به‌نام آفریننده بیت‌ها

دانشکده‌ی برق و کامپیوتر
دانشگاه صنعتی اصفهان
نیم‌سال اول ۱۴۰۱ - ۱۴۰۲

ریزپردازنده

توضیحات پروژه

اعضای پروژه:
دانیال خراسانی‌زاده
پارسا صادقیان
استاد درس:
دکتر کریمی افشار

۱ عملکرد اصلی سیستم

در این پروژه بازی خاطره انگیز مار را بازسازی می‌کنیم. در این بازی یک مار در صفحه قرار دارد و هدف این است که این مار بدون برخورد با بدن خودش، نقاط غذایی که به صورت رندوم در صفحه بازی ظاهر می‌شوند را خورده و با هر بار خوردن این نقاط یک نقطه به طول مار اضافه می‌شود. این بازی تا زمانی که مار با خودش برخورد کند ادامه دارد.

۲ رابط SPI

رابط Serial Peripheral Interface (SPI) یک رابط همزمان سریال است که در دهه هشتاد میلادی توسط شرکت موتورولا طراحی و به یکی از مهم‌ترین رابط‌ها در دنیای سیستم‌های تعبیه شده تبدیل شد. نحوه کار این رابط به صورت Full Duplex و Master - Slave (یک دستگاه اصلی و یک یا چند دستگاه متصل به دستگاه اصلی) بوده و برای برقراری ارتباط به چهار پین نیاز دارد:

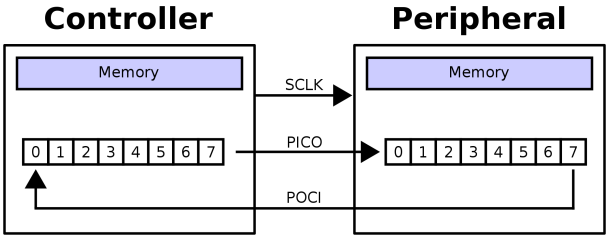
۱. $SCLK$ (Serial Clock): این پین که از دستگاه اصلی یا مستر خارج می‌شود کلاک انتقال اطلاعات را مشخص می‌کند.

۲. $COPI$ (Controller Out Peripheral In): این پین که خروجی دستگاه اصلی است برای ارسال اطلاعات به دستگاه وابسته به کار می‌رود.

۳. $CIPO$ (Controller In Peripheral Out): این پین که ورودی دستگاه اصلی است برای ارسال اطلاعات از دستگاه وابسته به دستگاه اصلی به کار می‌رود.

۴. $\overline{CS}$ (Chip Select): این پین برای انتخاب و فعال کردن دستگاه فرعی به کار می‌رود.

نحوه کار این رابط به این صورت است که ابتدا دستگاه اصلی یکی از دستگاه‌های فرعی را فعال می‌کند. سپس با یک کلاک که توسط دستگاه فرعی نیز پشتیبانی می‌شود، دیتای مورد نظر خود را به صورت سریال از طریق پین COPI به دستگاه فرعی فرستاده و همزمان اگر نیاز به گرفتن دیتا داشته باشد این دیتا را از طریق پین CIPO دریافت می‌کند.



۳ آی‌سی MAX7221

این آی‌سی توسط شرکت Maxim Integrated تولید شده و برای ایجاد ارتباط بین دستگاه‌های میکروکنترلر و ماتریس‌های ال‌ای‌دی هشت در هشت یا هشت سون‌سگمنت کاتد مشترک به کار می‌رود. در اینجا به طور کلی و در حد نیاز این پروژه به نحوه کار این آی‌سی می‌پردازیم: این دستگاه از سیزده رجیستر ۸ بیتی و یک شیفت رجیستر ۱۶ بیتی تشکیل شده. برای ارسال دستور به این دستگاه از رابط SPI استفاده می‌کنیم و با توجه به نحوه اتصال شیفت رجیستر به مدارهای کنترلی ابتدا آدرس مورد نظر و سپس دیتای مورد نظر را به صورتی که MSB آن‌ها اول ارسال شود برای آی‌سی ارسال می‌کنیم. رجیستر صفر این دستگاه عملکرد خاصی ندارد و برای عمل No-Op استفاده می‌شود. رجیسترهای یک تا هشت این دستگاه برای انتخاب یکی از ستون‌های ماتریس ال‌ای‌دی یا یک سون سگمنت و روشن کردن آن به کار می‌روند. رجیستر نه، نوع دیکد شدن دیتای درون رجیسترهای یک تا هشت را مشخص می‌کند و اگر دیتای مورد نظر ما BCD باشد می‌توانیم با تنظیم این رجیستر آن را به صورت صحیح روی سون سگمنت نمایش دهیم اما در این پروژه با توجه به اینکه نیازی به این عملکرد نداریم آن را غیر فعال می‌کنیم. رجیستر ده این دستگاه برای تنظیم شدت نور ال‌ای‌دی‌ها از طریق عملکرد PWM تعبیه شده درون آی‌سی به کار می‌رود. رجیستر یازده این دستگاه تعداد سون سگمنت‌های متصل را مشخص می‌کند. با توجه به اینکه در این پروژه از این آی‌سی به عنوان درایور ماتریس ال‌ای‌دی استفاده می‌کنیم باید عدد هشت را در این رجیستر قرار دهیم. رجیستر دوازده این دستگاه خاموش یا روشن بودن عملکرد آن را مشخص کرده و رجیستر سیزده نیز برای تست نمایشگر استفاده می‌شود.

