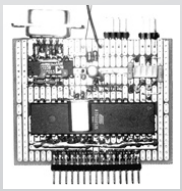
NedoPC is not PC

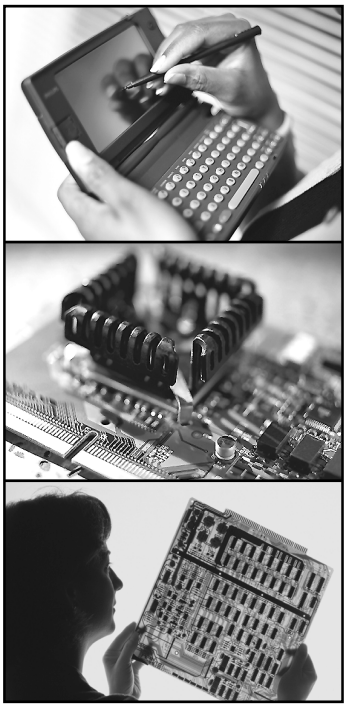
NedoPC

второй номер лето 2005

анонс



содержание



анонс следующего номера:

Новая статья из серии NedoPC-Конструктор.

Еще немного троичности.

Программирование на компьютере Sprinter.

Редакция журнала:

Главный Редактор Shaos

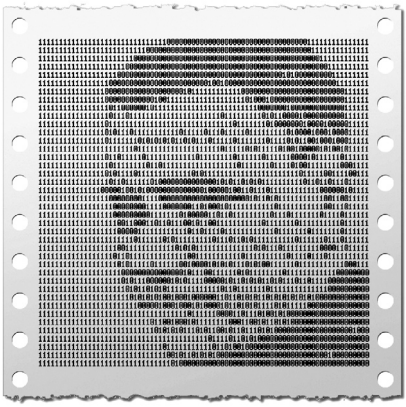
Оформление Olga

Тираж издания произвольный. Издание не подлежит регистрации, так как выпускается тиражом менее 1000 экземпляров.

E-mail: nedopc@mail.ru

http://www.nedopc.org http://www.nedopc.com

По вопросам подписки и размещения рекламы обращаться по адресу: 109451, Москва, ул.Братиславская, д.13, корп.1, кв.228, Чунину Роману Валерьевичу



колонка редактора

Ну вот мы и опять встретились, уважаемый читатель! Надеюсь ты тоже рад, что мы уже добрались до второго номера, с твоей, кстати, помощью. Второй номер выходит с небольшим отставанием от плана, однако это отставание меньше, чем было в прошлый раз – будем надеяться, что следующий номер выйдет вовремя.

Тема второго номера может показаться слишком узкой, однако мы решились пойти на такой шаг – большая часть материала касается одной и той же разработки – вычислительного модуля NedoPC-90, построенного на микроконтроллере AVR и пришедшего к нам из холодного и далекого Томска. В трех статьях подробно рассказывается о сборке модуля (это очень просто и доступно новичку), программировании модуля (причем не на низкоуровневом ассемблере, а на языке высокого уровня Си) и использовании кросс-средств для разработки (которые устанавливаются в ОС Linux). Если после прочтения материала останутся какие-то вопросы – то ты можешь пойти прямиком в раздел нашего форума, посвященный микроконтроллерам AVR, по адресу http://www.nedopc.org/forum/viewforum.php?f=82, в котором большинство топиков так или иначе касаются вычислительного модуля NedoPC-90, и задать свой вопрос непосредственно автору.

Кроме AVR в номере можно найти бессменную рекламу "АТМ Турбо-2+" и окончание бессмертного футуристического шедевра Ивана Мака "Путь домой". Также не обошли мы своим вниманием и тему программирования на компьютере Спринтер – в номере можно найти короткую статью, рассказывающую о строении заголовка исполняемого файла Спринтера – народ об этом спрашивал.

К сожалению, в этот номер не вошло ничего, касающегося троичных вычислений и построения троичного компьютера – это не значит что мы забросили эту тему. Наш троичный портал http://www.ternary.info продолжает наполнятся, а мы не останавливаем работ по созданию троичного компьютера. Не сомневайся – в будущих номерах тема троичности еще будет поднята неоднократно.

Пользуясь случаем опять не забуду призвать всех любителей паяльника к сотрудничеству – без ваших материалов журнал просто не сможет выходить. Кроме всего прочего мы планируем в будущем публиковать письма читателей и отвечать на них – поэтому пиши, уважаемый читатель, пиши обычной почтой (адрес можно найти на странице слева внизу), пиши электронной почтой (nedopc@mail.ru), наконец пиши в форум http://forum.nedopc.org – там для нашего с тобой общения подготовлен отдельный форум "Журнал NedoPC".

Не забудь воспользоваться случаем и разместить у нас свое объявление – в конце номера как и раньше – бланк бесплатного объявления. По поводу размещения рекламы – обращайся по почтовому адресу, который можно увидеть на соседней странице. А сейчас готовься познакомиться с миром AVR и переворачивай страницу :)

Главный редактор Shaos

NEDOPC-КОНСТРУКТОР

автор SfS

Часть 3. Модуль NedoPC-90 на основе AVR

«И возникла у меня гениальная мысль – сделать NedoPC конструктор, состоящий из подобных простейших элементов (индикаторы, клавиатуры, микропроцессорные платы, преобразователи интерфейсов и т.д.), которые можно продавать народу, а народ будет из них собирать что-то свое. Я думаю, спрос на такие штучки будет – хотя бы у тех, кому к своим поделкам надо срочно приделать клавиатуру или 7-сегментный индикатор с декодером.»

(C) Shaos, форум, 24 Сент. 2004, 01:38

...Идея эта вошла в мой мозг со страниц нулевого выпуска журнала NedoPC, когда я прочитал первую статью на тему «NedoPC Конструктор». Побродив немного по мозгу и постучавшись в разные места черепной коробки, идея нашла выход... Я задумался – что прежде всего должен обеспечить такой конструктор? Для кого он прежде всего предназначен? Какие минимальные требования к нему будут предъявлены народом?

Но выполнить социологическое исследование на тему «кому нужны электронные самоделки» мне вряд ли по силам. Поэтому я решил исходить из собственного опыта разработки электронной аппаратуры.

Прежде всего – выбор процессора. Главными критериями выбора были легкость программирования и доступность. Первое даже важнее чем второе. В наше время найти в магазине или заказать по почте почти любую микросхему не представляет трудности. А вот запрограммировать... Ведь далеко не каждому доступны программаторы на работе, не говоря уж о доме. Конечно, люди, профессионально занимающиеся электроникой (вроде меня), могут и усмехнуться – дескать, тоже мне, проблема – программатор найти. Но если мы желаем, чтобы конструктор использовали не только «профи от электроники», но и студенты-школьники, у которых нет доступа к программаторам, то это уже довольно серьезная проблема. На наше счастье фирма ATMEL выпускает целый ряд довольно дешевых процессоров-микроконтроллеров с ядром AVR. Эти процессора поддерживают внутрисхемное программирование, причем весь программатор – это 5 проводков, припаянных к разъему DB-25, воткнутому в LPT-порт компьютера. Исходя из различных соображений, был выбран процессор AT90S8535.

Следующий вопрос – это общение процессора с внешним миром. В статье «NedoPC Конструктор» из нулевого номера был предложен интерфейс NI-15. Сам по себе он неплох, но вот беда – он нестандартен. По нему нельзя передать данные от модуля к ПК или наоборот. А так хочется, чтобы была возможность принимать-передавать данные от ПК к устройству, собранному из конструктора, получать от него данные и выводить на экран... Чай не в 19-ом веке живем... Да и глупо не использовать для отладки рабочего устройства тот же компьютер, на котором пишется программа. Но этот вопрос решился не просто, а очень просто – в процессоре AT90S8535 имеется встроенный UART – последовательный асинхронный приемо-передатчик или почти то, что в простонародье называется «COM-порт» или «интерфейс RS232». «Почти» – потому что уровни сигналов интерфейса RS232 – ±(3..15)В, а уровни сигналов встроенного UART – 0 и +5В. Чтобы привести уровни сигналов к стандарту RS232 была использована микросхема-преобразователь уровней ADM202 (или, что почти то же – ADM232). Сигналы RxD, TxD и «Общий» выведены на разъем DB9M, точно такой, какой стоит во всех ПК. Таким образом, модуль NedoPC-90 имеет возможность соединяться с ПК с помощью обычного нуль-модемного кабеля.

Неприятно иметь «мертвый» модуль, не имеющий никакой индикации. Поэтому были добавлены 4 светодиода – благо ног у контроллера аж 40 штук и 32 из них – программируемые входы-выходы портов.

Также были выведены на отдельный разъем три входа встроенного АЦП. Возможность оцифровать аналоговый сигнал еще никому не вредила – так что не надо ее упускать.

Плюс – был добавлен «Однопроводный интерфейс», хорошо зарекомендовавший себя для связи на значительные расстояния (в моей практике – метров 200) при стандартных TTL-уровнях сигнала.

