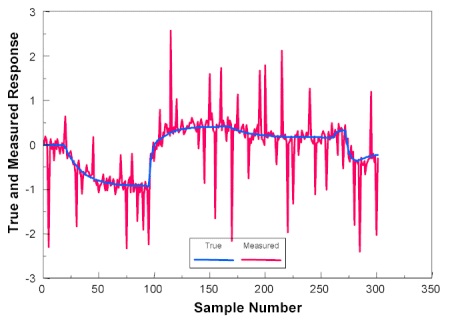
<http://icviet.vn/bai-hoc/thiet-ke-bo-loc-thong-thap-thong-cao-va-kalman-voi-stm32f4.html>

**1. Giới thiệu khái quát về bộ lọc (Filter)**

       Bộ lọc được hiểu nôm na là một quá trình lọc những dữ liệu không cần thiết, loại bỏ đi những dữ liệu này và chỉ giữ lại những thứ mà bạn quan tâm, mong muốn. Đối với lĩnh vực điện-điện tử có lẽ sẽ rất quen thuộc chẳng hạn như sử dụng tụ với mục đích lọc nhiễu. Lọc nhiễu không có nghĩa là bạn loại bỏ hoàn toàn nhiễu, bạn chỉ có thể giảm thiểu nhiễu gây nên. Không phải bộ lọc đều tốt trong mọi tình huống, bạn chỉ sử dụng nếu mô hình của bạn cần đến nó do thực chất bộ lọc sẽ làm trễ tín hiệu.

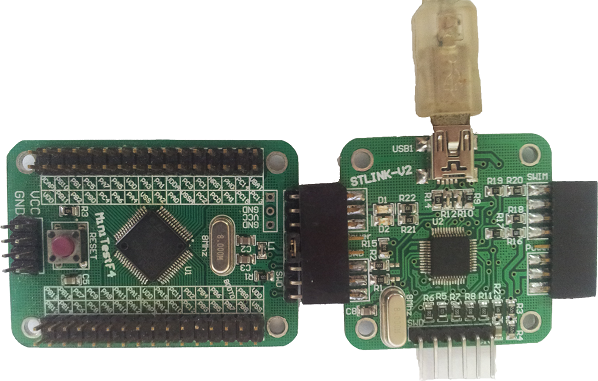


       Ngoài khả năng lọc nhiễu bộ lọc nâng cao như kalman còn có khả năng ước lượng và dự đoán mẫu.

       Ứng dụng của bộ lọc rất đa dạng trong các hệ thống xử lý tín hiệu, xử lý ảnh, phân tích chuyển động,…

**2. Kết nối phần cứng**

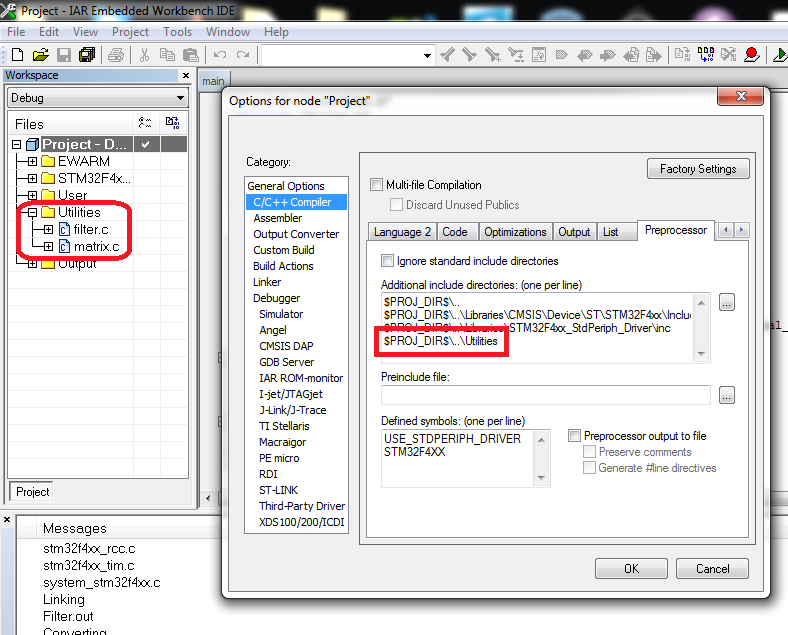
       Bạn chỉ sử dụng MiniTestF4 + STlink-V2 mà không cần kết nối với bất kỳ thiết bị nào.



