УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Проектирование вычислительных систем»

Лабораторная работа №4

Вариант 6

Студент

Белогаев Д. В.

Кузнецов М. А.

P34131

Преподаватель

Пинкевич В. Ю.

Санкт-Петербург, 2023 г.

Задание лабораторной работы

Разработать программу, которая использует интерфейс I2C для считывания нажатий
кнопок клавиатуры стенда SDK-1.1.

Подсистема опроса клавиатуры должна удовлетворять следующим требованиям:

- реализуется защита от дребезга
- нажатие кнопки фиксируется сразу после того, как было обнаружено, что кнопка нажата (с учетом защиты от дребезга), а не в момент отпускания кнопки; если необходимо, долгое нажатие может фиксироваться отдельно
- кнопка, которая удерживается дольше, чем один цикл опроса, не считается повторно нажатой до тех пор, пока не будет отпущена (нет переповторов)
- распознается и корректно обрабатывается множественное нажатие (при нажатии более чем одной кнопки считается, что ни одна кнопка не нажата, если это не противоречит требованиям к программе)
- всем кнопкам назначаются коды от 1 до 12 (порядок на усмотрение исполнителей).

Программа должна иметь два режима работы, переключение между которыми
производится по нажатию кнопки на боковой панели стенда:

- режим тестирования клавиатуры
- прикладной режим.

Уведомление о смене режима выводится в UART.
В режиме тестирования клавиатуры программа выводит в UART коды нажатых кнопок.
В прикладном режиме программа обрабатывает нажатия кнопок и выполняет действия в соответствии с вариантом задания.

Вариант задания

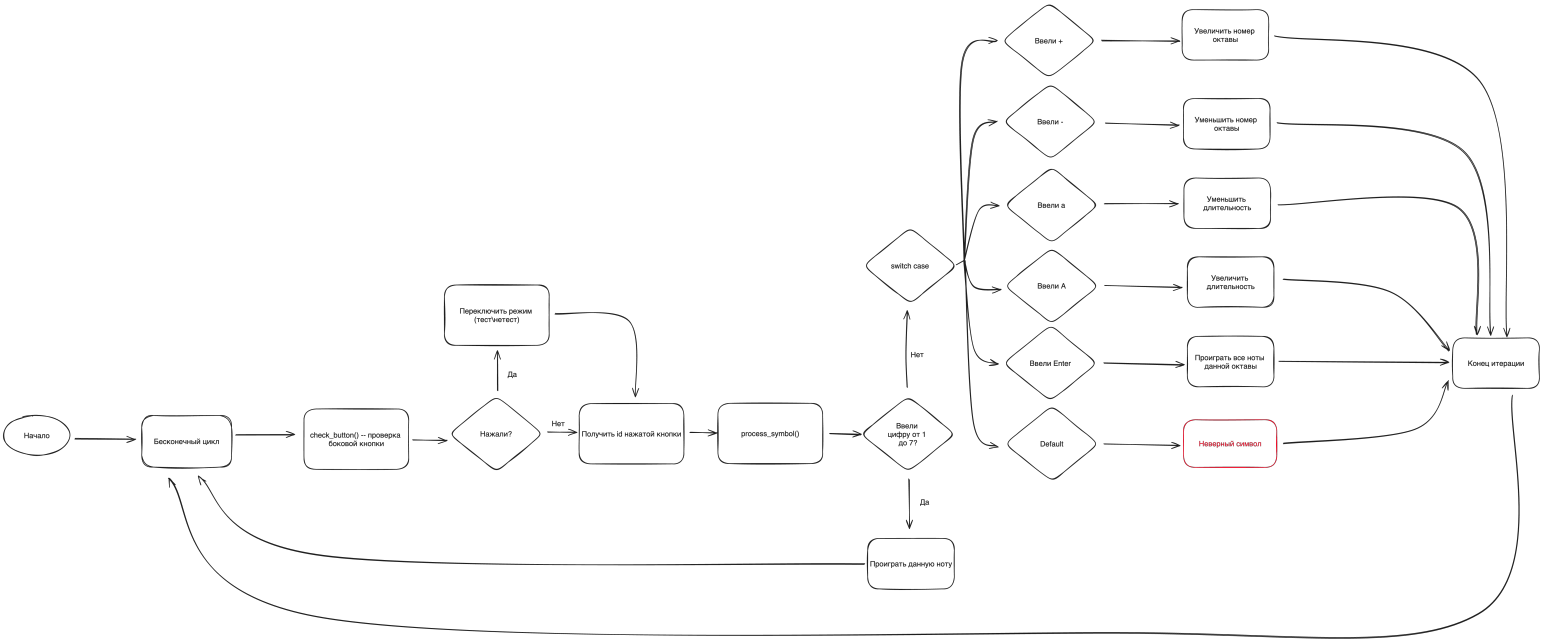
Реализовать «музыкальную клавиатуру» с помощью излучателя звука. Существует девять стандартных октав от субконтроктавы (первая по порядку) до пятой октавы (девятая по порядку) (более подробно об октавах см. в специализированных источниках). Частоты нот в соседних октавах отличаются ровно в два раза и растут с номером октавы. Частоты для первой октавы (пятая по порядку):

