



embedded-projects.net

JOURNAL

OPEN SOURCE SOFT-AND HARDWARE PROJECTS

[EDITORIAL]

PROJECT START

Nr.1 - start embedded-projects-journal



[PROJECTS]

- **USBprog Programming Tool**
- **OpenOCD - On Chip Debugger**
- **ARM Getting Started: Using USBprog and OpenOCD**
- **Keep it Simple Stupid -TFT am Grasshopper (AVR32)**
- **Compact graphics framework für LCDs (cmg)**
- **XSVF Player with USBprog**

[WISSEN]

USB Grundkurs - USB verstehen und anwenden

[WETTBEWERB]

ARTICLE CONTEST with www.mikrocontroller.net



embedded-projects.net

OPEN SOURCE HARDWARE PROJEKTE



**Von Entwicklern -
für Entwickler**

Jetzt schneller zum Shop:
www.eproo.net

embedded-projects.net

AVR32 Board Grasshopper

- 140 MHz max. 200 MHz
- 64 MB SDRAM
- 8 MB Flash
- 10/100 MBit/s Netzwerk
- 1 USB Highspeed Device Anschluss
- 8 LEDs
- 1 Taster
- Spannungsversorgung: 5-10V
- über die Pinleisten sind alle wichtigen Ports herausgeführt

Mit GNU / Linux Kernel
+ C-Beispielprojekt

ab 85,00 € *inkl. gesetzl. MwSt.

<http://www.embedded-projects.net/grasshopper>

[EDITORIAL] Benedikt Sauter

WELCOME

Erstausgabe von Embedded-Projects-Journal

Das Journal

Seit ca. 20 Jahren gibt es in der Programmierszene eine Provinz in Gallien ... Äh eine Open-Source-Community, die sich scheinbar gegen alles kommerzielle und proprietäre sträubt und wehrt, und sich quasi gegen den Rest der Welt stellt. Begonnen hat damals wie heute alles mit dem Traum, frei und unabhängig zu sein.

Das Resultat gegenwärtiger Freiheit der Open-Source-Community ist GNU/Linux und all die anderen offenen Betriebssysteme samt einem riesigen freiem Softwarearchiv. Das Konzept der Open-Source-Community wurde bereits auch in andere Gebiete übernommen. So gibt es beispielsweise freie Bücher [1], Musikdownloadportale [2] und viele andere Ideen.

Mit dem Projekt Embedded Projects Journal ist eine weitere Idee verwirklicht worden. Ein freies Magazin für Mikrocontrollerprojekte, die vollständig unter freien Lizenzen veröffentlicht worden sind, oder für andere interessante Artikel rund um dieses Thema, die natürlich ebenfalls unter freien Lizenzen stehen.

Für mich ist das Beste an Open Source Projekten, dass kein einziger Anwender auf Grund fehlender finanzieller Mittel ausgeschlossen wird. Alle Quelltexte können kostenlos aus dem Internet heruntergeladen und am besten mit freien Tools be- und verarbeitet werden.

Die Zeitschrift wird es deshalb zusätzlich zur digitalen Ausgabe auch in gedruckter Form kostenlos - über das Internet anforderbar - geben. Die entstehenden Kosten für Druck und Porto sollen soweit als möglich von Sponsoren getragen werden. Mein Wunsch ist es, dass wir es schaffen das Heft lange am Leben zu erhalten. In diesem Sinne viel Spass beim Lesen der ersten Ausgabe.

An dieser Stelle möchte ich mich bei den Personen bedanken, ohne die dieses Heft nie möglich gewesen wäre.

Und so wird es hoffentlich wie nach jeder erfolgreichen Schlacht mit den Römern in Gallien ein Festmahl geben, bei dem sich alle freuen und jubeln. In diesem Sinne wünsche ich viel Spass beim Lesen und freue mich über konstruktive Kritik in allen Bereichen.

Open-Source heisst nicht nur kostenlos nehmen - sondern aktiv mitgestalten.

Benedikt Sauter
sauter@embedded-projects.net

[1] <http://www.gutenberg.org>

[2] <http://www.jamendo.com/en/>

Contact:

Embedded Projects:
Dipl.-Inf. (FH) Benedikt Sauter
Kettengässchen 6
D-86152 Augsburg

Fon: +49 (0) 821 - 50 81 581
Fax: +49 (0) 821 - 50 81 921
Mail: journal@embedded-projects.net

Anzeigemöglichkeiten & Preisliste
auf Anfrage via Mail

Journal:

Herausgeber: Benedikt Sauter
Layout/Satz: Das-Medienkollektiv.de
Veröffentlichung: geplant 4x / Jahr
Ausgabeformate: PDF / Print
Auflage Print: 2000 Stk.



Alle Artikel in diesem Journal stehen unter der freien Creative Commons Lizenz. Die Texte dürfen - wie bekannt von Open-Source - modifiziert und in die eigene Arbeit mit aufgenommen werden. Die einzige Bedingung ist, dass der neue Text ebenfalls wieder unter der gleichen Lizenz, unter der dieses Heft steht, veröffentlicht werden muss, und zusätzlich auf den originalen Autor verwiesen werden muss.



Except where otherwise noted,
this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

USBprog Programming Tool

Bernhard Walle <bernhard.walle@gmx.de>

1. Introduction

1.1. Why a new Programming Tool

The existing tool was very simple and didn't work at the author's computer out of the box. In fact, error handling was non-existent. After looking into the code and fixing some bugs, I contacted the original author for inclusion of the patches.

After a short discussion, we decided that it's better to write a completely new tool with a library so that the GUI frontend and

the command line frontend use the same backend functions when communicating with the device.

1.2. Features

The requirements and therefore the key features of the new tool were:

- same functionality of GUI and command line version,
- automatic download of firmware files from the Internet,
- offline usage (for systems without a permanent Internet connection),
- only "common" dependencies for easy installation,
- support of major Operating Systems including Linux, Windows and MacOS.

2. Design

2.1. Components of USBprog

As mentioned above, the USBprog toolset consists of three components (see also figure "USBprog components and dependencies" below):

1. a library called `libusbprog` that takes over
 - handling the firmware pool (see later),
 - managing the USB devices and
 - upload the firmware to the USBprog device,
2. a command line interface (CLI) called `usbprog` that can both be used
 - interactively, in a shell-like manner and
 - non-interactively for scripting (e.g. in Makefiles),
3. a graphical user interface (GUI) called `usbprog-gui` that can be used instead of the CLI for users that feel more comfortable with GUIs.

2.1.1. The Library

Firmware Pool

One key feature of both the GUI and the CLI is that it can handle local firmware files (which is important for firmware development) and the online firmware pool. Handling local firmware files is trivial, and therefore no description is necessary.

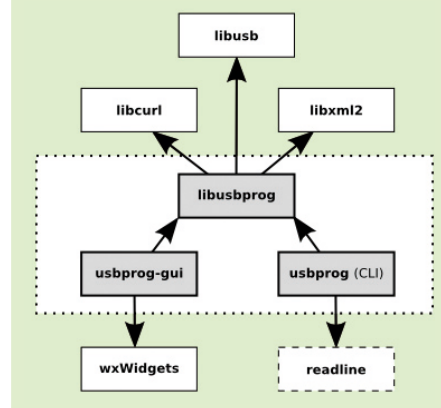
However, for the online firmware pool, USBprog uses a local firmware cache. On each startup it reads a (relatively small) XML file from the USBprog site that contains information about all available firmwares. The user can now download firmware files from that pool and use that firmware on the USBprog device. The important part is that the firmware is stored on the disk, so each

version is downloaded only once.

This saves download time and enables the user to use that firmware also when the Internet connection doesn't work, which is important for Laptops that have no permanent Internet connection. The integrity of the cache is ensured both with versioning and MD5 checksums.

Since both the CLI and GUI contain all functions to manage that firmware pool (including cleanup), the user doesn't have to know where and in which format the firmware is stored. However, for curious users: it's `$HOME/.usbprog/` on Unix and `%APPDATA%\usbprog [%APPDATA% expands to c:\Documents and Settings\\Application Data\ on English Windows versions, but you can just enter %APPDATA% in the Windows Explorer.]` on Windows.

USBprog components and dependencies



To parse the XML files, `libxml2` is used, and for downloading all the files, the widespread `curl` library is utilised.

Device Manager

Another important goal was to design the tool in a way that supports multiple USBprog devices on one machine. Since USBprog has a very broad scope, this could become a quite common approach - for example one might have one USBprog as AVR programmer and another as JTAG interface for debugging.

To support future firmware versions, the number of devices recognised as "USBprog" is not hard-coded but the USB IDs are read from the XML file. Only devices which are already in update mode are automatically chosen as update device, all others have to be selected manually. Currently, it's not possible for USBprog to discriminate whether the device is a emulated "foreign" device (such as the AVR ISPMKII), if it's USBprog or the original device [However, the case might be rare since it's unlikely that a user both has an original AVR ISPMKII and an emulated one connected at the same time.].

Communicating with the Device

The low-level communication with the USBprog device is done via the `libusb` library. This library is available for all Unix-like operating systems (including MacOS) that support USB and with a special Win32 port also for Windows. Using a userspace library has lots of advantages over a kernel driver. It simplifies

Deployment

On Linux, kernel drivers have to be re-compiled for (nearly) every kernel release.

Development

Debugging and developing a userspace program is much easier.



Portability

With libusb, the same API can be used across various operating systems.

Since speed doesn't matter for USBprog (the amount of transferred data is small) and USBprog doesn't need any interfaces of the operating system (such as character devices, Video for Linux etc.), using libusb makes much sense.

While this works flawlessly on Unix, handling Windows is a bit more problematic since it's necessary to install some sort of "glue driver" for every USB device ID the program (or libusb in general) should support. However, it's still much easier than Windows kernel programming and the USBprog installer comes with everything the user needs.

2.1.2. The Command Line Tool

Implementing the CLI was relatively straightforward since almost all needed

functionality is in the library. The idea to have one CLI tool that supports both, interactive usage and non-interactive usage was "stolen" from avrdude. However, its implementation is different: While avrdude has a different syntax for interactive and non-interactive usage, USBprog uses the same syntax.

The key is to have commands with a fixed number of arguments. That way, no delimiter is necessary and all commands including their arguments can be passed to *usbprog* as arguments. For example: If *usbprog* should execute `c1 a1` and `c2 a2 a3`, then it can be called as *usbprog* `c1 a1 c2 a2 a3`.

This concept simplifies usage, documentation and — of course — implementation.

2.1.3. The GUI

Especially Windows users tend to like GUIs more than command line interfaces. The

USBprog GUI was written using the wxWidgets library because it is available on both Unix and Windows with a "native" look and feel (which GTK is not on Windows), it's well supported by the DevCpp IDE which is used to compile USBprog on Windows and because the "old" GUI was written using wxWidgets and so re-using a bit of the old code was easier.

2.2. Implementation Notes

Until that paragraph, the reader doesn't know in which programming language all the stuff was written in - it's C++. The author prefers C++ over C for application programming (especially for GUIs). A scripting language was not used because deployment is more difficult, especially on the Windows side: While a native binary only needs a few DLLs that can be put together in an installer, a Python program needs an interpreter which is likely to be not available on Windows.

3. Usage

The usage of USBprog should be relatively straightforward: After installation, the user can call *usbprog* or *usbprog-gui* or click on the desktop icons. While the GUI should be usable without any

documentation (the figure "Screenshot of the GUI" should give an impression), a few words describe the command line interface in the next section.

3.1. Command Line Interface

The most important command is *help* which lists all available commands. The second most important command is *? command* which gives help to a specific command, including the expected arguments. There's also a manual page (on Unix) which describes all commands and options.

A typical session includes:

1. displaying all devices with *devices*,

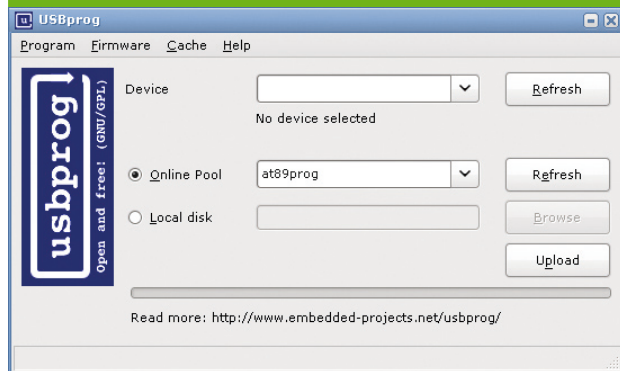
2. setting the update device with *device number*,
3. listing all firmwares with *firmwares*,
4. downloading the firmware from the internet with *download firmware* and
5. upload the firmware to the device with *upload firmware*.

If the user needs information about a firmware, the *info firmware* command helps, and even the pin/LED/jumper assignment is available via *pins firmware*.

3.2. Update Mode

While the USBprog program can put devices into update mode, this leads to problems on Windows: The driver manager seems to be confused if a device changes its USB IDs at runtime too often. Therefore, it's better if the user switches to update mode before plugging the in the device. The procedure is described at the USBprog website.

Screenshot of the GUI



4. Availability

Since the installation of USBprog should be as easy as possible, one important goal is to provide binaries for all widespread Operating Systems and distributions. At the main site, we provide a Windows installer and a source code archive.

The author maintains packages for openSUSE (starting with 10.1), SUSE Linux Enterprise 10 and Fedora (starting with version 7) at the openSUSE BuildService. The search interface can be used to find the software. On openSUSE, the "One Click Install" system simplifies installation.

Debian packages will be available soon directly in the Debian distribution.

If you're package maintainer in another Linux distribution and would like to maintain USBprog for your distribution, please contact us in the USBprog forum or via e-mail. The same is true if you're an advanced MacOS, *BSD or (open)Solaris user and are able to provide binary packages for your favourite operating system!

5. Summary

The new USBprog flash tool unifies the old command line and GUI flash tool with a library that is shared between both programmers. The command line interface is both usable interactively and non-interactively. Both tools can be used offline without internet connection by using a firmware cache and also support multiple devices. The tool is available for all major operating systems.

