Быстрая разработка программ управления устройствами на основе GNU/Linux

Дмитрий ОСИПОВ
ossipov.dmitry@gmail.com

На примере простой программы управления LCD-дисплеем показаны возможности быстрой разработки устройств со встроенной ОС Linux.

Использование одноплатных компьютеров с предустановленной ОС Linux позволяет разработчику отвлечься от тонкостей реализации работы тех или иных периферийных устройств и сконцентрироваться на алгоритме работы системы. Более того, наличие драйверов (модулей ядра) для работы с периферией и внешними интерфейсами позволяет быстро создавать прототипы системы, используя высокоуровневые скриптовые языки программирования. Особенностью таких языков является ориентированность на «склейку» уже существующих программных средств, а не на разработку новых функций и процедур. Действительно, за годы развития в ОС Linux накопилось уже достаточно утилит и программ, которые подходят для решения отдельных задач алгоритма работы системы. Ввиду небольшого количества языковых конструкций такие языки отличаются простотой освоения. Таким образом, прототип системы может быть написан гораздо быстрее, чем на более традиционном для подобных задач языке С. Наиболее естественным является использование скриптов оболочки типа Bash (стандартная оболочка Linux), Csh (оболочка с С-подобным синтаксисом) и др. Большинство пользователей Linux уже владеет на базовом уровне командным языком одной из этих оболочек.

Кроме того, в настоящее время появилось большое количество готовых программно-аппаратных решений на базе открытой ОС Linux, относящихся к так называемой «открытой аппаратуре». Этот термин означает прежде всего открытость всей документации, включая проектные файлы печатных плат. Таким образом, пользователь получает все возможности для использования и модификации не только программного, но и аппаратного обеспечения. Ядро ОС уже сконфигурировано для работы с данной платформой, прежде всего это касается необходимых для работы разнообразной периферии модулей. К наиболее популярным решениям подобного плана можно отнести Beagleboard на базе 1-Гц процессора TI DM3730 (ядро ARM Cortex-A8) [1], ECB_AT91 на базе 180-Гц процессора Atmel AT91RM9200 (ядро ARM9) [2] и др.