Нота	Частота, Гц
До	261,63
Ре	293,67
Ми	329,63
Фа	349,23
Соль	392,00
Ля	440,00
Си	493,88

Символ	Действие
«1» – «7»	Воспроизведение одной ноты (от «до» до «си») текущей октавы с текущей длительностью звучания. Начальные значения: первая октава (пятая по порядку), длительность 1 с.
«+»	Увеличение номера текущей октавы (максимальная – пятая).
«-»	Уменьшение номера текущей октавы (минимальная – субконтроктава).
«A»	Увеличение длительности воспроизведения ноты на 0,1 с (максимум – 5 с).
«a»	Уменьшение длительности воспроизведения ноты на 0,1 с (минимум – 0,1 с).
«Enter»	Последовательное воспроизведение всех нот текущей октавы с текущей длительностью без пауз.

По вводу каждого символа в UART должно выводиться сообщение:

- для символов «1» – «7», «Enter»: какая нота какой октавы и с какой длительностью проигрывается
- для символов настройки: новые значения номера октавы и длительности звучания ноты;
- для символов, не перечисленных в таблице выше: сообщение «неверный символ» и его код.



Исходный код

```
const uint16_t row_size = 3;

uint32_t keys[] = {

    '1', '2', '3',

    '4', '5', '6',
```

```
    '7', '+', '-',
    'A', 'a', '\r'
};

const uint32_t oct_size = 7;
uint32_t freqs[] = { 16350, 18350, 20610, 21820, 24500, 27500, 30870, 32700, 36950,
                     41210, 43650, 49000, 55000, 61740, 65410, 73910, 82410, 87310, 98000,
                     110000, 123480, 130820, 147830, 164810, 174620, 196000, 220000, 110000,
                     261630, 293670, 329630, 349230, 392000, 440000, 493880, 523260, 587340,
                     659260, 698460, 784000, 880000, 987760, 1046520, 1174680, 1318500,
                     1396900, 1568000, 1720000, 1975500, 2093000, 2349200, 2637000, 2739800,
                     3136000, 3440000, 3951000, 4186000, 4698400, 5274000, 5587000, 6271000, 7040000,
7902000 };
uint32_t note_index = 0;
uint32_t octave = 4;
uint32_t duration = 1000;
uint8_t is_all_playing = 0;
char* note_name[] = {"До", "Ре", "Ми", "Фа", "Соль", "Ля", "Си"};

uint8_t is_writing_now = 0;
char read_buffer[100];
char write_buffer[100];
char* cur_process_char = read_buffer;
char* cur_read_char = read_buffer;
char* transmit_from_pointer = write_buffer;
char* write_to_pointer = write_buffer;

uint32_t key_press_delay = 200;
uint32_t side_press_delay = 250;
uint32_t last_press_time = 0;
uint32_t last_side = 0;

uint8_t is_test = 0;
```

```
int32_t last_index = -1;



/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM6_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
char* concat(char *s1, char *s2) {
   char *result = malloc(strlen(s1) + strlen(s2) + 1);
   strcpy(result, s1);
   strcat(result, s2);
   return result;
}

void next(char **pointer, char *buffer) {
   if(*pointer >= buffer + 100){
      *pointer = buffer;
   }
   else {
      (*pointer)++;
   }
}
```

```
void write_char_to_buff(char c) {
   *write_to_pointer = c;
   next(&write_to_pointer, write_buffer);
}

void write(char* str) {
   char* str_with_newlines = concat("\r\n", str);
   int size = sizeof(char)*strlen(str_with_newlines);
   for(size_t i = 0; str_with_newlines[i] != '\0'; i++) {
      write_char_to_buff(str_with_newlines[i]);
   }
}

