Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

«Таймеры»

по дисциплине «Проектирование вычислительных систем»

Вариант – 1

Выполнили:

Студенты группы P3432

Чмурова М.В.

Комягин Д.А.

Преподаватель:

Пинкевич Василий Юрьевич

Санкт-Петербург

2025

Задание

Разработать программу, которая использует таймеры для управления яркостью светодиодов и излучателем звука (по прерыванию или с использованием аппаратных каналов). Блокирующее ожидание (функция HAL_Delay()) в программе использоваться не должно.

Стенд должен поддерживать связь с компьютером по UART и выполнять указанные действия в качестве реакции на нажатие кнопок на клавиатуре компьютера. В данной лабораторной работе каждая нажатая кнопка (символ, отправленный с компьютера на стенд) обрабатываются отдельно, ожидание ввода полной строки не требуется.

Для работы с UART на стенде можно использован один из двух вариантов драйвера (по прерыванию и по опросу) на выбор исполнителя. Поддержка двух вариантов не требуется.

Частота синхросигнала процессорного ядра и сигнала ШИМ для управления яркостью светодиодов (если используется) должны соответствовать указанным в варианте задания.

Вариант 1:

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.

Стенд должен выполнять следующие действия при получении символов от компьютера:

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.

Частота процессорного ядра – 120 МГц.

Описание организации программы и структуры драйверов

Для установки необходимых значений PSC (Prescaler), ARR (Auto-loader) и CCR (регистр фиксации), необходимо оперировать следующими формулами:

$$f_{out} = \frac{f_{tim}}{PSC + 1} \cdot \frac{1}{ARR + 1}$$

$$f_{tick} = \frac{f_{tim}}{PSC + 1}$$

$$duty = \frac{CCR}{ARR + 1}$$

- f_{out} – частота переполнения счетчика или проще говоря, частота, которая необходима для звучания определенной ноты
- f_{tim} – частота, с которой таймер тактируется из шины (от RCC). По нашему варианту $f_{tim} = 120$ [МГц]
- f_{tick} – частота, с которой реально происходит «тик» после делителя (PSC). $f_{tick} = 1$ [МГц] для удобства в расчётах
- $duty$ – коэффициент заполнения/скважность – процент времени, в течение которого сигнал имеет высокий уровень. Для звукоизлучателя необходим $duty = 0.5 = 50\%$

Таким образом, получается функцию для установки необходимой для ноты частоты:

```
void tone_start(uint32_t freq_hz)
{
    const uint16_t PSC = (F_TIM / F_TICK) - 1;
    const uint16_t ARR = (F_TICK / freq_hz) - 1;
    const uint16_t CCR = (ARR + 1) / 2;

    __HAL_TIM_SET_PRESCALER(&htim1, PSC);
    __HAL_TIM_SET_AUTORELOAD(&htim1, ARR);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, CCR);

    htim1.Instance->EGR = TIM_EGR_UG;
}
```

TIM_EGR_UG – необходим, чтобы сгенерировать «событие обновления» - без установки флага таймер также применит новые значения,

но только после завершения текущего цикла, после переполнения. Поэтому используется этот флаг для моментального применения значения.

Также для управления звуков используются функции:

```
void tone_update(void)
{
    if (tone_active) {
        if ((int32_t)(HAL_GetTick() - tone_deadline_ms) >= 0) {
            tone_stop();
        }
    }
}

void tone_stop(void)
{
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    tone_active = 0;
}
```

Они необходимы для остановки звучания (tone_stop, устанавливает скважность в 0%) и для обеспечения звучания ноты в течение заданной длительности (tone_update).