Итак, на основании всего вышеизложенного, получился модуль, имеющий следующие характеристики:

- ЦП AT90S8535.
- Память программ 8КБайт (встроенная в чип).
- Память данных 512Байт (встроенная в чип).
- Память EEPROM данных 512Байт (встроенная в чип).
- Шина NI-15 (все сигналы эмулируются программно, поэтому возможно их переназначение для иных нужд).

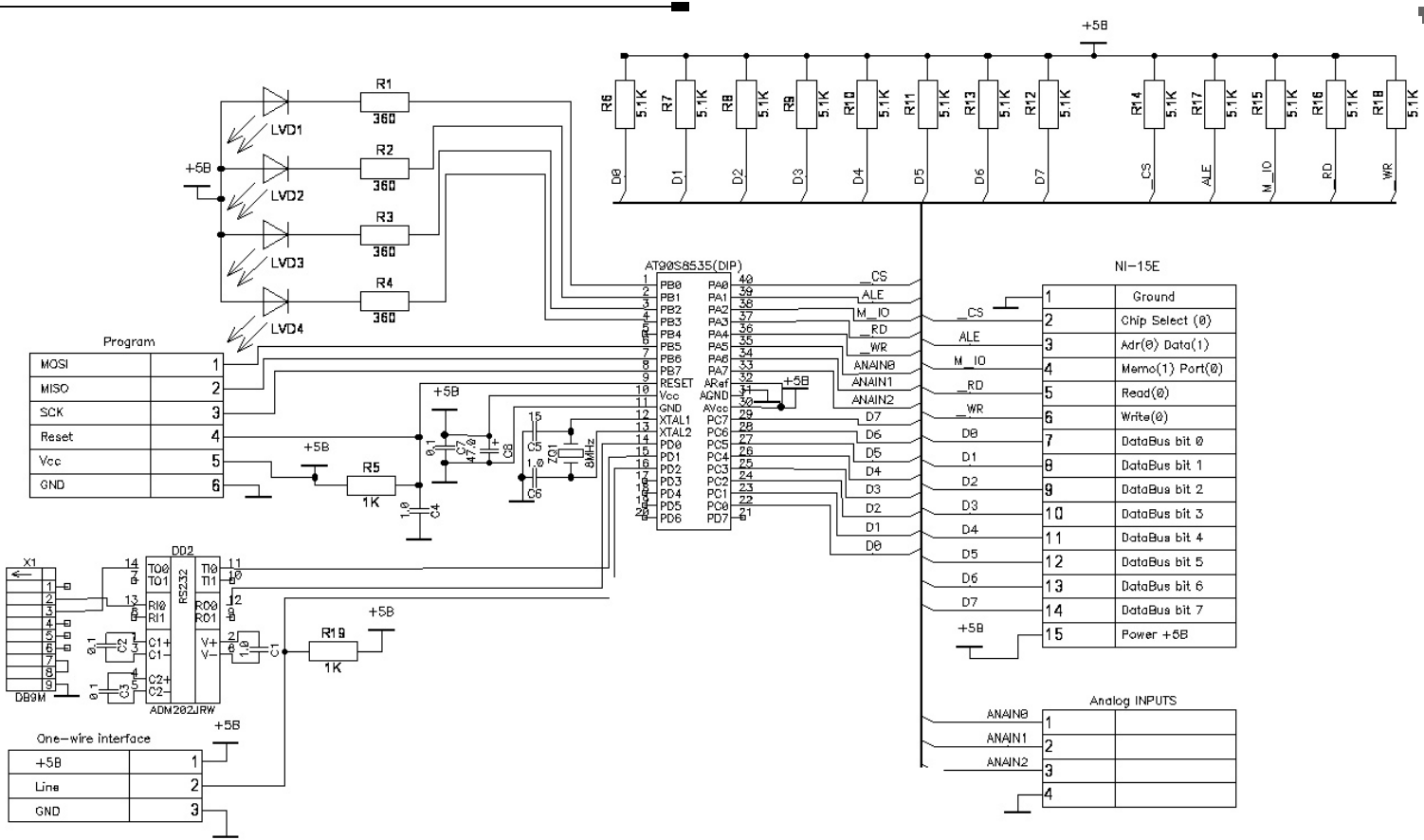


Рис.1. Схема электрическая принципиальная модуля NedoAVR-90.8535

- RS232. Усеченный стандартный RS232. Используются только сигналы RxD и TxD и «общий».
- 4 светодиода для индикации работы устройства.
- Три аналоговых входа, выведенных на отдельный разъем, позволяют оцифровать сигнал встроенным АЦП. Точность 10 бит. Но ее вряд ли можно достигнуть, поскольку нет стабильного опорного напряжения, входных фильтров и т.п.
- Однопроводный интерфейс.

Таким образом, получился модуль-ежик, имеющий сразу три цифровых интерфейса и трехканальный аналоговый вход. Такой модуль, как мне кажется, удобно использовать для ввода данных, сопряжения конструктора с ПК и другими подобными целями. Впрочем, мое дело предложить, куда приложить этот модуль – фантазия других товарищей по делу, это уж не мне решать.

Схема электрическая принципиальная модуля NedoPC-90 представлена на Рис.1. В схеме есть несколько недостатков и упрощений, но они несущественны. Например, нет монитора питания, нет какой-либо развязки по однопроводному интерфейсу. Но все эти упрощения сделаны лишь с целью упростить схему и не отпугнуть начинающих целой кучей, почти ни на что не влияющих, схемных элементов. В общем, подход примерно следующий: «для промышленности сделаем как надо, а для наколенного эксперимента – и так прекрасно».

Рис. 2. Схема кабеля для программирования. В скобках указаны номера контактов разъемов.

Так же приведу распайку кабеля (рис.2) для загрузки программы в микроконтроллер при помощи программы uisp (о ней смотри далее).

Ну и в заключение первой части приведу фото готового устройства (Рис. 3), которое получилось у меня. Не просто из хвастовства, а чтобы те, кто решит его собрать на макетке, могли прикинуть расположение и размеры основных элементов.

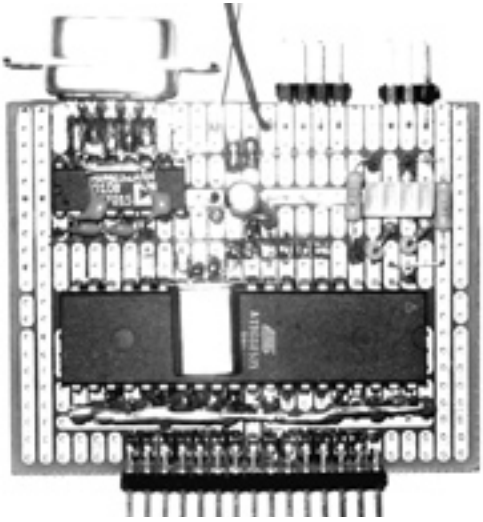


Рис.3. Внешний вид готового устройства на макетной плате. Вид сверху.

Интерфейс NI-15 – снизу платы, нумерация контактов справа налево (по рисунку).

Вот пожалуй и все, что касается «железа». О программировании NedoPC-90 будет рассказано в моих следующих статьях.

Вопросы и пожелания присылайте по электронной почте по адресу salex@cc.tpu.edu.ru.

(C) SfS, г.Томск, salex@cc.tpu.edu.ru

Статья предназначена специально для журнала NedoPC. Опубликование в других бумажных и электронных изданиях – только с разрешения автора статьи.

кросс-средства разработки для AVR

автор SfS

1. О чем эта статья

При разработке электронных устройств на микропроцессорах (или микроконтроллерах – в дальнейшем их так же будем называть микропроцессорами) часто возникает проблема – а на чем собственно писать для них программы? Где взять языки программирования? И какие?

Практика показала, что самыми распространенными языками программирования встроенных микропроцессорных систем являются ассемблер и C (который изначально является по сути портируемым макроассемблером).

В данной статье рассказывается о способах установки свободных средств разработки проекта GNU для процессоров AVR – GNU ассемблера, GNU C, GNU линкера и других утилит GNU, без которых разработка программ делается невозможной.

Все, что рассказано в данной статье, проверено и работает под управлением ОС Linux.

2. Требуемые кросс-средства

Итак, что же надо, чтобы иметь возможность программировать наши процессоры с ядром AVR? Составим список:

- Пакет. Он содержит низкоуровневые средства, необходимые для создания объектных файлов, включает в себя ассемблер для ядра AVR (avr-as), линкер (avr-ld), средства управления библиотеками (avr-ranlib, avr-ar), программы для управления объектными кодами и преобразования их из одного формата в другой (avr-objcopy), дизассемблер(avr-objdump) и утилиты avr-strip и avr-size.
- Пакет -gcc. Содержит собственно компилятор C, который так и называется avr-gcc.
- Пакет -libc. Библиотека поддержки различных процессоров с ядром AVR. Без нее процесс написания программ весьма затягивается.
- Пакет. Программа, предназначенная для загрузки исполняемого кода в память программ процессора.

3. Варианты установки

Поскольку все эти средства свободно распространяются в виде исходных текстов, скомпилированных в пакеты программ или входят в состав вашего дистрибутива Linux, то существует несколько способов установки этих пакетов. Рассмотрим их все.

