TERMEX

ТЕРМОМЕТРЫ ЛАБОРАТОРНЫЕ ЭЛЕКТРОННЫЕ LTA

Руководство программиста

СОДЕРЖАНИЕ

1 Введение 3
1.1 Назначение 3
1.2 Pawn 3
1.3 Ресурсы памяти 3
1.4 Запуск компилятора 4
1.5 Загрузка скрипта в термометр 4
1.6 Пример 5
2 Стандартные модули 7
2.1 Core 7
2.2 String 7
2.3 Float 7
2.4 Console 8
2.5 Lta 11
3 Законченный сложный пример 16
3.1 Дополнительный контроль биологического инкубатора 16

Настоящее описание распространяется на термометры лабораторные электронные LTA (далее по тексту — термометры) и содержит сведения, необходимые для написания, компиляции, загрузки и запуска скриптов в термометре.

1 ВВЕДЕНИЕ

1.1 Назначение

1.1.1 В термометр LTA встроена виртуальная машина, исполняющая пользовательский код — скрипт. Скрипт позволяет расширять функциональность термометра.

1.1.2 В общем виде процесс создания и выполнения скрипта показан на рисунке 1 и состоит из следующих этапов:

1 - написание текста программы (файл с расширением `*.p`);
2 - компиляция исходного кода в байт-код (файл с расширением `*.amx`);
3 - загрузка байт-кода в термометр;
4 - выполнение байт-кода виртуальной машиной.

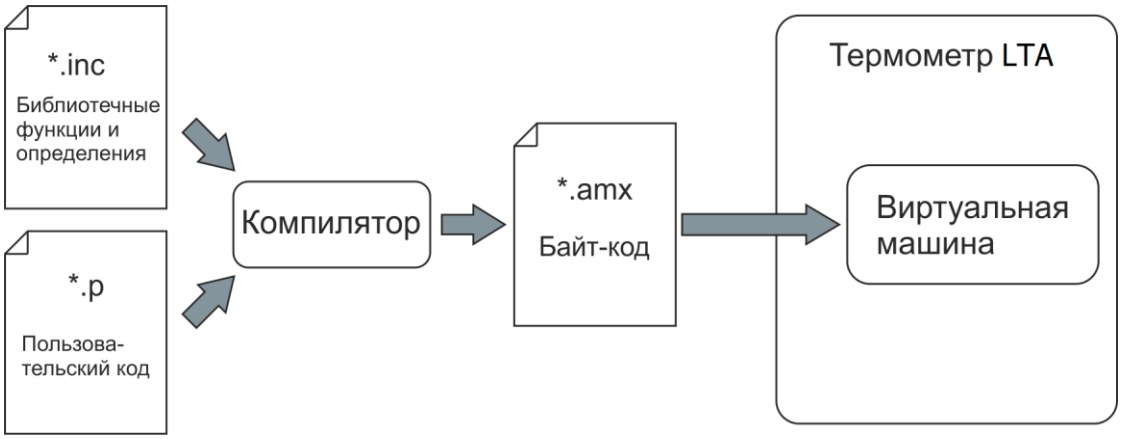


Рисунок 1 — Процесс создания и выполнения скрипта

1.2 Pawn

1.2.1 Для написания текста программ используется язык `pawn`.

1.2.2 `Pawn` — это бестиповой процедурно-ориентированный встраиваемый скриптовый язык программирования с С-подобным синтаксисом.

1.2.3 Подробное описание языка `pawn` выходит за рамки данного документа. Более подробно изучить язык можно на его домашней интернет-странице, где доступна вся документация по нему: http://www.compuphase.com/pawn/pawn.htm

1.3 Ресурсы памяти

1.3.1 Виртуальная машина предоставляет следующие ресурсы памяти для скрипта:

- 32512 байт энергонезависимой памяти для хранения байт-кода скрипта;
- 16384 байт ОЗУ для переменных и стека скрипта.

1.4 Запуск компилятора

1.4.1 Компилятор запускается следующей командой:

```
pawncc [options] <filename ...>
```

где `options` — список опций компилятора, `filename…` — список файлов `*.p` для компиляции.

