Использование скриптового языка Embedded Lua во встраиваемых системах

Антон МИКАНОВИЧ
info@promwad.com

Статья знакомит читателя с возможностями применения скриптовых языков во встраиваемых системах. Особое внимание уделено Embedded Lua, изложены сферы применения этого языка и его преимущества перед другими технологиями. В качестве практического примера рассматриваются сборка и запуск интерпретатора языка Lua на микроконтроллерной платформе.

Довольно часто при разработке сложных систем приходится максимально упрощать работу по написанию конечных приложений пользовательского пространства. При этом желательно как можно дальше отойти от архитектурных особенностей используемых систем и сконцентрироваться на алгоритмах и логике работы программ. Одно из возможных решений в таких случаях — использование простых скриптовых языков программирования, таких как Lua.

Скриптовые языки позволяют меньше задумываться над специфическими типами данных, порядке байт и других внутренних особенностях платформы. Обычно такой код менее громоздкий и более легкий в восприятии. А упрощенный синтаксис обеспечивает освоение языка в минимальные сроки.

Язык Lua является свободно распространяемым (лицензия MIT), с открытыми исходными кодами на C. Это интерпретируемый язык, а значит, программы, написанные на нем, преобразуются в байт-код непосредственно во время исполнения, без предварительной компиляции. Этот байт-код исполняется на виртуальной машине (представляющей из себя абстрактный виртуальный процессор с определенным набором регистров), что обеспечивает отличную кроссплатформность. Аналогично .Net и Java, виртуальная машина Lua содержит сборщик мусора. Цена такого подхода — ухудшение скорости работы и компактности хранимого кода, однако оба эти недостатка в Lua сведены к минимуму. Несмотря на то, что язык носит процедурную направленность и не поддерживает принципы объектно-ориентированного программирования, некоторые возможности ООП (например, наследование) вполне реализуются стандартными средствами языка.

С самого начала Lua предназначался для использования не самостоятельно, а совместно с языками более низкого уровня, такими как C. Малый размер интерпретатора, относительно высокая скорость исполнения и легкая расширяемость позволили этому языку найти применение в сфере разработки компьютерных игр (в 2003 г. Lua был признан самым популярным скриптовым языком для разработки игр по версии сообщества GameDev.net). Также Lua активно используется для написания расширений и плагинов к программным продуктам, при создании пользовательских интерфейсов и в конфигурационных файлах. На данный момент язык Lua лежит в основе Codea — единственной среды разработки для iPad, позволяющей создавать приложения на самом устройстве.

Благодаря проекту eLua этот язык нашел применение в качестве промежуточного звена между программистом и платформой во встраиваемых системах на базе микроконтроллеров. Довольно часто встречается ситуация, когда разработчик не может предоставить заказчику все исходные коды проекта, но при этом ему крайне важно заложить возможность изменения алгоритмов поведения каких-либо систем. В этом случае основную логику работы можно переложить на открытый интерпретируемый код, а остальную часть реализовать на более низкоуровневом языке, без необходимости раскрывать исходный код.

О проекте eLua

Проект eLua (Embedded Lua) позволяет использовать полноценную поддержку языка Lua во встраиваемых системах. Проект разрабатывается совместно Богданом Маринеску, программистом из Румынии, и Дадо Саттером, главой Led Lab в бразильском университете PUC-Rio. Как и сам язык, проект eLua является свободным, открытым и распространяется по лицензии MIT. Этот продукт работает как самостоятельное приложение, без необходимости использования операционных систем. В случае необходимости многопоточность обеспечивается стандартными средствами скриптового языка.

Однако eLua — это не только интерпретатор. Благодаря модульной структуре это еще и множество полезных компонентов, серьезно расширяющих его функционал. На сегодня в eLua версии 0.8 присутствуют следующие полезные компоненты:

- консоль через UART или Ethernet с возможностью передачи файлов исходных кодов через Xmodem;
- поддержка файловых систем (виртуальной ROMFS, FAT на SD/MMC-картах либо удаленной NFS);
- DHCP-клиент и DNS-resolver, а также мультиплексирование последовательных портов.

Набор используемых компонентов задается при компиляции проекта и может быть скорректирован для снижения объема кода.

Само ядро интерпретатора и используемый язык не являются сколько-нибудь урезанными по сравнению с полноценной версией Lua для ПК. Более того, разработчики eLua концентрируют свои усилия именно на расширении возможностей языка, делают его более «дружелюбным» по отношению к встраиваемым системам, уменьшая использование памяти и увеличивая производительность. Все это означает, что благодаря eLua программист получает возможность использовать абсолютно полнофункциональный язык, дополненный специфическими для встраиваемого использования функциями.

