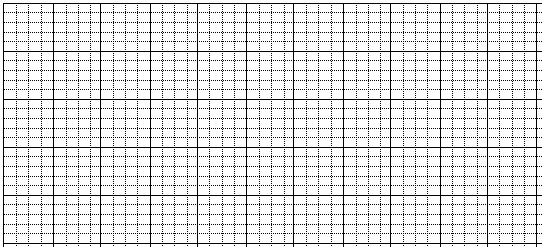
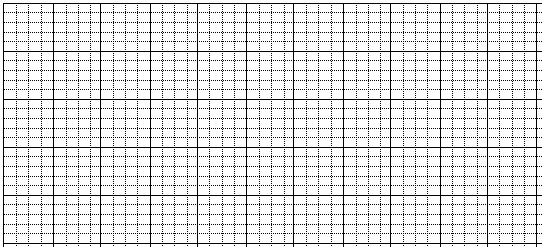
|  |  |
| --- | --- |
| **TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TPHCM**  **KHOA CƠ KHÍ CHẾ TẠO MÁY**  **BỘ MÔN CƠ ĐIỆN TỬ**  **Họ và tên: Võ Hồng Quân**  **Mã số SV: 22134012** | **BÁO CÁO**  **THỰC TẬP LẬP TRÌNH VI XỬ LÝ**  **Ngày:**  **Tuần học: 03**  **Nội dung: Giao tiếp thanh ghi dịch để điều khiển LED 7 đoạn** |

1. **Khảo sát sơ đồ nguyên lý:**

**Sinh viên vẽ sơ đồ nguyên lý khối LED 7 đoạn và giải thích**

****

**Sinh viên vẽ sơ đồ nguyên lý khối thanh ghi dịch và giải thích**

****

1. **Bảng kết nối input - output**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên phần cứng** | **Vị trí chân vi điều khiển tương ứng** | **Mô tả chức năng** |
| 1 | MOSI | PB5 | Chân dữ liệu đầu ra từ vi điều khiển |
| 2 | SCK | PB3 | Chân xung clock cho giao tiếp |
| 3 | RCL | PE2 | Chốt dữ liệu vào thanh ghi dịch |
| 4 | RCK | PE0 | Xung clock để đẩy dữ liệu ra các chân đầu ra |

1. **Lập trình**
   1. **Tính bảng mã LED 7 đoạn:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hiển thị** | **QA**  **F** | **QB**  **A** | **QC**  **E** | **QD**  **D** | **QE**  **DP** | **QF**  **B** | **QG**  **G** | **QH**  **C** | **Mã nhị phân** | **Mã HEX** |
| **0** | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 11111101 | 0xFD |
| **1** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 00001001 | 0x9 |
| **2** | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 10110111 | 0xB7 |
| **3** | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 10101111 | 0xAF |
| **4** | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 01001111 | 0x4F |
| **5** | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 11001011 | 0xCB |
| **6** | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 11011011 | 0xDB |
| **7** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 10100001 | 0xA1 |
| **8** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 11111111 | 0xFF |
| **9** | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 11001111 | 0xCF |
| **(Tắt LED)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000000 | 0x0 |
| **A** | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 11100111 | 0xE7 |
| **b** | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 11001101 | 0xCD |
| **C** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 00001111 | 0xF |
| **d** | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 11101100 | 0xEC |
| **E** | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 01001111 | 0x4F |
| **F** | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 01000111 | 0x47 |
| **H** | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 11100101 | 0xE5 |
| **i** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 00000101 | 0x5 |
| **J** | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 10101000 | 0xA8 |
| **L** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 00001101 | 0xD |
| **N** | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 10100111 | 0xA7 |
| **O** | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 10101111 | 0xAF |
| **P** | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 01100111 | 0x67 |
| **S** | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 11001011 | 0xCB |
| **U** | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 10101101 | 0xAD |