1.4.2 Для правильной работы виртуальной машины в списке опций обязательно должна быть опция `-d1`, которая обеспечивает генерирование компилятором необходимых отладочных инструкций в байт-коде. Если скрипт откомпилирован с опцией `-d0`, то виртуальная машина прекратит выполнение скрипта с кодом ошибки `-1`. Опции `-d2` или `-d3` приемлемы, но бесполезно увеличивают размер байт-кода.

1.4.3 Для получения минимального размера результирующего байт-кода следует использовать максимальный уровень оптимизации, задаваемый опцией `-O3`.

1.4.4 Чтобы компилятор мог контролировать расход стека, следует указать максимальный размер стека опцией `-S<num>`, где `num` — размер стека в байтах. В случае превышения программой указанного размера, компилятор выдаст предупреждение. Рекомендуемое значение `-S1024`.

1.4.5 С учетом изложенного, компилятор должен запускаться следующей командой:

```
pawncc -d1 -O3 -S1024 <filename …>
```

1.4.6 Чтобы не вводить каждый раз одни и те же опции, они записаны в файле `pawn.cfg`, который автоматически загружается компилятором перед началом компиляции. Таким образом, при наличии файла `pawn.cfg`, запуск компилятора выполняется следующей командой:

```
pawncc <filename …>
```

1.4.7 По желанию, можно указать опцию `-D<path_to_p_files>`, где `path_to_p_files` — абсолютный или относительный путь к каталогу, содержащему файлы исходного кода. В этом случае, при указании списка файлов, не потребуется указывать путь к каждому файлу.

1.5 Загрузка скрипта в термометр

Для загрузки откомпилированного байт-кода скрипта в термометр может быть использована программа с графическим интерфейсом `LtaGraph` или консольная программа `ltacons`. Подробнее см. документы: «Термометры лабораторные электронные LTA. Программа LtaGraph. Руководство пользователя» и «Термометры лабораторные электронные LTA. Программа ltacons. Руководство пользователя».

1.6 Пример

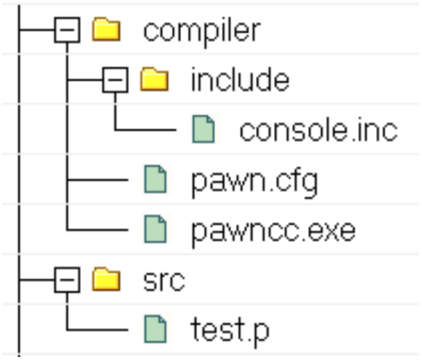
В этом разделе демонстрируются основные приемы работы. Будет написан, откомпилирован, загружен в термометр и выполнен небольшой скрипт.

1.6.1 Создайте в любом текстовом редакторе новый документ с именем `test.p` со следующим содержимым:

```
#include <console.inc>
main()
{
    new buf[81];
    for (;;)
    {
        getstring(buf);
        putstring(buf);
        flushoutput();
    }
}
```

Скрипт ожидает прихода любых данных по USB и отсылает их обратно.

1.6.2 Пусть для определенности имеется следующая структура файлов и папок:



1.6.3 Зайдите в каталог `compiler` и запустите компиляцию командой `pawncc ../src/test.p`

Т.к. есть файл `pawn.cfg` с необходимыми опциями компилятора (см. 0), то в командной строке их указывать не надо.

После компиляции, рядом с файлом `test.p` появится файл `test.amx` — байт-код скрипта.

1.6.4 Подключите термометр к компьютеру, включите его и запустите программу "LTA Utility". Перейдите на вкладку "Скрипт" (см. рисунок 2), нажмите кнопку 15 "Открыть" и укажите файл `test.amx`. Файл будет загружен во внутренний буфер программы. Нажмите кнопку 8 "Записать" и подтвердите действие. Код скрипта будет записан в память термометра.

1.6.5 Нажмите кнопку 5 "Запустить", при этом поле 2 должно принять значение "Да".

1.6.6 В поле 11 введите произвольные данные и нажмите кнопку 13 "Отправить". Данные будут отправлены скрипту и он вернет их обратно, что будет отображено в поле 12.

