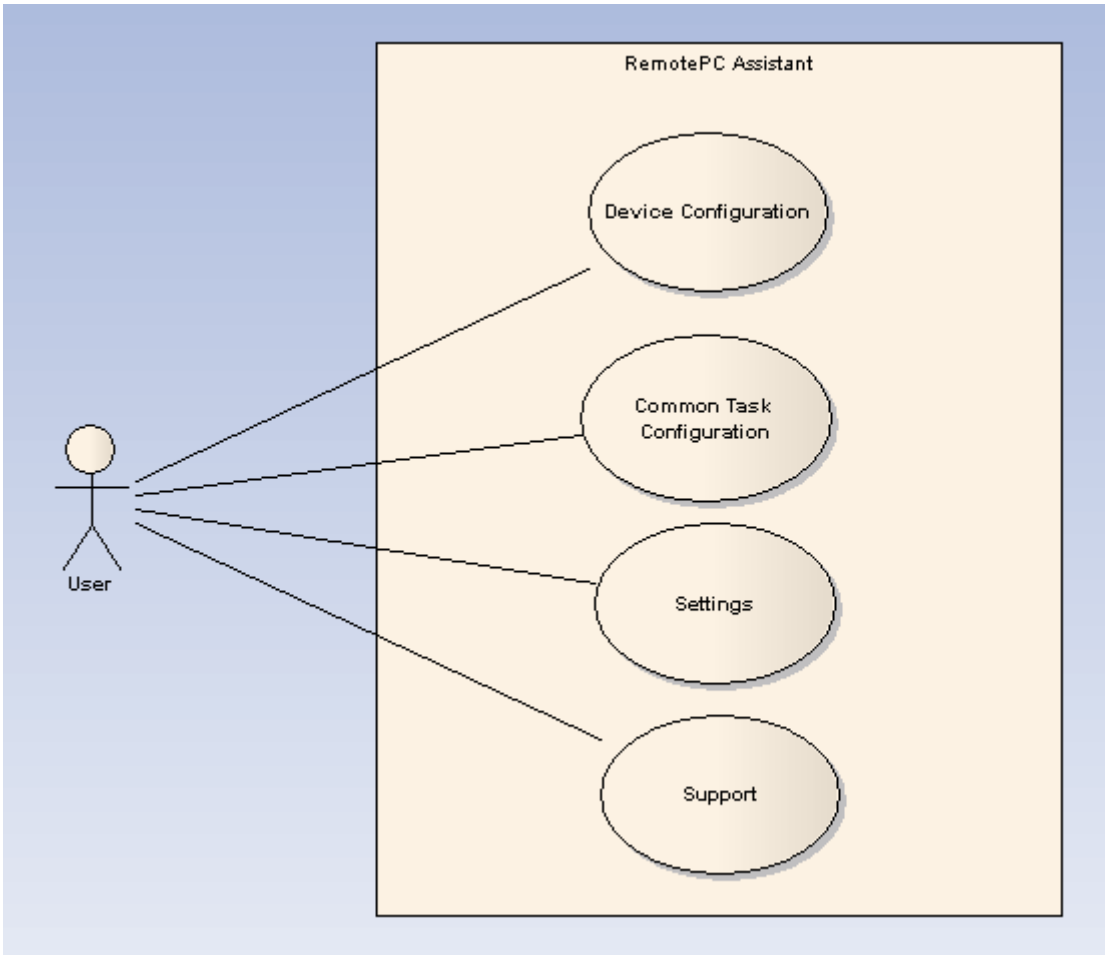
Vì đây là một báo cáo mang nhiều màu sắc về phần cứng, nên chi tiết về thiết kế lớp trong phần mềm cũng như hoạt động sẽ không được trình bày nhiều. Nhóm sẽ tập trung vào việc nêu ra:

+ Phần mềm có thể hỗ trợ người sử dụng điều khiển những tác vụ nào?

+ Phần mềm đã sử dụng HID và dữ liệu chuyển đến từ thiết bị RemotePC Controller thế nào?