OpenOCD - On Chip Debugger

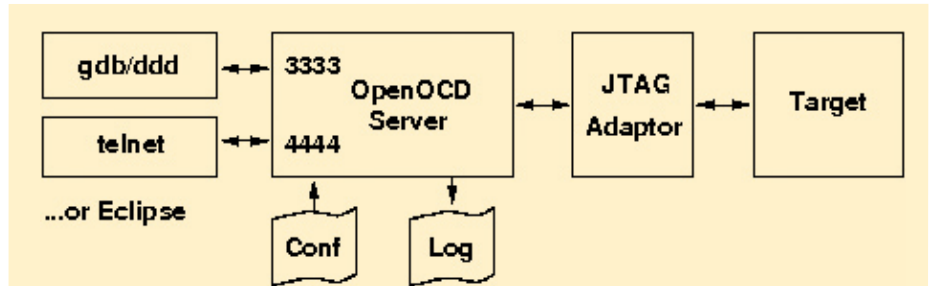
Hubert Hoegl <Hubert.Hoegl@hs-augsburg.de>

1. Introduction

OpenOCD is a free open-source JTAG debugger for microprocessors with ARM7, ARM9 and related cores written by Dominic Rath. Some of the microcontrollers known to work with OpenOCD are ADuC7xxx, AT91R40008, AT91SAM7, AT91SAM9, AT91RM9200, EP93xx, LM3S811, LPC21xx, LPC22xx, LPC3xxx, STR7xx, STR9xx, STM32, XScale and many others using the ARM architecture.

OpenOCD is an ideal complement for the GNU GCC toolchain for ARM processors. It can be controlled from the GNU debugger GDB and from a Telnet commandline interface. OpenOCD is capable of driving a variety of open homebuilt or commercial JTAG hardware interfaces and can easily be adapted to new interfaces. It is licensed under the General Public License (GPL).

The following figure shows how OpenOCD fits in the overall picture. OpenOCD provides network connections for the GNU Debugger gdb and for an optional Telnet client. You can either run GDB and Telnet on the same



machine on which OpenOCD runs or on different machines according to the networking nature of both protocols.

OpenOCD talks to the target board by a JTAG interface. A number of open and commercial interfaces exist for plugging into the USB or Parallel Port and it is easy to add new interfaces. Most of the available interfaces are listed in section 5.

On startup OpenOCD reads one or more configuration files containing target specific configuration information. While the program is running it writes log messages to a logfile.

OpenOCD is available for Linux and Windows. This text will focus on Linux. Windows users should get the Yagarto distribution [3] containing a comprehensive and easy to install set of tools including OpenOCD and Eclipse.

2. Getting OpenOCD

OpenOCD is contained in the package pool of the major Linux distributions. For example the Debian package is available here: <http://packages.debian.org/unstable/embedded/openocd>

The binary packages are very easy to install, on Debian e.g. with `apt-get install openocd`. However the version of these packages is sometimes a bit outdated. If you want the current release, you can easily check it out from the code repository with the Subversion command

```
svn co http://svn.berlios.de/svn-root/repos/openocd
```

To compile the software follow the build instructions in file `openocd/trunk/INSTALL`.

3. How to work with OpenOCD

All files needed to follow this short demo are available at [7]. I use an Olimex SAM7-P64 board („target“) and a USB JTAG adaptor. The basic work with OpenOCD is very simple:

1. Connect a JTAG cable between the PC and the target 20-pin JTAG header. Easiest is an USB to JTAG cable, see section 5.
2. Optionally connect a RS-232 cable between the PC serial port and the target. Note that some JTAG adaptors also have a separate UART or RS-232 channel.
3. Connect target to power (may also work over USB).

4. Start OpenOCD

```
sudo openocd -f sam7s-ooocdlink.cfg
```

The configuration file `oocd-demo.cfg` contains some settings which OpenOCD needs to correctly start, the most important are

- JTAG adaptor type and speed
- Port numbers for telnet and gdb services
- Details of the reset signals for ARM controller and JTAG port
- Details of the JTAG chain
- Startup behaviour of the OpenOCD server

5. Start a user interface to interact with the target. In the simplest case this will be a telnet connection established with

```
telnet localhost 4444
```

The number 4444 is the port number of the telnet server built into OpenOCD. After telnet is running, about 90 commands can be entered from the telnet commandline (enter `help` at the telnet prompt).

+ From the telnet user interface one can do many tasks, e.g. download the program to RAM or Flash, run it, stop it and display registers. However, if it comes to source



level debugging, you will need to run a real debugger:

```
arm-none-eabi-gdb -x init.gdb
```

The gdb configuration file `init.gdb` contains at least the port number of the GDB server, which is a part of the OpenOCD server and the name of the program which shall be loaded in the target.

Gdb provides only a spartanic commandline user interface (though it is quite effective to use). Some more comfort provides the *Data Display Debugger* graphical front end to gdb.

```
ddd --debugger arm-elf-gdb --
command=init.gdb main.elf
```

DDD's home is at <http://www.gnu.org/software/ddd>

```
cgdb -d arm-none-eabi-gdb --
command=init.gdb
```

Another nice front end is `cgdb`, which has a character-based GUI (<http://cgdb.sourceforge.net>). Run it with

6. Start a terminal program
User interaction with the target board can easiest be achieved by using a serial

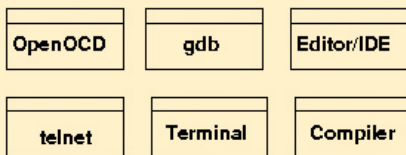
connection („UART“, „RS-232“). On the development computer you need a *terminal* program to communicate over a serial line. One very easy to use terminal program is `picocom`. Start it with

```
sudo picocom -b 9600 /dev/ttyUSB1
```

The `-b` option specifies the baud rate and the `/dev/ttyUSB1` argument is the serial port device. In this case my USB JTAG adaptor has beside the JTAG port also a RS-232 interface — I highly recommend this! Some other often used terminal programs are `kermit`, `cutecom` and `minicom`.

4. Screen

Usually you will have multiple windows open on your embedded development PC running the tools shown in the following figure:



How the various windows are arranged and managed is left to the preferences of the user. I recommend to use the `screen` utility for this purpose. The following text describes how to use it.

The `screen` utility is capable to run multiple applications in one console window. Here is a part of the screen configuration file which starts all the necessary applications:

screen configuration

```
screen -t shell 1 bash
screen -t emacs 2 emacs -nw --load emacs.el
screen -t openocd 3 bash --rcfile startocd.sh
screen -t gdb 4 bash --rcfile startgdb.sh
screen -t telnet 5 bash --rcfile starttelnet.sh
screen -t uart 6 bash --rcfile startpicocomm.sh
```

The following figure shows a screen-controlled console window. Look at the status line (bottom) showing all active tools:

The `screen` homepage is at <http://www.gnu.org/software/screen/screen.html>.

5. Configuration files

A sample configuration file for OpenOCD

```
# sam7s-ocdlink.cfg
# Openocd configuration for OpenOCD-USB connected to an AT91SAM7S64
# 2008-04-26 <Hubert.Hoegl@hs-augsburg.de>

#daemon configuration
telnet_port 4444
gdb_port 3333

#interface
interface ft232
ft232_device_desc „Dual RS232“
ft232_layout ocdlink
ft232_vid_pid 0x0403 0x6010
jtag_speed 4
reset_config trst_and_srst

#jtag scan chain
# format L IRC IRCM IDCODE (Length, IR Capture, IR Capture Mask, IDCODE)
jtag_device 4 0x1 0xf 0xe

#target configuration
daemon_startup reset
#target arm7tdmi <endian> <reset mode> <chainpos> <variant>
target arm7tdmi little run_and_halt 0 arm7tdmi_r4
run_and_halt_time 0 30
```

Example:

A sample GDB command file `init.gdb`

```
### -init.gdb-
target remote localhost:3333
monitor arm7_9 force_hw_bkpts enable
add-symbol-file main.elf 0x4000000
load main.elf
```


6. JTAG Adaptors

Table: JTAG Adaptors

JTAG Adaptor	Manufacturer	Interface	UART	ca. Price (Euro)	Business Model
O OCD-Link	[a]	USB (FT2232)	yes	-	Open
USB JTAG	[b]	USB (FT2232)	yes	-	Open
USBprog	[f]	USB (USBN9604)	no	30	Open
JTGkey	Amontec [d]	USB (FT2232)	no	129	Closed (Commercial)
TAGkey-Tiny	Amontec	USB (FT2232)	no	29	Closed (Commercial)
Chameleon	Amontec	Parallelport	no	119	Closed (Commercial)
ARM-USB-OCD	Olimex [e]	USB (FT2232)	yes	55	Closed (Commercial)
ARM-USB-TINY	Olimex [e]	USB (FT2232)	no	40	Closed (Commercial)
Wiggler	[c]	Parallelport	no	10	Open

[a] Will soon be available at <http://www.embedded-projects.de>.

[b] My attempts to build a USB-to-JTAG interface. PCB is not available but you can have all PCB data (<http://www.hs-augsburg.de/~hhoegl/proj/usbjtag/usbjtag.html>).

[c] The Wiggler cable is described here (beside others): http://wiki.openwrt.org/OpenWrt-Docs/Customizing/Hardware/JTAG_Cable. It is available e.g. from [4].

[d] Amontec <http://www.amontec.com>

[e] Olimex <http://www.olimex.com>

[f] This one is the only USB-to-JTAG connector for OpenOCD which does not use an FT2232. Instead it uses a USB bridge USBN9604 and an Atmel AVR Mega32. The project home is <http://www.embedded-projects.net>.

Note that the FT2232 has two independent channels. While channel one is always used for JTAG debugging, the second channel can freely be used for another purpose, e.g. for driving the target serial port. Not all JTAG adaptors in the above table have the second channel available for use (see above table, column „UART“).

7. References

[1] Home <http://openocd.berlios.de>

[2] Forum (Sparkfun). <http://forum.sparkfun.com/viewforum.php?f=18>

[3] Yargarto. <http://www.yagarto.de>

[4] Embedded Projects. <http://www.embedded-projects.net>, <http://shop.embedded-projects.net>.

[5] Reasonably priced target boards. Some firms building reasonably priced target boards usable with OpenOCD are Olimex (<http://www.olimex.com>), Embeddedartists (<http://www.embeddedartists.com>), Makingthings (<http://www.makingthings.com>), Embest (<http://www.embedinfo.com>) and Propox (<http://www.propox.com>). ARM

products from Olimex are also available from [4].

[6] Documentation. The OpenOCD source tree contains a manual in GNU Texinfo format (see directory / doc/). From this file other formats can be created which can be used for online reading on the screen (info, html) or for reading on paper (pdf).

[7] Sample project. <http://www.hs-augsburg.de/~hhoegl/proj/oocd-starter/index.html>

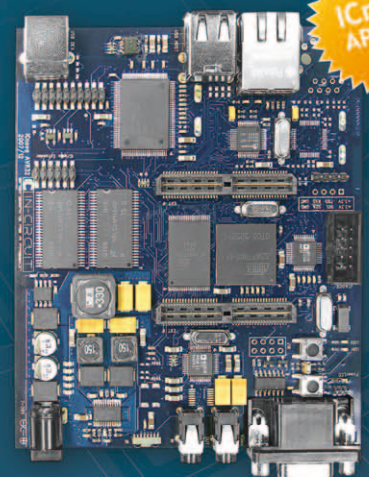


Our Youngstar

ICnova AP7000

In-Circuit GmbH your partner for application specific solutions for embedded signal processing and control.

- system partitioning
- schematic design
- PCB layout
- in-house prototyping
- volume production
- fault diagnostics and test
- embedded software development



ISO 9001 : 2000 certified

www.ic-board.de

In-Circuit GmbH
Königsbrücker Str. 69
01099 Dresden
Germany

Fon: +49 (0) 351 - 42 66 850
Fax: +49 (0) 351 - 42 66 849
Mail: office@in-circuit.de
Web: www.In-Circuit.de

3. Make a USBprog as OpenOCD Adapter

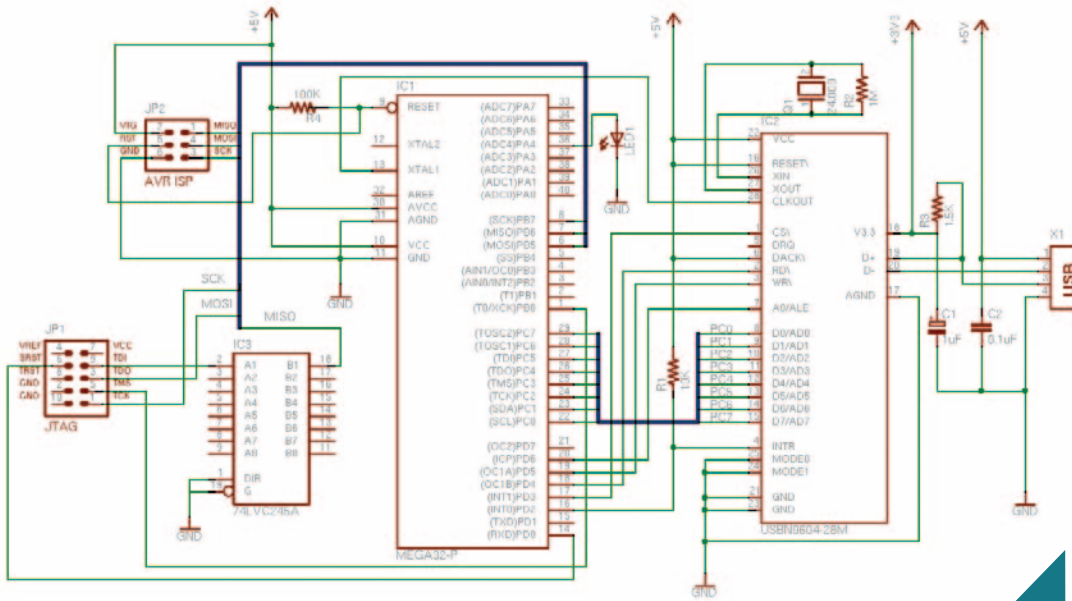


Figure 3. schematics of USBprog as a OpenOCD adapter.

You can build a new USBprog just like me, or add a level shifter and a JTAG header on your existing USBprog. Before or after the electronic task, firmware for OpenOCD [http://www.embedded-projects.net/index.php?page_id=177] should be downloaded to ATMEGA32.

