Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

ОТЧЁТ ПО КУРСОВОЙ РАБОТЕ

по дисциплине

«Встраиваемое программное обеспечение и микропроцессорные системы»

Создание игры на базе микропроцессора Raspberry Pi

Выполнил: Симоненко И. С.
Группа: 5130904/20102
№ зач. Книжки: 22350270
Преподаватель: Иночкин Ф. М.

Санкт-Петербург
2024

Оглавление

1. Условие задачи

Задача команды, реализовать игру на базе микропроцессора Raspberry Pi с использованием системы реального времени.

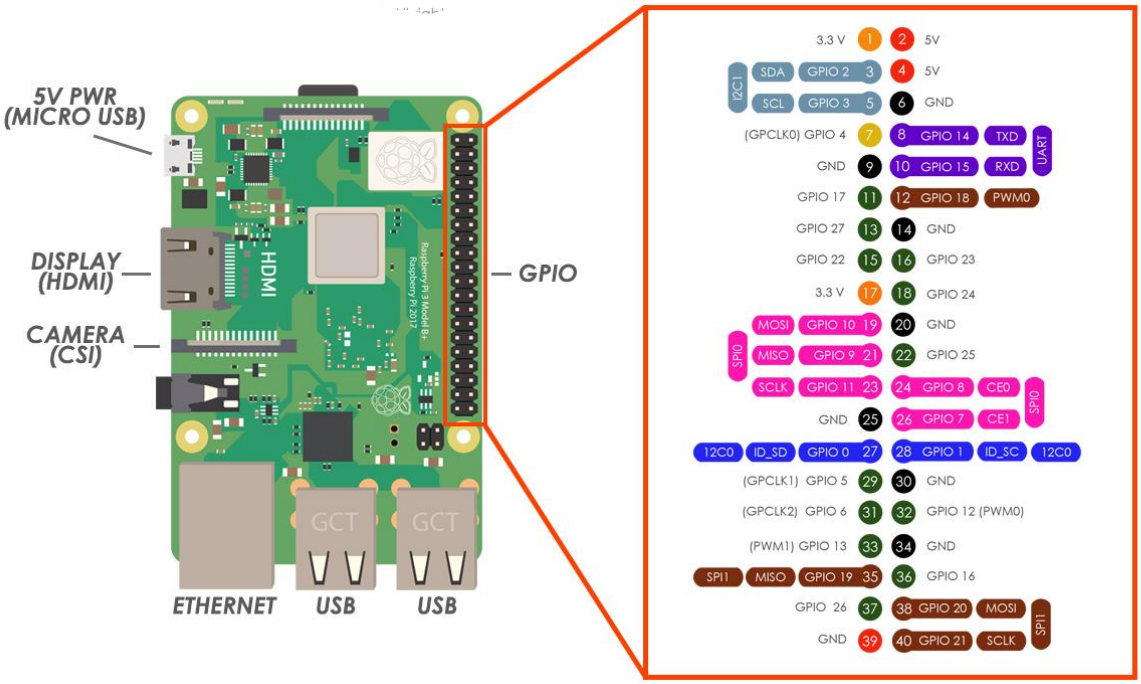
2. Используемое оборудование

В качестве операционной системы используется Raspbian OS — основанная на Debian операционная система для Raspberry Pi. Таким образом на данной плате мы можем не только запустить проект, но и вывести результаты благодаря порту HDMI в конструкции. Ниже представлены технические характеристики:

Микропроцессор
Raspberry Pi 3 Model B+ — это улучшенная версия Raspberry Pi 3 Model B с обновлёнными характеристиками и расширенными возможностями подключения. Она популярна благодаря высокой производительности, универсальности и поддержке множества интерфейсов.

1. **Процессор**:
   - Broadcom BCM2837B0, 4 ядра Cortex-A53, тактовая частота 1.4 ГГц.
2. **Графика**:
   - Broadcom VideoCore IV (графический процессор).
3. **Оперативная память**:
   - 1 ГБ LPDDR2 SDRAM.
4. **Сети**:
   - Wi-Fi: 802.11ac (2.4 ГГц и 5 ГГц).
   - Bluetooth 4.2 (BLE).
   - Ethernet: Gigabit Ethernet через USB 2.0 (до 300 Мбит/с).
5. **Использованная операционная система**:
   - Linux rpi 6.6.51+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.51-1+rpt3 (2024-10-08) aarch64 GNU/Linux

3. Элементы платы



Raspberry Pi 3 Model B+ имеет **40-контактный GPIO** и множество других портов для подключения периферии.