int is_number(char* str) {
   for (size_t i = 0; str[i] != '\0'; i++) {
     if (!isdigit(str[i])) return 0;
   }
   return 1;
}

void restart_timer() {
   TIM6->CNT = 0;
}

void mute() {
   TIM1->CCR1 = 0;
}

void unmute() {
   TIM1->CCR1 = (TIM1->ARR+1) / 2;
}
```

```
void set_frequency(uint32_t freq_millis) {
   TIM1->PSC = ((2 * HAL_RCC_GetPCLK2Freq()) / (2 * (TIM1->ARR) * (freq_millis / 1000))) - 1;
}

int get_frequency(uint32_t index) {
   return freqs[index + (octave * oct_size)];
}

void play(uint32_t index) {
   int freq = get_frequency(index);
   if (freq > 0) {
      set_frequency(freq);
      restart_timer();
      unmute();
   } else {
      if (is_all_playing) {
         mute();
         is_all_playing = 0;
      } else {
         char answer[100];
         sprintf(answer, "Нет ноты %s в октаве %d!", note_name[note_index], octave);
         write(answer);
      }
   }

}

void start_playing() {
   is_all_playing = 1;
   note_index = 0;
   char answer[100];
   sprintf(answer, "Нота: %s, октава: %d", note_name[note_index], octave+1);
   write(answer);
```

```
  play(note_index);
}


void duration_decrease(){
   if (duration > 100) {
      duration -= 100;
      TIM6->ARR = duration;
      restart_timer();
   }
}

void duration_increase() {
   if (duration < 5000) {
      duration += 100;
      TIM6->ARR = duration;
      restart_timer();
   }
}

void switch_mode() {
  if (is_test) {
    is_test = 0;
  } else is_test = 1;
}

void check_button() {
  uint32_t now = HAL_GetTick();
  if ((now - last_side) > side_press_delay) {
    if (HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_15) == 0) {
      last_side = now;
      switch_mode();
      if (is_test) {
```

```
        write("Режим теста ВКЛ");
      } else write("Режим теста ВЫКЛ");
    }
  }
}


void process_symbol(char current) {
  char answer[100];
  if (current >= '1' && current <= '7') {
    note_index = current - '1';
    sprintf(answer, "Note: %s, octave: %d", note_name[note_index], octave+1);
    write(answer);
    play(note_index);
  } else {
    switch (current) {
    case '+':
      octave++;
      if (octave > 8) octave = 8;
      sprintf(answer, "New octave is: %d", octave+1);
      write(answer);
      break;
    case '-':
      if (octave != 0) {
        octave--;
      }
      sprintf(answer, "New octave is: %d", octave+1);
      write(answer);
      break;
    case 'a':
      duration_decrease();
      sprintf(answer, "New duration is: %d millis", duration);
      write(answer);
```

```
      break;
    case 'A':
      duration_increase();
      sprintf(answer, "New duration is: %d millis", duration);
      write(answer);
      break;
    case '\r':
      start_playing();
      break;
    default:
      sprintf(answer, "Incorrect symbol %u", current);
      write(answer);
      break;
    }
  }
}


HAL_StatusTypeDef init_keyboard( void ) {
  HAL_StatusTypeDef ret = HAL_OK;
  uint8_t buf;

  buf = 0;
  ret = HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, 0x02, 1, &buf, 1, 100);
  if( ret != HAL_OK ) {
    return ret;
  }

  buf = 0;
  ret = HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, 0x01, 1, &buf, 1, 100);

  return ret;
}
```

