Hướng dẫn sử dụng thư viện Trí uẩn

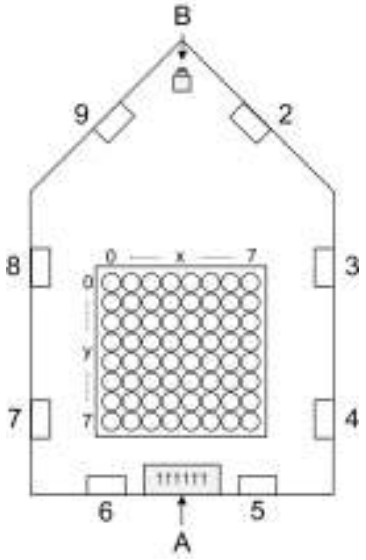
1. Các khối Trí Uẩn

Một số ký hiệu trong hình vẽ của các khối Trí Uẩn

Ký hiệu	Ý nghĩa
A	Cổng giao tiếp máy tính
B	Nút bấm cấp nguồn
Số từ 2 tới 9	Các cổng giao tiếp dữ liệu tương ứng được kết nối tới chân từ 2 tới 9 của Arduino
Số 10	Bộ cảm biến ánh sáng được nối tới chân số 10 của Arduino
Mặt trên	Mặt đáy của khối trí uẩn, mặt không có cổng giao tiếp dữ liệu được gắn vào
Mặt dưới	Mặt đáy của khối trí uẩn, mặt có cổng giao tiếp dữ liệu được gắn vào

Ở hình vẽ của các khối Trí Uẩn 1, 2, 3, 4, hình vẽ "Mặt trên" thể hiện vị trí các cổng giao tiếp dữ liệu khi nhìn khối Trí Uẩn từ mặt trên; còn hình vẽ "Mặt dưới" thể hiện vị trí các cổng giao tiếp dữ liệu khi nhìn khối Trí Uẩn từ mặt dưới của khối đó.

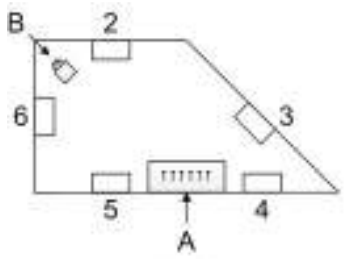
1.1. Khối Trí Uẩn 0



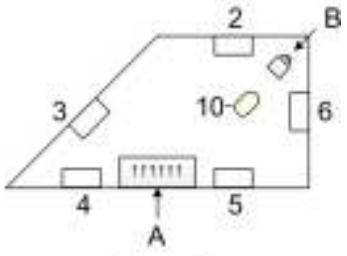
Khối Trí Uẩn 0 gồm có 8 cổng thu phát dữ liệu được kết nối tới các chân từ 2 tới 9 của vi điều khiển Arduino. Trên khối Trí Uẩn này có một đèn LED ma trận 8x8 được thiết kế để hiển thị một hình ảnh bất kỳ lên trên nó.

Vi điều khiển Arduino sử dụng trong khối Trí Uẩn 0 thuộc loại **Arduino Uno**, do đó, khi trước khi lập trình cho khối này bạn cần lựa chọn board là **Arduino Uno** để có thể nạp được chương trình vào đó.

1.2. Khối Trí Uẩn 1



Mặt trên

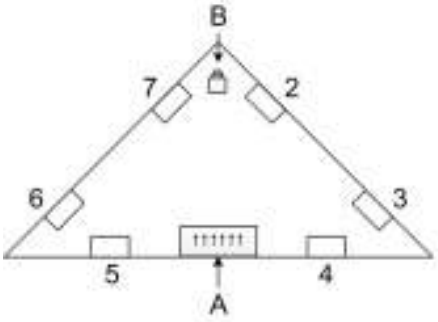


Mặt dưới

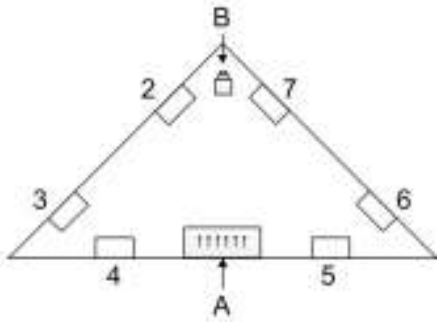
Khối Trí Uẩn 1 gồm có 5 cổng thu phát dữ liệu được kết nối với vi điều khiển Arduino từ chân 2 tới chân 6 và được bố trí như hình vẽ. Ngoài 5 cổng giao tiếp dữ liệu thì ở mặt dưới của khối Trí Uẩn 1 còn được gắn một bộ cảm biến ánh sáng và bộ cảm biến ánh sáng này được kết nối với chân số 10 của Arduino. Trong quá trình lập trình, có thể sử dụng bộ cảm biến này để xác định khối Trí Uẩn đang ngửa lên hay là úp xuống.