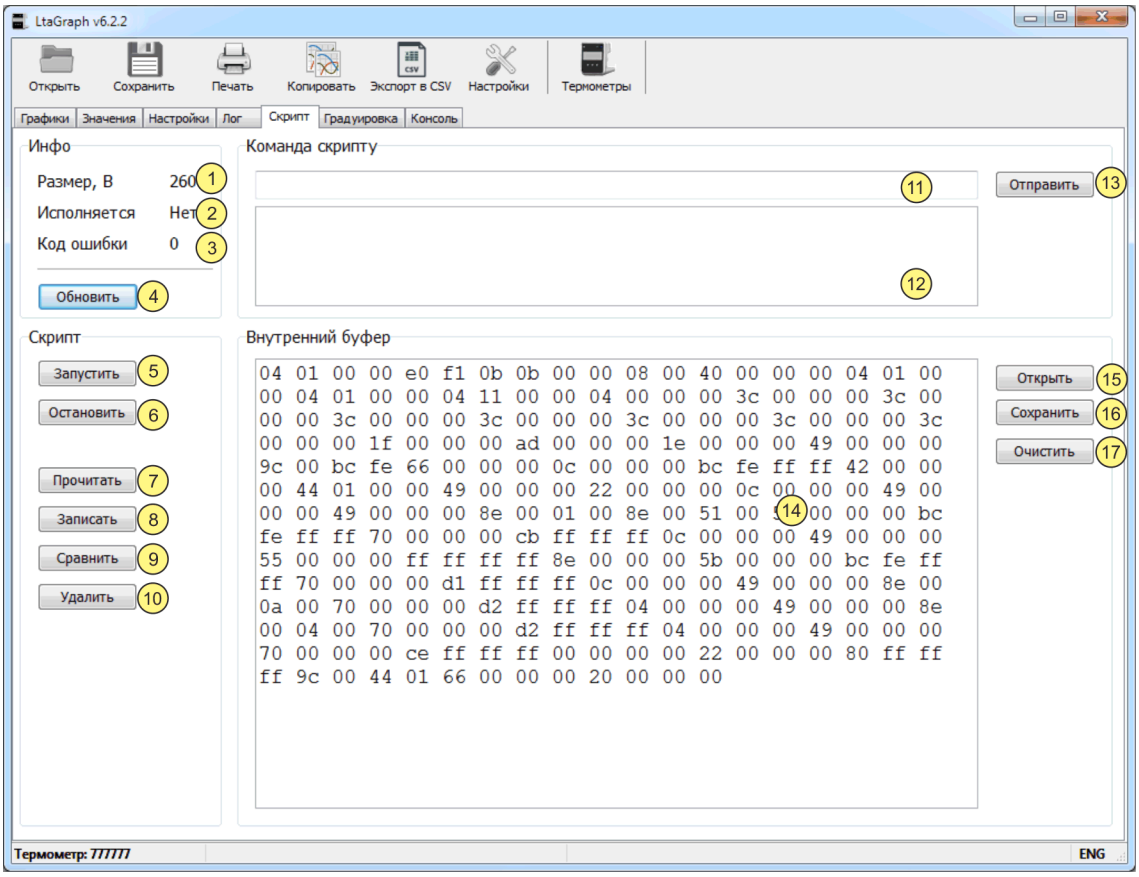


Рисунок 2 — Управление скриптом из программы LtaGraph

2 СТАНДАРТНЫЕ МОДУЛИ

С компилятором поставляется набор модулей (файлы `*.inc` каталога `include`), обеспечивающих решение типовых задач.

2.1 Core

2.1.1 В модуле `core` находятся функции, предназначенные для поддержки самого языка `pawn`. Например, здесь есть функции, обеспечивающие поддержку функций с переменным числом аргументов.

2.1.2 Описание функций содержится в документе "Pawn Language Guide", который доступен для скачивания с интернет-страницы `pawn`.

2.1.3 Не все функции, описанные в указанном документе, доступны для использования в коде скрипта. Список доступных функций содержится в файле `core.inc`.

2.2 String

2.2.1 В модуле `string` находятся функции, предназначенные для манипулирования строками.

2.2.2 Описание функций содержится в документе "String manipulation library", который доступен для скачивания с интернет-страницы `pawn`.