Функция для изменения длительности, если она не достигла максимального или минимального допустимого значения:

```
void change_duration(char ch)
{
      if (ch == 'A')
      {
            if (duration_ms < 5000)
            {
                  duration_ms += 100;
                  char msg[32];
                  sprintf(msg, "New duration (ms): %lu", duration_ms);
                  uart_transmitln_string("");
                  uart_transmitln_string(msg);
            }
            else
            {
                  uart_transmitln_string("");
                  uart_transmitln_string("LIMIT: max duration reached");
            }
      }
      else if (ch == 'a')
      {
            if (duration_ms > 100)
            {
                  duration_ms -= 100;
                  char msg[32];
                  sprintf(msg, "New duration (ms): %lu", duration_ms);
                  uart_transmitln_string("");
                  uart_transmitln_string(msg);
            }
            else
            {
```

```
                uart_transmitln_string("");
                uart_transmitln_string("LIMIT: min duration reached");
            }
      }

}
```

Функции для реализации последовательного воспроизведения всех нот текущей октавы:

```
void harmony_cancel(void)
{
    if (harmony_active) {
        harmony_active = 0;
        tone_stop();
    }
}

void play_harmony_start()
{
      harmony_index = 0;
      harmony_active = 1;

      tone_start((int)(octava * NOTE_FREQS[harmony_index]));
      log_note_info(harmony_index);
      harmony_deadline_ms = HAL_GetTick() + duration_ms;
}

void play_harmony_update()
{
      if (!harmony_active) return;

      if ((int32_t)(HAL_GetTick() - harmony_deadline_ms) >= 0) {
            harmony_index += 1;
            if (harmony_index < (int)(sizeof(NOTE_FREQS) /
sizeof(NOTE_FREQS[0])))
            {
                  tone_start((int)(octava * NOTE_FREQS[harmony_index]));
                  log_note_info(harmony_index);
                  harmony_deadline_ms = HAL_GetTick() + duration_ms;
            }
            else
            {
                  harmony_active = 0;
                  tone_stop();
            }
      }
}
```

Здесь используется флаг *harmony_active* для «включения» режима последовательного воспроизведения, после чего, используя *harmony_index* происходит перебор/включение звука для всех нот из списка *NOTE_FREQS* с нужной октавы. В случае необходимости остановить уже начавшееся последовательное воспроизведение *harmony_cancel()* устанавливает флаг *harmony_active* в ноль.

Функция для реализации смен между октавами:

```
void change_octava(char ch)
{
    if (ch == '+')
    {
        if (octave_index >= OCT_MAX_INDEX)
        {
            uart_transmitln_string("");
            uart_transmitln_string("LIMIT: highest octave reached");
        } else
        {
            octave_index++;
            octava *= OCT_TWO;

            char msg[64];
            sprintf(msg, "Octave: %s, Duration (ms): %lu",
OCTAVE_NAMES[octave_index], duration_ms);
            uart_transmitln_string("");
            uart_transmitln_string(msg);
        }
    }
    else if (ch == '-')
    {
        if (octave_index <= OCT_MIN_INDEX) {
            uart_transmitln_string("");
            uart_transmitln_string("LIMIT: lowest octave reached");
        } else {
            octave_index--;
            octava *= OCT_HALF;

            char msg[64];
            sprintf(msg, "Octave: %s, Duration (ms): %lu",
OCTAVE_NAMES[octave_index], duration_ms);
            uart_transmitln_string("");
            uart_transmitln_string(msg);
        }
    }
}
```

Так как каждая соседняя октава отличается изменением частоты для каждой ноты в два раза, используется множитель *octava*, который умножает каждую ноту из первой октавы на необходимое значение частоты и, непосредственно, звучания.

Функция, логирования:

```
void log_note_info(int index)
{
    char msg[80];
    sprintf(msg, "Note: %s, Octave: %s, Duration (ms): %lu",
NOTE_NAMES[index], OCTAVE_NAMES[octave_index], duration_ms);
    uart_transmitln_string("");
    uart_transmitln_string(msg);
}
```

Для вывода в лог текущей ноты, октавы и установленной длительности звучания одной ноты.

