

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ

--------



BÁO CÁO BÀI TẬP LỚN

MÔN: VI XỬ LÝ

Giảng viên hướng dẫn: *ĐỖ NGỌC CẨM*

Nhóm thực hiện: *Nhóm 3*

Lớp : *L18*

Đề tài: *2 – LED ma trận trái tim 16x16*

STT	HỌ VÀ TÊN	MSSV
1	TÙ CAO DUY	2012851
2	PHẠM DUY LỢI	2013706
3	CAO VĂN HIẾU	2010251

Tp.HCM, Tháng 6/2022

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ



BÁO CÁO BÀI TẬP LỚN
MÔN: VI XỬ LÝ

Giảng viên hướng dẫn: *ĐỖ NGỌC CẨM*

Nhóm thực hiện: *Nhóm 3*

Lớp : *L18*

Đề tài: *2 – LED ma trận trái tim 16x16*

STT	MSSV	Họ và Tên	Công việc thực hiện	Ghi chú
1	2012851	TÙ CAO DUY	Tìm hiểu và đảm nhận phần mềm	
2	2013706	PHẠM DUY LỢI	Tìm hiểu và đảm nhận phần cứng	
3	2010251	CAO VĂN HIẾU	Thiết kế mạch, mở rộng, báo cáo	

LỜI NÓI ĐẦU

Trong thời đại ngày nay khi nhân loại đang trải qua những bước tiến vượt bậc về khoa học công nghệ, thì ngành điện tử cũng có những bước tiến quan trọng đặc biệt là trong lĩnh vực vi điều khiển. Các bộ vi xử lý ngày càng phát triển và hoàn thiện hơn, được sử dụng trong hầu hết các hệ thống điều khiển tự động trong công nghiệp, nhúng, khoa học kỹ thuật, cũng như trong các thiết bị dân dụng. Chính nhờ vai trò, chức năng của vi xử lý đã đem lại nhiều ưu điểm, nhiều tính năng đặc biệt cho các hệ thống điều khiển thay thế con người trong các công việc đòi hỏi sự phức tạp và yêu cầu kỹ thuật cao.

Học thì đi đôi với hành. Để áp dụng tính năng đặc biệt đó của vi xử lý vào thực tiễn, áp dụng những kiến thức đã học trong quá trình thí nghiệm, lên lớp, Nhóm 2 hân hạnh đem tới bài báo cáo BTL đã được giao làm và tìm hiểu. Sử dụng IC AT89S52 thuộc họ 8051 là IC điều khiển chính. Ngoài AT89S52 còn thêm một số linh kiện phụ trợ khác.

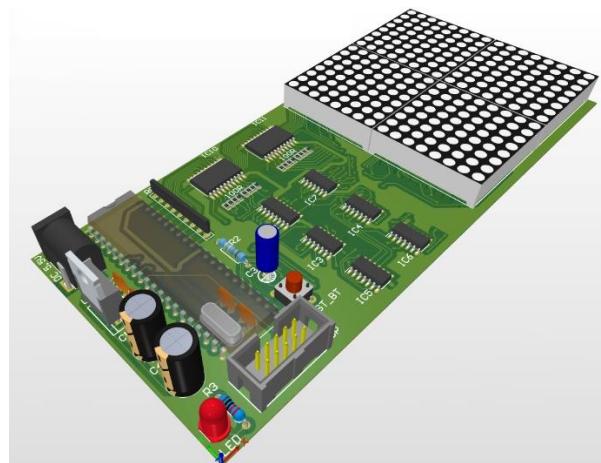
Bài báo cáo này được xây dựng trên cơ sở kiến thức đã tiếp thu từ các bài giảng, giáo trình điện tử, quá trình thực nghiệm và tài liệu thông tin trên Internet.

Do điều kiện, kiến thức cũng như trình độ hiểu biết còn hạn chế, bài báo cáo của nhóm 2 chúng em không tránh khỏi những thiếu sót, khuyết điểm. Chúng em rất mong nhận được sự cảm thông, chia sẻ, góp ý từ phía Thầy và các bạn nhằm giúp nhóm hoàn thiện hơn không những ở bản báo cáo này mà còn là những mặt kiến thức còn thiếu sót, qua đó hoàn thành các tiêu chí thành phần thí nghiệm tốt hơn.

Qua đây, chúng em xin được chân thành cảm ơn Thầy Đỗ Ngọc Cẩm đã nhiệt tình hướng dẫn, chỉ bảo, hỗ trợ nhóm hoàn thành bài báo cáo này.

TÓM TẮT BÀI BÁO CÁO

- Thiết kế phần cứng ứng dụng 8051
- Thiết kế phần mềm ứng dụng 8051
- Mô phỏng, thiết kế bằng các phần mềm chuyên dụng



MỤC LỤC

LỜI NÓI ĐẦU	i
TÓM TẮT BÀI BÁO CÁO	ii
YÊU CẦU - ĐỀ TÀI 2:	1
1. Đề tài	1
2. Yêu cầu khi làm trên Proteus	1
3. Yêu cầu khi làm PCB	2
CHƯƠNG 1. THIẾT KẾ PHẦN CỨNG TRÊN PROTEUS	3
1.1. Sơ đồ khối tổng quan	3
1.1.1. Các sơ đồ	3
1.1.2. Giải thích các khối.....	4
1.1.3. Sơ đồ chi tiết mạch.....	4
1.2. Sơ đồ chi tiết từng phần.....	4
1.2.1. Giới thiệu linh kiện chính.....	5
1.2.2. Khối MCU.....	6
1.2.3. Khối giải mã.....	8
1.2.4. Khối quét LED	11
1.2.5. Khối hiển thị	16
1.3. Thiết kế giải mã địa chỉ.....	18
CHƯƠNG 2. THIẾT KẾ PHẦN MỀM BẰNG KEILC - ASM	26
2.1. Lưu đồ giải thuật chương trình	26
2.1.1. Ý tưởng	27
2.2. Sơ đồ giải thuật chương trình chính.....	30
2.2.1. Chi tiết giải thuật	30

2.3. Sơ đồ giải thuật các chương trình ngắn	39
2.3.1. Ngắt timer 0	39
2.3.2. Ngắt ngoài	41
CHƯƠNG 3. MỎ RỘNG	42
 3.1. Thiết kế phần mềm bằng KeilC – C	42
3.1.1. Ý tưởng – Khác biệt so với ASM	42
3.1.2. Lưu đồ giải thuật chương trình	42
3.1.3. Giải thích thuật toán	45
3.1.4. Tổng kết	50
 3.2. Thiết kế mạch PCB bằng Altium	51
3.2.1. Sơ đồ nguyên lý	51
3.2.2. Layout (Phủ đồng GND)	53
3.2.3. 3D viewer	57
CHƯƠNG 4. ĐO ĐẠC MÔ PHỎNG VÀ KẾT LUẬN	59
 4.1. Kiểm chứng thời gian trên Proteus	59
4.1.1. ASM	59
4.1.2. C	61
 4.2. Hướng phát triển đề tài	61
4.2.1. Ưu và nhược điểm của mạch	61
4.2.2. Hướng phát triển	61
 4.3. Kết luận	62
PHỤ LỤC	63
1. Bản thiết kế mạch điện trên Proteus	63
2. Bản thiết kế mạch PCB Altium Schematic	63

BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18

3. Bản PDF PCB các lớp	63
4. CODE ASM.....	63
5. CODE C.....	68
6. Video chạy mô phỏng	74



YÊU CẦU - ĐỀ TÀI 2:

1. Đề tài

Thiết kế mạch điện tử sử dụng vi xử lý 8051 và một vi mạch 74'138 cùng các cổng logic cần thiết để điều khiển một bảng LED ma trận có kích thước 16x16 (được ghép từ 4 LED ma trận 8x8) với địa chỉ truy suất được cho ở bảng dưới đây.

Địa chỉ truy suất	Ngoại vi giao tiếp
0000H - 17FFH	8 bit cao của cột
1800H – 3FFFH	8 bit thấp của cột
4000H – 8FFFH	8 bit cao của hàng
9000H - AFFFH	8 bit thấp của cột

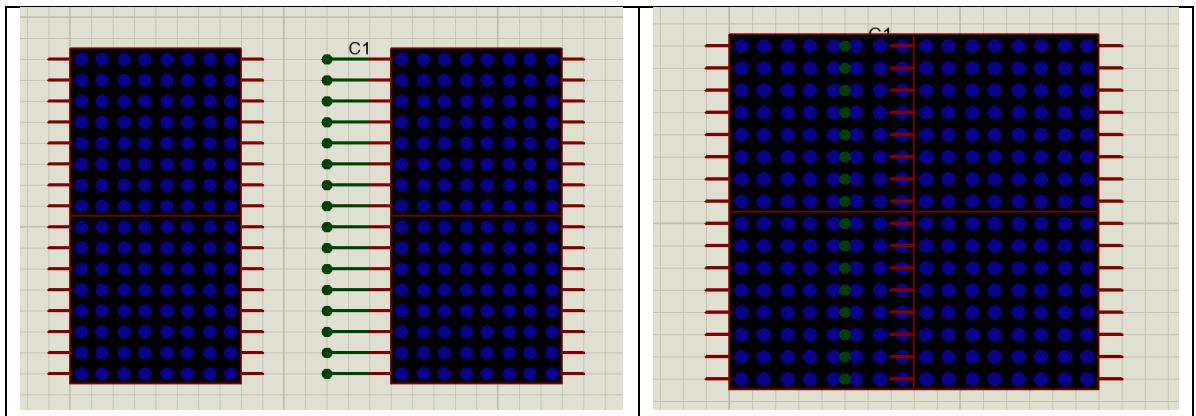
Nhóm sinh viên lựa chọn một trong 2 yêu cầu dưới đây để làm.

(Đối với các bạn sinh viên thuộc chương trình tài năng, cần làm thêm bài mở rộng, đề tài bài mở rộng là yêu cầu còn lại (mà nhóm không làm), làm cá nhân)

2. Yêu cầu khi làm trên Proteus

Vẽ mô phỏng trên Proteus, lập trình cho 8051 hiển thị hình trái tim trên bảng LED đã thiết kế. Hình ảnh trái tim được giữ nguyên trong suốt 1s, sau đó làm hiệu ứng biến mất từng điểm ảnh của trái tim (tắt từng điểm ảnh) từ trái qua phải, từ trên xuống dưới, mỗi điểm ảnh biến mất sau mỗi 0.1s. Sau khi hình ảnh trái tim biến mất hết, bắt đầu vòng lặp mới từ việc hiển thị toàn bộ hình trái tim trên màn LED. Chúc may mắn <3

P/s: Khi thiết kế bảng LED trên Proteus, sinh viên cứ nối dây bằng cách đặt tên, nối dây xong rồi kéo bảng LED lại. Chú ý, sinh viên thiết kế sao cho trọng số thấp của hàng nằm phía tay trái, trọng số thấp của cột nằm ở dưới.



3. Yêu cầu khi làm PCB

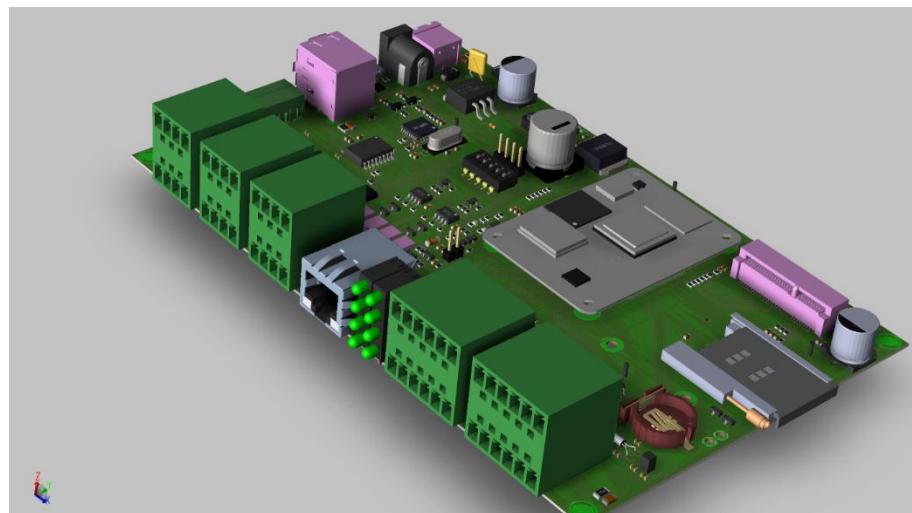
Vẽ sơ đồ thiết kế: sơ sò thiết kế phải được vẽ một cách hoàn chỉnh (cấp nguồn, clock...)

Vẽ PCB: Mạch 2 lớp, linh kiện dán hoặc xuyên lỗ đều được, sắp xếp linh kiện hợp lý, thẩm mỹ.

Sinh viên thiết kế trên mọi phần mềm (online/offline) đều được. Các phần mềm được gợi ý là: Orcad, Altium, EasyEDA, KiCad, EAGLE...

P/s: Có 3D view sẽ được cộng điểm.

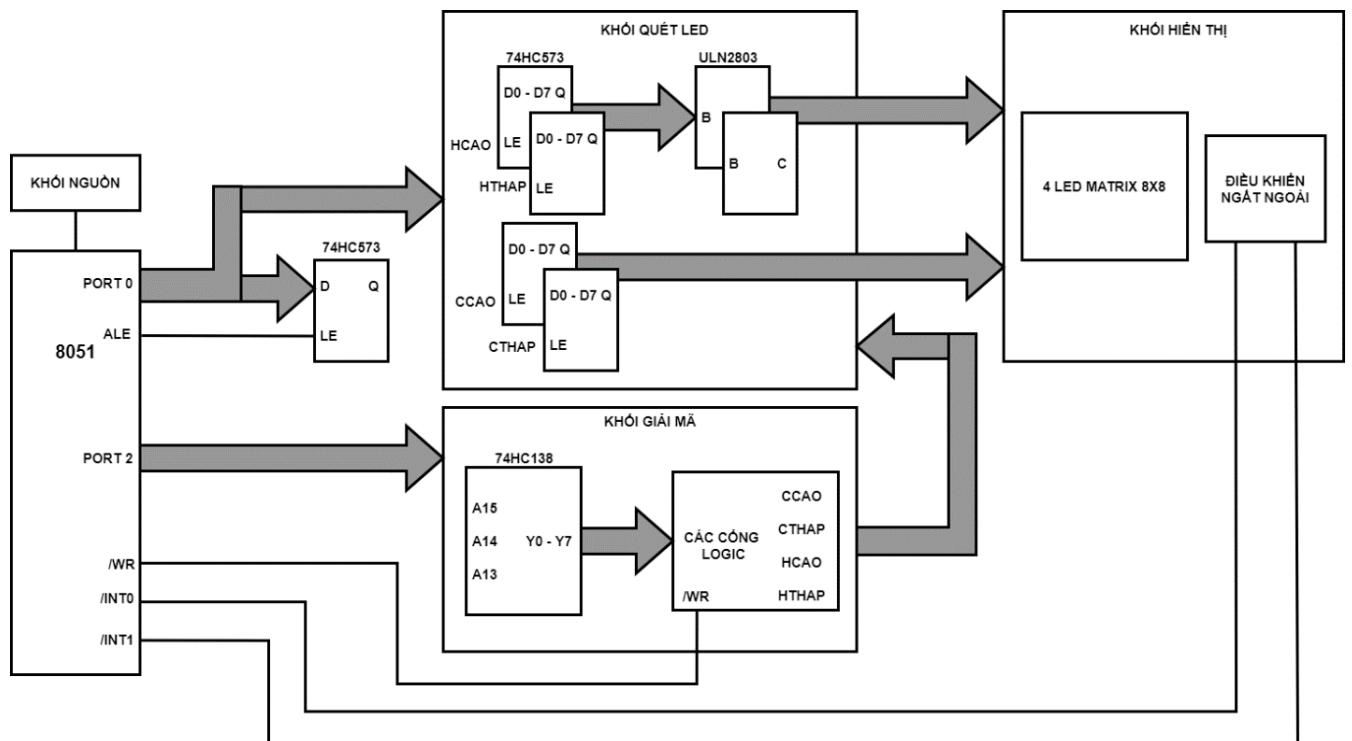
(Hình minh họa 3D view)



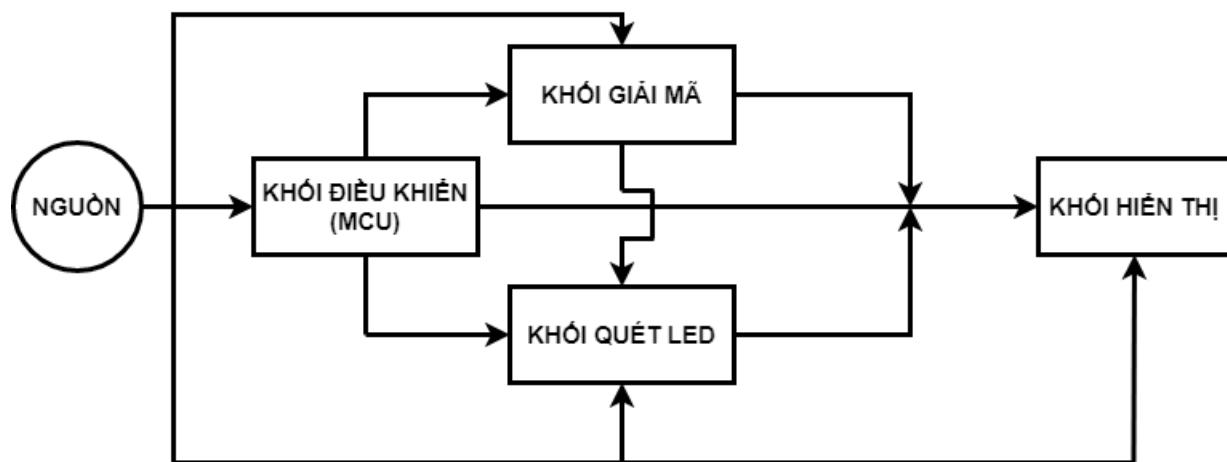
CHƯƠNG 1. THIẾT KẾ PHẦN CỨNG TRÊN PROTEUS

1.1. Sơ đồ khối tổng quan

1.1.1. Các sơ đồ



Hình 1.1. Sơ đồ khối tổng quan

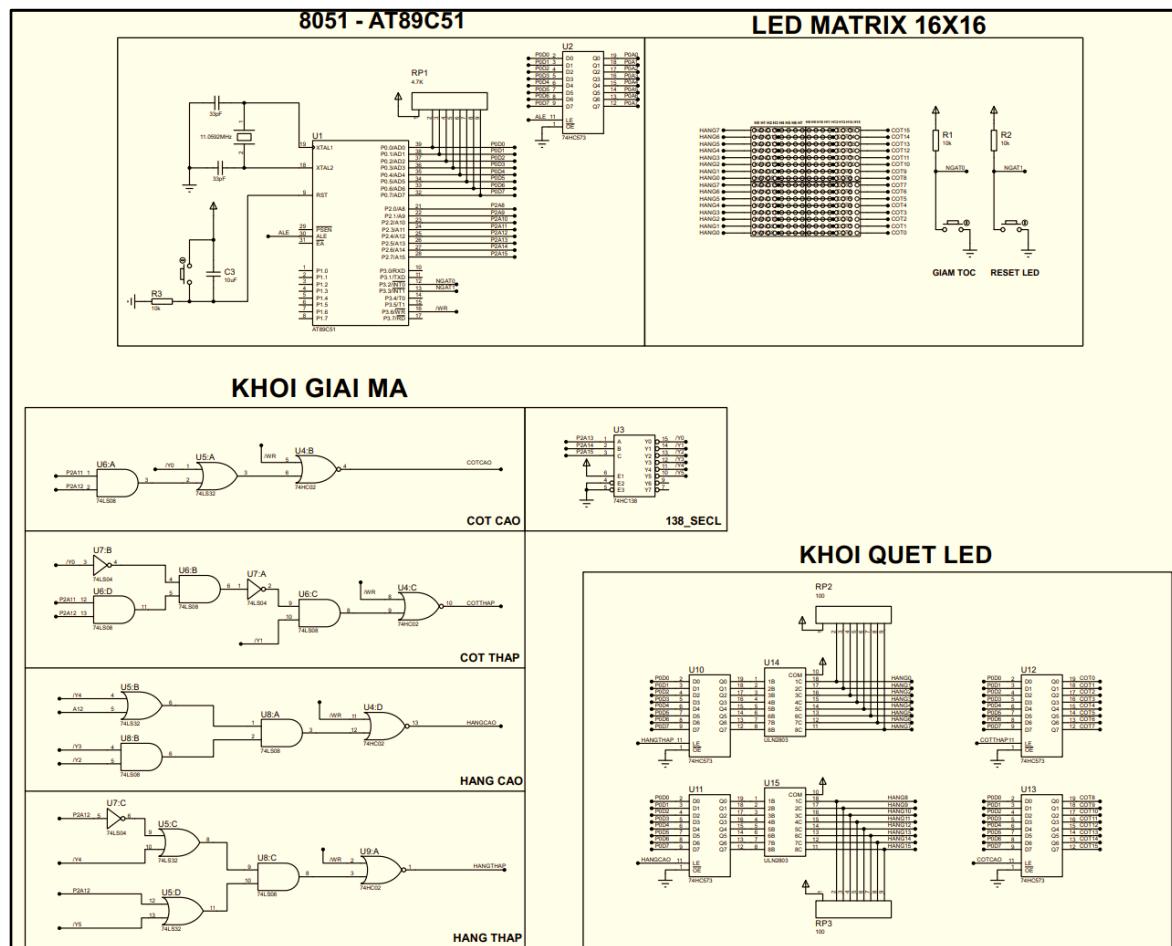


Hình 1.2. Sơ đồ khối tóm lược

1.1.2. Giải thích các khối.

- Khối nguồn: Cáp nguồn ổn áp 5V cho mạch hoạt động
- Khối MCU: 8051 làm vi xử lý trung tâm
- Khối giải mã: Gồm 1 IC 74HC138 và các công logic để lựa chọn truy xuất các ngoại vi.
- Khối quét LED: Gồm các IC 74HC573 với nhiệm vụ nhận dữ liệu, IC ULN2803 đệm đảo.
- Khối hiển thị: 4 LED Matrix 8x8, bổ sung 2 nút nhấn ngắt ngoài để dễ quan sát (SLOW và RESET).