1. GPIO (General Purpose Input/Output)

- **Количество пинов**: 40.
- **Основные функции**:
  - 26 программируемых пинов общего назначения.
  - Поддержка протоколов:
    - I²C (GPIO2 и GPIO3).
    - SPI (GPIO7, GPIO8, GPIO9, GPIO10, GPIO11).
    - UART (GPIO14 и GPIO15).
    - ШИМ (GPIO12, GPIO13, GPIO18, GPIO19).
- **Особенности**:
  - Напряжение пинов: 3.3 В.
  - Максимальный ток: до 16 мА на пин (не более 50 мА суммарно).

---

2. USB порты

- **Количество**: 4 порта USB 2.0.
- **Назначение**:
  - Подключение периферийных устройств (клавиатуры, мыши, флеш-накопителей и т. Д.).
  - Поддержка USB Wi-Fi адаптеров или USB-модемов.

---

3. HDMI

- **Тип разъёма**: полноразмерный HDMI.
- **Назначение**:
  - Вывод видео и аудио на монитор или телевизор.

- Поддержка разрешения до 1920x1080 (Full HD) при 60 кадрах в секунду.

---

4. Ethernet (RJ-45)

- **Скорость**: до 300 Мбит/с через USB 2.0.
- **Особенности**:
  - Ethernet работает через встроенный USB-адаптер, что ограничивает пропускную способность.
  - Используется для подключения к проводной сети.

---

5. Wi-Fi и Bluetooth

- **Wi-Fi**:
  - Поддержка двух диапазонов: 2.4 ГГц и 5 ГГц.
  - Соответствие стандарту 802.11ac.
- **Bluetooth**:
  - Версия 4.2 с поддержкой BLE (Bluetooth Low Energy).

---

6. Микро-USB (питание)

- **Напряжение питания**: 5 В.
- **Максимальный ток**: до 2.5 А.
- **Назначение**:
  - Основное питание устройства.

---

7. Комбинированный аудио/видео выход (3.5 мм джек)

- **Функции**:
  - Вывод аналогового аудио (стерео).
  - Вывод композитного видео (CVBS).
- **Особенности**:
  - Требуется соответствующий кабель для подключения.

---

8. Микро-SD слот

- **Назначение**:
  - Для установки операционной системы.
  - Хранение пользовательских данных.
- **Поддержка карт**: microSDHC, microSDXC.

---

9. Камера и дисплей

- **Камера (CSI, Camera Serial Interface)**:
  - Для подключения камер Raspberry Pi (например, Pi Camera).
- **Дисплей (DSI, Display Serial Interface)**:
  - Для подключения сенсорных дисплеев Raspberry Pi.

---

10. LED-индикаторы

- **PWR**: показывает наличие питания.
- **ACT**: индикатор активности microSD карты.
- **LAN**: светодиоды активности Ethernet (жёлтый и зелёный).

4. Распиновка

LCD дисплей

Я использую интерфейс I2C.

Схема: RPi - I2C LCD module

5V - VCC

GND - ЗАЗЕМЛЕНИЕ

Контакт 3 (GPIO 2) - SDA

Контакт 5 (GPIO 3) - SLC

Кнопки

Я использую матричную клавиатуру 4x4 (16 кнопок), но я использую только 4 угловые кнопки для 4 действий.

