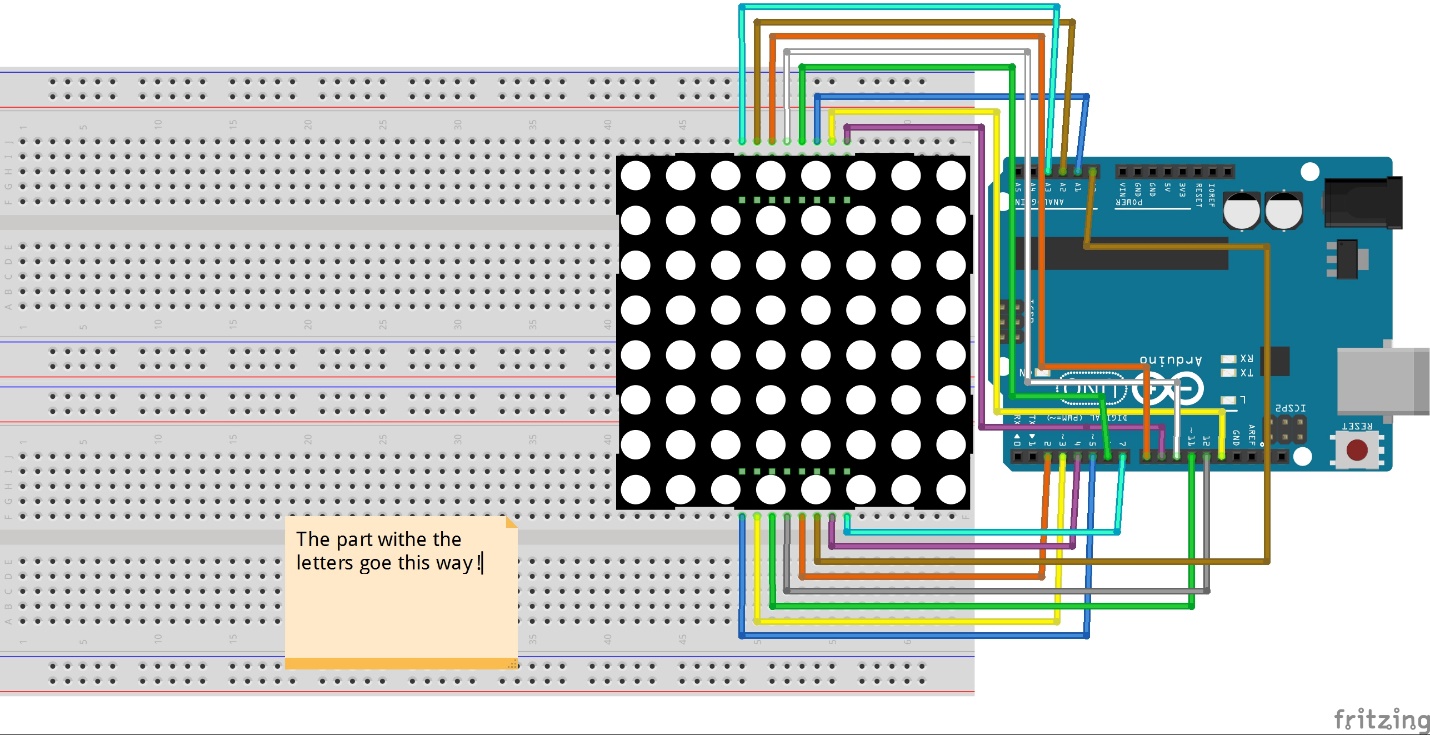
Running Charater On Led Matrix 8\*8

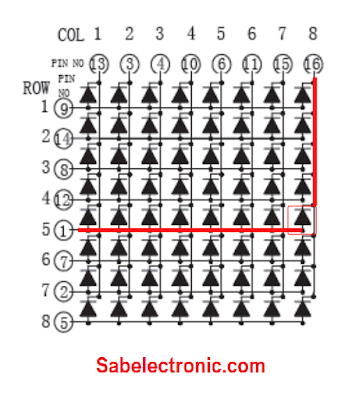
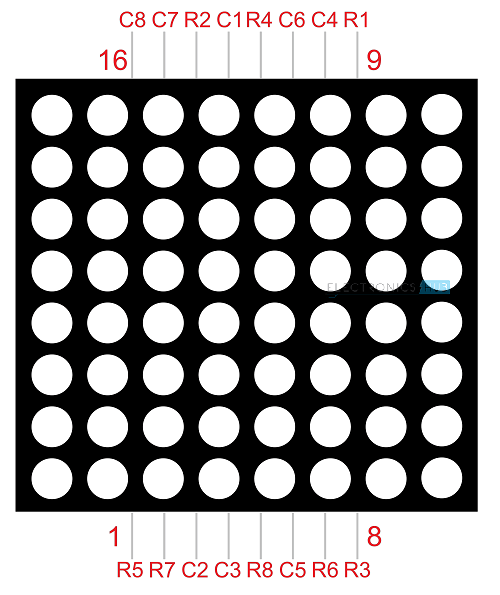
Hardware Required