1.1.3. Sơ đồ chi tiết mạch.

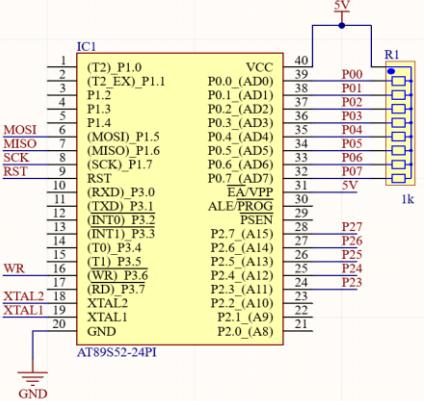
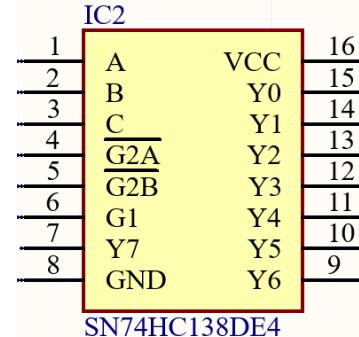
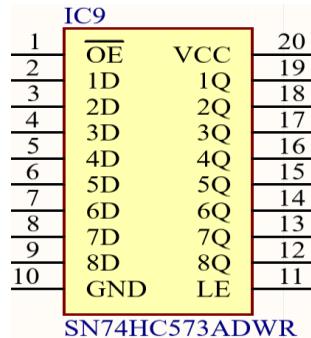
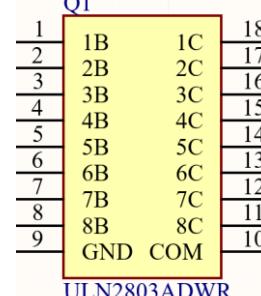


Hình 1.3. Sơ đồ chi tiết mạch

1.2. Sơ đồ chi tiết từng phần.

1.2.1. Giới thiệu linh kiện chính

Bảng 1.0.1. Các linh kiện chính sử dụng trong mạch

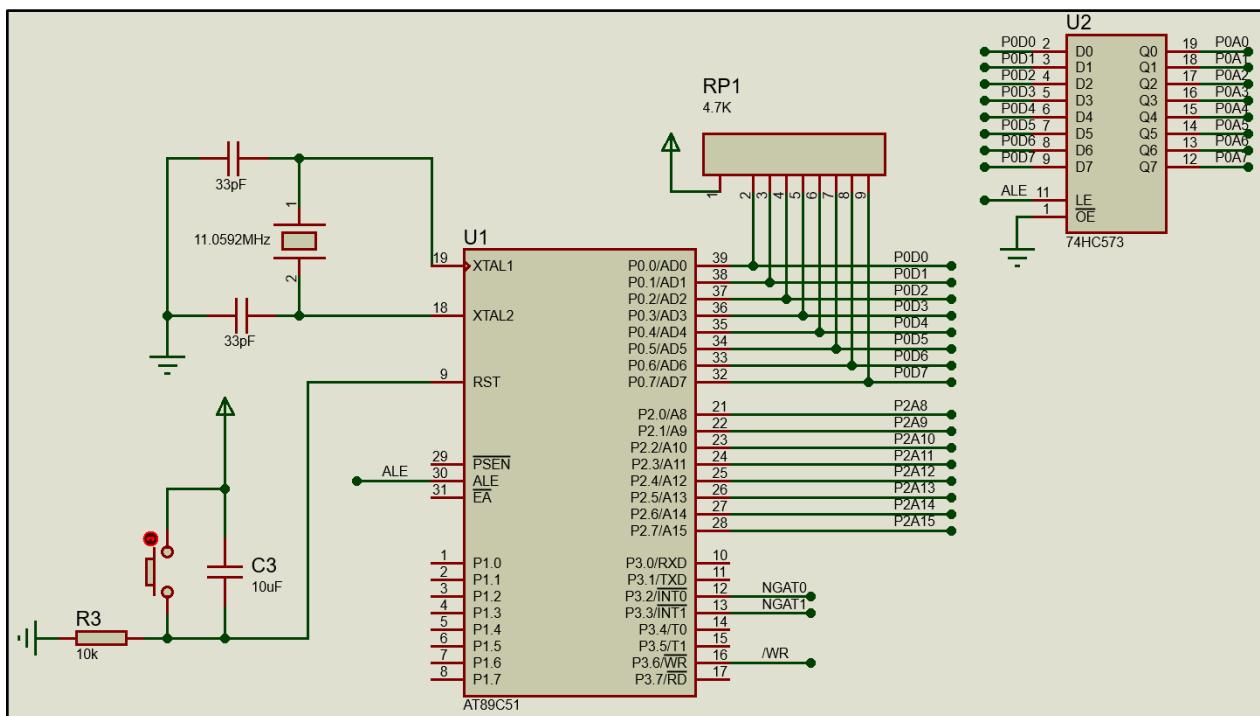
AT89S52	<ul style="list-style-type: none"> - AT89S52 là một hệ vi tính 8 bit đơn chip CMOS có hiệu suất cao, công suất nguồn tiêu thụ thấp. - 8Kbyte bộ nhớ ROM Flash xóa được lập trình được. 	 <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>IC1</td></tr> <tr><td>1 (T2) P1.0</td><td>P0.0 (AD0)</td></tr> <tr><td>2 (T2) EX_P1.1</td><td>P0.1 (AD1)</td></tr> <tr><td>3 P1.2</td><td>P0.2 (AD2)</td></tr> <tr><td>4 P1.3</td><td>P0.3 (AD3)</td></tr> <tr><td>5 P1.4</td><td>P0.4 (AD4)</td></tr> <tr><td>MISO 6 (MOSI) P1.5</td><td>P0.5 (AD5)</td></tr> <tr><td>MISO 7 (MISO) P1.6</td><td>P0.6 (AD6)</td></tr> <tr><td>SCK 8 (SCK) P1.7</td><td>P0.7 (AD7)</td></tr> <tr><td>RST 9</td><td>EA/VPP</td></tr> <tr><td>10 (RXD) P3.0</td><td>ALE/PROG</td></tr> <tr><td>11 (TXD) P3.1</td><td>PSEN</td></tr> <tr><td>12 (INT0) P3.2</td><td>P2.7 (A15)</td></tr> <tr><td>13 (INT1) P3.3</td><td>P2.6 (A14)</td></tr> <tr><td>14 (T0) P3.4</td><td>P2.5 (A13)</td></tr> <tr><td>15 (T1) P3.5</td><td>P2.4 (A12)</td></tr> <tr><td>WR 16 (WR) P3.6</td><td>P2.3 (A11)</td></tr> <tr><td>XTAL2 18 (RDI) P3.7</td><td>P2.2 (A10)</td></tr> <tr><td>XTAL1 19 XTAL2</td><td>P2.1 (A9)</td></tr> <tr><td>GND 20</td><td>P2.0 (A8)</td></tr> <tr><td colspan="2">AT89S52-24PI</td></tr> </table>	IC1	1 (T2) P1.0	P0.0 (AD0)	2 (T2) EX_P1.1	P0.1 (AD1)	3 P1.2	P0.2 (AD2)	4 P1.3	P0.3 (AD3)	5 P1.4	P0.4 (AD4)	MISO 6 (MOSI) P1.5	P0.5 (AD5)	MISO 7 (MISO) P1.6	P0.6 (AD6)	SCK 8 (SCK) P1.7	P0.7 (AD7)	RST 9	EA/VPP	10 (RXD) P3.0	ALE/PROG	11 (TXD) P3.1	PSEN	12 (INT0) P3.2	P2.7 (A15)	13 (INT1) P3.3	P2.6 (A14)	14 (T0) P3.4	P2.5 (A13)	15 (T1) P3.5	P2.4 (A12)	WR 16 (WR) P3.6	P2.3 (A11)	XTAL2 18 (RDI) P3.7	P2.2 (A10)	XTAL1 19 XTAL2	P2.1 (A9)	GND 20	P2.0 (A8)	AT89S52-24PI	
IC1																																											
1 (T2) P1.0	P0.0 (AD0)																																										
2 (T2) EX_P1.1	P0.1 (AD1)																																										
3 P1.2	P0.2 (AD2)																																										
4 P1.3	P0.3 (AD3)																																										
5 P1.4	P0.4 (AD4)																																										
MISO 6 (MOSI) P1.5	P0.5 (AD5)																																										
MISO 7 (MISO) P1.6	P0.6 (AD6)																																										
SCK 8 (SCK) P1.7	P0.7 (AD7)																																										
RST 9	EA/VPP																																										
10 (RXD) P3.0	ALE/PROG																																										
11 (TXD) P3.1	PSEN																																										
12 (INT0) P3.2	P2.7 (A15)																																										
13 (INT1) P3.3	P2.6 (A14)																																										
14 (T0) P3.4	P2.5 (A13)																																										
15 (T1) P3.5	P2.4 (A12)																																										
WR 16 (WR) P3.6	P2.3 (A11)																																										
XTAL2 18 (RDI) P3.7	P2.2 (A10)																																										
XTAL1 19 XTAL2	P2.1 (A9)																																										
GND 20	P2.0 (A8)																																										
AT89S52-24PI																																											
SN74HC138	<ul style="list-style-type: none"> - 74HC138 giải mã ba đầu vào địa chỉ có trọng số nhị phân (A0, A1 và A2) thành tám đầu ra loại trừ lẫn nhau (Y0 đến Y7). - Thiết bị có ba đầu vào bật (E1, E2 và E3). Mọi đầu ra sẽ ở mức CAO trừ khi E1 và E2 là THẤP và E3 là CAO 	 <p>IC2</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1 A</td><td>VCC</td><td>16</td></tr> <tr><td>2 B</td><td>Y0</td><td>15</td></tr> <tr><td>3 C</td><td>Y1</td><td>14</td></tr> <tr><td>4 G2A</td><td>Y2</td><td>13</td></tr> <tr><td>5 G2B</td><td>Y3</td><td>12</td></tr> <tr><td>6 G1</td><td>Y4</td><td>11</td></tr> <tr><td>7 Y7</td><td>Y5</td><td>10</td></tr> <tr><td>8 GND</td><td>Y6</td><td>9</td></tr> </table> <p>SN74HC138DE4</p>	1 A	VCC	16	2 B	Y0	15	3 C	Y1	14	4 G2A	Y2	13	5 G2B	Y3	12	6 G1	Y4	11	7 Y7	Y5	10	8 GND	Y6	9																	
1 A	VCC	16																																									
2 B	Y0	15																																									
3 C	Y1	14																																									
4 G2A	Y2	13																																									
5 G2B	Y3	12																																									
6 G1	Y4	11																																									
7 Y7	Y5	10																																									
8 GND	Y6	9																																									
SN74HC573	<ul style="list-style-type: none"> - IC 74HC573 là 1 vi mạch chốt dữ liệu khi LE tích cực. - Đầu vào của 74HC574 tương thích với đầu ra chuẩn CMOS, bên cạnh đó IC còn tương thích với đầu ra LS/ALSTTL. 	 <p>IC9</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1 OE</td><td>VCC</td><td>20</td></tr> <tr><td>2 1D</td><td>1Q</td><td>19</td></tr> <tr><td>3 2D</td><td>2Q</td><td>18</td></tr> <tr><td>4 3D</td><td>3Q</td><td>17</td></tr> <tr><td>5 4D</td><td>4Q</td><td>16</td></tr> <tr><td>6 5D</td><td>5Q</td><td>15</td></tr> <tr><td>7 6D</td><td>6Q</td><td>14</td></tr> <tr><td>8 7D</td><td>7Q</td><td>13</td></tr> <tr><td>9 8D</td><td>8Q</td><td>12</td></tr> <tr><td>10 GND</td><td>LE</td><td>11</td></tr> </table> <p>SN74HC573ADWR</p>	1 OE	VCC	20	2 1D	1Q	19	3 2D	2Q	18	4 3D	3Q	17	5 4D	4Q	16	6 5D	5Q	15	7 6D	6Q	14	8 7D	7Q	13	9 8D	8Q	12	10 GND	LE	11											
1 OE	VCC	20																																									
2 1D	1Q	19																																									
3 2D	2Q	18																																									
4 3D	3Q	17																																									
5 4D	4Q	16																																									
6 5D	5Q	15																																									
7 6D	6Q	14																																									
8 7D	7Q	13																																									
9 8D	8Q	12																																									
10 GND	LE	11																																									
ULN2803	<ul style="list-style-type: none"> - ULN2803 là chuỗi các transistor Darlington điện áp và dòng điện cao. - Thông thường, các chip đang được sử dụng với mục đích làm IC đệm đảo. 	 <p>Q1</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1 1B</td><td>1C</td><td>18</td></tr> <tr><td>2 2B</td><td>2C</td><td>17</td></tr> <tr><td>3 3B</td><td>3C</td><td>16</td></tr> <tr><td>4 4B</td><td>4C</td><td>15</td></tr> <tr><td>5 5B</td><td>5C</td><td>14</td></tr> <tr><td>6 6B</td><td>6C</td><td>13</td></tr> <tr><td>7 7B</td><td>7C</td><td>12</td></tr> <tr><td>8 8B</td><td>8C</td><td>11</td></tr> <tr><td>9 GND</td><td>COM</td><td>10</td></tr> </table> <p>ULN2803ADWR</p>	1 1B	1C	18	2 2B	2C	17	3 3B	3C	16	4 4B	4C	15	5 5B	5C	14	6 6B	6C	13	7 7B	7C	12	8 8B	8C	11	9 GND	COM	10														
1 1B	1C	18																																									
2 2B	2C	17																																									
3 3B	3C	16																																									
4 4B	4C	15																																									
5 5B	5C	14																																									
6 6B	6C	13																																									
7 7B	7C	12																																									
8 8B	8C	11																																									
9 GND	COM	10																																									

1088AS	<p>- LED ma trận Cột Anode thường được sử dụng trong thực tiễn với kích thước 8x8</p>	 1088AS
---------------	---	-------------------

Cùng các IC cổng Logic như: 74LS04 (NOT), 74LS08 (AND), 74LS32 (OR)

74HC02 (NOR). Linh kiện khác: Trở, trở băng, tụ, thạch anh...

1.2.2. Khởi MCU.



Hình 1.4. Khởi MCU trên Proteus

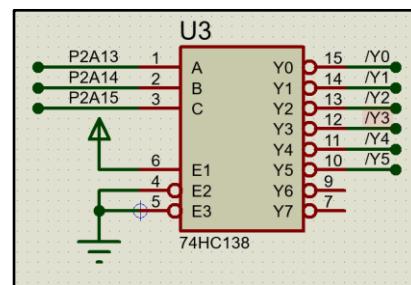
Bảng 1.2. Chức năng và kết nối các chân Pin trên MCU

Chân (pin)	Kiểu chân	Chức năng
Port 0 (pin 39 → 32)	InOut	<ul style="list-style-type: none"> - Nửa đầu chu kỳ truy xuất, ALE = 1, Port 0 định địa chỉ A0 → A7. Sau đó được chốt ‘573 qua P0A0 → P0A7. - Nửa sau chu kỳ truy xuất, ALE = 0, Port 0 xuất Data được chốt vào 1 trong 4 ‘573 nằm trong Khối quét LED.
Port 2 (pin 21 → 28)	Output	<ul style="list-style-type: none"> - Khi truy xuất dữ liệu, Port 2 đóng vai trò làm 8 bit cao của địa chỉ truy xuất 16 bit A8 → A15. - 8 chân này được nối tới Khối giải mã để lựa chọn ‘573 hoạt động.
/INT0 (pin 12)	Input	<ul style="list-style-type: none"> - Kết nối tới nút nhấn SLOW Khối hiển thị, nhận tín hiệu 0 sẽ thực hiện chương trình ngắt ngoài 0.
/INT1 (pin 13)	Input	<ul style="list-style-type: none"> - Kết nối tới nút nhấn RESET Khối hiển thị, nhận tín hiệu 0 sẽ thực hiện chương trình ngắt ngoài 0.
/WR (pin 16)	Output	<ul style="list-style-type: none"> - Kết nối tới Khối giải mã, sẽ được NOR với các tín hiệu giải được. - Được tích cực thấp khi giao tiếp với RAM ngoài và đã ổn định dữ liệu trên BUS DATA.

EA (pin 31)	Input	- Mặc định được tích cực 1 để sử dụng chương trình ROM nội.
ALE (pin 30)	Output	<ul style="list-style-type: none"> - Xung chốt địa chỉ → xác định tín hiệu trên các chân dòn kênh AD là tín hiệu địa chỉ hay dữ liệu. - Khi ALE=1 thì tín hiệu trên các chân dòn kênh AD là tín hiệu địa chỉ. - Kết nối tới ‘573-IC chốt địa chỉ Port 0.
RST(pin 9)	Input	<ul style="list-style-type: none"> - Ngõ vào RST ở mức cao ít nhất 2 chu kỳ máy (MC) thì 8051 reset; các thanh ghi bên trong nạp giá trị khởi động lại hệ thống. - Kết nối tới nút nhấn RST .
XTAL1, XTAL 2 (pin 19 , pin 18)	Input	<ul style="list-style-type: none"> - 2 chân này được dùng theo dạng dao động trên chip với bộ thạch anh. Tần số thạch anh được sử dụng là 11.059Mhz.

1.2.3. Khối giải mã.

a. 74HC138

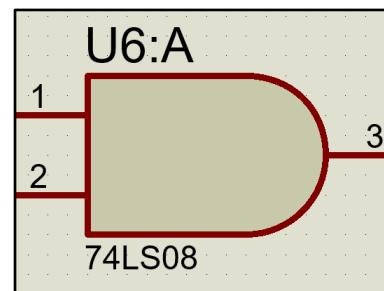


Hình 1.5. 74HC138 trong Proteus

Bảng 1.3. Chức năng và kết nối của các Pin 74HC138

Chân (pin)	Kiểu chân	Chức năng
A,B,C (pin 1 2 3)	Input	<ul style="list-style-type: none"> - Kết nối tới 3 chân của Port 2: P2.5, P2.6, P2.7 - Một trong những bước phân vùng giải mã địa chỉ truy xuất (sẽ được nói rõ phần chi tiết giải mã địa chỉ).
/Y0 → /Y5 (pin 15 → pin 10)	Output	<ul style="list-style-type: none"> - Tích cực thấp dựa trên tín hiệu được đưa vào 3 chân ABC, Demux 3 sang 8. - Các chân được nối tới các IC công thực hiện tiếp tục việc giải mã địa chỉ.
E1,/E2,/E3 (pin 6, pin 4, pin 5)	Input	<ul style="list-style-type: none"> - E1 tích cực cao, /E2 và /E3 phải tích cực thấp thì IC mới hoạt động, nếu không IC bị khoá (các chân Output = 1)

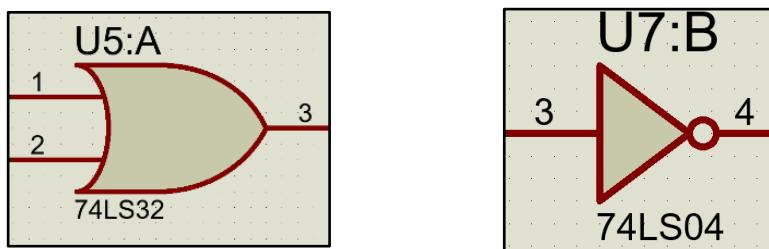
b. 74LS08 (AND), 74LS04 (NOT), 74LS32 (OR)

*Hình 1.6. 74LS08 trong Proteus*

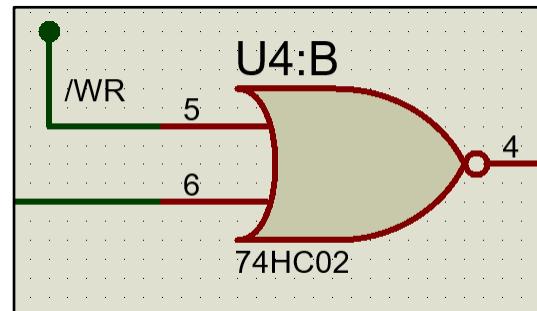
Bảng 1.4. Chức năng và kết nối các Pin 74LS08

Chân (pin)	Kiểu chân	Chức năng
Pin 1, Pin 2	Input	<ul style="list-style-type: none"> - 2 đầu vào một phép Logic số - Được sử dụng trung gian trong Khối giải mã. (Chi tiết kết nối được nói rõ trong phần giải mã địa chỉ)
Pin 3	Output	<ul style="list-style-type: none"> - Kết quả phép Logic AND - Kết nối trung gian

Các IC công khác tương tự như trên, chi tiết sẽ được nói tới sau.

*Hình 1.7. 74LS32 và 74LS04 trong Proteus*

c. 74HC02 (NOR)

*Hình 1.8. 74HC02 trong Proteus*

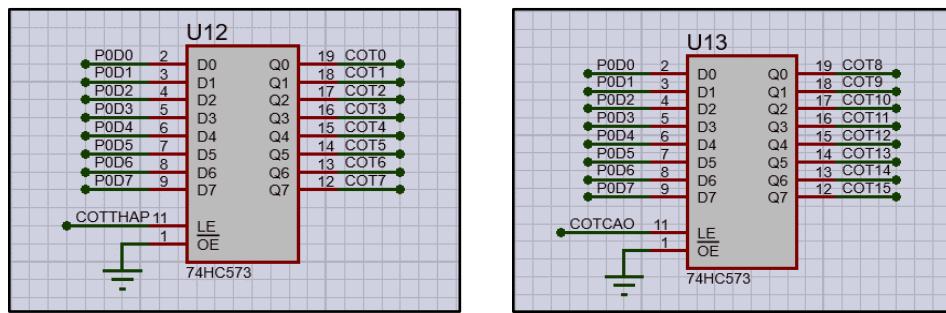
Bảng 1.5. Chức năng và kết nối các Pin 74HC02

Chân (pin)	Kiểu chân	Chức năng
Pin 5	Input	<ul style="list-style-type: none"> - Là 1 chân phép Logic số NOR. Kết nối với chân /WR. Khi ổn định dữ liệu thì, chân /WR tích cực cho phép các cổng này mở kèm theo tín hiệu chọn ngoại vi. - Mục tiêu kiểm soát dữ liệu luôn được ổn định.
Pin 6	Input	<ul style="list-style-type: none"> - Là 1 chân phép Logic số NOR. Được đưa tới từ các phép trung gian sau khi đã gần như hoàn thành giải mã với NOR là bước cuối cùng.
Pin 4	Output	<ul style="list-style-type: none"> - Kết quả phép Logic AND - Kết nối trung gian

1.2.4. Khối quét LED

Sau khi giải mã xong địa chỉ theo yêu cầu của đề bài, dữ liệu chọn chip được đưa từ khối giải mã sang khối quét LED bao gồm các bit chọn để xuất data điều khiển LED của cột cao, cột thấp, hàng cao, hàng thấp.

a. 74HC573

**Hình 1.9. 74HC573 trong Proteus**

Nhắc lại: IC 74HC573 là 1 vi mạch chốt dữ liệu với nguyên lý hoạt động:

Cho dữ liệu vào 8 chân đầu vào từ D0 → D7 khi chân chốt LE chưa hoạt động hay ở mức thấp thì trạng thái đầu ra chưa thay đổi. Khi chân LE lên mức cao thì dữ liệu đầu ra Q0 → Q7 sẽ ứng dữ liệu đầu vào tại thời điểm chốt (tạo ra 1 xung). Sau thời điểm chốt dữ liệu đầu ra sẽ không thay đổi khi dữ liệu đầu vào có thay đổi như thế nào đi nữa.

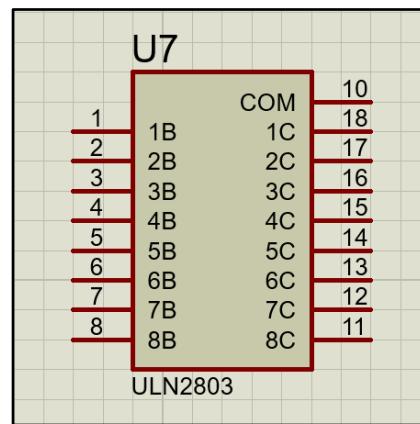
Trên thực tế, còn có IC 74HC373 có nguyên lý hoạt động tương tự. Ngoại trừ thông số hoạt động, '373 có bộ trí Pin khá kén mạch nên chúng ta không sử dụng ở đây.

Bảng 1.6. Chức năng và kết nối các Pin 74HC573

Chân (pin)	Kiểu chân	Chức năng
D0 → D7 (pin 2 → pin 9)	Input	- Các Pin nhận dữ liệu để chốt - Kết nối tới Port 0 để nhận dữ liệu
Q0 → Q7 (pin 19 → pin 12)	Output	- Dữ liệu được chốt chuyển sang chân Q này, không thể thay đổi trừ khi Ngắt /OE hoặc tích cực LE.
LE (pin 11)	Input	- Chân chốt LE mức thấp thì dữ liệu chốt không thể thay đổi - Được kết nối với các chân chọn ngoại vi của Khối giải mã địa chỉ đưa tới

/OE (pin 1)	Input	<ul style="list-style-type: none"> - Chân cho phép, /OE được tích cực thấp thì IC mới được phép hoạt động. Ngoài ra, ngõ ra tùy định. - Được nối đất để luôn cho '573 hoạt động.
----------------	-------	--

b. ULN2803



Hình 1.10. ULN2803 trong Proteus

ULN2803 hay được gọi với cái tên thân thuộc là IC đệm đảo. Để sử dụng, tác dụng lớn nhất IC cực kỳ phổ biến, nhất là các mạch sử dụng LED.

Trong mạch này ULN2803 được kết nối trung gian giữa dữ liệu xuất hàng.

Khi mô phỏng, thực ra không cần thiết tới ULN2803 thì ta cũng xem được tính chất mạch ra sao. Tuy nhiên, trên thực tế, khi tích cực cột và xuất dữ liệu ra hàng ở mức thấp, dòng đổ về từ LED lên IC Hàng là rất lớn (Từ Data Sheet chúng ta có thể lấy khoảng $20mA * 8 = 160mA$ nếu toàn bộ đều sáng). Việc này sẽ gây tổn hại đến IC, linh kiện trong mạch. Vì vậy, chúng ta mắc thêm ULN2803 với 2 mục đích chính:

- Bảo vệ IC trong mạch khỏi dòng đổ về (tối đa lên tới 500 mA)
- Đảo giá trị cần xuất ra hàng để tiện trong việc lập trình.

DataSheet ULN2803 được cho dưới đây.

BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18

PARAMETER		SYMBOL	RATINGS	UNIT
Input Voltage		V _{IN}	-0.5~30	V
Output Sustaining Voltage		V _{CE(SUS)}	-0.5~50	V
Output Current		I _{OUT}	500	mA/ch
Clamp Diode Reverse Voltage		V _R	50	V
Clamp Diode Forward Current		I _F	500	mA
Power Dissipation	DIP-18	P _D	1.47	W
	SOP-18		0.54/0.625 (Note)	
	SOP-20		0.56	
	TSSOP-20		0.52	
Operating Temperature		T _{OPR}	-40 ~ +85	°C
Storage Temperature		T _{STG}	-40 ~ +150	°C

Hình 1.11. DataSheet ULN2803

c. Điện trở hạn dòng cho LED

Điện trở này không ảnh hưởng đến việc mô phỏng Proteus. Tuy nhiên, ULN2803 trong Proteus không hoạt động đúng như dự kiến, vì nó cần điện trở để kéo lên (nếu không, giá trị đưa ra sẽ là tùy định). Vì vậy, ta có thể mắc thêm điện trở, thứ nhất là có thể giải quyết vấn đề trên của Proteus, thứ hai cũng là để thể hiện mạch có tính ứng dụng và thực tế cao.

(Vị trí của những điện trở này sẽ được thay đổi cho phù hợp, chi tiết xem MẠCH NGUYÊN LÝ).