4. Установка заранее откомпилированных пакетов, входящих в ваш дистрибутив

Самое простое – это если ваш дистрибутив содержит все вышеназванные пакеты с кросс-средствами. Вам достаточно их установить в систему тем способом, которым вы устанавливаете любые другие программы.

Я знаю пока лишь один дистрибутив, который содержит эти пакеты (за исключением пакета uisp) – это ALT Linux (официальная страница – www.altlinux.ru). Можете взять там как исходные тексты, так и откомпилированные пакеты.

5. Установка заранее откомпилированных пакетов, не входящих в ваш дистрибутив

Тут все чуть посложнее, чем в предыдущем случае. Скачиваем все пакеты в формате rpm в удобный вам каталог. Предположим вы скачали пакеты:

```
avr-binutils-2.14-alt1.i586.rpm
avr-gcc-3.3.1-alt1.i586.rpm
avr-libc-1.0.4-alt1.i586.rpm
uisp-20011025.rpm
```

Затем даем команды (находясь каталоге с пакетами под root'ом):

```
rpm -i avr-binutils-2.14-alt1.i586.rpm
rpm -i avr-gcc-3.3.1-alt1.i586.rpm
rpm -i avr-libc-1.0.4-alt1.i586.rpm
rpm -i uisp-20011025.rpm
```

Если же ваша система содержит менеджер пакетов apt, то (вместо указанных) лучше давать команды:

```
apt-get install avr-binutils-2.14-alt1.i586.rpm
apt-get install avr-gcc-3.3.1-alt1.i586.rpm
apt-get install avr-libc-1.0.4-alt1.i586.rpm
apt-get install uisp-20011025.rpm
```

Если в ходе выполнения указанных команд возникли ошибки, то вероятнее всего не хватает какой-то библиотеки (хотя я мало себе представляю, чтобы такая недостающая библиотека вам попалась). Посмотрите чего не хватает для установки, доставьте необходимое и повторите команду. Никаких особых сложностей процесс установки не должен вызвать.

6. Установка из исходных текстов

Самый сложный способ установки. Но зато – универсальный. Разумеется, в системе должен стоять компилятор gcc. Скачиваем пакеты (где качать - смотри следующую статью). :

```
avr-binutils-2.14.tar.bz2
avr-gcc-3.3.1.tar.gz
avr-libc-1.0.4.tar.gz
uisp-20011025.tar.gz
```

(если вы скачаете другие версии пакетов – ничего страшного не должно быть)
И далее – по порядку (все команды make install – выполняются под root'ом):

- Ставим binutils. Команды подаем, находясь в каталоге, содержащем файл avr-binutils-2.14.tar.bz2.

```
bunzip2 -c avr-binutils-2.14.tar.bz2 | tar xvf
cd avr-binutils-2.14
./configure --target=avr –prefix=/usr/local
make
make install
```

- Ставим avr-gcc. Команды подаем, находясь в каталоге, содержащем файл avr-gcc-3.3.1.tar.gz.

```
tar zxvf avr-gcc-3.3.1.tar.gz
cd avr-gcc-3.3.1.tar.gz
,/configure --target=avr --prefix=/usr/local --disable-nls –enable-language=c
make
make install
```

- Ставим avr-libc. Команды подаем, находясь в каталоге, содержащем файл avr-libc-1.0.4.tar.gz.
  Установите следующие переменные среды:

```
export CC=avr-gcc
export AS=avr-as
export AR=avr-ar
export RANLIB=avr-ranlib
export PATH=/usr/local/bin:${PATH} #(В большинстве систем можно не устанавливать)
```

  Далее:

```
tar zxvf avr-libc-1.0.4.tar.gz
cd avr-libc-1.0.4
./configure --prefix=/usr/local --target=avr --enable-languages=c –host=avr
make
make install
```

- Ставим uisp.

```
tar zxvf uisp-20011025.tar.gz
cd uisp-20011025/src
make
cp uisp /usr/local/bin
```

Ну вот и все, что хотелось бы вкратце рассказать о том, как установить кросс-средства для процессоров с ядром AVR в вашу систему. ▲

(C) SfS, г.Томск, alex@cc.tpu.edu.ru
Статья предназначена специально для журнала NedoPC. Опубликование в других бумажных и электронных изданиях – только с разрешения автора статьи.
Примечание редакции: Более подробно об установке кросс-средств для AVR можно узнать в форуме http://forum.nedopc.org

Пишем модульную ОС для AVR

автор SfS

Немного истории

Что возникает в представлении современного человека, немного знакомого с ВТ, при словах «операционная система»? Нечто большое, мощное, способное вывести на экран миллионы цветов и показать фильм, загрузить последний игровой хит или, на худой конец, проиграть mp3-файл... Судя по моему опыту – примерно так, хоть и большинство мной перечисленного делает не ОС, а другие программы, в ОС не входящие. Зачем же говорить о какой-то ОС применительно к модулю с 8К памяти программ, которые невозможно даже увеличить, и 512 байтами данных? Зачем городить огород? Да как всегда – для облегчения жизни своей и близких по духу людей, которые не хотят изобретать велосипед.

В свое время мной было сделано немало устройств на различных микроконтроллерах фирмы ATMEL с ядром AVR. Примерно через два устройства родилась идея. Идея старая и рождавшаяся у многих до меня. «А на кой черт мне делать из раза в раз одно и тоже? Инициализировать таймеры, протоколы, порты и прочую дребедень, изменив пару бит в настройках? Не проще все это привести в систему и сделать некий скелетный проект, в который можно с легкостью добавлять новые модули, удалять ненужные, изменять настройки?» Как обычно идея дошла до реализации не сразу и бродила в голове, временами натыкаясь на углы в закутках мозга, довольно долго. По пути она преображалась, обрастала новыми идейками и приняла некий законченный вид. Родились приведенные ниже пожелания к моей операционной системе для процессоров с ядром AVR:

- Вся система должна быть четко разделена на несколько уровней.
- Модули должны иметь возможность включаться и отключаться от системы изменением максимум одной строчки в каком-либо файле.
- Все модули должны писаться так, чтобы при изменении минимума настроек их можно было использовать с разными процессорами с ядром AVR.

После нескольких неудачных попыток, эти пожелания вылились в то, что я назвал AVROS (AVR Operation System), а сейчас переименовал в NedoPC-90.AVROS.v0.1.

Но, прежде чем перейти к рассказу о том, как работает NedoPC-90.AVROS и как писать под нее программы, я хочу немного сказать о том, какие средства нужны для работы с этой ОС.

Итак, вам нужны следующие средства – компилятор С, ассемблера, линкер, плюс программа, которая прошивает FLASH-ПЗУ микроконтроллера. Я пользовался для написания AVROS и пользуюсь сейчас только средствами проекта GNU – свободными программами. Если вы будете собирать NedoPC-90.AVROS чем-либо другим, то она вряд ли откомпилируется.

Все эти программы можно взять в интернете по следующим адресам:

avr-binutils – ассемблер, линкер и другие утилиты.
ftp://ftp.informatik.rwth-aachen.de/pub/gnu/binutils/
ftp://gatekeeper.dec.com/pub/GNU/binutils/avr-gcc – компилятор С.
ftp://ftp.informatik.rwth-aachen.de/pub/gnu/gcc/
ftp://gatekeeper.dec.com/pub/GNU/gcc/

avr-libc – стандартная библиотека С для AVR.
http://www.amelek.gda.pl/avr/libc/

uisp – программа для загрузки кода во flash-память микроконтроллера через LPT-порт ПК.
http://savannah.nongnu.org/download/uisp/uisp-20050207.tar.gz.

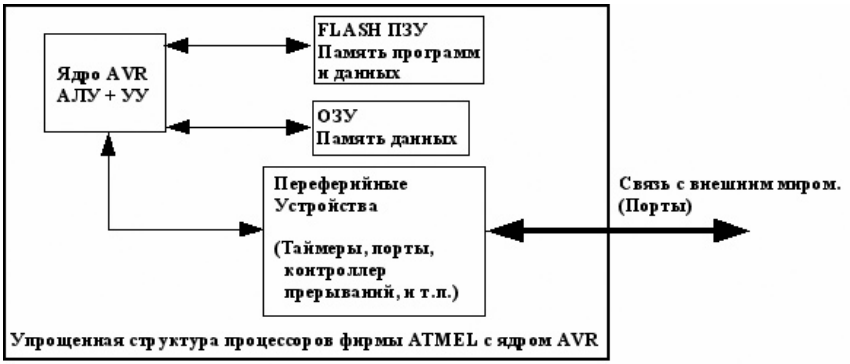


Рис.1. Упрощенная структура процессора с ядром AVR

Кроме того, все эти программы (кроме uisp) входят в дистрибутивы ALT Linux, и их можно скачать с сайта компании http://www.altlinux.ru/ или его зеркал.

О том, как установить названные мной программы, я рассказывал в предыдущей статье. Поэтому в дальнейшем буду считать, что читатель смог установить все программы на свой ПК любым удобным ему способом.