```
int get_key_index() {
  int index = -1;
  uint8_t reg_buffer = ~0;
  for (int row = 0; row < 4; row++) {
    uint8_t buf = ~((uint8_t) (1 << row));

    int16_t n_key = 0x00;
    HAL_StatusTypeDef ret = HAL_OK;

    ret = init_kb();

    ret = HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, 0x03, 1, &buf, 1, 100);
    HAL_Delay(10);
    ret = HAL_I2C_Mem_Read(&hi2c1, KB_I2C_READ_ADDRESS, 0x00, 1, &buf, 1, 100);

    uint8_t mask = 0x7;
    n_key = (~(buf>>4)) & mask;

    switch (n_key) {
    case 0x1:
      if (index != -1) return -1;
      index = row*row_size;
      break;
    case 0x2:
      if (index != -1) return -1;
      index = (row*row_size) + 1;
      break;
    case 0x4:
      if (index != -1) return -1;
      index = (row*row_size) + 2;
      break;
    }
```

```
  }
  return index;
}

int32_t check_key() {
  const uint32_t t = HAL_GetTick();
  if (t - last_press_time < key_press_delay) return -1;
  int16_t ind = get_key_index();
  if (ind != last_index) {
    last_index = ind;
    if (ind != -1) {
      last_press_time = t;
    }
    return ind;
  }
  return -1;
}


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
	if (huart->Instance == huart6.Instance) {
		next(&cur_read_char, read_buffer);
		HAL_UART_Receive_IT(&huart6, (uint8_t*) cur_read_char, sizeof(char));
	}
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
	if (huart->Instance == huart6.Instance) {
		is_writing_now = 0;
		next(&transmit_from_pointer, write_buffer);
	}
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if(htim->Instance == TIM6) {
    if (TIM1->CCR1 > 0) {
      note_index++;
      if (note_index < 7 && is_all_playing) {
        play(note_index);
        char answer[100];
        sprintf(answer, "Нота: %s", note_name[note_index]);
        write(answer);
      } else {
        note_index = 0;
        is_all_playing = 0;
        mute();
      }
    }
  }

}
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

```
/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM1_Init();
MX_TIM6_Init();
MX_USART6_UART_Init();
MX_I2C1_Init();
/* USER CODE BEGIN 2 */

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_Base_Start_IT(&htim6);
TIM6->ARR = duration;
HAL_UART_Receive_IT(&huart6, (uint8_t *) cur_read_char, sizeof( char ));
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
        check_button();
        int ind = check_key();
        if (ind != -1) {
                if (is_test) {
                        char answer[100];
```

```
                    sprintf(answer, "%d", ind+1);
                    write(answer);
                }
                last_index = ind;
                char cur = keys[ind];
                process_symbol(cur);
            }
            if(is_writing_now == 0){
                if(transmit_from_pointer != write_to_pointer) {
                    is_writing_now = 1;
                    HAL_UART_Transmit_IT( &huart6, (uint8_t *) transmit_from_pointer, sizeof(
char ));
                }
            }
  }

  /* USER CODE END 3 */
}
```

Вывод

Во время выполнения лабораторной работы мы:

- изучили работу интерфейса I2C
- разработали программу, использующую его для считывания нажатий клавиатуры.