- DataSheet của LED Ma Trận 1088AS được sử dụng

Manufacturer: OASIS RoHS: Yes Emitted Colour: Red Face Color: Black Type: Row Cathode Column Anode Wavelength : 625 ~ 630nm Forward Voltage : 2.1V ~ 2.5V Forward Current: 20mA Dimesions: 32mm x 32mm x 8.0mm

Hình 1.12. DataSheet LED Matrix 1088AS

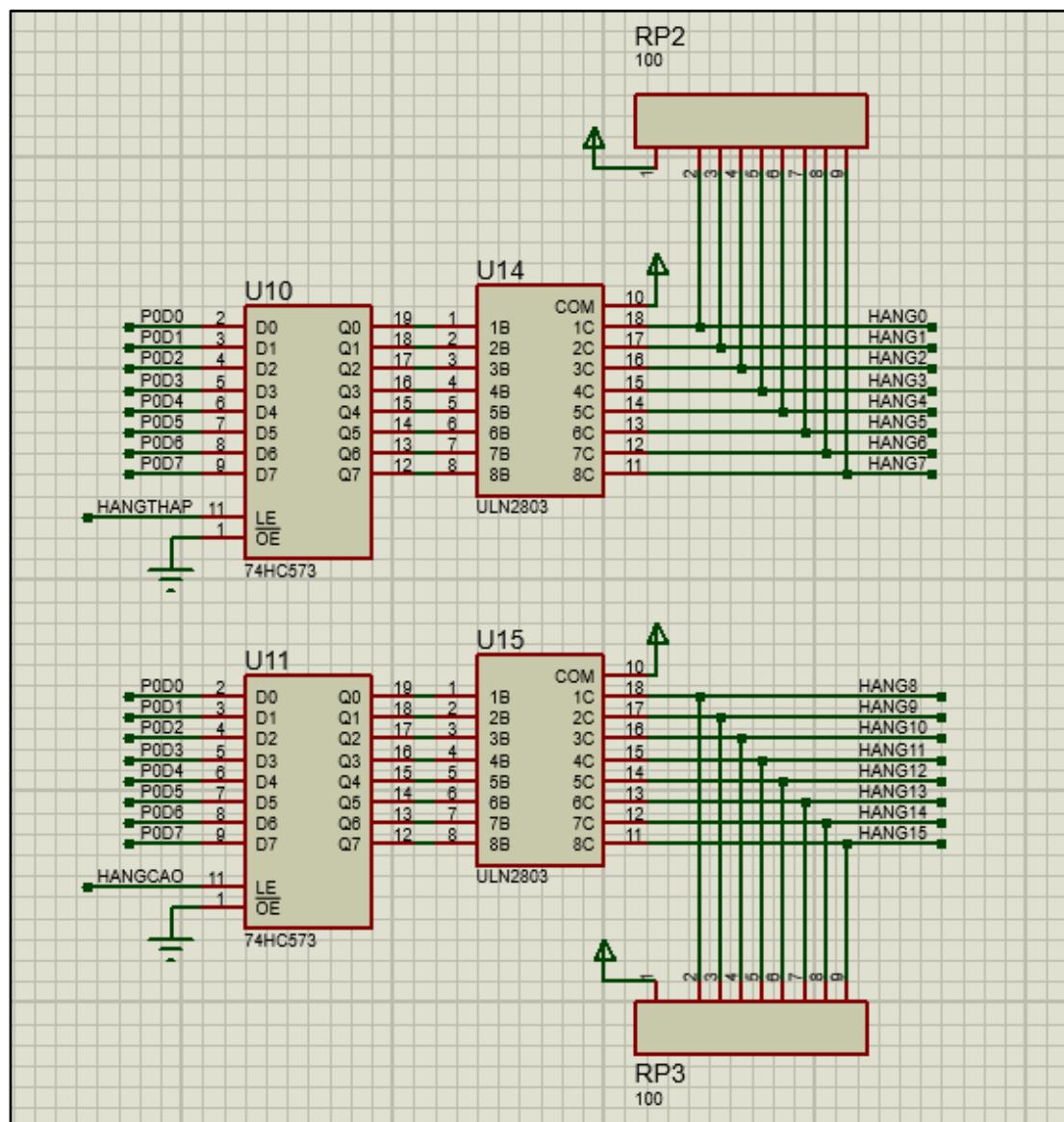
- Tính toán: Dựa trên DataSheet

- Điện áp hoạt động mạch khoảng 5V.

- Dòng của LED là 20mA và điện áp của LED sụt khoảng 2.1V – 2.5 V.

$$R = \frac{5 - (2.1 \div 2.5)}{20m} \approx 100\Omega \div 150\Omega$$

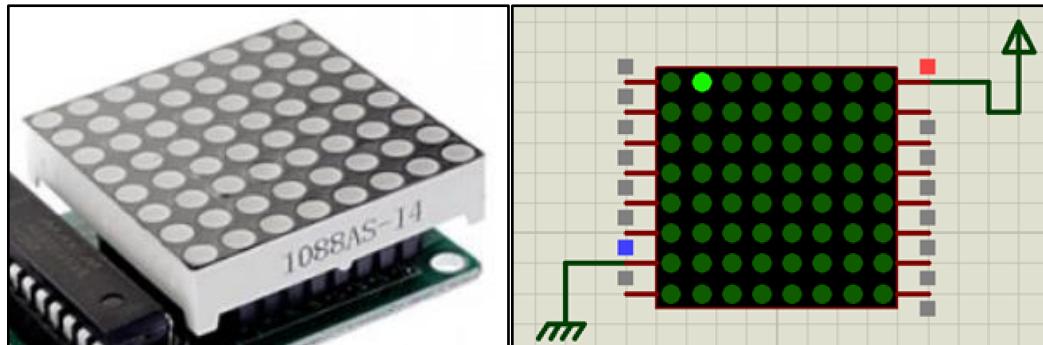
- Ta chọn 100Ω để có độ sáng rõ ràng, thuận tiện trong việc mua linh kiện.



Hình 1.13. 74HC573 được đệm bởi ULN2803 cùng điện trở

1.2.5. Khối hiển thị

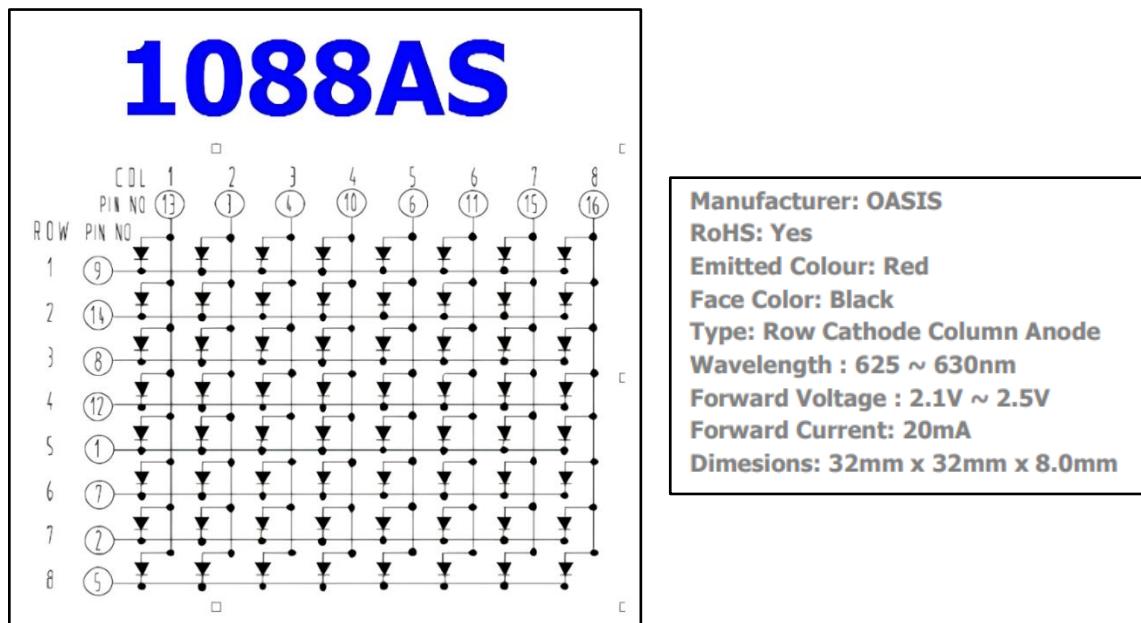
a. LED Ma Trận 1088AS



Hình 1.14. LED Ma Trận 1088AS thực tế và trong Proteus.

1088AS là LED ma trận 8x8 loại cột Anode, được sử dụng thường xuyên trong đòn sóng như mạch quang báo, mạch tín hiệu .v.v...

Bên trong cũng như thông số được cho trong DataSheet.



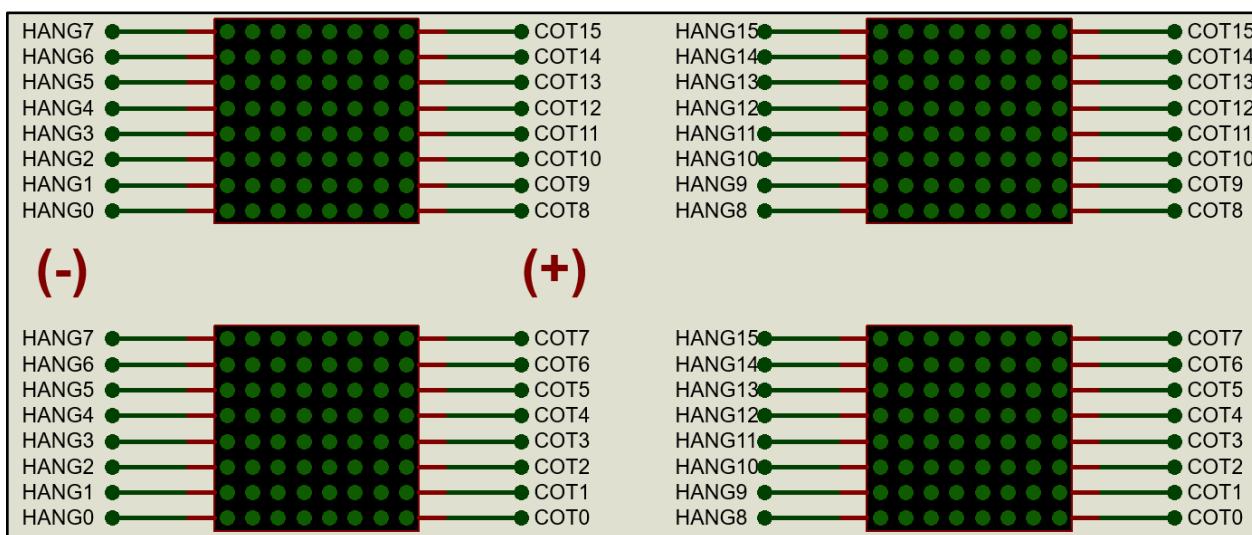
Hình 1.15. Mạch cầu tạo và DataSheet 1088AS

Để sáng LED, ta đưa vị trí cột lên mức 1, và đưa vị trí hàng xuống mức 0.

Bảng 1.7. Chức năng và kết nối Pin 1088AS

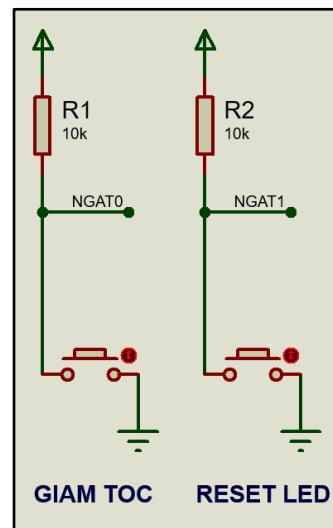
Chân (pin)	Kiểu chân	Chức năng
Pin 13, 3, 4, 10, 6, 11, 15, 16	Input	- Pin cột - Dữ liệu các chân này được đưa tới từ '573 có nhiệm vụ truy xuất cột.
Pin 9, 14, 8, 12, 1, 7, 2, 5	Input	- Pin hàng - Dữ liệu được đưa tới '573 hàng qua ULN 2803 để đếm đảo rồi mới xuất ra hàng.

Đề bài quy định: “Sinh viên thiết kế sao cho trọng số thấp của hàng nằm phía tay trái, trọng số thấp của cột nằm ở dưới.” Dựa vào đó, ta có các kết nối như sau.

*Hình 1.15. Kết nối 1088AS trong Proteus*

b. Bổ sung nút nhấn SLOW, RESET

Trên thực tế, 2 kết nối này là không cần thiết. Tuy vậy, để thuận tiện trong việc quan sát mô phỏng, Nhóm 2 xin được kết nối thêm 2 nút nhấn có tác dụng làm chậm tốc độ, và Reset gấp quá trình.

**Hình 1.16. Nút nhấn SLOW và Reset**

Nút nhấn được kết nối tới các chân Ngắt ngoài của MCU. Khi nhấn, dữ liệu tới chân Ngắt là tích cực thấp nên MCU sẽ thực hiện chương trình ngắt.

1.3. Thiết kế giải mã địa chỉ.

Bảng 1.8. Chia vùng địa chỉ đối với ngoại vi giao tiếp

Địa chỉ truy xuất	Ngoại vi giao tiếp
0000H - 17FFH	8 bit cao của cột
1800H - 3FFFH	8 bit thấp của cột
4000H - 8FFFH	8 bit cao của hàng
9000H - AFFFH	8 bit thấp của cột

Theo đề bài ta có bảng phân vùng:

Bảng 1.9. Chi tiết phân vùng địa chỉ

VÙNG ĐỊA CHỈ	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0000H	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X
	17FFH	0	0	0	1	0	X	X	X	X	X	X	X	X	X	X
1800H	0	0	0	1	1	X	X	X	X	X	X	X	X	X	X	X
	3FFFH	0	0	1	1	X	X	X	X	X	X	X	X	X	X	X
4000H	0	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X
	8FFFFH	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X
9000H	1	0	0	1	0	X	X	X	X	X	X	X	X	X	X	X
	AFFFFH	1	0	1	0	X	X	X	X	X	X	X	X	X	X	X

Xét các địa chỉ truy xuất, các ngõ từ A10 → A0 là tương tự cho các phân vùng.

Nên ta chọn các bit địa chỉ cao A15 → A11 làm các địa chỉ phân biệt các vùng với nhau.
Điều này cũng giải thích cho việc ở Khối MCU, chúng ta không cần thiết phải dùng ‘573 để chốt địa chỉ từ Port 0 nữa vì cơ bản là chúng ta không sử dụng tới.

3 bit địa chỉ A15, A14, A13 đưa vào ngõ vào của IC 74HC138, các ngõ ra của IC cùng với 2 ngõ A12 và A11 sẽ qua các cổng logic để được tín hiệu chọn vùng theo yêu cầu của đề bài.

Xét riêng IC 74HC138 được cấp 3 ngõ vào là 3 bit địa chỉ A15, A14, A13:

Bảng 1.10. Ngõ ra ‘138 dựa trên 3 ngõ vào được cấp

A15	A14	A13	Ngõ ra ‘138
0	0	0	/Y0
0	0	0	
0	0	1	
0	1	0	/Y2

0	1	1	/Y3
1	0	0	/Y4
1	0	0	
1	0	1	/Y5

Sau khi có các ngõ ra /Y0 → /Y5, dùng các công logic phù hợp để kết nối với 2 chân cắp bit A12 và A1 để tạo được tín hiệu chọn IC hoàn chỉnh.

- Xét vùng chọn CỘT CAO:

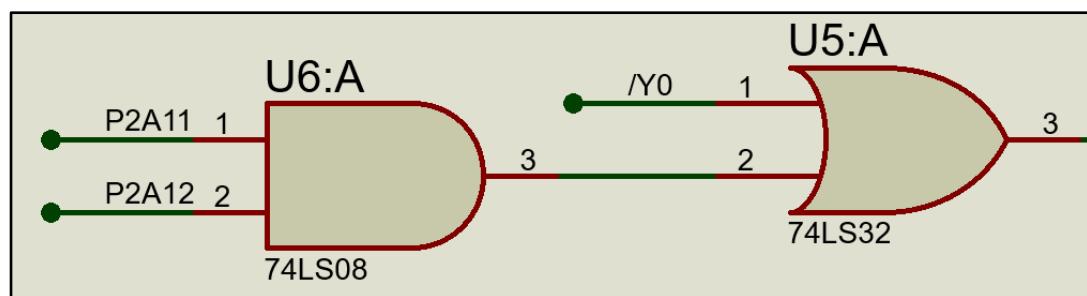
Bảng 1.11. Chi tiết vùng địa chỉ chọn CỘT CAO

Địa chỉ truy xuất	A15	A14	A13	A12	A11
0000H - 17FFH	0	0	0	0	0
	0	0	0	0	1
	0	0	0	1	0

Tín hiệu cần thiết ở đây là tích cực thấp nên từ bảng trên để có tín hiệu đầu ra như mong muốn thì /Y0 được chọn ($= 0$) và 2 BIT A12, A11 không được đồng thời bằng 1.

Biểu thức để thỏa điều kiện : $/Y_0 + A_{12} \cdot A_{11}$

Từ đó ta có sơ đồ các công logic:



Hình 1.17. Sơ lược công Logic vùng CỘT CAO

- Xét vùng chọn CỘT THẤP:

Bảng 1.12. Chi tiết vùng địa chỉ chọn CỘT THẤP

Địa chỉ truy xuất	A15	A14	A13	A12	A11
1800H – 3FFFH	0	0	0	1	1
	0	0	1	0	0
	0	0	1	0	1
	0	0	1	1	0
	0	0	1	1	1

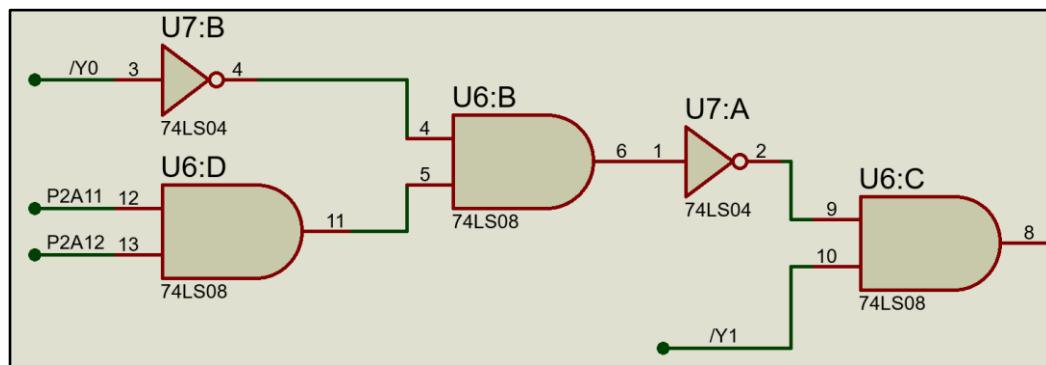
Tín hiệu cần thiết ở ngõ ra là tích cực thấp nên xét trong 2 trường hợp sau:

TH1: $/Y_0 = 0$ và 2 bit A11, A12 đồng thời bằng 1, khi đó ta có biểu thức để thỏa điều kiện là $/Y_0 + \bar{A}_{11} + \bar{A}_{12} = \overline{\bar{Y}_0 \cdot A_{11} \cdot Y_{12}}$

Hoặc

TH2: $/Y_1$ tích cực.

Từ đó ta có sơ đồ các cổng logic:

**Hình 1.18. Sơ lược các cổng logic vùng CỘT THẤP**

- Xét vùng chọn HÀNG CAO:

Bảng 1.13. Chi tiết vùng địa chỉ chọn HÀNG CAO

Địa chỉ truy xuất	A15	A14	A13	A12	A11
4000H – 8FFFH	0	1	0	0	0
	0	1	0	0	1
	0	1	0	1	0
	0	1	0	1	1
	0	1	1	0	0
	0	1	1	0	1
	0	1	1	1	0
	0	1	1	1	1
	1	0	0	0	0
	1	0	0	0	1

Tín hiệu cần thiết ở ngõ ra là tích cực thấp nên xét trong 3 trường hợp sau:

TH1: /Y2 tích cực thấp.

Hoặc

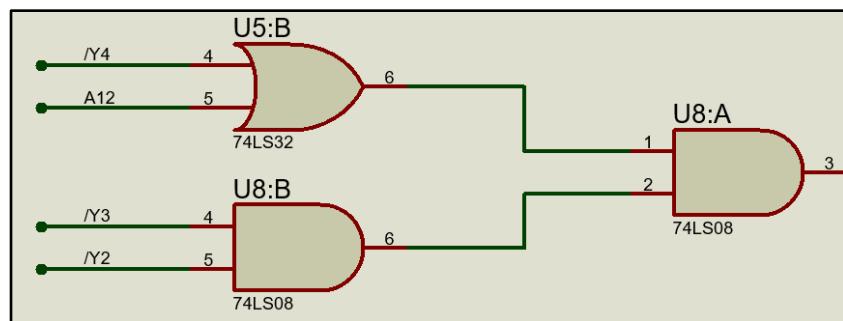
TH2: /Y3 tích cực thấp.

Hoặc

TH3: /Y4 tích cực thấp và A12=0.

Khi đó ta có biểu thức thỏa điều kiện : $(/Y_4 + A_{12}) \cdot (/Y_2 \cdot /Y_3)$

Từ đó ta có sơ đồ các công logic:



Hình 1.19. Sơ lược công Logic vùng HÀNG CAO

- Xét vùng chọn HÀNG THẤP:

Bảng 1.14. Chi tiết vùng địa chỉ chọn HÀNG THẤP

Địa chỉ truy xuất	A15	A14	A13	A12	A11
9000H – AFFFH	1	0	0	1	0
	1	0	0	1	1
	1	0	1	0	0
	1	0	1	0	1

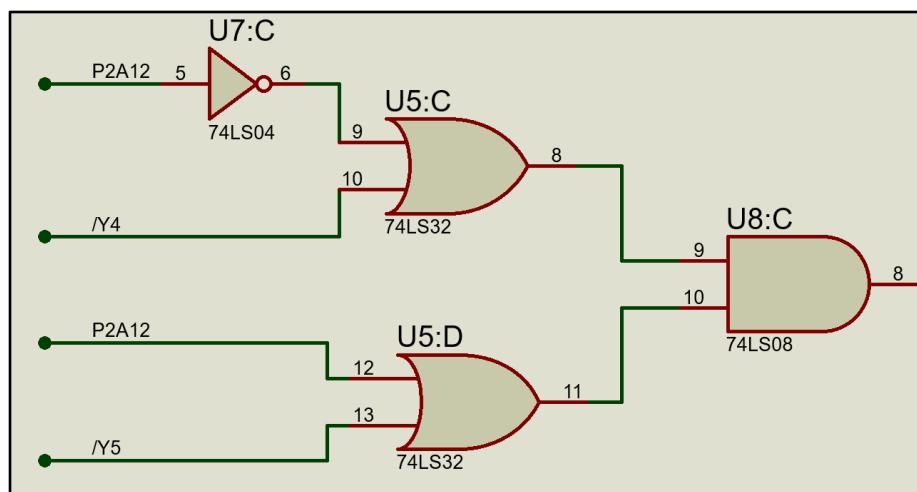
Tín hiệu cần thiết ở ngõ ra là tích cực thấp nên xét trong 2 trường hợp sau:

TH1: $/Y_4$ tích cực thấp và $A_{12} = 1$. Khi đó ta có biểu thức $/Y_4 + \overline{A_{12}}$

Hoặc

TH2: $/Y_5$ tích cực thấp và $A_{12} = 0$. Khi đó ta có biểu thức $/Y_5 + A_{12}$

Từ đó ta có sơ đồ cổng logic:

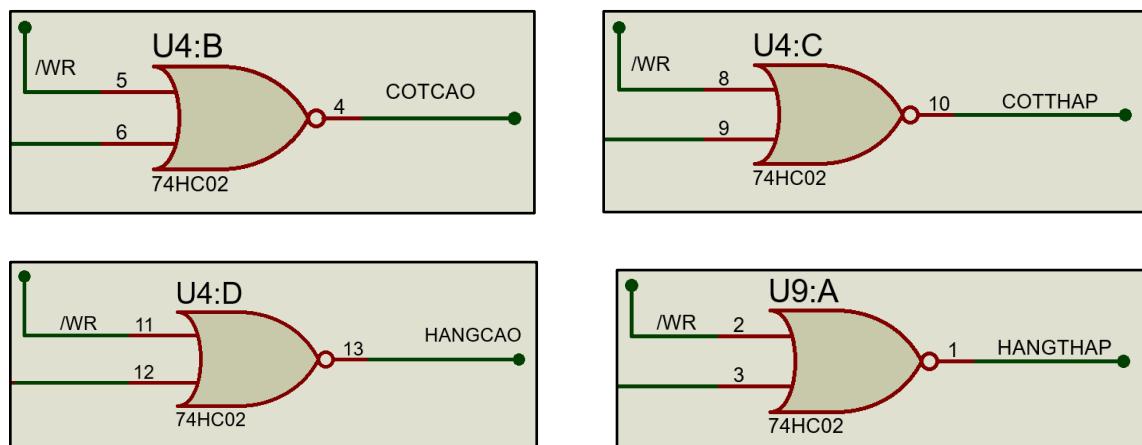
**Hình 1.20. Sơ lược cổng Logic vùng HÀNG THẤP**

Xét yêu cầu Khối giải mã:

- Tín hiệu đưa ra để chọn IC phải là tích cực cao (chân LE của IC ‘573).

