# Rapid Prototyping Embedded Systems Using Ethernut Boards

Marius Strobl
University of Applied Sciences Regensburg
Regensburg, Germany
Email: marius.strobl@hs-regensburg.de

Norbert Balbierer, Angelika Schingale
Continental Automotive GmbH
Regensburg, Germany
Email: firstname.lastname@continental-corporation.com

***Abstract*—Often it is necessary to be able to rapidly prototype an embedded system to proof a concept or for actually developing a product. Typically, this means that one has the need for an existing microcontroller based design including usual components like a General Purpose Input/Output (GPIO) interface, an IEEE 802.3 compliant Ethernet Media Access Controller (MAC), an Universal Asynchronous Receiver Transmitter (UART) as well as other common peripherals. Plus, ideally, one would like to have the source code of existing firmware for use as a building block besides the hardware itself. In this paper, the family of the Ethernut microcontroller reference design boards and Nut/OS, the real time operating system available for them, are presented. Together, these hard- and software components nicely fit the job. Finally, by giving real-world examples of prototypes that have been built upon this combination, the viability of this approach is demonstrated.**

***Index Terms*—microcontroller, Ethernet, Ethernut, rapid prototyping, embedded systems**

## I. Introduction

When developing a proof-of-concept destined for the embedded world, for example for testing a new peripheral chip, or in the early stages of a product design, one often has the need for existing hard- and software to build upon in order to rapidly craft a prototype. This typically means that one is looking for a microcontroller based reference design, which provides the necessary interfaces like Analog/Digital Converters (ADCs), General Purpose Input/Output (GPIO) pins, an IEEE 802.3 [1] compliant Ethernet Media Access Controller (MAC), Universal Asynchronous Receiver Transmitters (UARTs) for EIA RS-232-C [2] or RS-485 [3], and a watchdog. Besides, there are basic requirements like internal or external volatile (RAM) and non-volatile (NVRAM) random-access memory as well as a flash read-only memory (ROM). For ease of use, there should be a IEEE 1149.1 [4] compliant Joint Test Action Group (JTAG) interface or a Serial Peripheral Interface (SPI) for In-System Programming (ISP) the flash memory and optionally debugging. It certainly also helps, when the source code of a real-time kernel including device drivers for the on-chip interfaces of the microcontroller and those on the reference design board plus stacks for the corresponding communication protocols are available. Nowadays, implementations of the Transmission Control Protocol/Internet Protocol (TCP/IP), as specified in RFC 793 [5] and RFC 791 [6] respectively, are almost mandatory for the latter.

Table I
Microcontrollers and memory available with the Ethernut family of reference design boards

| Model | Microcontroller | RAM [Bytes] | Flash [Bytes] | MAC [Mbps] |
|---|---|---|---|---|
| Ethernut 1 | AVR® 8-bit ATmega128 | 4k int. 32k ext. | 128k | 10 |
| Ethernut 2 | AVR® 8-bit ATmega128 | 4k int. 512k ext. | 128k | 10 |
| Ethernut 3 | ARM7-TDMI 32-bit AT91R40008 | 256k | 4M | 10/100 |
| Ethernut 5 | ARM9 32-bit AT91SAM9XE512 | 32k int. 128M ext. | 512k int. 1G ext. | 10/100 |

Table II
Specific Features of the Ethernut reference design boards

| Model | ADC | GPIO [lines] | NVRAM [Bytes] | $I^2C$, RTC, SD-card slot | UART/ USART |
|---|---|---|---|---|---|
| Ethernut 1 | 8 chan. 10-bit | 22 | 4k | | 0/2 |
| Ethernut 2 | 8 chan. 10-bit | 28 | 4k | | 0/2 |
| Ethernut 3 | | 17+ | 32k (3.0) 4M (3.1) | available | 0/2 |
| Ethernut 5 | 4 chan. 10-bit | 15 | 4M | available | 1/4 |

In the following, the family of the Ethernut microcontroller reference design boards and their accompanying Real-Time Operating System (RTOS) called Nut/OS, which are both developed and built by egnite GmbH with support from the Ethernut community and that nicely fit the bill, are presented.

## II. The Ethernut family of microcontroller reference design boards and Nut/OS

### A. Ethernut board models

The website of the Ethernut Project [7] provides detailed information on the Ethernut family of boards, documentation and links to the community. At the time of this writing in February 2012, the family consists of the members shown in Table I. All of these models have in common that they are equipped with an Atmel® Reduced Instruction Set Computer (RISC) based microcontroller, have a dimension of 98 mm ×