Môi trường phát triển: C++, sử dụng IDE:MS Visual Studio 2008, MS Driver Development Kit (DDK) 2003.

4.1 Use cases



Đặc tả chức năng:

+ Device Configuration: Thực ra, đây là chức năng giúp cho chương trình có thể hoạt động 'ăn ý' với thiết bị của người sử dụng. Thông qua một quá trình 'học', chương trình sẽ biết được mã của các tín hiệu hồng ngoại và hành động mà người sử dụng gán cho. Ví dụ như: lấy một cái điều khiển TV nào đó, ấn nút sang trái để cho chương trình nhận được và chọn hành động là mở chương trình PowerPoint.

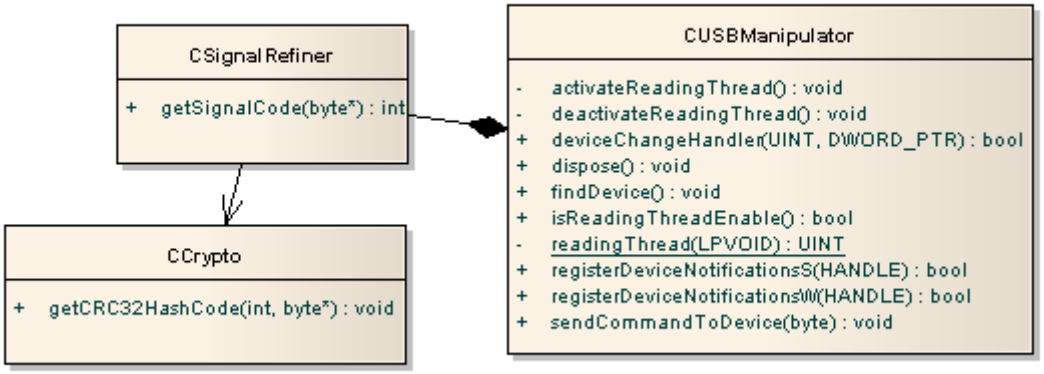
+ CommonTask Configuration: Vì số lượng chức năng mà chương trình hỗ trợ người sử dụng là tương đối nhiều, nên sẽ khó nhớ hết (kể cả việc chương trình+ thiết bị có thể hỗ trợ nhiều phím bấm) các chức năng mà mình đã gán cho các nút. Nhận xét này dẫn đến ý tưởng tập trung các tác vụ mà người dùng hay sử dụng vào một nhóm để truy cập nhanh về sau. Chẳng hạn: các chức năng hay dùng là: Lock (khóa máy tính), shutdown (tắt máy tính), bật PowerPoint....

+ Settings: một vài tùy chọn thiết lập cho chương trình. Trong đó có chức năng đáng chú ý là: "Use Context-based Control", tạm hiểu là điều khiển theo ngữ cảnh. Đây là chức năng hỗ trợ việc đơn giản hóa sử dụng phím bấm trên cơ sở việc phân tích ngữ cảnh.

+ Support: hiện một vài thông tin liên quan đến sản phẩm và hỗ trợ từ nhóm phát triển.

4.2 Thiết kế lớp

4.2.1 Xử lý các yêu cầu liên quan tới thiết bị



Lớp CUSBManipulator là lớp cơ bản, quản lý việc thao tác với thiết bị dựa trên PID và VID định trước. Sau khi tìm được thiết bị rồi, nó 'âm thầm' tạo ra một thread mới để liên tục đọc các thông tin xuất ra từ thiết bị (64 bytes định trước). Dưới đây là khuôn dạng của 64bytes này:

0	1	2	3	…	63
Len	0	…	…	…	…

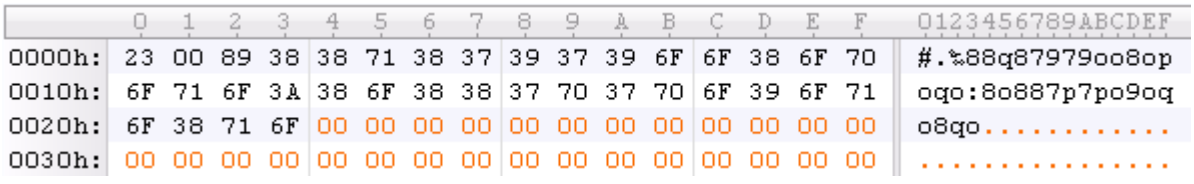
Trong đó: byte ở vị trí 0 (Len) chỉ độ dài của tín hiệu (tính theo số lượng các byte 0 và 1 nhận được)

byte ở vị trí 1: luôn bằng 0.

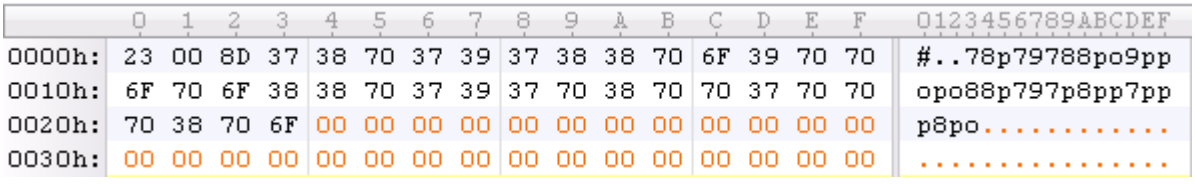
Các byte tiếp theo thể hiện độ dài của mã 0 và 1.

Cũng có điều phải nói rằng:

+ Nếu bạn dùng điều khiển từ xa, ấn 2 lần 1 phím bất kì nào đó thì 64 bytes thu được từ thiết bị cho cả 2 lần không thể hoàn toàn trùng khớp lẫn nhau. Chẳng hạn, trong hình vẽ dưới đây mô tả 64 bytes nhận được khi nhấn cùng 1 phím trên điều khiển LG