Я подключил 8 контактов к gpio: C4, C3, C2, C1, R1, R2, R3, R4 -> 16, 20, 21, 5, 6, 13, 19, 26

Схема: Кнопка на матрице (GPIO_Column, GPIO_Row) - действие

S1 (5, 6) - движение вверх;

S4 (16, 6) - перемещение вправо;

S13 (5, 26) - перемещение влево;

S16 (16, 26) - перемещение вниз;

“ee” – (ск. easter egg) «пасхалка» с фамилиями авторов работы
“pause” – пауза с мониторингом состояния системы

```
52    # Настройка матричной клавиатуры
53    KEYPAD = [
54        ["up", None, None, "right"],
55        [None, None, "pause", None],
56        ["ee", None, None, None],
57        ["left", None, None, "down"]
58    ]
```

5. Используемые технологии

I2C протокол

I2C (Inter-Integrated Circuit) — это протокол последовательной передачи данных, широко используемый для соединения микроконтроллеров с различными периферийными устройствами, такими как датчики, дисплеи, часы реального времени, EEPROM и другие. На Raspberry Pi поддержка I2C встроена, что делает его удобным для использования с различной периферией. Так же этот протокол позволяет удобно работать с Python
Особенности I2C

1. Двухпроводной интерфейс:
    - SDA (Serial Data): передача данных.
    - SCL (Serial Clock): синхронизация данных.
2. Мастер-ведомый протокол: Raspberry Pi обычно выступает в роли мастера, управляющего ведомыми устройствами.
3. Адресация устройств: каждое ведомое устройство имеет уникальный адрес (7-битный или 10-битный).
4. Поддержка нескольких устройств: возможно подключение нескольких устройств к одной шине

GPIO

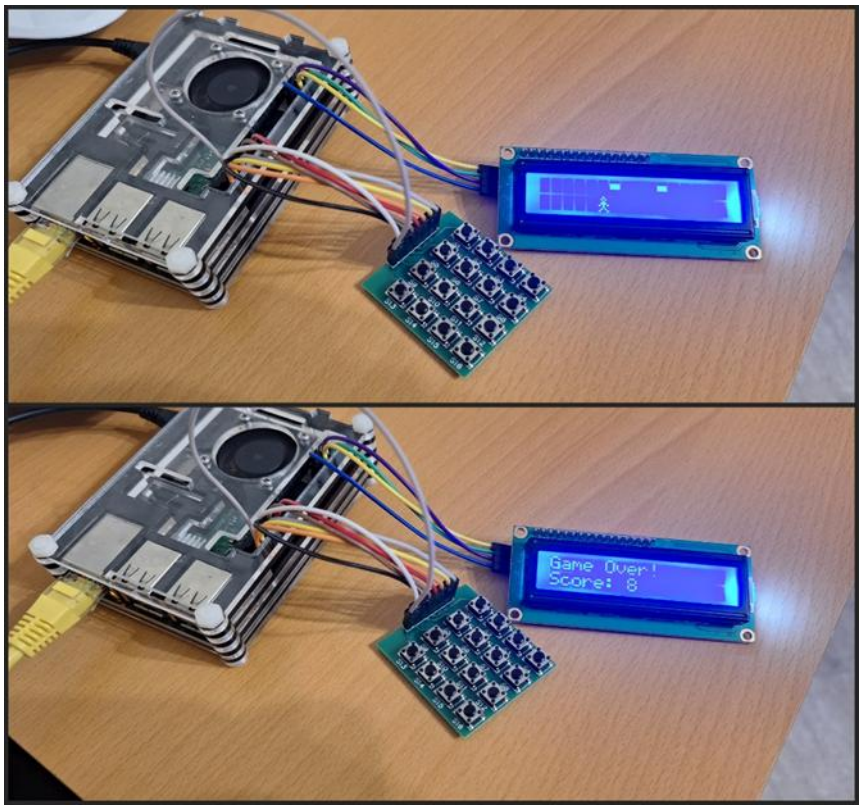
GPIO (General Purpose Input/Output) — это интерфейс общего назначения для ввода/вывода сигналов, доступный на Raspberry Pi. GPIO позволяет подключать и управлять различными внешними устройствами, такими как светодиоды, кнопки, реле, датчики и многое другое. Это одна из ключевых особенностей Raspberry Pi, делающая его мощным инструментом для прототипирования и работы с электроникой.
Основные характеристики GPIO Raspberry Pi:

1. Универсальность:
   Пины могут быть настроены как входы или выходы, а также поддерживать различные протоколы, такие как I2C, SPI и UART. В нашем случае важна работа с протоколом I2C
2. Уровни напряжения:
   GPIO работают с напряжением 3.3 В, и подключение устройств с уровнем 5 В может повредить Raspberry Pi без использования уровневого преобразователя.
3. Физическое расположение пинов:
   Количество GPIO зависит от модели Raspberry Pi. Например, у нас есть 40 пинов, из которых часть является GPIO.