2.2.3 Не все функции, описанные в указанном документе, доступны для использования в коде скрипта. Список доступных функций содержится в файле `string.inc`.

2.3 Float

2.3.1 В модуле `float` находятся функции для манипулирования вещественными числами. Наличие такого модуля определено тем, что язык pawn не имеет встроенного типа для таких чисел.

2.3.2 Описание функций содержится в документе "Floating point support library", который доступен для скачивания с интернет-страницы `pawn`.

2.3.3 Дополнительно к функциям, описанным в указанном документе, в модуле содержатся функции:

```
floatstr(dest[], Float:value, bool:pack=true, maxlength=sizeof dest)
```

Преобразует вещественное число в строку.

`dest` строка для вывода результата;
`value` значение для преобразования;
`pack` определяет будет ли результат упакованной строкой или нет;
`maxlength` определяет доступный для вывода размер строки `dest`.

Для чисел с абсолютным значением меньше 1000, формт вывода равен `"%.3f"`. Для остальных чисел: `"%.5e"`.

```
bool:floatisnan(Float:value)
```

Возвращает `true`, если значение не является числом (`NaN`).

`value` проверяемое значение.

```
bool:floatisinf(Float:value)
```

Возвращает true, если значение равно бесконечности (INF).

`value` проверяемое значение

2.4 Console

2.4.1 В модуле `console` находятся функции для ввода/вывода данных в/из потока, связанного либо с USB, либо с Bluetooth.

2.4.2 Виртуальная машина имеет входной буфер размером 256 байт для данных, поступающих как от USB, так и от Bluetooth. Скрипт выбирает данные из входного буфера, используя одну из функций `getxxx()`. Для скрипта нет способа определить, откуда пришли данные — из USB или Bluetooth. Если при вызове `getxxx()`, во входном буфере нет данных, то виртуальная машина блокирует выполнение скрипта до прихода данных.

2.4.3 Вывод данных осуществляется одной из функций `putxxx()` в выходной поток, связанный либо с USB (действие по умолчанию), либо с Bluetooth.

`setoutstream(outstream:s)`

Устанавливает в какой поток будут выводиться данные.

`s` поток для вывода. Должен принимать одно из следующих значений:
- `usbstream` выводимые данные будут отправляться в USB;
- `btstream` выводимые данные будут отправляться в Bluetooth.

`flushoutput()`

Выводит в устройство все данные из выходного буфера и очищает буфер.

`flushinput()`

Очищает входной буфер.

Функции `putxxx()` используют параметры `width` и `precision`.

Параметр `width` задает ширину поля вывода. Если количество выводимых символов меньше значения `width`, то данные дополняются пробелами слева. Значение по умолчанию ноль, т.е. данные выводятся без дополнения пробелами. Если значение отрицательное, то оно переводится в положительное и устанавливается выравнивание влево, т.е. данные дополняются пробелами справа.

Параметр `precision` задает точность вывода. Для целых чисел — минимальное количество выводимых цифр. Для вещественных чисел — количество цифр после десятичной точки. Для строк — количество выводимых символов строки. Если значение параметра отрицательно, то точность вывода устанавливается в значение по умолчанию, равное -1. Это означает, что для целых десятичных чисел выводится как минимум одна цифра; для целых шестнадцатеричных чисел — две цифры; для вещественных чисел — две цифры после десятичной точки; для строки — все символы.

```
putchar(c)
```

Выводит символ `c`.

```
putstring(string[], width=0, precision=-1)
```

Выводит указанную строку.

`string` строка для вывода;
`width`, `precision` см. выше

Из-за внутренних ограничений, выводятся только первые 80 символов из `string`. Оставшиеся символы игнорируются.

```
putvalue(value, width=0, precision=-1, iformat:format=fdec)
```

Выводит указанное целочисленное значение.

`value` значение для вывода;
`width, precision` см. выше;
`format` формат вывода целых чисел. Может принимать одно из следующих значений:
- `fdec` вывод в десятичном виде;
- `fhex` вывод в шестнадцатеричном виде.

```
putfloat(Float:value, width=0, precision=-1, fformat:format=ffixed)
```

Выводит указанное вещественное число.

