Nguyễn Thiện Nhân **23134041**

**Câu 1:**

Dùng 2N2222, R1 = R2 = 10 kΩ, nguồn +5 V, STM32 xuất mức 3.3 V. Lấy gần đúng VBE​ = 0.7 V khi dẫn và VCE(sat) ​= 0.4 V.

**TH1: Logic 1 (3.3V):**

Dòng Base:

Dòng Collector:

Hệ số ép: ⇒ Q1 bão hòa, Y ≈ 0.4 V.

**Công suất & phân bố:**

Từ nguồn 5 V: P5V=5 × IC ≈ 2.3 mW.

Công suất tiêu tán trên R2: PR2 = IC × (5−0.4) = 2.116 mW.

Công suất tiêu tán trong Q1 (phần collector): PC = VCE(sat) × IC = 0.184  mW

Từ nguồn 3.3 V (dòng base): P3V3=3.3 × IB ≈ 0.858  mW.

Công suất tiêu tán trên R1: PR1=IB × (3.3−0.7) = 0.676 mW.

Công suất tiêu tán trong Q1 (mối BE, lấy VBE = 0.7V): PBE ​= 0.18 mW

Tổng công suất tiêu thụ của mạch: P5V + P3V3 = 3.158  mW.

Công suất toàn transistor: PQ = VCE x IC + VBE x IB =0.366 mW.

**TH2: Logic 0 (0 V):**

IB ≈ 0 ⇒ IC ≈0. Nút Y được kéo lên 5 V qua R2.

Dòng rò của 2N2222 cỡ chục nA → có thể coi công suất tiêu thụ ~ 0 mW (không tải ngoài).

**Câu 2:**

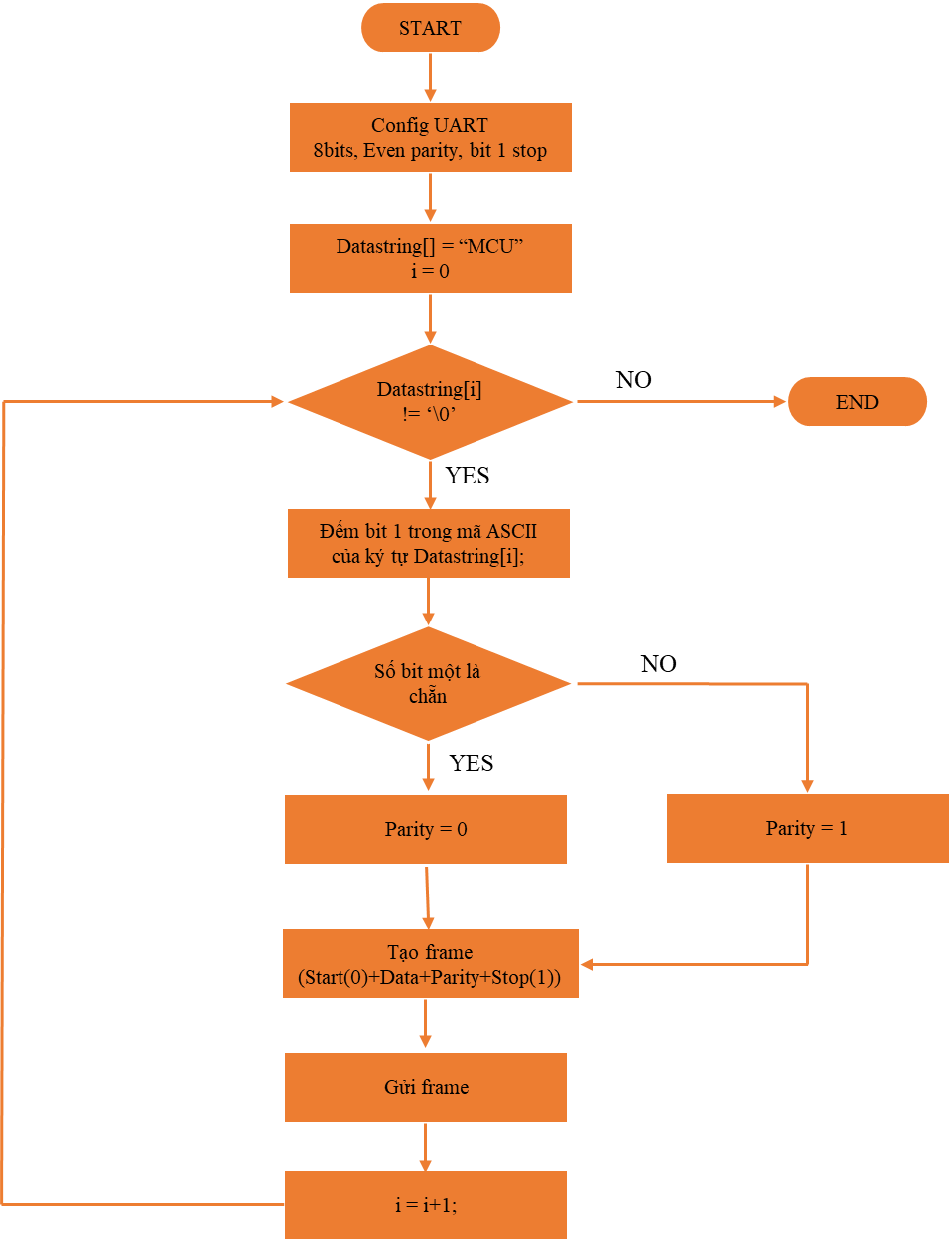
**2a.** Nếu STM32F103C8 gắn tải là đèn LED có rơi áp 1,5V, dòng điện 10mA thì điện trở hạn dòng cho LED là:

**2b.** Nếu tải là LED 7 đoạn có dòng mỗi LED 5mA, STM32F103C8 điều khiển tối đa được:

Dòng mỗi LED 7:

Điều khiển được tối đa:

**Câu 3:**

****

**Câu 4:**

Memory-Mapped I/O là cơ chế mà các thiết bị ngoại vi được ánh xạ trực tiếp vào không gian địa chỉ bộ nhớ. Khi CPU thực hiện lệnh đọc hoặc ghi tại địa chỉ của ngoại vi, lệnh này sẽ:

* Truy cập trực tiếp vào thanh ghi của ngoại vi.
* Thiết bị ngoại vi sẽ thực hiện hành vi tương ứng.

GPIOA: 0x40010800-0x400107FF

GPIOB: 0x40010C00-0x40010FFF

GPIOC: 0x40011000-0x400113FF

GPIOD: 0x40011400-0x400117FF

**Câu 5:** Tiêu chí lập trình nhúng

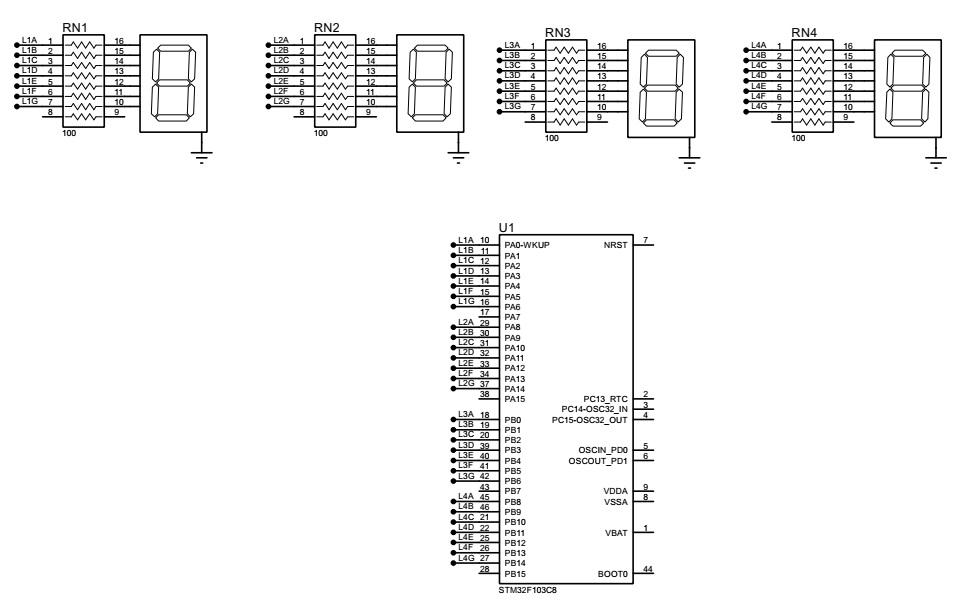
- Easy to debug, including both finding and fixing mistakes (Dễ gỡ lỗi, bao gồm cả việc tìm ra và sửa các lỗi). Lỗi trong hệ thống nhúng có thể gây treo thiết bị hoặc hỏng phần cứng, việc gỡ lỗi khó dẫn đến tốn thời gian và rủi ro cao.

- Easy to verify, meaning we can prove it is correct (Dễ kiểm chứng, có nghĩa là chúng ta có thể chứng minh chương trình là đúng). Hệ thống nhúng thường dùng trong ứng dụng quan trọng, thời gian thực hoặc an toàn, nếu không thể kiểm chứng, phần mềm có thể gây nguy hiểm hoặc không đáng tin cậy.

- Easy to maintain, meaning we can add new features (Dễ bảo trì, có nghĩa là chúng ta có thể thêm các tính năng mới). Hệ thống nhúng thường sử dụng lâu dài, cần cập nhật hoặc mở rộng chức năng.