**3. Lập trình cơ bản**

       Trong bài viết hôm nay tôi sẽ giới thiệu với bạn về các bộ lọc cơ bản về lý thuyết và các bước thiết kế bộ lọc. Phần lý thuyết về các bộ lọc sẽ được trình bày song song với các bước lập trình.

       Tôi đã đính kèm theo project bộ thư viện filter và matrix bạn cần thêm vào như hình bên dưới.



       Trong chương trình main tôi sử dụng hai ngoại vi là TIM2 dùng để đo tổng thời gian thực thi của các bộ lọc và system tick để định thời gian mỗi 1ms sẽ thực hiện 1 lần mục đích làm cho thời gian xử lý bộ lọc là hằng số có tần số cố định.

|  |
| --- |
| 18 int main(void)  19 {  20 TIM2\_Configuration();  21 SysTick\_Config(SystemCoreClock / 999);  22 while (1)  23 {  24  25 }  26 } |

       Từ dòng 28 -> 51 là hàm Threads() hàm này sẽ được system tick gọi trong ngắt do đó đều đặn cứ mỗi 1ms sẽ nhảy vào thực hiện hàm này một lần.

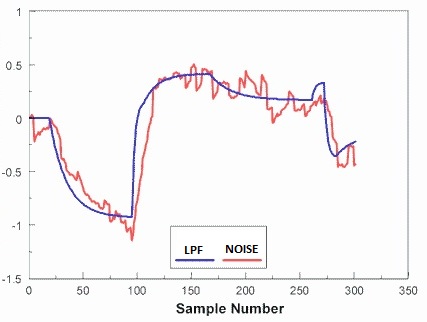
|  |
| --- |
| 28 void Threads(void)  29 {  30 int i;  31 static float x=0;  32 lasttime = TIM\_GetCounter(TIM2);  33  34 \_sin = sin(x)\*1000;  35 x +=2\*PI/1000;  36 \_random = (float)rand()/1000000;  37  38 random\_LPF\_HPF = \_random + \_sin;  39 signal\_LPF = LPF(random\_LPF\_HPF,1,1000);  40 signal\_HPF = HPF(random\_LPF\_HPF,10,1000);  41  42 random\_kalman = (float)rand()/1000000;  43 signal\_kalman = kalman\_single(random\_kalman, 500, 10);  44  45 for(i=0;i<3;i++)  46 random[i] = (float)rand()/1000000;  47 kalman((float\*)random,(float\*)signal,500,10);  48  49 nowtime = TIM\_GetCounter(TIM2);  50 lengthtime = (float)(nowtime - lasttime)/84; // us  51 } |

       Trong hàm này bạn sẽ thấy nhiều hàm rand(), hàm rand() sẽ trả về một số ngẫu nhiên với kiểu giá trị là interger. Các biến random\_LPF\_HPF,random\_kalman,random sẽ lấy giá trị ngẫu nhiên này xem như nhiễu và đem vào các bộ lọc để lọc nhiễu. Giá trị trả về đã được lọc nhiễu với các tên biến là signal. Các hàm thực hiện lọc sẽ được phân tích kỹ bên dưới cho riêng từng loại hàm lọc.

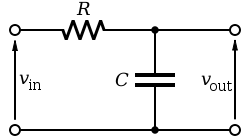
       Dòng 32, 49 và 50 thực hiện việc đọc giá trị counter timer để mục đích để đo khoảng thời gian thực hiện từ dòng 34 -> 47.