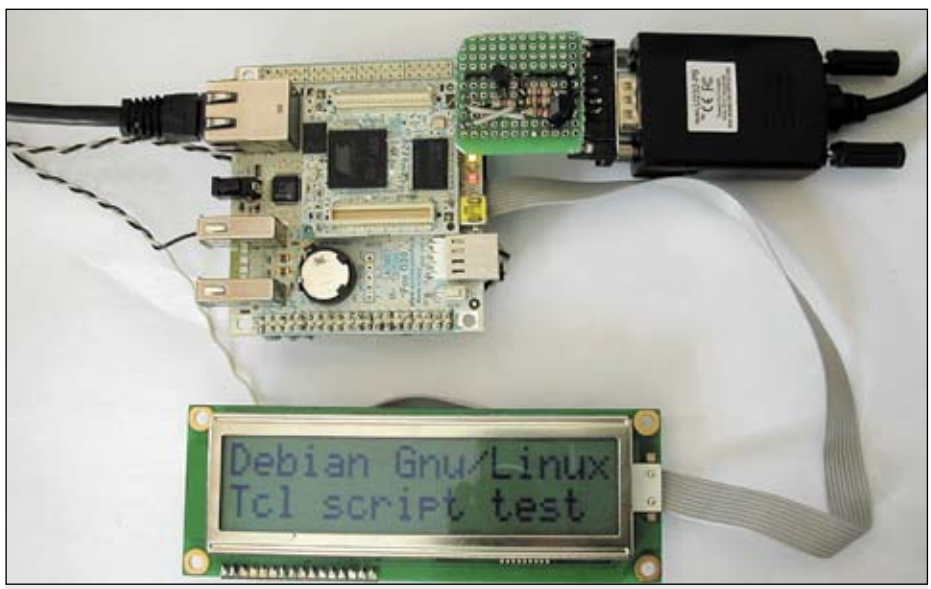

Рис. 1. Плата ACME FOX Board 20 с подключенным ЖКИ

В настоящее время для проекта разработки альфа-дозиметра в НИЯУ МИФИ [3] в качестве блока управления используется одноплатный компьютер ACME FOX Board 20 [4]. Данная материнская плата обладает всеми типичными для подобного класса устройств интерфейсами — USB (Host и HID), Ethernet, RS-232 — и поставляется вместе с SD-картой с предустановленным дистрибутивом Debian GNU/Linux 6.0. При желании можно установить любой другой, благо конфигурационные файлы для компиляции ядра находятся в свободном доступе на сайте производителя. Например, другим популярным дистрибутивом для данной платформы является Gentoo. Используемый процессор — AT91SAM9G20. Это 32-разрядный микропроцессор компании Atmel на базе ядра ARM9, с частотой работы 400 МГц. Использование ARM-процессора позволило обеспечить весьма низкий уровень потребления — устройство потребляет около 90 мА (не достигая, правда, заявленных на сайте производителя 60 мА. Такое потребление возможно только без использования SD-карты и, соответственно, ОС). На рис. 1 представлен внешний вид платы (физические размеры — 66×66 мм). Плата доступна для заказа в магазине ООО «Терра-электроника». Для подключения к HOST-системе можно использовать COM-порт (требуется специальная распайка небольшого переходника) или Ethernet. По умолчанию поставляемый вместе с платой дистрибутив Debian сконфигурирован на работу с DHCP.

Программирование устройства может осуществляться на любом языке, компилятор или интерпретатор для которого установлены в системе. Решением «в лоб» является использование языка оболочки Bash. Такое решение не требует дополнительной установки каких-либо пакетов. Программирование Bash-скриптов обычно интуитивно понятно любому пользователю UNIX-систем с некоторым опытом программирования. В качестве примера работы Bash-скрипта рассмотрим реализацию традиционной программы мигания светодиода, подключенного к выводу GPIO микропроцессора. Для подключения

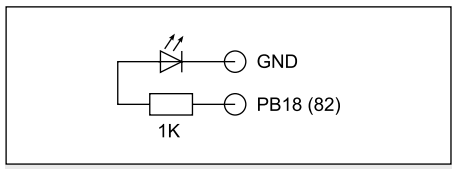

Рис. 2. Схема подключения светодиода

светодиода может использоваться обычная схема, приведенная на рис. 2.

Стандартный Linux-драйвер GPIO сопоставляет выводам процессора беззнаковые целые от 0 до MAX_INT, где MAX_INT — общее число выводов GPIO.

Конкретная привязка выводов к идентификатору является платформно-зависимой. При работе с системами с предустановленной ОС Linux необходимо обратиться к документации для определения идентификатора конкретного вывода. Для получения сведений о самостоятельном конфигурировании GPIO можно обратиться к описанию драйвера [5].

Драйвер обеспечивает возможность работы с выводами через виртуальную файловую систему sysfs, обычно монтируемую в */sys*. Для работы с GPIO используется путь */sys/class/gpio*. Данный каталог содержит файлы для контроля над GPIO, доступные только для записи, прежде всего это:

- *export* — запись идентификатора GPIO в данный файл позволяет пользователю (пользовательской программе) получить контроль над GPIO.
- *unexport* — запись идентификатора GPIO в данный файл лишает пользователя возможности контролировать GPIO, возвращая данный вывод обратно ядру ОС.

После записи идентификатора GPIO в файл *export* в каталоге */sys/class/gpio* появляется каталог */sys/class/gpioN*, где *N* — идентификатор вывода GPIO. Данный каталог также содержит несколько необходимых для управления GPIO файлов:

- *direction* — направление GPIO. Для настройки вывода можно записать следующие выражения: ***out*** — GPIO будет сконфигурирован как выход; ***in*** — GPIO будет сконфигурирован как вход; ***low*** или ***high*** для конфигурирования GPIO как выхода с начальным значением «0» или «1» соответственно. Данный файл доступен для чтения.
- *value* — значение GPIO. Если GPIO сконфигурирован как вывод, запись любого ненулевого значения в данный файл приведет к установлению на выводе GPIO «высокого» значения сигнала. Данный файл также доступен для чтения.