Common error: Programmers who sacrifice clarity in favor of execution speed often develop software that runs fast but is error-prone and difficult to change (Lỗi phổ biến: Những lập trình viên hi sinh tính rõ ràng để ưu tiên tốc độ thực thi thường phát triển phần mềm chạy nhanh nhưng dễ xảy ra lỗi và khó thay đổi).

**Câu 6:**



|  |
| --- |
| #define NUM\_START (uint32\_t) 9999  typedef union {  uint8\_t ALL\_SEG;  struct  {  uint8\_t SEG\_A :1; // ->LSB  uint8\_t SEG\_B :1;  uint8\_t SEG\_C :1;  uint8\_t SEG\_D :1;  uint8\_t SEG\_E :1;  uint8\_t SEG\_F :1;  uint8\_t SEG\_G :1;  uint8\_t DP :1; // -> MSB  } SEG;  } SEGbit;  typedef struct {  uint32\_t PIN\_SEG\_A;  uint32\_t PIN\_SEG\_B;  uint32\_t PIN\_SEG\_C;  uint32\_t PIN\_SEG\_D;  uint32\_t PIN\_SEG\_E;  uint32\_t PIN\_SEG\_F;  uint32\_t PIN\_SEG\_G;  uint32\_t PIN\_SEG\_DP;    GPIO\_TypeDef\* PORT;  } LED\_pin;  char \_7seg\_code[] = {  0x3F, // -> 0  0x06, // -> 1  0x5B, // -> 2  0x4F, // -> 3  0x66, // -> 4  0x6D, // -> 5  0x7D, // -> 6  0x07, // -> 7  0x7F, // -> 8  0x6F, // -> 9  };  LED\_pin L1 = {  .PIN\_SEG\_A = GPIO\_PIN\_0,  .PIN\_SEG\_B = GPIO\_PIN\_1,  .PIN\_SEG\_C = GPIO\_PIN\_2,  .PIN\_SEG\_D = GPIO\_PIN\_3,  .PIN\_SEG\_E = GPIO\_PIN\_4,  .PIN\_SEG\_F = GPIO\_PIN\_5,  .PIN\_SEG\_G = GPIO\_PIN\_6,  .PORT = GPIOA  };  LED\_pin L2 = {  .PIN\_SEG\_A = GPIO\_PIN\_8,  .PIN\_SEG\_B = GPIO\_PIN\_9,  .PIN\_SEG\_C = GPIO\_PIN\_10,  .PIN\_SEG\_D = GPIO\_PIN\_11,  .PIN\_SEG\_E = GPIO\_PIN\_12,  .PIN\_SEG\_F = GPIO\_PIN\_13,  .PIN\_SEG\_G = GPIO\_PIN\_14,  .PORT = GPIOA  };  LED\_pin L3 = {  .PIN\_SEG\_A = GPIO\_PIN\_0,  .PIN\_SEG\_B = GPIO\_PIN\_1,  .PIN\_SEG\_C = GPIO\_PIN\_2,  .PIN\_SEG\_D = GPIO\_PIN\_3,  .PIN\_SEG\_E = GPIO\_PIN\_4,  .PIN\_SEG\_F = GPIO\_PIN\_5,  .PIN\_SEG\_G = GPIO\_PIN\_6,  .PORT = GPIOB  };  LED\_pin L4 = {  .PIN\_SEG\_A = GPIO\_PIN\_8,  .PIN\_SEG\_B = GPIO\_PIN\_9,  .PIN\_SEG\_C = GPIO\_PIN\_10,  .PIN\_SEG\_D = GPIO\_PIN\_11,  .PIN\_SEG\_E = GPIO\_PIN\_12,  .PIN\_SEG\_F = GPIO\_PIN\_13,  .PIN\_SEG\_G = GPIO\_PIN\_14,  .PORT = GPIOB  };  LED\_pin\* LED\_ARR[] = {&L1, &L2, &L3, &L4};  void LED\_put(LED\_pin \*led, uint8\_t num) {  SEGbit num\_split\_bit;  num\_split\_bit.ALL\_SEG = \_7seg\_code[num%10];  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_A, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_A);  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_B, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_B);  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_C, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_C);  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_D, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_D);  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_E, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_E);  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_F, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_F);  HAL\_GPIO\_WritePin(led->PORT, led->PIN\_SEG\_G, (GPIO\_PinState) num\_split\_bit.SEG.SEG\_G);  }  void LED\_Display(LED\_pin\* led[], uint32\_t num) {  for(int8\_t idx = 3; idx >= 0; idx--) {  LED\_put(led[idx], num%10);  num /= 10;  }  }  int main(void)  {  HAL\_Init();  SystemClock\_Config();  MX\_GPIO\_Init();  while (1)  {  LED\_Display(LED\_ARR, num);  HAL\_Delay(300);  if(num == 0) {  num = NUM\_START;  } else {  num--;  }  } |