Lần 1

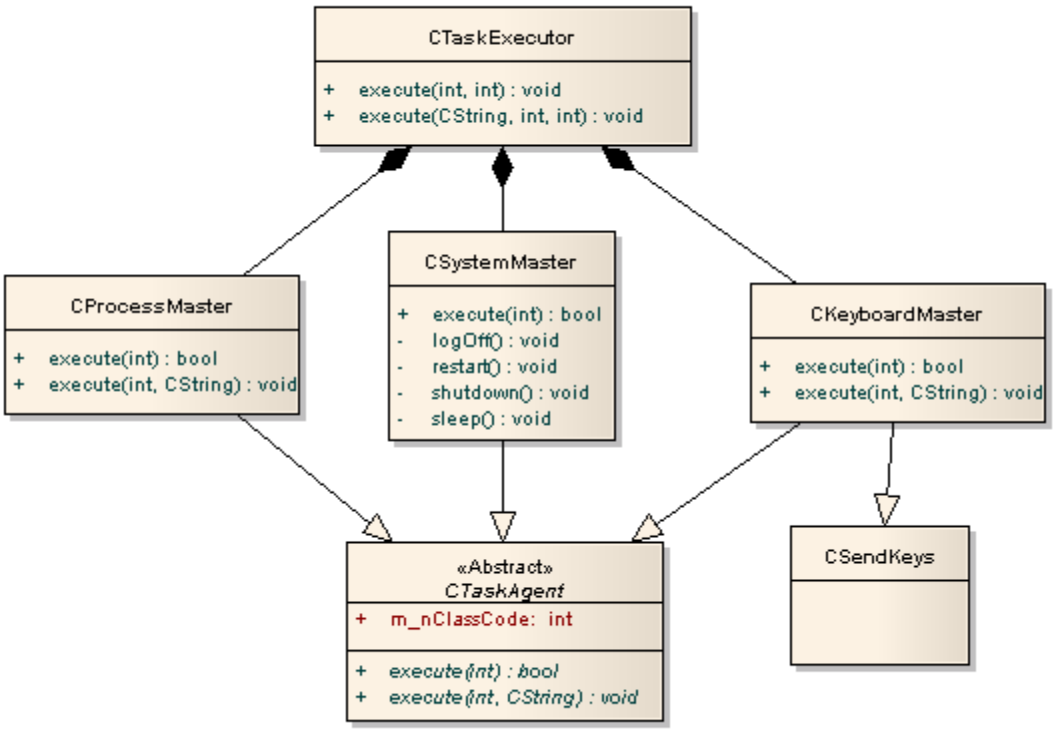


Lần 2

Lí do đơn giản được giải thích là do: hàm delay của thiết bị không tốt, tính ngẫu nhiên của việc xuất hiện các tín hiệu hồng ngoại.... Để tăng độ chính xác cho việc nhận các tín hiệu, ta phải sử dụng thêm lớp CSignalRefiner. Kiểm nghiệm thức tế nhiều lần, cho