А также самая главная функция: обрабатываются команды, пришедший на МК по UART6. Возможные команды выделены в отдельную структуру:

```
typedef enum {
    CMD_UNKNOWN = 0,
    CMD_NOTE,
    CMD_DURATION,
    CMD_OCTAVE,
    CMD_PLAY_HARMONY
} note_command_t;
```

Таким образом, что:

- CMD_NOTE – для воспроизведения одной ноты при нажатии клавиш «1» - «7»
- CMD_DURATION – для изменения длительности воспроизведения нот
- CMD_OCTAVE – для изменения текущей октавы
- CMD_PLAY_HARMONY – для воспроизведения последовательности всех нот текущей октавы.
- CMD_UNKNOWN – начальное состояние, когда команда еще не определена

Главная функция для управления всеми командами:

```
void uart_notes_handler(void)
{
    char c;
    while (uart_receive_char(&c))
    {
        uart_transmit_char(c);
        harmony_cancel();

        note_command_t command_type = CMD_UNKNOWN;
            int note_index = -1;
            if (c >= '1' && c <= '7') {
                  command_type = CMD_NOTE;
                  note_index = c - '1';
            }
            else if (c == 'A' || c == 'a') { command_type = CMD_DURATION; }
            else if (c == '+' || c == '-') { command_type = CMD_OCTAVE; }
            else if (c == '\n') { command_type = CMD_PLAY_HARMONY; }
            else if (c == '\r') { continue; }
            else {
                  if (isprint((unsigned char)c))
                  {
                        uart_transmitln_string("");
                        uart_transmitln_string("ERROR 1: Incorrect symbol");
                  }
```

```
                continue;
            }

            switch (command_type)
            {
            case CMD_NOTE:
                tone_start((int)(octava * NOTE_FREQS[note_index]));
                log_note_info(note_index);
                tone_active = 1;
                tone_deadline_ms = HAL_GetTick() + duration_ms;
                break;
            case CMD_DURATION: change_duration(c); break;
            case CMD_OCTAVE: change_octava(c); break;
            case CMD_PLAY_HARMONY: play_harmony_start(); break;
            default: break;
        }
    }

    tone_update();
    play_harmony_update();
}
```

Обрабатывает все имеющиеся команды и вызывает необходимые для них функции (или сообщение об ошибке)