If USBprog cannot be recognized by Linux or OpenOCD software

1. decoupling capacitor between VCC and GND is absolutely necessary for this high frequency circuit.
2. substitute the 24MHz crystal by a 24MHz external oscillator.
3. if you use a 24MHz crystal for USBN9604 other than a oscillator, 1M Ohm resistor between XIN and XOUT is necessary.

4. Install GNU Tool Chain For ARM

```
$mkdir [binutils-build] #In this way, we can keep the source code tree clean.
$cd [binutils-build]
$[binutils-source]/configure --target=arm-elf --prefix=[toolchain-prefix] \
--enable-interwork --enable-multilib
$make
$su -c ,make install`
$export PATH="$PATH:[toolchain-prefix]/bin"
```

```
$mkdir [gcc-build] #In this way, we can keep the source code tree clean.
$cd [gcc-build]
$su # you should become to root to execute the next configure script.
#[gcc-source]/configure --target=arm-elf --prefix=[toolchain-prefix] \
--enable-interwork --enable-multilib \
--enable-languages="c,c++" --with-newlib \
--with-headers=[newlib-source]/newlib/libc/include
$make
$su -c ,make install`
```

```
$mkdir [newlib-build] #In this way, we can keep the source code tree clean.
$cd [newlib-build]
$[newlib-source]/configure --target=arm-elf --prefix=[toolchain-prefix] \
--enable-interwork --enable-multilib
$make
$su -c ,make install`
```

```
$mkdir [gdb-build] #In this way, we can keep the source code tree clean.
$cd [gdb-build]
$[gdb-source]/configure --target=arm-elf --prefix=[toolchain-prefix] \
--enable-interwork --enable-multilib
$make
$su -c ,make install`
```

We have a evaluation board and a JTAG now, next you would like to install a free cross compiler and debug environment on your computer. Installation procedure is almost the same as AVR's, first fetch binutils-2.16, gcc-4.0.2, newlib-1.14.0, and gdb-6.6 source. Of course it's recommended to use the latest versions of these tools. Then uncompress these packages.

5. Install OpenOCD

First fetch the source archive from SVN Repository [http://developer.berlios.de/svn/?group_id=4148], then compile as follows.

```
$cd trunk
$./configure --enable-usbprog
$make
$su -c ,make install`
```

6. An example project

As an example project, download and unpack at91sam7s_getting_started_1.0.zip [http://www.atmel.com/dyn/resources/prod_documents/at91sam7s_getting_started_1.0.zip] from ATMEL. Edit Makefile and change two variables to the following values:

```
ERASE_FCT=rm -f
TARGET=AT91SAM7S32
```

I would like to add some lines for further use.

```
GDBINITFILE=gdbinit-$(OUTFILE_FLASH)
$(GDBINITFILE): $(OUTFILE_FLASH).elf
    @echo „file $(OUTFILE_FLASH).elf“ > $(GDBINITFILE)
    @echo „target remote localhost:3333“ >> $(GDBINITFILE)
    @echo „monitor arm7_9 force_hw_bkpts enable“ >> $(GDBINITFILE)
    @echo
    @echo „Use ,arm-elf-gdb -x $(GDBINITFILE) ``
```

Because we use a 16MHz crystal for main oscillator to generate 48MHz main clock, not a 18.432MHz one which was used by ATMEL's evaluation board, we should also

edit `lowlevel.c`, change `AT91C_BASE_PMC->PMC_PLLR` to a different value: also edit `include/AT91SAM7S-EK.h`:

```
/*
#define AT91B_MAIN_OSC 18432000 // Main Oscillator
MAINCK
#define AT91B_MCK ((18432000*73/14)/2) // Output PLL Clock
(48 MHz)
*/
#define AT91B_MAIN_OSC 16000000 // Main Oscillator
MAINCK
#define AT91B_MCK ((16000000*96/16)/2) // Output PLL Clock
(48 MHz)
#define AT91B_MASTER_CLOCK AT91B_MCK

#endif /* AT91SAM7S-EK_H */
```

run `$make` to compile the example source.

7. Use OpenOCD

We need two configuration files and one script file for OpenOCD to program and debug the target. They are not really necessary, but will automate our work. Here is my example.

```
#daemon configuration
telnet_port 4444
gdb_port 3333
daemon_startup reset
#interface
interface usbprog
jtag_speed 0
#use combined on interfaces or targets that can't set TRST/SRST separately
#reset_config trst_and_srst srst_pulls_trst
reset_config srst_only
#jtag scan chain
#format L IRC IRCM IDCODE (Length, IR Capture, IR Capture Mask, IDCODE)
jtag_device 4 0x1 0xf 0xe
#target configuration
#target <type> <startup mode>
#target arm7tdmi <reset mode> <chainpos> <endianness> <variant>
target arm7tdmi little run_and_init 0
run_and_halt_time 0 30
#flash configuration
#working_area 0 0x00200000 0x4000 nobackup
flash bank at91sam7 0 0 0 0
```

`openocd.cfg:`



openocd_flash.cfg:

```
#daemon configuration
telnet_port 4444
gdb_port 3333
daemon_startup reset
#interface
interface usbprog
jtag_speed 0
#use combined on interfaces or targets that can't set TRST/SRST separately
#reset_config trst_and_srst srst_pulls_trst
reset_config srst_only
#jtag scan chain
#format L IRC IRCM IDCODE (Length, IR Capture, IR Capture Mask, IDCODE)
jtag_device 4 0x1 0xf 0xe
ARM Getting Started: Using
USBprog and OpenOCD
6
#target configuration
#target <type> <startup mode>
#target arm7tdmi <reset mode> <chainpos> <endianness> <variant>
target arm7tdmi little run_and_init 0
run_and_halt_time 0 30
#flash configuration
#working_area 0 0x00200000 0x4000 nobackup
flash bank at91sam7 0 0 0 0
target_script 0 reset openocd_at91sam7s_flash.script
```

openocd_at91sam7s_flash.script:

```
##
The following command will be executed on
# reset (because of run_and_init in the config-file)
# - halt target
# - init ecr
# - flash content of file main.bin into target-memory
# - shutdown openocd
##
created by Martin Thomas
# http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects
# based on information from Dominic Rath
#
halt
sleep 10
# Init - taken from the script openocd_at91sam7_ecr.script
mww 0xfffffd44 0x00008000 # disable watchdog
mww 0xfffffd08 0xa5000001 # enable user reset
#mww 0xfffffc20 0x00000601 # CKGR_MOR : enable the main oscillator(18.432MHz)
mww 0xfffffc20 0x00000601 # CKGR_MOR : enable the main oscillator(16.000MHz)
sleep 10
#mww 0xfffffc2c 0x00481c0e # CKGR_PLLR: 96.1097 MHz(18.432MHz/14*(72+1))
mww 0xfffffc2c 0x005f1c10 # CKGR_PLLR: 96MHz(16.000MHz/16*(95+1))
sleep 10
#mww 0xfffffc30 0x00000007 # PMC_MCKR : MCK = PLL / 2 ~ = 48 MHz
mww 0xfffffc30 0x00000007 # PMC_MCKR : MCK = PLL / 2 = 48 MHz
sleep 10
mww 0xfffff60 0x003c0100 # MC_FMR: flash mode (FWS=1,FMCN=60)
# arm7_9 force_hw_bkpts enable # program resides in flash
# AT91SAM7 flash command-"batch"
# adapted by Martin Thomas based on information from Dominic Rath - Thanks
arm7_9 dcc_downloads enable
sleep 10
poll
flash probe 0
flash write 0 at91sam7s_getting_started_flash.bin 0x0
reset run
sleep 10
shutdown
```

```
$make gdbinit
$arm-elf-gdb -x gdbinit- at91sam7s_getting_started_flash
```

The only difference between openocd.cfg and openocd_flash.cfg is the last line.

openocd_at91sam7s_flash.script:

Copy these three files to the directory of example project, connect USBprog to target board, run

```
$openocd -f openocd_flash.cfg
```

at91sam7s_getting_started_flash.bin will be programmed to target's flash ROM and the program will start to run. If connect a LED to the PA0 port of our mini AT91SAM7S32 board, the LED will flash regularly.

For debug mode, run

```
$openocd
```

OpenOCD will open two TCP ports, we can telnet to port 4444 to access the command line interface.

```
$telnet localhost 4444
```

Or we can run gdb to debug the target. 

We can find more details of OpenOCD at its web page [http://openfacts.berlios.de/index-en?title=Open_On-Chip_Debugger] or „AT91SAM7S mit OpenOCD programmieren“ [http://www.mikrocontroller.net/articles/AT91SAM7S_mit_OpenOCD_programmieren].

Not The End

Congratulations, we have spent a meaningful weekend with pleasure. Take a shower and have a good dream...

But wait...

Although ARM is no longer mysterious to us, on the other hand, we just start a new trip to the unknown world.

Adventure is going on.

Artikelschreibwettbewerb

www.mikrocontroller.net <wettbewerb@mikrocontroller.net>

Kennst du dich mit einem Thema besonders gut aus?

Hast du ein interessantes Projekt das du schon lange mal vorstellen wolltest?

Dann hast du jetzt die Gelegenheit darüber einen Artikel unter einer Open-Source-Lizenz zu veröffentlichen und einen Preis zu gewinnen.

Vor 5 Jahren wurde das Mikrocontroller.net-Wiki eingerichtet. Ursprünglich nur als Plattform zur einfacheren Pflege des AVR-Tutorials und ein paar anderer Artikel gedacht, enthält das Wiki mittlerweile hunderte Seiten zu allen Bereichen der Elektronik. Einige sind kurze Übersichtsseiten zu einem bestimmten Thema, andere sind vollständige Artikel hoher Qualität, zum Beispiel über effiziente Software-PWM, eine Java-VM für AVR oder einen FPGA-basierten Audio-DSP.

Ein Problem das das Mikrocontroller.net-Wiki aber leider wie jedes andere Wiki auch hat, ist, dass es zwar viele Leute gibt die etwas beitragen möchten, aber nur wenige davon die Motivation haben ein Projekt oder einen Artikel komplett abzuschließen. Um das zu ändern veranstaltet Mikrocontroller.net zusammen mit Embedded Projects einen Artikelwettbewerb. Die besten neuen Artikel im Mikrocontroller.net-Wiki werden mit verschiedenen Preisen im Gesamtwert von mehreren hundert Euro prämiert.

Bei der Wahl des Themas gibt es wenig Einschränkungen: es sollte im weitesten Sinne etwas mit Elektronik zu tun haben und für möglichst viele Leser interessant sein. Es gibt keine Mindestanforderungen an Länge oder Umfang — die besten Artikel gewinnen, egal wie lang sie sind.

Ein paar Beispiele für mögliche Themen:

- Vorstellung eines eigenen Projektes
- AVR-Ada
- RTOS-Einführung, z.B. FreeRTOS
- Tutorial für ein Embedded Linux-Board (z.B. AVR32 Grasshopper)
- Funktion und Vergleich von IP-Stacks (uip, ...)
- Funkmodule
- Stromversorgung für Batteriebetrieb
- Beschleunigungssensoren
- Zustandsautomaten in C
- Filter in C

Die Artikel sollen wie alle Wiki-Inhalte unter die Creative-Commons-Lizenz gestellt werden.

Weitere Informationen zur Teilnahme und den Preisen findest du auf <http://www.mikrocontroller.net/wettbewerb>

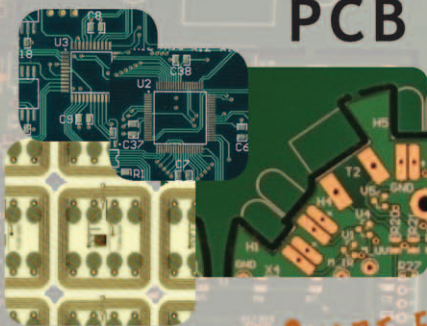
WWW.TOM-IC.COM

A dotSourcing Sàrl Company
CP 5909
1002 Lausanne
Switzerland

<http://www.tom-ic.com>
info@tom-ic.com
RFQ: sales@tom-ic.com
F: +41-22-545 78 63
T1: +41-22-548 09 00
T2: +41-22-548 09 04

One-Stop-Shop for prototype and volume PCB & Assembly services

PCB



STANDARD PCB SPECIFICATIONS:

- 1-8 layers (more layers on request)
- 8 working days leadtime
- No POOLING!
- 400 mm x 550 mm max panel size
- min hole size 10 mils
- min track width & gap 6 mils
- min location holes 1 mm
- min cutting space (between 2 PCBs) 3 mm
- V-Cut available for board thickness 0.4 mm to 1.6 mm
- Gold (Au) finishing
- RoHS
- All multilayer boards are 100% tested with flying probes. 1 & 2 layers are visually inspected.

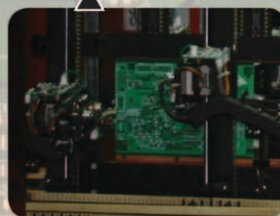
START FROM
30€!



The flying probes testing equipment used at Tom-IC for PCB products to test for major PCB faults is shown on the right.

A close up view on the probes themselves and a PCB is depicted as well below.

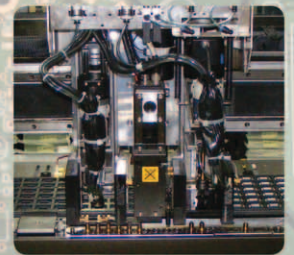
All open/short tests are automatically generated by the machine from the supplied customer Gerber sources so that no particular further test specifications are needed.