۴ توضیح کد پروژه

در این قسمت به توضیح کلی هر یک از توابعی که در کد پروژه استفاده شده به ترتیبی که از تابع `main()` فراخوانی شده‌اند خواهیم پرداخت:

۱.۴ `void input_buttons_init(void)` (خط ۲۴)

این تابع پین‌های صفر تا سه پورت C که برای ورودی دکمه‌های کنترل بازی استفاده می‌شود را عنوان ورودی تنظیم کرده و مقاومت پول‌آپ آن‌ها را نیز فعال می‌کند.

۲.۴ `void SPI_init(void)` (خط ۲۸)

این تابع پین‌هایی از پورت B که برای کار با رابط SPI استفاده می‌شوند را روی خروجی تنظیم کرده و همچنین رجیستر کنترلی SPI را فعال کرده و روی حالت مستر قرار می‌دهد.

۳.۴ `void SPI_send(uint8_t register_address, uint8_t data)` (خط ۳۲)

این تابع با گرفتن یک آدرس و یک مقدار، این مقدار را از طریق رابط SPI برای آی‌سی MAX7221 ارسال می‌کند. نحوه کار این تابع به این صورت است که ابتدا آی‌سی را فعال کرده و سپس آدرس مورد نظر را در رجیستر دیتای SPI قرار می‌دهد. سپس تا نوشته شدن کامل دیتا صبر کرده و این روند را برای دیتا تکرار می‌کند و در نهایت با اتمام این فرایند، آی‌سی را غیر فعال می‌کند.

۴.۴ `void LED_matrix_init(void)` (خط ۴۲)

این تابع رجیسترهای کنترلی آی‌سی MAX7221 را طبق چیزی که پیشتر در توضیح این آی‌سی گفته شد، تنظیم می‌کند.

۵.۴ `void timer0_init(void)` (خط ۴۸)

این تابع، تایمر ۰ را طوری تنظیم می‌کند که با اسکیل $\frac{\text{Clock}}{256}$ کار کرده و با هر بار اورفلو شدن خود، وقفه متناظر با این اتفاق را فعال کند. سپس در روتین سرویس دهی به این وقفه تابع `update_buttons_status()` را فراخوانی می‌کنیم تا اگر یکی از دکمه‌های کنترلی بازی فشرده شده باشد، عمل متناظر با این دکمه را انجام دهد.

۶.۴ `inline void button_update(uint8_t *button, uint8_t pin_value)` (خط ۵۲)

این تابع مقدار متناظر با یک دکمه را گرفته، آن را یک واحد به سمت چپ شیفت داده و مقدار پین متناظر با این دکمه را در LSB مقدار جدید قرار می‌دهد.

۷.۴ `void update_buttons_status(void)` (خط ۵۵)

در این تابع ابتدا مقادیر هر یک از دکمه‌های کنترلی را بروزرسانی می‌کنیم. با توجه به اینکه ممکن است با فشردن دکمه به خاطر خاصیت فنر مانند دکمه‌های فیزیکی، یک دکمه چند بار فشرده شود، با استفاده از این عمل منتظر می‌مانیم تا ۸ بار به صورت متوالی دکمه را فعال ببینیم و سپس به دستور آن دکمه عمل کنیم. پس از به‌روز رسانی وضعیت دکمه‌ها به صورت اولویت دار جهت جدید را از یکی دکمه‌هایی که در حال حاضر فعال هستند انتخاب می‌کنیم.

