Университет ИТМО

Проектирование вычислительных систем

Лабораторная работа №1

Интерфейсы ввода-вывода общего назначения (GPIO)

ФИО студентов:
Готовко Алексей Владимирович
Руденко Илья Александрович
Вариант: 5
Направление подготовки: 09.03.04 (СППО)
Учебная группа: P34101
ФИО преподавателя: Пинкевич Василий Юрьевич

Санкт-Петербург
2024г.

Содержание

1 Цели работы **2**

2 Вариант задания **2**

3 Описание программы **2**

4 Блок-схема алгоритма **4**

5 Код программы **5**

6 Вывод **7**

1 Цели работы

1. Получить базовые знания о принципах устройства стенда `SDK-1.1M` и программировании микроконтроллеров;
2. изучить устройство интерфейсов ввода-вывода общего назначения (GPIO) в микроконтроллерах и приемы использования данных интерфейсов.

2 Вариант задания

Реализовать «кодовый замок». После ввода единственно верной последовательности не менее чем из восьми коротких и длинных нажатий должен загореться зелёный светодиод, обозначающий «открытие» замка. Светодиод горит некоторое время, потом гаснет, и система вновь переходит в режим ввода. Каждый неправильно введённый элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием жёлтого. После трёх неправильных вводов начинает мигать красный светодиод, и через некоторое время возвращается в режим ввода. Если за некоторое ограниченное время код не введен до конца, происходит сброс в «начало»

3 Описание программы

Будем поддерживать следующий набор переменных:

- `start_time` – момент начала отсчета времени зажатия кнопки;
- `code` – массив, хранящий последовательность длинных и коротких нажатий, составляющих пароль;
- `ptr` – указатель на текущий элемент массива `code`;
- `last_pressed_time` – момент последнего нажатия кнопки;
- `failed_attempts_cnt` – счетчик количества ошибок ввода пароля.

И следующий набор констант:

- `SHORT_PRESS_MAX_MS = 500` – максимальное время зажатия кнопки, соответствующее короткому нажатию;
- `RESET_ALL_TIMEOUT_MS = 3000` – время таймаута при вводе пароля.

На каждой итерации программы будем выполнять следующую последовательность действий:

1. Если `ptr` не равен нулю (т.е. на предыдущих операциях уже был введен хотя бы один правильный символ пароля) и с момента последнего нажатия на кнопку прошло необходимое время таймаута `RESET_ALL_TIMEOUT_MS`, то это значит, что на кнопку не нажимали достаточно долго, чтобы произошел сброс ввода пароля. В таком случае установим значения переменных `ptr` и `failed_attempts_cnt` в ноль и поморгаем зеленым и красным светодиодом одновременно, чтобы сообщить пользователю о сбросе.
2. Считаем текущее состояние кнопки.
3. Если кнопка зажата:
   - если момент начала отсчета времени зажатия не установлен (`start_time == 0`), то на предыдущей итерации кнопка еще не была зажата, а значит на текущей итерации нужно начать отсчет и обновить `start_time`;
   - случай с `start_time != 0` означает, что на предыдущей итерации кнопка уже была зажата; в таком случае просто продолжаем ожидать ее отжатия.

Возвращаемся в начало цикла.

4. Если кнопка не зажата и момент времени отсчета не задан (`start_time == 0`), то и на предыдущей, и на текущей итерации кнопка не была зажата. Тогда возвращаемся в начало цикла и ожидаем нажатия.

5. На этом шаге мы точно знаем, что на предыдущей итерации кнопка еще была зажата, а на текущей итерации ее отжали. Тогда вычислим длительность нажатия `press_time` и проверим, что она не меньше 20 мс, чтобы исключить случай дребезга контактов: если `press_time` меньше 20 мс, то возвращаемся в начало цикла, в противном случае обновим значение `last_pressed_time`.

6. Установим, какое нажатие произошло, короткое или длинное, сравнив `press_time` с `SHORT_PRESS_MAX_MS`. Зафиксируем тип нажатия в переменной `button_press_type`.