**4. Low pass filter (lọc thông thấp)**

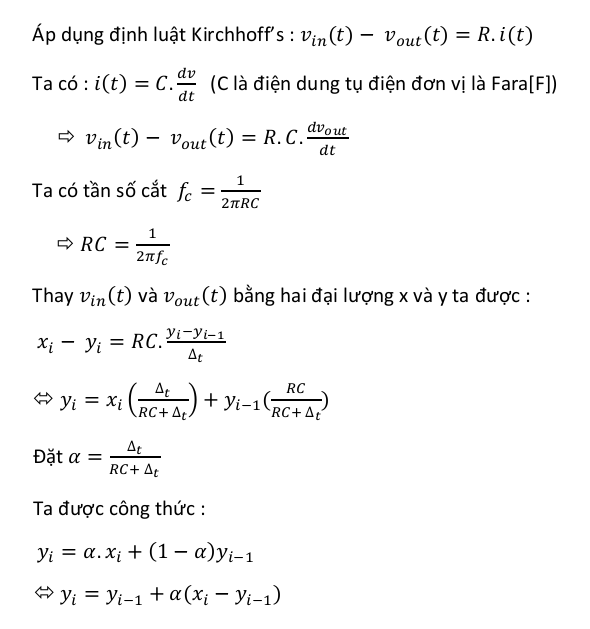
       Bộ lọc thông thấp là bộ lọc chỉ cho phép tín hiệu hoạt động từ 0 Hz -> tần số cắt (cutoff) của nó, các tín hiệu tần số cao sẽ được lọc đi.



       Bộ lọc thông thấp được phát triển dựa trên lý thuyết của bộ lọc RC



       Do đó ta có các công thức sau :



       Trong file filter.c hàm LPF() có 4 đối số 3 đầu vào và 1 đối số trả về kết quả sau lọc.

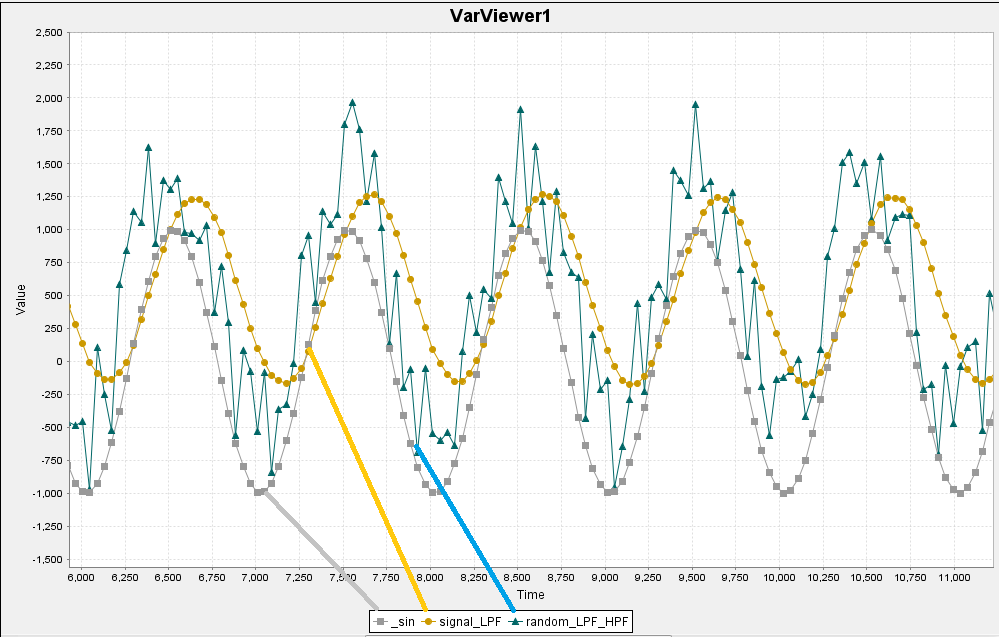
Trong đó:

-          x là biến đầu vào đại diện cho tín hiệu nhiễu cần được lọc

-          CUTOFF là tần số cắt đơn vị Hz

-          SAMPLE\_RATE là tần số lấy mẫu của hàm LPF() đơn vị Hz

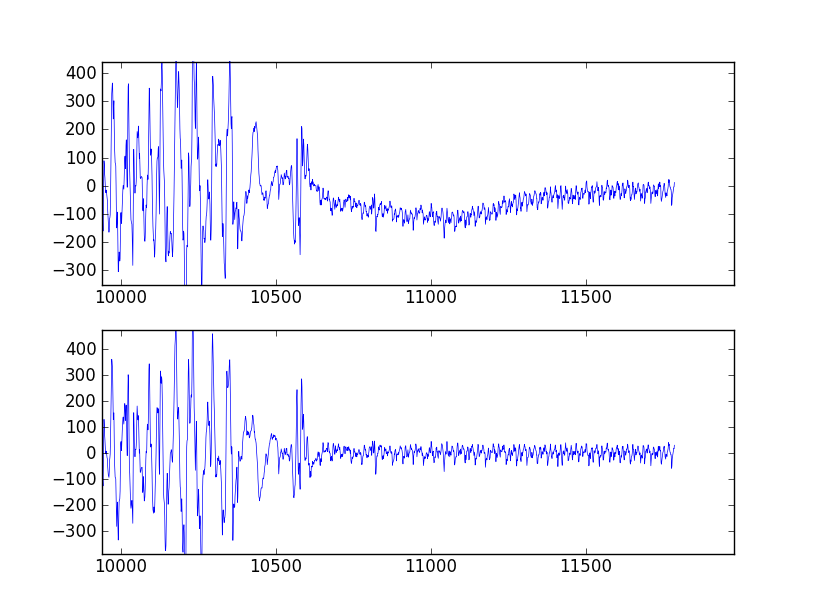
       Quay lại hàm Threads() trong main.c đối số thứ nhất truyền vào hàm LPF() là random\_LPF\_HPF tín hiệu này gồm \_sin là một tín hiệu sin với tần số dao động là 1 Hz và nhiễu ngẫu nhiên \_random. Lý thuyết bộ lọc thông thấp đã trình bày bên trên tín hiệu tần số cao sẽ bị lọc đi và chỉ giữ lại tín hiệu tần số thấp trong trường hợp của ta thì tín hiệu sin tần số 1 Hz sẽ được giữ lại nhưng sẽ trễ pha hơn tần số sin thực tế. Do tần số của tín hiệu mong muốn là 1 Hz nên đối số CUTOFF  tần số cắt tôi truyền vào là 1 và tần số hoạt động của hàm Threads() là 1 Khz nên đối số SAMPLE\_RATE cần truyền vào là 1000.



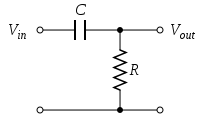
       Hình bên trên là 3 tín hiệu được tôi trích ra từ đồ thị quan sát STMStudio, đặc tính của các tín hiệu vào và ra của hàm LPF().

**5. High pass filter (lọc thông cao)**

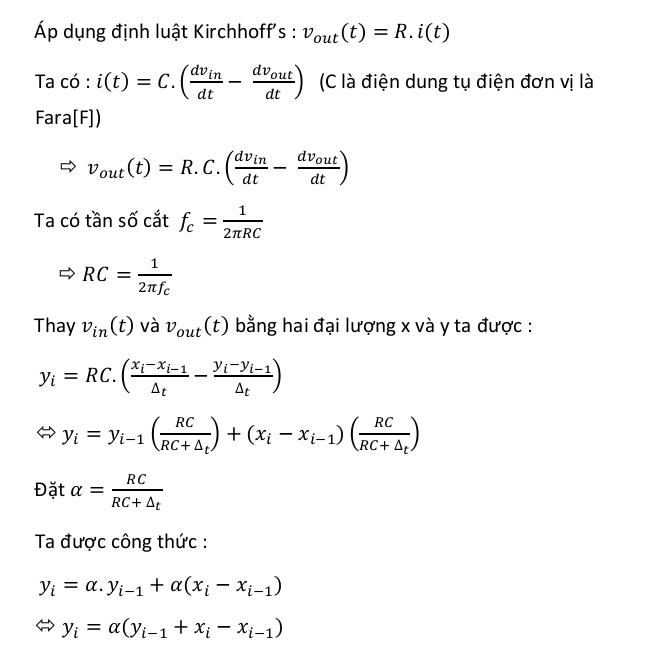
       Bộ lọc thông cao là bộ lọc chỉ cho phép tín hiệu hoạt động từ tần số cắt của nó đến vô cùng các tín hiệu tần số thấp nhỏ hơn tần số cắt sẽ được lọc đi.



       Tương tự như bộ lọc thông thấp, bộ lọc thông cao cũng được phát triển dựa trên lý thuyết của mạch lọc RC.

