Исследование вариантов создания интеллектуальных систем робототехники на базе одноплатных компьютеров и свободных операционных систем

А.Б. Колкер, Д.А. Ливенец, Кошелева А.И., В.А. Жмудь

***Аннотация*: Обсуждаются варианты создания простейших роботов на основе конструктора «Лего» и одноплатных компьютеров с ипсользованием свободного (открытого) программного обеспечения. Приведен пример решение указанной задачи.**

***Ключевые слова*: робототехника, свободное программное обеспечение**

1. ВВЕДЕНИЕ

В предыдущих статьях был выполнен анализ современных математических и инженерных пакетов на предмет их применимости для создания систем управления методом визуального программирования. Как показывает исследование, проведенное ранее [1], наилучшими характеристиками обладает пакет *SciLab*, способный создавать исходный код модели на языке Си (попадающий под термины лицензии *GPL*). В рамках второй главы рассмотрим задачи создания встраиваемого варианта операционной системы под управлением ядра *Linux* для запуска *SciLab* приложений, а также для осуществления процессов мониторинга и телеуправления.

Таким образом, основными задачами данной главы являются:

– исследование и выбор операционной системы для одноплатного компьютера;

– разработка программной системы удаленной передачи данных устройству на малом расстоянии;

– исследование и тестирование работоспособности разработанной системы в различных условиях;

– использование совместно системы управления и системы передачи данных в программной реализации.

2 ИНТЕЛЛЕКТУАЛЬНАЯ РОБОТОТЕХНИЧЕСКАЯ СИСТЕМА

Интеллектуальная робототехническая система включает объект управления совместно со средой, в которой она работает. Объект управления представляет непосредственно механизмы перемещения инструмента и изделия. В состав манипуляторов входят исполнительные двигатели, которые осуществляют их перемещение по заданным законам $R_Д$ и $R_И$. Информация о положении выходных звеньев манипуляторов определяется датчиками, расположенными в шарнирах звеньев манипуляторов, которые получают информации о выходных координатах механизмов перемещения, их скоростях, ускорениях и силах. Основная функция системы управления манипуляторами состоит в формировании законов перемещения исполнительными механизмами манипуляторов в реальном времени $U_И(t)$ и $U_Д(t)$. Данные системы обычно работают в следящем режиме, обеспечивающем выполнение каждой степенью подвижности манипуляторов заданной траектории перемещения с требуемыми точностью, скоростью и усилием. Выходными координатами манипуляторов являются $R_Д$ и $R_И$. В результате взаимодействия инструмента с деталью создается усилие $P(t)$, которое воздействует на исполнительные органы манипуляторов. Применительно к рассматриваемой системе в качестве объекта управления и внешней среды следует рассматривать манипуляторы перемещения изделия, инструмента и непосредственно сам технологический процесс.

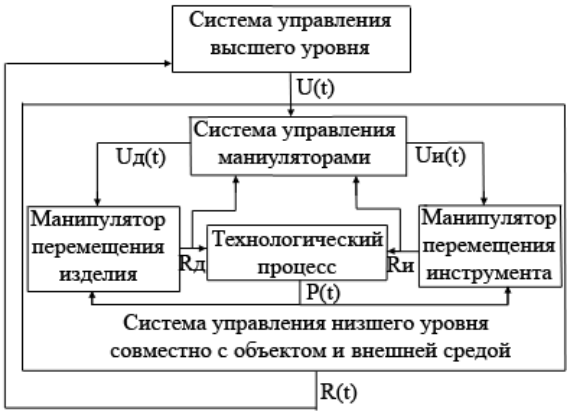

Рис. 1. Структура интеллектуальной робототехнической системы

На *рис.* 1 представлена обобщенная структура интеллектуальной робототехнической системы. В качестве системы управления высшего уровня выступает система, основанная на методах управления знаниями и принятия решений, использующая алгоритмы, рассмотренные в предыдущем разделе. Общим информационным управляющим каналом на систему управления

низшего уровня является канал передачи управляющих сигналов $U(t)$ и обратной связью от системы низшего уровня - сигнал $R(t)$. Управляющее воздействие $U(t)$ представляет выбранную программу действия из некоторого множества U и соответствующую заданной детали, либо обработке заданной поверхности детали. Какую из программ следует выбрать, решается системой высшего уровня как на основе информации от системы распознавания поверхности, так и на основе указаний оператора, управляющего робототехнической системой. Выбранная программа $U(t)$ задается непрерывно в реальном масштабе времени. Обратная связь $R(t)$ может нести полную информацию о работе системы управления низшего уровня в виде логических сигналов о ее состоянии, непрерывную информацию о геометрических размерах, качестве обработки поверхности детали и информацию о состоянии внешней среды, например, о температуре окружающей среды или двигателей, о состоянии сопутствующих обработке других устройств.

3. ОБЗОР СУЩЕСТВУЮЩИХ СРЕДСТВ ДЛЯ РЕШЕНИЯ ПОСТАВЛЕННЫХ ЗАДАЧ

3.1 Обзор, сравнение и выбор одноплатных встраиваемых компьютеров

Для реализации нашего проекта, необходим встраиваемый одноплатный компьютер [2–24].

Преимущества одноплатного компьютера:

- компактность;
- оптимизация за счет правильного подбора встроенных компонентов;
- возможность расширения за счет горизонтального масштабирования (при наличии поддержки соответствующей шины, например, PC/104).

Недостатки:

- невозможность улучшения конфигурации;
- низкая скорость шины PC/104 (ISA).

Таблица 1

Сравнение характеристик одноплатных компьютеров

Характеристика	**IEI Wafer-945gse2-n270-r10**	**Advantech PCM-4153**
Процессор	Intel® Atom™ N270 1.6GHz	AMD Geode® LX800
Оперативная память	DDR2 SDRAM 1024MB	DDR333 SDRAM 512 MB
Чипсет	Intel® 945GSE +ICH7M	AMD Geode LX800
BIOS	AMI BIOS, SPI 8Mbit Flash ROM	AWARD 4Mbit Flash BIOS
Ethernet	2 × RTL8111E PCIe Interface GbE Controllers	Поддерживаемый чипсетом:2 x 10/100Mbps - Intel 82551ER
Интерфейс ввода/вывода	2 × SATA; 6 × USB 2.0 (2 на поверхности, 4 поддерживаемых чипсетом); 1 × CF Type II; 1 × RS-232; 1 × RS-232/422/485 (поддерживаемые чипсетом); Клавиатура/мышь (поддерживаемые чипсетом); Цифровые вх/вых 8-битт , 4-бит вход/ 4-бит выход; Super I/O: ITE IT8718F	IDE интерфейс: Поддерживаемый чипсетом; Последовательный порт: 3×RS-232, Поддерживаемые чипсетом: 1xRS-232/422/485 Параллельный порт: 1 Клавиатура/мышь; 4×USB: Цифровые вх/вых: 8бит
Дополнительно	Звук: Realtek ALC662	Звук: Realtek ACL203
Питание	AT/ATX, +5V, +12V для LCD панели. Потребление: 5V×2A	AT, +5 V , +12 V для PC104 LCD инвертора. Потребление: +5 V × 1.46 A, +12 V × 0.02 A
Рабочая температура, °C	0 ~ 60	0 ~ 60
Размеры и вес	146 мм x 102 мм, 230г	96 мм x 115 мм, 162г
Поддержка ОС	Windows (все) Linux	Win XP, Win CE and Win XPe, Linux
Расширения	PC/104	PC/104
Стоимость, долл.	432	553

Сравнивая имеющиеся одноплатные компьютеры фирмы *IEI* модели *Wafer*-945*gse*2-*n*270-*r*10 и фирмы *Advantech* модели *PCM*-4153, можно выявить следующие сравнительные характеристики, представленные в *Таблице* 1. Как видно из *Таблицы* 1, главное отличие данных одноплатных компьютеров состоит в центральном процессоре и оперативной памяти. Основным

требованием для работы в нашем проекте является быстродействие, стоимость, а также совместимость по размерам с другим оборудованием, используемым в исследовании, и поддержка производителем операционной системы *Linux*. В результате был сделан выбор компьютера фирмы *IEI* (приобретен за счет средств проекта).