Vi điều khiển Arduino sử dụng trong khối Trí Uẩn 1 là **Arduino Atmega8**.

1.3. Khối Trí Uẩn 2



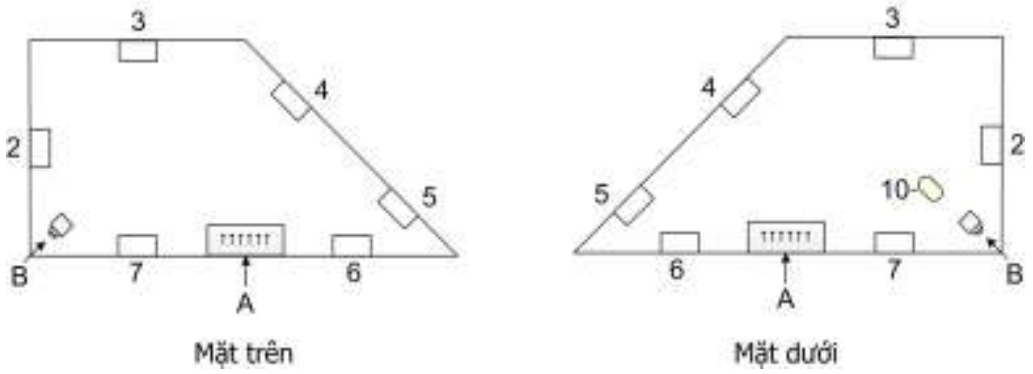
Mặt trên



Mặt dưới

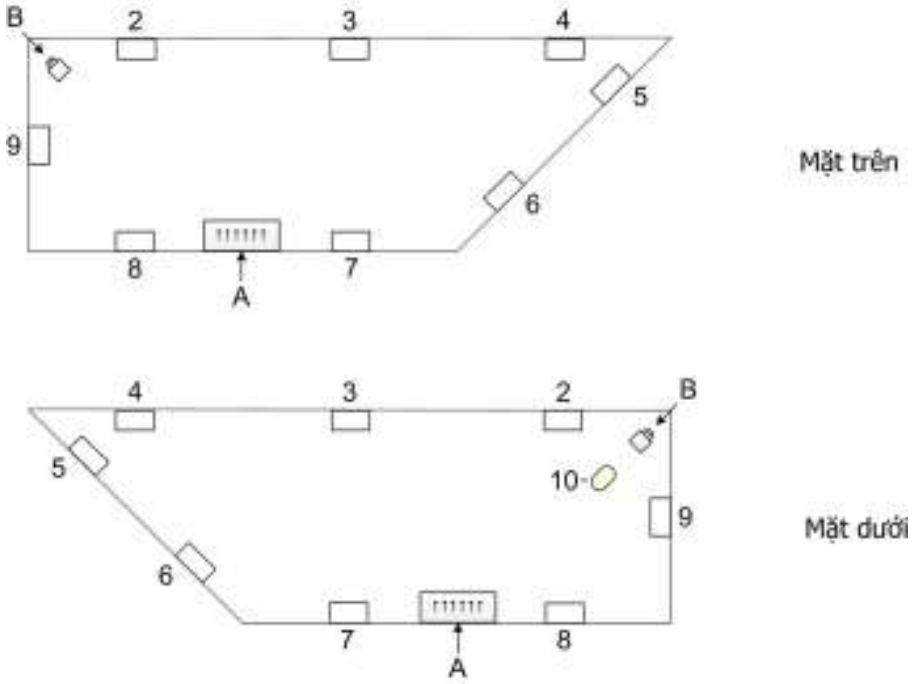
Khối Trí Uẩn 2 gồm có 6 cổng giao tiếp dữ liệu được kết nối với các chân từ 2 tới 7 của vi điều khiển **Arduino Atmega8**. Thứ tự của các cổng giao tiếp trong khối Trí Uẩn 2 được thể hiện ở hình vẽ trên.