Еще замечу, что все примеры в статье рассчитаны на работу с модулем NedoPC-90 и проверены на нем. При использовании иных модулей, естественно, придется перенастраивать ОС под них.

Структура NedoPC-90.AVROS

Прежде, чем взяться за разработку структуры ОС, надо рассмотреть, какие возможности имеет и какие ограничения накладывает аппаратура, на которой предстоит работать нашей ОС. В данном случае, аппаратура – это собственно микропроцессор с ядром AVR и некоторым набором периферии. Рассмотрим, что представляют собой процессоры фирмы ATMEL с ядром AVR. Упрощенная структура такого процессора представлена на рис. 1.

Процессор имеет раздельные поля памяти программ и данных. Попросту это значит, что в ОЗУ не может быть записана программа, только переменные и другие данные. Программа может хранится только в ПЗУ. Причем (поскольку ПЗУ встроенное) объем памяти программ изменить нельзя. Это значит, что ОС и программа пользователя обязаны быть объединены и совместно загружаться в ПЗУ программ. Также следует отметить, что в ПЗУ могут храниться данные пользователя (различные константы и строки), доступ к ним осуществляется с помощью специальных команд процессора. И, хотя объем FLASH-ПЗУ изменить нельзя, объем ОЗУ в некоторых процессорах с ядром AVR увеличить можно с помощью подключения внешней микросхемы ОЗУ.

К преимуществам архитектуры с раздельными полями памяти относится то, что можно без ухищрений адресовать (при размере адресной шины 16бит) 64Кслова памяти программ (реально – 128 Кбайт, поскольку команда занимает 2 байта) и, независимо от памяти программ, до 64Кбайт памяти данных (реально – чуть меньше,поскольку младшие несколько десятков байт памяти данных заняты под регистры управления периферией).

Все процессоры с ядром AVR имеют некий набор минимальной переферии, а именно – хотя бы один таймер, контроллер прерываний, хотябы один порт ввода-вывода для связи с внешним миром, «ноги» которого можно произвольно программировать на ввод или вывод.

На основании этого микро-обзора архитектуры к нашим пожеланиям добавляются еще несколько требований:

- ОС и программы пользователя компилируются за раз в единый загрузочный модуль, который затем прошивается во FLASH-ПЗУ.
- В ОС должна иметься поддержка минимального (а в идеале – и полного) набора перефирии для ее функционирования. Причем минимальный набор (таймер, прерывания, порты ввода-вывода) должен настраиваться под систему с минимумом усилий.

На основании всех требований и пожеланий структура ОС приняла следующий вид (рис. 2). Рассмотрим назначение всех уровней ОС и способы взаимодействия между ними. Рассмотрение будем вести снизу вверх, по иерархии уровней – от взаимодействия с «железом» к пользовательским приложениям. Сразу буду ссылаться на исходные тексты ОС, которые можно взять на сайте http://shaos.ru/nedopc. Каталог, обозначенный двумя точками – самый верхний в иерархии каталогов ОС.

- Уровень аппаратного обеспечения (hardware level, HL). Файлы, относящиеся к этому уровню, хранятся в каталоге ../hl. Данный уровень обеспечивает инициализацию и взаимодействие ПО со встроенными и внешними периферийными устройствами микроконтроллера. Все остальные уровни общаются с аппаратным обеспечением только через уровень HL.
- Уровень задач реального времени (real-time level, RTL). Файлы, относящиеся к этому уровню, хранятся в каталоге ../rtl. Данный уровень обеспечивает запуск задач реального времени (по событию от таймера, внешнего прерывания и т.п.). Задачи, запущенные в данном уровне, наиболее приоритетные. Они не могут быть прерваны другими задачами. Исключение – когда задача уровня RTL запускает задачи более высоких уровней. Задачи не-RTL уровней (в отличии от задач RTL) могут быть прерваны другими задачами, но имеют защиту от повторного вхождения. Программист так же может разрешить прерывания в задаче уровня RTL, но в этом случае он должен предусмотреть сам защиту от повторного вхождения в эту задачу.
- Уровень операционной системы (operating-system level, OSL). Файлы, относящиеся к этому уровню хранятся в каталоге ../osl. Данный уровень обеспечивает запуск процессов нереального времени. Но при этом запуск может быть осуществлен по событию с уровня RTL (например, по таймеру). Кроме того, уровень OSL обеспечивает поддержку высокоуровневых протоколов (например, прием-передача пакетов по линии связи RS232 – низкоуровневая задача для RTL, а разбор принятых пакетов и формирование ответа-подтверждения - высокоуровневая задача для OSL). Так же OSL содержит различные вызовы, не имеющие отношения к задачам реального времени, например – получение кода нажатой клавиши, библиотеки вывода символов на индикатор и преобразования форматов.
- Уровень программ пользователя (users program level, UPL). Файлы, относящиеся к этому уровню, хранятся в каталоге ../upl. Этот уровень по приоритетам выполнения процессов абсолютно равноправен с уровнем OSL , но отделен от него, поскольку в каталоге ../upl находится программа пользователя. Пользователь так же может запускать свои задачи по тем же событиям и с теми же приоритетами, как и на уровне OSL. На уровне программ пользователя находится точка входа в программу – функция main().

Все настройки операционной системы хранятся в каталоге ../system. Пользователь при модернизации непользовательских уровней (то есть всех уровней, кроме UPL) NedoPC-90.AVROS также должен сохранять свои файлы настройки в этом каталоге, следуя определенным правилам. В следующих разделах мы подробно рассмотрим как писать

Рис. 2. Структура операционной системы NedoPC-90.AVROS

приложения пользователя и модули других уровней операционной системы. Все примеры будут приводится исходя из того, что ОС работает на плате-модуле NedoPC-90.

Правила написания приложений уровня пользователя (UPL)

Самое простое – это написание уровня пользователя. Для этого достаточно в каталоге ../upl Написать необходимые вам процедуры в файле main.c. Программа пользователя начинает выполняться с функции int main().

```
int main()
     {AUTOINIT_ALL_MODULES();
     // Код программы пользователя
     .....
     // конец программы пользователя
     return(0);
     }
```

Обратите внимание на самую первую строчку функции – AUTOINIT_ALL_ MODULES();. Эта строчка обязательно должна быть самой первой в функции int main(), иначе OS не сможет корректно запустится.

Далее идет код программы пользователя – в принципе абсолютно любой. В качестве примера рассмотрим как зажечь первый и четвертый светодиоды.

Светодиоды управляются через встроенный порт B контроллера AT90S8535 младшими четырьмя битами. При выводе 0 светодиод зажигается, при выводе 1 – гаснет.

```
int main()
     {AUTOINIT_ALL_MODULES();
     // Код программы пользователя
// Инициализируем светодиодный порт (программируем биты 0 – 3 на вывод)
     output(LEDDDR,input(LEDDDR) | 0x0F);_NOP();_NOP();.
     // Зажигаем 1й и 4й светодиоды (то есть выводим 0 в биты 0й и 3й)
     output(LEDPORT,(input(LEDPORT) | 0x0F) & ~(0x01 | 0x08));_NOP();_NOP();
     // конец программы пользователя (бесконечный цикл)
     while(1){}
     return(0);
}
```

Как видим, ничего сложного.

Часто программа пользователя разбита на несколько *.c файлов с процедурами. Если это так – то самый простой способ откомпилировать такое приложение – включить все *.c файлы пользователя в файл main.c с помощью директивы #include. Есть и другой способ – включить пользовательские файлы в Makefile. Рассмотрим оба способа.

Пусть у нас имеется процедура зажигания светодиодов void uled(), сохраненная в файле ../upl/uled.c.

```
void uled()
     {// Инициализируем светодиодный порт (программируем биты 0 – 3 на вывод)
     output(LEDDDR,input(LEDDDR)0x0F);_NOP();_NOP();.
     // Зажигаем 1й и 4й светодиоды (то есть выводим 0 в биты 0й и 3й)
     output(LEDPORT,(input(LEDPORT) | 0x0F) & ~(0x01 | 0x08));_NOP();_NOP();
}
```

И пусть нам надо вызвать ее из int main().

СПОСОБ 1-ый.