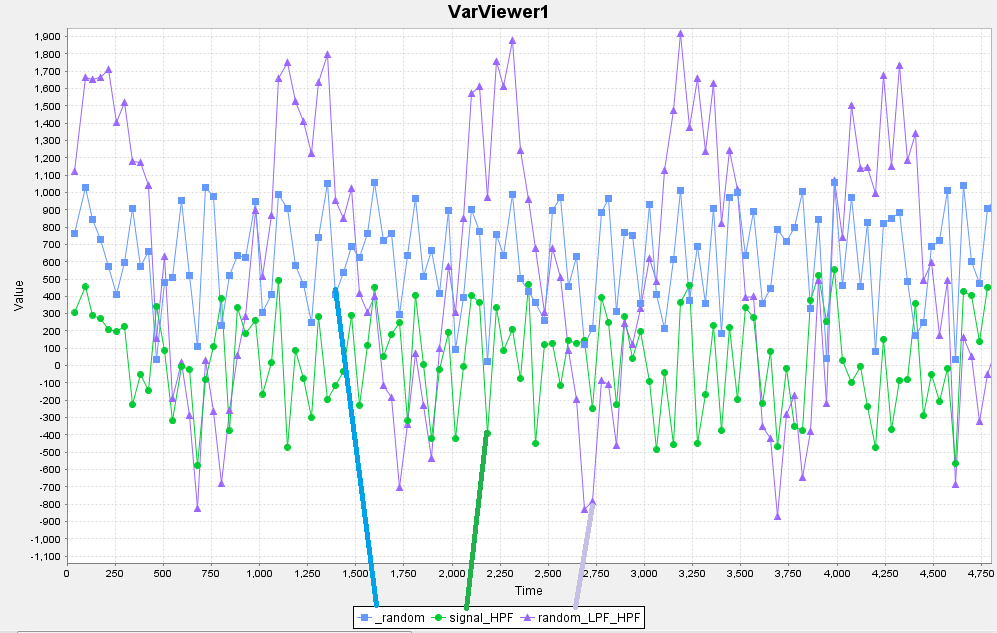


       Do đó ta có các công thức sau :



       Các đối số truyền vào hàm HPF() tương tự như hàm LPF().

       Trong hàm Threads() đối số đầu tiên truyền vào hàm HPF() là random\_LPF\_HPF, như đã được giải thích bên trên tín hiệu này gồm \_sin là một tín hiệu sin với tần số dao động là 1 Hz và nhiễu ngẫu nhiên \_random. Bộ lọc thông cao sẽ loại bỏ đi tín hiệu tần số thấp là \_sin và chỉ giữ lại tín hiệu tần số cao là tín hiệu đầu ra giống với \_random, điều này có vẻ ngược với bộ lọc thông thấp do định nghĩa của lọc thông cao và thông thấp hoàn toàn trái ngược nhau. Tôi chọn tần số cắt CUTOFF =10Hz do tín hiệu tôi mong muốn lọc bỏ ở tần số 1Hz, những tần số cao hơn 10Hz sẽ được giữ lại. Cách tốt nhất để chọn thông số này tốt là bạn cần nắm rõ tần số nhiễu và tần số mong muốn, hoặc nếu không nắm rõ bạn có thể tinh chỉnh thay thế giá trị CUTOFF cho đến khi nào bạn quan sát thấy ổn định nhất.

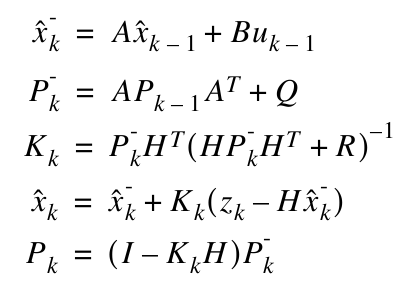


       Như đồ thị bên trên tín hiệu đầu ra signal\_HPF đã được loại bỏ tần số thấp nên hình dáng nó giống với tín hiệu \_random, điều này là điều mà ta mong muốn nhưng pha lại trễ hơn \_random.

**6. Kalman filter**

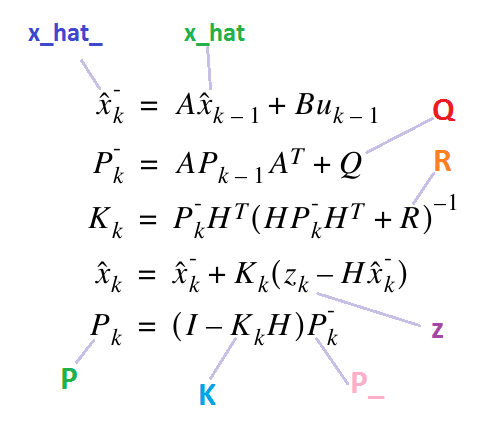
       Do trong khuôn khổ bài viết chỉ tóm tắt lại khai quát cơ bản và ứng dụng về bộ lọc kalman nên sẽ không đề cập đến bộ lọc kalman mở rộng, phương pháp chọn kích thước ma trận trạng thái hay các vấn đề năng cao hơn.