Файлы *edge* и *active_low* пока останутся за пределами наших интересов, они необходимы при настройке прерываний от внешних выводов.

В качестве работы с портами ввода/вывода в стандартной оболочке Bash приведем скрипт *Toggle.sh*, выполняющий мигание значения на выводе:

```
01  #!/bin/bash
02
03  if [ -d /sys/class/gpio/gpio82 ]; then
04      echo "Pin 82 is already in use"
05      exit
06  fi
07
08  trap ' echo 82 > /sys/class/gpio/unexport ; exit ' INT
09
10  echo 82 > /sys/class/gpio/export
11  echo out > /sys/class/gpio/gpio82/direction
12
13  OutValue=1
14
15  while true; do
16      echo $[!$OutValue] > /sys/class/gpio/gpio82/value
17      sleep $1
18  done
```

Рассмотрим данный скрипт чуть подробнее. В первой строке содержится указание необходимой для исполнения скрипта программы-интерпретатора. В строках 03–05 проверяется, не занят ли еще данный вывод какой-либо другой программой. Интерес представляет восьмая строка. Команда ***trap*** («ловушка») позволяет перехватить ввод пользователем сочетания клавиш <CTRL>+<C> (стандартное в UNIX сочетание для завершения программы). Так как далее используется «вечный» цикл, прерывание исполнения скрипта возможно только таким путем. Используя «ловушку», можно выполнить произвольную команду при получении скриптом какого-либо сигнала от операционной системы. В данном случае при получении сигнала «INT» (прерывание процесса по нажатии <CTRL>+<C>) скрипт сначала вернет вывод в пространство ядра и лишь затем завершит свою работу. В строках 10–11 происходит описанная выше настройка вывода. В 13-й строке переменной ***OutValue*** присваивается инициализирующее значение «1». Строки 15–18 реализуют вечный цикл, в процессе которого ***OutValue*** меняет свое значение с «1» на «0» и обратно на каждом шаге, а само значение записывается в файл *value*. Команда ***sleep*** служит для задержки выполнения на заданное пользователем в качестве аргумента скрипта время. Таким образом, для мигания светодиодом раз в 2 с необходимо ввести в консоли:

```
# ./Toggle.sh '2 s'
```

Желающим более подробно ознакомиться с программированием с использованием оболочки Bash рекомендуем ознакомиться с [6].

Однако для быстрого прототипирования и разработки более-менее сложных программ для встраиваемых систем Bash-скрипты — не самый удачный выбор ввиду очень медленного исполнения. Интерпретатор Bash последовательно выполняет все команды так, как если бы они вводились пользователем с клавиатуры. Процедурное программирование на Bash также имеет ряд ограничений.

Выходом являются скриптовые языки программирования типа Python или Tcl [7] (произносится как «тикль»), сочетающие простоту написания программ как последовательности команд со скоростью выполнения, сравнимой с компилируемыми языками типа C/C++. Кроме того, такие языки имеют расширенный по сравнению с Bash арсенал средств для классического процедурного программирования. Интерпретаторы таких языков используют предкомпиляцию, что существенно ускоряет работу программ.

Выбором автора статьи является язык Tcl ввиду его тесной интеграции со средством автоматизации работы программ Expect, рассмотренным, например, в [8]. Однако в данный момент нас скорее интересует чистый Tcl. Поддержка процедурного программирования позволяет быстро создавать пользовательские библиотеки функций. Рассмотрим библиотеку LcdTclLib, написанную автором для работы с LCD-дисплеем Winstar WH1602 со стандартным интерфейсом HD44780. Использованная схема подключения LCD показана на рис. 3. По ходу изложения мы будем пояснять требуемые конструкции языка Tcl. Первая функция, которую мы рассмотрим, выполняет инициализацию заданного в качестве аргумента вывода GPIO:

```
01  proc ExportGpio GPIO {
02      set exportIO [ open /sys/class/gpio/export w 0600 ]
03
04      puts $exportIO $GPIO
05      flush $exportIO
06
07  close $exportIO
08  }
```