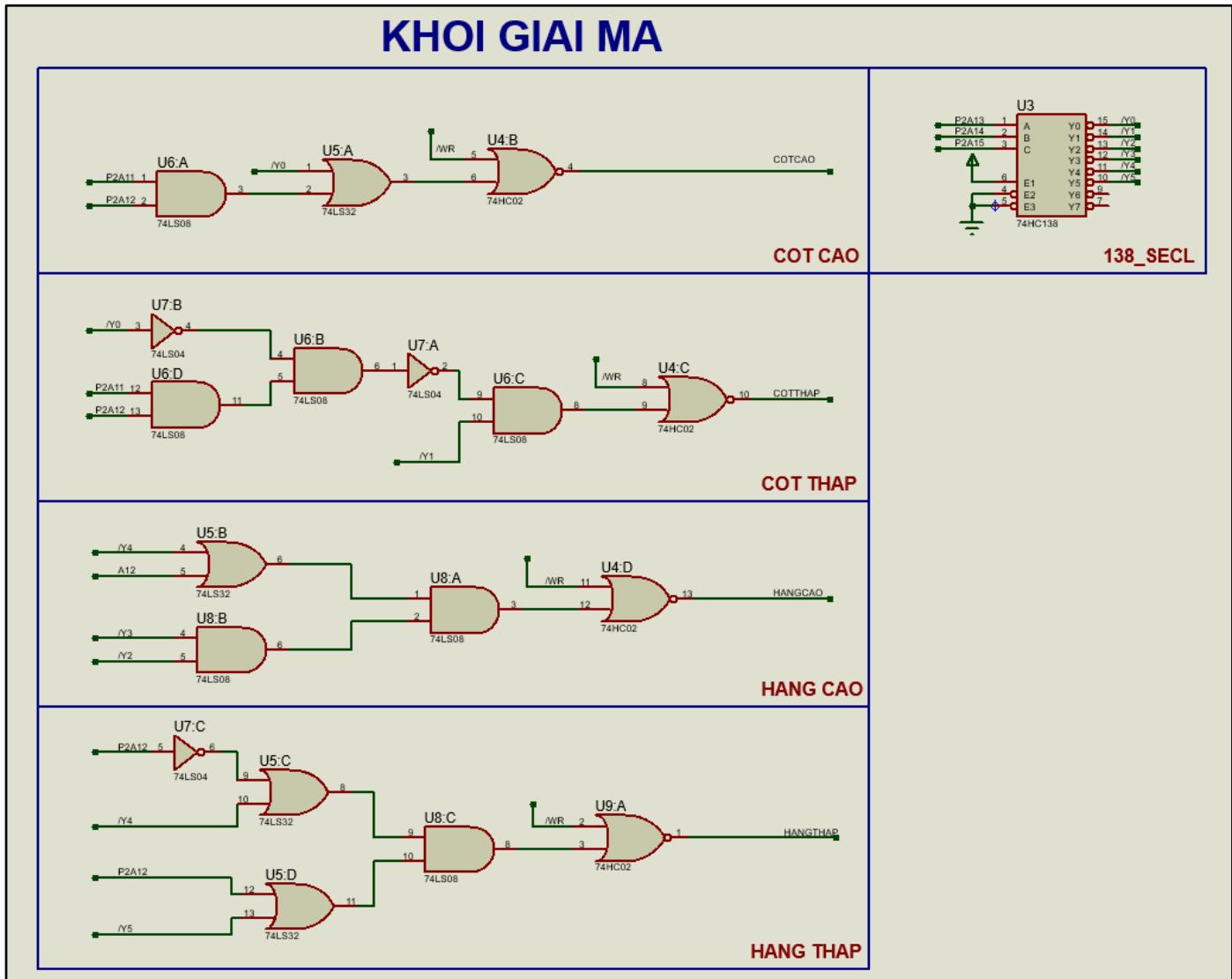
- Dữ liệu phải ổn định trong quá trình giải mã địa chỉ, truy xuất dữ liệu. Tức nghĩa là cần $/WR = 0$. Khi $/WR = 0$, dữ liệu đã ổn định trên bus dữ liệu và được ghi vào bộ nhớ hoặc thiết bị vào ra khi $/WR = 1$. Hơn nữa, lúc $/WR = 0$, ta phát dữ liệu để giải mã địa chỉ, tránh việc phát địa chỉ khi dữ liệu đang được ghi vào RAM ($/WR = 1$).

➔ Các tín hiệu phải đi qua cổng NOR với $/WR$ trước khi được chuyển đến chân LE '573 để chọn IC.



Hình 1.21. Tín hiệu cuối cùng cần phải được NOR với $/WR$

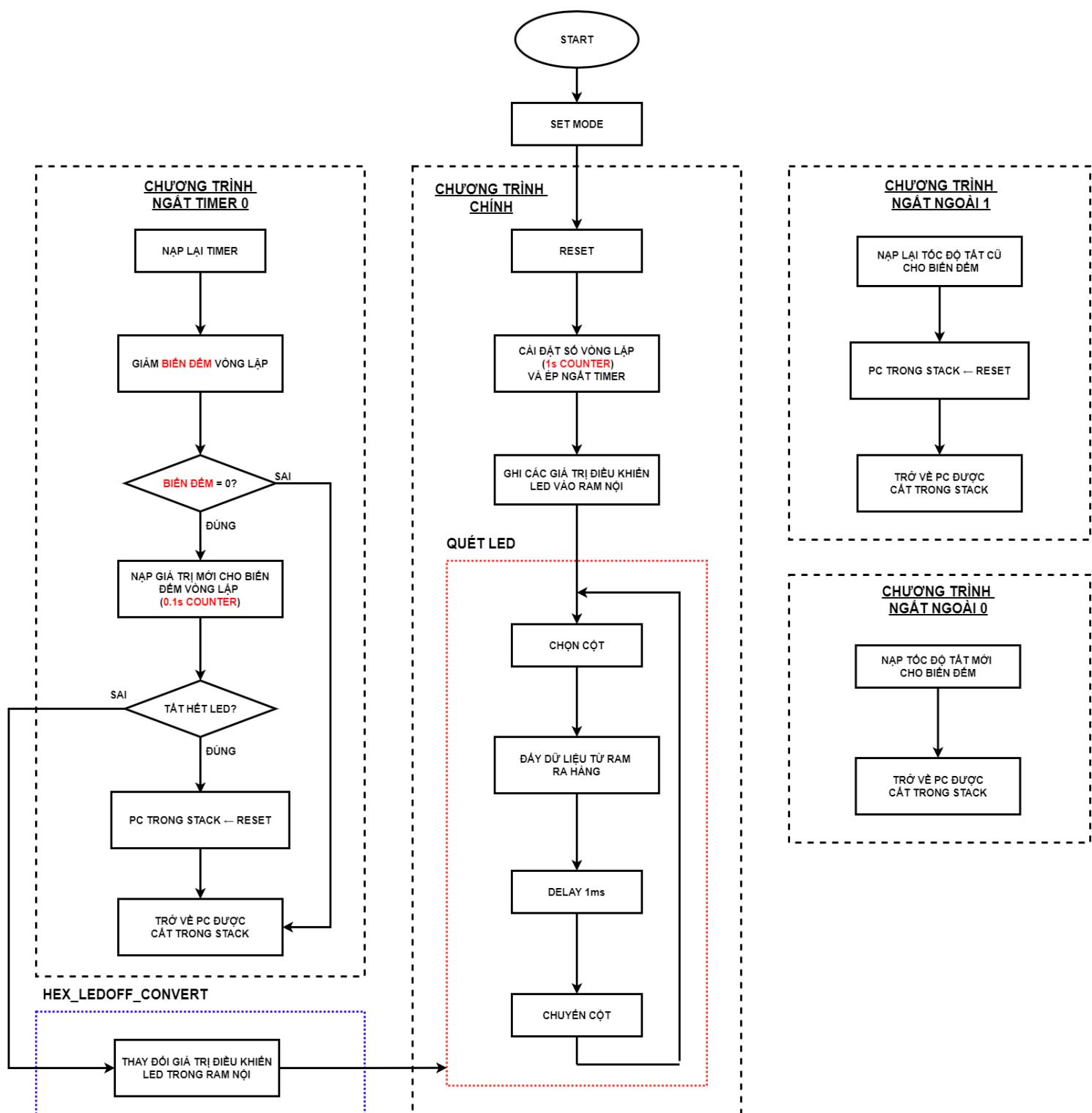
Cuối cùng, Khối giải mã được hoàn thành:



Hình 1.21. Khối giải mã hoàn chỉnh

CHƯƠNG 2. THIẾT KẾ PHẦN MỀM BẰNG KEILC - ASM

2.1. Lưu đồ giải thuật chương trình

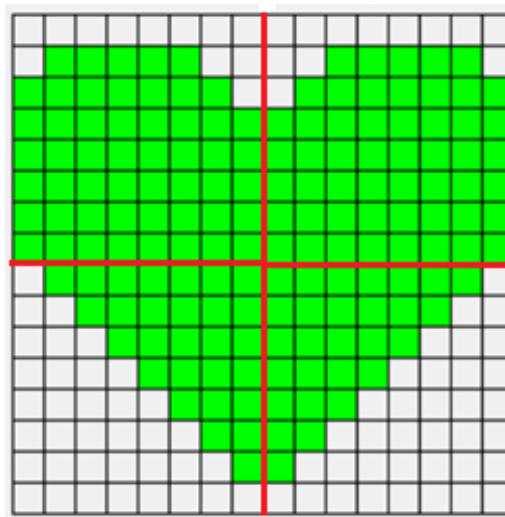


Hình 2.1. Lưu đồ giải thuật chương trình

2.1.1. Ý tưởng

LED Ma Trận có cách thức hoạt động khá tương tự LED 7 đoạn, mỗi lần chúng ta chỉ sáng được 1 phần điểm ảnh mà thôi. Vì vậy chúng ta sẽ sử dụng phương pháp Quét LED (sử dụng 1088AS - LED Ma Trận cột Anode).

Ý tưởng cho chương trình là sử dụng thuật toán bảng tra để lấy các giá trị mã hex và xuất lên LED ma trận, từ đó ta sẽ quét được hình trái tim như yêu cầu được đặt ra.



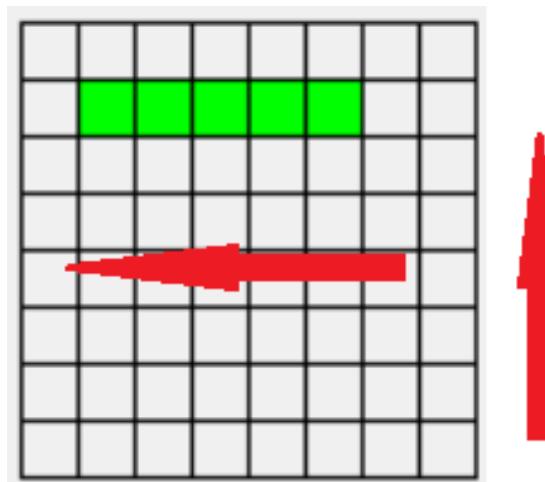
Hình 2.1. Hình trái tim được hiển thị trên 4 LED Ma trận 8x8

a. Cách thức sáng LED

Chương trình sẽ điều khiển 4 LED ma trận được sắp xếp để trọng số thấp của hàng nằm bên trái trái, trọng số thấp của cột nằm ở dưới. Mỗi lần quét chỉ xuất được 1 cặp dữ liệu ra cột và hàng, do đó ta phải sử dụng phương pháp quét LED. Phương pháp quét LED được sử dụng trong giải thuật này là “Phương pháp Quét Cột” (chọn cột, đẩy dữ liệu ra hàng).

Ta sẽ quét lần lượt từ cột thấp đến cột cao (từ dưới lên trên), từ hàng cao đến hàng thấp (từ bên phải sang bên trái). Tạo mã Hex điều khiển LED sáng bằng cách đặt bit cần sáng lên mức cao (1), nhìn từ phải sang trái và lần lượt các hàng từ dưới lên trên.

Ví dụ: Ở hình sau, để điều khiển LED sáng như hình (5 bit sáng ở hàng 7 từ dưới lên), ta cần đặt mã Hex 3EH ở vị trí thứ 7 của bảng tra, các vị trí còn lại mang mã Hex 00H để tắt. Tương tự với 4 LED ma trận, **tạo mã Hex từ phải sang trái, từ dưới lên trên, đặt ở vị trí từ phải sang trái, từ dưới lên trên.**

**Hình 2.2. Điều khiển LED Ma trận 8x8 hiển thị**

Mã Hex điều khiển LED sáng được tạo ở trong bảng tra *HEXTABLE* như sau:

HEXTABLE:									
;32H									
DB	000H, 000H,	001H, 080H,	003H, 0COH,	007H, 0EOH,	00FH, 0FOH,	01FH, 0F8H,	03FH, 0FCH,	07FH, 0FEH	
DB	OFFH, OFFH,	0FEH, 07FH,	07CH, 03EH,	000H, 000H	;				

Hình 2.3. Bảng tra mã Hex điều khiển 4 LED ma trận sáng hình trái tim

b. Cách thức tắt LED

Trước hết ta cần nắm rõ số LED phải tắt. Qua tính toán có khoảng 162 LED cần tắt.

Bảng tra *LEDCOUNT_TABLE* được tạo bằng cách đếm số điểm LED được sáng ở các hàng trong hình trái tim, đếm từ trên xuống, từ trái qua phải. Ví dụ: có 5 điểm sáng trên tầng trên cùng của trái tim ở LED ma trận bên trái, có 5 điểm sáng trên tầng trên cùng của trái tim ở LED ma trận bên phải, ở tầng tiếp theo tương tự có 7 điểm sáng ở LED ma trận bên trái, 7 điểm sáng ở LED ma trận bên phải...

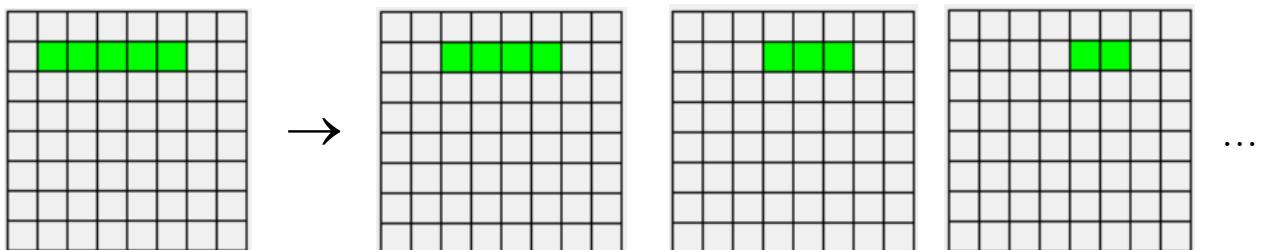
LEDCOUNT_TABLE:															
DB	5,5 ,	7,7 ,	8,8 ,	8,8 ,	8,8 ,	8,8 ,	8,8 ,	7,7 ,	6,6 ,	5,5 ,	4,4 ,	3,3 ,	2,2 ,	1,1	

Hình 2.4. Bảng tra số LED được sáng trên 4 LED ma trận sáng hình trái tim.

Từ đó đưa ra ý tưởng: Muốn tắt LED, ta phải thay đổi mã Hex quét sao cho mỗi lần thực hiện, bảng mã Hex sáng LED vào đáp ứng được việc có LED tắt.

Bảng tra *HEXOFFSETTABLE* dùng để điều khiển LED tắt dần, được tạo ra theo thứ tự ngược

lại với khi điều khiển LED sáng, lần lượt từng hàng ở mỗi LED ma trận, theo thứ tự từ trái qua phải, từ trên xuống dưới. Ví dụ: để điều khiển 5 điểm sáng mang mã 3EH tắt dần, ta tạo lần lượt các mã Hex 3CH, 38H, 30H,... **Sau khi điểm sáng ở hàng này tắt hết, chuyển sang hàng khác, theo thứ tự trái qua phải, trên xuống dưới.**



Hình 2.5. Điều khiển LED tắt dần điểm ảnh

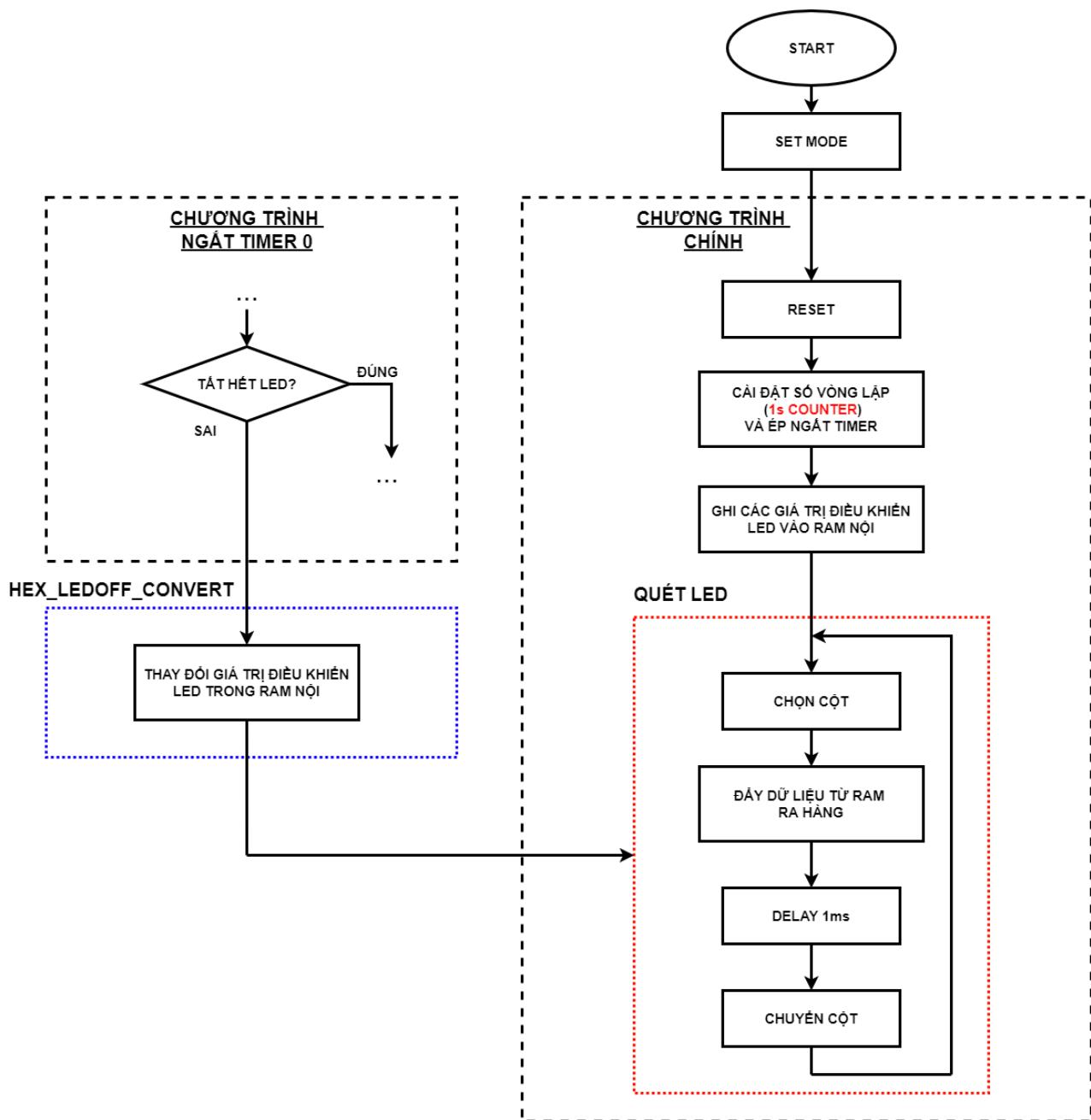
HEXOFFTABLE:
DB 03CH, 038H, 030H, 020H, 000H, 078H, 070H, 060H, 040H, 000H
DB 07EH, 07CH, 078H, 070H, 060H, 040H, 000H, OFCH, 0F8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H
DB 0FEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, 07EH, 07CH, 078H, 070H, 060H, 040H, 000H
DB 0F8H, OFOH, OEOH, OC0H, 080H, 000H, 03EH, 03CH, 038H, 030H, 020H, 000H
DB OFOH, OEOH, OC0H, 080H, 000H, 01EH, 01CH, 018H, 010H, 000H
DB OEOH, OC0H, 080H, 000H, 00EH, 00CH, 008H, 000H
DB OC0H, 080H, 000H, 006H, 004H, 000H
DB 080H, 000H, 002H, 000H
DB 000H, 000H

Hình 2.6. Bảng tra dùng để thay đổi mã Hex cho LED tắt dần

Tuy nhiên khi sử dụng bảng tra, các giá trị mã Hex sẽ được lưu trong bộ nhớ chương trình ROM và **không thể thay đổi được → khó thực hiện ý tưởng trên**. Do đó thay vì truy xuất trực tiếp từ bảng tra trong ROM, trước khi quét LED, ta sẽ ghi các giá trị mã Hex vào trong bộ nhớ RAM để có thể thay đổi các giá trị đó khi cần (cụ thể là khi tắt dần các điểm LED) và quét các giá trị được lưu trong RAM thay vì ROM. Điều này còn thuận tiện cho việc phục hồi dữ liệu điều khiển LED khi Reset chương trình.

Ngoài ra chương trình còn có thêm một số tùy chọn được thêm vào như giảm tốc độ tắt dần các LED và Reset gấp lại quá trình hiển thị về ban đầu.

2.2. Sơ đồ giải thuật chương trình chính



Hình 2.7. Sơ đồ giải thuật chương trình chính

2.2.1. Chi tiết giải thuật

a. Chỉ thị tiền xử lý

Trước khi bắt đầu chương trình, để thuận tiện trong việc điều chỉnh giá trị hay Debug, ta định nghĩa các giá trị như sau:

- Theo yêu cầu, địa chỉ truy xuất bit cao của cột nằm trong khoảng từ 0000H – 17FFH, ta chọn 1000H là địa chỉ giao tiếp với 8 bit cao của cột, định nghĩa cho biến *TruyXuatCOTCAO* = **1000H**. Tương tự với các địa chỉ truy xuất còn lại, *TruyXuatCOTTHAP* = **1F00H** là địa chỉ giao tiếp với 8 bit thấp của cột, *TruyXuatHANGCAO* = **4000H** là địa chỉ giao tiếp với 8 bit cao của hàng, *TruyXuatHANGTHAP* = **9000H** là địa chỉ giao tiếp với 8 bit thấp của hàng.

- Định nghĩa các địa chỉ ô nhớ RAM nội:

- + *CHONCOT* = **21H** dùng để chọn cột quét,
- + *LEDCOUNTER* = **25H** dùng để đếm số LED còn sáng
- + *TOCDOTAT* = **26H** dùng để điều chỉnh tốc độ tắt các điểm LED
- + *POINTER_OFFSETTABLE* = **27H** dùng để làm biến con trỏ ghi dữ liệu điều khiển LED mới vào RAM nội.
- + *CASEOFF* = **28H** dùng làm biến đếm trường hợp khi tắt LED
- + *POINTER_DATARAM* = **29H** dùng làm biến con trỏ ghi dữ liệu ban đầu điều khiển LED sáng hình trái tim vào RAM nội.

```

TruyXuatCOTCAO    EQU      1000H
TruyXuatCOTTHAP   EQU      1F00H
TruyXuatHANGCAO   EQU      4000H
TruyXuatHANGTHAP  EQU      9000H
;-----DINH NGHIA O RAM NOI-----
CHONCOT           EQU      21H      ;CHON COT DE QUET
LEDCOUNTER         EQU      25H      ;SO LED CON LAI CHUA TAT
TOCDOTAT          EQU      26H      ; = 10 <=> 0.1S
POINTER_OFFSETTABLE EQU      27H      ;VI TRI HIEN TAI TRONG BANG TAT LED
CASEOFF            EQU      28H
; 1 CASEOFF <=> 1/2 HANG TAT (HANG CAO HOAC THAP) THUOC 1 COT
; <=> CO 28 CASE, 28 TRUONG HOP
POINTER_DATARAM   EQU      29H      ;CON TRO SU DUNG DE CHUYEN DATA VAO RAM NOI

```

Hình 2.8. Định nghĩa các giá trị trước khi bắt đầu chương trình

b. Thuật toán trong chương trình chính – Hiển thị LED

Khi bắt đầu chương trình, chương trình sẽ cài đặt tốc độ tắt LED mặc định (nạp giá trị cho biến *TOCDOTAT* là 10).

SET_MODE: MOV JMP	TOCDOTAT, # 10 MAIN
--------------------------------	-------------------------------

Hình 2.9. Đoạn chương trình SET_MODE

Sau đó sẽ vào chương trình chính, bắt đầu **Reset** các giá trị thanh ghi STACK SP^1 , biến đếm trường hợp tắt LED *CASEOFF*, con trỏ chỉ vị trí hiện tại trong bảng tắt LED *POINTER_OFFTABLE* và biến đếm số LED sáng *LEDCOUNTER*.

RESET:	
MOV	SP, #07H
MOV	CASEOFF, #0
MOV	POINTER_OFFTABLE, #0
MOV	LEDCOUNTER, #0

Hình 2.10. Đoạn chương trình RESET.

Kế đó, **khởi động Timer**, nạp giá trị cho biến đếm vòng lặp là $R4 \leftarrow 100$ (100 lần ngắt timer $0.01s = 1s$), ép ngắt timer để vào chương trình ngắn, nạp giá trị và chạy Timer.

MOV	TMOD, #01H
MOV	R4, #100
MOV	R5, #1
SETB	EA
SETB	TF0
SETB	ET0 ; CHO NGATTIMERO

Hình 2.11. Đoạn chương trình khởi động Timer.

Tiếp đến, chương trình thực hiện **ghi các giá trị mã Hex** dùng để quét LED sáng hình trái tim từ bảng tra *HEXTABLE* vào các địa chỉ từ 30H đến 4FH, và ghi số điểm LED sáng ở mỗi hàng ở 4 LED ma trận từ bảng tra *LEDCOUNT_TABLE* vào các địa chỉ từ 50H tới 6BH.

HEARTDATA:	
MOV	A, POINTER_DATARAM
MOV	DPTR, #HEXTABLE
MOVC	A, @A+DPTR
MOV	@R0, A; GHI DATA VAO RAM NOI
MOV	A, POINTER_DATARAM
MOV	DPTR, #LEDCOUNT_TABLE
MOVC	A, @A+DPTR
MOV	@R1, A; GHI SO LED SANG VAO RAM NOI
INC	R1
INC	R0
INC	POINTER_DATARAM
CJNE	R0, #50H, HEARTDATA

Hình 2.13. Đoạn chương trình ghi dữ liệu bảng tra vào RAM nội.