Изменяем файл main.c. Добавляем в него строчку #include «uled.c».

```
// Включаем файл с нашей процедурой.
#include «uled.c»
// Основная программа
     int main()
               {AUTOINIT_ALL_MODULES();
               // Код программы пользователя
     uled(); // Вызов нашей процедуры
               // конец программы пользователя (бесконечный цикл)
               while(1){}
               return(0);
}
```

СПОСОБ 2-ой.
Создаем заголовочный файл ../upl/uled.h, такого содержания:

```
// Файл ../upl/uled.h
void uled();
```

Открываем Makefile. Там есть строчки примерно такого вида:

```
# Hardware level
    HL-SRC = timer0.c int0.c adc0.c uart0.c pwm.c
    HL-OBJ = hl.o
    # Real-Time Level
    RTL-SRC = rt_task.c
    RTL-OBJ = rtl.o
    # Operating-System Level
    OSL-SRC = syscall.c
    OSL-OBJ = osl.o
    # User Program Level
    UPL-SRC = main.c
UPL-OBJ = upl.o
```

Эти строчки говорят какие файлы в какой уровень будут включены при компиляции. Изменяем строчку UPL-SRC (поскольку мы работаем сейчас на уровне пользователя UPL), добавив в нее наш новый файл.

```
UPL-SRC = main.c uled.c
```

Изменяем файл main.c. Заметьте, что при таком способе добавлять строчку #include «uled.c» в файл main.c не надо. Но файл ../upl/uled.h должен быть создан обязательно! Даже если он пустой. Иначе компилятор может выдать ошибку.

```
// Основная программа
    int main()
            {AUTOINIT_ALL_MODULES();
            // Код программы пользователя
    uled(); // Вызов нашей процедуры
            // конец программы пользователя (бесконечный цикл)
            while(1){}
            return(0);
}
```

Первый способ лучше применять, когда проект небольшой, и все процедуры пользователя умещаются в 1-2 дополнительных файла. Если же файлов с процедурами пользовательского уровня много – то лучше применить второй способ. Впрочем, это дело вкуса и личных предпочтений.

Правила написания модулей непользовательских уровней (HL, RTL, OSL)

Написание модулей для системных уровней (HL, RTL, OSL) почти ничем не отличается от написания пользовательского уровня.

В качестве примера рассмотрим как добавить к ОС модуль поддержки интерфейса NI-15:

- Принимаем решение на каком уровне будет размещен модуль. Если модуль напрямую взаимодействует с аппаратурой, то он должен располагаться в каталоге hl (уровень аппаратного обеспечения). В нашем случае это так и есть. (Иначе модуль должен был быть размещен на уровнях RTL или OSL).
- Пишем драйвер, который сохраним в файле hl/ni_15.c. Общая структура файла hl/ni_15 .c следующая:

```
#ifdef NI_15_ENABLED
// код драйвера
void ni_15_init(){/*код инициализации*/}
....
#endif /* NI_15_ENABLED */
```

- Пишем заголовочный файл hl/ni_15.h. Этот файл обязателен. В нем должен быть определен макрос ni_15_init_mac(). Это рекомендуется делать следующим образом.

```
      #ifdef NI_15_ENABLED
      // Ссылка на процедуру инициализации модуля
      #define ni_15_init_mac() ni_15_init()
// Все определения для драйвера, которые постоянны
...
#else
#define ni_15_init_mac()
      #endif /* NI_15_ENABLED */
```

Макрос ni_15_init_mac() используется для автоматического вызова процедуры инициализации драйвера. Если даже драйвер не используется или процедура инициализации не нужна, то этот макрос все равно должен быть определен, хоть он и пустой. Общее правило определения имени макроса начальной инициализации #define <имя_модуля>_init_mac() <Имя процедуры инициализации>(), где <имя_модуля> – имя *.h файла, который описывает модуль. (В нашем случае ni_15, т.к. файл называется ni_15.h). <Имя процедуры инициализации> – имя процедуры инициализации, описанной в файле с кодом драйвера (В нашем случае это файл ni_15.c, процедура void ni_15_init()).

- Создаем файл system/res_ni_15.h. В этом файле обязательно в начале должна быть строчка #define NI_15_ENABLED. Если эту строчку закомментировать, то все связанное с нашим драйвером не будет включено в код программы. Таким образом, чтобы добавить или удалить модуль из программы, достаточно раскомментировать или закомментировать всего одну строчку в файле res_<имя модуля>.h в каталоге system. Имя файла обязательно должно иметь формат: res_<имя_модуля>.h, поскольку файл добавляется в систему автоматически. Все файлы в каталоге system, имеющие формат имени res_<имя_модуля>.h, при компиляции системы воспринимаются как заголовочные файлы модулей и будут включены в систему.
  Общая структура файла system/res_userpwm.h следующая:

```
#define NI_15_ENABLED /* Закомментировать для отключения модуля */
    #ifdef NI_15_ENABLED
    // Описание всех определений, которые часто изменяются
    #endif /* NI_15_ENABLED */
```

- Открываем Makefile. Там есть строчки примерно такого вида:

```
# Hardware level
    HL-SRC = timer0.c int0.c adc0.c uart0.c pwm.c
    HL-OBJ = hl.o
    # Real-Time Level
    RTL-SRC = rt_task.c
    RTL-OBJ = rtl.o
    # Operating-System Level
    OSL-SRC = syscall.c
    OSL-OBJ = osl.o
    # User Program Level
    UPL-SRC = main.c
    UPL-OBJ = upl.o
```

Поскольку наш новый модуль находится в каталоге hl, то нетрудно догадаться, что для его добавления надо изменить строчку:

```
HL-SRC = timer0.c int0.c adc0.c uart0.c pwm.c
```

на

```
HL-SRC = timer0.c int0.c adc0.c uart0.c pwm.c ni_15.c
```

То есть просто добавить имя файла с исходными текстами нашего нового модуля в список файлов с исходными текстами того уровня, где расположен наш модуль.

Вызов процедур по системному таймеру

После запуска системы прерывания всегда разрешены, и периодически приходят прерывания от системного таймера. Пользователь может добавлять процедуры своих модулей для вызова их по прерыванию. Для

этого надо открыть файл rtl_mac.h (для модулей уровней upl и osl) или hl_mac.h (для модулей уровня rtl), раскомментировать в нем одну из строчек и добавить в нее имя своей процедуры, которая должна вызываться по таймеру.

Для модулей уровней upl и osl доступны следующие периоды вызова процедур:

- 0.1 сек. (вид #define UPL_TASK_100MS_xx или #define OSL_TASK_100MS_xx)
- 1 сек. (вид #define UPL_TASK_1S_xx или #define OSL_TASK_1S_xx)
- 1 минута (вид #define UPL_TASK_1M_xx или #define OSL_TASK_1M_xx)
- 1 час (вид #define UPL_TASK_1H_xx или #define OSL_TASK_1H_xx)

Для модулей уровня rtl доступны следующие периоды вызова процедур:

- FMAX (вид #define T_INT_MAX_xx)
- 0.1 сек. (вид #define T_INT_100MS_xx)
- 1 сек. (вид #define T_INT_1S_xx)
- 1 минута (вид #define T_INT_1M_xx)
- 1 час (вид #define T_INT_1H_xx)

Частота FMAX – максимально возможная частота прерываний таймера. Определяется как F_CLK (частота кварца), деленная на количество тактов, через которые таймер посылает прерывания. Или FMAX= F_CLK/TIMER0_DEVIDER. Параметр F_CLK должен быть корректно указан в файле system/sysdef.h, а параметр TIMER0_DEVIDER – в файле system/res_timers.h.

Процедуры защищены от повторного вхождения, но для корректной работы системы все же лучше соразмерять время выполнения процедуры, вызываемой по таймеру с периодом ее вызова.

Приведем пример как создать эффект «бегущий огонь» на плате NedoPC-90.

Заходим в каталог upl.

Пишем файл main.c (или изменяем его):

```
//-------------------------------------------------------------------------
     #define LEDPORT PORTB /*светодиодный порт*/
     #define LEDDDR DDRB /*регистр управления этим портом*/
//-------------------------------------------------------------------------
     char sled = 0x01; // 1 в том бите, где зажжен светодиод
     // Процедура, которая будет вызываемой по таймеру с периодом 1сек
     void ttask_leds_shift()
     {sled = sled << 0x01; // Сдвигаем 1 влево на 1 разряд
     if(sled > 0x0F){sled=0x01;}  // Если 1 вышла за пределы младших 4х бит,
                                  // возвращаем ее на место
     output(LEDPORT,(input(LEDPORT) | 0x0F) & (~sled));_NOP();_NOP();
     // Изменяем состояния порта
     }
//-------------------------------------------------------------------------
     // Основная программа
     int main() {AUTOINIT_ALL_MODULES();
     // Инициализируем светодиодный порт
     output(LEDPORT,input(LEDPORT) | 0x0F);_NOP();_NOP();
     output(LEDDDR,input(LEDDDR) | 0x0F);_NOP();_NOP();
     //
     while(1){}
     //
     return(0);
     }
//-------------------------------------------------------------------------
Пишем файл main.h (или изменяем его):
//-------------------------------------------------------------------------
     // File: main.h
     // Contain headers for main.c
//-------------------------------------------------------------------------
     void ttask_leds_shift();
//-------------------------------------------------------------------------
```

Заметьте, что имя процедуры, вызываемой по прерыванию, ОБЯЗАТЕЛЬНО должно быть указано в заголовочном файле . Иначе – ошибка компиляции. Пишем файл rtl_mac.h (или изменяем его):

```
//-----------------------------------------------------------------------
// Макросы вызова процедур по таймеру уровня UPL
//-----------------------------------------------------------------------
// 100ms
//#define UPL_TASK_100MS_0()
//#define UPL_TASK_100MS_1()
//#define UPL_TASK_100MS_2()
//#define UPL_TASK_100MS_3()
//-----------------------------------------------------------------------
// 1s
#define UPL_TASK_1S_0() ttask_leds_shift()
//#define UPL_TASK_1S_1()
//#define UPL_TASK_1S_2()
//#define UPL_TASK_1S_3()
//-----------------------------------------------------------------------
// 1minute
//#define UPL_TASK_1M_0()
//#define UPL_TASK_1M_1()
//#define UPL_TASK_1M_2()
//#define UPL_TASK_1M_3()
//-----------------------------------------------------------------------
// 1hour
//#define UPL_TASK_1H_0()
//#define UPL_TASK_1H_1()
//#define UPL_TASK_1H_2()
//#define UPL_TASK_1H_3()
//-----------------------------------------------------------------------
```

Можете компилировать вашу новую систему и наслаждаться тем, как красиво бегает огонек.