По ходу отметим, что практически все «ключевые» слова Tcl являются командами и могут выполняться непосредственно в интерпретаторе языка Tclsh. Команда ***proc*** служит для описания функции, в качестве первого аргумента команда принимает имя функции, затем следуют аргумент и тело функции. Команда ***set*** во второй строке инициализирует переменную, указанную в первом аргументе, значением, указанным в качестве второго аргумента. Квадратные скобки [] служат для подстановки возвращаемого значения. Внутри скобок может находиться любая команда, возвращающая значение. В данном случае мы использовали команду ***open***, возвращающую идентификатор открываемого файла. Файл */sys/class/gpio/export* был открыт для записи с правами доступа 0600. Подробнее о правах доступа можно узнать в любой книге по администрированию ОС Linux, например [9]. Символ «$» используется для подстановки значения переменной. Таким образом, команда ***puts*** записывает в файл с идентификатором ***$exportIO*** значение переменной ***$GPIO***. Кстати, о переменных: как и Bash, Tcl «не знает» других типов, кроме строковых.

Команда ***puts*** использует буферизацию, это означает, что запись в файл может быть выполнена с задержкой (обычно все опера-

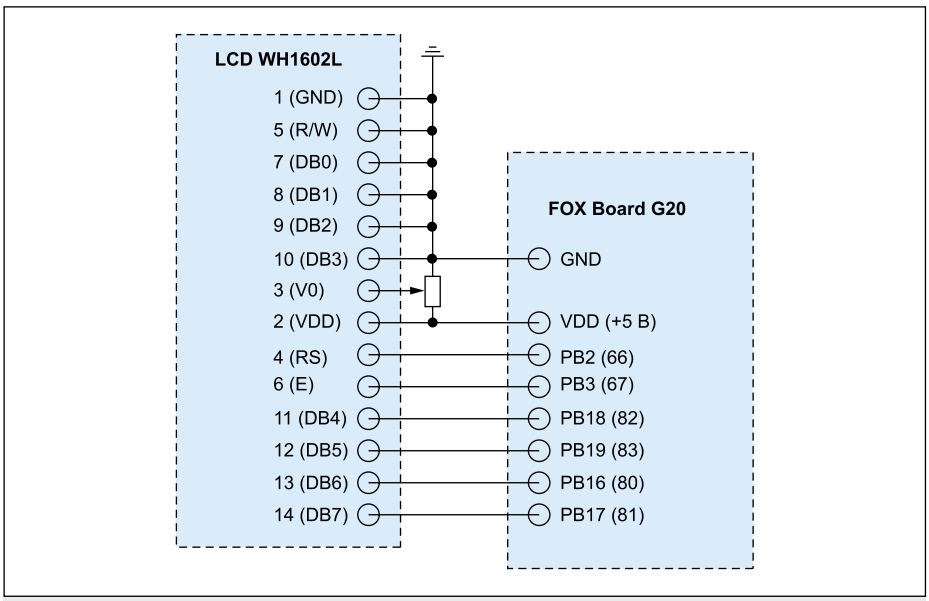

Рис. 3. Схема подключения ЖКИ

ции записи выполняются перед закрытием файла). В случае работы со специальными файлами типа *export* или *unexport* это недопустимо. Чтобы заставить произвести запись мгновенно, используется команда ***flush***.

В восьмой строке происходит закрытие файла */sys/class/gpio/export* командой ***close***.

Следующая функция выполняет настройку вывода как выхода или входа:

```
09    proc SetDirection {GPIO DIRECTION} {
10        set DirectionFile [ open /sys/class/gpio/gpio$GPIO/direction
w 0600 ]
11        puts $DirectionFile $DIRECTION
12        flush $DirectionFile
13        close $DirectionFile
14    }
```

Фигурные скобки служат для группировки, и вся комбинация ***{GPIO DIRECTION}*** воспринимается Tcl как один аргумент.