Для лучшего понимания принципов взаимодействия eLua с «внешним миром» стоит остановить внимание на таком понятии, как модуль. С точки зрения Lua-программиста модуль представляет собой таблицу в глобальном окружении, полями которой являются определенные функции и переменные, объединенные по смысловому принципу. В какой-то степени аналогом понятия модуль является namespace в C++. Определяемые пользователем модули можно использовать для разделения большой программы на логические части для «эмуляции» некоторых принципов объектно-

ориентированного программирования и для вынесения вовне обособленных частей кода, отвечающих за определенные функции.

Последнее широко используется в eLua. В виде модулей представлены все функции, связывающие интерпретатор с периферией. В версии 0.8 можно обнаружить следующие модули:

- pio — доступ к портам ввода/вывода;
- tmr — доступ к физическим и виртуальным таймерам;
- pwm — доступ к ШИМ;
- uart — доступ к последовательным портам;
- spi — доступ к интерфейсу SPI;
- net — поддержка сети TCP/IP;
- adc — доступ к АЦП;
- cpu — низкоуровневый доступ к регистрам процессора;
- pd — получение информации о платформе;
- term — доступ к терминалу;
- bit — битовые операции;
- pack — упаковка/распаковка бинарных данных в строки;
- i2c — доступ к интерфейсу I²C;
- can — доступ к интерфейсу CAN;
- rpc — удаленный вызов процедур;
- elua — управление поведением системы.

Реализация модулей содержит переменные, константы и функции на языке C, доступные в глобальном окружении Lua. Сами модули реализованы в определенном формате, описание которого можно найти на сайте проекта.

Например, чтобы отправить данные через UART с помощью одноименного модуля, достаточно в любой части Lua-скрипта вызвать функцию:

```
uart.write(ID, Data1, [Data2], …, [DataN])
```

где ID — идентификатор нужного последовательного порта; Data1–DataN — данные для передачи, в качестве которых может выступать строка либо восьмибитное число.

Синтаксис и общие принципы напоминают доступ к методу объекта в объектно-ориентированных языках, с той разницей, что таблица uart всегда присутствует в глобальном окружении без необходимости создания каких-либо переменных, объектов и т. п. Это облегчает работу с модулями и в целом упрощает использование скриптового языка Lua.

Возможности применения eLua

Проект eLua может использоваться на многих аппаратных платформах и архитектурах, как на базе микроконтроллеров и микропроцессоров, так и в виде приложений для ПК x86. В таблице представлен список платформ, поддерживаемых eLua.

Рекомендуемые минимальные требования к платформе: 32-разрядный CPU, не менее 256 кбайт flash-памяти и не менее 64 кбайт RAM. Точные требования зависят от набора инструкций конкретного процессора и используемого набора расширений.

Таблица. Платформы, поддерживаемые eLua

Платформа	Модели микропроцессоров
Cortex-M3	LM3S8962, LM3S6965, LM3S6918, LM3S9B92, LM3S1968 LPC1768 STM32F103ZE, STM32F103RE
AVR32	AT32UC3A0512, AT32UC3B0256, AT32UC3A0512
ARM7TDMI	AT91SAM7X256, AT91SAM7X512 LPC2468, LPC2888 STR711FR2
ARM966E-S	STR912FAW44
x86	Эмуляция на i386

Низкоуровневая работа с платформой организована также по модульному принципу, что позволяет разработчику легко добавлять поддержку новых платформ и плат.

Можно выделить три варианта использования eLua:

- использование готовых образов;
- сборка образа с помощью Web Builder;
- сборка образа из исходных кодов.

На своем сайте [1] авторы проекта предоставляют готовые для использования образы последней стабильной версии eLua для наиболее распространенных комплектов разработчика. Это позволяет максимально упростить процесс первоначального ознакомления с системой.

Если необходимо внести какие-либо изменения в настройки готового образа, можно воспользоваться сервисом eLua Web Builder, он доступен на официальном сайте проекта после регистрации. Этот сервис позволяет выполнить компиляцию eLua с заданными настройками на стороне веб-сервиса, получив на выходе готовый образ и его исходный код с указанными настройками.

Но наибольший интерес, конечно, представляет самостоятельная сборка eLua из исходных кодов. Код eLua доступен на сайте GitHub по адресу [2] в виде готового для кросс-компиляции проекта. Проект написан на языке C, часть кода инициализации для конкретных платформ написана на ассемблере. Также при компиляции проекта используются Python-скрипты.