Блок-схема прикладного алгоритма

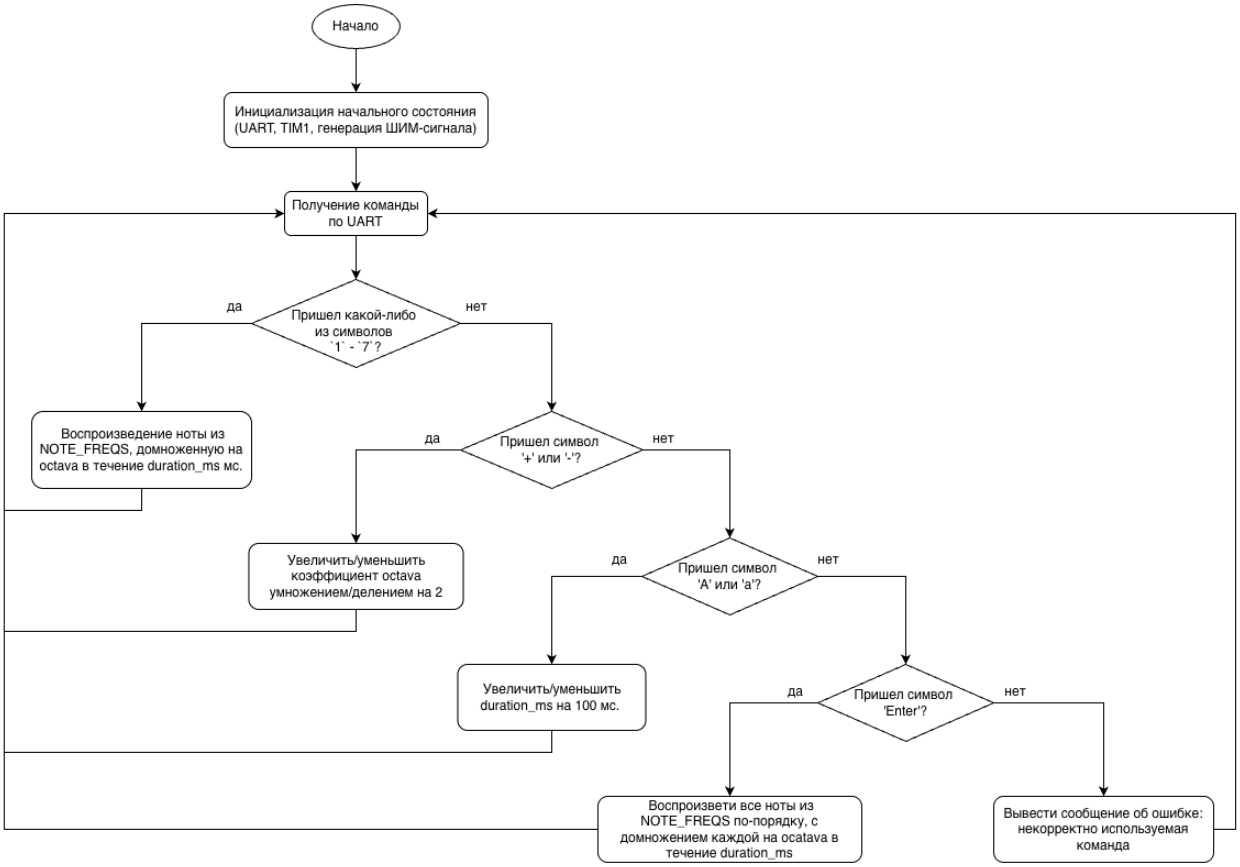

Рисунок 1. Блок-схема алгоритма

Исходный код

В последующем разделе приведён разбор только исходных файлов с расширением *.c*. Заголовочные файлы *.h* не рассматриваются, поскольку не содержат полезной информации, описания каких-либо структур или же они были рассмотрены выше.