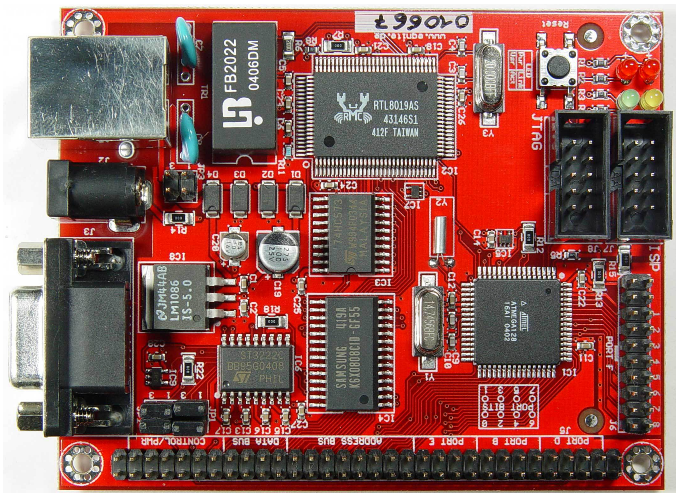
Figure 1. Ethernut version 1.3 Revision G board [12]

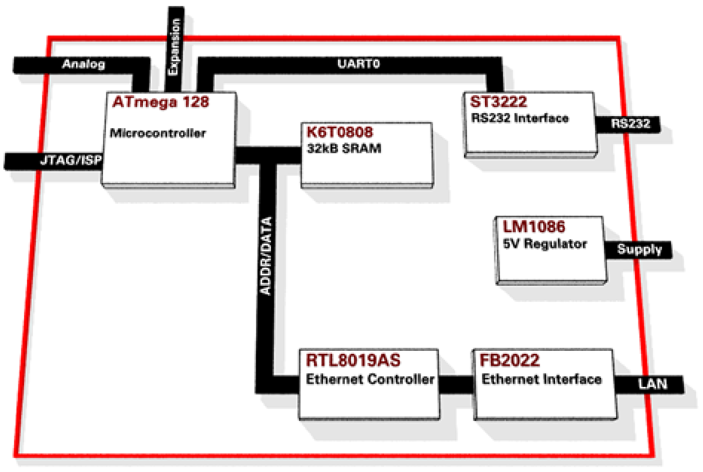

Figure 2. Ethernut version 1.3 function blocks [13]

78 mm × 17 mm and a 64-pin digital expansion port including GPIO pins. The expansion port is largely compatible across the various models, with the main difference that beginning with the Ethernut 3 its voltage level has been lowered from 5 V to 3.3 V. Other components always present on these Ethernut boards are ISP- and/or JTAG-interfaces as well as a watchdog for resetting the board when the software is hung and no longer sends keep-alives to this device. Common to all models also is a separate non-volatile memory for configuration data. However, the sizes and implementations (Electrically Erasable Programmable Read-Only Memory (EEPROM) or serial flash memory) of the latter differ between the models.

Depending on the exact model, there are also specific features like ADCs, an Inter-Integrated Circuit ($I^2C$) interface, a Real-Time Clock (RTC), a Multimedia Card (MMC)/Secure Digital (SD) card slot and an Universal Synchronous Receiver Transmitter (USART) instead of or in addition to the asynchronous-only UART. The majority of these specific features and the exact number of GPIO pins of each model are summarized in Table II. Properties like an image sensor interface, the ability to supply Power Over Ethernet (POE) as per IEEE 802.3af [8], and an Universal Serial Bus (USB) 2.0 interface are only available with the Ethernut 5 boards and therefore not listed in that table.

The Ethernut 5 is also the only board available for the extended temperature range from −25 °C to +85 °C, except for the Ethernut 2, which is tested over the full industrial temperature range from -40 °C to +85 °C.

One unique feature of the Ethernut 5 boards also is that they can run a vendor supplied Linux kernel [9] in addition to Nut/OS. The latter is presented in subsection II-B. As described in [10], the FreeBSD operating system [11] recently has been ported to the Ethernut 5 as well.

It is also noteworthy, that the layouts and schematics of all Ethernut board models are available on the website [7] in form of files for the EAGLE (Easily Applicable Graphical Layout Editor) Printed Circuit Board (PCB) design software [14]. These files are distributed under the following Berkeley Software Distribution (BSD) style license (taken from the Ethernut 1 schematics [15]):

```
Copyright (C) 2001-2004 by egnite Software GmbH.
All rights reserved.
Redistribution and use with or without modification
are permitted provided that the following conditions
are met:
1. Redistributions must reproduce the above
   copyright notice and this list of conditions.
2. Neither the name of the copyright holders nor the
   names of contributors may be used to endorse or
   promote products derived from this hardware
   design without specific prior written permission.
3. This hardware design is provided by egnite
   Software GmbH and contributors as is without any
   warranties.
Ethernut is a registered trademark of egnite
Software GmbH.
```

So when one has, for example, successfully built a prototype based on an off-the-shelf Ethernut board, it is possible to also design a custom product based on it using these layouts and schematics due to this permissive BSD style license.

The model used in the application examples described in section III is the Ethernut 1, which is pictured in Figure 1. Its function blocks are diagramed in Figure 2.

There are also several add-ons intended to be connected to the expansion port of Ethernut boards, like the MediaNut 2.0 hardware MPEG-1/2 Audio Layer III (MP3) player, available. A complete list of these extensions can be found on the website at [7]. However, presenting all of these would exceed the limits of this paper.

### *B. Nut/OS*

As mentioned above, the operating system intended to be run on Ethernut boards, but which is not limited to that platform, is Nut/OS. The website hosting Nut/OS archives for download, its bug tracker and the Subversion source code repository is at [16].