`value` значение для вывода;
`width, precision` см. выше;
`format` формат вывода вещественных чисел. Может принимать одно из следующих значений:
- `ffixed` вывод с фиксированной точкой;
- `fscientific` вывод в экспоненциальном (научном) виде.

Если при вызове одной из функций `getxxx()`, во входном буфере нет данных, то виртуальная машина блокирует выполнение скрипта до прихода данных.

`getchar()`

Возвращает символ из входного потока.

`getstring(string[], maxlength=sizeof string, bool:pack=true)`

Извлекает символы из потока и сохраняет их в `string` до тех пор, пока не встретится символ разделитель, либо не будет извлечено `maxlength` символов. Символ разделитель извлекается из потока, но в `string` не записывается. Символом разделителем считается один из символов: `"\r\n"`

`string` буфер, куда будет помещена входная строка;
`maxlength` максимальное количество символов для чтения из потока;
`pack` признак является ли `string` упакованной строкой.

Возвращает количество прочитанных в `string` символов.

Из-за внутренних ограничений, в строку максимально может быть прочитано только 80 символов из потока.

`getvalue(iformat:base=fdec)`

Извлекает из потока символы, конвертирует их в целое число со знаком и возвращает его. Начальные пробельные символы (`" \r\n\t\v\f"`) пропускаются. Если извлеченные символы не могут быть интерпретированы как целочисленные значения, то возвращает ноль.

`base` основание системы счисления. Может принимать одно из следующих значений:
- `fdec` читаемое значение рассматривается как десятичное;
- `fhex` читаемое значение рассматривается как шестнадцатеричное.

`Float:getfloat()`

Извлекает из потока символы, конвертирует их в вещественное число и возвращает его. Начальные пробельные символы (`" \r\n\t\v\f"`) пропускаются. Если извлеченные символы не могут быть интерпретированы как вещественное число, то возвращает ноль.

2.5 Lta

2.5.1 В модуле `lta` находятся функции, дающие доступ к самому термометру.

`version()`

Возвращает программно-аппаратную версию устройства, которая состоит из четырех чисел: `XX.YY.ZZ-HH`, где

`XX` — число, соответствующее функциональности термометра. Добавление тех или иных новых функций в термометр увеличивает это число;

`YY` — число, соответствующее улучшениям в ПО термометра, которые не влияют на его функциональность;

`ZZ` — число, соответствующее исправленным ошибкам;

`HH` — версия платы, на которой способно работать ПО термометра.

Версия упакована в 32-битное шестнадцатеричное число следующего вида: `0xXXYYZZHH`

`serial_number(string[], maxlength=sizeof string, bool:pack=true)`

Копирует серийный номер термометра в указанную строку. Возвращает количество скопированных символов.

`string` строковый буфер, куда копируется серийный номер;
`maxlength` размер `string`;
`pack` признак является ли `string` упакованной строкой или нет.

`bool:has_2()`

Возвращает `true`, если термометр имеет второй измерительный канал.

`bool:has_bluetooth()`

Возвращает `true`, если термометр имеет модуль Bluetooth.

`bool:has_dio()`

Возвращает `true`, если термометр имеет дискретные входы/выходы.

`bool:is_usb_configured()`

Возвращает `true`, если термометр подключен к USB порту и находится в сконфигурированном состоянии.

`bool:has_external_power()`

Возвращает `true`, если к термометру подключено внешнее (через разъем microUSB) питание.

`sample_period()`

Возвращает период измерения.

`shutdown_period()`

Возвращает период автовыключения.

`log_period()`

Возвращает период логирования.

`bool:is_logging()`

Возвращает `true`, если логирование результатов измерения включено.

`enable_logging(bool:v)`

Включает (`v=true`) или выключает (`v=false`) логирование результатов измерения.

`bool:is_bluetooth_enabled()`

Возвращает `true`, если модуль Bluetooth в термометре включен.

`enable_bluetooth(bool:v)`

Включает (`v=true`) или выключает (`v=false`) модуль Bluetooth в термометре.

`bkpmem_count()`

Возвращает количество 32-битных ячеек EEPROM, доступных для использования в скрипте. Как правило, это значение составляет 14 ячеек.