* 8 330 ohm resistors
* Led Matrix anot 8\*8
* Arduino UNO R3
* breadboard

Circuit



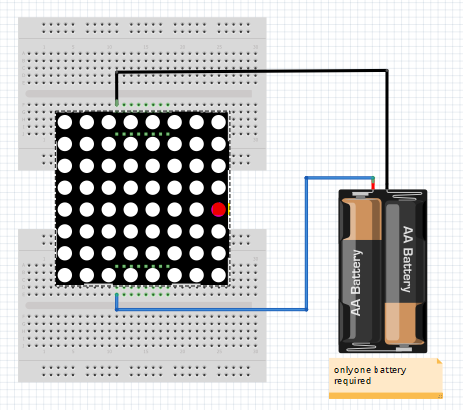
Overview On Led matrix anot 8\*8:

+ Row Pin: 1 – 8

+ Column Pin: 9 – 16

Example:   
 + Row pin no 1(+), Column Pin no 16(-) => Led in 5th row and 8th column is on



How to make Character, String run on led matrix

1. How to make Character display on led matrix.

|  |  |
| --- | --- |
|  | B00000000,  B00100100,  B00100100,  B00100100,  B00000000,  B01000010,  B00111100,  B00000000 |

* 8 hàng dùng 8 dãy bit để biểu diễn, mỗi dãy bit có độ dài 8 để biểu diễn trạng thái cái cột.(1 là on, 0 là off)
* Vùng giá trị của các dãy bit:
  + B11111111 (Bin value) = 255(Dec) = 0xFF(Hex)
  + B00000000 (Bin value) = 0 (Dec) = 0x00(Hex)
* Để lưu 1 kí tự chúng ta dùng 1 mảng 1 chiều lưu 8 dãy bit có độ dài bằng 8.

* Led matrix generator : [**link**](http://embed.plnkr.co/3VUsekP3jC5xwSIQDVHx/previewhttp:/embed.plnkr.co/3VUsekP3jC5xwSIQDVHx/preview)

1. How to make Character run on led matrix

|  |  |
| --- | --- |
|  | * Để dịch kí tự sang trái 1 cột, ta dịch các dãy bit (có 8 dãy) sang trái 1 đơn vị.   ⬄ ( để dịch kí tự sang phảil) |

1. How to make String run on led matrix

* Giả sử String cần chạy: “AB”
* Có 2 cách làm trong phần này
* Cách 1:
  + Nếu chúng ta dịch từng kí tự như ở phần trên thì sẽ có 1 khuyết điểm đó là sẽ ko hiện được phần chuyển giao giữa 2 kí tự trên led matrix( tức 1 phần của kí tự A và 1 phần của kí tự B ko đồng thời xuất hiện được trong lúc đang chạy, mà kí tự A phải chạy hết mới đến kí tự B)
* Cách 2:
  + Chúng ta sẽ hiện được phần chuyển giao

|  |  |
| --- | --- |
|  | A[8] = {  B00000000,  B00111100,  B01100110,  B01100110,  B01111110,  B01100110,  B01100110,  B01100110  } |

|  |  |
| --- | --- |
|  | B[8] = {  B01111000,  B01001000,  B01001000,  B01110000,  B01001000,  B01000100,  B01000100,  B01111100  } |

* Để in xâu “AB” chạy mà có được phần chuyển giao kí tự:

Ta sẽ tạo ra 1 mảng String mới là Root[8] sao cho

* + Root[i] = A[i] nối B[i] ( ko thể dùng toán tử “+” ở đây vì các giá trị thực tế là Binvalue, sau khi cộng sẽ là cộng trên hệ Dec rồi chuyển về Bin)
  + Ta sẽ chuyển A[i], B[i] sang Bin String rồi mới “+” để nối xâu.
  + Eg: Root[0] = A[0] nối B[0] = “00000000” + “01111000” = “0000000001111000”
  + Ở mảng xâu Root, lúc này Root[i] có length = 8\*số kí tự trong string.
  + Ta sẽ cắt liên tiếp các Block có chiều ngang = 8 từ Root để hiện thị các kí tự.
    - => Vậy là sẽ hiển thị được dãy chữ đang chạy mà có phần chuyển giao.