¹ Sử dụng các lệnh thay đổi PC (JMP, CALL,...) hay các lệnh sử dụng STACK làm nơi lưu trữ tạm thời (PUSH, POP) có thể làm lấp đầy dữ liệu trong thanh ghi STACK khi lặp đi lặp lại nhiều chu kỳ của chương trình. Khi đó, chương trình sẽ không còn chạy ra đúng kết quả mà ta mong muốn. Để giải quyết vấn đề này, trước khi bắt đầu một chu kỳ mới, ta nên Clear bộ STACK bằng cách đưa con trỏ STACK về lại 07H.

Tiếp tục, chương trình sẽ vào một vòng lặp vô tận, trong vòng lặp này làm nhiệm vụ **liên tục** quét LED ma trận:

QUETMATRAN:	
MOV	CHONCOT, #000000001B
MOV	R2, #8
LAPQUET:	
PUSH	83H
PUSH	82H
MOV	A, CHONCOT
MOVX	@DPTR, A
MOV	A, @R1
MOV	DPTR, #TruyXuatHANGCAO
MOVX	@DPTR, A
INC	R1
MOV	A, @R1
MOV	DPTR, #TruyXuatHANGTHAP
MOVX	@DPTR, A
INC	R1
CALL	DELAY
POP	82H
POP	83H
MOV	A, CHONCOT
RL	A
MOV	CHONCOT, A
DJNZ	R2, LAPQUET
MOV	A, #00000000B
MOVX	@DPTR, A
RET	

Hình 2.13. Đoạn chương trình dùng để quét LED

- **Truy xuất cột thấp:** thực hiện đầy dữ liệu ra hàng từ cột 0 đến 7, ở mỗi lần đầy, chọn cột cần cho sáng bằng cách đặt bit của biến *CHONCOT* (địa chỉ 21H RAM) tương ứng lên tích cực mức cao (1).

Ví dụ: 0000 1000 tương ứng với chọn cột 3.

Xuất giá trị này bằng cách đưa vào địa chỉ giao tiếp của cột thấp (*TruyXuatCOTTHAP*) bằng

lệnh MOVX.

Sau đó đưa dữ liệu vào địa chỉ giao tiếp của hàng cao (*TruyXuatHANGCAO*), hàng thấp (*TruyXuatHANGTHAP*). Dữ liệu là các giá trị mã Hex tương ứng được lấy từ bảng tra (từ các địa chỉ RAM **30H** đến **4FH**).

- **Truy xuất cột cao:** tương tự với truy xuất cột thấp, thực hiện đầy dữ liệu ra hàng từ cột 8 đến 15, ở mỗi lần đầy, chọn cột cần cho sáng bằng cách đặt bit của biến *CHONCOT* (địa chỉ 21H RAM) tương ứng lên tích cực mức cao (1).

Ví dụ: 0100 000 tương ứng với chọn cột 14.

Xuất giá trị này bằng cách đưa vào địa chỉ giao tiếp của cột cao (*TruyXuatCOTCAO*) bằng lệnh **MOVX**.

Sau đó đưa dữ liệu vào địa chỉ giao tiếp của hàng cao (*TruyXuatHANGCAO*), hàng thấp (*TruyXuatHANGTHAP*). Dữ liệu là các giá trị mã Hex tương ứng được lấy từ bảng tra (từ các địa chỉ RAM **30H** đến **4FH**).

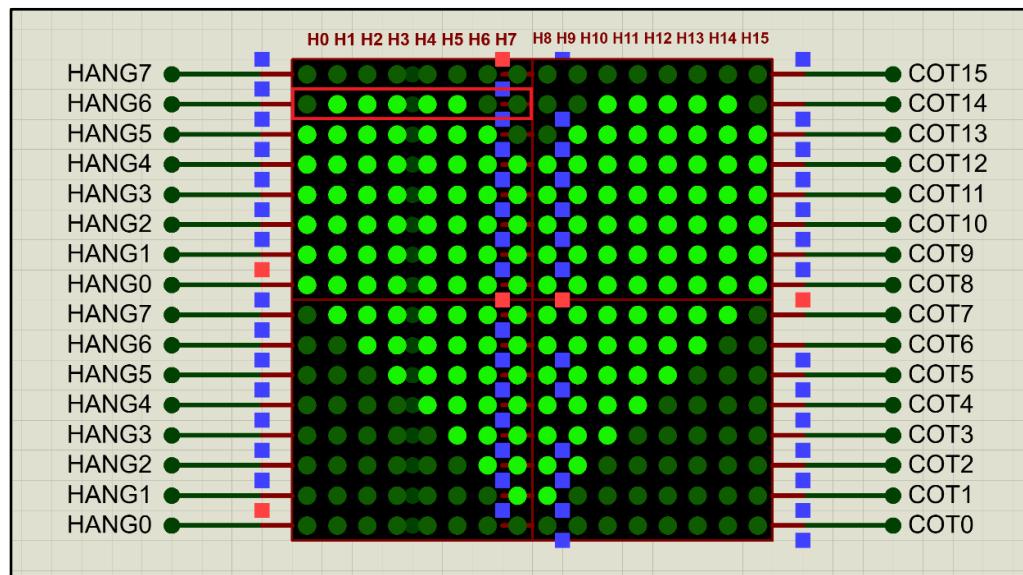
Sau khi thực hiện 1 lượt quét (quét đủ 16 cột) thì cần phải tắt biến *CHONCOT* (tích cực mức thấp toàn bộ các bit cột) để tránh hiện tượng **Ghosting**².

Khi đã chạy đủ 100 lần *Ngắt Timer 0* (LED ma trận đã hiển thị hình trái tim đủ 1 giây) thì nạp giá trị cho biến *đếm vòng lặp* là 10 (0.1 giây).

Sau đó, mỗi khi Timer chạy đủ 0.1s thì các giá trị dùng để quét quét LED được lưu trong các địa chỉ từ **30H** đến **4FH** sẽ thay đổi để hình trái tim tắt dần. Cụ thể ví dụ như:

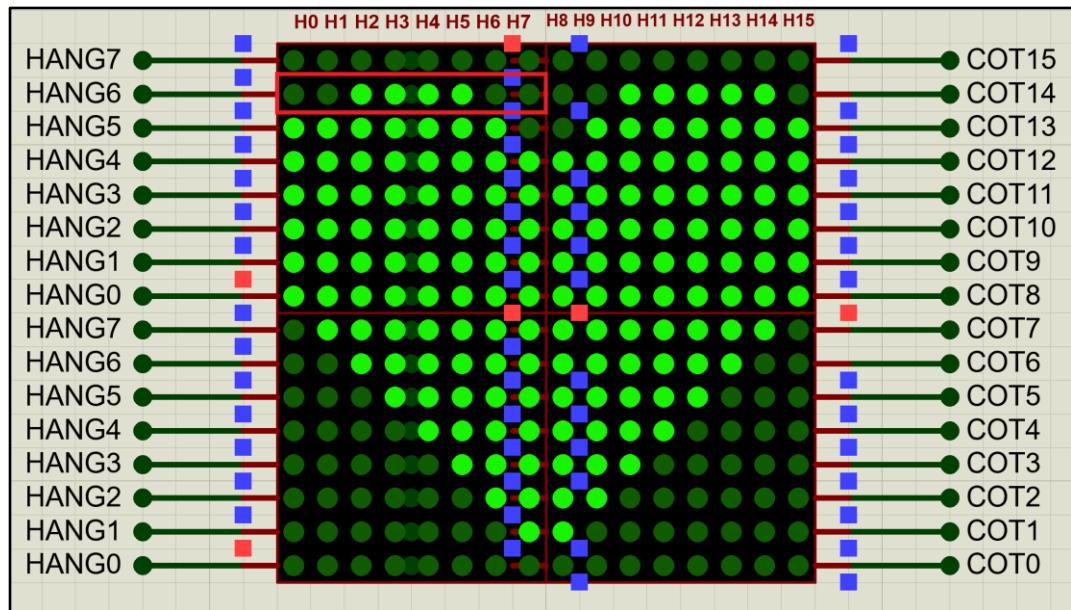
- Ở địa chỉ **4DH**, mang giá trị mã Hex 3EH, thì sẽ có 5 điểm LED sáng như hình:

² Ghosting: là hiện tượng thường gặp của các phương pháp quét LED, xảy ra khi bị chèn giữa dữ liệu được xuất và dữ liệu hàng (cột) được chọn để quét hay còn thường gọi với tên là *lem điểm ảnh – bóng ma*. Để tránh hiện tượng này, ta phải tắt LED để tránh việc hiển thị sai lệch trong quá trình quét.

*Hình 2.14. Điều khiển hiển thị LED bằng mã Hex 3EH*

c. Thuật toán trong chương trình chính – Tắt LED

- Khi tắt dàn LED từ trái sang phải, từ trên xuống dưới, thì ta cần **thay đổi mã Hex** trong 4DH từ 3EH thành 3CH, khi đó sẽ còn 4 điểm LED sáng như hình:

*Hình 2.14. Điều khiển hiển thị LED bằng mã Hex 3CH*

Trong khi lần lượt thay đổi giá trị mã Hex thì LED ma trận liên tục quét để hiển thị hình trái tim mới. Sau mỗi 0.1s sẽ là 1 bộ mã hex mới để quét và hiển thị, mỗi 1 bộ mã hex ta cho hiển thị

ít hơn 1 điểm LED sáng, do đó ta sẽ thấy các hình trái tim tắt dần từng điểm sáng như yêu cầu được đặt ra.

Ta sẽ lần lượt tắt các hàng ở trên mỗi LED ma trận, có 28 hàng trên 4 LED ma trận là có điểm sáng, do đó sẽ có 28 trường hợp (case). Ở mỗi trường hợp, **đưa dữ liệu mới** ở bảng tra *HEXOFFTABLE* vào RAM nội, bắt đầu từ địa chỉ 4DH.

Sau khi tắt hết điểm LED sáng trên hàng đó sẽ **chuyển sang trường hợp khác** bằng, sử dụng biến *CASEOFF* để lưu giá trị thứ tự các trường hợp.

MOV	R0, #4DH; DIA CHI CHUA LED DAU` TIEN CAN TAT
-----	--

```

HEX_LED OFF_CONVERT:
    PUSH          01H
    MOV           A, POINTER_OFFTABLE
    MOV           DPTR, #HEXOFFTABLE
    MOVC          A, @A+DPTR
    MOV           @R0, A; GHI DATA MOI VAO RAM NOI
    INC           POINTER_OFFTABLE

    MOV           A, #50H
    ADD           A, CASEOFF
    MOV           R1, A
    MOV           A, @R1; GHI SO LED CON SANG VAO RAM NOI
    MOV           LEDCOUNTER, A

    DJNZ          LEDCOUNTER, EXIT_RET
    DEC           R0
    INC           CASEOFF

EXIT_RET:
    MOV           @R1, LEDCOUNTER
    POP           01H
    RET

```

Hình 2.14. Đoạn chương trình thay đổi dữ liệu trong RAM nội.

Để biết số điểm LED cần tắt ban đầu trong mỗi trường hợp, ta sẽ truy xuất các địa chỉ có giá trị từ 50H đến 6FH (nơi ghi các giá trị số LED sáng từ bảng *LEDCOUNT_TABLE*) và ghi vào biến *LEDCOUNTER*.

Mỗi 0.1s trôi qua, thì số LED sáng giảm xuống 1 (giảm biến *LEDCOUNTER* đi 1), khi số LED sáng bằng 0 thì chuyển sang trường hợp khác bằng cách tăng biến *CASEOFF* và giảm biến

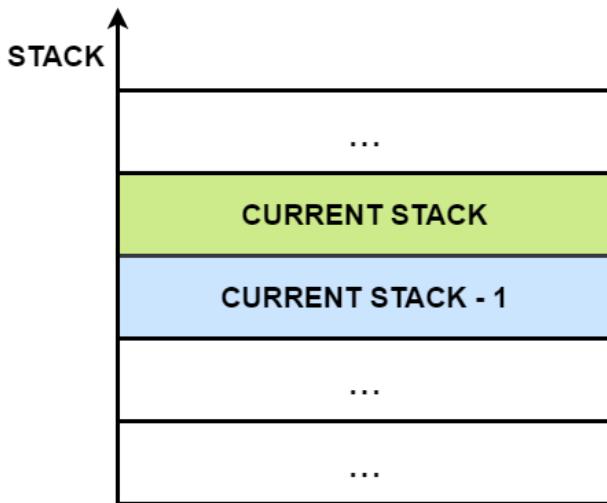
R0 (vị trí chứa mã Hex của Case đó).

Ví dụ: Khi ở case 6 thì truy xuất số LED còn sáng bằng cách truy xuất vào địa chỉ $56H = 50H + CASEOFF = 50H + 6$.

HEXTABLE:	
	;32H
DB 000H, 000H, 001H, 080H, 003H, 0C0H, 007H, 0E0H, 00FH, 0F0H, 01FH, 0F8H, 03FH, 0FC0H, 07FH, 0FEH	DB OFFH, OFFH ;4DH

Hình 2.15. Chiều thay đổi dữ liệu trong RAM nội.

Sau khi chạy qua hết 28 trường hợp ($CASEOFF = 0 \div 27$, kiểm tra biến này ở chương trình ngắt Timer – sau khi đủ 0.1 giây) thì LED đã **tắt hết**, để **quay trở về ban đầu** ta sẽ thay đổi giá trị PC được lưu trong stack SP về giá trị địa chỉ *RESET*, và dùng lệnh RETI hoàn thành ngắt để quay về chương trình chính và lặp lại một chu kì mới.³



PC₁₅₋₈ của chương trình chính $\leftarrow \#HIGH(RESET)$

PC₇₋₀ của chương trình chính $\leftarrow \#LOW(RESET)$

Hình 2.16. Phương pháp reset chương trình

³ Khi được đáp ứng ngắt và nhảy sang chương trình ngắt, PC lệnh kế tiếp của chương trình chính được lưu ở trong 2 ô nhớ tiếp theo của thanh ghi SP và được nạp vào lại PC hiện hành sau khi hoàn thành ngắt bằng lệnh RETI.

Do đó trước khi hoàn thành ngắt, ta thay đổi PC lệnh kế tiếp đã được cất trong stack bằng PC của địa chỉ mà ta muốn chương trình tiếp tục ở đó rồi mới dùng RETI hoàn thành ngắt để điều hướng chương trình theo mong muốn.

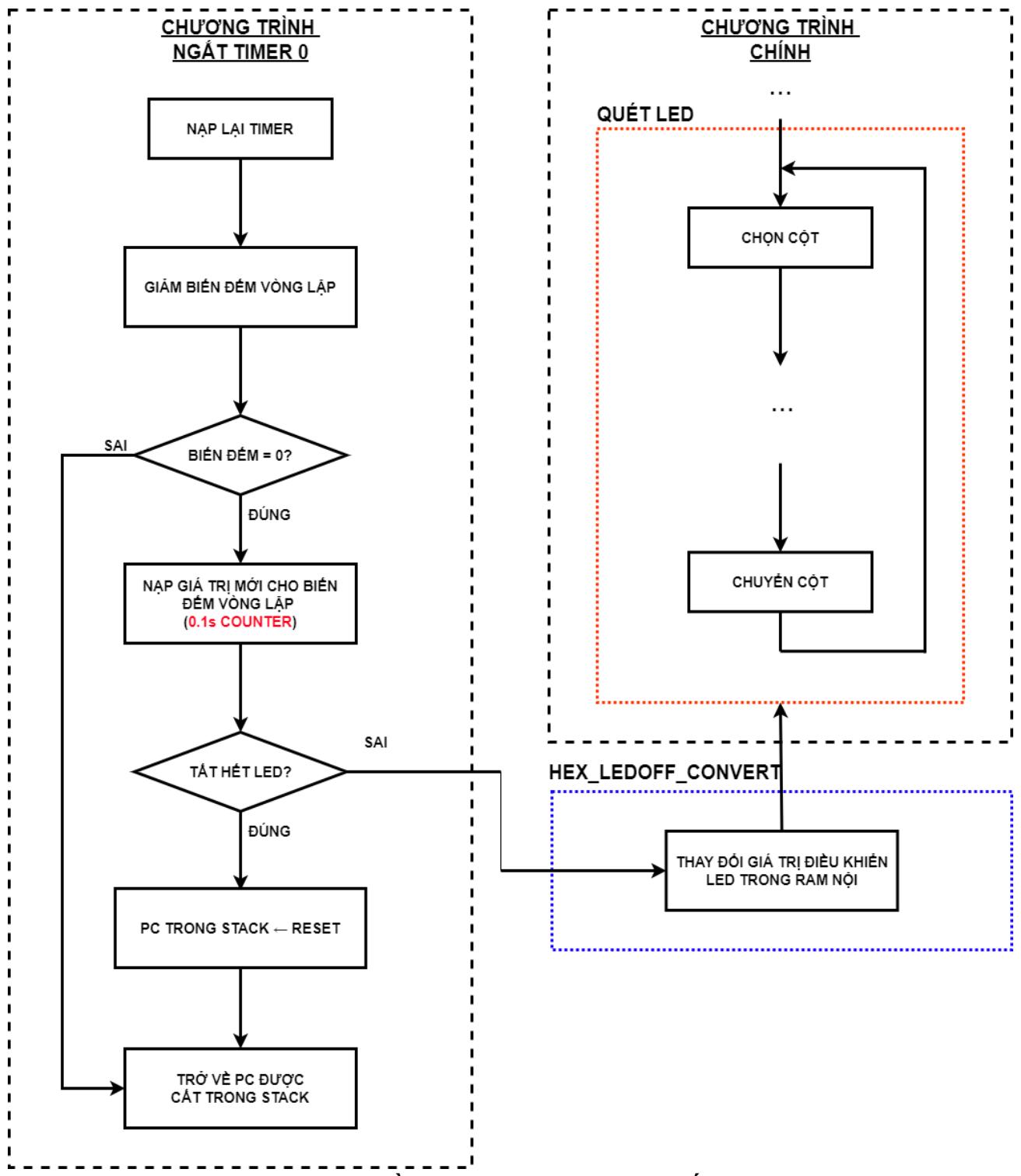
BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18

```
MOV      A, CASEOFF
CJNE    A, #28, NEXT
;NEU (28H) = 28 THI DA TAT XONG
MOV      R0, SP
MOV      @R0, #HIGH(RESET) ;SP SAU CAT BYTE CAO
DEC     R0
MOV      @R0, #LOW(RESET)  ;SP TRUOC CAT BYTE THAP
RETI
NEXT:
CALL    HEX_LEDOFF_CONVERT
EXIT:  RETI
```

Hình 2.17. Đoạn chương trình kiểm tra trường hợp.

2.3. Sơ đồ giải thuật các chương trình ngắn

2.3.1. Ngắt timer 0



Hình 2.18. Sơ đồ giải thuật chương trình Ngắt Timer 0

Chương trình ngắt timer được nạp giá trị để cho Timer chạy đủ 0.01s thì cờ TF0 sẽ báo ngắt, do đó ta sử dụng 2 biến đếm vòng lặp $R4 = 100$ (100 lần $0.01s = 1s$) để sử dụng cho điều khiển LED sáng hình trái tim trong vòng $1s$, biến $R5 = 10$ (10 lần $0.01s = 0.1s$) để để sử dụng cho **tắt dàn tùng điểm LED** sau khi biến $R5$ đã trừ về 0 .

Khi bắt đầu chương trình ngắt, ta sẽ dừng Timer lại để nạp lại giá trị cho Timer bắt đầu một vòng lặp mới rồi mới tiếp tục chạy Timer.

Trong khi LED hiển thị hình trái tim. Sau mỗi vòng lặp, giảm giá trị biến đếm $R4$ đi 1 , nếu chưa đủ số vòng lặp thì quay trở lại chương trình chính.

Khi Timer đã chạy đủ $1s$, thì cài đặt biến đếm $R4$ bằng 1 để **bỏ qua biến này** và sử dụng biến $R5$ cho các vòng lặp $0.1s$ sau.

Sau đó, mỗi $0.1s$ thì ngắt timer được sử dụng để kiểm tra các điểm sáng LED đã tắt hết chưa (kiểm tra biến *CASEOFF*), nếu đã tắt hết thì quay về địa chỉ *RESET*, nếu chưa thì gọi chương trình con *HEX_LED OFF_CONVERT* để **thay đổi bộ mã hex hiển thị LED**.

```

NGATTIMER0:
    CLR      TR0
    MOV      TH0, #HIGH(-9216)
    MOV      TL0, #LOW(-9216)
    SETB

    DJNZ     R4, EXIT
    MOV      R4, #1
    DJNZ     R5, EXIT
;NGATIMER LAN 1: BAT DAU` TAT DAN

    MOV      R5, TOCDOTAT

    MOV      A, CASEOFF
    CJNE    A, #28, NEXT
;NEU (28H) = 28 THI DA TAT XONG
    MOV      R0, SP
    MOV      @R0, #HIGH(RESET) ; SP SAU CAT BYTE CAO
    DEC
    MOV      R0
    MOV      @R0, #LOW(RESET) ; SP TRUOC CAT BYTE THAP
    RETI

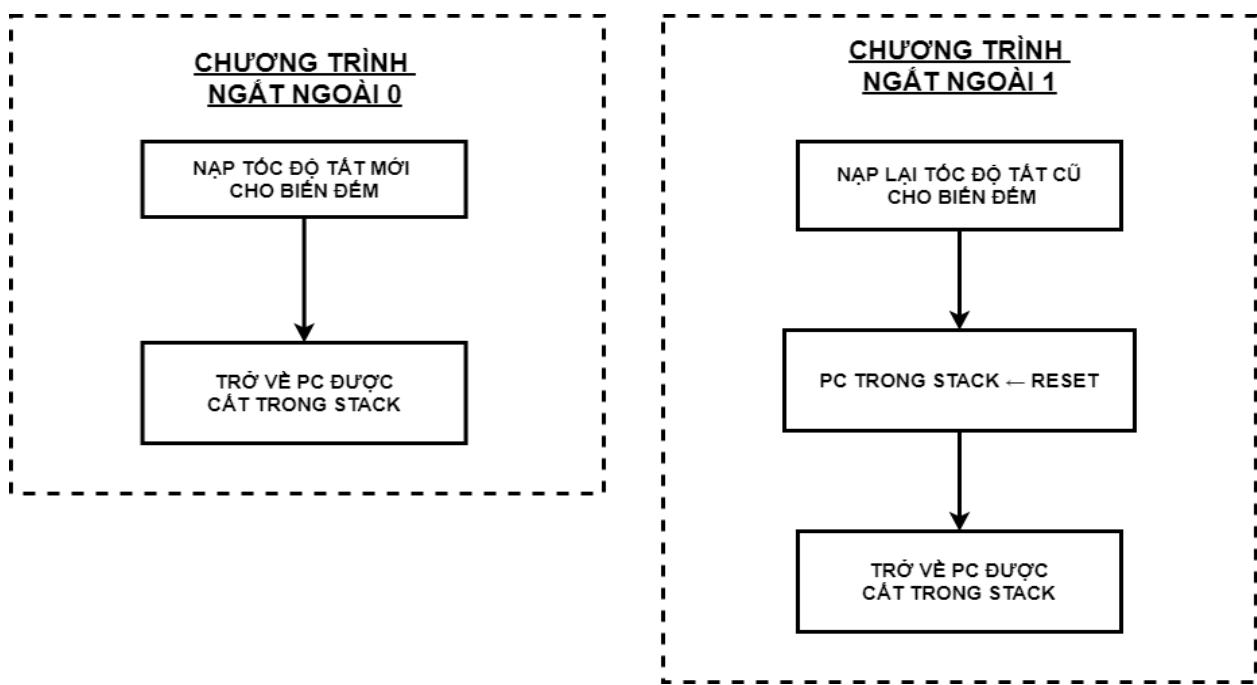
NEXT:
    CALL    HEX_LED OFF_CONVERT

EXIT:   RETI

```

Hình 2.19. Đoạn chương trình Ngắt Timer 0

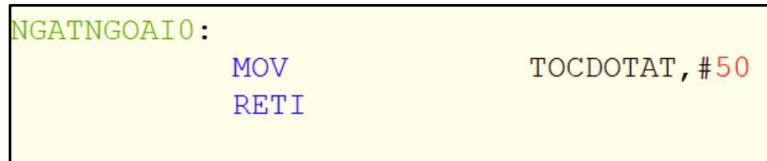
2.3.2. Ngắt ngoài



Hình 2.20. Sơ đồ giải thuật chương trình ngắt ngoài.

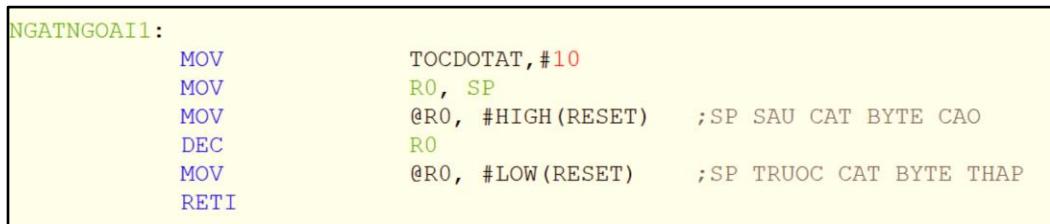
Chương trình còn 2 tùy chọn phụ khi nhấn SWITCH và thực hiện ngắt ngoài:

- Khi nhấn Nút “Ngắt ngoài 0”: thay vì nạp biến TOCDOTTAT là 10 (10 lần 0.01s = 0.1s mới tắt 1 điểm sáng) thì ta nạp cho giá trị 50 (50 lần 0.01s = 0.5s) để tăng thời gian 1 điểm LED tắt đi.



Hình 2.21. Đoạn chương trình Ngắt ngoài 0.

- Khi nhấn Nút “Ngắt ngoài 1”: nạp lại cho biến TOCDOTTAT là 10, và thay đổi giá trị PC được lưu trong stack là giá trị địa chỉ RESET và hoàn thành ngắt bằng lệnh RETI để quay về địa chỉ RESET.