ASSEMBLY

Process:

- Max 15 working days leadtime (depends on components availability)
- RoHS



- Smallest component 0402
- Normal pitch > 0.4mm
- Solder SMT, THD type, BGA available
- BGA we solder only when pitch > 0.8mm
- All SMT are machine placed
- Edge clearance of 1mm

Quality:

- Visual Check
- X-Ray for BGA

Showtime –

TFT at the Grasshopper (AVR32) | Claude Schwarz <claude.schwarz@gmail.com>

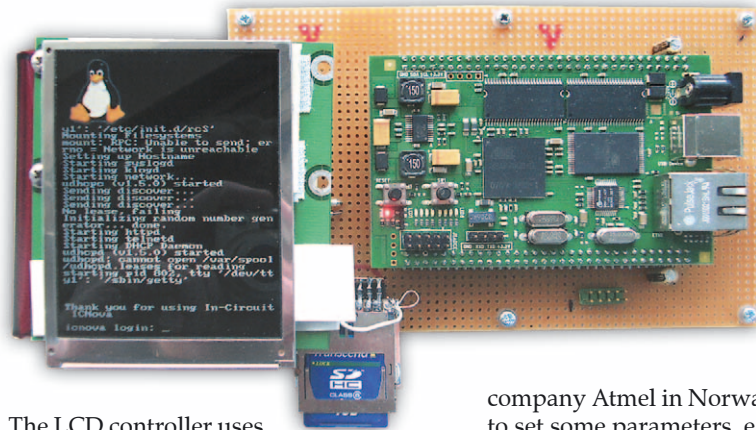
1. How to connect a TFT to the Grasshopper AVR32 Board

A picture is worth a thousand words. This sentence also applies to the world of embedded computing, embedded devices are more and more equipped with high resolution displays and GUIs. Today displays and GUIs are almost everywhere, from cellphones, MP3 players, digital cameras, television remote controls to even appliances as simple as coffee machines.

In this tutorial I will show you how to connect a TFT display to the Grasshopper AVR32 board and doing the necessary steps to get it working in Linux. As sample application I chose the famous mplayer mediaplayer, especially patched to get use of the advanced features of the AT32AP7000 application processor on the Grasshopper such as the vector multiplication unit and DSP/SIMD instructions .

The performance gain from this advanced features is really impressive compared to pure software implementations on similar processor platforms such as ARM or PowerPC.

The AP7000 houses a feature rich LCD controller which is able to drive most types of displays used today. Ranging from monochrome STN displays to state of the art TFT displays with 16.7 million colors and resolutions up to SVGA.



The LCD controller uses parts of the SDRAM connected to the AT32AP7000 as frame buffer memory, similar to on board graphics on PC mainboards. On the Grasshopper this SDRAM is 32 bit wide connected and the

LCD controller frame buffer data is fed from a DMA engine, so there is plenty of memory bandwidth and CPU performance available to even display PC desktop like resolutions with ergonomic refresh rates and pleasing color depths. The LCD controllers user interface consists of nearly 300! 32 bit registers.

Fortunately the hard part to get this LCD controller working in Linux is already done by some people from the company Atmel in Norway. We only have to set some parameters, easily to get from the displays data sheet.

So let us start with the tutorial.

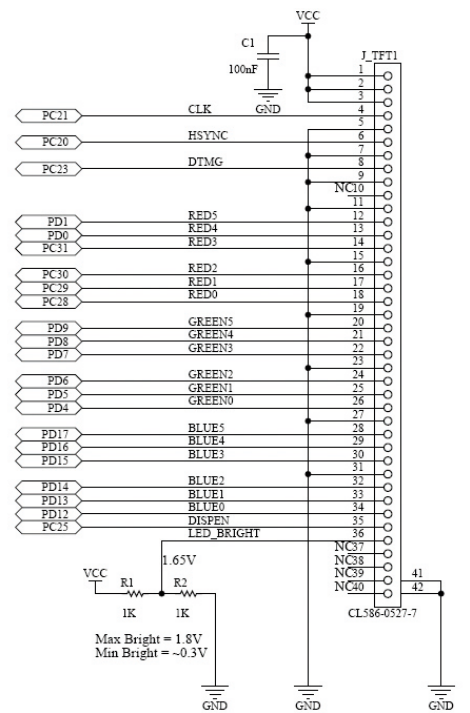
2. Hardware implementation

At first I wanted to use the Sharp Q043T3DX02 480x272 24bit TFT display for this tutorial. This type of display is cheap and easily obtainable from game console spare part shops and online auctions as "PSP Display". But mine did not arrive in time so I have to stay with a Hitachi TX09D70VM1CDA 240x320 18bit TFT which I used in a customer project at work. A little bit more expensive than the Sharp one , but very simple to use and extremely reliable with a MTBF of over 10k hours. It only needs a single operating voltage of 3.3V, all typical voltages required for a TFT are generated on board.

From the software point of view the two displays differ only in some parameters, so you will get a clue of how to hook up a TFT in general form this tutorial.

The wiring between the Grasshopper and the TFT is really simple.

Care should be taken on the CLK line, excessive ringing or jitter may result in an distorted picture or malfunction of the TFT. Adding a series termination resistor (20 to 50 Ohm) on this line can improve a lot.



3. Software implementation

The Linux frame buffer device is an ideal choice for embedded devices since it is easy to implement and very universal in it's use. You can run anything from plain text consoles to full blown X11 on it. It is also available on desktop PCs, so you can develop and debug frame buffer applica-

tions on your desktop machine. The LCD controller on the Grasshopper is disabled by default, so we first must enable it in the startup file of the Grasshopper and tell the Linux kernel that the LCD controller can be used as frame buffer device.



The startup file is located at :

```
/your_grasshopper_directory/build_avr32/linux-2.6.24
-icnova/arch/avr32/boards/icnova/icnova_base.c
```

Open it with your favorite editor and add the following lines:

```
#include <linux/fb.h>
#include <video/atmel_lcdc.h>
```

and after

```
#undef ICNOVA_USB_CP2102
```

this lines:

```
static struct fb_videomode grasshopper_tft_modes[] = {
    {
        .name          = „TX09D70 @ 60“,
        .refresh       = 60,                /* refresh rate */
        .xres          = 240,              /* horizontal resolution */
        .yres          = 320,              /* vertical resolution */
        .pixclock      = KHZ2PICOS(4965), /* pixel clock in kHz */
        .left_margin   = 1,                /* h. front porch */
        .right_margin  = 33,               /* h. back porch */
        .upper_margin  = 1,                /* v. front porch */
        .lower_margin  = 0,                /* v. back porch */
        .hsync_len     = 5,                /* hsync length */
        .vsync_len     = 1,                /* vsync lenght */
        .sync          = FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT,
        /* Active high hsync impulse , active high vsync im-
pulse */
        .vmode         = FB_VMODE_NONINTERLACED,
        /* always send a full frame to the display */
    },
};

static struct fb_monspecs __initdata grasshopper_default_monspecs = {
    .manufacturer    = „ATM“, /* ATM = Atmel */
    .monitor         = „GENERIC“, /* Generic type */
    .modedb          = grasshopper_tft_modes,
    .modedb_len      = ARRAY_SIZE(grasshopper_tft_modes),
    .hfmin           = 14820, /* doesn't affect the lcdc! */
    .hfmax           = 32000, /* doesn't affect the lcdc! */
    .vfmin           = 30, /* doesn't affect the lcdc! */
    .vfmax           = 200, /* doesn't affect the lcdc! */
    .dclkmax         = 30000000, /* doesn't affect the lcdc! */
};

struct atmel_lcdfb_info __initdata grasshopper_lcdc_data = {
    .default_bpp     = 16, /* Colur depth */
    .default_dmacon  = ATMEL_LCD_CDMAEN | ATMEL_LCD_CDMA2DEN,
    .default_lcdcon2 = (ATMEL_LCD_DISTYPE_TFT
        | ATMEL_LCD_CLKMOD_ALWAYSACTIVE
        | ATMEL_LCD_MEMOR_BIG),
    .default_monspecs = &grasshopper_default_monspecs,
    .guard_time      = 2,
};
```

```
at32_add_device_lcdc(0, &grasshopper_lcdc_data, fbmem_start, fbmem_size);
```

Finally, to enable the LCD Controller and make it usable as frame buffer device add this line in the static int __init icnova_init(void) function:

```
export PATH=/your_grasshopper_directory/build_avr32/staging_dir/bin:$PATH
```

In the next steps we configure the Linux Kernel to use the LCD controller and frame buffer. For easy cross compiling of the kernel and applications add the Grasshopper toolchain temporarily to your path variable...

```
alias crossmake='ARCH=avr32 CROSS_COMPILE=avr32-linux- ,
```

and make an alias (crossmake)...

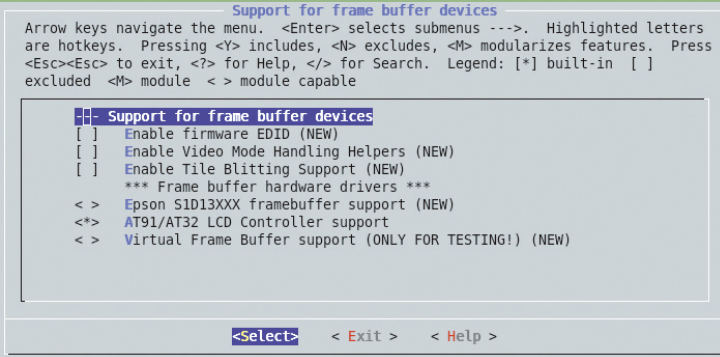
```
crossmake make clean menuconfig uImage
```

to start the kernel configuration “cd” in your grasshopper directory to /build_avr32/linux-2.6.24-icnova/ and type :

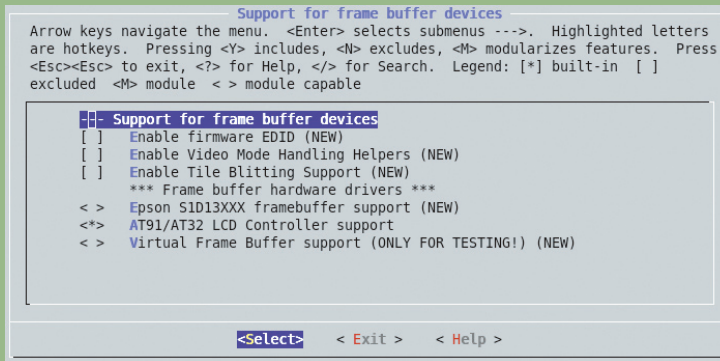
Go to the "Device Drivers" submenu and select "Graphics support" Select the options according to the screenshots.

```
/your_grasshopper_directory/build_avr32/linux-2.6.24-icnova/arch/avr32/boot/images/uImage
```

This step enables the Linux frame buffer and the Penguin bootup logo. In the frame buffer device submenu :



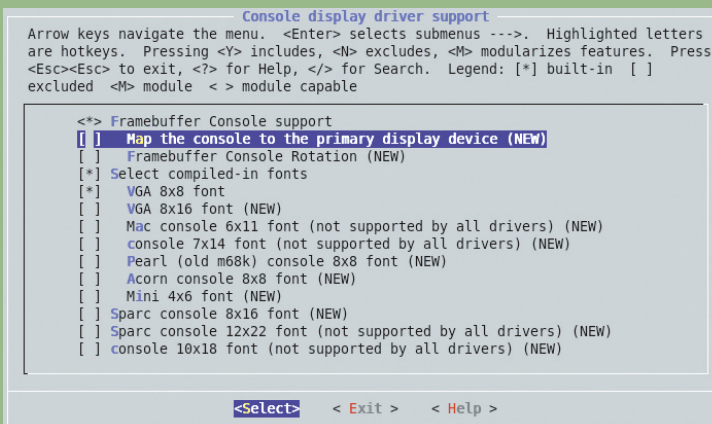
This step selects the AVR32 LCD controller .



The next step is optional, if you want a console terminal on your frame buffer go one level up and in the "Char devices" submenu

and select the options like in the screenshot. If you don't want a frame buffer terminal just leave this submenu as it is.

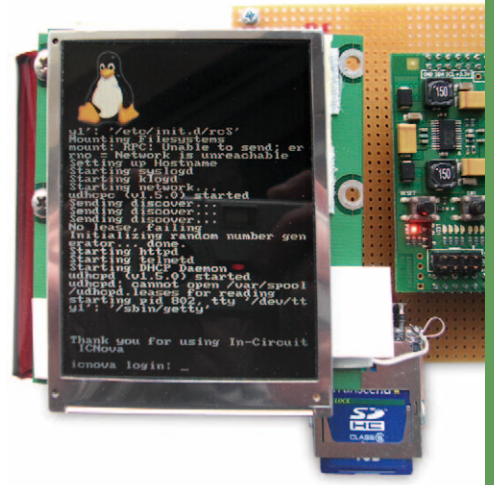
Optional console on frame buffer



Now exit and save the kernel configuration. The build process should start and if all goes well your new kernel should be in:

Copy your new kernel (with nfs,ftp,tftp or whatever you like) to the board in the /boot directory . Don't forget to rename and/or backup your old kernel. I renamed mine to uImage_old , so I can load the old kernel in U-Boot with "fsload uImage_old" in case the new kernel doesn't work as expected.

If all went fine you should see something similar to this :



4. Sample application (mplayer)

Everybody who has ever watched a movie in Linux knows mplayer. Atmel published a patch for mplayer which adds support for the advanced features of the AT32AP7000 processor.

There is a nice howto for AVR32 at : <http://www.avr32linux.org/twiki/bin/view/Main/MPlayer>

Since we don't have GTK , X11, some libraries and sound support at this point, follow the instructions for the command line version and edit the configuration script disabling all also, sound and libmad stuff.

Now just use the alias "crossmake" explained earlier to compile mplayer and everything should work.