1.4. Khối Trí Uẩn 3



Khối Trí Uẩn 3 gồm có 6 cổng giao tiếp dữ liệu được kết nối với các chân từ 2 tới 7 của vi điều khiển **Arduino Atmega8**. Ở mặt dưới của khối Trí Uẩn này cũng có một bộ cảm biến ánh sáng được kết nối với chân số 10 của Arduino.

1.5. Khối Trí Uẩn 4



Khối Trí Uẩn 4 gồm có 8 cổng giao tiếp dữ liệu được kết nối với các chân từ 2 tới 9 của vi điều khiển **Arduino Atmega8**. Một bộ cảm biến ánh sáng cũng được thiết kế ở mặt dưới của khối này và nó được kết nối với chân số 10 của Arduino.

2. Thư viện thVLC.

Thư viện thVLC được sử dụng cho việc truyền nhận dữ liệu giữa các khối Trí Uẩn. Thư viện này bao gồm các hàm cơ bản như sau:

```
void begin();
```

```
byte getID();

char receiveReady();
byte receiveResult();

void sendByte();
```

Ngoài ra thư viện thVLC còn có một số hàm truyền nhận nâng cao là:

```
int sensorRead();
void ledWrite();
void txSetByte();
void txSend();
```

2.1. Hàm `void begin()` – Hàm khởi tạo

Mô tả

Hàm `begin()` sử dụng để khởi tạo các cổng giao tiếp trên khối Trí Uẩn.

Mỗi khối Trí Uẩn có một số cổng giao tiếp dữ liệu, các cổng giao tiếp này thực hiện việc truyền dữ liệu thông qua đèn LED và đọc về dữ liệu từ khối Trí Uẩn khác thông qua quang trở (cảm biến ánh sáng). Chúng được kết nối với các chân tín hiệu số từ chân 2 tới chân 9 của vi điều khiển Arduino.

Cú pháp

```
thVLC.begin()
```

Ví dụ

```
#include <thVLC.h>
#include <EEPROM.h>
#include <thAvr.h>

void setup()
{
    thVLC.begin();
}

void loop()
```

```
{
}
```

2.2. Hàm `getID()`

Mô tả

Hàm `getID()` dùng để đọc ra mã số của các khối Trí Uẩn.

Bộ ghép hình Trí Uẩn gồm có 7 khối ghép, trong 7 khối ghép đó thì chỉ có 5 loại khác nhau (như đã kể ra ở mục 1). Các khối Trí Uẩn cùng loại đã được cài đặt sẵn một mã số giống nhau, có thể đọc ra các mã số đó bằng cách sử dụng hàm `getID()`.

Cú pháp

```
thVLC.getID()
```

Trả về

Hàm trả về một trong các giá trị sau:

`TRI_UAN_0 (0), TRI_UAN_1 (1), TRI_UAN_2 (2), TRI_UAN_3 (3) hoặc TRI_UAN_4 (4).`

(Các giá trị trả về có kiểu là byte)

Ví dụ

```
#include <thVLC.h>
#include <EEPROM.h>
#include <thAvr.h>

void setup()
{
  Serial.begin(9600);
  thVLC.begin();

  int IDBlock = thVLC.getID();

  switch (IDBlock)
  {
    case TRI_UAN_0:
      Serial.println("Tri Uan 0");
      break;
```

```
    case TRI_UAN_1:
      Serial.println("Tri Uan 1");
      break;
    case TRI_UAN_2:
      Serial.println("Tri Uan 2");
      break;
    case TRI_UAN_3:
      Serial.println("Tri Uan 3");
      break;
    case TRI_UAN_4:
      Serial.println("Tri Uan 4");
      break;
    default:
      Serial.println("ID not set!");
      break;
  }
}

void loop()
{}
```

2.3. Hàm `receiveReady()`

Mô tả

Hàm `receiveReady()` có chức năng là kiểm tra xem dữ liệu nhận về tại cổng giao tiếp được nối với chân `pin` của Arduino đã sẵn sàng để đọc hay chưa

Cú pháp

```
thVLC.receiveReady(pin)
```

Tham số

`pin`: cổng giao tiếp dữ liệu của khối Trí Uẩn. `pin` nằm trong khoảng từ 2 đến 9.