3.2 Обзор, сравнение и выбор операционных систем для одноплатного компьютера

В настоящее время среди высокопроизводительных компьютеров все еще актуальны легковесные дистрибутивы, которые бы весили немного и загружались менее чем за минуту, и были бы просты в установке. Был произведен обзор компактных дистрибутивов систем, и выбраны наиболее подходящие из них для сравнения.

Damn Small Linux (*DSL*) – один из первых из всех мини-дистрибутивов на основе *Debian* для архитектуры x86. Со временем *Damn Small Linux* вырос в общественный проект, в который было вовлечены сотни человеко-часов точной настройки, включая полностью автоматическую систему установки, как локальных, так и сетевых программ. Кроме того, появилась система сохранения данных, которые можно было сохранять на любой носитель, включая жесткий диск, гибкий диск или устройство *USB*. Использует программу определения устройств от *Knoppix*, которая довольно долго грузится (для маленького дистрибутива), однако, тем не менее, правильно определяет большинство оборудования. Включает свою систему пакетов .dsl, но и позволяет использовать репозитории пакетов *Debian*. Оконный менеджер по умолчанию – *Fluxbox*. Создавался для использования на старых компьютерах, поэтому имеет низкие требования к процессору и оперативной памяти и маленький размер (50 мегабайт). Минимальные системные требования:ЦП – 486*DX*, ОЗУ – 16 МБ.

Slax основан на программном обеспечении Slackware Linux. Очень легко расширяется/перестраивается дополнительными модулями. SLAX загружается в память и освобождает носитель, с которого был загружен. Использует в качестве рабочей графической среды KDE (также в дистрибутив включён Fluxbox). Одно из преимуществ SLAX – модульная структура, что позволяет легко модифицировать его под конкретные нужды пользователя. Дистрибутив можно собрать самому (модули имеют расширение *.lzm), включив в состав диска нужные приложения. Также есть скрипты, осуществляющие конвентирование в формат *.lzm пакетов других дистрибутивов, в частности, tgz2lzm для конвертирования пакетов Slackware и deb2lzm для конвертирования пакетов Debian/Ubuntu. На сайте программы есть возможность в онлайне сформировать свою сборку дистрибутива с произвольным выбором модулей, и скачать её впоследствии.

Tiny Core Linux – легковесный дистрибутив с графической оболочкой, цель которого обеспечение базовой системы с использованием BusyBox, FLTK и другого легковесного программного обеспечения. Маленький размер достигается в том числе за счет отсутствия дополнительного ПО: в нём нет ни браузера, ни офисных программ. Но все необходимые приложения могут быть легко загружены из интернета после запуска системы. Размер образа 10Мб, либо 6Мб без графической оболочки (*MicroCore*). Минимальные требования: ОЗУ 48Мб (36Мб для *MicroCore*), процессор i486DX. В TCL используется *Tiny X* вместо *X*-сервера, и *uClibc* вместо *glibc* для экономии памяти и уменьшения размера дистрибутива.

Puppy Linux – дистрибутив операционной системы GNU/Linux. Это дистрибутив, похожий в использовании на Windows, который включает все необходимые приложения, но при этом имеет достаточно малый размер (около 70 мегабайт) и скромные системные требования (процессор Pentium I, оперативная память — 32 мегабайта). Может полностью загружаться в оперативную память, не используя носитель. Отличается от других мини-дистрибутивов по скорости (загружается в 2-3 раза быстрее любого другого мини-дистрибутива) и системным требованиям (для нормальной работы дистрибутива достаточно 32 мегабайта оперативной памяти). Содержит минимальный набор самых нетребовательных к ресурсам программ. Использует оконный менеджер JWM. Puppy Linux работает практически на любом оборудовании, легок в освоении и использовании, а также легко настраивается под личные нужды конкретного пользователя.

Gentoo – дистрибутив, основной особенностью которого является сборка пакетов из исходных кодов. Это достаточно популярный дистрибутив *Linux* с мощной и гибкой технологией Portage, которая совмещает в себе возможности конфигурирования, настройки, а также автоматизированную систему управления пакетами. Отличительной особенностью *Gentoo* является возможность качественной оптимизации под конкретное аппаратное обеспечение.

В *Таблице* 2 представлены сравнительные характеристики операционных систем. Исходя из *Таблицы* 2, был выбран дистрибутив *Tiny Core Linux*, благодаря таким преимуществам, как легкость, быстрота и огромный функционал. С помощью данной системы были достигнуты поставленные задачи: управление двигателями на близком расстоянии.

Особенностью *Tiny Core Linux* является не её компактность как таковая, а как раз наоборот – расширяемость. Посредством специального ПО можно собрать систему, в которой не было бы ни единой лишней программы, причём сделать это куда быстрее и эффективнее, нежели при использовании дистрибутивов типа *Gentoo*.

Таблица 2

Сравнение дистрибутивов *Linux*

Название	Достоинства	Недостатки
Tiny Core Linux	Легкая в установке, быстрая загрузка, никаких лишних пакетов, только минимальный пакет для работы, много устанавливаемых дополнений, размер дистрибутива 8-64мб. Очень хороший функционал	Не очень удобный интерфейс. Ориентирован на опытных пользователей.
Slax	Легкая в установке, быстрая загрузка. Простая в настройке система. Удобный интерфейс. Размер около 200Мб	Много лишних пакетов, которые нам для работы не нужны. Требует минимум 256Мб *RAM*
Damn Small Linux	Легкая в установке, быстрая загрузка. Компактная, всего 50Мб. Нетребовательная система, хватает всего 24Мб *RAM*	Многих модулей не хватает, необходимо доустанавливать.
Puppy Linux	Легкая в установке, быстрая загрузка. Удобный интерфейс. Если *RAM* более 256мб – система может работать в памяти, позволяя вытащить накопитель. Простая установка дополнений. Размер дистрибутива около 100Мб.	Много лишних пакетов, которые нам для работы не нужны.
Gentoo	Сборка с нуля, установка всего, что необходимо самому.	Сборка с нуля – сложная и долгая процедура.

3.3 Обзор шасси

Гусеничное шасси *RP*5 *Tank* является идеальным и недорогим для того чтобы построить на его базе мобильного робота. Основной характеристикой данного шасси является наличие двух электромоторов в комплекте с платформой (по 1 мотору на гусеницу), которые могут работать от 6 батареек АА, что является несомненным плюсом. Шасси маневренно и легко в управлении, способно преодолевать небольшие препятствия. В механизме шасси есть технологическое отверстие в центре корпуса, которое занимает 6 батареек типа АА для питания двигателей. Кроме того, в устройстве предусмотрены крепления для компьютера или какой-либо платы.

Характеристики шасси:

- напряжение двигателей: (7.2-12В) ;
- рабочий ток: 160, 180 мА;
- максимальная скорость: 30 см/с;
- коэффициент торможения: 80:1;
- преодолеваемый подъем: > 30 °;
- эффективная нагрузка:> 7.5 Кг;
- размеры:190мм х 130мм х 60мм;
- вес 700г.

На данный момент существует много проектов на данном шасси, есть проекты RP5 Tank и Arduino. Существует много различных модулей к плате, однако все эти проекты основаны на микроконтроллере *Arduino*. До текущего момента не было реализовано программных и аппаратных модулей для встраиваемых компьютеров. В данной работе разрабатывается и исследуется система управления на одноплатном компьютере с полноценной операционной системой и сетевыми интерфейсами, так как данный набор программных и аппаратных средств предоставляет производительность и гибкость в управлении, в отличие от 8-битного контроллера *Arduino*. К тому же, одноплатный компьютер предоставляет в полное распоряжение сетевые интерфейсы *Wi-Fi* и *Ethernet*, в то время как на микроконтроллере *Arduino* эти интерфейсы доступны только через специальные платы расширения. В рамках первого этапа проекта была реализована система управления на базе одноплатного компьютера и шасси *RP*5 *Tank*. Другим перспективным шасси является более мощный вариант *DAGU*6*wd* (приобретена также в рамках гранта). Данная платформа представляет собой 6и колесную платформу с возможностью размещения до 5 кг полезной нагрузки, однако требует более мощной силовой части (рабочий ток до 6А 7.2в на двигатель). Работы по созданию робота на базе *DAGU* 6*WD* будут выполнены в рамках второго этапа проекта.

4. РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ

4.1 Установка ОС на носитель. Установка дополнений

Для того, чтобы установить данную систему на носитель необходимо выполнить несколько шагов:

- Скачать дистрибутив. *Tiny Core Linux* с официального сайта;

- С помощью программы *UnetBootIn* записать скачанный ISO образ на носитель.

Также данная программа записывает загрузчик в загрузочную область(MBR) на данный носитель. После этих простых шагов можно использовать данный носитель в одноплатном компьютере.

Для поставленных задач нам необходимо дополнительно установить пакеты программ.

Нужно использовать менеджер управления пакетами. Для начала необходимо соединиться с репозиторием, после чего появляется список доступных пакетов. При выборе какого-либо из них выводится его описание, список зависимостей, которые будут дополнительно установлены. Для установки достаточно нажать кнопку *Install Selected*, после чего пакет скачивается и устанавливается. Разрешение зависимостей происходит автоматически.

Дополнительные пакеты, необходимые для работы.

- *OpenSSH* - для удаленного подключения к одноплатному компьютеру;
- *mc* – для более удобной работы с файлами;
- *ffserver* – для захвата и трансляции видео по сети. Нужно написать конфигурацию *ffserver* (в приложении) и его необходимо положить в папку *etc*;
- *gcc* для удобства компиляции на одноплатном компьютере, необязателен;
- *gfortran* – для запуска суперблока *Scicoslab*.

Кроме того в загрузочном файле необходимо прописать автозапуск всех необходимых используемых нами программ.(*wi-fi*, *openssh*, *ffserver*). Ниже представлено содержимое файла *bootlocal.sh*.

```
#запуск openssh
sudo /usr/local/etc/init.d/openssh start
#запуск и начальные установки wi-fi
sudo ifconfig wlan0 down
sudo iwconfig wlan0 mode ad-hoc
sudo iwconfig wlan0 channel 1
sudo iwconfig wlan0 Bit 54Mb/s
sudo iwconfig wlan0 essid livliv
sudo ifconfig wlan0 192.168.1.3 netmask
255.255.255.0 up
sudo ifconfig wlan0 up
#запуск ffserver и ffmpeg для трансляции
камеры по сети
sudo ffserver
sudo ffmpeg -s 320x240 -f video4linux2 -i
/dev/video0 http://localhost:54345/webcam.ffm
```

Также необходимо модифицировать файл *.filetool.lst*, содержимое которого представлено ниже.

```
opt
home
#для доступа к ssh
usr/local/etc/ssh
#для сохранения пароля пользователя
etc/passwd
etc/shadow
```

4.2 Работа с сокетами в Linux. Программа передачи данных по сокету

Сокет (*socket*) - это конечная точка сетевых коммуникаций. Приложение просто пишет данные в сокет; их дальнейшая буферизация, отправка и транспортировка осуществляется используемым стеком протоколов и сетевой аппаратурой. Чтение данных из сокета происходит аналогичным образом. В программе сокет идентифицируется дескриптором типа *int*. Программа получает дескриптор от операционной системы при создании сокета, а затем передаёт его сервисам *socket API* для указания сокета, над которым необходимо выполнить то или иное действие.

Тип сокета определяет способ передачи данных по сети. Существуют следующие типы сокетов:

- потоковые сокеты – требуют установки соединения. Передача потока данных с предварительной установкой соединения. Обеспечивается надёжный канал передачи данных, при котором фрагменты отправленного блока не теряются, не переупорядочиваются и не дублируются;
- дейтаграмные сокеты, не требуют установки соединения. Передача данных в виде отдельных сообщений (датаграмм). Предварительная установка соединения не требуется. Обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться в пути, дублироваться и переупорядочиваться. Допускается передача сообщения нескольким получателям (*multicasting*) и широковещательная передача (*broadcasting*);
- доменные сокеты – срелства для обеспечения межпроцессного взаимодействия (*IPC*);
- низкоуровневые сокеты (raw sockets). Их отличие от обычных сокетов состоит в том, что с их помощью программа может взять на себя формирование некоторых заголовков, добавляемых к сообщению.

Рассмотрим пользовательский интерфейс работы с сокетами.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
int bind(int sockfd, struct sockaddr *addr, int
addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, void *addr, int
*addrlen);
int connect(int sockfd, struct sockaddr
*serv_addr, int addrlen);
int send(int sockfd, const void *msg, int len,
int flags);
int sendto(int sockfd, const void *msg, size_t
len, int flags, const struct sockaddr *to, socklen_t
tolen);
int recv(int sockfd, void *buf, int len, int
flags);
```

```
int recvfrom(int sockfd, void *buf, size_t len,
int flags, struct sockaddr *from, socklen_t *fromlen);

int close(int fd);
```

Рассмотрим атрибуты сокета.

С каждым сокетом связываются три атрибута: домен, тип и протокол. Эти атрибуты задаются при создании сокета и остаются неизменными на протяжении всего времени его существования. Для создания сокета используется функция socket, имеющая следующий прототип.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Домен определяет пространство адресов, в котором располагается сокет, и множество протоколов, которые используются для передачи данных. Чаще других используются домены *Unix* и *Internet*, задаваемые константами *AF_UNIX* и *AF_INET* соответственно. При задании *AF_UNIX* для передачи данных используется файловая система ввода/вывода Unix. В этом случае сокеты используются для межпроцессного взаимодействия на одном компьютере и не годятся для работы по сети. Константа *AF_INET* соответствует *Internet*-домену. Сокеты, размещённые в этом домене, могут использоваться для работы в любой *IP*-сети.

Последний атрибут определяет протокол, используемый для передачи данных. Как было отмечено ранее, часто протокол однозначно определяется по домену и типу сокета. В этом случае в качестве третьего параметра функции *socket* можно передать 0, что соответствует протоколу по умолчанию. Тем не менее, иногда (например, при работе с низкоуровневыми сокетами) требуется задать протокол явно. Числовые идентификаторы протоколов зависят от выбранного домена.

Прежде чем передавать данные через сокет, его необходимо связать с адресом в выбранном домене (эту процедуру называют именованием сокета). Иногда связывание осуществляется неявно (внутри функций *connect* и *accept*), но выполнять его необходимо во всех случаях. Вид адреса зависит от выбранного вами домена. В Unix-домене это текстовая строка - имя файла, через который происходит обмен данными. В Internet-домене адрес задаётся комбинацией *IP*-адреса и 16-битного номера порта. *IP*-адрес определяет хост в сети, а порт - конкретный сокет на этом хосте. Протоколы *TCP* и *UDP* используют различные пространства портов.

Для явного связывания сокета с некоторым адресом используется функция *bind*. Её прототип имеет вид:

```
int bind(int sockfd, struct sockaddr *addr, int
addrlen);
```

В качестве первого параметра передаётся дескриптор сокета, который мы хотим привязать к заданному адресу. Второй параметр, addr, содержит указатель на структуру с адресом, а третий - длину этой структуры.

```
struct sockaddr_in {
short int sin_family; // Семейство адресов
unsigned short int sin_port; // Номер порта
struct in_addr sin_addr; // IP-адрес
unsigned char sin_zero[8]; // "Дополнение"
до размера структуры sockaddr
};
```

Здесь поле *sin_family* соответствует полю *sa_family* в *sockaddr*, в *sin_port* записывается номер порта, а в *sin_addr* - *IP*-адрес хоста. Поле *sin_addr* само является структурой, которая имеет вид:

```
struct in_addr {
unsigned long s_addr;
};
```

Существует два порядка хранения байтов в слове и двойном слове. Один из них называется порядком хоста (host byte order), другой - сетевым порядком (network byte order) хранения байтов. При указании IP-адреса и номера порта необходимо преобразовать число из порядка хоста в сетевой. Для этого используются функции htons (Host TO Network Short) и htonl (Host TO Network Long). Обратное преобразование выполняют функции ntohs и ntohl.

Для TCP-сокетов процесс взаимодействия рассмотрен в *Таблице* 3.

Установка соединения на стороне сервера состоит из четырех этапов:

- Создание сокета (функция *socket*);
- Привязка сокета к локальному адресу (функция *bind*). Если компьютер имеет несколько сетевых интерфейсов с различными IP-адресами, вы можете принимать соединения только с одного из них, передав его адрес функции *bind*. Для соединения с клиентами через любой интерфейс необходимо задать в качестве адреса константу *INADDR_ANY*. Что касается номера порта, можно задать конкретный номер или 0;
- На следующем шаге создаётся очередь запросов на соединение (функция *listen*);

```
int listen(int sockfd, int backlog);
```

Первый параметр - дескриптор сокета, а второй задаёт размер очереди запросов. Каждый раз, когда очередной клиент пытается соединиться с сервером, его запрос ставится в очередь, так как сервер может быть занят обработкой других запросов. Если очередь заполнена, все последующие запросы будут игнорироваться.

- При наступлении готовности сервера к обслуживанию запроса вызывается функция *accept*.