thấy, thuật toán sử dụng cho kết quả khá tốt.

+ 64 bytes thu được sau quá trình 'tinh chế', được sử dụng để tính mã vòng CRC32. Lí do tại sao sử dụng CRC32 là vì: lập trình tương đối đơn giản và đụng độ thấp.

4.2.2 Tác vụ hỗ trợ



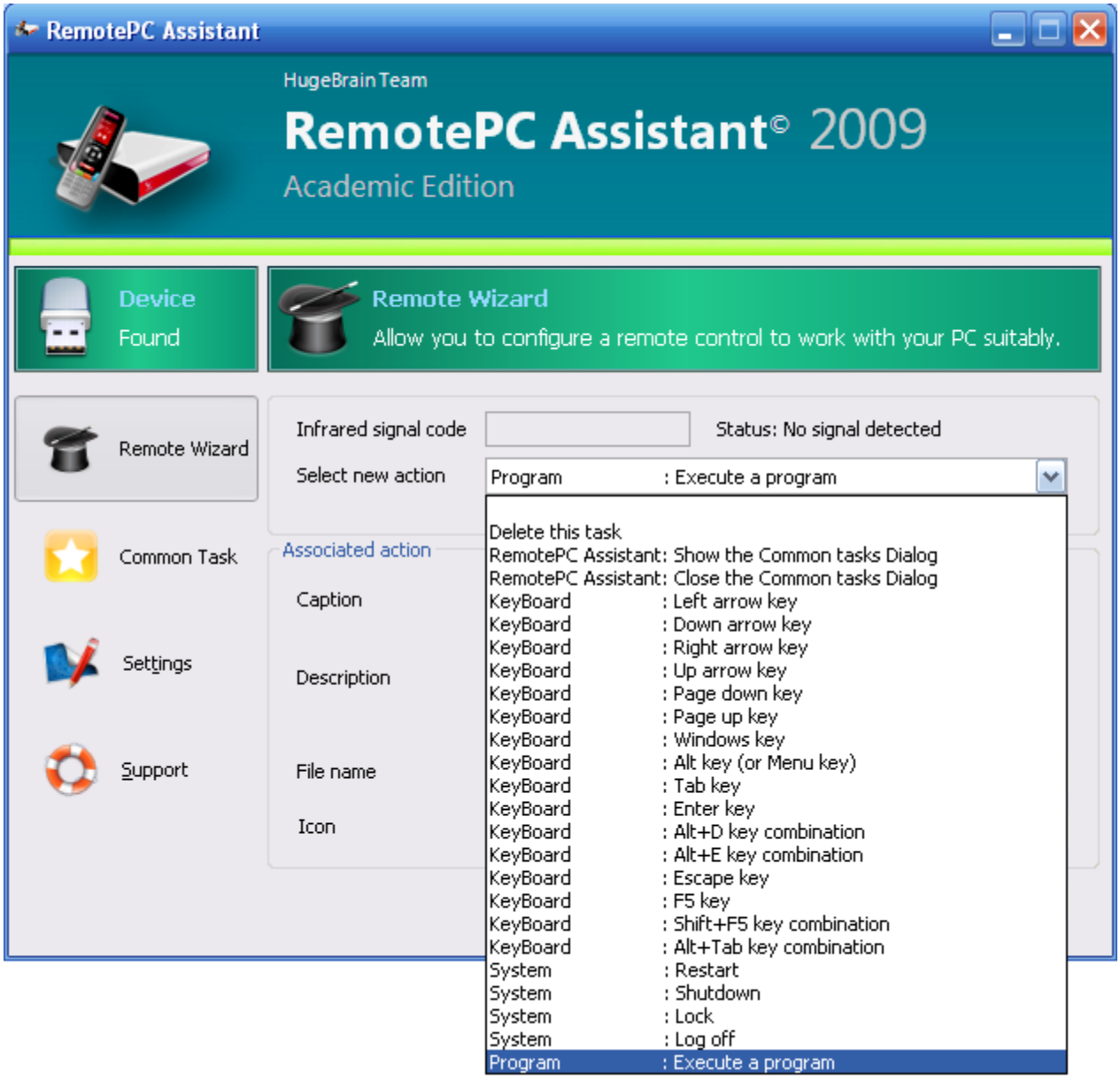
Cho đến nay, chương trình hỗ trợ việc:

+ Tạo ra các sự kiện bàn phím (như ấn phím Windows chẳng hạn) (Lớp CKeyboardMaster đảm nhiệm)

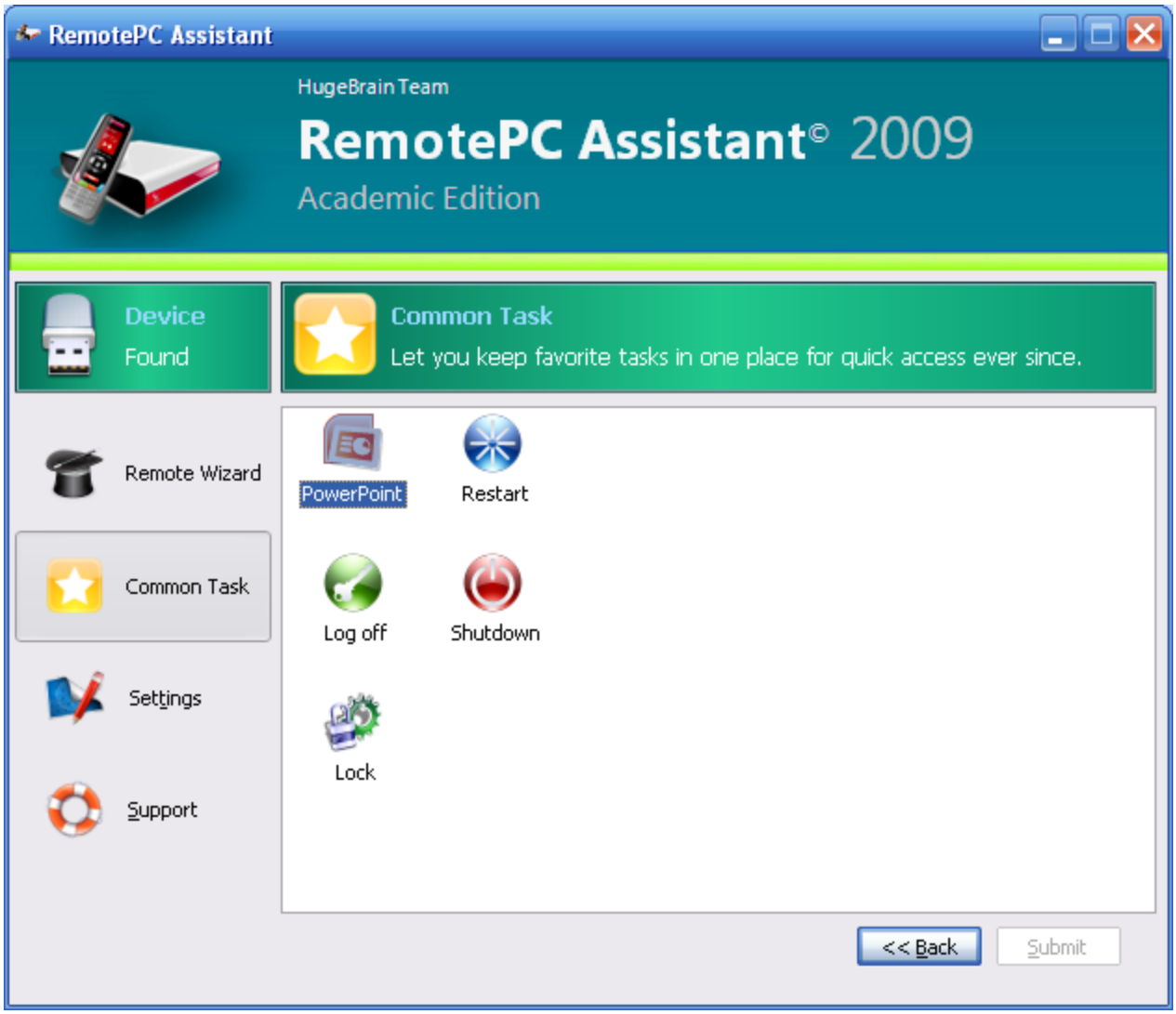
+ Thực thi một chương trình (Lớp CProcessMaster)