Множество полезных описаний и руководств, которые помогут в настройке, использовании и написании расширений для eLua, можно найти на сайте проекта.

Пример использования eLua

Рассмотрим пример использования eLua с комплектом разработчика EK-LM3S9B92 на базе микроконтроллера Stellaris компании Texas Instruments. Микроконтроллер lm3s9b92 имеет на борту 256 кбайт flash-памяти и 96 кбайт RAM, чего вполне хватит для использования всех возможностей eLua, доступных для этой платформы.

Для повторения данного примера понадобится следующее:

- Сама плата с целевым 32-разрядным микроконтроллером (рис. 1) (в примере использована EK-LM3S9B92 от TI) и внутрисхемный отладчик для прошивки контроллера (рис. 2).

Рис. 1. Оценочная плата EK-LM3S9B92 Texas Instruments

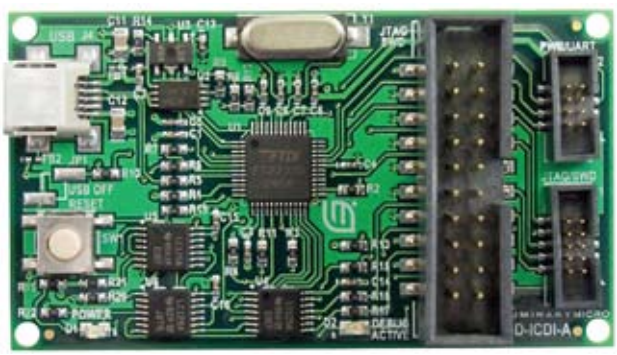

Рис. 2. Интерфейсная плата для внутрисхемной отладки

- ПК под управлением любой операционной системы семейства Linux (в примере использована ОС Ubuntu 12.04).
- Установленный ARM toolchain (в примере использован бесплатный пакет Sourcery CodeBench Lite Edition).
- Установленные пакеты SCons и Python.

Для начала загрузим исходные коды последней стабильной версии eLua (на момент подготовки статьи — 0.8). Это можно сделать с помощью Git:

```
git clone git://github.com/elua/elua.git
```

Кроме того, исходные коды можно загрузить в виде *.zip* либо *.gzip* архива через веб-интерфейс ветки проекта eLua на GitHub [2]. Так или иначе, мы получим папку с проектом eLua. Взглянем на ее содержимое. Основная часть кода расположена в папке *src*. Наибольший интерес представляет содержимое *src/platform*, это платформозависимые модули для различных микроконтроллеров (код инициализации на ассемблере, функции для работы с периферией и т. д.).

В папке *romfs* содержатся Lua-скрипты, которые будут включены в создаваемый бинарный образ проекта и будут доступны для запуска через встроенную файловую систему read-only. Если в этой папке будет обнаружен скрипт с именем *autorun.lua*, он будет автоматически запущен после загрузки системы.

В корне папки проекта находится файл *Sconstruct*, который содержит все параметры сборки eLua (используемый для конфигурирования язык — Python), включая указание ссылок на установленный набор инструментов ARM toolchain (массив *toolchain_list*) и список отношений платформа–контроллер–плата для поддерживаемых платформ (*platform_list* и *board_list*). Поскольку используемый микроконтроллер (lm3s9b92 от TI) и toolchain (Sourcery CodeBench Lite) изначально поддерживаются eLua, никаких изменений в этот файл вносить не будем.

Теперь сконфигурируем eLua для платформы lm3s. Для этого откроем файл */src/platform/lm3s/platform_conf.h* — именно здесь описан список компонентов и модулей, которые будут включены в проект.

Используемые компоненты заданы с помощью директив ***#define***. В данном примере понадобится поддержка сети (стек uIP), telnet-консоль и ROM FS. Закомментируем все директивы ***#define BUILD_*****, кроме:

```
#define BUILD_SHELL
#define BUILD_ROMFS
#define BUILD_UIP
#define BUILD_CON_TCP
```

Список подключаемых модулей (то есть расширений языка Lua) задает ***#define***-секция ***LUA_PLATFORM_LIBS_ROM***. По умолчанию все поддерживаемые модули включены и будут добавлены в проект. Те из них, которые мы не планируем использовать, можно отключить для уменьшения объема занимаемой flash-памяти. Поскольку перед нами сейчас не стоит задача экономии, оставим эту директиву без изменений.

Мы не будем использовать получение IP-адреса через DHCP (который включен по умолчанию и был отключен нами ранее), нам необходимо указать параметры сети вручную. Для этого в секции Static TCP/IP configuration файла *platform_conf.h* укажем IP-адрес 192.168.1.100. Маску подсети, адреса шлюза по умолчанию и DNS сервера можно оставить без изменения.

