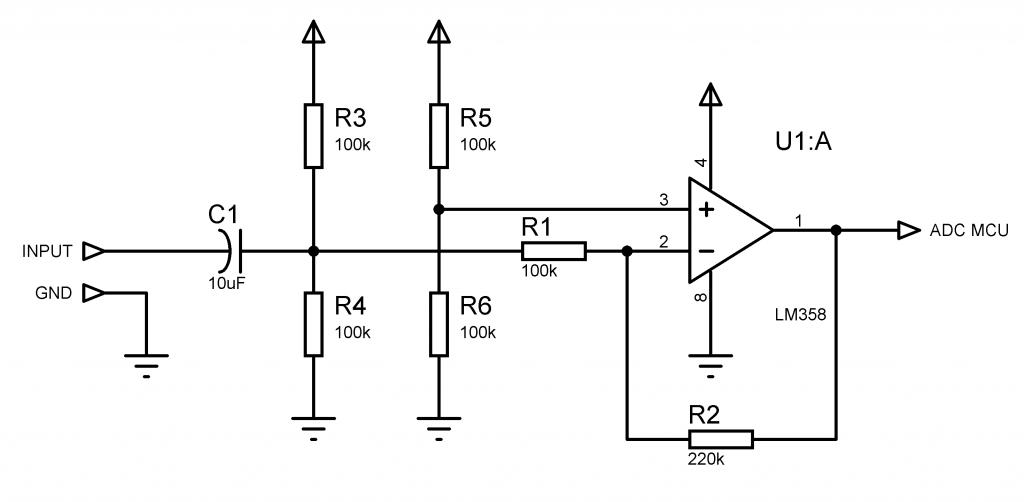
**Nháy theo nhạc**  
Tất cả các người làm LED Cube đều mún cube mình phải có chức năng nháy theo nhạc đúng ko. Điều đó cần yêu cầu phần cứng của các bạn phải có hỗ trợ đưa nhạc vào, có thể từ microphone, có thể từ sound jack từ PC. Một phần cứng đơn giản có thể miêu tả như hình.  
[](http://www.dientuvietnam.net/forums/filedata/fetch?id=1362811&d=1339812992)  
Input của bạn sau khi đưa vào được khuếch đại nhờ một opamp và sau đó đưa vào ADC của vi xử lý. Bạn có thể điều chỉnh R1 và R2 để thay đổi độ khuếch đại của mạch.  
  
Việc lấy mẫu bằng ADC trong ví dụ này là hàm **get\_adc()**, các bạn tự thay đổi theo từng lọai MCU.  
  
**Các loại nháy** **theo nhạc**:  
*Nháy theo biên độ*: Khi biên độ của âm càng lớn, hiệu ứng nháy càng mãnh liệt (hay ngược lại), đơn giản chỉ cần lấy kết quả ADC, càng cao (so vs mức cân bằng 1/2Vcc) thì càng nháy mạnh.  
*Nháy theo tần số*: Khi tần số âm càng cao thì càng nháy mạnh (hoặc ngược lại). Khi mỗi lần lấy mẫu ADC (so vs mức cân bằng), ta so sánh vs kết quả ngay trước đó, nếu trái dấu thì tăng 1 cho tần số.  
*Âm phổ (Audio Spectrum)*: Phân tích âm thanh thành tập hợp của nhiều sóng harmonic, thành phần sóng nào càng nhiều thì thành phần đó nháy càng mạnh (hay ngược lại). Chúng ta phải dùng biến đổi Fourier nhanh (FFT) để lấy đc kết quả.  
  
Audio Spectrum:  
Thuật toán FFT khá nặng và khá rắc rối, MCU khuyên sử dụng là PIC 18F4550, 16F877 hoặc tương đương. Bạn cần có bộ nhận âm thanh được mô tả ở trên (hoặc tương đương). Thuật tóan này hiện nay mình mới thử trên máy tính thôi.  
  
**Tần số lấy mẫu:**Nếu gọi tần số lấy mẫu là F (tần số lấy ADC), chúng ta chỉ tính toán đc chính xác đến tần số F/2. Và LED Cube có 8 cột để hiện thị, cộng thêm một cột bị bỏ ban đầu, tức 9 cột tương ứng vs tần số F/2 ở trên. Vậy mỗi cột tương đương vs một tần số F/18.  
Ví dụ: Tần số F = 7200 Hz, tức cột 0 (cột này bỏ do ko có nghĩa) có tần số từ 0 - 400 Hz, cột 1 là 400 - 800 Hz, cột 2 là 800 - 1200Hz và tiếp tục đến cột 8 là 3200 - 3600 Hz, cột 9 đến cột 17 là sự đối xứng của cột 0 - 8 nên chúng ta bỏ (do tính đối xứng của bến đổi Fourier)  
Và để được 18 cột, chúng ta phải lấy được 18 mẫu, tức tần số tính FFT một lần là 400 Hz.  
Tuy nhiên, chúng ta có thể tăng số lượng mẫu để tăng độ chính xác, giảm số lần tính FFT, nhưng mỗi lần tính sẽ nặng hơn. Số lượng mẫu của mình thường là 24 đến 32. Các bạn có thể lấy trung bình của các lần tính FFT để ra một giá trị chậm hơn. Ví dụ, vs tốc độ lấy mẫu là 24, tần số lấy FFT là 300 Hz, và mình muốn hiển thị trên LED Cube chỉ 20Hz thôi, thì chúng ta sẽ lấy trung bình 15 giá trị FFT để hiển thị.  
  