Copy the resulting binary to your board, set executable permissions and type in :

```
./mplayer -nosound path_to_my_video_file/my_video_file.mpXX
```

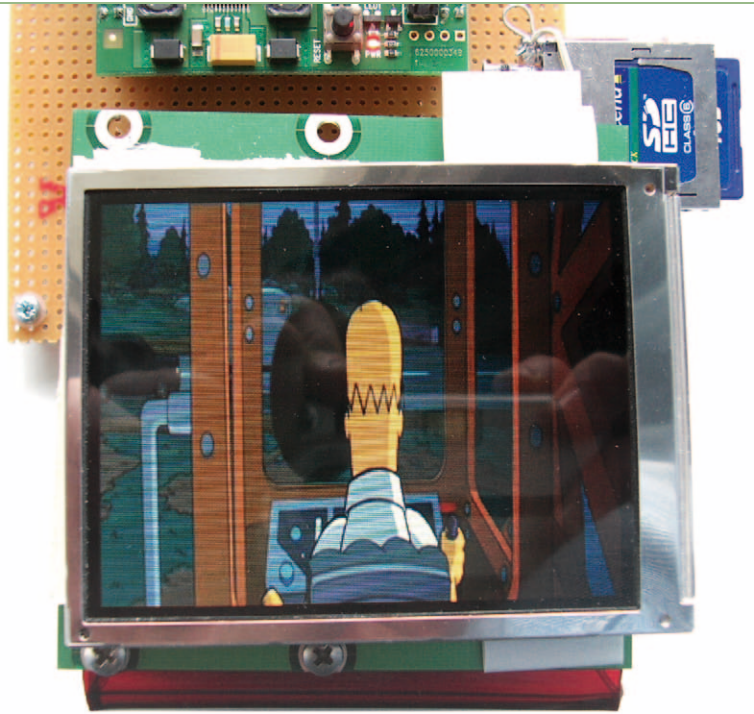

I tested mplayer with a couple of MPEG2 MP4 and AVI files, all of them worked without problems. Playing a fullscreen 900kbps MPEG2@24fps leads in roughly 30% processor load!

A slightly higher clocked (190MHz) ARM9 hardly reaches 15fps with this file at 100% processor load. Watching a movie trailer on the Grasshopper.

Sorry for the bad picture quality, it is difficult to take a good picture from a TFT display. In real the picture on the screen is really sharp and vivid!

I hope this tutorial was informative for you, if you are planing to add a display to the Grasshopper I suggest you read further documentation because this tutorial will give you just a brief overview. Most documents of interest you will find in the /your_grasshopper_directory/build_avr32/linux-2.6.24-icnova/Documentation directory.

You can download all project files and as soon as my display arrives, a patched icnova_base.c for this type of displays at :<http://forum.embedded-projects.net/> just search the AVR32 forum.



Compact graphics framework

für LCDs (cmg) | Hubert Hoegl <Hubert.Hoegl@hs-augsburg.de>

1. CMG Allgemein

CMG ist eine GPL-lizenzierte, modulare Grafikbibliothek mit folgender Zielsetzung:

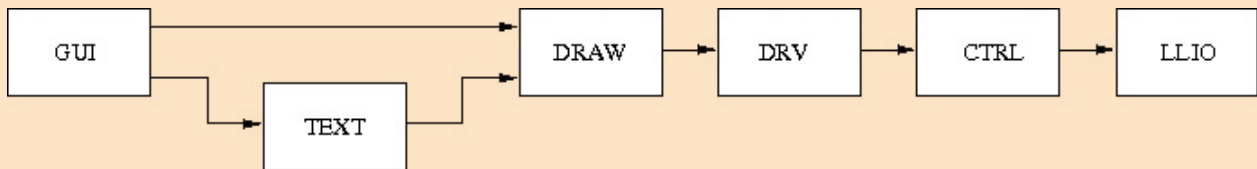
- Verwendbar auf verschiedenen (kleinen) Mikrocontrollern, z.B. AVR und ARM7.
- Unabhängigkeit vom Grafikcontroller auf dem Display. Es werden zur Zeit

die folgenden Typen unterstützt:

T6963C (Toshiba), S1D13305 (Epson), S1D13700, LDS176 (Leadis), LH155 (Sharp).
Die getesteten Displays waren PG240128-A (Powertip, T6963, 240x128), DG12864-12 (Datavision, T6963, 128x64), ED32FY0FLW (EDT, S1D13700, 320x240), M078CKA (Sharp, LH155, 240x64).

- Einfaches API zur Ansteuerung.
- Minimum an Ressourcenverbrauch.
- Simulation der Grafik auf dem PC. Autor von CMG ist Christian Merkle (<christian@cmerkle.de>). Er hat die Bibliothek im Rahmen einer Masterarbeit an der Fachhochschule Augsburg geschrieben.

Schichten der CMG Grafikbibliothek.



2. CMG-Schichten

GUI

Sie greift auf die Schichten DRAW und TEXT zu. Eigenschaften von GUI sind:

- Fenster
- Vorgefertigte Elemente: Dialogfenster, Schaltflächen, Texteingabefelder, Auswahlknöpfe und andere.

Die GUI Schicht ist leider noch nicht ausgeführt.

TEXT

CMG kann Texte mit unterschiedlichen Fonts auf das Display schreiben. Ein Font in CMG wird in einer C Datenstruktur gespeichert. Mit Hilfe des auch in dieser Arbeit entstandenen Werkzeugs CMGFontConvert kann ein beliebiger Windows Font in diese Struktur umgewandelt werden. Da kleine Mikrocontroller wenig RAM Speicher haben, wurde auch daran gedacht, bei Bedarf neue Font-Daten aus einem grösseren (Flash) Speicher nachzuladen.

1. Auszug aus dem Arial Font

(Die Abbildung auf der nächsten Seite unten zeigt ein simuliertes Display mit verschiedenen bekannten Fonts.)

```

/* Character Data „A“ (width, 20 data bytes): */
0x07, 0x10, 0x00, 0x28, 0x00, 0x28, 0x00, 0x28, 0x00, 0x44, \
0x00, 0x7c, 0x00, 0x82, 0x00, 0x82, 0x00, 0x00, 0x00, 0x00, 0x00,
  
```

DRAW

Die DRAW Schicht stellt allgemeine Zeichenfunktionen zur Verfügung.

- Pixel und Linien
- Füllbare geometrische Figuren (Rechtecke, Dreiecke, Kreise, Ellipsen, abgerundete Rechtecke). Diese haben einen Rand um die Figur und einen inneren Füllbereich. Der Rand wird mit einer Stift gezeichnet, der Innenbereich mit einem Pinsel.
- Bit-Block Transfer („blitting“)

DRV

Die Graphic-Driver-Schicht DRV abstrahiert das CTRL-Interface - vom Zugriff auf einen

linearen Speicherbereich - zu einem Interface der grundlegendsten Grafikfunktionen.

- Stifte: Spooling und Skalierung
- Farbe: Zur Zeit nur schwarz/weiß (1bpp)
- ROP (rasterization operation): COPY, XOR, AND, OR
- Mode management

CTRL

Die Controller Schicht hat die Aufgabe, die Komplexität des verwendeten LCD Hardware-Controllers zu verbergen. Es geht auch darum, die im Controller bereits vorhandenen schnellen Funktionen zu nutzen. Bei einfacheren Controllern werden diese Funktionen

```

// Init/Exit
cmg_Result CMG_CTRL_Init(void);
void CMG_CTRL_Exit(void);
void CMG_CTRL_PowerManagement(cmg_mu8 uState);
cmg_Result CMG_CTRL_ControllerSpecific(cmg_mu8 uFunction, cmg_void *pParam);
// API
void CMG_CTRL_FastClearScreen(cmg_u8 byCol);
void CMG_CTRL_SetAddressPtr(cmg_ScrAddr wScreenAddress);
void CMG_CTRL_SetDirections(cmg_ms8 iDirections);
void CMG_CTRL_SetByte(cmg_u8 byData);
void CMG_CTRL_GetByte(void);
  
```

Beispiel 1. Das CTRL API

in Software nachgebildet. Zur Zeit werden die Controller T6963, LH155 und S1D13700 unterstützt.

LLIO

Die Low Level Input Output Schicht ermöglicht den plattformunabhängigen Zugriff auf das Display. Als Plattformen sind zur Zeit AVR, ARM und X86 (Parallelport) vorgesehen. Der Anschluss des Displays an den Rechner kann über freie I/O-Leitungen oder SPI erfolgen. Die Displays können im 8080- (Intel), 6800- (Motorola) oder SPI-Modus betrieben werden.



Neben den „realen“ Displays kann man CMG auch auf einem simulierten Display betreiben. Dabei wird auch der Controller - in diesem Fall immer der T6963 - simuliert. Die grafische Ausgabe kann entweder unter Linux (GTK+) oder Windows (GDI) erfolgen.

3. Demo-Code

4. Download

Mit folgender Kommandozeile kann man den aktuellen Quelltext aus dem Subversion Repository auschecken:

```
svn co https://rabbit.informatik.fh-augsburg.de/cmg/svn
```

4. Links

Heimat des Projektes:

<https://rabbit.informatik.fh-augsburg.de/cmg/trac>

Subversion Quelltext-Repository

<https://rabbit.informatik.fh-augsburg.de/cmg/svn>

Masterarbeit (steht unter Creative Commons Lizenz) <http://www.hs-augsburg.de/~hhoegl/tmp/ma-merkle/>

Beispiel 2. Das LLIO API

```
cmg_Result CMG_LLIO_Init( void )
void CMG_LLIO_Exit( void );
void CMG_LLIO_PowerManagement( cmg_mu8 uState );
void CMG_LLIO_WriteA0( cmg_u8 byData );
void CMG_LLIO_WriteA1( cmg_u8 byData );
cmg_u8 CMG_LLIO_ReadA0( void );
cmg_u8 CMG_LLIO_ReadA1( void );
```

Display	Hersteller	Size	Controller
PG240128-A	Powertip	240x128	T6963 (Toshiba)
EW32FY0FLW	EDT	320x240	S1D13700 (Epson)
M078CKA	Sharp	240x64	LH155 (Sharp)

Beispiel 3. CMG Demo

```
[..]
void Test( void )
{
    // desktop background
    CMG_SetROPMode( ROPMODE_COPY );
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_SetBrush( &BrushLight );
    CMG_Rectangle( 0, 0, iScreenSizeX - 1, iScreenSizeY - 1 );
    Dialog( 5, 6, 220, 100, „CMG - Christian Merkle Graphics“ );
    Dialog( 30, 31, 236, 125, „Hello from my second dialog“ );
}

int main( void )
{
    cmg_Result iResult;
    iResult = CMG_TEXT_DrawAndTextInit();
    if ( iResult != CMG_OK )
    {
        //printf( „ *** ERROR 0x%02x ***\n\n“, iResult );
        return -1;
    }

    CMG_Sleep_ms( 500 );
    CMG_GetScreenSize( &iScreenSizeX, &iScreenSizeY );
    CMG_CreatePen( &PenDashDotDot, dataPenDashDotDot, 2 );
    CMG_CreateBrush( &Brush50, dataBrush50, 1, 2 );
    CMG_CreateBrush( &BrushLight, dataBrushLight, 3, 6 );
    CMG_CreateBrush( &BrushHelloWorld, dataHelloWorld, 6, 6 );
    CMG_TEXT_Font_Create( &Font_Arial_8, g_Font_Arial_8, 1 );
    CMG_TEXT_Font_Set( &Font_Arial_8 );
    Test();
    while ( 1 );
    CMG_TEXT_Exit();
    return 0;
}
```

Printed Cervice Boards

►► Jetzt Neu!

Flexible Leiterplatten ONLINE!

►► Starre Leiterplatten bis 8 Lagen online

Expressdienst ab 12 Stunden
Pünktlich oder kostenlos!

www.leiton-gmbh.de
kontakt@leiton-gmbh.de
+49-(0)30-701 73 49 10

XSVF Player with USBprog

Programming Xilinx devices with USBprog | [Sven Luetkemeier <sven@sl-ware.de>](mailto:sven@sl-ware.de)

1. Introduction

Programmable Logic Devices (PLDs) have become important microelectronic components in the past decades, filling the gap between microprocessors and Application Specific Integrated Circuits (ASICs). Microprocessors are very flexible but offer only limited performance and dissipate a lot of electric power. ASICs on the other hand provide high performance and small power consumption but flexibility is limited as all functionality is fixed at design time. PLDs may be seen in between, combining high flexibility and high perfor-

mance while power dissipation is still in an acceptable range. Other advantages of PLDs with respect to ASICs are the lower costs when considering small device quantities, the shorter and highly automated application development process, and the ability to reprogram the device in case of bug fixes or new functionality. Today there are mainly two types of PLDs, Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs). Both types consist of sets of so called macro cells which are able to implement combi-

national or sequential logic functions and a configurable interconnection structure. CPLDs and FPGAs differ in their architecture and their complexity, as FPGAs are usually far more complex and faster than CPLDs and often contain special blocks like dedicated multipliers, on-chip RAM, or even embedded microprocessors. The aim of this tutorial is to show how to develop a CPLD application. We will demonstrate how to implement a simple LED blinker and how to program the blinker code into a Xilinx XC9572 CPLD using usbprogXSVF.

2. Tutorial

This device type was chosen because it is very cheap, easy to use, and available through a lot of online shops. The minimum hardware we will assume in this tutorial is shown in figure 1. You may also use different hardware if you make appropriate changes to the code examples shown in this article. For example, you might want to connect the LED to a different port of your CPLD or you might want to use an XC95144XL instead of the XC9572.