Модификация NedoPC-90.AVROS

В заключение хотелось бы остановиться на вопросах переработки ОС под другие типы микроконтроллеров с ядром AVR и другие модули.

В начале Makefile есть строчка, определяющая тип микроконтроллера, для которого собирается ОС. В нашем случае (для модуля NedoPC-90) она имеет вид:

```
AVR-MCU= at90s8535
```

Для того, чтобы ОС компилировалась для микроконтроллера другого типа, достаточно изменить эту строчку. Например, если мы используем в модуле вместо микроконтроллера at90s8535 контроллер atmega8535 (который поконтактно совместим с at90s8535, но имеет большие объемы ПЗУ и ОЗУ, а также увеличенное число функций), то нам достаточно заменить строчку AVR-MCU= at90s8535 на AVR-MCU= atmega8535.

Если же мы хотим, чтобы наша ОС работала с периферией, отличной от той, что имеет на борту модуль NedoPC-90, то все несколько сложнее. Надо написать модули, которые работают с данной периферией (как было описано в разделе «Правила написания модулей непользовательских уровней»). Так же надо настроить существующие модули, если периферия, которую они обслуживают подключена к иным «ногам» микроконтроллера, чем в модуле NedoPC-90. Для этого, как правило, надо отредактировать файл, описывающий модуль, обслуживающий нужную периферию, (res_<имя модуля>.h в каталоге system). ▲

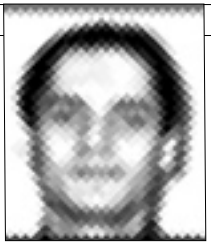
Вопросы и пожелания присылайте по электронной почте по адресу salex@cc.tpu.edu.ru.

(C) SfS, г. Томск, salex@cc.tpu.edu.ru

Статья предназначена специально для журнала NedoPC. Опубликование в других бумажных и электронных изданиях – только с разрешения автора статьи.

Заголовок исполняемого файла для «Спринтера»

автор Mac Buster



Решив попробовать свои силы в программировании для «Спринтера», вы можете воспользоваться ассемблером на самом «Спринтере» (например, OrgAsm), либо любым подходящим вам кросс-ассемблером. Спринтеровские ассемблеры умеют автоматически создавать исполняемые файлы. Однако, если вы выбрали второй вариант, то для создания исполняемого файла вам необходимо добавить к своей программе заголовок, содержащий в себе служебную информацию, указывающую операционной системе как и куда загружать исполняемый код. Вид заголовка показан на рисунке (префикс 0x означает запись в шестнадцатеричной системе счисления).

Обычно код программы размещают начиная с адреса 0x8100, и первая строка в заголовке указывает начальный адрес компиляции – 0x7F00, т.е. 0x8100–0x0200 (общая длина заголовка 512 байт). После этого следуют три латинские литеры «EXE», служащие для определения типа файла. За ними указывается номер версии исполняемого файла (сейчас используется нулевая версия). Далее располагается информация о смещении кода вашей программы от начала исполняемого файла, т.е. фактически указывается длина заголовка (512 байт). Если ваш ассемблер не поддерживает тип «двойное слово», можно заменить эту строку на:

```
Defw    0x0000,0x0200
```

Теперь надо указать длину начального загрузчика, если он есть (либо 0 если его нет). В том случае, если ваша программа разбита на независимые модули (или секции кода и данных), вы можете организовать программу таким образом, чтобы при запуске она самостоятельно принимала решение о том, что и как загружать в память и запускать, выделив часть кода в начальный загрузчик. Затем идут 6 байт, зарезервированных под дальнейшее развитие формата исполняемого файла. Обратите внимание, что для соблюдения совместимости с последующими форматами следует всегда заполнять зарезервированные байты нулевыми значениями. После зарезервированных байт указывается начальный адрес загрузки кода вашей программы. За ним указывается адрес ее запуска, т.е. значение, которое будет занесено в регистр PC. Сразу после него идет адрес вершины стека (для загрузки регистра SP), устанавливаемый при запуске вашей программы, которое не рекомендуется менять. И в конце заголовка идут 490 зарезервированных байт. Далее с метки START начинается код вашей программы.

```
      Org       0x7F00

      Defw      0x5845          ; EXE Signature
      Defb      0x45                 ; Reserved (EXE type)
      Defb      0x00                 ; Version of EXE file
      Defd      0x00000200      ; Code offset
      Defw      0x0000          ; Primary loader size or 0
      Defd      0x00000000      ; Reserved
      Defw      0x0000          ; Reserved
      Defw      START           ; Loading address
      Defw      START           ; Starting address (register PC)
      Defw      0x0BFFFh        ; Stack address (register SP)
      Defs      0x1EAh          ; Reserved 1

START     первая команда вашей программы
```

ПУТЬ ДОМОЙ

РАССКАЗ

автор Ivan Mak

Окончание.

Рядом появился Тайсио. Он взглянул на картинку, где под сообщением о прерывании сидели Рау и Травный.

— Ты еще жив? — удивленно спросил Дэн.

— Жив, а в чем дело?

— В том, что я дальше первого уровня не доходил. Заканчивалось все тем, что я оказывался в клетке, появлялся тот миу, которого поймали. Он сбегал и убивал.

— Вот он этот миу, — произнес Алекс. — И не убивает. — Добавил он. — Ты, наверно, не знаешь что такое мир.

— Мир? При чем здесь мир?

— При том, что миу разумны, и с ними надо дружить, а не воевать.

— Вот черт. И это в игре?!

Алекс взглянул на Дэна.

— Ты, кажется, хотел, чтобы я в чем-то разобрался? И, мне не кажется, что в сюжете игры.

— Хорошо. Забудем сюжет. Можешь сохранить ситуацию и выйти.

Алекс так и сделал. Экран погас, затем возникла заставка Spectrum-а...

— Ну и что скажешь? — спросил Дэн.

— Что? Про реализацию? Средне. Не плохой интеллект. Графика могла быть и лучше. Скорость средненького 3d акселератора.

— Звук?

— Тоже.

— Что тоже? На чем такой можно сделать?

— Звуковая карта. Плюс распознаватель речи.

Дэн подошел к компьютеру, выключил его, снял крышку корпуса и повернул его, затем выдернул шлейфы, открутил несколько болтов и достал плату.

— Бери и смотри, на чем все сделано.

Алекс взял плату, некоторое время рассматривал микросхемы. Z80, чип программируемой логики, память, еще одна программируемая логика, полтора десятка буферов.

— Ты хочешь сказать, что игра работала на этом? — Алекс даже не знал, как назвать плату.

— Именно.

— Не смеши меня.

— Господин Трипольский, вы не забыли, откуда вас вытащили? Не слишком ли круто делать подобное ради того, чтобы вас повеселить розыгрышем?

Официальный тон шефа вернул Алекса к реальности. Дело действительно серьезно.

— Может, у нее есть скрытая связь? — спросил Трипольский.

— Шутите? Какая связь? Мы под землей. Помещение экранировано.

Передача по сети питания, что ли? Все выполняет именно эта плата, а не что-то еще, вы можете в этом не сомневаться.

— Может, у него винчестер с хитрым чипом...

— Ерунда. Винчестер меняли. Блок питания меняли, так что связи не может быть через сеть. Меняли все, кроме платы и железяки.

— Дэн постучал по корпусу.

— А чипы проверяли? Может, они другие?

— Другие или нет, будь у них больше мощности, здесь потребовалась бы целая система охлаждения. Это стандартные чипы. И Zilog, и ALTERA, и память. Он работает. Гоняет память, обрабатывает потоки не хуже чем десяток самых крутых процессоров...

— Что? К-как это десяток? — Алекс даже поднялся со стула.

— Вот это и есть ваша работа. Вы же специалист по железу.

Вот и ответьте на вопрос: «КАК?»

— Не слабо, — произнес Алекс. Плата уже находилась в его руках, и он рассматривал микросхемы. — Надеюсь, у меня будет вся информация? — спросил он.

— У вас будет второй компьютер. Связь через наш сервер с мировой сетью. Не забудьте, что вас будут контролировать.

— Не забуду, — ответил Алекс.

Трипольский начинал с простого поиска данных на русскую фирму. Все оказалось проще некуда. Данные давно собрали, и Алексу оставалось лишь читать и перечитывать. Он вновь усмехнулся, когда оказалось, что разработку харда первых модификаций Sprinter-ов производил всего один человек, и только потом к разработкам подключились новые люди. Фирма почти не скрывала своих секретов. Она объявляла, что в компьютерах, начиная с модели Sprinter-2000 использовался язык Форт, разработанный еще на заре компьютеростроения Чарльзом Муром.