* 1. **Tạo thư viện myLED:**
* Viết thư viện *myLED.h* và *myLED.c* với các hàm và macro sau đây:
* Hàm Data8\_put(uint8\_t d8) để đẩy số kiểu 1 byte d8 ra thanh ghi dịch. Chân QH là MSB của d8; chân QA là LSB của d8.
* Hàm Data16\_put(uint16\_t d16) để đẩy số kiểu 2 byte d16 ra 2 thanh ghi dịch nối tiếp. Chân QH của thanh ghi dịch cuối là MSB của d16; chân QA là LSB của thanh ghi dịch đầu là LSB của d16.
* LED\_data(3) : Hien thi so 3
* Hàm LED\_data(uint8\_t n) để giải mã hiển thị số n lên LED 7 đoạn.
* Hàm LED\_put(uint8\_t n, uint8\_t p) để giải mã hiển thị số n lên LED 7 đoạn tại vị trí p. với p như sau:
  + p = 0: Tắt hết các LED
  + p = 1: Hiển thị LED vị trí k1 (00000001)
  + p = 2: Hiển thị LED vị trí k2 (00000010)
  + p = 4: Hiển thị LED vị trí k3 (00000100)
  + p = 8: Hiển thị LED vị trí k4 (00001000)
  + p = 16: Hiển thị LED vị trí k5 (00010000)
  + p = 3: Hiển thị LED vị trí k1 và k2. (00000011)
  + p = 7: Hiển thị LED vị trí k1 và k2 và k3. (00000110)
* Hàm LED\_putstring(uint8\_t \* s) để đẩy một chuỗi lên LED 7 đoạn:
  + Chỉ hiển thị tối đa 5 ký tự đầu tiên của chuỗi s
  + Nếu ký tự nằm ngoài bảng mã thì tắt hiển thị.
* myLED.h

|  |
| --- |
| #ifndef MYLED\_H  #define MYLED\_H  #include "main.h"  #include <stdint.h>  void Data8\_put(uint8\_t d8);  void Data16\_put(uint16\_t d16);  void LED\_data(uint8\_t n);  /\*\*  \* @brief Decodes and displays the digit n on the 7-segment LED at position p.  \* p = 0 : Turn off all LEDs.  \* p = 1 : Display at position k1 (00000001).  \* p = 2 : Display at position k2 (00000010).  \* p = 4 : Display at position k3 (00000100).  \* p = 8 : Display at position k4 (00001000).  \* p = 16: Display at position k5 (00010000).  \* p = 3 : Display at positions k1 and k2 (00000011).  \* p = 7 : Display at positions k1, k2, and k3 (00000110).  \* @param n: The digit (0-9) to display.  \* @param p: The position mask.  \*/  void LED\_put(uint8\_t n, uint8\_t p);  void LED\_putstring(uint8\_t \*s);  #define A 0xE7  #define b 0xCD  #define C 0x0F  #define d 0xEC  #define E 0x4F  #define F 0x47  #define H 0xE5  #define I 0x05  #define J 0xA8  #define L 0x0D  #define N 0xA7  #define O 0xB7  #define P 0x67  #define S 0xCB  #define U 0xAD  #define LED\_OFF 0x00  #endif // MYLED\_H |