**Cơ số của FFT:**FFT là thuật toán phát triển dựa trên DFT (biến đổi Fourier rời rạc), bằng cách chia nhỏ các mẫu thành từng phần và tính từng phần đó, sau đó ghép lại. Số phần đó dựa trên thừa số của thuật toán. Cơ số chọn sao cho tổng của chúng là nhỏ nhất.  
Ví dụ: lấy số mẫu là 18, chúng ta có các ước số: 2x9 = 18, 3x6 = 18, nhưng 2+9 = 11, 3+6 = 9, vậy chúng ta chọn thừa số 3 và 6.  
Còn số mẫu là 24, chúng ta có 2x12 = 24, 3x8 = 24, 4x6 = 24, nhưng 2+12 = 14, 3+8 = 11, 4+6 = 10, vậy chúng ta chọn thừa số 4 và 6.  
Số mẫu chúng ta gọi là **N**, số hàng là **L**và số cột là **M = N / L**.  
  
**Xây dựng hàm FFT:**  
FFT là một thuật toán tính toán trên số phức, vì vậy chúng ta cần 2 biến global là res\_r chứa kết quả phần thực, res\_i chứa kết quả phần ảo.

Code:

int N = 24; //Số mẫu

int L = 4, M = 6;

float res\_i, res\_r;

float[6] dft1\_r, dft1\_i; //Kết quả DFT có số phần tử là M = 6

float[24] fft\_r, fft\_i; // Kết quả FFT có số phần tử N = 24

// Một số hàm tính toán số phức cần thiết

void mul\_comp(float r1, float i1, float r2, float i2) { // Hàm nhân phức

res\_r = r1 \* r2 - i1 \* i2;

res\_i = r1 \* i2 + r2 \* i1;

}

void exp\_comp(float angle) { // Đổi từ dạng e mũ phức sang số phức, bạn nên xác lập hàm này là separate, nếu inline nó sẽ rất nặng

res\_r = Cos(angle);

res\_i = Sin(angle);

}

// Biến đổi Fourier rời rạc, bạn có thể dùng trực tiếp DFT nếu số mẫu của bạn ít

void DFT(float[6] in\_r, float[6] in\_i, byte num) { //num là số mẫu để DFT

for (int k = 0; k < num; k++)

{

dft\_r[k] = 0; dft\_i[k] = 0;

for (int n = 0; n < num; n++)

{

exp\_comp(-2 \* PI \* k \* n / num); // Kết quả exp\_comp lưu trong res\_r và res\_i

mul\_comp(in\_r, in\_i, res\_r, res\_i); // Lấy tích rồi lưu lại vào res\_r và res\_i

dft\_r[k] += res\_r; // Cộng lại

dft\_i[k] += res\_i;

}

}

return outcom;

}

// Biến đổi Fourier nhanh

void FFT(float[24] in\_r, float[24] in\_i) {

float[24] F\_r, F\_i; // Mảng phụ

for (int l = 0; l < L; l++) {

float[6] comp\_r, comp\_i; // Biến đưa vào tính DFT lần 1

for (int m = 0; m < M; m++) {

comp\_r[m] = in\_r[l + m \* L]; // Tách mảng

comp\_i[m] = in\_i[l + m \* L];

}

DFT(comp\_r, comp\_i, M); // Tính DFT vs M = 6 mẫu

for (int q = 0; q < M; q++) {

exp\_comp(-2 \* PI \* l \* q / N);

mul\_comp(dft\_r[q], dft\_i[q], res\_r, res\_i);

F\_r[l + q \* L] = res\_r;

F\_i[l + q \* L] = res\_i;

}

}

for (int q = 0; q < M; q++) {

float[6] comp\_r, comp\_i; // Biến đưa vào tính DFT lần 2

for (int l = 0; l < L; l++) {

comp\_r[l] = F\_r[l + q \* L]; // Tách mảng

comp\_i[l] = F\_i[l + q \* L];

}

DFT(comp\_r, comp\_i, 4); //Tính DFT lần 2 vs L = 4 mẫu

for (int p = 0; p < L; p++) {

F\_r[p + q \* L] = dft\_r[p];

F\_i[p + q \* L] = dft\_i[p];

}

}

for (int p = 0; p < L; p++) { // Biến đổi mảng

for (int q = 0; q < M; q++) {

fft\_r[M \* p + q] = F\_r[p + q \* L];

fft\_i[M \* p + q] = F\_i[p + q \* L];

}

}

}