Система реального времени

Система реального времени (Real-Time System) — у нас происходит прямое взаимодействие с системой в реальном времени. Для автозапуска программы при подаче питания на плату используется сервис (systemctl). Мы работаем с через интерфейсы, такие как GPIO, I2C. Для этой задачи мы используем GPIO из модуля RPi.GPIO и CharLCD из модуля RPLCD.i2c соответственно. Для получения информации о состоянии процессора и оперативной памяти используется модуль psutil.

6. Результаты работы



7. Код программы

```
import time
import random
import RPi.GPIO as GPIO
from RPLCD.i2c import CharLCD
import psutil  # Импортируем библиотеку psutil для мониторинга системы

GPIO.setwarnings(False)

lcd = CharLCD(i2c_expander='PCF8574', address=0x27, port=1,
              cols=16, rows=2, dotsize=8,
              charmap='A02',
              auto_linebreaks=True,
              backlight_enabled=True)
lcd.clear()
stickman = (
    0b00100,
    0b01010,
    0b00100,
    0b01110,
    0b10101,
    0b00100,
    0b01010,
    0b10001
)
```

```
obstacle = (
    0b00000,
    0b00000,
    0b11111,
    0b11111,
    0b11111,
    0b00000,
    0b00000,
    0b00000
)
bonus = (
    0b00000,
    0b00000,
    0b01010,
    0b00100,
    0b11111,
    0b00100,
    0b01010,
    0b00000
)

lcd.create_char(1, stickman)
lcd.create_char(2, obstacle)
lcd.create_char(3, bonus)

# Настройка матричной клавиатуры
KEYPAD = [
    ["up", None, None, "right"],
    [None, None, "pause", None],
    ["ee", None, None, None],
    ["left", None, None, "down"]
]

ROWS = [6, 13, 19, 26]
COLS = [5, 21, 20, 16]

GPIO.setmode(GPIO.BCM)

for row_pin in ROWS:
    GPIO.setup(row_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

for col_pin in COLS:
    GPIO.setup(col_pin, GPIO.OUT)
    GPIO.output(col_pin, GPIO.HIGH)

def get_key():
    key = None
    for col_num, col_pin in enumerate(COLS):
        GPIO.output(col_pin, GPIO.LOW)
        for row_num, row_pin in enumerate(ROWS):
            if GPIO.input(row_pin) == GPIO.LOW:
```

```
                key = KEYPAD[row_num][col_num]
                while GPIO.input(row_pin) == GPIO.LOW:
                    time.sleep(0.05)
        GPIO.output(col_pin, GPIO.HIGH)
    return key


class Player:
    def __init__(self):
        self.x = 2
        self.y = 1
        self.update()

    def update(self):
        lcd.cursor_pos = (self.y, self.x)
        lcd.write_string(chr(1))

    def move(self, direction):
        lcd.cursor_pos = (self.y, self.x)
        lcd.write_string(' ')
        if direction == 'up' and self.y > 0:
            self.y -= 1
        elif direction == 'down' and self.y < 1:
            self.y += 1
        elif direction == 'left' and self.x > 0:
            self.x -= 1
        elif direction == 'right' and self.x < 12:
            self.x += 1
        self.update()


class Obstacle:
    def __init__(self, x=12):
        self.x = x
        self.y = random.randint(0, 1)
        self.type = 'obstacle'

    def update(self):
        lcd.cursor_pos = (self.y, self.x)
        lcd.write_string(chr(2) if self.type == 'obstacle' else chr(3))

    def move(self):
        lcd.cursor_pos = (self.y, self.x)
        lcd.write_string(' ')
        self.x -= 1
        self.update()


def display_system_status():
    # Получаем информацию о загрузке процессора и памяти
    cpu_usage = psutil.cpu_percent()
```

```
    memory = psutil.virtual_memory()
    memory_usage = memory.percent

    lcd.cursor_pos = (0, 0)
    lcd.write_string(f'CPU: {cpu_usage}%')
    lcd.cursor_pos = (1, 0)
    lcd.write_string(f'Mem: {memory_usage}%')


def game(best_score):
    player = Player()
    obstacles = []
    game_over = False
    score = 0
    speed = 0.5
    last_obstacle_x = 12
    paused = False
    lives = 3

    def display_status():
        lcd.cursor_pos = (0, 13)
        lcd.write_string(f'{score:03}')
        lcd.cursor_pos = (1, 13)
        lcd.write_string(f'{best_score:03}')
        lcd.cursor_pos = (1, 0)
        lcd.write_string(f'{lives}')

    display_status()