```
int accept(int sockfd, void *addr, int
*addrlen);
```

Таблица 3

Схема взаимодействия серверного и клиентского сокета

Сервер	Клиент
1. Создание сокета, ожидающего запросы на соединения от клиентов.	2. Создание клиентского сокета и попытка соединения с сервером
3. Разрешение попытки соединения от клиента	-
4. Передача и прием данных.	4. Передача и прием данных.
5. Закрытие соединения.	5. Закрытие соединения.

Функция *accept* создаёт для общения с клиентом новый сокет и возвращает его дескриптор. Параметр *sockfd* задаёт слушающий сокет. После вызова он остаётся в слушающем состоянии и может принимать другие соединения. В структуру, на которую ссылается *addr*, записывается адрес сокета клиента, который установил соединение с сервером. В переменную, адресуемую указателем *addrlen*, изначально записывается размер структуры; функция *accept* записывает туда длину, которая реально была использована. Если вас не интересует адрес клиента, вы можете просто передать *NULL* в качестве второго и третьего параметров.

Следует обратить внимание, что полученный от accept новый сокет связан с тем же самым адресом, что и слушающий сокет, но дело в том, что адрес *TCP*-сокета не обязан быть уникальным в *Internet*-домене. Уникальными должны быть только соединения, для идентификации которых используются два адреса сокетов, между которыми происходит обмен данными.

На стороне клиента для установления соединения используется функция *connect*, которая имеет следующий прототип.

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

Здесь *sockfd* - сокет, который будет использоваться для обмена данными с сервером, *serv_addr* содержит указатель на структуру с адресом сервера, а *addrlen* - длину этой структуры.

После того как соединение установлено, можно начинать обмен данными. Для этого используются функции *send* и *recv*. Функция send используется для отправки данных и имеет следующий прототип.

```
int send(int sockfd, const void *msg, int len, int flags);
```

Здесь *sockfd* - это, как всегда, дескриптор сокета, через который мы отправляем данные, *msg* - указатель на буфер с данными, *len* - длина буфера в байтах, а *flags* - набор битовых флагов, управляющих работой функции.

Для чтения данных из сокета используется функция recv.

```
int recv(int sockfd, void *buf, int len, int flags);
```

В целом её использование аналогично *send*. Она точно так же принимает дескриптор сокета, указатель на буфер и набор флагов. Флаг *MSG_OOB* используется для приёма срочных данных, а *MSG_PEEK* позволяет «подсмотреть» данные, полученные от удалённого хоста, не удаляя их из системного буфера (это означает, что при следующем обращении к *recv* вы получите те же самые данные). Полный список флагов можно найти в документации. По аналогии с *send* функция *recv* возвращает количество прочитанных байтов, которое может быть меньше размера буфера.

Закончив обмен данными, нужно закрыть сокет с помощью функции *close*. Это приведёт к разрыву соединения.

```
int close(int fd);
```

Для дейтаграмных сокетов (*SOCK_DGRAM*) установка соединения не требуется. Вместо этого после открытия сокета можно использовать функции *recvfrom* и *sendto*

```
int recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
int sendto(int sockfd, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);
```

Однако можно все равно использовать соединение для дейтаграммных сокетов. На системном уровне установления соединения не произойдет, но вместо функций *recvfrom* и *sendto* можно будет использовать функции *send* и *recv*.

В данной работе для удаленного управления используются дейтаграмные сокеты (протокол *UDP*) по следующим причинам:

- отсутствие необходимости установки соединения
- высокая скорость передачи
- Отсутствие установки соединения перекладывает ответственность по обработке сбоев при передаче пакетов на разработчика системы. В данной системе нет необходимости

использовать протокол *TCP* с установлением соединения ввиду того, что потерянные данные при повторной передаче могут быть неактуальны и кроме того, повторная отправка вносит дополнительные задержки в процесс коммуникации. Протокол *UDP* избавлен от проблем с установкой соединения, однако ненадежность передачи данных может привести к неправильному порядку принятия пакетов. Для устранения ошибок принятия устаревших пакетов дополнительно передается отметка времени (*timestamp*), и все пакеты с отметкой времени меньше предыдущих пакетов отбрасываются.

Так как для принятия пакетов используется функция recv(recvfrom), блокирующая вывод, то при отсутствии пакетов функция заблокирует поток. Существует несколько способов решения данной проблемы:

- работа с сокетами в отдельном потоке;
- неблокирующие сокеты;
- установка таймаута для выхода из функции принятия пакетов.

Самым простым способом является установка таймаута чтения.

Для установки таймаута чтения пакетов используется функция *setsockopt*.

```
#include <sys/types.h>
#include <sys/socket.h>
int setsockopt(int s, int level, int optname,
const void *optval, socklen_t optlen);
```

Функция setsockopt позволяет манипулировать флагами, установленными на сокете. Флаги могут существовать на нескольких уровнях протоколов; они всегда присутствуют на самом верхнем из них.

При манипулировании флагами сокета должен быть указан уровень, на котором находится этот флаг, и имя этого флага. Для манипуляции флагами на уровне сокета *level* задается как *SOL_SOCKET*. Для манипуляции флагами на любом другом уровне этим функциям передается номер соответствующего протокола, управляющего флагами. Например, для указания, что флаг должен интерпретироваться протоколом *TCP*, в параметре *level* должен передаваться номер протокола *TCP*.

Флаг *optname SO_RCVTIMEO* устанавливает значение тайм-аута, который определяет максимальное количество времени, в течение которого функция ожидает его завершения. Он принимает значение timeval структуры с указанием предела в секундах и микросекундах.

Существует множество других флагов, документацию по ним можно прочитать в интернете.

Параметры *optval* и *optlen* используются в функции setsockopt для доступа к значениям флагов. Если значение флага не используется, то параметр *optval* может быть *NULL*.

В случае успеха возвращается ноль. При ошибке возвращается -1, а значение *errno* устанавливается должным образом.

4.3 Создание блока в ScicosLab . Разработка программного решения для работы с устройством

Для передачи данных по беспроводной сети нам необходимо написать небольшую программу на *C*. Для этого в *ScicosLab* есть специальный блок.

Существующий блок *C block* содержит в себе следующий код:

```
#include <scicos/scicos_block4.h>

void socket_client(scicos_block *block,int
flag)
{
/* init */
if (flag == 4) {
/* output computation */
} else if(flag == 1) {
/* ending */
} else if (flag == 5) {
}
}
```

Функция *socket_client* имеет на входе следующие параметры: *block* – указатель на текущий блок и *flag* – состояние работы модели. Значение флага 4 – инициализация схемы – открытие доступа к устройствам, открытие файлов

Значение флага 1 – вход в блок при моделировании – в данном случае при выполнении этого условия нужно считывать данные и отправлять их удаленному компьютеру.

Значение флага 5 – завершение моделирования, закрытие файлов и портов.

При создании блока в нашем случае надо указать, что выходных портов нет, то есть, убрать значения *Output port size*, подробнее как это сделать показано на *рис*. 2.

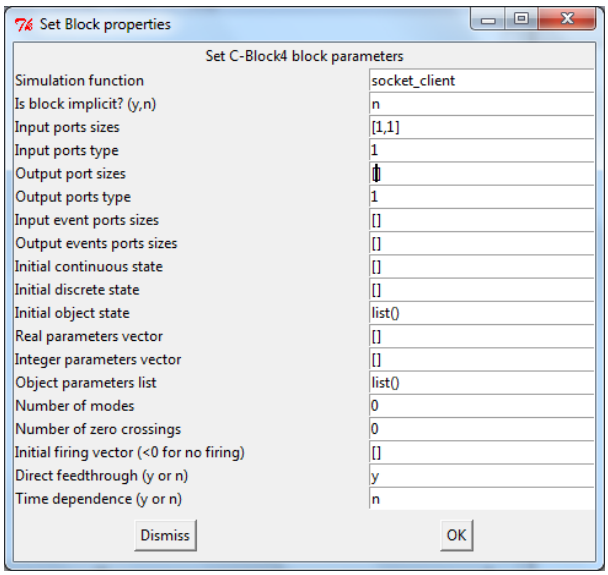

Рис. 2. Создание блока *Cblock*

Для использования входных сигналов в *C block* используется функция *GetRealInPortPtrs*(*block*,1). Далее, в код внедряется работа с сокетами в *C block*. Простейшая схема передачи синусоидального сигнала для тестирования представлена на *рис*. 3.

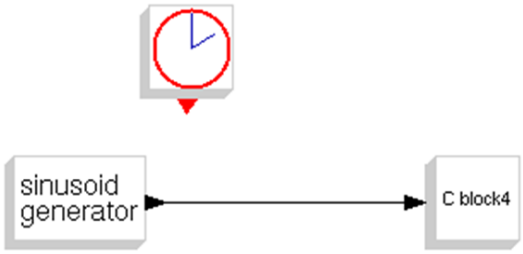

Рис. 3 Схема передачи синусоидального сигнала

Чтобы создать программу для управления шасси необходимо собрать схему в один суперблок. С помощью *scicos* и функции *Code* Generation можно получить код проекта, созданного в ScicosLab, и скомпилировать его в единую программу, которую можно запустить на одноплатном компьютере.

На *рис*. 4–5 представлена схема ШИМ в *ScicosLab*, которая передает данные по беспроводной связи на шасси на одноплатном компьютере под управлением ОС *Linux*.

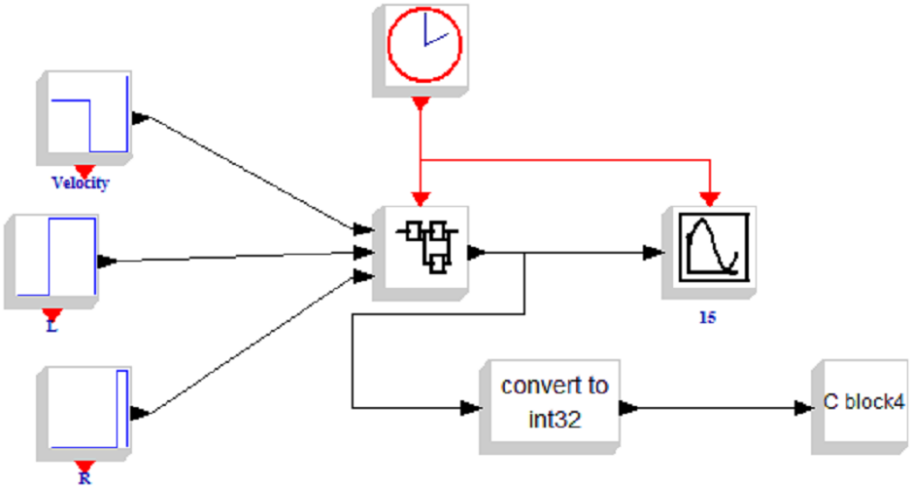

Рис. 4. Схема ШИМ для управления двигателями шасси

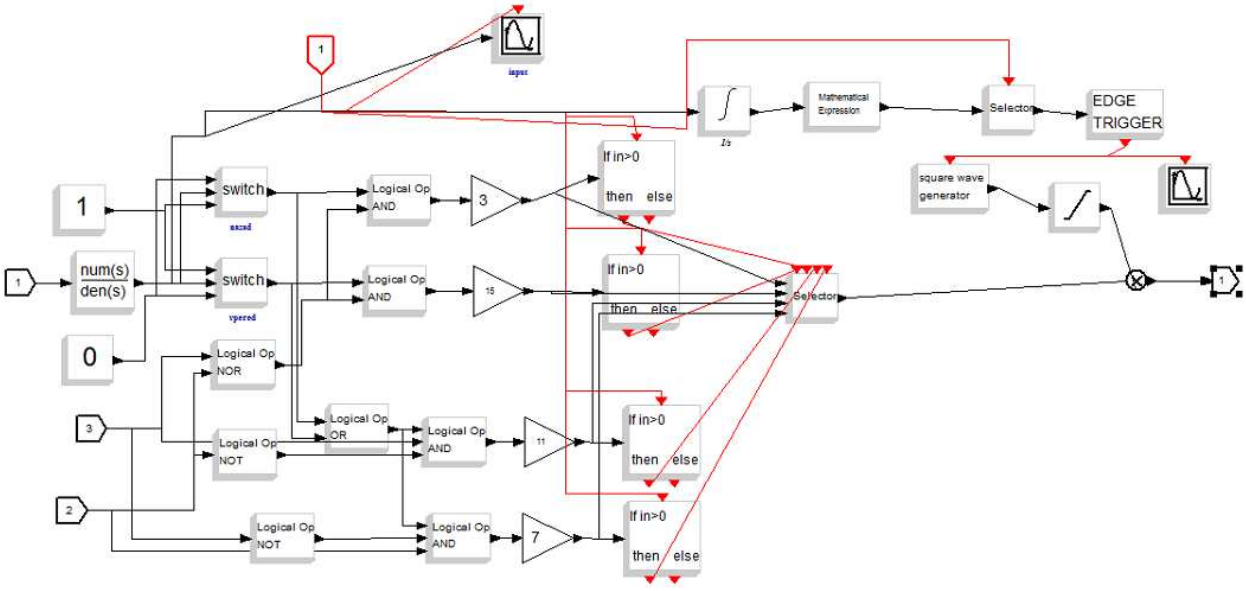

Рис. 5. Схема ШИМ для управления двигателями шасси

4.4 Работа с портами gpio

GPIO (*General Purpose Input-Output*) – это портв общего назначения, входы / выходы микроконтроллера, доступные для прямого управления.

GPIO являются двунаправленными выводами (регистрами) ввода/вывода, которые нужны для реализации функций предусмотренных производителем микросхемы (по умолчанию),

или функций задаваемых (описываемых программно) пользователем.

Управление *GPIO* можно осуществлять несколькими способами:

- из пользовательского пространства:
- использование *DebugFS* (со специально скомпилированным ядром);
- использование функций *inb* и *outb* (при наличии отображения портов в *BIOS*).
- Из пространства ядра:
- использование драйвера для работы с *GPIO*.

Самый простой способ – это использование команд чтения и запись в порт *inb* и *outb*. Однако, такой способ доступен только при наличии отображенных портов, что запорграммировано в прошивке *BIOS*.

Использование *DebugFS* для работы с *GPIO* подразумевает наличие драйвера для установленного микроконтроллера *SuperIO* и скомпилированное ядро с поддержкой *DebugFS* и *GPIO*. В случае работы через *DebugFS* порты *GPIO* будут доступны в виде файлов из пользовательского пространства.

При работе в пространстве ядра необходимо писать свой драйвер, а также запускать его на ядре, скомпилированном с поддержкой *GPIO*.

На плате одноплатного компьютера установлен контроллер ввода-вывода *SuperIO IT*8718*F*, для которого в *BIOS* была поддержка портов ввода-вывода.

Для работы с функциями *inb* и *outb* необходимо из-под суперпользователя разрешить ввод / вывод в порты. Для этого используются функции *iopern* и *iopl*