7. Далее сравним `button_press_type` с текущим правильным типом нажатия (длинное/короткое) `code[ptr]`. Если нажатие правильное:

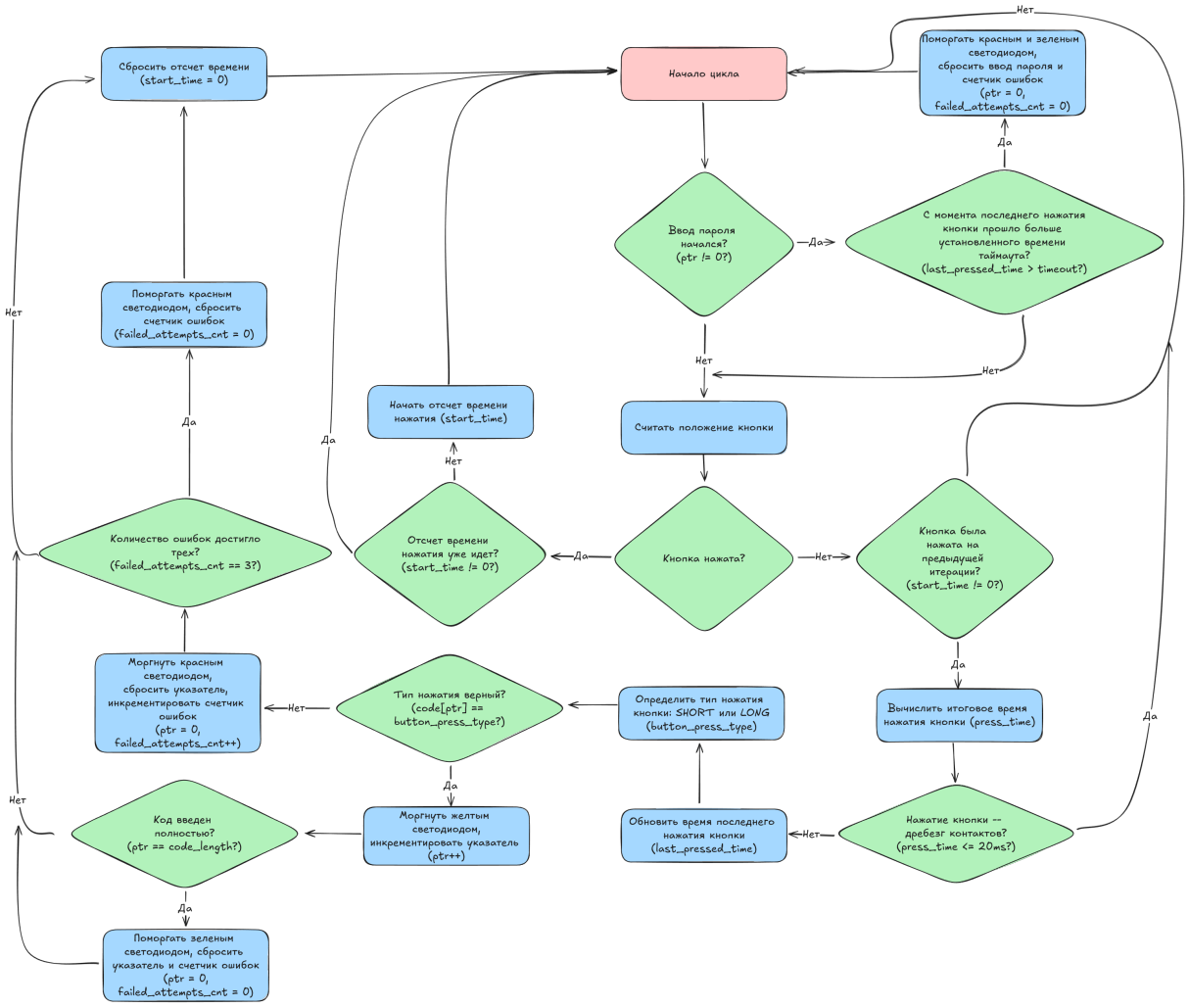
   - моргнем желтым светодиодом;
   - инкрементируем указатель `ptr`;
   - сравним `ptr` с длинной массива `code`: их равенство означает, что код введен полностью, а значит нужно поморгать зеленым светодиодом и занулить значения переменных `ptr` и `failed_attempts_cnt`.