* myLED.c

|  |
| --- |
| #include "myLED.h"  #include "main.h"  void Data8\_put(uint8\_t d8) {  for (int i = 0; i < 8; i++) {  // Iterate from MSB to LSB: check the bit (0x80 >> i)  HAL\_GPIO\_WritePin(MOSI\_GPIO\_Port, MOSI\_Pin,  ((d8 & (0x80 >> i)) ? GPIO\_PIN\_SET : GPIO\_PIN\_RESET));  // Clock pulse: set SCK low then high.  HAL\_GPIO\_WritePin(SCK\_GPIO\_Port, SCK\_Pin, GPIO\_PIN\_RESET);  HAL\_GPIO\_WritePin(SCK\_GPIO\_Port, SCK\_Pin, GPIO\_PIN\_SET);  }  // Latch the data by toggling RCK.  HAL\_GPIO\_WritePin(RCK\_GPIO\_Port, RCK\_Pin, GPIO\_PIN\_RESET);  HAL\_GPIO\_WritePin(RCK\_GPIO\_Port, RCK\_Pin, GPIO\_PIN\_SET);  }  void Data16\_put(uint16\_t d16) {  for (int i = 0; i < 16; i++) {  HAL\_GPIO\_WritePin(MOSI\_GPIO\_Port, MOSI\_Pin,  ((d16 & (0x8000 >> i)) ? GPIO\_PIN\_SET : GPIO\_PIN\_RESET));  HAL\_GPIO\_WritePin(SCK\_GPIO\_Port, SCK\_Pin, GPIO\_PIN\_RESET);  HAL\_GPIO\_WritePin(SCK\_GPIO\_Port, SCK\_Pin, GPIO\_PIN\_SET);  }  HAL\_GPIO\_WritePin(RCK\_GPIO\_Port, RCK\_Pin, GPIO\_PIN\_RESET);  HAL\_GPIO\_WritePin(RCK\_GPIO\_Port, RCK\_Pin, GPIO\_PIN\_SET);  }  void LED\_data(uint8\_t n) {  uint8\_t LED\_num[10] = { 0xFD, 0x09, 0x6E, 0xEA, 0xE1, 0xCB, 0xD7, 0xA2, 0xEF, 0xEB };  if (n < 10)  Data8\_put(LED\_num[n]);  else  Data8\_put(n);  }  void LED\_put(uint8\_t n, uint8\_t p) {  if (p == 0) {  Data16\_put(0x0000);  return;  }  uint8\_t LED\_num[10] = { 0xFD, 0x09, 0x6E, 0xEA, 0xE1, 0xCB, 0xD7, 0xA2, 0xEF, 0xEB };    uint8\_t digit;  if (n < 10)  digit = LED\_num[n];  else  digit = n;  uint16\_t pos\_mask = (uint16\_t)p << 8;  uint16\_t d16 = pos\_mask | digit;  Data16\_put(d16);  }  static uint8\_t LED\_decode\_char(uint8\_t ch) {  if (ch >= '0' && ch <= '9') {  uint8\_t LED\_num[10] = { 0xFD, 0x09, 0x6E, 0xEA, 0xE1, 0xCB, 0xD7, 0xA2, 0xEF, 0xEB };  return LED\_num[ch - '0'];  }  switch(ch) {  case 'A': return A;  case 'b': return b;  case 'C': return C;  case 'd': return d;  case 'E': return E;  case 'F': return F;  case 'H': return H;  case 'I': return I;  case 'J': return J;  case 'L': return L;  case 'N': return N;  case 'O': return O;  case 'P': return P;  case 'S': return S;  case 'U': return U;  default: return LED\_OFF;  }  }  void LED\_putstring(uint8\_t \*s) {  // Position masks for up to 5 digits: k5, k4, k3, k2, k1  // (Each value will be shifted into the high byte)  uint8\_t pos[5] = {0x08, 0x10, 0x20, 0x40, 0x80};  // LED\_putstring((uint8\_t \*)"AAAAA");  for (int i = 0; i < 5; i++) {  uint8\_t code;  // If we've reached the end of the string, turn off this digit  if (s[i] == '\0') {  code = LED\_OFF;  } else {  // Decode the character into a 7-segment code  code = LED\_decode\_char(s[i]);  }  // pos[i] -> shifted into the high byte, 'code' stays in the low byte  uint16\_t d16 = ((uint16\_t)pos[i] << 8) | code;    // Send the 16-bit value to the shift registers  Data16\_put(d16);  // Small delay to stabilize multiplexed display  HAL\_Delay(2);  }  } |

* 1. **Lập trình ứng dụng 1**
* Khi mới mở điện đèn LED 7 đoạn hiển thị 5 số cuối của mã số sinh viên.

|  |
| --- |
| while (1)  // Display 22134012  { LED\_putstring((uint8\_t \*)"34012"); } |

* 1. **Lập trình ứng dụng 2**

Viết chương trình cấp dữ liệu cho IC74HC595 để giải mã hiển thị LED 7 đoạn

-         Mới mở điện lên LED hiển thị số 12345.

-         Nhấn nút P1 số hiển thị tăng lên 1 đơn vị (Ví dụ từ 12345 tăng lên 12346)

-         Nhấn nút P2 số hiển thị giảm đi 1 đơn vị (Ví dụ từ 12345 giảm thành 12344)

-         Nhấn nút P3 số hiển thị trở về 12345

-         Nhấn nút P4 số hiển thị tự đếm lên, mỗi giây tăng 1 đơn vị. Tăng đến 99999 thì dừng.