Алекс знал про Форт. Достаточно простой язык. Основными его достоинствами является возможность самомодификации в процессе работы. История языка в чем-то напоминает историю Spectrum-а. Появление, развитие, бум, спад, полклонники и почитатели. Странное сочетание. Может, в нем все и дело?

Алекс углубился в изучение. Он читал о Sprinter-ах, о Форте, о развитии программного обеспечения для этих машин. Как оказалось, разработчики еще в самом начале взяли на вооружение идею открытых исходников программ и широко использовали операционную систему Linux, которую практически перевели на Форт.

Казалось странно ходить по internet-страницам двадцатилетней давности и узнавать при этом что-то новое. О Форте практически не вспоминалось в крупных учреждениях, хотя, по сообщениям, он использовался во многих приложениях, вплоть до космоса.

Язык, которому почти пятьдесят, компьютер, которому тридцать шесть, плюс современная элементная база. В чем может быть дело?

В скорости? Да, в скорости. Алекс встал на эту идею и отправился в новый поиск.

«Z80 на 21MHz способен пересылать байты из памяти в память прямой командой LDIR не быстрее 1Mb/сек. Некоторыми программными ухищрениями эту скорость можно немного поднять, но не более чем в полтора раза» — прочитал Алекс ответ одного из специалистов. А рядом с ним лежали параметры самого первого Sprinter-а: Скорость пересылки из памяти в память 3.5Mb/сек, ограничена только пределом рабочей частоты SIMM-а. Почему? Ответ лежал рядом — использование акселератора в ПЛМ.

Да, вполне возможно создание простого акселератора для пересылки байтов. Элементарный прямой доступ к памяти — ПДП.

Алекс продолжал разбираться с первым компьютером. Немного хитрый экраный контроллер, графический акселератор, простейший звуковой синтезатор. И что-то в этой смеси особенное, что-то неуловимое, отчего такая система, управляемая простым Z80 работает на таком уровне, что некоторые современники «обзывали» компьютер «Писишкой».

Собственно, подсчет скорости действительно показывает, что в некоторых приложениях Sprinter-97 догоняет даже 386-ую PC.

Трипольский вновь и вновь вчитывался в данные машины. Что о ней писали тогда, в том числе и сам разработчик? Каковы достоинства?

Самое первое — возможность изменения конфигурации машины.

Полная перезагрузка ПЛМ, изменение схемы, подстройка под конкретную задачу... Да! Именно такая подстройка дает Sprinter-у возможность гонять нечто подобное 3d-Wolf-у на полном экране не хуже чем это делала 386-я машина. ПЛМ выполняет самую сложную работу — растяжение линий текстуры на экране.

Простейшая мысль. Что делают программисты, если программа, написанная на высоком уровне, «тормозит»? Программисты выискивают в ней самый часто выполняемый цикл и реализуют его на ассмеблере, чтобы он работал как можно быстрее. Результат — резкое повышение скорости. Что делать, если ассмеблер медленный? Например, как у Z80. Надо взять самый часто выполняемый цикл и реализовать его в железе! Да! Вот он, первый принцип ускорения для «бегуна на короткие дистанции»!

Z80 не выполняет растяжение текстуры. Он только управляет процессом. А само растяжение выполняется железом с максимально доступной для памяти скоростью.

Первый камень в понимании вложен. Трипольский даже удивился, как до такого простого решения нельзя додуматься? Впрочем, додумались. Но не кто-то, а «товарищи».

Sprinter-2000. Алекс пронесся по его описанию и понял еще одну вещь. Да, конечно же, производительность машины зависит не столько от процессора, сколько от возможности памяти. От ее пропускной способности. Объем памяти повысился. Разрядность увеличилась в два раза. Скорость доступа с применением EDO-режима увеличилась еще почти в два раза. Плюс более объемная ПЛМ. Так ли важен ее объем? Разумеется! Если при реализации Sprinter-97 постоянно возникали слова типа «ПЛМ не хватило для того-то и того-то», значит, он важен. И очень важен! А в 2000-м объем ПЛМ увеличился в 3 раза. Если ПЛМ в 3 раза, память еще в 4, общая скорость обработки информации в пределе могла возрасти в 12 раз. Вот он тот самый порядок прироста скорости. А Z80? А что Z80?

Трипольский усмехнулся. Z80 остался только управлять процессом. Он лишь руководитель, а исполнитель — ПЛМ. Много ли надо, чтобы управлять? Не мало, но и не так много. 21 мегагерц хватает.

И нужна ли разрядность для управления? Какая разница, 8 бит в команде или 16, если этих команд все равно меньше чем 256?

На волне эйфории Алекс добрался до компьютера Sprinter-II.

Более высокоскоростная память, большая ПЛМ, и все тот же Z80. Нет проблем! Все ясно, как на ладони. Управление в руках Z80, а ПЛМ только успевает выполнять инструкции.

И все же, что-то в этом не так. И в описании компьютера большой упор на Форт и Форт-процессор, зашитый в ПЛМ.

Процессор в ПЛМ? Много ли в нее можно записать? И что значит Форт-процессор? В него вшит язык высокого уровня?

Да и только да. Алекс видел, что ответ таков, но как его понять? Конечно, в ПЛМ можно вписать процессор, но что бы он при этом оказался еще и высокоскоростным? 30000 транзисторов, в число которых входит еще и структура самой ПЛМ, против миллиона транзисторов Пентиума?

Нет. Ошибка. ПЛМ крупнее, число транзисторов побольше. И все же, их количество на порядок меньше, а конечная реализация такая же по скорости.

Дело может быть только в принципе. А этот принцип можно основать лишь на одном факте. Структура Пентиума — существенно избыточна.

Да!

«Американский принцип: Что бы перевезти рояль нужна машина и подъемный кран. Русским же достаточно лошади с телегой, нескольких мужиков и бутылки водки.» От этой мысли Алекс усмехнулся.

Мысль даже не его собственная. В былые времена так говаривал отец.

Избыточность Пентиума. А значит меньшими средствами можно добиться большего. Остается лишь понять как?

Из неясного в прочитанном остался только Форт-процессор.

Алекс обратился на страницы поиска, и уже через минуту начал чтение.

Форт-процессорами занимался даже Чарльз Мур. Их реализации использовались в самых разных приложениях. Не мало фирм производили чипы, но ни один из них так и не получил широкого распространения.

Intel продолжала выпуск своих процессоров. Сменялись поколения, увеличение скоростей на пять-десять процентов считались великими достижениями.

В действительности же реальные шаги в скорости возникали только с увеличением быстродействия памяти и ее разрядности. Процессоры давно обогнали память: как можно работать на 1000 мегагерц, если скорость памяти всего 100? Ухищрения и еще раз ухищрения.

Но, как говорится, выше головы не прыгнешь. Если память работает на 100, из нее не вытащишь данные со скоростью 110. А дальше только обработка. Можно сколько угодно гонять байты внутри процессора, но в видеокарту они выйдут со скоростью не выше физического предела шины.

Вот и возникает вопрос. А нужно ли толочь воду в ступе?

Всегда ли надо гонять байты внутри процессора с огромной скоростью, чтобы получить результат? Ответ прост — нет, не всегда и далеко не всегда.

Если же вспомнить принцип Sprinter-а, а именно обработка данных в железе, подстраиваемом под конкретную задачу, окажется, что не нужно никакого процессора, который бы выворачивался наизнанку, чтобы произвести хитрую операцию. Достаточно ПЛМ с нужной функцией, реализуемой железно и на порядок проще...

Вот он передовой край. Вот! Алекс смеялся над своими же мыслями, возникавшими еще неделю назад.

Трипольский сидел за компьютером и продолжал игру.

Смех! Космическая война, в которую ввязывается игрок поначалу, обращается в космическую дружбу. Цель — не война. Цель — мир!

Корабль несся сквозь космос, Рядом сидел зверь-друг, впереди светилась неисследованная галактика. Экран встал и погас. Из динамиков донесся новый голос.

— Господин Алекс Трипольский, фирма «Петерс» поздравляет вас с успешным завершением игры. На данный момент ваш показатель прохождения наивысший по сравнению с другими игроками данной системы.

— Можно вопрос? — спросил Алекс. Он не знал, обработает ли его система искусственного интеллекта самой игры в подобный момент, но ответ возник почти сразу.

— Да.

— Как называется эта система?

— «Sprinter-III» или «Sprinter-2015».

Голос стих. Через несколько мгновений машина вышла в меню Spectruma.

Дверь открылась. Дэн Тайсио молча вошел в помещение, сел в кресло.

— Я решил отказаться от контракта, — сказал Алекс.

— Что? Как это отказаться?! — воскликнул Дэн, вскакивая.

— «Умом Россию не понять» — произнес Алекс. — В контракте есть пункт, где я имею право отказаться, не так ли?

— Да, но в этом случае, вы вернетесь в тюрьму.

— Без проблем...

Тайсио замолк. Он смотрел на Алекса, а тот не выдержав заулыбался.