   Если нажатие неправильное:

   - моргнем красным светодиодом;
   - сбросим указатель массива `code` в начальное положение (занулим `ptr`);
   - инкрементируем счетчик ошибок `failed_attempts_cnt`;
   - если `failed_attempts_cnt` стал равен трем, то поморгаем красным светодиодом и сбросим значение счетчика ошибок в ноль.

8. Наконец, сбросим значение момента начала отсчета времени зажатия кнопки `start_time` в ноль.

4 Блок-схема алгоритма



5 Код программы

Вспомогательные определения:

```
typedef enum {
    SHORT = 0,
    LONG
} ButtonPressType;

#define BUTTON_PIN              GPIO_PIN_15
#define GREEN_LED_PIN           GPIO_PIN_13
#define YELLOW_LED_PIN          GPIO_PIN_14
#define RED_LED_PIN             GPIO_PIN_15
#define SHORT_PRESS_MAX_MS      500
#define RESET_ALL_TIMEOUT_MS    3000
```

Инициализация портов GPIO:

```
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

    GPIO_InitStruct.Pin = GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

Функции-драйверы и вспомогательные функции:

```
GPIO_PinState read_button() {
    return HAL_GPIO_ReadPin(GPIOC, BUTTON_PIN);
}

uint32_t max(uint32_t a, uint32_t b) {
    return a > b ? a : b;
}

void light_led(uint16_t led_pin, uint8_t blink_cnt, uint8_t time) {
    while (blink_cnt--) {
        HAL_GPIO_WritePin(GPIOD, led_pin, GPIO_PIN_SET);
        HAL_Delay(time);
        HAL_GPIO_WritePin(GPIOD, led_pin, GPIO_PIN_RESET);
        HAL_Delay(100);
    }
}
```

Основной цикл:

```
uint32_t start_time = 0;
uint32_t press_time;
ButtonPressType button_press_type;
ButtonPressType code[] = {SHORT, SHORT, SHORT, SHORT, LONG, LONG, LONG, LONG};
uint8_t code_length = sizeof(code) / sizeof(code[0]);
uint8_t ptr = 0;
uint8_t failed_attempts_cnt = 0;
uint32_t last_pressed_time = 0;

while (1) {
    // if password input has started, terminate input process (reset state) if button
    ↪  was pressed last time more than `RESET_ALL_TIMEOUT_MS` ago
    if (ptr != 0 && HAL_GetTick() - last_pressed_time >= RESET_ALL_TIMEOUT_MS) {
        ptr = 0;
        failed_attempts_cnt = 0;
        light_led(RED_LED_PIN | GREEN_LED_PIN, 5, 200);
        last_pressed_time = HAL_GetTick();
        continue;
    }

    GPIO_PinState button_state = read_button();

    // if button was just pressed, start counting time
    if (button_state == GPIO_PIN_RESET) {
        if (start_time == 0) start_time = max(HAL_GetTick(), 1);
        continue;
    }

    // if button is not pressed now and was not pressed one iteration ago, just continue
    if (start_time == 0) continue;
    press_time = HAL_GetTick() - start_time;

    // contact bounce protection
    if (press_time <= 20) continue;
```

```
34
35          // if button is pressed and it's not a contact bounce, update `last_pressed_time`
36          last_pressed_time = HAL_GetTick();
37
38          // check whether press was short or long
39          if (press_time <= SHORT_PRESS_MAX_MS) button_press_type = SHORT;
40          else button_press_type = LONG;
41
42          // check whether `button_press_type` is correct
43          if (code[ptr] == button_press_type) {
44              light_led(YELLOW_LED_PIN, 1, 200);
45              if (++ptr >= code_length) {
46                  light_led(GREEN_LED_PIN, 3, 200);
47                  ptr = 0;
48                  failed_attempts_cnt = 0;
49              }
50          } else {
51              light_led(RED_LED_PIN, 1, 200);
52              ptr = 0;
53              if (++failed_attempts_cnt == 3) {
54                  failed_attempts_cnt = 0;
55                  light_led(RED_LED_PIN, 5, 200);
56              }
57          }
58
59          start_time = 0;
60      }
```

6 Вывод

В результате выполнения лабораторной работы были получены базовые знания о принципах устройства стенда SDK-1.1M и программировании микроконтроллеров, а также изучены устройство интерфейсов ввода-вывода общего назначения (GPIO) в микроконтроллерах и приемы использования данных интерфейсов.