Следующая функция выполняет инициализацию всех выводов микропроцессора, к которым подключен LCD:

```
15    proc LcdGpioInit {} {
16
17        ExportGpio 82 ; # D4
18        ExportGpio 83 ; # D5
19        ExportGpio 80 ; # D6
20        ExportGpio 81 ; # D7
21        ExportGpio 66 ; # RS
22        ExportGpio 67 ; # E
23
24        SetDirection 82 out
25        SetDirection 83 out
26        SetDirection 80 out
27        SetDirection 81 out
28        SetDirection 66 out
29        SetDirection 67 out
30
31    }
```

Точка с запятой «;» используется для принудительного завершения строки. С символа «#» начинается строка комментария.

Следующие функции служат для установки значений на выводах, подключенных ко входам RS и E жидкокристаллического дисплея:

```
32    proc LcdRsSet Val {
33        set Gpio66 [open /sys/class/gpio/gpio66/value w 0600]
34        puts $Gpio66 $Val
35        close $Gpio66
36    }
37
38    proc LcdESet Val {
39        set Gpio67 [open /sys/class/gpio/gpio67/value w 0600]
40        puts $Gpio67 $Val
41        close $Gpio67
42    }
```

Функция для подачи короткого импульса на вывод микропроцессора, подключенного ко входу E жидкокристаллического дисплея:

```
43    proc ToggleE {} {
44        LcdESet 0
45        after 1
46        LcdESet 1
47        LcdESet 0
48    }
```

Так как данная функция не принимает аргументов, в качестве второго аргумента команды ***proc*** необходимо указать пустой аргумент *{}*. Команда ***after*** задерживает исполнение скрипта на время, указанное в качестве аргумента. В отличие от команды ***sleep***, использованной нами в Bash-скрипте, здесь значение указывается в миллисекундах.

Так как LCD подключен с помощью шести сигнальных линий, контроллеру дисплея может быть отправлена только половина байта за один цикл записи. Функция, реализующая передачу:

```
49    proc LcdData {Val ML} {
50        set Gpio82 [open /sys/class/gpio/gpio82/value w 0600]
51        set Gpio83 [open /sys/class/gpio/gpio83/value w 0600]
52        set Gpio80 [open /sys/class/gpio/gpio80/value w 0600]
53        set Gpio81 [open /sys/class/gpio/gpio81/value w 0600]
54
55        switch -exact -- $ML {
56            M {
57                set DB4 [expr { ( $Val & 0x10 ) >> 4 } ]
58                set DB5 [expr { ( $Val & 0x20 ) >> 5 } ]
59                set DB6 [expr { ( $Val & 0x40 ) >> 6 } ]
60                set DB7 [expr { ( $Val & 0x80 ) >> 7 } ]
61                puts $Gpio82 $DB4
62                puts $Gpio83 $DB5
63                puts $Gpio80 $DB6
64                puts $Gpio81 $DB7
65            }
66            L {
67                set DB4 [expr { ( $Val & 0x01 ) } ]
68                set DB5 [expr { ( $Val & 0x02 ) >> 1 } ]
69                set DB6 [expr { ( $Val & 0x04 ) >> 2 } ]
70                set DB7 [expr { ( $Val & 0x08 ) >> 3 } ]
71                puts $Gpio82 $DB4
72                puts $Gpio83 $DB5
73                puts $Gpio80 $DB6
74                puts $Gpio81 $DB7
75            }
76            default {}
77        }
78        close $Gpio82
79        close $Gpio83
80        close $Gpio80
81        close $Gpio81
82    }
```

Данная функция принимает два аргумента: пересылаемый байт и модификатор ***ML***, который служит для отправки младшей или старшей половины байта. Команда ***switch*** реализует обычный переключатель в зависимости от значения аргумента ***ML***. Опция команды ***-exact*** требует точного соответствия значения селектора ***ML*** одной из меток. Здесь «--» означает конец опций, следующий аргумент команды трактуется как селектор.