Giá trị trả về

`true` hoặc `false`

- Hàm trả về giá trị `true` nếu dữ liệu nhận về đã sẵn sàng để đọc
- Hàm trả về giá trị `false` nếu như cổng giao tiếp không nhận được dữ liệu hoặc dữ liệu đang được nhận nhưng chưa sẵn sàng để đọc.
- Khi dữ liệu sẵn sàng để đọc thì hàm `receiveReady` sẽ trả về giá trị `true` cho tới khi người dùng đọc dữ liệu trong bộ đệm nhận (đọc dữ liệu bằng hàm `receiveResult()`) hoặc cổng giao tiếp bắt đầu một quá trình nhận dữ liệu mới.

Ví dụ

```
#include <thVLC.h>
#include <thAvr.h>
#include <EEPROM.h>

int channel = 2;

void setup()
{
  Serial.begin(9600);
  thVLC.begin();
}

void loop()
{
  if (thVLC.receiveReady(channel) != 0)
  {
    Serial.print("Data ready to read on channel ");
    Serial.println(channel);
    thVLC.receiveResult(channel);
  }
}
```

2.4. Hàm `receiveResult()`

Mô tả

Có chức năng là đọc về dữ liệu nhận được trên cổng giao tiếp được nối với chân `pin` của Arduino.

Mỗi khi cổng giao tiếp nhận được một dữ liệu mới thì hàm `receiveReady()` sẽ trả về giá trị `true` để thông báo rằng dữ liệu nhận được đã sẵn sàng để đọc, khi này ta có thể sử dụng hàm `receiveResult()` để đọc về dữ liệu. Nếu như đọc dữ liệu trong bộ đệm nhận khi mà hàm `receiveReady()` là `false` thì có thể sẽ nhận được một dữ liệu không chính xác.

Cú pháp:

```
thVLC.receiveResult(pin)
```

Tham số

`pin`: cổng giao tiếp dữ liệu của khối Trí Uẩn. `pin` nằm trong khoảng từ 2 đến 9.

Giá trị trả về

`byte(từ 0 đến 255)`

Ví dụ

```
#include <thVLC.h>
#include <thAvr.h>
#include <EEPROM.h>

int channel = 2;

void setup()
{
  Serial.begin(9600);
  thVLC.begin();
}

void loop()
{
  if (thVLC.receiveReady(channel))
  {
    byte result = thVLC.receiveResult(channel);
    Serial.print("Data received: ");
    Serial.println(result, HEX);
  }
```

```
}
```

2.5. Hàm `sendByte()`

Mô tả

Có chức năng là truyền một `byte` dữ liệu ra cổng giao tiếp `pin` của khối Trí Uẩn.

Cú pháp:

```
thVLC.sendByte(pin, value)
```

Tham số

- `pin` : chỉ cổng giao tiếp được chọn để truyền dữ liệu
- `value` : dữ liệu sẽ được truyền đi

Ví dụ

```
#include <thVLC.h>
#include <thAvr.h>
#include <EEPROM.h>

int channel = 2;

void setup()
{
  Serial.begin(9600);
  thVLC.begin();
}

void loop()
{
  byte sendValue = 0xab;
  thVLC.sendByte(channel, sendValue);
  delay(1000);
}
```

❖ Các hàm truyền nhận nâng cao

2.6. Hàm `sensorRead()`

Mô tả

Có chức năng là đọc về giá trị của cảm biến ánh sáng được kết nối với chân `pin` của Arduino.

Trên mỗi cổng giao tiếp trong các khối Trí Uẩn đều có một bộ cảm biến ánh sáng để nhận về dữ liệu từ khối Trí Uẩn khác. Ngoài các cảm biến ánh sáng trên các cổng giao tiếp thì ở các khối Trí Uẩn 1, Trí Uẩn 3 và Trí Uẩn 4 còn có một cảm biến ánh sáng được kết nối với chân 10 của Arduino.

Việc đọc về giá trị của cảm biến ánh sáng trên các khối Trí Uẩn sẽ cung cấp cho bạn thông tin về mức độ sáng, tối của môi trường xung quanh mỗi mắt cảm biến. Và do cảm biến ánh sáng trên chân 10 được bố trí ở một mặt đáy của khối Trí Uẩn nên việc biết thông tin này sẽ cho biết khối Trí Uẩn đang úp xuống hay đang ngửa lên.

Cú pháp

```
thVLC.sensorRead(pin)
```

Tham số

`pin`: chân của Arduino được kết nối tới các bộ cảm biến ánh sáng. `pin` nằm trong khoảng từ 2 đến 10.