+ Tắt, khóa ... máy tính (Lớp CSystemMaster)

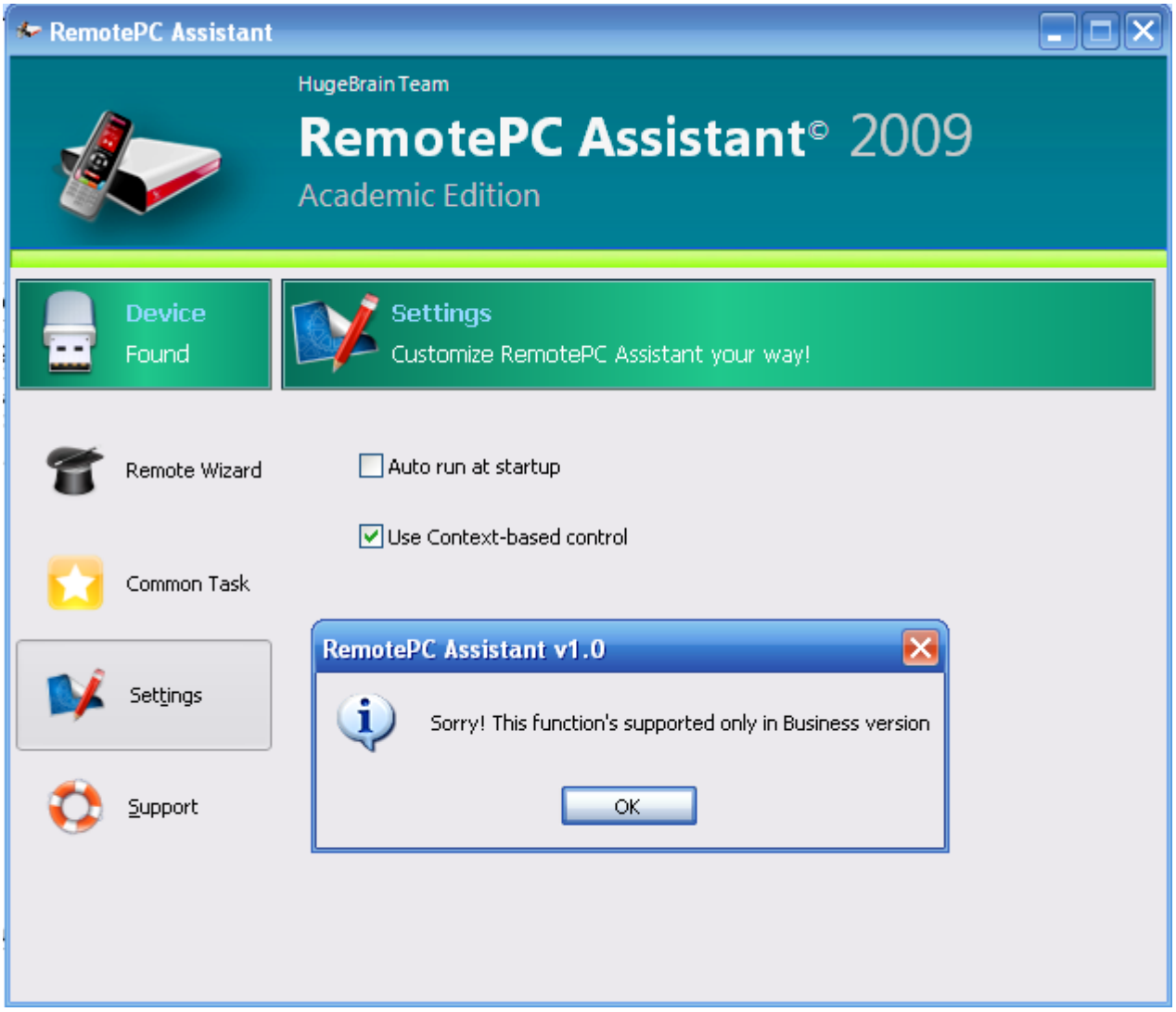
4.3 Một số hình ảnh về chương trình



Remote Wizard: cho phép chương trình 'học' điều khiển từ xa của bạn



CommonTask: lưu giữ các tác vụ hay sử dụng của bạn vào một nơi duy nhất để có thể truy cập nhanh về sau mà không phải nhớ quá nhiều phím bấm.



Settings: Thiết lập vài tham số cho chương trình.

Support: thông tin về nhóm phát triển

5 Đánh giá

5.1 Về mặt sử dụng

Chưa có điều kiện để cho nhiều người đánh giá về sản phẩm. Tuy nhiên, có thể rút ra vài điều như sau:

Ưu điểm:

+ Số tác vụ hỗ trợ nhiều, dễ cấu hình và sử dụng.

+ Có thể sử dụng bất kì loại điều khiển từ xa dùng hồng ngoại nào.

Nhược điểm:

+ Sử dụng hồng ngoại nên chỉ chấp nhận trong vòng bán kính ~ 20 m

+ Một số chức năng trong phần mềm vẫn còn đang dang dở, chưa hoàn thiện như: "Use Context-based control".

5.2 Về mặt kinh tế

Thiết bị	Giá thành (VND)
PIC 18F4550	95 000
Mạch in	30 000
Các thiết bị khác	20 000
Phần mềm RemotePC Assistant	0
Tổng chi phí	155 000

Như vậy, số tiền dành cho PIC 18F4550 là tương đối lớn. Định hướng tương lai là: sử dụng một MCU khác có giá thành rẻ hơn, và tự lập trình giải mã tín hiệu trong giao tiếp USB.

Mặt khác, vì thiết bị và phần mềm không thể tách rời nhau nên cũng hơi bất tiện so với các thiết bị có ngoài thị trường. Lí do ở đây là: thiết bị ngoài thị trường sử dụng HID, gửi trực tiếp các tham số tới hàm API của hệ thống để tạo ra một sự kiện bàn phím nên không cần phần mềm kèm theo. Tuy nhiên, nhóm hi vọng rằng: với việc hỗ trợ nhiều tác vụ, phần mềm đi kèm theo sẽ tạo nên một lợi thế so sánh với các sản phẩm khác.

6 Tài liệu tham khảo

1. USB Complete (3rd edition) by Jan Axelson.

2. Diễn đàn PicVietnam.com

3. Các thông tin khác từ Internet.