       Công thức chuẩn của bộ lọc kalman:



       Trong thư viện kalman.c tôi sẽ lần lược giải thích hàm kalman\_single() gồm 1 đầu vào – 1 đầu ra và hàm kalman() 3 vào 3 ra.

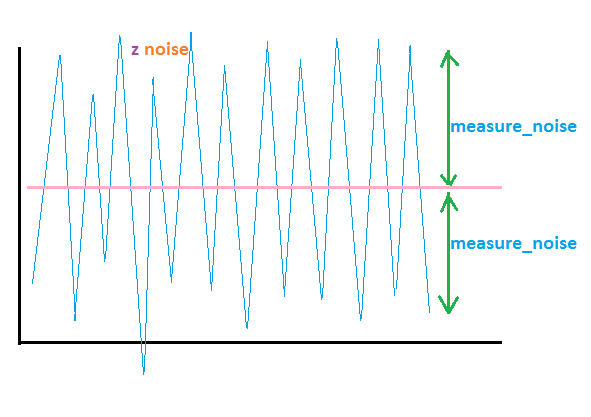
       Trong hàm kalman\_single() các biến được đặt theo công thức trên như sau:



       Đầu tiên cần xác định thông số đầu vào cần truyền vào đó là

z : tín hiệu nhiễu cần được lọc

measure\_noise : phương sai nhiễu đo lường. Có thể lấy giá trị này bằng việc xác định hình dạng nhiễu z và tính toán phương sai.



Xác định phương sai nhiễu đo lường có nghĩa là xác định chiều cao trung bình so với trục màu hồng như hình trên.

process\_noise : phương sai nhiễu quá trình được xác định bằng việc thử các giá trị phù hợp với hệ thống của bạn, rất khó giải thích nhiễu quá trình trong một trường hợp cụ thể, cách tốt nhất bạn phải thay thế và quan sát đồ thị cho đến khi bạn chọn được hệ số thích hợp để truyền cho hàm kalman làm việc.

       Bước tiếp theo cần xác định ma trận R và Q, nhưng do hàm chỉ gồm 1 in – 1 out nên R và Q là biến.

Q : được định nghĩa là ma trận hiệp biến phương sai nhiễu hệ thống theo phân bố gauss, Q không phải ma trận nên ta không xét phân bố gauss và không nhân hiệp biến nên chỉ cần bình phương measure\_noise để tính Q.

R : tương tự như tính Q , R = process\_noise^2

       Hàm kalman\_single() đã được rút gọn công thức chuẩn kalman do chỉ có một đầu vào và một đầu ra. Ta xét các bước

-          Bước 1: x\_hat\_ = x\_hat;

Ở hàm 1 đầu vào 1 đầu ra thì ma trận sẽ không cần thiết nên cần loại bỏ hết ma trận đi ngầm hiểu nhân với 1. Ta cũng không sử dụng biến u do trong hệ thống chỉ cần lọc nhiễu, đầu vào đầu ra có trị trung bình bằng nhau, khi thêm u vào hai giá trị đầu vào đầu ra sẽ cách nhau một khoảng trên trục y, và điều này là không cần thiết.

-          Bước 2 : P\_ = P + Q;

-          Bước 3 : K = P\_/(P\_ + R);

-          Bước 4 : x\_hat = x\_hat\_ + K\*(z - x\_hat\_);

-          Bước 5 : P = ( 1 - K)\*P\_ ;

Các ma trận như A, H đã được loại bỏ còn các ma trận tính toán P\_, P, Q thì đổi thành biến.

Lưu ý hai biến x\_hat,P luôn nhớ giá trị mỗi khi thoát khỏi hàm nên khi khai báo hàm tôi đã thêm từ khóa static

       Trước khi thoát khỏi hàm,  cần truyền đầu ra x\_hat  đã được lọc nhiễu.

***COMING SOON ...***

**7. Quan sát bằng STMStudio**

       Sau khi đã tìm hiểu kỹ lý thuyết bên trên, ta tiến hành quan sát kết quả bằng đồ thị dựa trên phần mềm STMStudio để có cái nhìn trực quan nhất