Giá trị trả về

```
int (từ 0 tới 1023)
```

Ví dụ

```
#include <thVLC.h>
#include <thAvr.h>
#include <EEPROM.h>

int sensor_2  = 2;
int sensor_10 = 10;

void setup()
{
  Serial.begin(9600);
  thVLC.begin();
```

```
}

void loop()
{
  int readValue2  = thVLC.sensorRead(sensor_2);
  int readValue10 = thVLC.sensorRead(sensor_10);

  Serial.print("[sensor 2] : ");
  Serial.print(readValue2);

  Serial.print(" \t[sensor 10] : ");
  Serial.println(readValue10);

  delay(500);
}
```

2.7. Hàm ledWrite()

Mô tả

Có chức năng là bật hoặc tắt đèn LED trên cổng giao tiếp `pin`.

- `value = HIGH` : bật sáng LED
- `value = LOW` : tắt đèn LED

Cú pháp

```
thVLC.ledWrite(pin, value)
```

Tham số

`pin`: cổng giao tiếp dữ liệu của khối Trí Uẩn. `pin` nằm trong khoảng từ 2 đến 9.

`value`: `HIGH` hoặc `LOW`.

Ví dụ

```
#include <thVLC.h>
#include <thAvr.h>
#include <EEPROM.h>
```

```
void setup()
{
  thVLC.begin();
}

void loop()
{
  thVLC.ledWrite(2, HIGH);
  delay(1000);
  thVLC.ledWrite(2, LOW);
  delay(1000);
}
```

2.8. Hàm `txSetByte()`

Mô tả

Hàm `txSetByte()` kết hợp với hàm `txSend()` để truyền dữ liệu đồng thời trên nhiều cổng giao tiếp của một khối Trí Uẩn. Trong đó, chức năng của hàm `txSetByte()` là khai báo và cài đặt dữ liệu sẽ truyền trên các cổng giao tiếp.

Cú pháp

```
thVLC.txSetByte(pin, value)
```

Tham số

- `pin`: cổng giao tiếp sẽ sử dụng để truyền dữ liệu
- `value`: dữ liệu sẽ truyền đi trên cổng giao tiếp `pin`

2.9. Hàm `txSend()`

Mô tả

Sau khi khai báo và cài đặt các `byte` sẽ truyền trên các cổng giao tiếp bằng hàm `txSetByte()` thì sử dụng hàm `txSend()` để bắt đầu quá trình truyền dữ liệu đồng thời trên nhiều kênh truyền.

Cú pháp

```
thVLC.txSend()
```

Ví dụ

```
#include <thVLC.h>
#include <thAvr.h>
#include <EEPROM.h>

const int channel_0 = 2;
const int channel_1 = 3;

void setup()
{
  thVLC.begin();
}

void loop()
{
  byte sendValue = 0xab;
  thVLC.txSetByte(channel_0, sendValue);
  thVLC.txSetByte(channel_1, sendValue + 1);
  thVLC.txSend();
  delay(1000);
}
```

3. Thư viện thLedMatrix

Trên khối Trí Uẩn 0 có thiết kế một đèn LED ma trận 8x8 (có hai màu cơ bản là xanh và đỏ), đèn LED ma trận này có thể được lập trình để hiển thị các hình ảnh lên trên đó.

Thư viện thLedMatrix được sử dụng để lập trình hiển thị cho đèn LED ma trận này. Các hàm trong thư viện thLedMatrix như sau:

3.1. Hàm begin()

Mô tả

Hàm begin() là hàm khởi tạo hoạt động cho đèn LED ma trận.

Cú pháp

```
thLedMatrix.begin()
```

Ví dụ

```
#include <thLedMatrix.h>
#include <SPI.h>
#include <thAvr.h>
#include <EEPROM.h>

void setup()
{
  thLedMatrix.begin();
}

void loop()
{
}
```

3.2. Hàm clear()

Mô tả

Dùng để xóa hiển thị trên đèn LED ma trận bằng cách điều khiển tất cả các pixel về trạng thái tối.