`read_bkpmem(idx)`

Возвращает значение из ячейки EEPROM с индексом `idx`. Значение `idx` должно быть в диапазоне `[0, bkpmem_count - 1]`. Возвращает ноль, если значение `idx` вне указанного диапазона.

`write_bkpmem(idx, v)`

Записывает значение `v` в ячейку EEPROM с индексом `idx`. Значение `idx` должно быть в диапазоне `[0, bkpmem_count - 1]`. Запись не выполняется, если значение `idx` вне указанного диапазона.

`Float:get_result(n, Result:r=res_temp)`

Возвращает результат измерения в виде температуры [°C] или сопротивления [Ом].

`n =[0, 1]` канал измерения;
`r` тип результата. Должен принимать одно из следующих значений:
- `res_temp` возвращается температура;
- `res_resist` возвращается сопротивление;

`Float:get_tempstats(n, Stats:s)`

Возвращает некоторую "статистику" по температуре [°C].

`n =[0, 1]` канал измерения;
`s` тип статистики. Должен принимать одно из следующих значений:
- `st_dt` разность температур между каналами.
  В этом случае параметр `n` игнорируется;
- `st_min` минимальное значение за все время измерений;
- `st_max` максимальное значение за все время измерений;
- `st_avg` среднее значение за все время измерений;
- `st_slope` скорость изменения температуры [`°C/sample_period`].

`get_tempcoef(n, Float:coef[4])`

Возвращает значения коэффициентов функции, используемой для расчета температуры.

`n =[0, 1]` канал измерения;
`coef` массив, куда будут скопированы значения коэффициентов, размером 4 элемента;

`get_filterparams(n, &size, &Float:level)`

Возвращает параметры фильтра.

`n =[0, 1]` канал измерения;
`size` ссылка на переменную, куда будет сохранена глубина фильтра;
`level` ссылка на переменную, куда будет сохранен порог фильтра.

Функции ниже `xxx_out_pin(Pin:p...)` манипулируют дискретным выходом `p`, который должен принимать одно из следующих значений:

- `pin1` дискретный выход 1;
- `pin2` дискретный выход 2.

`init_out_pin(Pin:p)`

Инициализирует для использования дискретный выход `p`.

`deinit_out_pin(Pin:p)`

Переводит дискретный выход `p` в неактивное низкопотребляющее состояние.

`write_out_pin(Pin:p, v)`

Устанавливает дискретный выход `p` в желаемое состояние.

`v = [0, 1]` значение для записи.

`toggle_out_pin(Pin:p)`

Переключает состояние дискретного выхода `p` на противоположное.

`init_in_pin(Pupd:pull)`

Инициализирует дискретный вход для использования.

`pull` подтяжка входа. Должен принимать одно из следующих значений:
- `no_pull` без подтяжки;
- `pull_up` подтяжка к питанию;
- `pull_down` подтяжка к земле.

`deinit_in_pin()`

Переводит дискретный вход в неактивное низкопотребляющее состояние.

`read_in_pin()`

Возвращает состояние `[0, 1]` дискретного входа.

`in_pin_interrupt_cfg(Trigger:edge)`

Устанавливает, по какому фронту сигнала на дискретном входе будет сгенерировано прерывание.

`edge` фронт сигнала. Должен принимать одно из следующих значений:
- `int_rising` по переднему фронту;
- `int_falling` по заднему фронту;
- `int_rising_falling` по обоим фронтам.

`in_pin_interrupt_enable(bool:v)`

Разрешает (`v=true`) или запрещает (`v=false`) генерирование прерываний на дискретном входе.

`set_timer(ticks, bool:singleshot=true)`

Запускает таймер на указанное количество тиков.

При срабатывании таймера формируется событие `ev_timer` (см. далее).

`ticks` количество тиков (период одного тика равен 100 мс). Если значение равно 0, то таймер останавливается;
`singleshort` определяет периодичность срабатывания таймера. Если равен `true`, то таймер срабатывает однократно; если равен `false`, то таймер срабатывает периодически до останова.

`pause(ticks)`

Останавливает работу скрипта на указанное количество тиков. Период одного тика равен 100 мс.

`cpucycles()`

Возвращает количество тактов процессора, прошедших с момента включения термометра.