Hình 2.22. Đoạn chương trình Ngắt ngoài 1

CHƯƠNG 3. MỞ RỘNG

3.1. Thiết kế phần mềm bằng KeilC – C

3.1.1. Ý tưởng – Khác biệt so với ASM

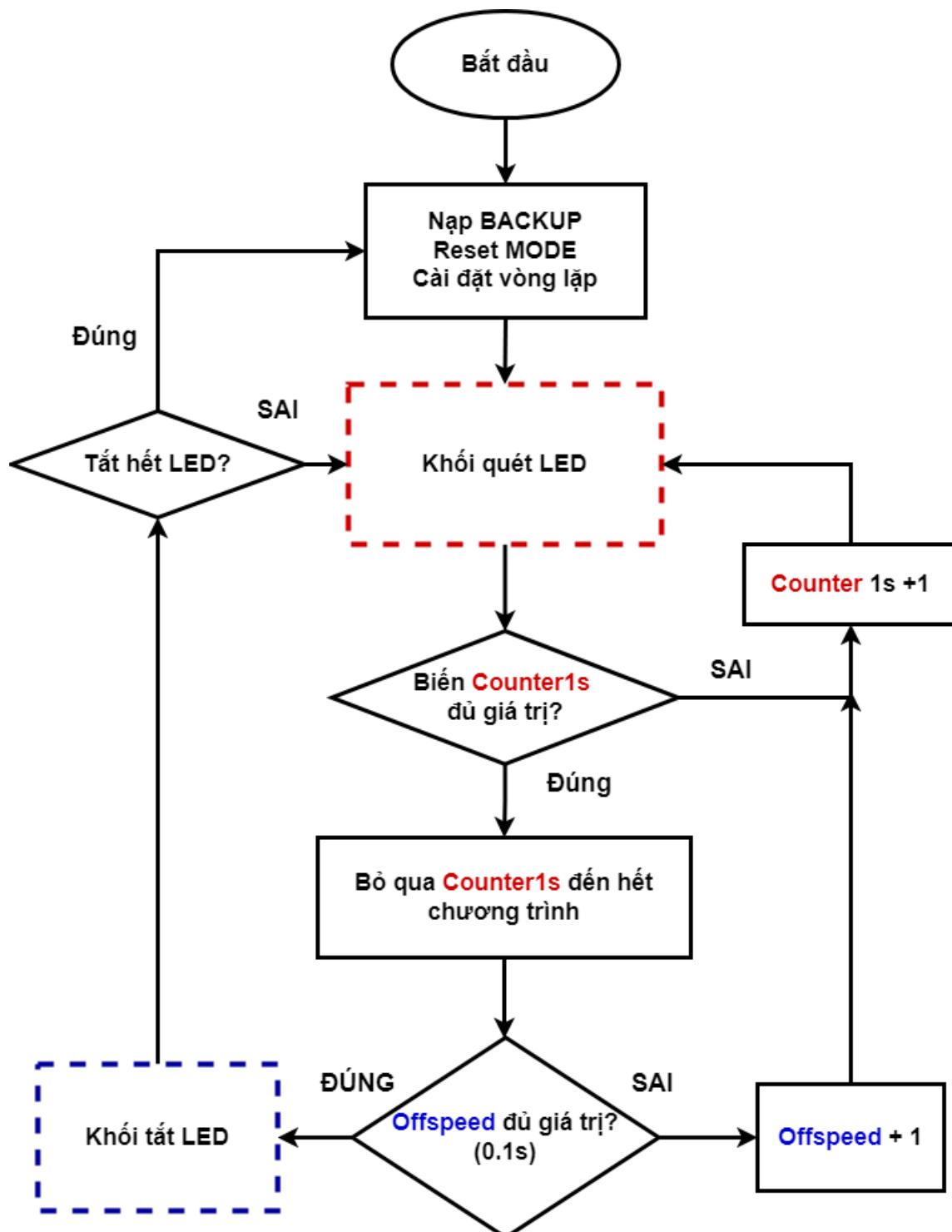
Ý tưởng ở đây là chúng ta sử dụng hoàn toàn mô phỏng (như Keil C) để đếm thời gian các vòng lặp quét , từ đó có thể đưa ra thời gian giữ LED phù hợp bằng cách thay đổi số vòng lặp.

Về cơ bản thuật toán được sử dụng trong ASM và C tương tự nhau, tuy nhiên có 1 số thay đổi như sau.

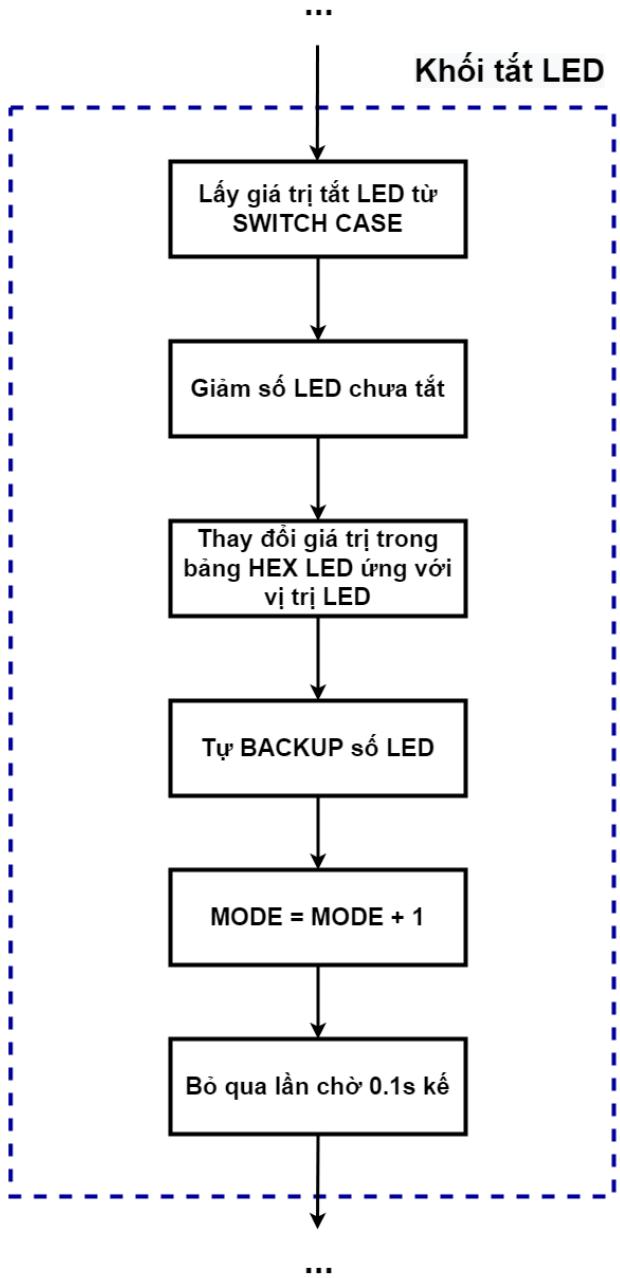
Bảng 3.1. So sánh thuật toán ASM và C

Mục đích	ASM	C
Giữ mã Hex để quét LED	Ghi giá trị từ ROM vào RAM để có thể thay đổi và giữ nguyên giá trị khi Reset.	Ngược lại, các biến mảng được khai báo sẽ được lưu vào RAM nên ta phải tạo 1 mảng BACKUP hoặc thuật toán BACKUP.
Thời gian	Sử dụng TIMER để căn chỉnh thời gian 1s hoặc 0.1s	Sử dụng hoàn toàn mô phỏng để căn chỉnh . (sẽ nói rõ ở phần sau)
Tắt LED	Thay đổi giá trị bảng HEX quét LED bằng cách đưa giá trị từ bảng HEXOFFTABLE thay thế dần dần.	Thay đổi giá trị bảng HEX quét LED bằng cách dùng thuật toán AND và dịch các bit sao cho tắt từng LED.

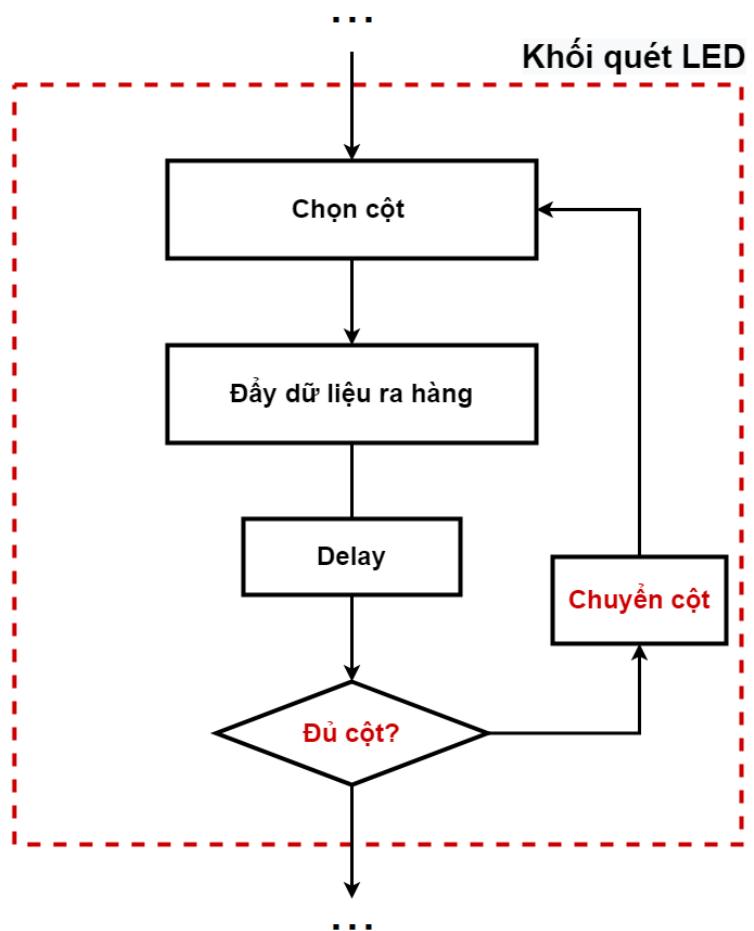
3.1.2. Lưu đồ giải thuật chương trình



Hình 3.1. Lưu đồ giải thuật chương trình



Hình 3.3. Sơ đồ giải thuật Khởi tắt LED



Hình 3.2. Sơ đồ giải thuật Khởi quét LED

3.1.3. Giải thích thuật toán

```
#include <REGX51.H>
#define choncotcao 1
#define choncotthap 0
//1 lan quét tinh toán được 0.0175s
//sử dụng hoàn toàn mờ phông
#define counter1s 52 //60 <-> 1.10492079s
#define off_speed 6 // 4 <-> delta = 0.0702s

unsigned char hextable[] = //Mã quét LED ma trix
{
0x00,0x00, 0x01,0x80, 0x03,0xC0, 0x07,0xE0,
0xF0,0xF0, 0x1F,0xF8, 0x3F,0xFC, 0x7F,0xFE,
0xFF,0xFF, 0xFF,0xFF, 0xFF,0xFF, 0xFF,0xFF,
0xFF,0xFF, 0xFE,0x7F, 0x7C,0x3E, 0x00,0x00
};

unsigned char ledcount[] = //Số lượng LED sáng
{
5,5 , 7,7 , 8,8 , 8,8 , 8,8 , 8,8 , 8,8 ,
7,7 , 6,6 , 5,5 , 4,4 , 3,3 , 2,2 , 1,1 ,
};

unsigned char hextable_copy[33] ; // Backup giá trị Hex
```

Hình 3.4. Chỉ thị tiền xử lý đối với hàng số và mảng

Đầu tiên, ta định nghĩa các giá trị cần thiết để tiện trong việc điều chỉnh và Debug.

Sau đó khai báo các mảng giá trị cần thiết. Lưu ý các giá trị mảng này sẽ được lưu vào vùng RAM.

Kế đến ta khai báo các biến, đặc biệt là các biến sau.

```
volatile unsigned char xdata cotcao _at_ 0x1000;
volatile unsigned char xdata cotthap _at_ 0x1F00;
volatile unsigned char xdata hangcao _at_ 0x4000;
volatile unsigned char xdata hangthap _at_ 0x9000;
```

Hình 3.5. Các biến đặc biệt khai báo tiền xử lý

Các biến được khai báo dưới dạng trên sẽ có tác dụng tương tự như địa chỉ xuất đến ngoại vi:

Bảng 3.2. Tương đồng giữa việc xuất Data ra ngoại vi ASM và C

C	ASM
<pre>volatile unsigned char xdata <TEN> _at_ <ĐỊA CHỈ>; ... <TEN> = DATA;</pre>	<pre>MOV D PTR, #ĐỊA CHỈ MOV A, #DATA MOVX @D PTR, A</pre>

Theo thứ tự thực hiện chương trình. Chúng ta sẽ vào hàm Main – chương trình chính

a. Main

```
void main () {
    mode = 0;
    for(cop = 0 ; cop < 32 ; cop++) {
        hextable_copy[cop] = hextable[cop];
    }
}
```

Hình 3.6. Hàm Main

Trước mắt là sẽ đưa dữ liệu vào mảng Backup. Ngược lại so với ASM, trong C, dữ liệu được định nghĩa sẽ lưu vào trong RAM. Vì vậy, khi thực hiện bất kỳ thay đổi gì đối với vùng dữ liệu này thì nó sẽ bị thay đổi vĩnh viễn → Do đó ta cần Backup.

Phản ứng tiếp là thực hiện quét ma trận, song song với việc kiểm tra điều kiện đã sáng 1 giây, và sáng 0.1 giây.

```

while(1) {
    pointer_quet = 0;
    quetmatran(choncotthap);
    quetmatran(choncotcao);

    y++;
    if(y == counter1s ) {
        y = counter1s - 1; //Neu Y dem du lan thi` truot qua Y
        x++;
        if(x == off_speed) {
            x = 0;
            tatled();
            if (mode == 28) {
                reset();
            }
        }
    }
}

```

*Hình 3.7. Quá trình thực hiện quét LED***b. Quét ma trận**

Lựa chọn cột bằng cách xuất xDATA giá trị lựa chọn.

Ví dụ muốn COT 14 (COT 6 cao) được đầy dữ liệu thì chọn cột bằng giá trị sau 0x40 tức là 0100 0000:

```

quetmatran(choncotcao);

choncot = 0x40;

cot cao = choncot;

```

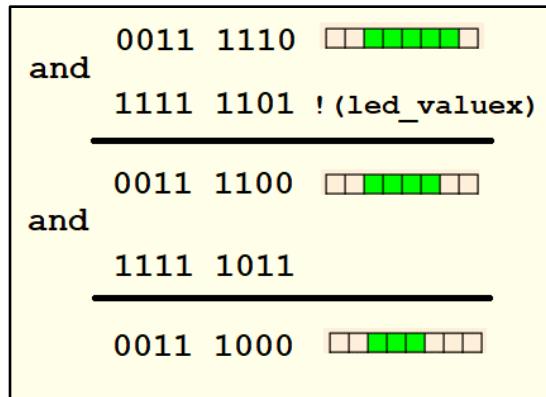
Ở vòng lặp trên, cứ qua mỗi vòng, biến choncot sẽ được *** 2**, tương tự với ASM – ta dịch trái bit **1**. Chọn được cột thì ta đầy dữ liệu từ bảng HEX ra. Đẩy ra HANGCAO trước rồi tới HANGTHAP.

Kết thúc vòng lặp, chúng ta khử Ghosting bằng việc đầy 0 ra không cột nào cả (tắt).

c. Tắt LED

Ý tưởng cho thuật toán tắt LED này có phần tương tự với ASM: Chúng ta thay đổi bảng HEX LED mang giá trị quét.

Ở ASM chúng ta đưa thẳng giá trị tắt từ 1 bảng mới vào, tuy nhiên trong C, vùng chia dữ liệu hạn hẹp, chúng ta khó có thể làm thế được (khoảng 162 giá trị thay đổi ~ 162 ô nhớ). Thay vào đó, chúng ta sẽ sử dụng thuật toán với ý tưởng là:



Hình 3.8. Ý tưởng tắt LED bằng AND

Cứ qua mỗi lần chúng ta dịch bit 0 của !(led_value) tương đương với việc dịch bit 1 của led_value thì ta sẽ được các giá trị cần để tính toán. (dịch bit 1 sẽ đơn giản hơn việc dịch bit 0).

Qua mỗi lần, ta sẽ thay đổi giá trị trong mảng, giảm số LED còn lại, và cũng như tự Backup giá trị LED còn lại (vì chúng nó được định theo cặp).

d. Vấn đề Delay và thời gian quét.

Thực ra C vẫn đầy đủ chức năng như ASM, bao gồm cả khởi động Timer .v.v...

Thay vì dùng Timer, để đổi mới 1 chút, chúng ta sẽ căn chỉnh thời gian hoàn toàn bằng mô phỏng Keil C (có 1 tỷ lệ sai số nhất định giữa mạch thật và mô phỏng, nhưng chúng ta hoàn toàn có thể chấp nhận được).

```

void delay_ms (int time) {           //Ham Delay ms
    int j, k;
    for(j = 0; j < time; j++) {
        for (k = 0; k < 115 ; k++)
        {}
    }
}

```

Hình 3.9. Ví dụ hàm delay_ms

```

47
48
49 void delay_ms (int time) { //Ham Delay ms
50     int j, k;
51     for(j = 0; j < time; j++) {
52         for (k = 0; k <115 ; k++)
53     {}
54 }
55 }

47
48
49 void delay_ms (int time) { //Ham Delay ms
50     int j, k;
51     for(j = 0; j < time; j++) {
52         for (k = 0; k <115 ; k++)
53     {}
54 }
55 }

```

Hình 3.10. Thời gian được đo bằng Keil C

Khoảng thời gian được tính vào khoảng 8.68×10^{-6} giây đối với thạch anh 11.0592 MHz. Vì thế 1 mili giây sẽ vào khoảng 115 vòng lặp (làm tròn xuống, bù trừ khai báo).

Tương tự cho các phần có tính toán đến thời gian khác:

```

y++;
if(y == counter1s ) {
    y = counter1s - 1; //Neu Y dem du lan thi` truot qua Y
    x++;
    if(x == off_speed) {
        x = 0;
        tatled();
        if (mode == 28) {
            reset();
        }
    }
}

```

Hình 3.12. Vòng lặp điều chỉnh thời gian trong Main

```

86
87 void tatled(){
88
89     counter_led = ledcount[mode]; //Lay so LED tu` trong bang
90     led_valueex = ledoff(mode); //Lay HEX thay doi tu` SWITCH
91
92     if(mode%2==0) {
93         delta_led = ledcount[mode+1] - counter_led; //So LED con lai
94     }
}

```

Hình 3.13. Đo thời gian vòng lặp điều chỉnh bằng Keil C

Để vào được hàm tatled() tốn 0.99 giây

e. Reset

```

void reset (void) {           //Tắt hết LED thi Reset

    for(cop = 0 ; cop < 32 ; cop++) {
        hextable[cop] = hextable_copy[cop]; //BACKUP
    }

    ((void (code *) (void)) 0x0000) (); //Về đầu ch. trình
}

```

Hình 3.13. Hàm Reset trong C

Reset → ta đưa dữ liệu cũ (đã Backup) vào trong RAM chuẩn bị cho việc quét lại từ đầu.

Cũng như đưa con trỏ PC về đầu chương trình 0x0000.

3.1.4. Tổng kết

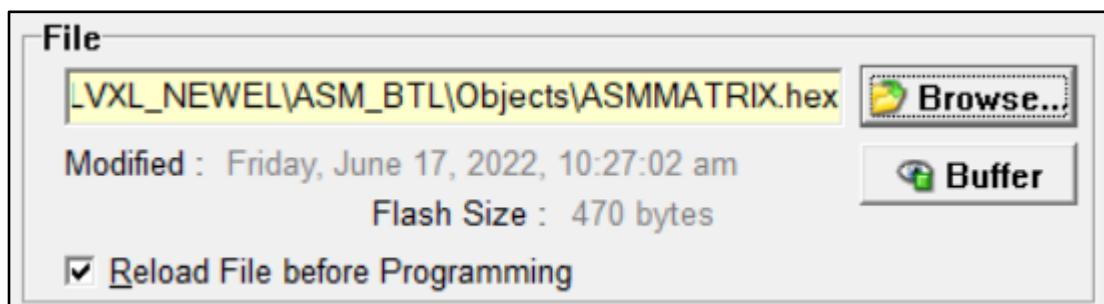
Mỗi bên đều là khoảng 220 dòng CODE, tuy vậy chúng ta có thể thấy rõ:

- C dễ hiểu, và dễ dàng hơn trong việc viết code, cũng như thực hiện xuất các giá trị.

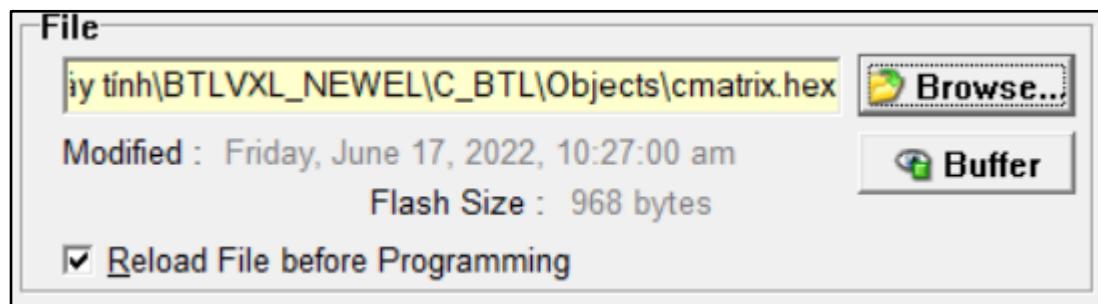
Tuy nhiên, thời gian build dài, Flash size ⁴lớn, cũng như khối lượng lưu trữ mảng hạn chế (lý do tại sao phải dùng SWITCH CASE), khoảng 128 bytes ~ 128 ô mảng

- ASM tuy dài dòng, 1 mục tiêu phải dùng khá nhiều công việc.

Nhưng thời gian build ngắn, Flash size bé, mảng dữ liệu có thể lưu trữ lớn. Chúng ta có thể can thiệp sâu hơn vào hệ thống và hiểu cách hệ thống hoạt động.



⁴ Dung lượng Code nạp cho MCU. Ở AT89S52 con số này giới hạn vào khoảng 8 Kbytes.