```
#include <sys/io.h>
int ioperm(unsigned long from, unsigned long num, int turn_on);
int iopl(int level);
```

Функции *ioperm* и *iopl* изменяет уровень привилегий ввода/вывода текущего процесса в соответствии с level. Этот вызов необходим для того, чтобы 8514-совместимые X-серверы могли работать под управлением Linux. Этим X-серверам необходим доступ ко всем 65536-и портам ввода/вывода, соответственно, им недостаточно вызова ioperm . В дополнение к тому, что на высоком уровне привилегий процессу разрешен неограниченный доступ к портам ввода/вывода, он может также запретить системные прерывания. Стандартный уровень привилегий ввода/вывода обычного процесса равен 0.

Функция ioperm() может дать доступ только к портам с 0x000 по 0x3ff; для других портов нужно использовать iopl(), который дает доступ ко всем портам сразу. Для предоставления доступа программе ко всем портам ввода/вывода нужно использовать системный вызов iopl(3) от суперпользователя.

- *outb, outw, outl, outsb, outsw, outsl* - функции вывода данных в порт;
- *inb, inw, inl, insb, insw, insl* - функции чтения данных порта

Это семейство функций используется для низкоуровневой работы с портами ввода/вывода. В основном, они предназначены для использования внутри ядра, но могут быть вызваны и пользовательской программой.

Для управления портами ввода-вывода, в том числе и GPIO, на одноплатном компьютере установлен микроконтроллер *ITE IT*8718F, у которого контакты 20-23 и 30-33 выведены на порты ввода и вывода соответственно. *BIOS* запрограммировано обращение к порта по адресу 0x*A*21.

Для работы с портами GPIO в нашей программе необходимо использовать функции inb и outb:

- *outb*(сигнал, 0xA21) – вывод сигнала по адресу 0x*A*21;
- *inb*(0xA22) – захват сигнала с адреса 0x*A*22.

4.5. Схема сопряжения одноплатного компьютера и шасси

Для сопряжения шасси и одноплатного компьютера необходима схема, которая бы принимала сигнал с выходов компьютера и передавала на двигатели шасси.

Для управления двигателями шасси необходимо устройство, которое бы преобразовывало управляющие сигналы малой мощности в токи, достаточные для управления моторами. Такое устройство называют драйвером двигателей.

Существует достаточно много самых различных схем для управления электродвигателями. Они различаются как мощностью, так и элементной базой, на основе которой они выполнены.

Для данного проекта наиболее подходящим является простой драйвер управления двигателями, выполненный в виде микросхемы L293D.

L293D содержит сразу два драйвера для управления электродвигателями небольшой мощности (четыре независимых канала, объединенных в две пары). Имеет две пары входов для управляющих сигналов и две пары выходов для подключения электромоторов. Кроме того, у L293D есть два входа для включения каждого из драйверов. Эти входы используются для управления скоростью вращения электромоторов с помощью широтно-модулированного сигнала (ШИМ).

L293D обеспечивает разделение электропитания для микросхемы и для управляемых ею двигателей, что позволяет подключить электродвигатели с большим напряжением питания, чем у микросхемы. Разделение электропитания микросхем и электродвигателей может быть также необходимо для уменьшения помех, вызванных бросками напряжения, связанными с работой моторов.

Принцип работы каждого из драйверов, входящих в состав микросхемы, идентичен, поэтому на рис. 2.6 приведена функциональная схема одного мотора.

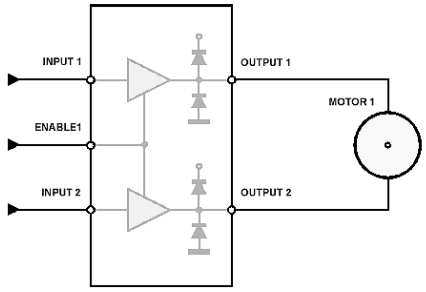

Рис. 6. Схема подключения драйвера

К выходам OUTPUT1 и OUTPUT2 подключен электромотор MOTOR1. На вход ENABLE1, включающий драйвер, подается сигнал (+5V). Если при этом на входы INPUT1 и INPUT2 не подаются сигналы, то мотор вращаться не будет.

Если вход INPUT1 соединить с положительным полюсом источника питания, а вход INPUT2 - с отрицательным, то мотор начнет вращаться. Если поменять полюсность – соответственно будет вращаться в другую сторону. Если на вход ENABLE1 ничего не подается, то при любых вариантах наличия сигналов на входах INPUT1 и INPUT2 мотор вращаться не будет.

Представить принцип работы драйвера двигателя можно, рассмотрев следующую *Таблицу* 4:

Таблица 4

Набор состояний на входах и выходах двигателя

ENABLE1	*INPUT1*	*INPUT2*	*INPUT2*	*OUTPUT1*	*OUTPUT2*
1	0	0	0	0	0
1	1	0	0	1	0
1	0	1	1	0	1
1	1	1	1	1	1

На *рис*. 7 представлены назначения выводов микросхемы L293D.

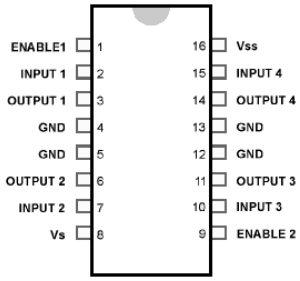

Рис. 7. Микросхема драйвера L293D

Из *рис*. 7 видно следующее:

- входы ENABLE1 и ENABLE2 отвечают за включение каждого из драйверов, входящих в состав микросхемы;
- входы INPUT1 и INPUT2 управляют двигателем, подключенным к выходам OUTPUT1 и OUTPUT2;
- входы INPUT3 и INPUT4 управляют двигателем, подключенным к выходам OUTPUT3 и OUTPUT4;
- контакт Vs отвечает за питание электродвигателей;
- контакт Vss обеспечивает питание самой микросхемы;
- четыре контакта GND соединяют с "землей" (общим проводом или отрицательным полюсом источника питания).

Рассмотрим характеристики микросхемы L293D

- напряжение питания двигателей (Vs) - 4,5...36В;
- напряжение питания микросхемы (Vss) – 5В;
- допустимый ток нагрузки - 600mA (на каждый канал);
- пиковый (максимальный) ток на выходе - 1,2А (на каждый канал);
- логический "0" входного напряжения - до 1,5В;
- логическая "1" входного напряжения - 2,3...7В;
- скорость переключений до 5 кГц;
- защита от перегрева.
- Кроме того, для сопряжения шасси и одноплатного компьютера необходим инвертор, чтобы задавать направление движения. Для этого используется микросхема К155ЛН1(шесть логических элементов НЕ). На *рис*. 8 представлено условно-графическое изображение К155ЛН1.
- 1,3,5,9,11,13 - входы;
- 2,4,6,8,10,12 - выходы;
- 7 - общий;

- 14 - напряжение питания.

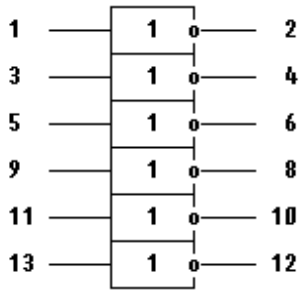

Рис. 8 Условно-графическое изображение микросхемы К155ЛН1.

На *рис*. 9 представлена структурная схема управления двигателями шасси с портов GPIO одноплатного компьютера.

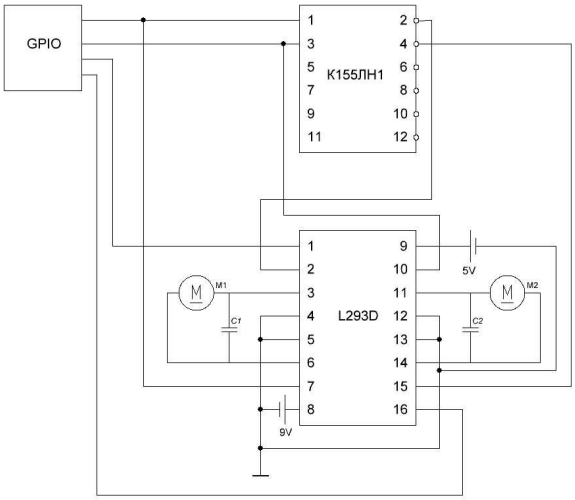

Рис. 9 Структурная схема управления двигателями с GPIO

Принцип работы схемы следующий:

На GPIO выходы одноплатного компьютера приходит сигнал в виде числа. Число кодируется в двоичную систему и на цифровые выходы устанавливаются последние 4 бита этого числа. Схема выводов представлена в *Таблице* 5.

Описание:

Выход 0 – ШИМ двигатель 1;

Выход 1 - ШИМ двигатель 2;

Выход 2 – направление двигателя 1;

Выход 3 – направление двигателя 2.

К драйверу L293D подключено +5В с цифровых выходов одноплатного компьютера для питания микросхемы и +9В от батареек типа АА для питания двигателей. Питание одноплатного компьютера от внешнего аккумулятора 5В. Параллельно к каждому двигателю подключен керамический конденсатор номиналом ~0,1 μФ, который помогает справиться с наводками от работающего электродвигателя

Таблица 5

Схема выводов цифровых выходов

Контакт	Описание	Контакт	Описание
1	Земля	2	+5В
3	Выход 3	4	Выход 2
5	Выход 1	6	Выход 0
7	Вход 3	8	Вход 2
9	Вход 1	10	Вход 0

4.6 Сборка системы

Структурная схема системы представлена на *рис*. 10. На *рис*. 11 представлен собранный макет системы управления шасси на одноплатном компьютере. Управление движением моторизованного компьютера происходит с помощью внешнего ПК, с которого на программный сервер в одноплатном компьютере посылаются управляющие сигналы; далее, эти сигналы обрабатываются на встроенном компьютере, а данные с камеры передаются отбратно на стационарный ПК. Передача данных производится по сокету по беспроводной связи, в данном случае *Wi-Fi*. На ПК может использоваться как отдельная программа передачи данных с клавиатуры, либо программа в блоке *Scilab*.

Для начала на ПК подключиться к сети *Ad-hoc livliv*. Для доступа к *Web* камере, подключенной к одноплатному компьютеру необходимо набрать в браузере в адресной строке http://192.168.1.3:54345 и открыть просмотр в удобном для пользователя формате: *flv* или *swf*. (*рис*. 12)

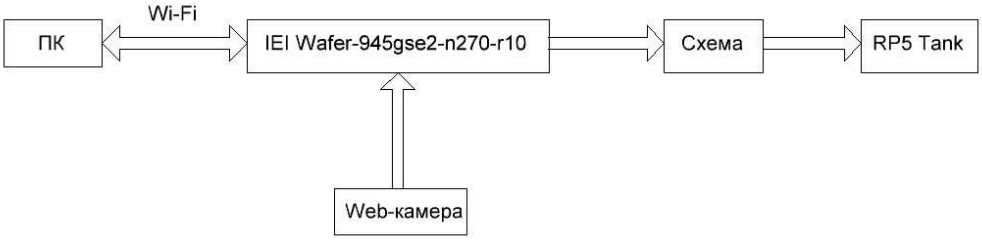

Рис. 10 Структурная схема системы управления шасси

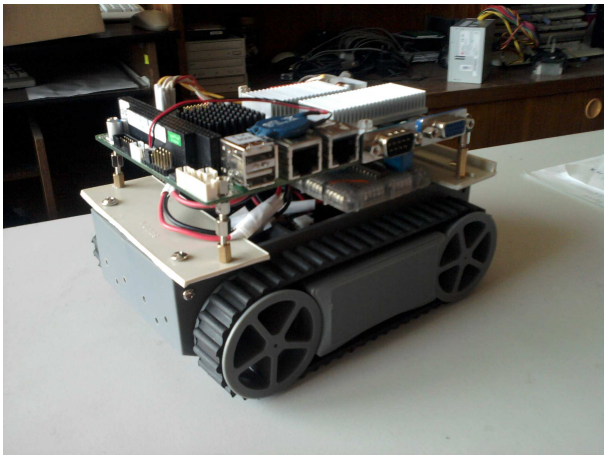

Рис. 11 Макет системы управления шасси со смонтированным одноплатным компьютером и установленной аккумуляторной батареей

Для запуска передачи сигнала необходимо подключиться по *ssh* к одноплатному компьютеру и запустить приемную часть:

```
ssh –l tc 192.168.1.3
sudo sh
cd /mnt/sda1/
./udpserver
```

А также запустить на ПК программу передачи сигнала из *ScicosLab*, либо простую программу передачи данных с клавиатуры написанную на *C*.

Для запуска программы управления с помощью клавиатуры на ПК:

```
./udpclient 192.168.1.3
```

Управление происходит с помощью клавиш *W*(вперед), *S*(назад), A(влево), *D*(вправо), пробел (стоп).

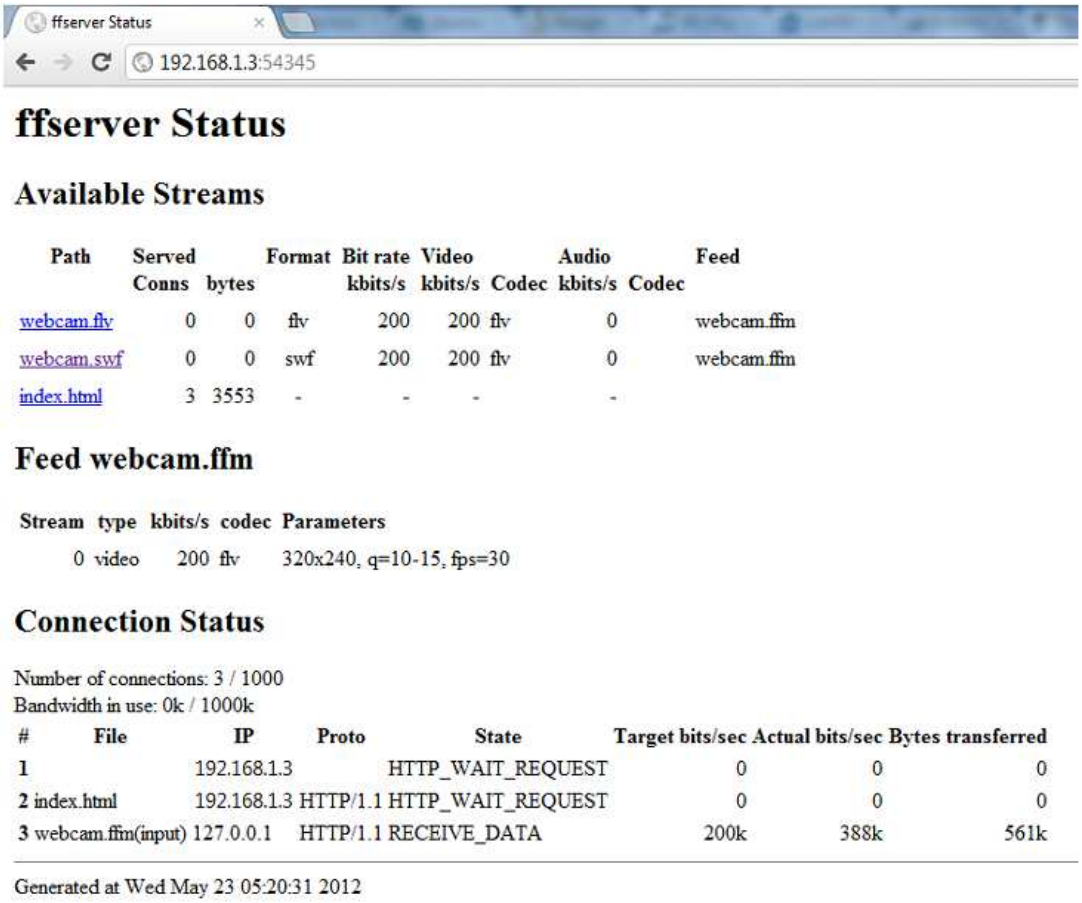

Рис. 12 Веб-интерфейс для доступа к Web-камере

4.7 Исследование работоспособности системы

Сети *Wi-Fi* (*IEEE* 802.11) могут работать в двух основных режимах:

- режим точки доступа;
- режим *Ad-hoc*.

В режиме маршрутизации вся информация передается через точку доступа-маршрутизатор. В результате топология сети приобретает конфигурацию типа «звезда».

IEEE 802.11— набор стандартов связи, для коммуникации в беспроводной локальной сетевой зоне частотных диапазонов 2,4; 3,6 и 5 ГГц.

В настоящее время существует два основных стандарта *Wi-Fi*:

- 802.11*n* – 150Мбит/с, 2,4—2,5 или 5,0 ГГц;

- 802.11*g* — 54 Мбит/с, 2,4 ГГц стандарт (обратная совместимость с b);
- 802.11*b* — Улучшения к 802.11 для поддержки 5,5 и 11 Мбит/с.

Среди перечисленных стандартов сейчас наиболее распространен стандарт 802.11*g* ввиду массовой доступности сетевых карт и чипсетов, поддерживающих его. Стандарт 802.11*n* только приобретает свою популярность, но у него есть очень много преимуществ, среди которых:

- высокая скорость работы (150 Мбит/с по одной антенне, 300Мбит/с по двум антеннам);
- два диапазона работы (2,4МГц и 5МГц) позволяют уйти от помех.

Сети *Ad-hoc* – децентрализованные беспроводные самоорганизующиеся сети, не имеющие постоянной структуры. Клиентские устройства соединяются «на лету», образуя собой неравномерную сеть. Каждый узел сети пытается переслать данные, предназначенные другим узлам. При этом определение того, какому узлу пересылать данные, производится динамически, на основании связности сети. Это является отличием от проводных сетей и управляемых беспроводных сетей, в которых задачу управления потоками данных выполняют маршрутизаторы (в проводных сетях) или точки доступа (в управляемых беспроводных сетях).

В *Таблицах* 6 и 7 представлены данные по тестированию функционирования работы системы управления шасси при свободном и загруженном канале.

Как видно из *Таблиц* 6 и 7, скорость работы в режиме *ad-hoc* немного выше, чем в централизованном виде связи. Количество потерянных пакетов приблизительно одинаковое в обоих режимах. Потери при средней загруженности канала, около 5% пакетов. Задержки при передаче пакетов незначительны и не вносят помех в процесс управления.

Таблица 6

Тестирование без загрузки канала

Вид беспроводной связи	***Ad-hoc***		***IEEE* 802.11 g**		***IEEE* 802.11 *b***	
Общее количество посланных пакетов	200	400	200	400	200	400
Количество потерянных пакетов	2-10	10-20	2-10	10-20	30-40	60-80

Таблица 7

Тестирование при средней загруженности канала

Вид беспроводной связи	***Ad-hoc***		***IEEE* 802.11*g***		***IEEE* 802.11 *b***	
Общее количество посланных пакетов	200	400	200	400	200	400
Количество потерянных пакетов	5-15	15-30	5-15	15-30	40-60	70-90

ЗАКЛЮЧЕНИЕ

В данной статье описан процесс сборки встраиваемого варианта операционной системы *Linux* для запуска *SciLab* приложений. Решены задачи обмена данными между приложением *SciLab*, выполняющимися на хост – системе и на объекте. Методы создания встраиваемых операционных систем и приложений для них могут быть рекомендованы для разработки систем управления реальных объектов.

Полученные результаты показывают, что современные свободные математические и инженерные пакеты могут быть успешно использованы для создания систем управления в робототехнике. Наиболее перспективным является применение свободно распространяемого открытого пакета *SciLab*, который позволяет проводить как моделирование, так и выгружать СИ код модели, открытый к модификации и компиляции. Исходный код хорошо анализируется, обладает интуитивно понятной структурой и легко модифицируется. В данной работе продемонстрирован пример создания аппаратно-программного комплекса управления автономным гусеничным шасси с использованием одноплатного компьютера и встраиваемой операционной системы *Linux*. Полученные результаты позволяют рекомендовать данный метод создания систем управления к использованию при синтезе методов и проектировании сложных систем автоматики и робототехники.