۸.۴ `void snake_init(void)` (خط ۷۰)

در این تابع نقطه اولیه و جهت اولیه مار را تعیین می‌کنیم. نقطه شروع به صورت رندوم و جهت اولیه بالا انتخاب می‌شود.

۹.۴ `uint8_t update_and_return_turn_ons(void)` (خط ۷۵)

این تابع تعداد دفعات روشن شدن دستگاه از آخرین باری که دستگاه پروگرم شده را از EEPROM خوانده، یکی به آن اضافه کرده و پس از نوشتن مقدار جدید آن را برای استفاده به عنوان سید رندوم باز می‌گرداند. (می‌توان از خواندن یک ورودی ADC نیز سید رندوم تولید کرد، با توجه به اینکه با ۲۵۶ بار روشن شدن دستگاه، اورفلو رخ داده و سید تکرار می‌شود روش فعلی، روش چندان خوبی نیست اما با توجه به اینکه در این پروژه نیازی به دقت بالای اعداد رندوم نداریم، از پیاده سازی سید با ADC صرفنظر شد.)

۱۰.۴ `void random_food_location(void)` (خط ۸۱)

این تابع برای قرار دادن غذای بعدی مار در یک نقطه رندوم به کار می‌رود، با توجه به اینکه غذا نباید در نقطه‌ای که مار در آن قرار دارد ظاهر شود، این تابع ابتدا یک نقطه رندوم انتخاب کرده و اگر این نقطه با مار برخورد داشت آنقدر این روند را تکرار می‌کند تا نقطه جدیدی که با مار برخورد ندارد پیدا شود.

۱۱.۴ `inline int16_t modulo_8(int16_t x)` (خط ۹۵)

در زبان سی، اپراتور باقیمانده مقداری هم علامت با عدد ورودی برمی‌گرداند. با توجه به اینکه در محاسبات برنامه نیاز به مقادیر باقیمانده به پیمانه ۸ که بین ۰ تا ۷ باشند داریم، از این تابع استفاده می‌کنیم.

۱۲.۴ `void display(void)` (خط ۹۶)

این تابع برای تبدیل نمایش مار و غذا به نمایش ماتریس ال‌ای‌دی به کار می‌رود. در این تابع ابتدا تمام ستون‌های صفحه را صفر می‌کنیم. سپس با گذر روی تمام قسمت‌های مار، آن‌ها را در ستون و سطر صحیح یک کرده و همین کار را برای نقطه غذا انجام می‌دهیم. در نهایت نیز این صفحه محاسبه شده جدید را برای نمایش ارسال می‌کنیم.

۱۳.۴ `void wait(uint16_t ms)` (خط ۱۰۸)

با توجه به اینکه مقدار پاس داده شده به تابع `_delay_ms(double __ms)` باید در زمان کامپایل معلوم باشد و ما نیاز به تاخیر با مقدار دینامیک داریم، در این تابع به تعداد ورودی تاخیر یک میلی‌ثانیه‌ای ایجاد می‌کنیم.

۱۴.۴ `bool has_snake_eaten_the_food(void)` (خط ۱۱۱)

این تابع، نقطه‌ای که در حرکت بعد سر مار به آن می‌رسد را پیدا کرده و اگر این نقطه، همان نقطه غذا بود، مقدار true و در غیر اینصورت مقدار false باز می‌گرداند.

۱۵.۴ `void move_snake(void)` (خط ۱۲۹)

در این تابع ابتدا از نقطه آخر مار شروع کرده و هر نقطه را در جایگاه نقطه قبلی خود قرار می‌دهیم. سپس با توجه به جهت فعلی مار، تقطه بعدی سر مار را پیدا کرده و جای آن را عوض می‌کنیم.

۱۶.۴ `bool has_snake_collided_with_itself(void)` (خط ۱۵۱)

این تابع برای چک کردن برخورد مار با خودش به کار می‌رود. با توجه به اینکه اگر طول مار کمتر از ۵ باشد، برخورد ممکن نیست ابتدا با چک کردن طول مار اگر این شرط برقرار بود مقدار false باز می‌گردانیم. در صورتی که این شرط برقرار نبود، قسمت‌های بدن مار را چک کرده و اگر مختصات هر یک از این قسمت‌ها با سر مار یکسان بود برخورد مار با خودش را اعلام می‌کنیم.

۱۷.۴ `void display_game_over_animation(void)` (خط ۱۵۹)

با استفاده از ارسال فرمان‌های این تابع، در هنگام برخورد مار با خودش و پایان بازی، یک صورتک ناراحت که چشم‌هایش به صورت متناوب باز و بسته می‌شوند نمایش داده می‌شود.