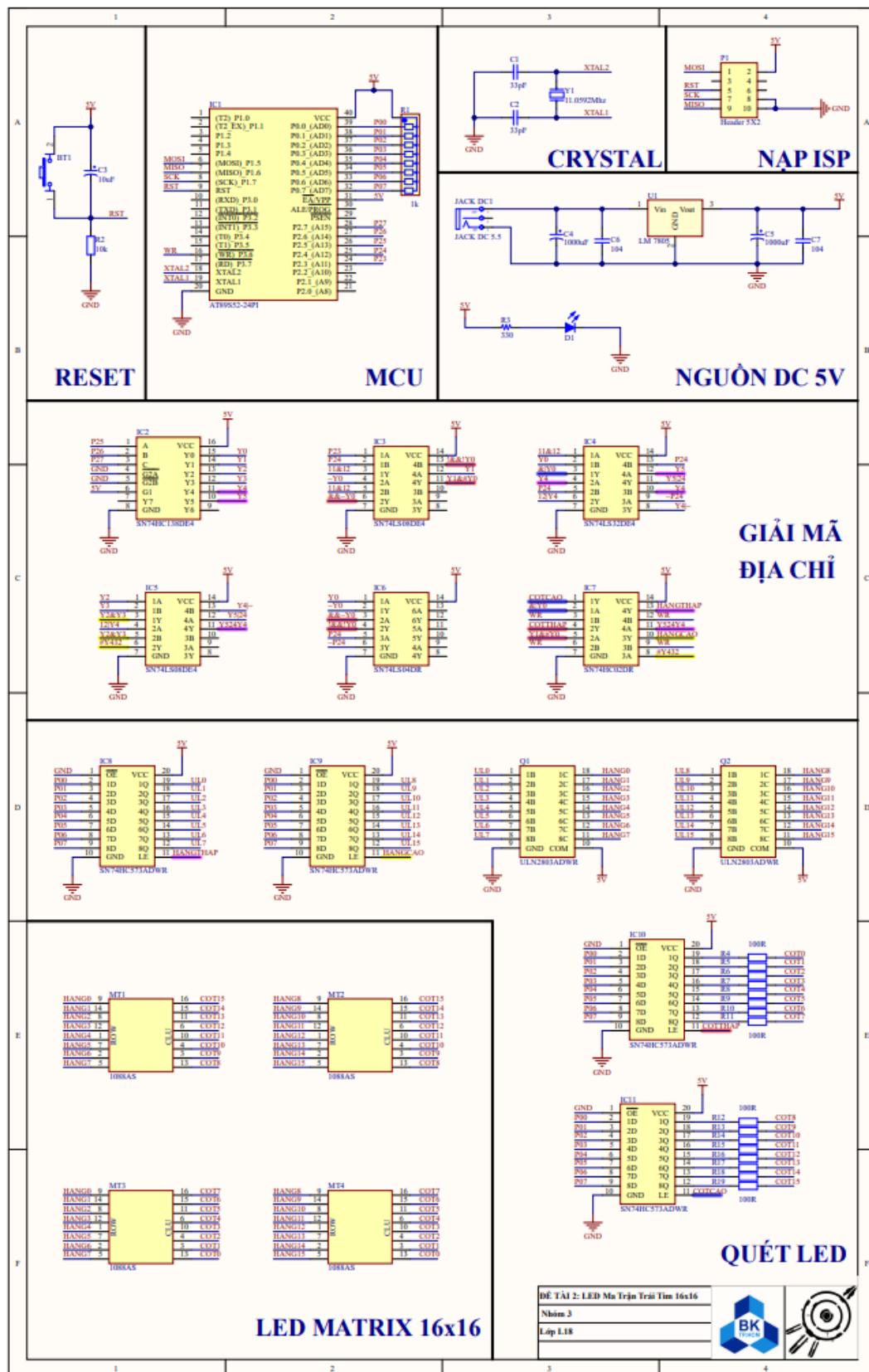


Hình 3.14. So sánh file Flash giữa 2 ngôn ngữ

3.2. Thiết kế mạch PCB bằng Altium

3.2.1. Sơ đồ nguyên lý

BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18



Hình 3.15. Sơ đồ nguyên lý mạch

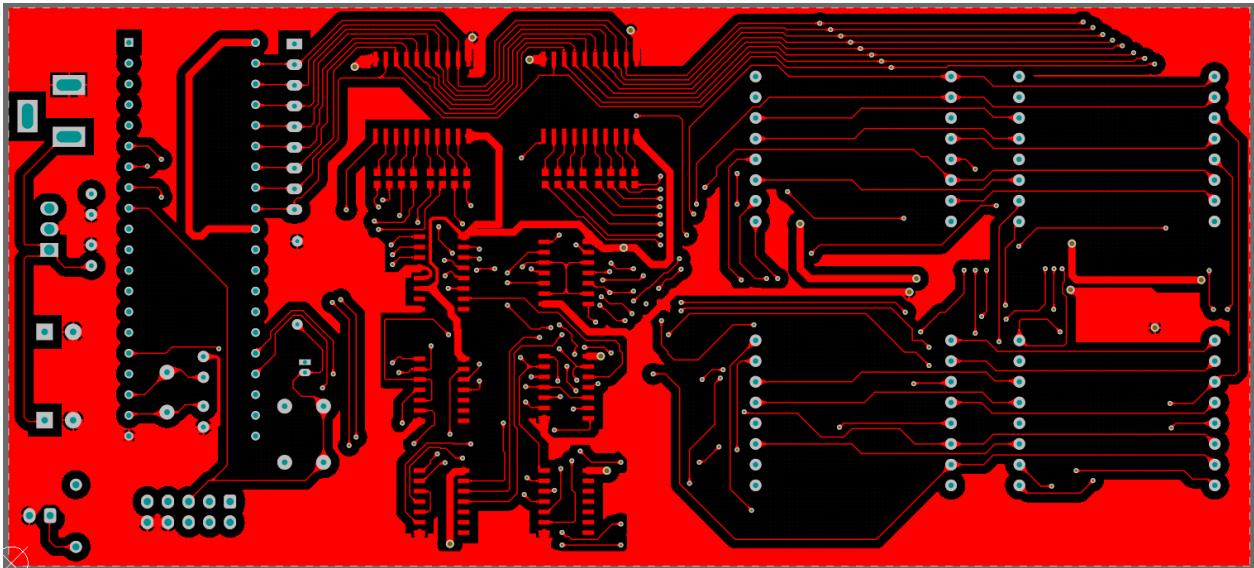
Với sơ đồ mạch trên, các con trỏ đã được về đúng vị trí của nó (dùng trỏ trên IC xuất CÔT). Kèm theo đó là thực tế hoá phần cứng đã mô phỏng trên Proteus khi kết nối đầy đủ các khối: Nguồn, MCU, Giải mã, Quét LED, Hiển thị.

Bổ sung so với Proteus:

- Khối nguồn: Cáp bằng JACK 5.5mm, qua một mạch ổn áp khử nhiễu bằng LM 7805.
- Khối ISP: Dock nạp 10 chân với tiêu chuẩn ISP.

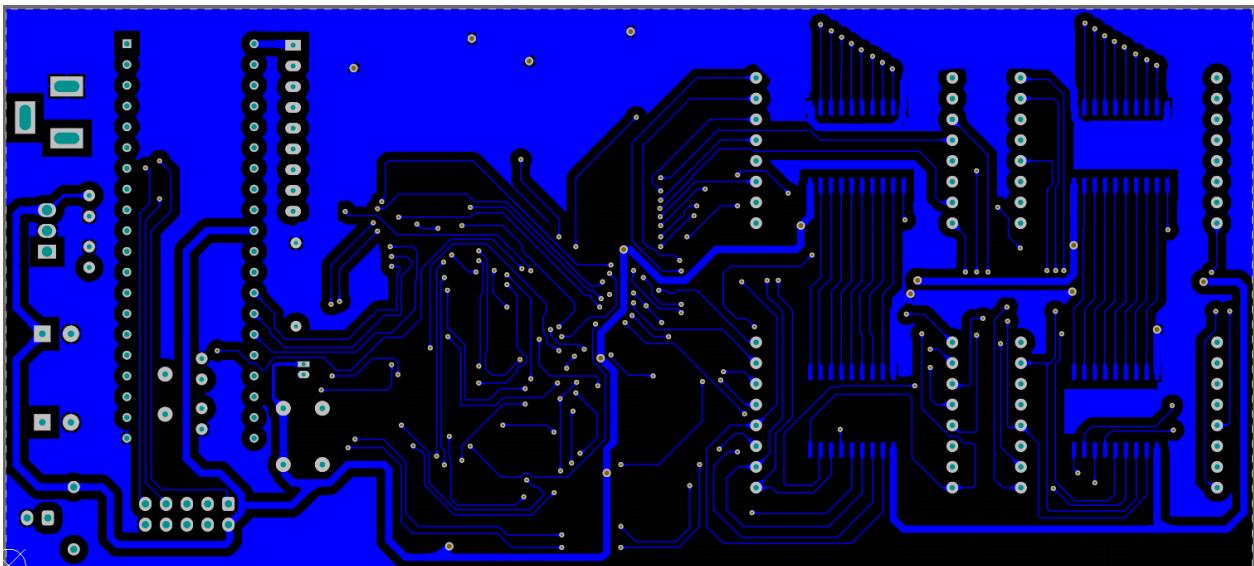
3.2.2. Layout (Phủ đồng GND)

- Lớp TOP:



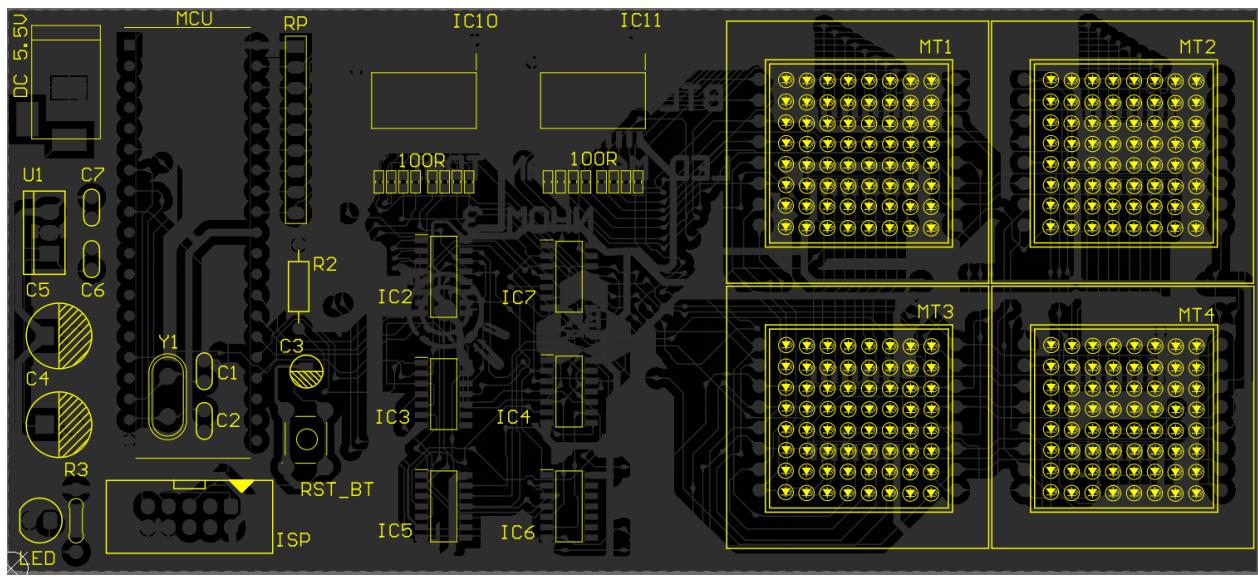
Hình 3.16. Lớp TOP mạch PCB

- Lớp BOTTOM (trên cùng một đường chiếu với TOP):

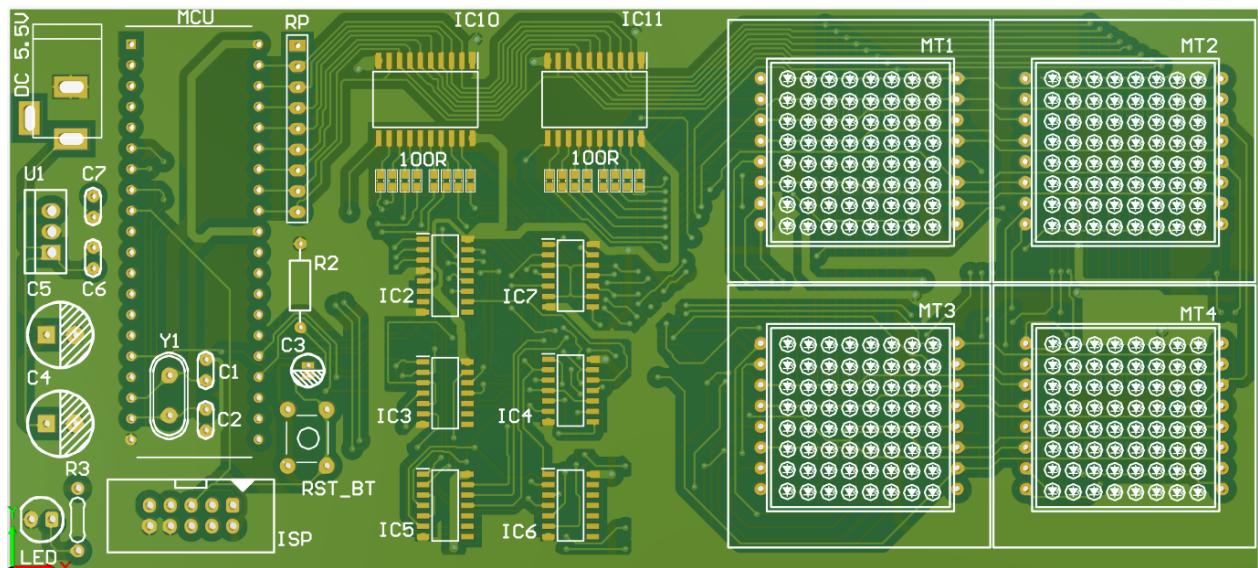


Hình 3.17. Lớp BOTTOM mạch PCB

- Silkscreen TOP:

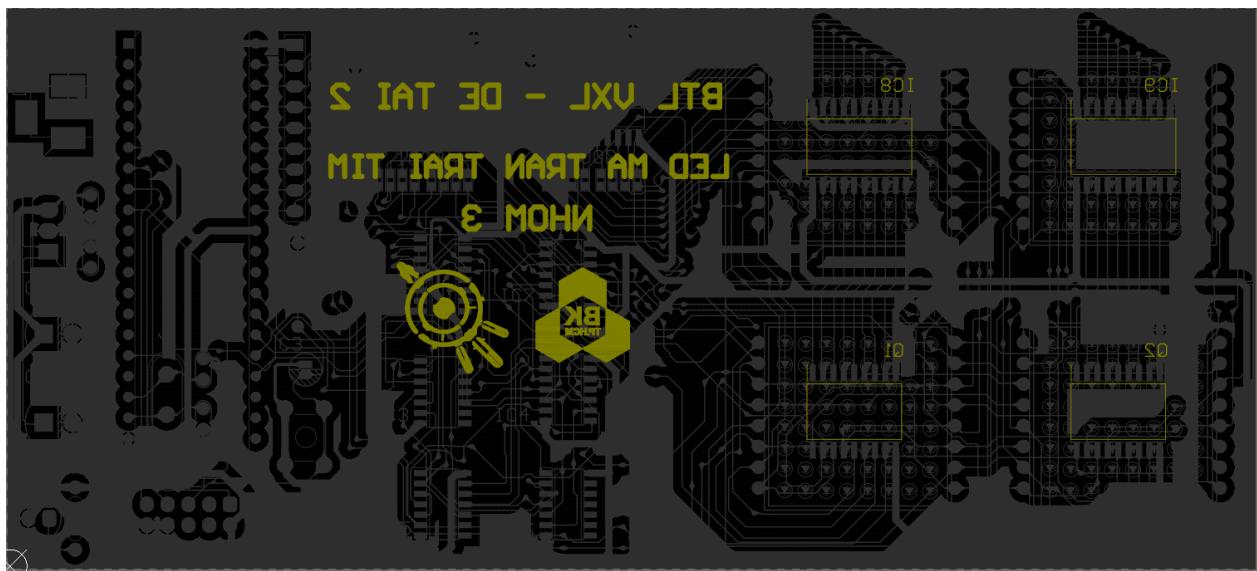


Hình 3.18. Lớp TOP Overlay mạch PCB



Hình 3.19. Silkscreen TOP 3D View

- Silkscreen BOTTOM:



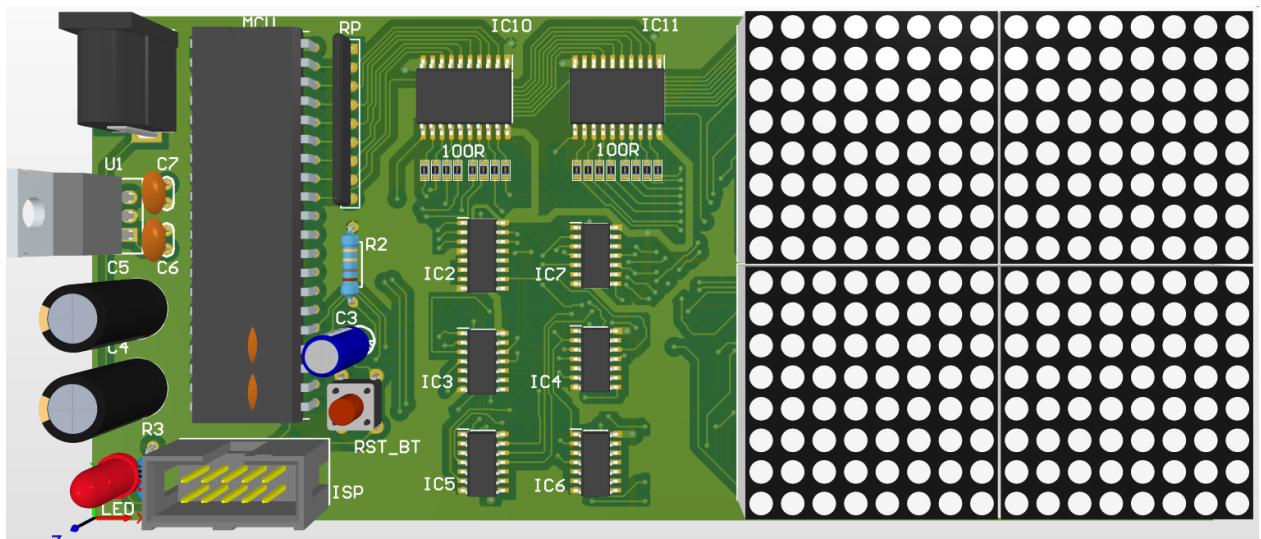
Hình 3.20. Lớp BOTTOM Overlay mạch PCB



Hình 3.21. Silkscreen BOTTOM 3D View

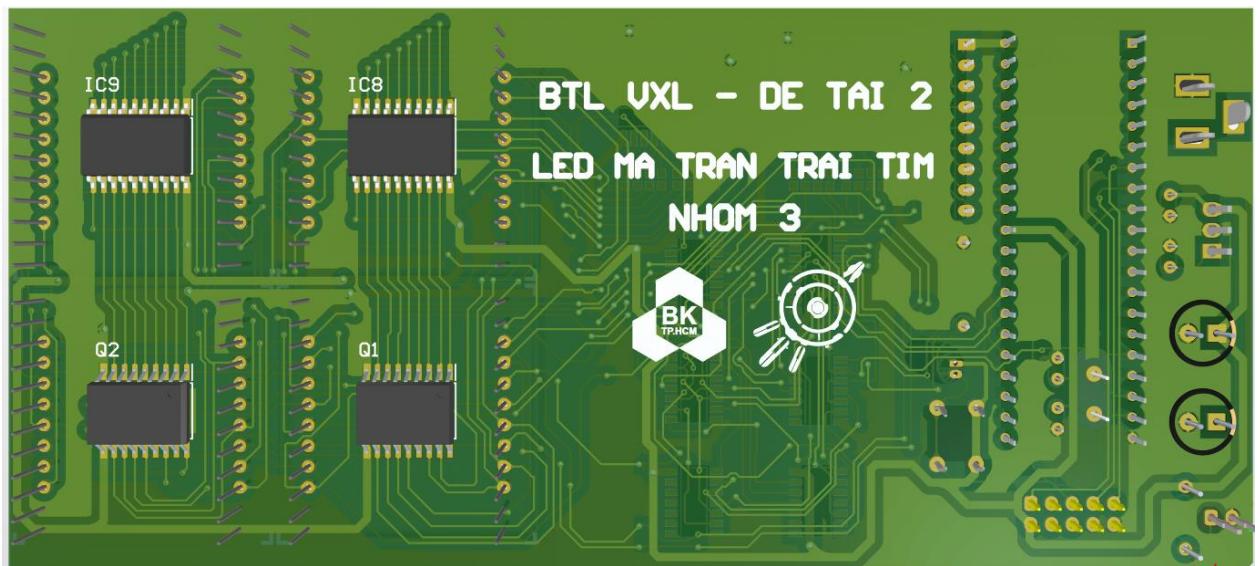
3.2.3. 3D viewer

- Mặt trên:



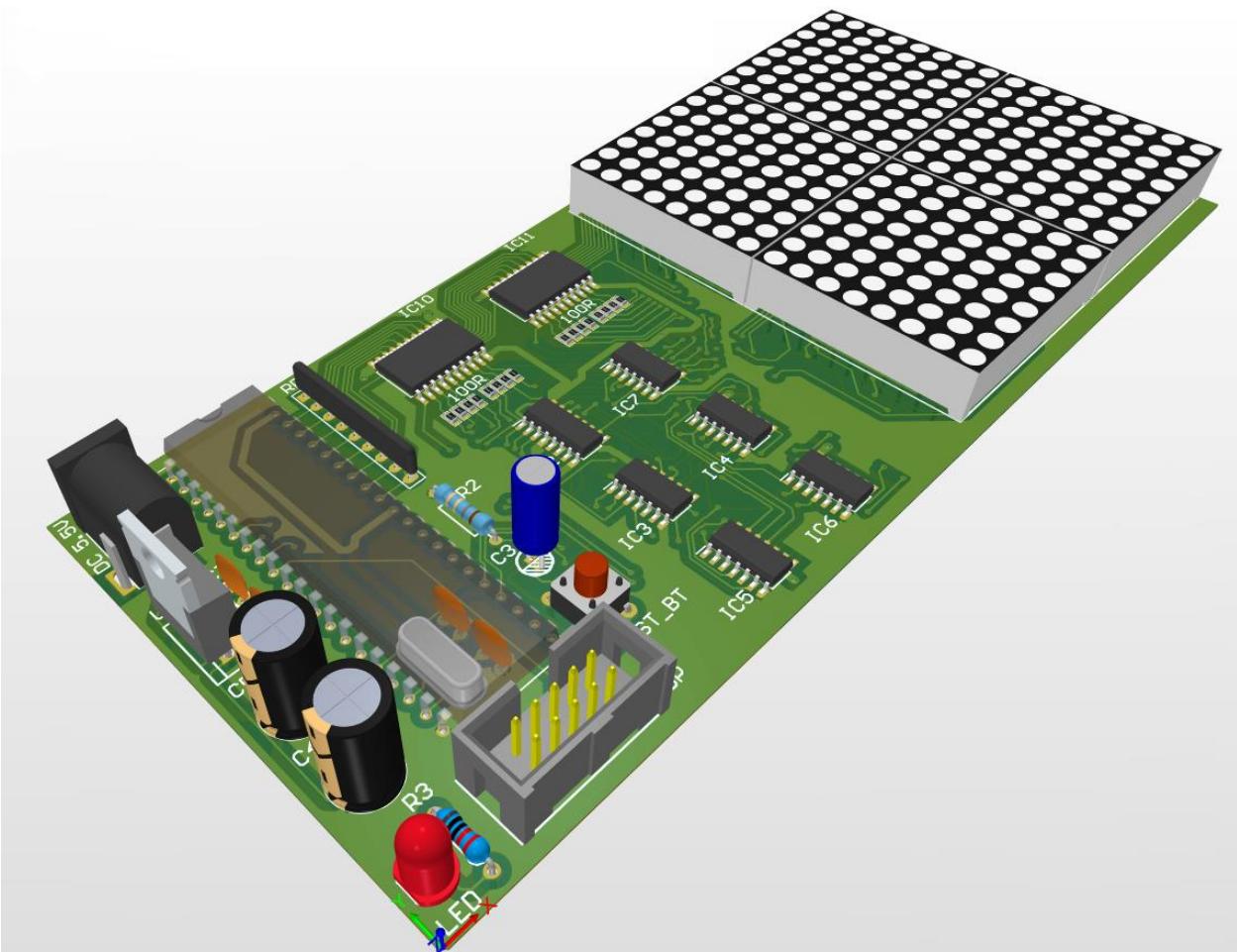
Hình 3.22. Mặt trên PCB 3D View

- Mặt dưới:



Hình 3.23. Mặt dưới PCB 3D View

- Góc nghiêng:



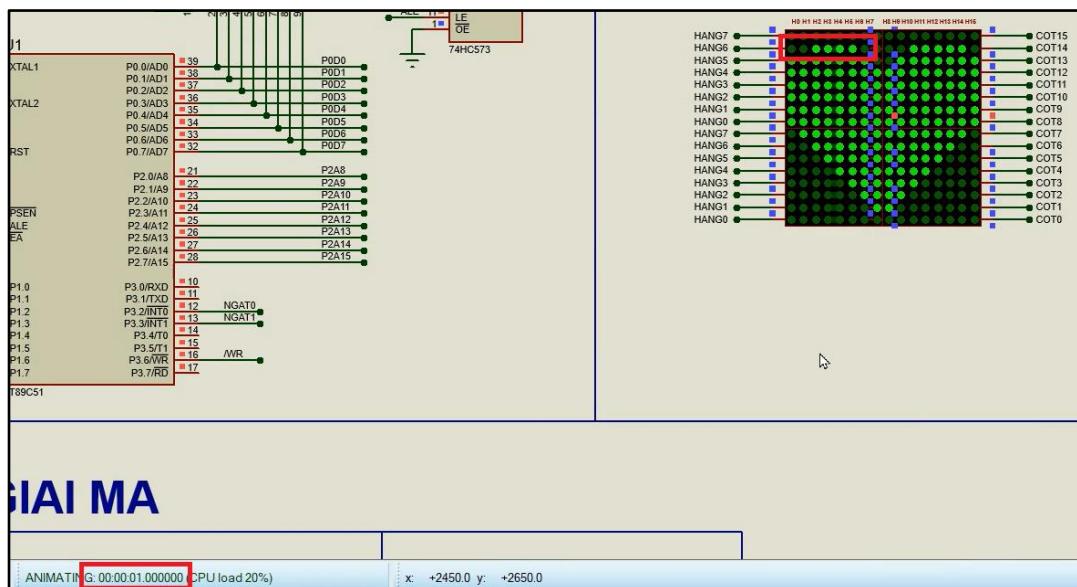
Hình 3.24. Mặt nghiêng PCB 3D View

CHƯƠNG 4. ĐO ĐẠC MÔ PHỎNG VÀ KẾT LUẬN.