|  |
| --- |
| #define P\_ACTIVE GPIO\_PIN\_RESET // Assuming active-low buttons  int main(void)  {  HAL\_Init();  SystemClock\_Config();  MX\_GPIO\_Init();  uint32\_t num\_display = 12345; // Start value  uint8\_t auto\_increment = 0; // Flag for auto-increment mode  uint32\_t last\_increment\_time = 0; // Stores the last time auto-increment happened  while (1)  {  // Convert num\_display to a string  char str\_num[6]; // 5 digits + null terminator  sprintf(str\_num, "%05lu", num\_display); // Convert number to string with 5 digits  // Display the number on the 7-segment display  LED\_putstring((uint8\_t \*)str\_num);  // Read button states  uint8\_t P1 = HAL\_GPIO\_ReadPin(P1\_GPIO\_Port, P1\_Pin);  uint8\_t P2 = HAL\_GPIO\_ReadPin(P2\_GPIO\_Port, P2\_Pin);  uint8\_t P3 = HAL\_GPIO\_ReadPin(P3\_GPIO\_Port, P3\_Pin);  uint8\_t P4 = HAL\_GPIO\_ReadPin(P4\_GPIO\_Port, P4\_Pin);  // P1: Increment number  if (P1 == P\_ACTIVE) {  num\_display++;  while (HAL\_GPIO\_ReadPin(P1\_GPIO\_Port, P1\_Pin) == P\_ACTIVE); // Wait for button release  }  // P2: Decrement number  if (P2 == P\_ACTIVE) {  num\_display--;  while (HAL\_GPIO\_ReadPin(P2\_GPIO\_Port, P2\_Pin) == P\_ACTIVE); // Wait for button release  }  // P3: Reset to 12345  if (P3 == P\_ACTIVE) {  num\_display = 12345;  while (HAL\_GPIO\_ReadPin(P3\_GPIO\_Port, P3\_Pin) == P\_ACTIVE); // Wait for button release  }  // P4: Start/stop auto-increment mode  if (P4 == P\_ACTIVE) {  auto\_increment = !auto\_increment; // Toggle auto-increment mode  while (HAL\_GPIO\_ReadPin(P4\_GPIO\_Port, P4\_Pin) == P\_ACTIVE); // Wait for button release  }  // Auto-increment mode (non-blocking)  if (auto\_increment && num\_display < 99999) {  if (HAL\_GetTick() - last\_increment\_time >= 1000) { // Check if 1 second has passed  num\_display++;  last\_increment\_time = HAL\_GetTick(); // Reset timer  }  }  // Prevent overflow (max 99999)  if (num\_display > 99999) {  num\_display = 99999;  }  }  } |

* 1. **Lập trình ứng dụng 3**

Lập trình hệ thống hiển thị LED 7 đoạn đồng hồ như sau:

* Khi mới mở điện, LED 7 đoạn hiển thị dạng phút và giây MM-SS . Ví dụ: 12-00 (12 phút 00 giây)
* Mỗi khi nhấn P1 thì bật/tắt chế độ đếm lên tự động (mỗi giây tăng 1 đơn vị giây hiển thị. (Lưu ý khi giây tăng 59 thì chuyển phút tăng 1 đơn vị).
* Mỗi khi nhấn P2 thì bật/tắt chế độ đếm xuống tự động (mỗi giây giảm 1 đơn vị giây hiển thị).
* Nút nhấn P3 và P4 sử dụng để điều chỉnh giá trị hiển thị phút và giây như sau:  
  + Mỗi lần nhấn P3 thì hiển thị LED 7 đoạn chuyển lần lượt qua 3 chế độ:
    - Chế độ chỉnh giây (2 LED giây nhấp nháy 0.5s liên tục);
    - Chế độ chỉnh phút (2 LED phút nhấp nháy 0.5s liên tục);
    - Chế độ nghỉ (LED hiển thị bình thường, không LED nào nhấp nháy hay tắt).
  + Mỗi lần nhấn P4:
    - Nếu đang ở chế độ chỉnh giây thì P4 làm tăng giá trị hiển thị giây 1 đơn vị;
    - Nếu đang ở chế độ chỉnh phút thì P4 làm tăng giá trị hiển thị phút 1 đơn vị;
    - Nếu đang ở chế độ nghỉ thì P4 không có tác dụng gì.
    - Lưu ý khi chỉnh giây hoặc phút tăng đến 59 thì lặp lại 00. Tăng giây từ 59 sang 00 không làm tăng giá trị của phút.