۱۸.۴ `int main(void)` (خط ۱۸۰)

در ابتدای تابع اصلی، توابع آماده سازی قسمت‌های مختلف که پیشتر توضیح داده شد را فراخوانی می‌کنیم. پس از فعال کردن وقفه‌ها وارد چرخه اصلی بازی می‌شویم. در این چرخه ابتدا صفحه بازی را نمایش داده و سپس به اندازه از پیش تعیین شده‌ای صبر می‌کنیم. سپس با توجه به جهت فعلی مار و جهت مشخص شده توسط دکمه‌ها اگر جهت مشخص شده تغییر کرده و جهت جدید در خلاف جهت فعلی نباشد، جهت مار را تغییر می‌دهیم. پس از تغییر جهت، با چک کردن اینکه مار غذا را خورده اگر این اتفاق افتاده باشد، یکی به طول مار اضافه کرده، سرعت بازی را افزایش داده، مار را حرکت داده و یک نقطه غذای جدید انتخاب می‌کنیم. در غیر اینصورت فقط مار را حرکت می‌دهیم. در نهایت چک می‌کنیم که آیا مار با خودش برخورد داشته یا نه و اگر این اتفاق افتاده باشد، صفحه بازی را برای آخرین بار نمایش داده و پس از مقداری صبر، انیمیشن پایان یافتن بازی را نمایش می‌دهیم.