4.1. Kiểm chứng thời gian trên Proteus

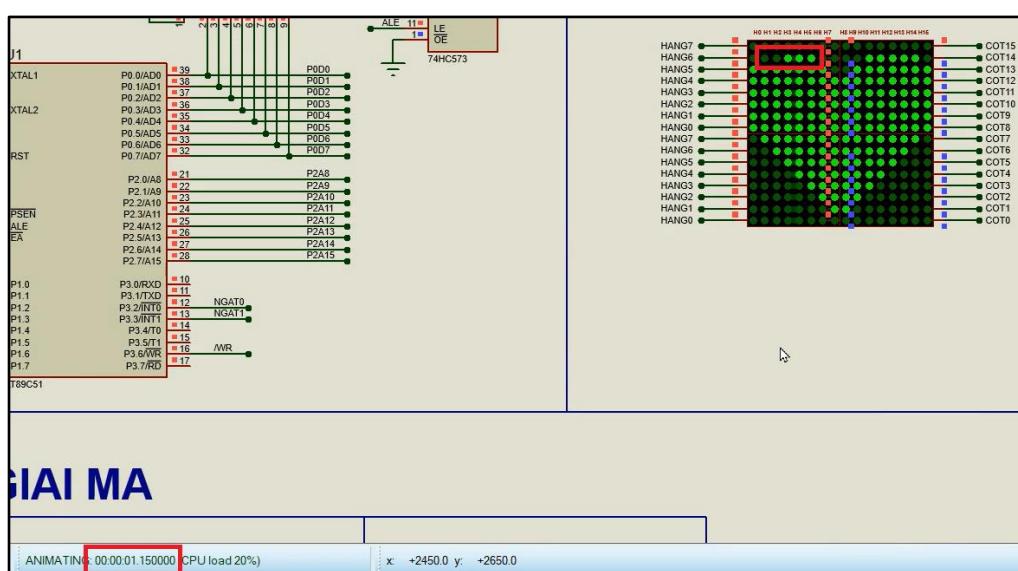
4.1.1. ASM

- Sau 1 giây, LED đã bắt đầu tắt vị trí đầu tiên



Hình 4.1. Sau 1 giây thì LED bắt đầu tắt

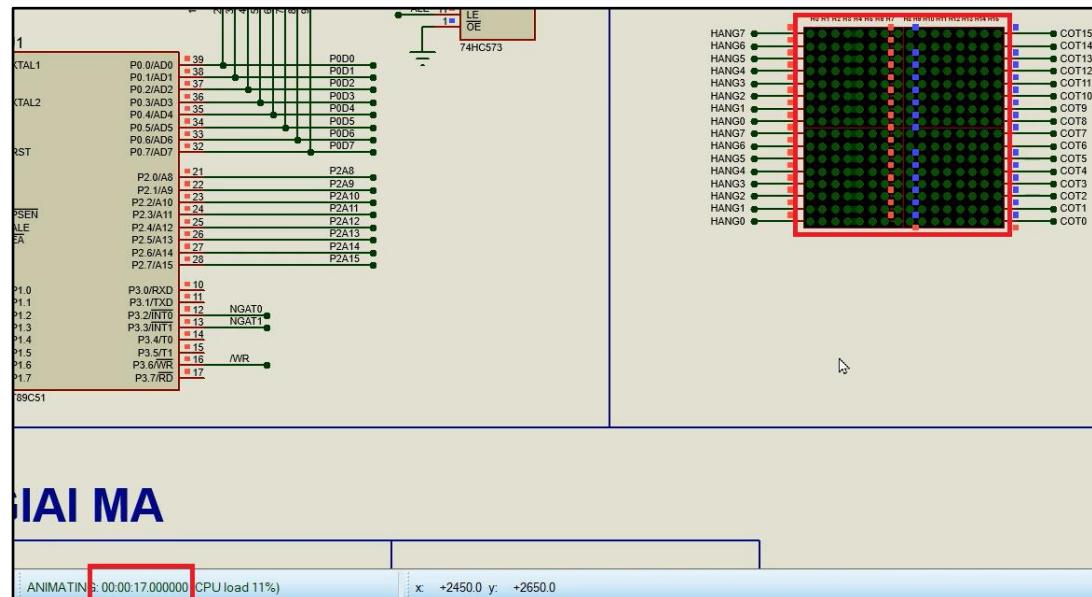
- Sau 1.15 giây, đã tắt LED thứ 2 nhưng chưa tắt LED thứ 3



Hình 4.2. Sau 1.15 giây, đã tắt được 2 LED

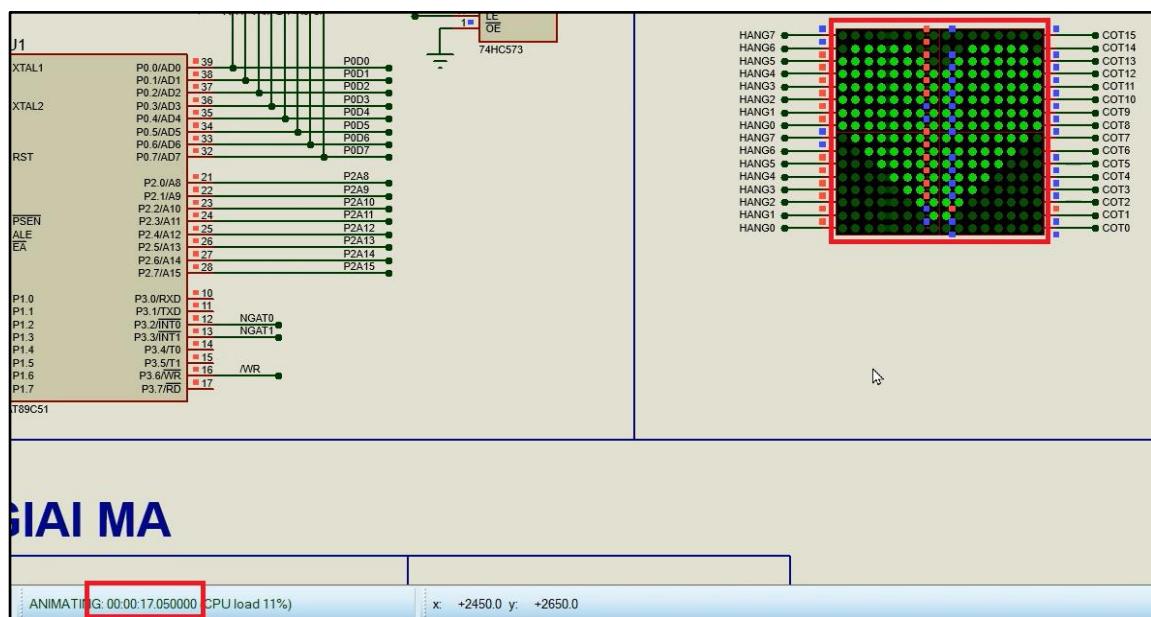
BÁO CÁO BÀI TẬP LỚP LỚP L18

- Có khoảng 160 LED, vậy khoảng thời gian cho quá trình từ đầu cho đến tắt hết khoảng **17 giây**.



Hình 4.3. Hoàn thành tắt LED

- Ngay sau đó sẽ **Reset** lại từ đầu.



Hình 4.4. Reset quá trình

4.1.2. C

Ta có thể đo đặc tương tự cho C với mức sai số khoảng 5%

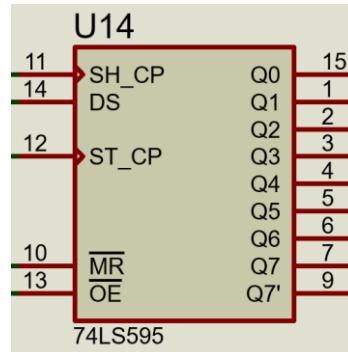
4.2. Hướng phát triển đề tài

4.2.1. Ưu và nhược điểm của mạch

- **Ưu điểm:** mạch tương đối dễ thiết kế, dễ hiểu và dễ sử dụng. Ngoại trừ giải mã địa chỉ, các linh kiện rất dễ kiểm và có tính ứng dụng thực tế cao.
- **Nhược điểm.** Vì chỉ dùng IC '573, và tốn hẳn 8 bit Port 0 cho Data, việc mở rộng mạch khá khó khăn. Trước mắt là 16x16, nếu tăng lên 24x24 hoặc 16x32 thì là một ván đề nan giải

4.2.2. Hướng phát triển

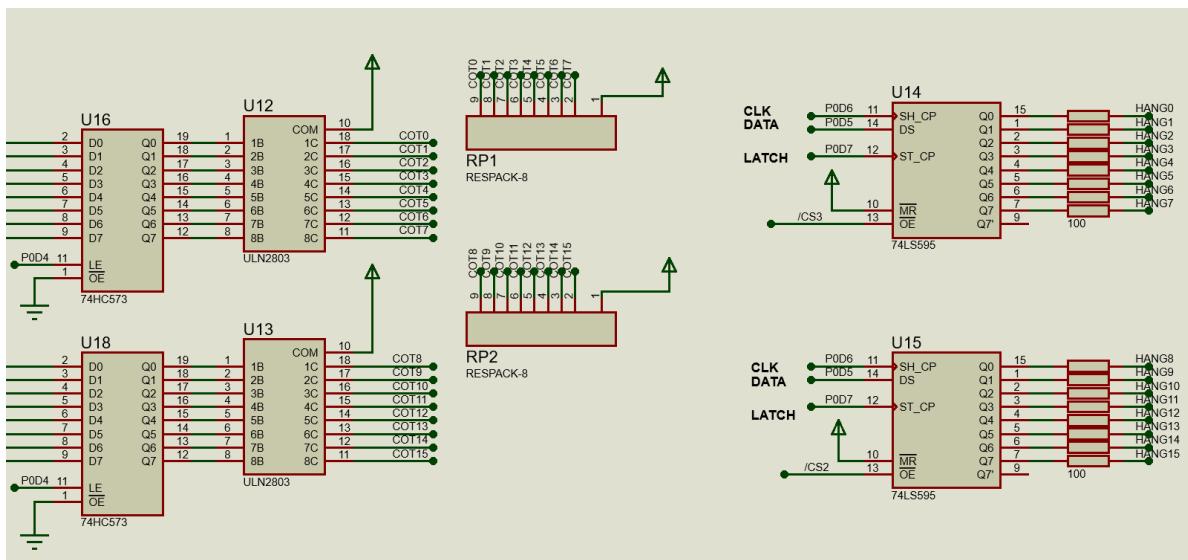
Từ ưu và nhược điểm trên, Nhóm 2 đưa ra ý tưởng sử dụng IC 74LS595 – một IC dịch bit, nhập nối tiếp xuất song song. IC này được sử dụng rất phổ biến với mục đích tiết kiệm Port khi truy xuất dữ liệu.



Hình 4.5. IC 74LS595 trong Proteus

Chúng ta có thể kết hợp sử dụng '595 và '573, hoặc hoàn toàn '595. Tuy có chút phức tạp, nhưng với việc có thể hoạt động những mạch ma trận lớn như 24x24, 32x32 thì là một ưu điểm rất ăn tiền.

Ngoài ra, thê mạch của '595 còn nằm ở việc nó có thể nối Cascade cực kỳ đơn giản, thuận lợi trong việc mở rộng mạch hoặc ghép các mạch lại với nhau.

**Hình 4.6 Đơn cử kết nối mạch dùng ‘573 và ‘595**

4.3. Kết luận

Đè tài 2 – LED Ma trận 16x16 hiện trái tim, là một đèn tài mang tính ứng dụng thực tiễn rất cao. LED Ma trận và MCU sử dụng rất phổ biến, trong mạch Quang báo nói riêng cũng như các mạch hiển thị thông tin nói chung.

Từ đó, có thể rút ra những kinh nghiệm, bài học trong quá trình hoạt động học tập, làm việc sau này. Rèn luyện thêm kỹ năng làm việc nhóm, bổ sung kiến thức, trau dồi khả năng sử dụng phần mềm và tìm kiếm tài liệu tham khảo.

Mặc dù quá trình làm BTL nhóm gặp khá nhiều khó khăn về thời gian cũng như khoảng cách, nhưng mọi người vẫn cố gắng hoàn thành một cách trọn vẹn nhất.

Qua đây Nhóm 2 hân hạnh một lần nữa cảm ơn thầy Đỗ Ngọc Cẩm đã đưa ra đề tài, hướng dẫn nhóm hoàn thành.

PHỤ LỤC

1. Bản thiết kế mạch điện trên Proteus



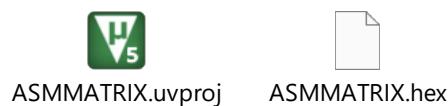
2. Bản thiết kế mạch PCB Altium Schematic



3. Bản PDF PCB các lớp



4. CODE ASM



```
TruyXuatCOTCAO      EQU      1000H
TruyXuatCOTTHAP     EQU      1F00H
TruyXuatHANGCAO    EQU      4000H
TruyXuatHANGTHAP   EQU      9000H
;-----DINH NGHIA O RAM NOI-----
CHONCOT            EQU      21H      ; CHON COT DE QUET
LEDCOUNTER          EQU      25H      ; SO LED CON LAI
CHUA TAT            EQU      26H      ; = 10 <=> 0.1S
TOCDOTAT           EQU      27H      ; VI TRI HIEN TAI
TRONG BANG TAT LED
CASEOFF             EQU      28H
; 1 CASEOFF <=> 1/2 HANG TAT (HANG CAO HOAC THAP) THUOC 1
COT
; <=> CO 28 CASE, 28 TRUONG HOP
POINTER_DATARAM    EQU      29H      ; CON TRO SU DUNG
DE CHUYEN DATA VAO RAM NOI

ORG                 0000H
```

BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18

JMP	SET_MODE
ORG	0003H
JMP	NGATNGOAI0 ; TANGTOC
ORG	000BH
JMP	NGATTIMER0
ORG	000BH
JMP	NGATNGOAI1 ; RESET SOM
 SET_MODE:	
MOV	TOCDOTAT, #10
JMP	MAIN
ORG	0030H
 MAIN:	
RESET:	
MOV	SP, #07H
MOV	CASEOFF, #0
MOV	POINTER_OFFTABLE, #0
MOV	LED COUNTER, #0
MOV	TMOD, #01H
MOV	R4, #100
MOV	R5, #1
SETB	EA
SETB	TF0
SETB	ET0 ; CHO NGATTIMER0
SETB	EX1
SETB	EX0
MOV	R0, #30H
MOV	POINTER_DATARAM, #0
MOV	R1, #50H
HEARTDATA:	
MOV	A, POINTER_DATARAM
MOV	DPTR, #HEXTABLE
MOVC	A, @A+DPTR
MOV	@R0, A
MOV	A, POINTER_DATARAM
MOV	DPTR, #LED COUNT_TABLE
MOVC	A, @A+DPTR
MOV	@R1, A

```

        INC          R1
        INC          R0
        INC          POINTER_DATARAM
        CJNE         R0, #50H, HEARTDATA

        MOV          R0, #4DH ; 4DH LA O NHO CHUA
LED DAU` TIEN CAN TAT

BACK:
        MOV          R1, #30H ; 30H LA O NHO CHUA
HEX DAU TIEN CAN QUET

        MOV          DPTR, #TruyXuatCOTTHAP
        CALL         QUETMATRAN

        MOV          DPTR, #TruyXuatCOTCAO
        CALL         QUETMATRAN

        JMP          BACK

DELAY:
        MOV          R7, #2
LAPDL:
        MOV          R6, #250
        DJNZ         R6, $
        DJNZ         R7, LAPDL
        RET

QUETMATRAN:
        MOV          CHONCOT, #00000001B
        MOV          R2, #8

LAPQUET:
        PUSH         83H ; DPH
        PUSH         82H ; DPL

        MOV          A, CHONCOT
        MOVX         @DPTR, A

        MOV          A, @R1
        DPTR, #TruyXuatHANGCAO
        @DPTR, A
        INC          R1

        MOV          A, @R1
        DPTR, #TruyXuatHANGTHAP
        @DPTR, A
        INC          R1

```

BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18

CALL	DELAY
POP	82H
POP	83H
MOV	A, CHONCOT
RL	A
MOV	CHONCOT, A
DJNZ	R2, LAPQUET
MOV	A, #0000000B
MOVX	@DPTR, A
RET	
NGATTIMER0:	
CLR	TR0
MOV	TH0, #HIGH(-9216)
MOV	TLO, #LOW(-9216)
SETB	TR0
DJNZ	R4, EXIT
MOV	R4, #1
DJNZ	R5, EXIT
;NGATIMER LAN 1: BAT DAU` TAT DAN	
MOV	R5, TOCDOTAT
MOV	A, CASEOFF
CJNE	A, #28, NEXT
;NEU (28H) = 28 THI DA TAT XONG	
MOV	R0, SP
MOV	@R0, #HIGH(RESET); SP SAU CAT
BYTE CAO	
DEC	R0
MOV	@R0, #LOW(RESET); SP TRUOC
CAT BYTE THAP	
RETI	
NEXT:	
CALL	HEX_LED OFF _ CONVERT
EXIT:	RETI
HEX_LED OFF _ CONVERT:	
PUSH	01H

BÁO CÁO BÀI TẬP LỚN MÔN VXL – LỚP L18

```

        MOV          A,  POINTER_OFFTABLE
        MOV          DPTR, #HEXOFFTABLE
        MOVC         A, @A+DPTR
        MOV          @R0, A
        INC           POINTER_OFFTABLE

        MOV          A, #50H
        ADD          A, CASEOFF
        MOV          R1, A
        MOV          A, @R1
        MOV          LEDCOUNTER, A

        DJNZ         LEDCOUNTER, EXIT_RET
        DEC          R0
        INC           CASEOFF

        EXIT_RET:
        MOV          @R1, LEDCOUNTER
        POP          01H
        RET

        NGATNGOAI0:
        MOV          TOCDOTAT, #50
        RETI

        NGATNGOAI1:
        MOV          TOCDOTAT, #10
        MOV          R0, SP
        MOV          @R0, #HIGH(RESET) ; SP SAU
CAT BYTE CAO
        DEC          R0
        MOV          @R0, #LOW(RESET) ; SP
TRUOC CAT BYTE THAP
        RETI

        HEXOFFTABLE:
DB      03CH, 038H, 030H, 020H, 000H, 078H, 070H, 060H, 040H, 000H
DB      07EH, 07CH, 078H, 070H, 060H, 040H, 000H, 0FCH, 0F8H, 0F0H, 0E0H,
      0C0H, 080H, 000H
DB      0FEH, 0FCH, 0F8H, 0F0H, 0E0H, 0C0H, 080H, 000H, 0FEH, 0FCH, 0F8H,
      0F0H, 0E0H, 0C0H, 080H, 000H
DB      0FEH, 0FCH, 0F8H, 0F0H, 0E0H, 0C0H, 080H, 000H, 0FEH, 0FCH, 0F8H,
      0F0H, 0E0H, 0C0H, 080H, 000H
DB      0FEH, 0FCH, 0F8H, 0F0H, 0E0H, 0C0H, 080H, 000H, 0FEH, 0FCH, 0F8H,
      0F0H, 0E0H, 0C0H, 080H, 000H

```

```

DB  OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H,
    OFOH, OEOH, OC0H, 080H, 000H
DB  OFEH, OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, OFEH, OFCH, OF8H,
    OFOH, OEOH, OC0H, 080H, 000H
DB  OFCH, OF8H, OFOH, OEOH, OC0H, 080H, 000H, 07EH, 07CH, 078H, 070H,
    060H, 040H, 000H
DB  OF8H, OFOH, OEOH, OC0H, 080H, 000H, 03EH, 03CH, 038H, 030H, 020H,
    000H
DB  OFOH, OEOH, OC0H, 080H, 000H, 01EH, 01CH, 018H, 010H, 000H
DB  OEOH, OC0H, 080H, 000H, 00EH, 00CH, 008H, 000H
DB  OC0H, 080H, 000H, 006H, 004H, 000H
DB  080H, 000H, 002H, 000H
DB  000H, 000H

LEDCOUNT_TABLE:
DB  5,5 , 7,7 , 8,8 , 8,8 , 8,8 , 8,8 , 8,8 , 7,7 , 6,6 ,
    5,5 , 4,4 , 3,3 , 2,2 , 1,1

HEXTABLE:
;32H
DB  000H, 000H, 001H, 080H, 003H, OC0H, 007H, OEOH, 00FH, OFOH,
    01FH, OF8H, 03FH, OFCH, 07FH, OFEH
DB  OFFH, OFFH, OFFH, OFFH, OFFH, OFFH, OFFH, OFFH, OFFH,
    OFFH, 07CH, 03EH, 000H, 000H
;4DH
END

```

5. CODE C

```

#include <REGX51.H>
#define choncotcao 1
#define choncotthap 0
//1 lan quet tinh toan duoc 0.0175s
//su dung hoan toan mo phong
#define counter1s 52 //60 <->
1.10492079s
#define off_speed 6 // 4 <-> delta =
0.0702s

unsigned char hextable[] = //Ma quet LED ma trix
{
0x00, 0x00, 0x01, 0x80, 0x03, 0xC0, 0x07, 0xE0,
0xF0, 0x0F, 0x1F, 0xF8, 0x3F, 0xFC, 0x7F, 0xFE,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

```

```

0xFF, 0xFF, 0xFE, 0x7F, 0x7C, 0x3E, 0x00, 0x00
};

unsigned char ledcount[] = // So luong LED sang
{
    5,5 , 7,7 , 8,8 , 8,8 , 8,8 , 8,8 , 8,8 ,
    7,7 , 6,6 , 5,5 , 4,4 , 3,3 , 2,2 , 1,1 ,
};

unsigned char hextable_copy[33] ; // Backup gia tri Hex

volatile unsigned char xdata cotcao _at_ 0x1000;
volatile unsigned char xdata cotthap _at_ 0x1F00;
volatile unsigned char xdata hangcao _at_ 0x4000;
volatile unsigned char xdata hangthap _at_ 0x9000;

unsigned char counter_led; // bien dem so led con lai
unsigned char led_value; // thay doi HEX de LED tat
unsigned char delta_led; // luong led tat <=> luong b.i.t dich
unsigned char pter; // bien vong lap dich b.i.t

int pointer_quet, mode; // con tro quet HEX, MODE(0-29)
int cop; // bien vong lap copy HEX table
int x,y = 0; // bien vong lap thoi gian hien thi

void quetmatran(unsigned char m);
int ledoff(int n);

```

```

void delay_ms (int time) {
    //Ham Delay ms
    int j, k;
    for(j = 0; j < time; j++) {
        for (k = 0; k <115 ; k++)
            {}
    }
}

void quetmatran(unsigned char m) {      //Ham chon cot -
    quet

    unsigned char choncot = 0x01;
    int i;
    for( i = 0; i < 8; i++) {
        if( m == choncotthap) {
            cotthap = choncot;
        }
        else {
            cotcao = choncot;
        }

        hangcao = hextable[pointer_quet] ;
        pointer_quet++;
        hangthap = hextable[pointer_quet] ;
        pointer_quet++;
        delay_ms(1);

        choncot = choncot * 2;
    }

    choncot = 0;
    if( m == choncotthap) {
        cotthap = choncot;
    }
    else {
        cotcao = choncot;
    }
}

void tatled() {

```

```

counter_led = ledcount[mode]; //Lay so
LED tu` trong bang
led_valuex = ledoff(mode);
//Lay HEX thay doi tu` SWITCH

if(mode%2==0) {
delta_led = ledcount[mode+1] - counter_led;
//So LED con lai
}
else {
delta_led = ledcount[mode-1] - counter_led;
}

for(pter = 0; pter < delta_led ; pter++) {
    led_valuex = led_valuex * 2;
    // value LED <=> Dich b.i.t
}
hextable[29-mode] = hextable[29-mode] &
(~led_valuex); //Doi HEX
ledcount[mode] = ledcount[mode] - 1;

if ( ledcount[mode] == 255 ) {

    if(mode%2 == 0) {
        ledcount[mode] = ledcount[mode+1]; // 5<->5 ,
7<->7
    }
    else {
        ledcount[mode] = ledcount[mode-1]; // Doi
xung, tu backup
    }
    mode = mode + 1;
    // Kich MODE
    x = off_speed - 1;
    // Bo qua 0.1s
    khi chuyen MODE
}
}

//0x02,0x04,0x01,0x02,0x01,0x01,0x01,

```

```
//0x01,0x01,0x01,0x01,0x01,0x01,0x01,  
//0x02,0x01,0x04,0x01,0x08,0x01,0x10,  
//0x01,0x20,0x01,0x40,0x01,0x80,0x01  
  
int ledoff(int n) {  
    //Khong du RAM => SWITCHCASE thay TABLE  
    int value_off;  
    switch(n) {  
        case 0 : value_off = 0x02;  
                  break;  
        case 1 : value_off = 0x04;  
                  break;  
        case 2 : value_off = 0x01;  
                  break;  
        case 3 : value_off = 0x02;  
                  break;  
        case 4 : value_off = 0x01;  
                  break;  
        case 5 : value_off = 0x01;  
                  break;  
        case 6 : value_off = 0x01;  
                  break;  
        case 7 : value_off = 0x01;  
                  break;  
        case 8 : value_off = 0x01;  
                  break;  
        case 9 : value_off = 0x01;  
                  break;  
        case 10 : value_off = 0x01;  
                  break;  
        case 11 : value_off = 0x01;  
                  break;  
        case 12 : value_off = 0x01;  
                  break;  
        case 13 : value_off = 0x01;  
                  break;  
        case 14 : value_off = 0x02;  
                  break;  
        case 15 : value_off = 0x01;  
                  break;  
        case 16 : value_off = 0x04;
```

```

                break;
case 17 :   value_off = 0x01;
            break;
case 18 :   value_off = 0x08;
            break;
case 19 :   value_off = 0x01;
            break;
case 20 :   value_off = 0x10;
            break;
case 21 :   value_off = 0x01;
            break;
case 22 :   value_off = 0x20;
            break;
case 23 :   value_off = 0x01;
            break;
case 24 :   value_off = 0x40;
            break;
case 25 :   value_off = 0x01;
            break;
case 26 :   value_off = 0x80;
            break;
case 27 :   value_off = 0x01;
            break;
default:     value_off = 0;
            break;
}
return value_off;
}

void reset (void)           //Tat het LED thi
Reset

for(cop = 0 ; cop < 32 ; cop++){
    hextable[cop] = hextable_copy[cop]; //BACKUP
}

((void (code *) (void)) 0x0000) (); //Ve dau
ch.trinh
}

```

```

//void ngattimer0 (void) interrupt 1
//{
//}

void main () {

    mode = 0;
    for(cop = 0 ; cop < 32 ; cop++) {
        hextable_copy[cop] = hextable[cop];
    }

    while(1) {

        pointer_quet = 0;
        quetmatran(choncotthap);
        quetmatran(choncotcao);

        y++;
        if(y == counter1s) {
            y = counter1s - 1; //Neu Y dem du lan thi` truot qua Y
            x++;
            if(x == off_speed) {
                x = 0;
                tatled();
                if (mode == 28) {
                    reset();
                }
            }
        }
    }
}

```

6. Video chạy mô phỏng



VIDEO_VXL.mp4