При работе термометра от элементов питания, один такт равен примерно 1 микросекунде. При работе термометра от внешнего источника (например, подключен USB), один такт равен примерно 0.08 микросекунд.

Аппаратура термометра может генерировать для нужд скрипта несколько событий. Событие можно либо подождать (приостановив выполнение скрипта), либо поместить в очередь, а в дальнейшем, вынуть из очереди и обработать.

Функции ниже предоставляют интерфейс для работы с событиями.

```
const Event:
{
    ev_timer = 0,
    ev_interrupt,
    ev_result,
    ev_data
}
```

Набор событий, генерируемых аппаратно термометром.

`ev_timer` срабатывание таймера;
`ev_interrupt` прерывание на дискретном входе;
`ev_result` обновился результат измерения;
`ev_data` во входном буфере есть данные (см. 2.4).

```
waitfor(Event: e)
```

Функция приостанавливает выполнение скрипта до тех пор, пока не наступит указанное событие `e`.

```
enable_event_enqueue(Event:e, bool:v)
```

Разрешает (`v=true`) или запрещает (`v=false`) помещение события `e` в очередь событий при его возникновении.

```
Event:queue_pop()
```

Вынимает из очереди событие и возвращает его. Если очередь пуста, то выполнение скрипта приостанавливается до появления события в очереди.

3 ЗАКОНЧЕННЫЙ СЛОЖНЫЙ ПРИМЕР

3.1 Дополнительный контроль биологического инкубатора

3.1.1 Биологический инкубатор — это суховоздушный термостат, предназначенный для поддержания внутри камеры температуры, подходящей для выращивания микроорганизмов. Важно быть уверенным, что в процессе работы, температура не выходила за определенные границы.

3.1.2 Для дополнительного контроля температуры инкубатора может быть использован термометр LTA совместно с внешней схемой . Вот краткое техническое задание.

1 - Термометр должен измерять температуру внутри инкубатора. Если температура находится внутри заданных границ, то должен гореть зеленый светодиод. Если температура вышла за границы, должен мигать красный светодиод.

2 - Задание уставки инкубатора и допустимых границ температуры должно осуществляться через USB или Bluetooth и сохраняться в энергонезависимой памяти.

3.1.3 На рисунке 3 показана электрическая схема, состоящая из двух светодиодов, подключенных к дискретным выходам термометра.

3.1.4 Ниже приведен исходный код скрипта, который необходимо откомпилировать и загрузить в термометр.