Все переменилось! Все миропонимание! Алекс Трипольский ощутил себя РУССКИМ.

— Что?! Ты что нибудь понял?!

— Да, — ответил Алекс. — И игру я прошел до конца. Кстати, сколько стоит «Sprinter-III»?

Дэн пропустил этот вопрос.

— Вы должны рассказать все что узнали!

— Извините, сэр, но я узнал не больше чем есть в вашем компьютере. Ваша задача неразрешима.

— Почему?

— Ответьте на простой вопрос. Что нужно, чтобы перевезти рояль?

— Грузовик и какое-нибудь подъемное устройство.

— Вот поэтому, — произнес Алекс вставая.

— Вы куда?

— К генералу.

Трипольский едва скрывал улыбку перед начальником тюрьмы.

Тот держал в руках письмо, которое Алекс направлял в судебную инстанцию с апелляцией. По правилам начальник должен его проверить.

Алекс этого не боялся. Документ не нарушал правила.

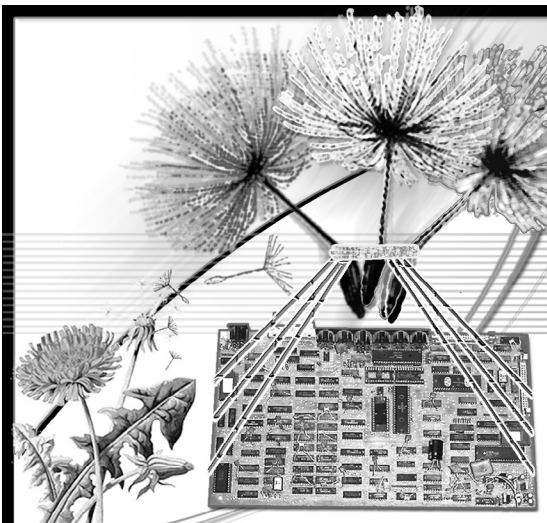
Состоялся новый суд. Не малую роль играл факт освобождения со стороны военных и контракт, который Алекс подписал, а затем расторг. Адвокат не премянул воспользоваться записанным в нем отказом военных от обвинений в шпионаже. Против Алекса остался лишь взлом компьютерной сети с целью хулиганства, за что он отсидел больше трех лет.

Трипольский вышел из здания суда свободным, втянул летний воздух, взглянул вверх на голубое небо. Он улыбался, потому что ЗНАЛ что делать. Ноги сами привели его к российскому представительству.

Алекс добился встречи с консулом и сказал человеку просто по-русски:

— Хочу в Россию. ▲

13-ноябрь-2000



АТМ Турбо от NedoPC

Позиция	Цена	Примечания
Плата голая Комплектность поставки: - плата; - ХЛ8 (прошитая); - описание (книжки); - софт (имиджи на сдроме).	700	Платы при изготовлении электрически не проверяются
Плата собранная Комплектность поставки: - отлаженная плата с 1556ХЛ8, ПЗУ, Z80, 1818ВГ93, панелька под AY; - описание (книжки); - софт (имиджи на сдроме)	2950	Платы будут собраны и отлажены, т.е гарантировано рабочие. Музыкальный сопроцессор не поставляется
Конструктор для самостоятельной сборки: Комплектность поставки: - плата; - набор радиодеталей, необходимых для сборки; - необходимые ПЗУ (прошитые), ХЛ8; - описание (книжки); - софт (имиджи на сдроме).	2300	Процессор, ОЗУ и ПЗУ проверяются перед отправкой. Поставщик не несет ответственности за сгоревшие в результате неправильной сборки или подключения детали
Комплект шлейфов: - шлейф FDD - шлейф IDE - шлейф COM - переходник AT-питание->5-DIN - шлейф для звука	200	Комплект облегчает установку в AT корпус
Музыкальный сопроцессор (YM2149)	160	
Прошивка ПЗУ основная (27С512 или аналог)	65	На текущий момент версия 1.07.13
Прошивка тест ОЗУ (27С512 или аналог)	65	На текущий момент версия 1.02
Прошивка знакогенератора (573РФ2 или аналог)	35	Символьная таблица знакогенератора
Прошивка контроллера клавиатуры (573РФ2 или аналог)	35	XT или AT клавиатура
Процессор КР1858ВМ3	40	Аналог Z80 на 6МГц
Процессор Z0840008	60	Z80 на 8МГц
Дополнительная, прошитая КР1556ХЛ8	40	Прошивка АТМ Турбо 2+

К общей стоимости заказа добавляется цена доставки (по России):

100 руб. для голой платы

150 руб. для платы в сборе

Порядок обработки заказов на платы:

1. Присылаем e-mail Чунину Роману Валерьевичу (chunin@mail.ru) или звоним по телефону +7(095)6547433, для выяснения есть ли свободные платы и согласования цены, комплектации;
2. Если не получен почтовый перевод в течении двух недель, то заявка снимается;
3. Дополнительно к стоимости добавляется стоимость услуг почты: 100 руб. для голой платы, 150 руб. для собранной;
4. Осуществляем почтовый перевод на адрес: 109451, Москва, ул.Братиславская, д.13, кор.1, кв.228, Чунину Роману Валерьевичу. Пожалуйста, указывайте обратный адрес (если через e-mail, то с указанием номера и даты почтового перевода);
5. После обработки и подготовки заказ отсылается по указанному вами адресу. Голые платы отсылаются в течении одной недели после оплаты, собранные в порядке очереди на сборку (на сборку одной платы уходит примерно неделя, собирать будут два человека параллельно);
6. Весь процесс оформления заказов будет отображаться на сайте http://nedopc.com/products.php

ОБЪЯВЛЕНИЯ

Просьба к читателям: сообщать об ошибках и опечатках, найденных на страницах издания по адресу nedopc@mail.ru

---

Ищу советские книги и журналы по вычислительной технике за период 1940...1991 годы. Желающие поделиться пишите по адресу mbr@nm.ru

---

Приму в подарок старое и нерабочее железо (матери/звук/видео и прочее) на утилизацию (т.е. на радиодетали и разъемы). Желательно из Москвы. Тел. +7 (910) 4552728. Роман

ЗДЕСЬ МОГЛА БЫ БЫТЬ ВАША РЕКЛАМА

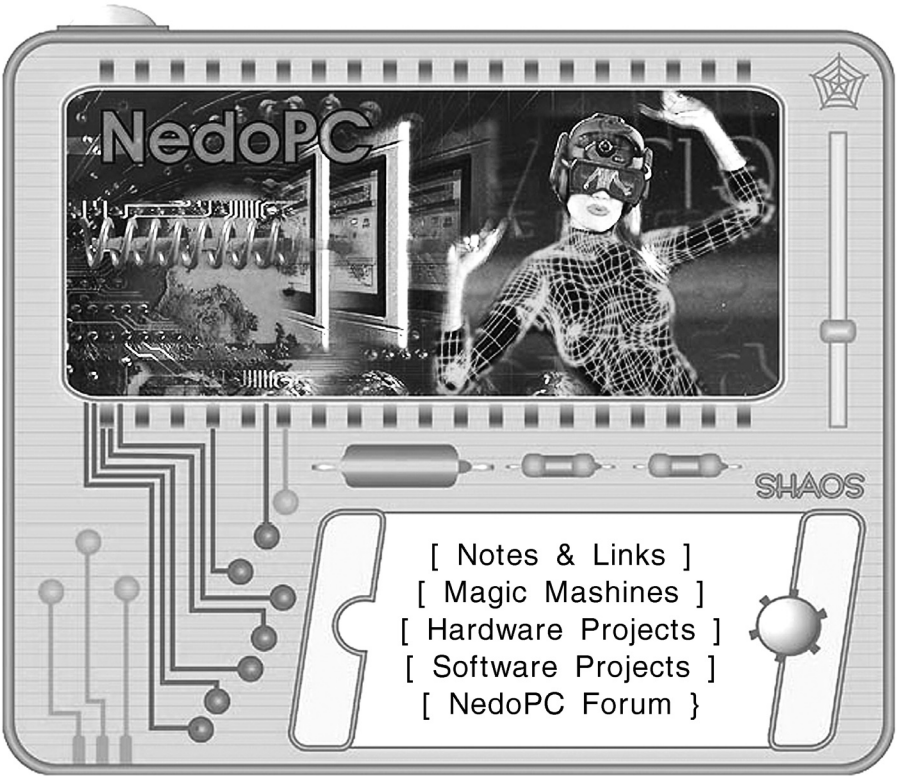
NedoPC is not PC

NedoPC

Бланк Объявления
в журнал "NedoPC"

Бланк для размещения объявлений рассчитан на тех, кто предпочитает пользоваться обычной почтой.
Адрес: 109451, Москва, ул.Братиславская, д.13, кор.1, кв.228, Чунину Роману Валерьевичу
Для тех же, кто желает отправить текст объявления через e-mail, пишите на адрес nedopc@mail.ru
Текст объявления должен содержать не больше 256 символов

WWW.NEDOPC.ORG



Copyright (c) 2002-2005 NedoPC team

ВСЕГДА В ON-LINE!