И, наконец, функция передачи контроллеру LCD байта данных или команды:

```
83    proc WriteToLcd {Data Mode} {
84        switch -exact -- $Mode {
85            Command {
86                LcdESet 0
87                LcdRsSet 0
88                LcdData $Data M
89                ToggleE
90                LcdData $Data L
91                ToggleE
92            }
93            Data {
94                LcdESet 0
95                LcdRsSet 1
96                LcdData $Data M
97                ToggleE
98                LcdData $Data L
99                ToggleE
100           }
101           default { }
102       }
103
104   }
```

В зависимости от значения аргумента ***Mode*** происходит передача либо данных, либо команды контроллеру LCD.

Для использования библиотеки LcdTclLib в скрипте необходимо подключить ее с помощью команды ***source***. Для примера рассмотрим скрипт инициализации LCD:

```
01    #!/usr/bin/tclsh
02    source LcdTclLib
03
04    LcdGpioInit
05
06    LcdESet 0
07    LcdRsSet 0
08    LcdData 0x30 M
09
10    ToggleE
11    ToggleE
12    ToggleE
13
14    LcdData 0x20 M
15    ToggleE

16    WriteToLcd 0x0C Command
```

```
17    WriteToLcd 0x06 Command
18    WriteToLcd 0x01 Command
19    WriteToLcd 0x80 Command
```

Данный скрипт производит инициализацию выводов GPIO, а так же базовую настройку LCD (переключение в режим работы по четырем линиям данных, отключение «мерцающего» курсора, очистку экрана). Интерпретатором данного скрипта является программа Tclsh, путь к которой должен быть указан в первой строке. Удобно сделать данный скрипт автоматически выполняемым при загрузке системы. Этого можно добиться, поместив вызов скрипта *LcdInit.sh* в файл */etc/rc.local*.

И, наконец, рассмотрим пример скрипта, реализующего вывод на LCD строки текста:

```
01    #!/usr/bin/tclsh
02    source LcdTclLib
03
04    set InputString [lindex $argv 0]
05    set InputStringLength [string length $InputString]
06    set LcdStringNumber [lindex $argv 1]
07
08    if {$InputStringLength < 16} {
09        append InputString “  ”
10    }
11
12    switch -exact -- $LcdStringNumber {
13        2       {WriteToLcd 0xC0 Command }
14        default     {WriteToLcd 0x80 Command }
15    }
16
17    for {set i 0} {$i<16} {incr i} {
18        scan [string index $InputString $i] %c Character
19        WriteToLcd $Character Data
20    }
```

Для преобразования символов строки в числовой ASCII-код в 18-й строке используется команда ***scan*** с аргументом ***%c***. ASCII-код помещается в переменную ***Character***. Команда ***string*** с аргументом ***index*** позволяет обратиться к символу строки под номером, указанным в качестве третьего аргумента.

Пример работы программы показан на рис 1. Таким образом, создание программы управления жидкокристаллическим дисплеем можно выполнить за очень небольшое количество времени, практически не вникая в тонкости С-программирования под Linux. Программы на Tcl имеют простой командный синтаксис. Отдельные места программы всегда могут быть отлажены в интерактивной оболочке Tclsh с быстрым просмотром всех необходимых результатов. Сам интерпретатор доступен для любой популярной операционной системы, таким образом, платформно-независимая часть кода всегда может быть запущена и отлажена на любом доступном компьютере.

Литература

1. **http://beagleboard.org**
2. **http://emqbit.com/products**
3. Бочаров Ю. И., Мирошниченко В. П., Онищенко Е. М. и др. Портативный прибор для детектирования источников альфа-радиоактивности // Датчики и системы. 2010. № 3 (130).
4. **http://acmesystems.it/**
5. GPIO Interfaces. Linux Documentation. — **http://lxr.linux.no/linux+v2.6.38/Documentation/gpio.txt**
6. Linux Shell Scripting with Bash. Burtch K. O. Sams Publishing. 2004.
7. Ousterhout J. K. Tcl and the Tk Toolkit // Adison-Wesley Professional. 2009.
8. Осипов Д., Бутузов В., Бочаров Ю. Автоматизация измерений с помощью программных средств Expect/Tcl на примере тестирования АЦП // Компоненты и технологии. 2011. № 8.
9. Кофлер М. Linux. Полное руководство. СПб.: Питер. 2011.