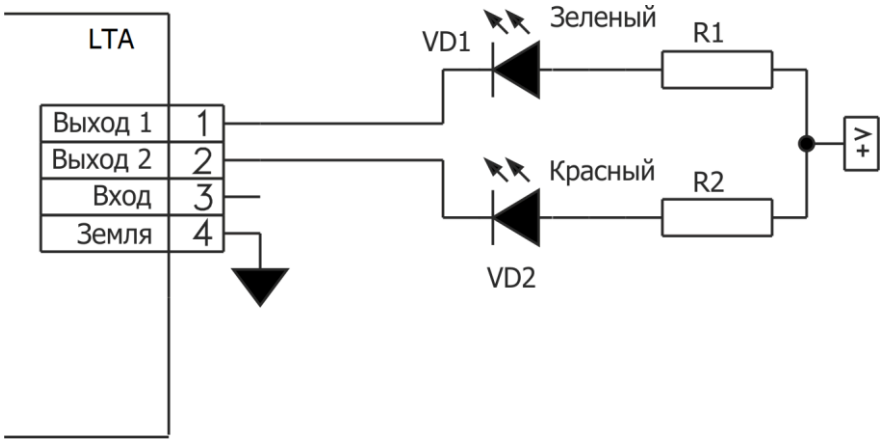


Рисунок 3

```
//-----------------------------------------------------------------------------
// Скрипт для контроля температуры в инкубаторе (термостате).
//
// Когда температура в допустимых границах, горит зеленый светодиод.
// Когда вышла за допустимые границы, мигает красный светодиод.
//
// Уставку инкубатора и допустимые границы можно задать через USB
// соответствующей командой. Значения сохраняются в энергонезависимой памяти.
//-----------------------------------------------------------------------------

#include <console.inc>
#include <lta.inc>
#include <float.inc>

new Float:setpoint; // Уставка в инкубаторе.
new Float:delta;    // Допустимый "коридор" вокуг уставки.
new bool:blink;     // Флаг для мигания красным светодиодом.

main()
{
    // Подготавливаем цифровые выходы.
    init_out_pin(pin1);
    init_out_pin(pin2);

    // Из ячеек eeprom считываем значения уставки и дельты
    // и проверяем их на корректность.
    setpoint = float(read_bkpmem(0)) / 10.0; // Float значение хранится в
    delta = float(read_bkpmem(1)) / 10.0;    // eeprom в виде целого:
                                             //     round(f*10)
    check();

    // Необходимые события будут помещаться в очередь.
    enable_event_enqueue(ev_timer, true);
    enable_event_enqueue(ev_result, true);
    enable_event_enqueue(ev_data, true);

    // Включаем периодический таймер
    // для организации мигания красного светодиода.
    set_timer(200/timer_tick, false);

    new Event:ev;

    // Основной цикл.
    for (;;)
    {
        // Ждем очередное событие и обрабатываем его.
        ev = queue_pop();
        switch (ev)
        {
        case ev_result: process_result();
        case ev_timer:  process_timer();
        case ev_data:   process_data();
        }
    }
}
```

```
//--------------------------------------------------------------------------------
// На каждое измерение температуры термометром, выполняется проверка,
// что она находится в допустимых границах.
//
process_result()
{
    // Получаем текущее значение температуры и вычисляем допустимый диапазон
    // для нее.
    new Float:v = get_result(0);
    new Float:tmin = setpoint - delta;
    new Float:tmax = setpoint + delta;

    if ( v > tmin && v < tmax )
    {
        // Температура в допустимых границах. Зажигаем зеленый светодиод,
        // красный гасим.
        write_out_pin(pin1, 1); // Green
        write_out_pin(pin2, 0); // Red
        blink = false;
    }
    else
    {
        // Температура вышла за допустимые границы. Зажигаем красный светодиод,
        // зеленый гасим.
        write_out_pin(pin1, 0); // Green
        write_out_pin(pin2, 1); // Red
        blink = true;           // Если blink = true, то по таймеру будет
                                // выполнятся мигание красного светодиода.
    }
}


//--------------------------------------------------------------------------------
// По таймеру осуществляется мигание красным светодиодом, если он включен.
//
process_timer()
{
    if ( blink )
    {
        toggle_out_pin(pin2);
    }
}
```

```
//--------------------------------------------------------------------------
// Обработка команды, полученной по USB.
// Формат команды: '<w | r> [s d]'
// w - команда на запись новых значений уставки s и дельты d.
// r - команда чтения текущих значений уставки и дельты.
// Пример команд.
// Чтение текущих значений: 'r'
// Запись новых значений:   'w 56 0.5'
//
process_data()
{
    new c = getchar();
    if ( 'w' == c )
    {
        setpoint = getfloat();
        delta = getfloat();
        check();
        // eeprom хранит целые числа. Поэтому вещественное значение сохраняем
        // как целое, использую функцию floatround().
        // Значение умножается на 10, чтобы сохранить его с точностью до
        // десятой градуса.
        // При чтении eeprom выполняется обратное преобразование.
        write_bkpmem(0, floatround(setpoint*10.0));
        write_bkpmem(1, floatround(delta*10.0));
        putstring("OK\n");
    }
    else
    {
        // Выводим текущие значения.
        putfloat(setpoint);
        putchar(' ');
        putfloat(delta);
        putchar('\n');
    }

    // Во входном буфере могут оставаться еще данные, которые стали не нужны.
    // Очищаем их.
    flushinput();
    // Форматный вывод буферизирован. Сбрасываем буфер в USB.
    flushoutput();
}


//--------------------------------------------------------------------------
// Функция проверяет допустимость уставки и дельты и, при необходимости,
// корректирует их.
//
check()
{
    if ( setpoint < 5.0 || setpoint > 99.0 )
        setpoint = 37.0;
    if ( delta < 0.1 || delta > 5.0 )
        delta = 1.0;
}
```