notes_handler.c

```
#include <music/notes_handler.h>
#include <stdint.h>

static int octave_index = 4;
static float octava = 1.0f;
static const char* OCTAVE_NAMES[OCT_COUNT] = {
    "Subcontra",
    "Contra",
    "Great",
    "Small",
    "First",
    "Second",
    "Third",
    "Fourth",
    "Fifth"
};

static const char* NOTE_NAMES[7] = { "do", "re", "mi", "fa", "sol", "la",
"si" };
static const uint16_t NOTE_FREQS[7] = { 262, 294, 330, 349, 392, 440, 494 };

static volatile uint8_t tone_active = 0;
static volatile uint32_t tone_deadline_ms = 0;
static volatile uint32_t duration_ms = 1000;

static volatile uint8_t harmony_active = 0;
static int harmony_index = 0;
static uint32_t harmony_deadline_ms = 0;

void uart_notes_handler(void)
{
    char c;
    while (uart_receive_char(&c))
    {
        uart_transmit_char(c);
        harmony_cancel();

        note_command_t command_type = CMD_UNKNOWN;
            int note_index = -1;
            if (c >= '1' && c <= '7') {
                  command_type = CMD_NOTE;
                  note_index = c - '1';
            }
            else if (c == 'A' || c == 'a') { command_type = CMD_DURATION; }
            else if (c == '+' || c == '-') { command_type = CMD_OCTAVE; }
            else if (c == '\n') { command_type = CMD_PLAY_HARMONY; }
            else if (c == '\r') { continue; }
```

```
            else {
                  if (isprint((unsigned char)c))
                  {
                        uart_transmitln_string("");
                        uart_transmitln_string("ERROR 1: Incorrect symbol");
                  }
                  continue;
            }

            switch (command_type)
            {
            case CMD_NOTE:
                  tone_start((int)(octava * NOTE_FREQS[note_index]));
                  log_note_info(note_index);
                  tone_active = 1;
                  tone_deadline_ms = HAL_GetTick() + duration_ms;
                  break;
            case CMD_DURATION: change_duration(c); break;
            case CMD_OCTAVE: change_octava(c); break;
            case CMD_PLAY_HARMONY: play_harmony_start(); break;
            default: break;
        }
    }

    tone_update();
    play_harmony_update();
}

void log_note_info(int index)
{
    char msg[80];
    sprintf(msg, "Note: %s, Octave: %s, Duration (ms): %lu",
NOTE_NAMES[index], OCTAVE_NAMES[octave_index], duration_ms);
    uart_transmitln_string("");
    uart_transmitln_string(msg);
}

void change_octava(char ch)
{
    if (ch == '+')
    {
        if (octave_index >= OCT_MAX_INDEX)
        {
            uart_transmitln_string("");
            uart_transmitln_string("LIMIT: highest octave reached");
        } else
        {
            octave_index++;
            octava *= OCT_TWO;

            char msg[64];
            sprintf(msg, "Octave: %s, Duration (ms): %lu",
OCTAVE_NAMES[octave_index], duration_ms);
            uart_transmitln_string("");
            uart_transmitln_string(msg);
        }
    }
    else if (ch == '-')
    {
        if (octave_index <= OCT_MIN_INDEX) {
            uart_transmitln_string("");
```

```
            uart_transmitln_string("LIMIT: lowest octave reached");
        } else {
            octave_index--;
            octava *= OCT_HALF;

            char msg[64];
            sprintf(msg, "Octave: %s, Duration (ms): %lu",
OCTAVE_NAMES[octave_index], duration_ms);
            uart_transmitln_string("");
            uart_transmitln_string(msg);
        }
    }
}

void harmony_cancel(void)
{
    if (harmony_active) {
        harmony_active = 0;
        tone_stop();
    }
}

void play_harmony_start()
{
      harmony_index = 0;
      harmony_active = 1;

      tone_start((int)(octava * NOTE_FREQS[harmony_index]));
      log_note_info(harmony_index);
      harmony_deadline_ms = HAL_GetTick() + duration_ms;
}

void play_harmony_update()
{
      if (!harmony_active) return;

      if ((int32_t)(HAL_GetTick() - harmony_deadline_ms) >= 0) {
            harmony_index += 1;
            if (harmony_index < (int)(sizeof(NOTE_FREQS) /
sizeof(NOTE_FREQS[0])))
            {
                  tone_start((int)(octava * NOTE_FREQS[harmony_index]));
                  log_note_info(harmony_index);
                  harmony_deadline_ms = HAL_GetTick() + duration_ms;
            }
            else
            {
                  harmony_active = 0;
                  tone_stop();
            }
      }
}

void change_duration(char ch)
{
      if (ch == 'A')
      {
            if (duration_ms < 5000)
            {
                  duration_ms += 100;
                  char msg[32];
```

```
                  sprintf(msg, "New duration (ms): %lu", duration_ms);
                  uart_transmitln_string("");
                  uart_transmitln_string(msg);
            }
            else
            {
                  uart_transmitln_string("");
                  uart_transmitln_string("LIMIT: max duration reached");
            }
      }
      else if (ch == 'a')
      {
            if (duration_ms > 100)
            {
                  duration_ms -= 100;
                  char msg[32];
                  sprintf(msg, "New duration (ms): %lu", duration_ms);
                  uart_transmitln_string("");
                  uart_transmitln_string(msg);
            }
            else
            {
                  uart_transmitln_string("");
                  uart_transmitln_string("LIMIT: min duration reached");
            }
      }

}

void tone_update(void)
{
    if (tone_active) {
        if ((int32_t)(HAL_GetTick() - tone_deadline_ms) >= 0) {
            tone_stop();
        }
    }
}
void tone_stop(void)
{
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    tone_active = 0;
}

void tone_start(uint32_t freq_hz)
{
    const uint16_t PSC = (F_TIM / F_TICK) - 1;
    const uint16_t ARR = (F_TICK / freq_hz) - 1;
    const uint16_t CCR = (ARR + 1) / 2;

    __HAL_TIM_SET_PRESCALER(&htim1, PSC);
    __HAL_TIM_SET_AUTORELOAD(&htim1, ARR);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, CCR);

    htim1.Instance->EGR = TIM_EGR_UG;
}
```

С полной версией программы можно ознакомиться, перейдя на GitHub-репозиторий, расположенный по ссылке:

https://github.com/kkettch/design-of-computing-systems-semester-7

Вывод

В ходе выполнения работы были успешно разработан и реализован звукоизлучатель «пианино», который позволяет воспроизводить ноты с переключением всех имеющихся на пианино октав. Для их воспроизведения используется интерфейс UART, который передает поступающие команды от ПК на МК. А также передает сообщения с МК на ПК для вывода логов.

Кроме того, реализована возможность изменения длительности звучания ноты в диапазоне от 0.1 до 5 секунд.