    while not game_over:
        key = get_key()
        if key:
            if key == 'pause':
                paused = not paused
                if paused:
                    lcd.clear()  # Очистить экран, чтобы показать данные о си-
стеме
                    while paused:  # Пока игра на паузе, показываем системную
информацию
                        display_system_status()  # Отображаем информацию о за-
грузке процессора и памяти
                        time.sleep(1)  # Обновляем данные каждую секунду
                        key = get_key()  # Проверяем, не снята ли пауза
                        if key == 'pause':
                            paused = False  # Снимаем паузу, если игрок нажал
кнопку
                            lcd.clear()  # Очистим экран перед возобновлением
игры
                            display_status()  # Покажем текущий статус игры
                            player.update()
                            for obstacle in obstacles:
```

```
                                obstacle.update()
                            break  # Прерываем цикл, продолжаем игру
                continue
            if key == 'ee':
                paused = not paused
                if paused:
                    lcd.cursor_pos = (0, 0)
                    lcd.write_string('Simonenko\r\nMitroshin')
                else:
                    lcd.clear()
                    player.update()
                    for obstacle in obstacles:
                        obstacle.update()
                    display_status()
                continue
            if not paused:
                player.move(key)

        if not paused:
            if random.random() < 0.1 and (len(obstacles) == 0 or last_obstacle_x
- obstacles[-1].x >= 2):
                new_obstacle = Obstacle()
                if random.random() < 0.2:
                    new_obstacle.type = 'bonus'
                obstacles.append(new_obstacle)
                last_obstacle_x = new_obstacle.x

            for obstacle in obstacles:
                obstacle.move()
                if (obstacle.y == 0 and obstacle.x == 0) or (obstacle.y == 1 and
obstacle.x == 1):
                    obstacles.remove(obstacle)
                    if obstacle.type == 'obstacle':
                        score += 1
                if obstacle.x == player.x and obstacle.y == player.y:
                    if obstacle.type == 'bonus':
                        score += random.randint(2, 5)
                        obstacles.remove(obstacle)
                    else:
                        lives -= 1
                        obstacles.remove(obstacle)
                        if lives == 0:
                            game_over = True

            time.sleep(speed)
            lcd.clear()
            display_status()
            player.update()
            for obstacle in obstacles:
                obstacle.update()
```

```
    lcd.clear()
    if score > best_score:
        best_score = score
        lcd.write_string(f'New Record!\r\nScore: {score}')
    else:
        lcd.write_string(f'Game Over!\r\nScore: {score}')
    time.sleep(3)
    return best_score

def main():
    best_score = 0
    while True:
        best_score = game(best_score)
        lcd.clear()
        lcd.write_string('Press any key\r\nto restart')
        while not get_key():
            time.sleep(0.1)
        lcd.clear()

try:
    main()
except KeyboardInterrupt:
    lcd.clear()
    lcd.close(clear=True)
    GPIO.cleanup()
```

8. Зависимости

```
RPi.GPIO==0.7.1
RPLCD==1.3.1
smbus2==0.5.0
psutil==6.1.0
```

9. Список литературы и электронных источников

https://micro-pi.ru/raspberry-pi-3-model-a%2B-rpi-plus-bcm2837b0/#___ENG
http://wiki.amperka.ru/products:raspberry-pi-3-model-a-plus
https://amperka.ru/product/troyka-temperature-humidity-sensor-dht11#docs
http://wiki.amperka.ru/%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%8B:troyka-dht11
https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf

Репозиторий проекта: https://github.com/simonoffcc/rpi-lcd-game/