The software suite we will use for developing our application is Xilinx ISE WebPACK 9.2i which can be downloaded for free from the Xilinx website [1]. It is assumed that you have installed this software if you want to take part in this tutorial. The development process of our application consists of the following steps:

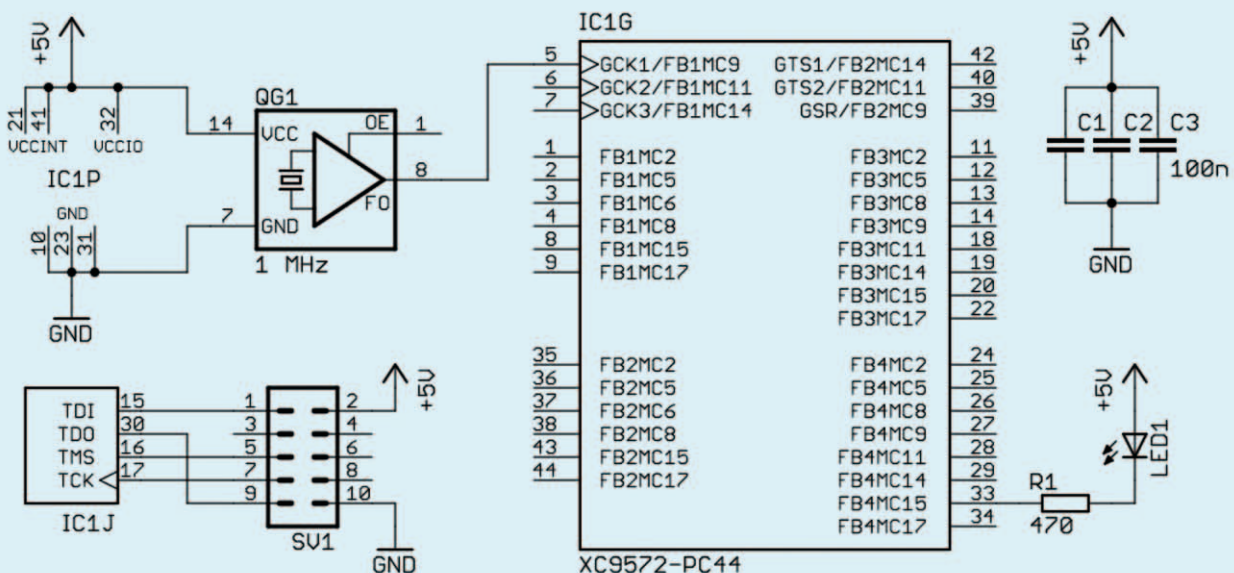
1. create a hardware description
2. define user constraints
3. compile the design
4. create a programming file
5. program the target device

The hardware description specifies the behaviour of our design, which is an LED blinker. We will use the VLSI Hardware

Description Language (VHDL) in this tutorial for this purpose. The VHDL Cookbook [2] is a good reference if you are interested in VHDL. User constraints are additional hints needed by the compiler, but which are independent of the hardware description itself. For example we have to tell the compiler to which I/O pins of the CPLD we want to connect the LED and the crystal oscillator. This is depending on our hardware but is independent of the blinker behaviour. The programming file is the interface between the compiler suite, ISE WebPACK, and the programming software, xsvfplayer, which is the command line tool of usbprogXSVF. We will now go step by step through the development process described above.



Figure 1. Minimum hardware used in the tutorial



After starting the Project Navigator of Xilinx ISE WebPACK, we create a new project by selecting „File - New Project“ from the main menu. The „New Project Wizard“ opens. Name the project „BLINK“, select the desired destination path, and choose „HDL“ as the Top-Level Source Type. After clicking on the „Next“ button set up all parameters as follows:

Property Name	Value
Product Category	All
Family	XC9500 CPLDs
Device	XC9572
Package	PC44
Speed	-15
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Preferred Language	VHDL
Enable Enhanced Design Summary	on
Enable Message Filtering	off
Display Incremental Messages	off

If you want to use different hardware, you have to specify the appropriate settings here. Be sure to set up exactly the correct device type and device package. When you're finished, click on „Next“ and then on „New Source“, as we want to create a new hardware description. In the „New Source Wizard“ select „VHDL Module“ and specify „BLINK.vhd“ as the file name. Click on „Next“. On the next page we can specify the ports of our new module. Ports are input and output connections a module has. Our LED blinker has a clock input and a LED output. Thus, we create the following two rows in the Ports Table:

Port Name	Direction	Bus	MSB	LSB
CLK	in	off		
LED	out	off		

You can now repeatedly click on „Next“ / „Finish“ until the „New Source Wizard“ and the „New Project Wizard“ close.

In the main window of the Project Navigator you now see the VHDL code framework that the New Source Wizard created for us. The crystal oscillator that is connected to the XC9572 CPLD operates at a clock frequency of 1 MHz. We will implement a 20 bit frequency divider (counter) which results in an LED blinking frequency of approximately 1 Hz. The VHDL code that is needed for this task is shown below (comments have been stripped from the code to make it more compact). Copy the code to your BLINK.vhd file.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity BLINK is
Port ( CLK : in STD_LOGIC;
LED : out STD_LOGIC);
end BLINK;
```

```
architecture Behavioral of BLINK is
signal CNT : STD_LOGIC_VECTOR (19 downto 0) := (others => ,0`);
begin
COUNTER: process (CLK)
begin
if CLK'event and CLK = ,1` then
CNT <= CNT + 1;
end if;
end process;
LED <= CNT(19);
=20
end Behavioral;
```

The process named COUNTER describes a register that increments on every positive edge of the clock signal CLK. The content of the register is available through the signal CNT. As CNT is a 20 bit wide vector, the most significant bit of the counter toggles at a rate of $1/2^{20}$ MHz which is approximately 1 Hz. The code line below the process connects the LED output to this counter bit.

The next step is to create a User Constraints File. In the upper left part of the Project Navigator select the item „BLINK - Behavioral (BLINK.vhd)“ from the tab „Source“. After that unfold the item „User Constraints“ (click on the plus symbol in front of it) in the tab „Processes“ which you find in the lower left part of the Project Navigator. Double-click on „Edit Constraints (Text)“ and answer „Yes“ to the question

that pops up. After that the editor shows an empty file „BLINK.ucf“ into which we can enter the required user constraints. For this application we have to specify that

- the signal CLK of our BLINK module should be physically connected to pin 5 of the XC9572 (which is GCK1 in the PC44 package), and that
- the signal LED of our BLINK module should be physically connected to pin 33 of the XC9572 (which is the output of Function Block 4, Macro Cell 15 in the PC44 package).

If you use the hardware proposed in this tutorial (see figure 1) enter the following two lines into the User Constraints File to achieve this:

```
NET „CLK“ LOC = „P5“;
NET „LED“ LOC = „P33“;
```

It is very important to specify the correct physical connections here, as you might damage your hardware if you set up something wrong! You might have to specify different connections here, if you use a different CPLD type, a different package, or a different wiring scheme. You should connect the crystal oscillator to one of the GCK pins of the CPLD and the LED to an arbitrary I/O pin of the CPLD.

After saving both the VHDL code and the User Constraints File we are ready to compile the design. In the „Processes“ tab double-click on „Implement Design“ and wait for the compilation process to finish. If compilation was successful a compilation report should automatically open in your browser. If you go back to the Project Navigator you should see an exclamation mark on yellow background in front of the „Implement Design“ item in the

„Processes“ tab because the compiler emitted some warnings. We can ignore these warnings for our design. If you encounter an x on red background instead of the exclamation mark, an error occurred during the compilation process. Examine the „Console“ or the „Errors“ tab at the bottom of the Project Navigator window to see what went wrong.

If compilation was successful we can now create a programming file for usbprogXSVF. XSVF files are a standard to describe arbitrary JTAG operations. As most CPLDs and FPGAs can be configured through a JTAG port, we use XSVF files as the interface between the compiler suite and usbprogXSVF. To create an XSVF file for our LED blinker unfold the „Implement Design“ item in the „Processes“ tab and also the „Optional Implementation Tools“ item, which will show up a few lines below. Double-click on „Generate SVF/XSVF/STAPL File“ and wait for the „iMPACT“ window to open. Select „Prepare a Boundary-Scan File“, choose „XSVF“ in the drop-down list below, and click on „Finish“. In the file selector that opens specify „BLINK.xsvf“ as the file name and click on „Save“. You can confirm the following message with „OK“. A new file selector with the title „Add Device“ opens. Select the file „BLINK.jed“ and click on „Open“. The main window should now contain a Xilinx CPLD symbol. We can now record arbitrary JTAG operations in the XSVF file. As we just want to program the CPLD with the LED blinker firmware, right-click on the Xilinx device symbol and select „Program“. You can leave the default settings in the following window as they are and just click „OK“. You should see the message „Program Succeeded“ afterwards. To stop recording JTAG operations select „Output - XSVF File - Stop writing to XSVF

Description	Pin	Pin	Description
TDI	1	2	VCC
	3	4	
TMS	5	6	
TCK	7	8	
TDO	9	10	GND

File“ from the main menu.

We can now program the CPLD with USBprog. Plug USBprog into a free usb port and make sure that the usbprogXSVF firmware is loaded into your programmer. You can download the most up-to-date firmware and command line tool from [3]. Power up the CPLD hardware and connect it to USBprog. The following pin assignment is used on the 10 pin connector of USBprog:

If you use the hardware proposed in figure 1, a 1:1 ribbon cable with 10 pin female connectors on both sides does the job. Open a console window on your PC and make sure that the command „xsvfplayer“ is on your path. Change to the directory where the Xilinx ISE files of the BLINK project are stored. You can then start the programming process with the command „xsvfplayer BLINK.xsvf“. The programming process takes a while. It is also normal that the progress stays at 0 % for some time. After programming is finished the LED connected to your CPLD should start to blink. If you encounter any errors you should check

- your CPLD hardware, especially the wiring of the JTAG interface,

- that your CPLD is powered up correctly,
- that you chose the exactly correct CPLD type of your hardware in Xilinx ISE (e.g. XC9572XL is something different than XC9572),
- that you have installed all libraries needed by USBprog correctly (libusb etc.),
- that you have sufficient permissions to access the USBprog hardware.

If you should have any questions or problems left, we invite you to visit the USBprog forum at <http://forum.embedded-projects.net/> and ask for help. We would be happy to assist you.

Congratulations, you've completed this tutorial!

3. References

[1] Xilinx ISE WebPACK, http://www.xilinx.com/ise/logic_design_prod/web-pack.html

[2] Peter J. Ashenden, The VHDL Cookbook, <http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf>

[3] usbprogXSVF firmware and command line tool from subversion repository, <http://svn.berlios.de/svnroot/repos/usbprog/trunk/usbprogXSVF/>

Qualitätsweine, powered by 8-Bit RISC

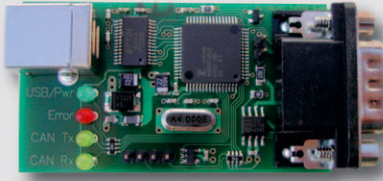
WEINGUT
HÖRNER

www.weingut-hoerner.de

Tiny-CAN I

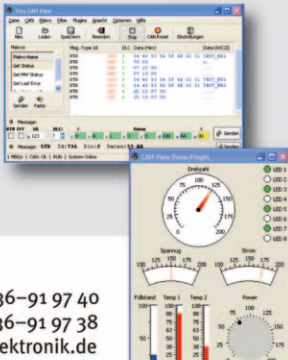
Low-Cost USB-CAN-Adapter

Neu!



Open Source CAN-Monitor für Windows und Linux

- Makros, Filter
- Plugin-fähig



- 4 LEDs zur Statusanzeige
- Datenraten bis 1 MBit/s
- Kompatibel zu anderen Produkten
- Treiber-API für Windows und Linux (Open Source)

MHS | **MHS-Elektronik GmbH & Co. KG** | F +49 (0) 85 36-91 97 40
 Fuchsöd 4 | info@mhs-elektronik.de
 D-94149 Kößlarn | www.mhs-elektronik.de

USB Grundkurs

USB verstehen und anwenden | **Benedikt Sauter** <sauter@ixbat.de>

1. Warum ist der Einstieg in USB so schwer?

USB ist für Mikrocontroller-Schaltungen eine ideale Ergänzung. Schnelle Übertragungsraten, flexible Kommunikationskanäle, flexible Kommunikationskanäle und eine integrierte Stromversorgung sind die bekanntesten Schlagwörter von USB. Doch obwohl USB bereits seit 1996 existiert, findet man immer noch sehr häufig die klassische RS232 Verbindung in neuen Projekten.

Es stellt sich also die Frage, warum es die USB Schnittstelle bis heute nicht geschafft hat, RS232 abzulösen. Meiner Meinung

nach ist eines der Hauptprobleme, dass auf diesem Gebiet viel zu wenig gute Literatur existiert. Oftmals hört man von Entwicklern, dass es bei Weitem nicht ausreicht, etwas flüchtig über die USB Schnittstelle zu lesen, um sie richtig verstehen und einsetzen zu können. Das liegt wohl daran, dass die meisten Texte und Bücher leicht veränderte Übersetzungen der knapp tausendseitigen USB Spezifikation sind.

Die USB Spezifikation ist zwar im Gegensatz zu vielen anderen Spezifikationen sehr

ausführlich verfasst, jedoch ist und bleibt sie die komplexe Spezifikation von USB. Daher ist sie als Lernunterlage für das Selbststudium oder als Entwicklungslitfadent völlig ungeeignet.

Mit dieser Internetseite (PDF Datei) soll diesem Problem entgegengewirkt werden. Es wird ein einfaches Modell eingeführt, mit dem man schnell in der Lage sein soll, USB verwenden zu können.

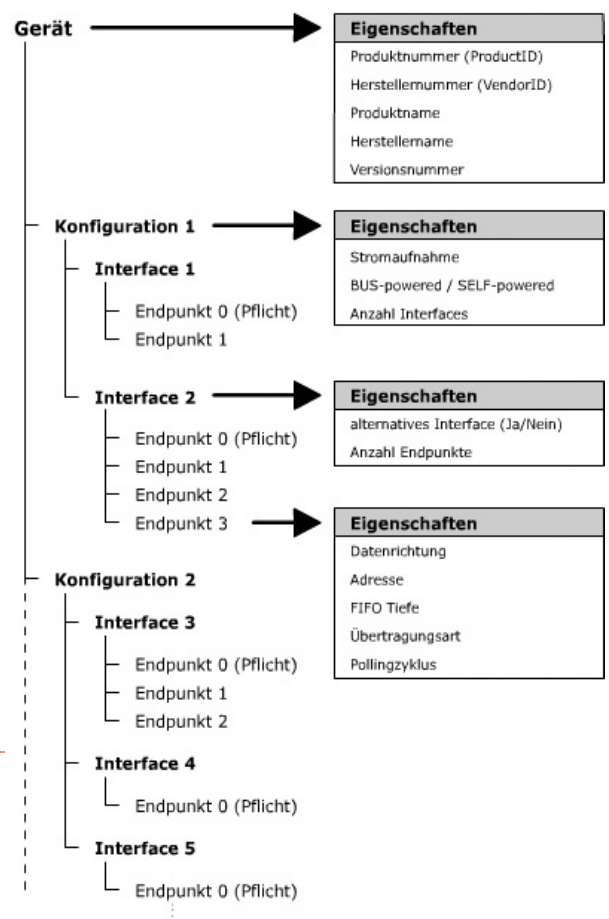
2. Ein einfaches Modell zum besseren Verstehen

Eine USB Schnittstelle hat kein typisches Aussehen, wie z.B. eine Schnittstelle vom Typ RS232. Bei einer RS232-Verbindung gibt es immer mindestens je eine TX und RX Leitung. Die TX Leitung ist ausschliesslich dafür da, Daten zu senden, die RX Leitung, um Daten zu empfangen. Eine solche feste Regel gibt es bei USB nicht. Hier ist es möglich, mehrere RX und TX Leitungen parallel in einer USB Schnittstelle zu definieren. Dadurch kann man verschiedene Daten über eigene Verbindungen übertragen.

Stellen wir uns die USB Schnittstelle kurz einmal folgendermassen vor: Wir können Daten gezielt über die verschiedenen Leitungen senden. Wenn wir beispielsweise eine Soundkarte bauen wollen, benötigen wir zwei TX Leitungen für die beiden Stereokanäle und eine weitere TX Leitung für Steuerkommandos. Um den Zustand der Soundkarte abfragen zu können, wäre zusätzlich noch eine RX Leitung sinnvoll.

In der Realität könnte man eine solche RS232 Verbindung nachbauen. Schliesslich würde sie aber keiner nutzen wollen, denn man würde mindestens drei serielle Schnittstellen auf einem Rechner blockieren. Desweiteren müsste man jedesmal darauf achten, dass man mit dem Gerät genau die richtigen Leitungen vom Computer aus verbindet. An dieser Stelle kommt eine wichtige Funktion von USB ins Spiel. Mit USB kann man sich virtuell die Leitungen so legen, wie man sie benötigt.

Damit ein Computer weiss, welche Leitungen ein Gerät hat, fragt er dies einfach nach dem Anstecken ab.



3. Endpunkte

An dieser Stelle wollen wir die Namen TX und RX vergessen und dafür den Namen Endpunkt verwenden, denn so werden die Datenübertragungskanäle eines USB-Gerätes genannt. Bei USB muss jedem Endpunkt eine Richtung und Adresse zugewiesen werden. Eine Adresse gibt es bei der klassischen RS232 Verbindung nicht, denn jeder Datenkanal (sprich RX, TX, CTS, RTS usw...) hat eine separate Leitung. Hier liegt der wohl grösste Unterschied zur klassischen RS232 Verbindung: USB ist, wie der Name

schon in sich trägt, ein Bus. Auf einem Bus werden immer nur Datenpakete übertragen. Daher muss jeder Endpunkt in einem Gerät ebenfalls eine Adresse haben. Weil jedes Gerät nach dem anstecken vom Betriebssystem eine Geräteadresse bekommt, kann man über die Geräte- und Endpunktadresse direkt diese Datenübertragungskanäle ansteuern.

USB Controller (kurzer Ausflug):

Will man Daten über eine Netzwerkverbindung senden, so macht man sich keine Gedanken darüber, wie man die einzelnen Kupferleitungen schalten muss, so dass die Daten richtig übermittelt werden. Genauso ist es bei USB. Bei Verwendung von USB sollte man mit dieser Problematik in der Regel nicht konfrontiert werden, denn sie wird vom USB Controller übernommen. Hier gibt es ein großes Feld an Controllern, beginnend mit sehr einfachen, die gerade einmal das Signal von Busleitungen generieren, bis hin zu sehr grossen komplexen Controllern mit internen Konfigurationsregistern für die Endpunkte, oder internem Speicher für ein- und ausgehende Daten u. v. m.

Die meisten Controller weisen intern eine verschiedene Anzahl von kleinen FIFO Speichern auf (meist bis zu 64 Byte). Diese FIFO Speicher kann man direkt einem Endpunkt zuweisen (für ein- und ausgehende Daten). In der Endpunktdefinition muss die FIFO Tiefe als Parameter angegeben werden.

Besprochen wurden bisher die Parameter Tiefe des FIFOs, die Adresse des Endpunktes und dessen Richtung. Zusätzlich gibt es einen weiteren Parameter, die Transferart. Für verschiedene Anwendungen bietet USB

bereits direkt die passenden Transferarten an. Dadurch muss man keine eigenen Algorithmen schreiben, welche die Daten auf Korrektheit überprüfen.

Bulk-Transfer

Der Bulk-Transfer wird am meisten genutzt. Es können grosse und zeitkritische Datenmengen übertragen werden. Zusätzlich überprüft dieser Transfer stets die Korrektheit der Datenübertragung.

Interrupt-Transfer

Diese Übertragungsart darf man nicht falsch verstehen. USB ist und bleibt ein Single Master Bus. Das heisst, dass nur der Master jegliche Kommunikation initiieren kann. Kein Gerät kann sich beim Master selbst anmelden und ihm mitteilen, dass es Daten übertragen will. Der Master muss zyklisch alle Geräte nach neuen Daten abfragen. Im Grunde ist der Interrupt Transfer nichts anderes als der Bulk-Transfer mit dem Unterschied, dass die Interrupt Endpunkte eine höhere Priorität und mehr Bandbreite bekommen. Auf diese Weise kann der Master immer zu dem gewünschten Zeitpunkt auf das Gerät zugreifen, selbst wenn gerade viel Datenverkehr auf dem Bus ist. Diesen zyklischen Zugriff stellt man über ein Pollingintervall in Millisekunden ein.

Isochron-Transfer

Mit dem Isochronen Modus kann man Daten übertragen, die eine konstante Bandbreite erfordern. Typische Anwendungsbeispiele sind die Übertragung von Audio oder Videosignalen. Geht hier ein Bit oder Byte verloren, äussert sich das nur in einem Knacken oder Rauschen. Würden die Daten aber verzögert ankommen, wäre die Sprache oder das Bild völlig verzerrt und daher unbrauchbar.


Man muss also genauso wie beim Interrupt-Transfer das Pollingintervall definieren, und angeben, wie oft der Master Daten abholen oder versenden soll.

Pro Endpunkt muss definiert werden:

1. Richtung
2. Adresse
3. FIFO Tiefer
4. Übertragungsart
5. Pollingzyklus

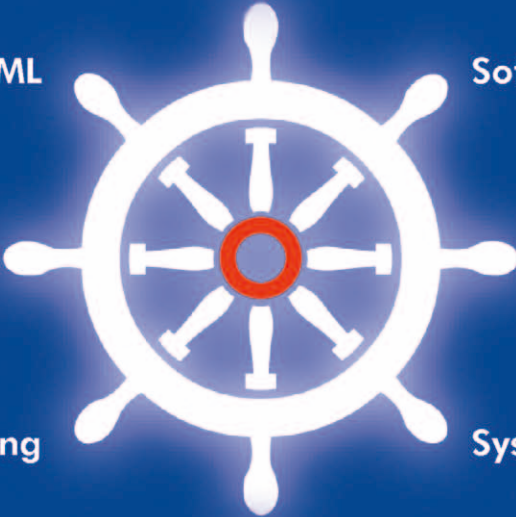
Jeder eigene Endpunkt kann mit diesen Parametern frei definiert werden. Die Daten, die man später über die Endpunkte sendet oder empfängt, sind selbst definierte Datenpakete.





Quategra

Softwareentwicklung



Quategra GmbH
Karl-Heine-Str. 99
D-04229 Leipzig

www.quategra.de
info@quategra.de

T. +49 341 49 12 335
F. +49 341 49 12 336

Embedded UML

Softwarequalität

Echtzeit Betriebssysteme

Sicherheitskritische Systeme

Hardwareentwicklung

System Architektur

LCD-GUI-Design		Training	
Intuitiver - Flexibler - Schneller mit dem		Kompaktschulung ARM Embedded	16.-20.6.2008 22.-26.9.2008
Embedded Display Builder		Softwareentwicklung nach IEC 61508-3	29.-30.9.2008
CASE Tool	Schnellere GUI-Entwicklung Mehrsprachigkeit PC-Simulator	Tag 1-3	ARM Mikrocontroller und Embedded-C
Tag 4-5	KEIL RTX Real-Time Kernel KEIL RL-ARM Realview® Library	2. Tage	Sicherheitslebenszyklus Sicherheitsanforderungen Planung, Entwurf, Entwicklung Funktionale Beurteilung
<ul style="list-style-type: none"> ■ Treiber für diverse LCD-Controller ■ Effizienter C-Code <p style="margin: 0;">www.embedded-gui.de</p>	<ul style="list-style-type: none"> ■ ARM7 und ARM CORTEX-M ■ Praxisbeispiele mit der KEIL RealView MDK ■ Softwarearchitektur und RTOS Grundlagen 	<ul style="list-style-type: none"> ■ Vorgehen nach dem V-Modell ■ Praxisbeispiel mit ausgewählten Methoden 	

Es gibt nur einen einzigen Endpunkt, der anders arbeitet, der Endpunkt 0. Er wird vom Betriebssystem benötigt, um das USB Gerät zu konfigurieren. Über ihn werden von der

USB Spezifikation definierte Nachrichten gesendet. Der Endpunkt 0 ist auch der einzige, der in zwei Richtungen betrieben werden kann, und nicht wie die anderen nur

in eine Richtung. Zusätzlich gibt es für den Endpunkt 0 eine eigene Übertragungsart, den Control Transfer, der auch nur vom Endpunkt 0 unterstützt wird.

4. Interface

Ein Interface ist ein Bündel an Endpunkten. Ein Gerät kann mehrere Interfaces anbieten. So kann eine Soundkarte ein Interface für den Mono- und eines für den Stereobetrieb haben. Das Interface für den Monobetrieb hat einen Endpunkt für die Steuerkommandos und einen weiteren für die Daten, die über einen Lautsprecher ausgegeben werden. Das Interface für den Stereobetrieb hat ebenfalls einen Endpunkt für Steuerkommandos, aber zwei für die Signalausgabe (linker und rechter Kanal). Die Software auf dem PC kann jederzeit zwischen den Interfaces hin- und herschalten. Oft liest man auch den Begriff