Cú pháp

```
thLedMatrix.clear()
```

Ví dụ

```
#include <thLedMatrix.h>
#include <SPI.h>
#include <thAvr.h>
```

```
#include <EEPROM.h>

void setup()
{
  thLedMatrix.begin();
}

void loop()
{
  thLedMatrix.setPixel(0, 0, ORANGE);
  thLedMatrix.setPixel(2, 0, RED);
  thLedMatrix.setPixel(4, 0, GREEN);
  delay(1000);
  thLedMatrix.clear();
  delay(1000);
}
```

3.3. Hàm setPixel()

Mô tả

Có chức năng điều khiển màu sắc hiển thị cho các pixel trên đèn LED ma trận. Do mỗi pixel trong đèn LED ma trận có hai màu cơ bản là xanh và đỏ do đó nó có thể hiển thị được 4 màu là: xanh(green), đỏ(red), vàng(yellow) và đen(black).

Cú pháp

```
thLedMatrix.setPixel(x, y, color)
```

Tham số

`x` : (byte) hoành độ của pixel

`y` : (byte) tung độ của pixel

`color` : (byte) màu sắc sẽ cài đặt cho pixel, color nhận một trong các giá trị là `BLACK`, `RED`, `YELLOW` hoặc `ORANGE`

Ví dụ

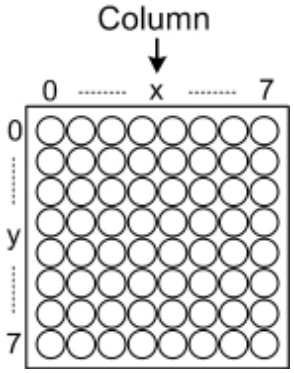
```
#include <thLedMatrix.h>
#include <SPI.h>
#include <thAvr.h>
```

```
#include <EEPROM.h>

void setup()
{
  thLedMatrix.begin();
}

void loop()
{
  thLedMatrix.setPixel(0, 3, ORANGE);
  thLedMatrix.setPixel(1, 3, RED);
  thLedMatrix.setPixel(2, 3, GREEN);
  thLedMatrix.setPixel(3, 3, BLACK);
}
```

3.4. Hàm setColumn()

Mô tả

Hàm setColumn() dùng để điều khiển hiển thị của 8 pixel nằm trên cùng một cột của đèn LED ma trận.



Cú pháp

```
thLedMatrix.setColumn(x, redBitmap, greenBitmap)
```

Tham số

`x` : (byte) số thứ tự của cột trong LED ma trận.

`redBitmap` : (byte) giá trị dùng để điều khiển hiển thị màu đỏ (red) của các pixel trong cột.

`greenBitmap` : (byte) giá trị dùng để điều khiển hiển thị màu xanh (green) của các pixel trong cột.

Ví dụ

```
#include <thLedMatrix.h>
#include <SPI.h>
#include <thAvr.h>
#include <EEPROM.h>

void setup()
{
  thLedMatrix.begin();
}

void loop()
{
  byte redBitmap = 0x0f;
  byte grnBitmap = 0xf0;
  thLedMatrix.setColumn(0, redBitmap, grnBitmap);
}
```

3.5. Hàm setBitmap()

Mô tả

Hàm setBitmap() được dùng để hiển thị một hình ảnh lên đèn LED ma trận. Hình ảnh này được mã hóa thành 16 byte và lưu trong một ma trận có cấu trúc như sau:

```
BITMAP image[] =
{
   red0, grn0,
   red1, grn1,
   red2, grn2,
   red3, grn3,
   red4, grn4,
   red5, grn5,
   red6, grn6,
   red7, grn7
};
```

Trong ma trận image[] , 16 giá trị là các giá trị mã hóa cho hình ảnh theo từng cột. Trong đó:

- 8 phần tử `red0, red1, …, red7` tương ứng là các giá trị mã hóa màu đỏ của các cột từ 0 tới 7
- 8 phần tử `grn0, grn1, …, grn7` tương ứng là các giá trị mã hóa màu xanh của các cột từ 0 tới 7

Cú pháp

```
thLedMatrix.setBitmap(image)
```

Tham số

`image`: hình ảnh muốn hiển thị trên LED ma trận

Ví dụ

```
#include <thLedMatrix.h>
#include <SPI.h>
#include <thAvr.h>
#include <EEPROM.h>

BITMAP image[] =
{
  0x3c, 0x00,
  0x42, 0x00,
  0x95, 0x14,
  0xa1, 0x20,
  0xa1, 0x20,
  0x95, 0x14,
  0x42, 0x00,
  0x3c, 0x00
};

void setup()
{
  thLedMatrix.begin();
}

void loop()
```

```
{
  thLedMatrix.setBitmap(image);
}
```

Kết quả chạy ví dụ