۵ کد پروژه

```
#define F_CPU 16000000UL
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <util/delay.h>
#define COPI 5
#define SCK 7
#define SS 4
enum direction { UP, DOWN, RIGHT, LEFT };
uint8_t button_up = 0;
uint8_t button_down = 0;
uint8_t button_right = 0;
uint8_t button_left = 0;
enum direction current_direction = UP;
uint8_t snake_length = 1;
uint8_t snake[64][2];
enum direction snake_direction;
uint8_t food_location[2];
uint8_t screen[8];
uint8_t EEMEM TURN_ONS = 0;
void input_buttons_init(void) {
  DDRC = 0x00;
  PORTC = 0x0F;
}
void SPI_init(void) {
  DDRB = (1 << COPI) | (1 << SCK) | (1 << SS);
  SPCR = (1 << SPE) | (1 << MSTR);
}
void SPI_send(uint8_t register_address, uint8_t data) {
  PORTB &= ~(1 << SS);
  SPDR = register_address;
  while (!(SPSR & (1 << SPIF))) {
  }
  SPDR = data;
  while (!(SPSR & (1 << SPIF))) {
  }
  PORTB |= (1 << SS);
}
void LED_matrix_init(void) {
  SPI_send(0x09, 0x00);
  SPI_send(0x0A, 0x0F);
  SPI_send(0x0B, 0x07);
  SPI_send(0x0C, 0x01);
}
void timer0_init(void) {
  TIMSK |= 1 << TOIE0;
  TCCR0 = (1 << CS00) | (1 << CS01);
}
inline void button_update(uint8_t *button, uint8_t pin_value) {
  *button = (*button << 1) | pin_value;
}
void update_buttons_status(void) {
  button_update(&button_up, (~PINC & (1 << PINC0)) >> PINC0);
```

```
  button_update(&button_right, (~PINC & (1 << PINC1)) >> PINC1);
  button_update(&button_down, (~PINC & (1 << PINC2)) >> PINC2);
  button_update(&button_left, (~PINC & (1 << PINC3)) >> PINC3);
  if (button_up == 0xFF) {
    current_direction = UP;
  } else if (button_down == 0xFF) {
    current_direction = DOWN;
  } else if (button_right == 0xFF) {
    current_direction = RIGHT;
  } else if (button_left == 0xFF) {
    current_direction = LEFT;
  }
}
void snake_init(void) {
  snake[0][0] = rand() % 8;
  snake[0][1] = rand() % 8;
  snake_direction = UP;
}
uint8_t update_and_return_turn_ons(void) {
  uint8_t turn_ons = eeprom_read_byte(&TURN_ONS);
  turn_ons += 1;
  eeprom_update_byte(&TURN_ONS, turn_ons);
  return turn_ons;
}
void random_food_location(void) {
  bool food_snake_collision;
  do {
    food_snake_collision = false;
    food_location[0] = rand() % 8;
    food_location[1] = rand() % 8;
    for (uint8_t i = 0; i < snake_length; i++) {
      if (food_location[0] == snake[i][0] && food_location[1] == snake[i][1]) {
        food_snake_collision = true;
        break;
      }
    }
  } while (food_snake_collision);
}
inline int16_t modulo_8(int16_t x) { return (x % 8 + 8) % 8; }
void display(void) {
  for (uint8_t i = 0; i < 8; i++) {
    screen[i] = 0;
  }
  for (uint8_t i = 0; i < snake_length; i++) {
    screen[snake[i][0]] |= 1 << modulo_8(snake[i][1] - 1);
  }
  screen[food_location[0]] |= 1 << modulo_8(food_location[1] - 1);
  for (uint8_t i = 1; i < 9; i++) {
    SPI_send(i, screen[i - 1]);
  }
}
void wait(uint16_t ms) {
  while (ms--) _delay_ms(1);
}
bool has_snake_eaten_the_food(void) {
  switch (snake_direction) {
    case UP:
      return ((food_location[0] == snake[0][0]) &&
              (food_location[1] == modulo_8(snake[0][1] + 1)));
```

```
    case DOWN:
      return ((food_location[0] == snake[0][0]) &&
              (food_location[1] == modulo_8(snake[0][1] - 1)));
    case RIGHT:
      return ((food_location[0] == modulo_8(snake[0][0] - 1)) &&
              (food_location[1] == snake[0][1]));
    case LEFT:
      return ((food_location[0] == modulo_8(snake[0][0] + 1)) &&
              (food_location[1] == snake[0][1]));
    default:
      return false;
  }
}
void move_snake(void) {
  for (uint8_t i = snake_length - 1; i > 0; i--) {
    snake[i][0] = snake[i - 1][0];
    snake[i][1] = snake[i - 1][1];
  }
  switch (snake_direction) {
    case UP:
      snake[0][1] = modulo_8(snake[0][1] + 1);
      break;
    case DOWN:
      snake[0][1] = modulo_8(snake[0][1] - 1);
      break;
    case RIGHT:
      snake[0][0] = modulo_8(snake[0][0] - 1);
      break;
    case LEFT:
      snake[0][0] = modulo_8(snake[0][0] + 1);
      break;
    default:
      break;
  }
}
bool has_snake_collided_with_itself(void) {
  if (snake_length < 5) return false;
  for (uint8_t i = 4; i < snake_length; i++) {
    if ((snake[0][0] == snake[i][0]) && (snake[0][1] == snake[i][1]))
      return true;
  }
  return false;
}
void display_game_over_animation(void) {
  SPI_send(1, 0b00000001);
  SPI_send(2, 0b00000010);
  SPI_send(3, 0b01110100);
  SPI_send(4, 0b00000100);
  SPI_send(5, 0b00000100);
  SPI_send(6, 0b01110100);
  SPI_send(7, 0b00000010);
  SPI_send(8, 0b00000001);
  _delay_ms(1000);
  SPI_send(1, 0b00000001);
  SPI_send(2, 0b00100010);
  SPI_send(3, 0b00100100);
  SPI_send(4, 0b00100100);
  SPI_send(5, 0b00100100);
  SPI_send(6, 0b00100100);
```

```
  SPI_send(7, 0b00100010);
  SPI_send(8, 0b00000001);
  _delay_ms(1000);
}
ISR(TIMER0_OVF_vect) { update_buttons_status(); }
int main(void) {
  bool is_snake_dead = false;
  uint16_t speed_factor = 650;
  input_buttons_init();
  SPI_init();
  LED_matrix_init();
  timer0_init();
  snake_init();
  srand(update_and_return_turn_ons());
  random_food_location();
  sei();
  while (!is_snake_dead) {
    display();
    wait(speed_factor);
    if (!((snake_direction == current_direction) ||
          (snake_direction == UP && current_direction == DOWN) ||
          (snake_direction == LEFT && current_direction == RIGHT) ||
          (snake_direction == DOWN && current_direction == UP) ||
          (snake_direction == RIGHT && current_direction == LEFT))) {
      snake_direction = current_direction;
    }
    if (has_snake_eaten_the_food()) {
      speed_factor -= 10;
      snake_length += 1;
      move_snake();
      random_food_location();
    } else {
      move_snake();
    }
    if ((is_snake_dead = has_snake_collided_with_itself())) {
      display();
      _delay_ms(1000);
    }
  }
  while (true) {
    display_game_over_animation();
  }
  return EXIT_SUCCESS;
}
```