Подготовим тестовый скрипт. Создадим в папке *romfs* файл *hello.lua* следующего содержания:

```
print("Hello world!")
```

Теперь все готово для окончательной сборки бинарного образа eLua. Здесь используется утилита сборки Scons со следующими параметрами:

```
scons [target=lua | lualong | lualonglong] [cpu=<target_mc>] [board=<taget_board>] [cpumode=arm | thumb] [allocator = newlib | multiple | simple] [toolchain = <toolchain>] [optram = 0 | 1] [romfs = verbatim | compress | compile] [prog]
```

Не будем подробно останавливаться на описании каждого параметра, поскольку подробная информация доступна в соответствующей документации [3]. В нашем случае команда сборки будет выглядеть так:

```
scons cpu=lm3s9b92 board=ek-lm3s9b92 toolchain=codesourcery prog
```

Процесс сборки займет некоторое время, после чего, в случае отсутствия ошибок, мы получим файл вида *elua_lua_lm3s9b92.bin* в корневой папке проекта. Данный бинарный образ готов к записи во flash-память используемой платы, что мы и делаем.

После прошивки подключим нашу плату к сети Ethernet и попробуем получить доступ ко встроенной консоли через telnet:

```
telnet 192.168.1.100
```

В случае использования DNS-resolving нет необходимости в знании IP-адреса платы, и достаточно выполнить:

```
telnet elua
```

```
user@hostname:~$ telnet 192.168.1.100
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.

eLua dev-fbebb29  Copyright (C) 2007-2011 www.eluaproject.net
eLua#
```

Рис. 3. Консоль eLua

Если мы все сделали верно на предыдущих этапах, перед нами появится консоль (рис. 3).

Список доступных команд не велик, но позволяет выполнять все необходимые нам операции:

- ***help*** — вывод в консоль списка доступных команд;
- ***ver*** — отображение версии eLua;
- ***recv*** — получение файла через Xmodem (только для UART);
- ***lua <имя файла>*** либо ***lua -e 'команда' -i*** — запуск скрипта на выполнение;
- ***ls*** либо ***dir*** — отображение списка файлов в файловых системах eLua;
- ***cat*** либо ***type <имя файла>*** — вывод содержимого файла в консоль;
- ***cp <источник> <назначение>*** — копирование файлов в пределах eLua;
- ***exit*** — закрытие telnet-соединения и выход из консоли.

При использовании консоли через UART работа с ней осуществляется аналогичным образом. Параметры порта: 115200 (38400 для STR7), 8N1. Возможна сборка eLua только с одним из способов доступа к консоли (через UART либо через Ethernet), готовые образы на сайте проекта собраны с использованием консоли по UART.

Пришло время проверить работу интерпретатора, запустив тестовый скрипт *hello.lua*. Для этого введем в консоль eLua следующую команду:

```
lua /rom/hello.lua
```

В результате выполнения скрипта в консоль будет выведено приветствие интерпретатора и заветная строчка "Hello world!". Такого же результата можно добиться, выполнив:

```
lua -e 'print("Hello world!")'
```

Существует множество различных примеров Lua-скриптов для eLua с использованием различных модулей, от мигания светодиодами до веб-сервера. Эти примеры помогут в минимальные сроки освоить синтаксис языка и работу с периферией.

В целом eLua является довольно неплохой средой для исполнения скриптов во встраиваемых системах, но далеко не единственной. Как альтернативу можно рассмотреть интерпретатор PyMite, использующий язык Python. При необходимости совсем легковесного решения можно порекомендовать C-подобный язык Pawn с минимальным размером интерпретатора. Ну и, конечно же, нельзя не вспомнить гораздо более популярное решение для схожих задач — Java, оно обладает более обширными возможностями, но и проявляет бóльшую требовательность к ресурсам.

Среда-интерпретатор eLua в свою очередь может похвастаться необычайно низким порогом вхождения (даже новичок в программировании сможет освоить написание несложных Lua-скриптов), легкой расширяемостью и портируемостью без серьезного ущерба для функциональности системы.

Так или иначе, использование скриптовых языков во встраиваемых системах позволяет решать широкий круг задач, и с развитием элементной базы популярность таких решений только увеличивается.

Литература

1. www.eluaproject.net
2. Git-репозиторий проекта git://github.com/elua/elua.git
3. Официальная документация проекта eLua http://www.eluaproject.net/doc
4. Примеры программ, библиотеки, описание готовых проектов http://wiki.eluaproject.net/