Code

* Ở đây sẽ là code của cách hiển thị được chuyển giao
* Github(Better view):
* Raw:

#include <binary.h>

#include <Arduino.h>

#define ROW\_1 2

#define ROW\_2 3

#define ROW\_3 4

#define ROW\_4 5

#define ROW\_5 6

#define ROW\_6 7

#define ROW\_7 8

#define ROW\_8 9

#define COL\_1 10

#define COL\_2 11

#define COL\_3 12

#define COL\_4 13

#define COL\_5 A0

#define COL\_6 A1

#define COL\_7 A2

#define COL\_8 A3

const byte rows[] = {

    ROW\_1, ROW\_2, ROW\_3, ROW\_4, ROW\_5, ROW\_6, ROW\_7, ROW\_8};

const byte col[] = {

    COL\_1, COL\_2, COL\_3, COL\_4, COL\_5, COL\_6, COL\_7, COL\_8};

const unsigned int characterHEX[][8] = {

    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},                                         //blank

    {0x3C, 0x66, 0x66, 0x6E, 0x76, 0x66, 0x66, 0x3C},                                         //0

    {0x18, 0x38, 0x58, 0x18, 0x18, 0x18, 0x18, 0x7E},                                         //1

    {0x3C, 0x66, 0x66, 0x0C, 0x18, 0x30, 0x7E, 0x7E},                                         //2

    {0x7E, 0x0C, 0x18, 0x3C, 0x06, 0x06, 0x46, 0x3C},                                         //3

    {0x0C, 0x18, 0x30, 0x6C, 0x6C, 0x7E, 0x0C, 0x0C},                                         //4

    {0x7E, 0x60, 0x60, 0x7C, 0x06, 0x06, 0x46, 0x3C},                                         //5

    {0x04, 0x08, 0x10, 0x38, 0x6C, 0x66, 0x66, 0x3C},                                         //6

    {0x7E, 0x46, 0x0C, 0x18, 0x18, 0x18, 0x18, 0x18},                                         //7

    {0x3C, 0x66, 0x66, 0x3C, 0x66, 0x66, 0x66, 0x3C},                                         //8

    {0x3C, 0x66, 0x66, 0x36, 0x1C, 0x08, 0x10, 0x20},                                         //9