Alternate Interface. Dieses Interface kann man parallel zu einem anderen Interface definieren. Definiert man ein normales Interface, so gibt man dort die Endpunkte, die zu ihm gehören, an. Entsprechend der FIFO Grösse eines Endpunktes wird die entsprechende Bandbreite auf dem USB Bus reserviert.

Die Bandbreite wäre auf diese Weise sehr schnell aufgebraucht, auch ohne dass Kommunikation auf dem Bus stattfindet. Würde man aber die benötigte Bandbreite immer nur kurz vor dem Senden oder Empfangen

reservieren, könnte man viel mehr Geräte über einen Bus bedienen. Daher wurde das Alternate Interface erfunden. Zu jedem Interface kann es also ein alternatives Interface geben. Die Endpunktstruktur sollte genauso aussehen wie die vom normalen Interface. Der einzige Unterschied ist der, dass überall als FIFO Grösse 0 Byte angegeben ist. Gibt es jetzt ein Alternate Interface, aktiviert das Betriebssystem beim Einstecken erst dieses, und nimmt so nicht voreilig anderen die Bandbreite weg. Kurz vor dem Senden und Empfangen wird dann auf das eigentliche Interface gewechselt.

5. Konfiguration

Genauso wie Interfaces kann ein Gerät mehrere Konfigurationen haben. Hier geht es um die elektrischen Eigenschaften. Bei USB kann können die Geräte direkt über das USB Kabel mit Strom versorgt werden. So kann man von einem Bus max 5V und

500mA beziehen. Bevor ein Gerät den Strom nutzen kann, muss es beim Master erfragen, ob noch genügend freie Kapazitäten vorhanden sind.

In einer Konfiguration muss man folgende Parameter definieren:

1. Stromaufnahme in 2 mA Einheiten
2. Attribute (z.B. Bus-powered, Remote- Wakup-Support)
3. Anzahl der Interfaces unter dieser Konfiguration

6. Deskriptoren

Jetzt wissen wir, wie man Endpunkte definiert und diese in Interfaces anordnet. Ebenfalls können wir verschiedene Interfaces einer Konfiguration zuweisen. In der Konfiguration werden dazu noch Parameter für den Stromverbrauch und Anschluss definiert. Alle diese Informationen sind immer in Datenstrukturen verpackt, die von der USB Spezifikation Deskriptoren genannt werden. Ein Deskriptor ist nichts anderes als ein Speicherarray, an dem jede Stelle für einen bestimmen Parameter steht. Ein Gerät muss intern an irgendeiner Stelle einen Speicher haben, in dem diese Strukturen liegen, denn das Betriebssystem kann jederzeit diese Informationen abfragen.

Es gibt also einen Endpunkt-Deskriptor, Interface-Deskriptor, Konfigurations-Deskriptor und einen Geräte-Deskriptor.

Die Parameter der ersten drei kennen wir bereits. Kommen wir nun zum Geräte-Deskriptor. Der Geräte-Deskriptor muss in jedem Gerät vorhanden sein. Hier ist definiert:

USB Version

USB Version, die das Gerät unterstützt (z.B. 1.1)

Klassen- / Subklassen- / Protokoll-Code

Das USB Konsortium hat nicht nur den USB Bus definiert, sondern gibt auch Beschreibungen von Endpunkt Bündeln für Geräte heraus. So können Betriebssysteme Standardtreiber anbieten. Mehr zu dieser Technik ist im Bereich USB Klassen dieses Dokuments zu finden.

FIFO Tiefe von EP0

Tiefe des FIFOs, der für den Endpunkt 0 zuständig ist. Dieser ist bei USB 1.1 meist 8 Byte tief.

Hersteller Nummer

Jeder Hersteller von USB Geräten muss sich bei www.usb.org registrieren. Dafür bekommt man dann eine eindeutige Nummer, die für die Treibersuche vom Betriebssystem von Bedeutung ist.

Produkt Nummer

Die Produktnummer wird ebenfalls (wenn sie definiert ist) vom Treiber verwendet, um das Gerät eindeutig zu identifizieren. Mehr zu diesen Nummern kann man im Bereich Plug and Play - Geräteerkennung erfahren.

Versions Nummer

Versionsnummer für das Gerät

String Index für Hersteller

Hier kann ein Name für den Hersteller angegeben werden, der vom Betriebssystem angezeigt werden kann.

String Index für Produkt

Hier kann ein Name für das Produkt angegeben werden.

String Index für Seriennummer

Und hier eine Seriennummer.

In dem Gerätedeskriptor wird nicht direkt der Name für Hersteller, Produkt oder Seriennummer gespeichert, sondern nur eine Nummer eines sogenannten String-Deskriptors. Er ist wiederum eine einfache Datenstruktur im USB Gerät, in dem dann die einzelnen ASCII-Buchstaben stehen.

Anzahl der Konfigurationen

Das ist die Anzahl der vorhandenen Konfigurationen für das Gerät. Eine Kamera könnte hier zwei Konfigurationen haben. Es gibt eine Konfiguration in der die Kamera, die den Strom vom USB Bus bezieht und eine, von der sie den Strom aus den eigenen Batterien bekommt.

7. Plug and Play - Geräteerkennung und -zugriff

Dadurch, dass im USB Gerät alle Eigenschaften mit den Deskriptoren gespeichert sind, kann das Betriebssystem direkt nach dem Anstecken viele Details von dem Gerät erkennen. Es kann z.B. dem Nutzer anzeigen, dass es ein Gerät X vom Hersteller Y gefunden hat.

Wie genau sieht eigentlich der Ablauf dahinter aus?

Kommen wir auf den Endpunkt 0 zurück. Über den EP0 werden definierte Nachrichten gesendet. Mit diesen Nachrichten kann das Betriebssystem alle definierten Deskriptoren abfragen. So gibt es z.B. eine Nachricht Get Descriptor, um einen beliebigen Deskriptor abfragen zu können, oder Get Configuration, um das Gerät nach der aktuellen Gerätekonfiguration zu fragen. Wenn man auf einen Endpunkt zugreifen

muss, dann immer über die Geräte- und Endpunktadresse.

Wie kommt aber ein Gerät zu einer Geräteadresse?

Direkt nach dem Anstecken hat das Gerät die Adresse 0. Der Master erkennt, dass ein neues Gerät eingesteckt worden ist, und sendet an dieses Gerät die Anfrage Get Descriptor? (Gerätedeskriptor). Daraufhin antwortet das Gerät mit dem entsprechenden Deskriptor. Jetzt weiss der Master, dass es sich um ein echtes USB Gerät handelt, und ordnet ihm eine neue endgültige Adresse zu. Ab diesem Zeitpunkt ist das Gerät über die neue Adresse erreichbar.

Nur woher soll der Programmierer wissen, welche Adresse das USB Gerät später auf einem Computer hat? Die Adresse wird

schliesslich nach der Anzahl der angesteckten Geräte erhöht.

Wenn man eine Verbindung zu einem USB Gerät aufbaut, kann man dies nicht über einen eindeutigen Punkt (wie z.B. c:/ oder /dev/hda0 bei einer Festplatte) machen. Die Prozedur ist immer folgende: Das Betriebssystem legt in einer eigenen internen Struktur alle abgefragten Deskriptoren von den angesteckten USB Geräten ab. Sucht man eine Adresse für ein Gerät, muss man in den Datenstrukturen nach dem dazu passenden Gerät suchen. Dies kann man entweder über die Hersteller- und Produkt-Nummer machen, oder über einen Klasse- / Sub- / oder Protokoll-Code. Falls ein Gerät nur über einen Stringdeskriptor beschrieben wird, kann man die Adresse auch über diesen ermitteln.

8. USB Klassen

Das USB Klassenmodell soll die Entwicklung von Treibern erheblich vereinfachen. Die Idee dahinter ist ganz einfach. Mit den Deskriptoren beschreibt man das Aussehen der Schnittstelle. Welche Controller und Techniken dahinter stehen ist aus der USB Sicht unerheblich. Wenn wir unsere Soundkarte ansehen, so haben wir nur gesagt, dass ein bestimmter Endpunkt für die Ausgabe der Soundsignale für den rechten Stereokanal da ist. Was genau mit dem Byte Strom hinter der USB Schnittstelle passiert, ist dem USB Treiber nicht wichtig. Er schickt nur die definierten Bytes, die das Audiosignal widerspiegeln. Und das ist die wesentliche Idee hinter den USB Klassen. Es sollen für Geräte mit gleichen Merkmalen und Eigenschaften Gruppen von Interfaces und Endpunkten definiert werden. Betriebssysteme können für diese Geräte Treiber anbieten, da sie kein einziges spezielles Register oder Zeitverhalten irgendeines Controllers kennen müssen. Der Treiber muss nur so geschrieben werden, dass er die Daten richtig formatiert an die Endpunkte

verteilt und abholt. Dies entlastet Hersteller typischer PC Komponenten (Tastatur, Maus, Soundkarte, Scanner, Drucker, ...) von der Bereitstellung spezieller Treiber.

Leider funktioniert diese Idee nur selten. Einzig bei Tastaturen und Mäusen nehmen die Hersteller an diesem Konzept teil. Irgendetwas hält die Hersteller davon ab, sich an diesen Standard zu halten. Sie entwickeln lieber eigene proprietäre Treiber.

Es gibt Klassenspezifikationen für:

1. Tastaturen, Mäuse, ... (Human Interface Device Class)
2. Soundkarten (Audio Device Class)
3. Kommunikationsschnittstellen z.B. RS232, Ethernet (Communication Device Class)
4. Content Security Class
5. Chip-/Smart Card Device Class
6. IrDA Bridge Device Class
7. Imaging Device Class
8. Printer Device Class

Für die meisten Klassen gibt es in allen bekannten Betriebssystemen Standardtreiber.

Wenn man einen solchen Standardtreiber verwenden will, muss man im Geräte-Deskriptor die richtigen Klassen- und Protokoll-Codes angeben. Man kann sogar einem Interface eine Klasse zuweisen. In der Praxis bedeutet dies z.B. bei einem dieser bekannten Multifunktionsgeräte (Fax/Kopierer/Scanner/Drucker), dass es für jedes virtuelle Gerät ein Interface mit dem entsprechenden Klassencode gibt. Im Idealfall muss man sich nur um die Firmware im Gerät kümmern, da man keinen einzigen Treiber selbst schreiben muss.

In diesem Sinne hoffe ich, dass die USB-Thematik etwas klarer geworden ist, und wünsche viel Spaß beim Basteln.

9. Glossar

FIFO-Speicher:

(engl. etwa Erster rein - Erster raus), häufig abgekürzt mit FIFO, gleichbedeutend mit First-Come First-Served bzw. FCFS, bezeichnet jegliche Verfahren der Speicherung, bei denen diejenigen Elemente, die zuerst gespeichert wurden, auch zuerst wieder aus dem Speicher entnommen werden (entnommen aus [<http://de.wikipedia.org/wiki/FIFO>]).

Polling:

Polling bezeichnet in der Informatik die Methode, den Status von bestimmter Hard- oder Software mittels zyklischem Abfragen zu ermitteln.

EPJ
No. 2

[EDITORIAL] Benedikt Sauter

EPJ

Embedded Projects Journal

Das Journal

Embedded Projects Journal
Open Source und Hardware Magazin - 100%
frei, kostenlos und offen!

Das Motto:
Von der Community für die Community!

Mit dem Embedded Projects Journal soll ein
freies Magazin für Open Source Hardware
Projekte entstehen. Alle im Journal vorgestellten
Artikel müssen 100% frei und offen sein (ohne
Ausnahme). Das Magazin soll idealerweise
viermal pro Jahr erscheinen.

Es gibt ein PDF zum Herunterladen und
zusätzlich soll das Journal kostenlos als
gedruckte Auflage über die Internetseite
zu bekommen sein. Das Ganze funktioniert
folgendermaßen: für jede Ausgabe werden
Sponsoren für Druck und Porto gesucht (für
die erste Ausgabe ist dies bereits geschehen).
Die Sponsoren bekommen als Dankeschön
eine Werbefläche im Magazin.

Artikelautoren und Projektinitiatoren gesucht!

Hast du Lust einmal etwas über ein inter-
essantes Thema zu schreiben, oder suchst
du als Projektinitiator weitere Leute für ein
interessantes Projekt?

Schreibe die Idee in ein paar Zeilen an
sauter@embedded-projects.net und lass
uns gemeinsam sehen, was wir draus ma-
chen können.

Interesse als Sponsor?

Wie bereits beschrieben, lebt das Konzept
von Sponsoren, die das benötigte Geld
für den Druck und Versand bereitstellen.
Bei Interesse, sich an dem Projekt zu be-

teiligen, nehmen Sie am besten Kontakt
mit _____ auf.



Die nächste Ausgabe / Nr.2
September 2008

embedded-projects.net
JOURNAL
OPEN SOURCE SOFTWARE AND HARDWARE PROJECTS

Artikelvorschau:

- **Ein Melodiegenerator**
aus Logikbausteinen
- **Flexible USB-Schaltung**
Octopus + Experimentierplatine
- **SimpleCPU -**
Entwurf einer einfachen
CPU in VHDL

100%
OPEN SOURCE
FOR FREE
embedded projects
JOURNAL
PDF + PRINT



ARM Boards

ARM USB Debugger (ARM-USB-Tiny)

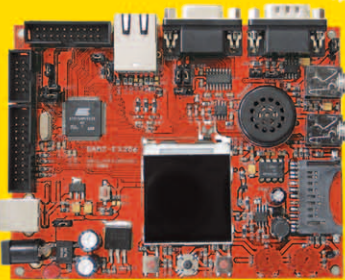
ARM7/ARM9/Cortex-M3 und XScale Debuginterface für OpenOCD und CrossWorks



ARM USB Debugger
45,00 €
embedded-projects SHOP

AT91SAM7X256 + TFT + Ethernet (SAM7-EX256)

Bestückt mit einem 32 Bit ARM-Mikrocontroller mit 256 kB Flash, 64 kB RAM, 55 MHz, Ethernet 10/100, USB 2.0, RS232, CAN, MMC/SD-Card Slot und TFT-Display



AT91 SAM7X256 TFT+ Ethernet
120,00 €
embedded-projects SHOP

MSP430 Boards

MSP430F1611 Adapterplatine (MSP430-H1611)

MSP430F1611 mit 48K Bytes Programm Flash, 256 Bytes Daten Flash, 10K Bytes RAM



MSP430F1611 Adapterplatine
25,00 €
embedded-projects SHOP

MSP430 USB JTAG Adapter (MSP430-JTAG-TINY)

Programmierung/ Debugging für alle MSP430Fxxx Flash-Microcontroller



MSP430 USB JTAG Adapter
65,00 €
embedded-projects SHOP

AVR Boards

Atmel AVRISP mkII (USB)

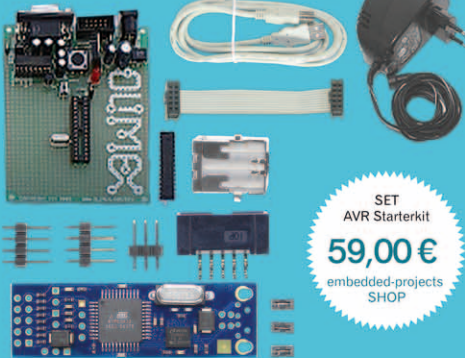
Original AVRISP mkII In-System Programmer von Atmel.



Atmel AVRISP mkII
40,00 €
embedded-projects SHOP

AVR Starterkit (inkl. USBprog, Netzteil und ATMega8)

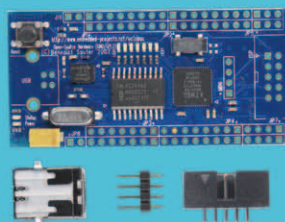
AVR-Starterkit bestehend aus USBprog, AVR-Starterplatine, ATMega8 und Steckernetzteil.



SET AVR Starterkit
59,00 €
embedded-projects SHOP

Octopus USB

Octopus bietet viele bekannte Schnittstellen aus der Mikrocontrollerwelt über ein einfaches USB Gerät an.



Octopus USB
39,00 €
embedded-projects SHOP

PIC Boards

PIC Entwicklungsplatine (PIC-P28-USB)

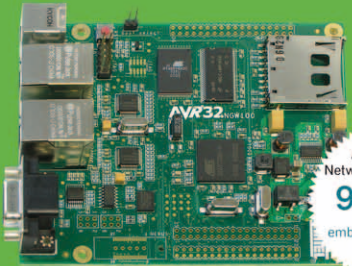
PIC Mikrocontroller Entwicklungsboard für 28-polige ICs + USB RS232.



PIC Entwicklungsplatine (PIC-P28-USB)
25,00 €
embedded-projects SHOP

AVR32 Boards

ATNGW100 Network Gateway Kit

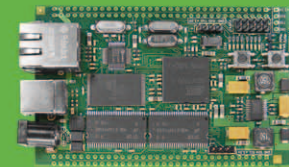


ATNGW100 Network Gateway Kit
90,00 €
embedded-projects SHOP

Grasshopper AVR32-Board

*Open-Source-Version (ohne CD+Kabel)

freie Plattform für die AVR32 Entwicklung



Grasshopper AVR32-Board*
ab **85,00 €**
embedded-projects SHOP



Jetzt schneller zum Shop:
www.eproo.net

- großes Sortiment an Evaluations- und Testboards
- bekannte Open Source Projekte
- übernommener Artikelbestand von www.mikrocontroller.net
- faire Preise
- Produkte direkt vom Hersteller
- bequeme Zahlungsabwicklung und schneller Versand



Jetzt Neu:
Versand weltweit

4,- Euro



Erfolg

sieht anders aus!

Sie wollen richtig mitmischen?

Ihre Innovationen sind kein kalter Kaffee, und dennoch schaffen Sie den erfolgreichen Marktaufstieg nur schleppend?

*Ihre Ideen, Ihre Leistungen, Ihre Produkte -
Sie bieten professionelle Lösungen auf höchstem Niveau!*

Transportieren Sie Kompetenz und know-how auch im allgemein, wahrnehmbaren Bereich - Ihrer Firmenwerbung. Optimale Marktpositionierung durch klare Reflexion des Kundennutzens. Überzeugen Sie - noch vor dem ersten Kundengespräch.

Mehrwert durch professionelle Gestaltung.

*Wir sind Ihr Partner für erfolgsorientierte
Außendarstellungen von Unternehmen.*

Integrative Kommunikations- und Kreativansätze zur Markenbildung und Markenführung Ihrer Produkte und Dienstleistungen. Hohe Qualität und Kompetenz zu bezahlbaren Preisen.

→ www.das-medienkollektiv.de

[DAS] **medien KOLLEKTIV**

Werbeagentur Dresden