Работа выполнена по заданию Министерства образования и науки РФ, проекты

№ 7.559.2011 (Темплан) и ГК № П761 от 20.05.2010.

ЛИТЕРАТУРА

[1] А.Б. Колкер, Д.А. Ливенец, Кошелева А.И. Обоснование выбора программного обеспечения для робототехники. Настоящий журнал «Автоматика и программная инженерия».

[2] User manual [Электронный ресурс] - URL: http://www.ieiworld.com/files/file_pool/0B32900019596624027 2/file/WAFER-945GSE2_UMN_v2.00.pdf

[3] RTAI - Real Time Application Interface Official Website [Электронный ресурс] - URL: https://www.rtai.org/

[4] Scicos: Block diagram modeler/simulator [Электронный ресурс] – URL: http://scicos.org/

[5] The Free Platform for Numerical Computation [Электронный ресурс] - URL: http://www.scilab.org/support/documentation

[6] SciLab refence manual. Scilab Group INRIA Meta2 Project/ENPC Cergrene. - Domaine de Voluceau – Rocquencourt: - BP 105 - 78153 Le Chesnay Cedex (France). 2001 – 700 с.

[7] Nikoukhah R., Steer S. SCICOS - A Dynamic System Builder and Simulator. 1998 – 15 с.

[8] Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4 Справочное пособие / Stephen L. Campbell и др. – Springer, 2010, 332 с.

[9] Campbell S. L. Modeling and Simulation in Scilab/Scicos /S. L. Campbell, J.P. Chancelier, R. Nikoukhah – NY 10013, USA: Springer Science , 2006. – 309с.

[10] Галынский В.М. Свободно распространяемые системы компьютерной алгебры и возможности их применения в образовании / В.М. Галынский // Белорусский Государственный Университет Минск, Беларусь. – 2006. – 4с.

[11] W. R. Stevens, S. A. Rago. Advanced Programming in the UNIX® Environment: Second Edition / Addison Wesley Professional, 2005 – 960 с.

[12] Стивенс У. UNIX: Разработка сетевых приложений / СПб.: Питер, 2007 – 1040 с.

[13] Марк Дж. Рочкинд. Программирование для UNIX. Наиболее полное руководство Advanced UNIX Programming / БХВ-Петербург, Русская Редакция, 2005 г. – 704 с.

[14] Н. Мэтью, Р. Стоунс. Основы программирования в Linux / БХВ-Петербург, 2009 г.

[15] SciLab refence manual. Scilab Group INRIA Meta2 Project/ENPC Cergrene. - Domaine de Voluceau – Rocquencourt: - BP 105 - 78153 Le Chesnay Cedex (France). 2001 – 700 с.

[16] Nikoukhah R., Steer S. SCICOS. A Dynamic System Builder and Simulator,1998 15 с.

[17] Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4 Справочное пособие / Stephen L. Campbell и др. – Springer, 2010, 332 с.

[18] Scicos: Block diagram modeler/simulator. [Электронный ресурс] – URL: http://scicos.org/

[19] The Free Platform for Numerical Computation [Электронный ресурс] URL: http://www.scilab.org/support/documentation

[20] CHIPINFO – крупнейший справочник по микросхемам.[Электронный ресурс] URL: http://www.chipinfo.ru/dsheets/ic/155/ln1.html

[21] DatasheetCatalog.com – свободный онлайн-справочник электронных компонентов и полупроводников [Электронный ресурс] URL: http://www.datasheetcatalog.org/datasheet/texasinstruments/l293d.pdf

[22] IEI, the industrial computer provider focus on industrial boards, display, embedded system, digital signage, medical solution, pos, networking platforms.[Электронный ресурс] URL: http://www.ieiworld.com/product_groups/industrial/content.aspx?gid=00001000010000000001 &cid=0905066510993774014 0&id=08323535591322447606

[23] Advantech, solutions in the e-world computing and web-based automation fields.[Электронный ресурс] URL: http://www.advantech.com/products/PCM-4153/mod_A8209479-19F4-47AD-8446-84FBD131D005.aspx

[24] DFRobot-An Online Opensource Robot [Электронный ресурс] URL: http://www.dfrobot.com/wiki/index.php?title=Tank_RP5_Chassis_(SKU:ROB0007)

Колкер Алексей Борисович – доцент кафедры Автоматики НГТУ, к.т.н.,

e-mail: fiery77@yandex.ru

Ливенец Дмитрий Александрович – магистр, окончил кафедру Автоматики НГТУ,

e-mail: dmitry.livenets@gmail.com

Кошелева Алёна Игоревна – магистр, окончила кафедру Автоматики НГТУ,

e-mail: kosheleva.alyona@mail.ru

Жмудь Вадим Аркадьевич – заведующий кафедрой Автоматики НГТУ, д.т.н.,

e-mail: oao_nips@bk.ru