    {B00000000, B00111100, B01100110, B01100110, B01111110, B01100110, B01100110, B01100110}, //A

    {B01111000, B01001000, B01001000, B01110000, B01001000, B01000100, B01000100, B01111100}, //B

    {B00000000, B00011110, B00100000, B01000000, B01000000, B01000000, B00100000, B00011110}, //C

    {B00000000, B00111000, B00100100, B00100010, B00100010, B00100100, B00111000, B00000000}, //D

    {B00000000, B00111100, B00100000, B00111000, B00100000, B00100000, B00111100, B00000000}, //E

    {B00000000, B00111100, B00100000, B00111000, B00100000, B00100000, B00100000, B00000000}, //F

    {B00000000, B00111110, B00100000, B00100000, B00101110, B00100010, B00111110, B00000000}, //G

    {B00000000, B00100100, B00100100, B00111100, B00100100, B00100100, B00100100, B00000000}, //H

    {B00000000, B00111000, B00010000, B00010000, B00010000, B00010000, B00111000, B00000000}, //I

    {B00000000, B00011100, B00001000, B00001000, B00001000, B00101000, B00111000, B00000000}, //J

    {B00000000, B00100100, B00101000, B00110000, B00101000, B00100100, B00100100, B00000000}, //K

    {B00000000, B00100000, B00100000, B00100000, B00100000, B00100000, B00111100, B00000000}, //L

    {B00000000, B00000000, B01000100, B10101010, B10010010, B10000010, B10000010, B00000000}, //M

    {B00000000, B00100010, B00110010, B00101010, B00100110, B00100010, B00000000, B00000000}, //N

    {B00000000, B00111100, B01000010, B01000010, B01000010, B01000010, B00111100, B00000000}, //O

    {B00000000, B00111000, B00100100, B00100100, B00111000, B00100000, B00100000, B00000000}, //P

    {B00000000, B00111100, B01000010, B01000010, B01000010, B01000110, B00111110, B00000001}, //Q

    {B00000000, B00111000, B00100100, B00100100, B00111000, B00100100, B00100100, B00000000}, //R

    {B00000000, B00111100, B00100000, B00111100, B00000100, B00000100, B00111100, B00000000}, //S

    {B00000000, B01111100, B00010000, B00010000, B00010000, B00010000, B00010000, B00000000}, //T

    {B00000000, B01000010, B01000010, B01000010, B01000010, B00100100, B00011000, B00000000}, //U

    {B00000000, B00100010, B00100010, B00100010, B00010100, B00010100, B00001000, B00000000}, //V

    {B00000000, B10000010, B10010010, B01010100, B01010100, B00101000, B00000000, B00000000}, //W

    {B00000000, B01000010, B00100100, B00011000, B00011000, B00100100, B01000010, B00000000}, //X

    {B00000000, B01000100, B00101000, B00010000, B00010000, B00010000, B00010000, B00000000}, //Y

    {B00000000, B00111100, B00000100, B00001000, B00010000, B00100000, B00111100, B00000000}  //Z

};

// Mảng myChar ở đây để ánh xạ từ Char sang các dãy bit quét.

// nên index các kí tự của mảng char ở đây phải được sắp xếp theo thứ tự tương ứng với index characterHEX[][8].

const char myChar[] = {

    ' ',

    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',

    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',

    'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',

    'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',

    'Y', 'Z'

};

// Trả về position của char a trong mảng myChar[]

// ở đây return 0; tức là ko tìm thấy, default: trả về blank character luôn( do kiểu unsigned int ko có giá trị âm)

unsigned int getPos(char a) {

  for (int i = 0; i < sizeof(myChar); ++i) {

    if (myChar[i] == a) {

      return i;

    }

  }

  return 0;

}

boolean hasBuildRoot = false;

float timeCount = 0;

String Root[8];

void setup() {

  Serial.begin(9600);

  for (byte i = 2; i <= 13; i++) {

    pinMode(i, OUTPUT);

  }

  pinMode(A0, OUTPUT);

  pinMode(A1, OUTPUT);

  pinMode(A2, OUTPUT);

  pinMode(A3, OUTPUT);

}

void loop() {

  // Input String, Lưu ý: ở đây chỉ nhận các kí tự in hoa

  String s = "HAI";

  // Build Root từ input String s

  // Chỉ cần thực hiện việc build 1 lần

  if(!hasBuildRoot) {

    for(int i = 0; i < 8; ++i) {

      Root[i] += "00000000";

    }

    for(int i = 0 ; i < s.length(); ++i) {

      int pos = getPos(s[i]);

      unsigned int temp[8];

      for(int j = 0; j < 8; ++j) {

        temp[j] = characterHEX[pos][j];

        String sTemp = String(temp[j],BIN);

        while(sTemp.length() < 8) {

          sTemp = "0" + sTemp;

        }

        Root[j] += sTemp;

      }

    }

    for(int i = 0; i < 8; ++i) {

      Root[i] += "00000000";

    }

    hasBuildRoot = true;

  }

  // Cắt thành các Block liên tục và hiển thị

  int z = 0;

  while( z++ < s.length() \* 9 - 1) {

    String Block[8];

    for(int i = 0; i < 8; ++i) {

      Block[i] = Root[i].substring(z,z+8);

    }

    unsigned int OutPutBlock[8];

    for(int i = 0; i < 8; ++i) {

      OutPutBlock[i] = StringToInt(Block[i], 2);

    }

    OffSet(OutPutBlock,20);

  }

}

// Quét liên tục trong khoảng thời gian time để hiển thị kí tự

void OffSet(unsigned int buffer[],int time) {

  while(timeCount++ < time) {

        delay(5);

        drawScreenByBuffer(buffer);

  }

  timeCount = 0;

}

// Quét 1 lần

void drawScreenByBuffer(const unsigned int buffer2[]) {

  // Turn on each row in series

  for (byte i = 0; i < 8; i++) // count next row

  {

    digitalWrite(rows[i], HIGH); //initiate whole row

    for (byte a = 0; a < 8; a++) // count next row

    {

      // if You set (buffer2[i] >> a) then You will have positive

      digitalWrite(col[a], (~buffer2[i] >> a) & 0x01); // initiate whole column

      delayMicroseconds(100); // uncoment deley for diferent speed of display

      //delayMicroseconds(1000);

      //delay(10);

      //delay(100);

      digitalWrite(col[a], 1); // reset whole column

    }

    digitalWrite(rows[i], LOW); // reset whole row

                                // otherwise last row will intersect with next row

  }

}

// Chuyển String s biểu diễn trong cơ số dạng xBase về DecValue

unsigned int StringToInt(String s, unsigned int xBase) {

  unsigned int res = 0;

  unsigned int base = 1;

  for(int i = s.length() - 1; i >= 0; --i) {

      res = res + (s.charAt(i) - '0') \* base;

      base \*= xBase;

  }

  return res;

}