|  |
| --- |
| #define P\_ACTIVE GPIO\_PIN\_RESET // Active-low buttons  // Clock variables  uint8\_t minutes = 12; // Initial minute value  uint8\_t seconds = 00; // Initial second value  uint8\_t auto\_increment = 0; // Flag for auto-increment mode (P1)  uint8\_t auto\_decrement = 0; // Flag for auto-decrement mode (P2)  uint8\_t adjust\_mode = 0; // 0 = Normal, 1 = Adjust Seconds, 2 = Adjust Minutes  uint32\_t last\_update\_time = 0; // Timer for updating time  uint32\_t blink\_time = 0; // Timer for blinking effect  uint8\_t blink\_state = 1; // State for LED blinking  void update\_display(void) {  char str[6]; // Format: "MM-SS"  sprintf(str, "%02d-%02d", minutes, seconds); // Format the time string  if (adjust\_mode == 1) { // Blinking mode for seconds  if (HAL\_GetTick() - blink\_time >= 5) { // 0.5s toggle  blink\_time = HAL\_GetTick();  blink\_state = !blink\_state;  }  if (!blink\_state) {  str[3] = str[4] = ' '; // Hide seconds  }  } else if (adjust\_mode == 2) { // Blinking mode for minutes  if (HAL\_GetTick() - blink\_time >= 5) { // 0.5s toggle  blink\_time = HAL\_GetTick();  blink\_state = !blink\_state;  }  if (!blink\_state) {  str[0] = str[1] = ' '; // Hide minutes  }  }  LED\_putstring((uint8\_t \*)str);  }  int main(void)  {  HAL\_Init();  SystemClock\_Config();  MX\_GPIO\_Init();  while (1)  {  // \*\*Time Update Logic\*\* (Every 1 second)  if (HAL\_GetTick() - last\_update\_time >= 1000) {  last\_update\_time = HAL\_GetTick();  if (auto\_increment) { // Auto count up mode  seconds++;  if (seconds > 59) {  seconds = 0;  minutes = (minutes + 1) % 60; // Rolls over at 59  }  }  if (auto\_decrement) { // Auto count down mode  if (seconds == 0) {  seconds = 59;  minutes = (minutes == 0) ? 59 : (minutes - 1);  } else {  seconds--;  }  }  update\_display(); // Refresh display  }  // \*\*Read Button Inputs\*\*  uint8\_t P1 = HAL\_GPIO\_ReadPin(P1\_GPIO\_Port, P1\_Pin);  uint8\_t P2 = HAL\_GPIO\_ReadPin(P2\_GPIO\_Port, P2\_Pin);  uint8\_t P3 = HAL\_GPIO\_ReadPin(P3\_GPIO\_Port, P3\_Pin);  uint8\_t P4 = HAL\_GPIO\_ReadPin(P4\_GPIO\_Port, P4\_Pin);  // \*\*P1: Toggle Auto Increment Mode\*\*  if (P1 == P\_ACTIVE) {  auto\_increment = !auto\_increment;  auto\_decrement = 0; // Stop decrement mode if active  while (HAL\_GPIO\_ReadPin(P1\_GPIO\_Port, P1\_Pin) == P\_ACTIVE);  }  // \*\*P2: Toggle Auto Decrement Mode\*\*  if (P2 == P\_ACTIVE) {  auto\_decrement = !auto\_decrement;  auto\_increment = 0; // Stop increment mode if active  while (HAL\_GPIO\_ReadPin(P2\_GPIO\_Port, P2\_Pin) == P\_ACTIVE);  }  // \*\*P3: Toggle Adjust Mode (Cycle through 3 modes)\*\*  if (P3 == P\_ACTIVE) {  adjust\_mode = (adjust\_mode + 1) % 3;  blink\_time = HAL\_GetTick(); // Reset blink timer  blink\_state = 1; // Reset blink state  while (HAL\_GPIO\_ReadPin(P3\_GPIO\_Port, P3\_Pin) == P\_ACTIVE);  }  // \*\*P4: Adjust Seconds or Minutes\*\*  if (P4 == P\_ACTIVE) {  if (adjust\_mode == 1) { // Adjust seconds  seconds = (seconds + 1) % 60;  } else if (adjust\_mode == 2) { // Adjust minutes  minutes = (minutes + 1) % 60;  }  update\_display(); // Refresh after adjustment  while (HAL\_GPIO\_ReadPin(P4\_GPIO\_Port, P4\_Pin) == P\_ACTIVE);  }  update\_display(); // Continuous refresh  }  } |