As with the Ethernut board layouts and schematics, the entire C source code of Nut/OS is provided under a BSD style license. So Nut/OS too can be used for developing own products without the need to publish their source code when distributing the software in binary form only. Its features, as taken from the description at [17], are:

*"Nut/OS is based on an intentionally simple RTOS kernel, which provides a minimum of services to run Nut/Net, the TCP/IP stack. It's features include:*

- *Cooperative multithreading*
- *Deterministic interrupt response times*
- *Priority based event handling*
- *Periodic and one-shot timers*
- *Dynamic memory management*

*Main features of the TCP/IP stack are:*

- *Base protocols Ethernet, ARP, IP, ICMP, UDP, TCP and PPP*
- *User protocols DHCP, DNS, SNTP, SMTP, FTP, SYSLOG, HTTP and others*
- *Socket API*
- *Host, net and default routing*

*Nut/OS is a modular operating system. Instead of providing a fixed kernel block, all code is packed in libraries. Only those parts are linked to the final binary image, which are directly or indirectly referenced by the application. This guarantees the lowest possible footprint. [...] Standard C libraries like newlib or avr-libc are supported and allow to write highly portable applications. While direct hardware access and native interrupts are possible in application code, the system offers ready-to-use drivers for a large number of devices, including:*

- *Ethernet controllers*
- *UART devices with RS-232 handshake and RS-485 mode*
- *SPI, I2C and CAN busses*
- *MultiMedia and SD cards*
- *Hardware and software audio codecs*
- *Analog-to-digital converters*
- *Serial flash memory*
- *Realtime clocks*
- *Infrared remote controls*
- *Watchdogs and reset controllers*
- *Character displays*
- *GPIO and interrupt control*
- *modem control*

*Note, that not all drivers may be available on all platforms. The following microcontrollers are actively supported:*

- *ATmega103, ATmega128, ATmega2561, AT90CAN128*
- *AT91SAM7S, AT91SAM7SE, AT91SAM7X*
- *AT91SAM9260, AT91SAM9XE*
- *Gameboy Advance"*

It should also be noted, that Nut/OS largely complies to the Portable Operating System Interface (POSIX®), as standardized in IEEE 1003.1 [18], and applications can be ported between these two with minimal effort.

Table III
SOFTWARE REQUIRED FOR COMPILING NUT/OS AND FOR FLASHING ETHERNUT BOARDS

| Name | Description |
|---|---|
| avr-binutils-2.20_1 | GNU binutils for Atmel® AVR® 8-bit RISC cross-development |
| avr-gcc-4.3.4_2 | FSF GCC 4.x for Atmel® AVR® 8-bit RISC cross-development |
| avr-libc-1.7.0_1,1 | A C and math library for the Atmel® AVR® controller family |
| avrdude-5.10 | Program for programming the on-chip memory of Atmel® AVR® CPU |
| gmake-3.81_4 | GNU version of 'make' utility |

Nut/OS can be compiled on several host operating systems, given that a cross-toolchain, i.e. assembler, compiler and linker, for the targeted microcontroller is installed. Instructions for typical operating systems are provided in the manual available at [19].

When intending to change the Nut/OS source code itself and not only building applications on top of it, using the in-tree build method described below should be chosen.

For developing the examples presented in section III, FreeBSD running on an i386 machine has been used as the host. However, with minor adjustments the following instructions should also apply to using GNU/Linux and other Unix-like operating systems like Apple Mac OS X instead.

In order to be able to build Nut/OS for the Atmel® AVR® based Ethernut boards and for flashing these, the software listed in Table III needs to be installed. After obtaining the Nut/OS source code from [16], either by downloading and extracting an archive or by checking it out from the Subversion repository, it can be compiled and installed as described in the following way using these tools.

```
cd nut
./configure --prefix=~/nutos
gmake all install
```

Afterwards, `~/nutos/bin` or the prefix chosen for the installation respectively has to be added to the `$PATH` environment variable. Additionally, Nut/OS has to be configured for the hardware platform and the tools. Using the in-tree build method, this is done by executing the `nutsetup` script (still in the sub-directory `nut`).

```
./nutsetup
```

For Ethernut 1, flashing with `avrdude` using the `stk500` protocol and compiling with `avr-gcc`, the following choices should be made there.

```
1) Atmel ATmega128
1) Ethernut 1 (10 Mbit Realtek RTL8019AS)
5) 14.7456 MHz
3) stk500
2) avrdude
```

This results in `Makedef`, `Makerule`, `UserConf.mk`,

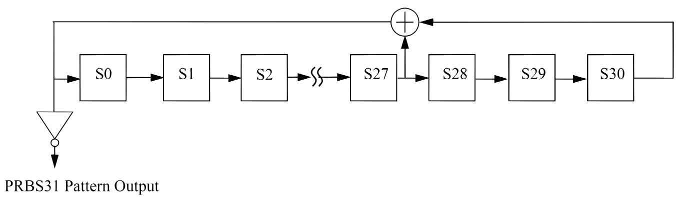

Figure 3. IEEE 802.3 PRBS31 pattern generator [1, section four, p. 271]

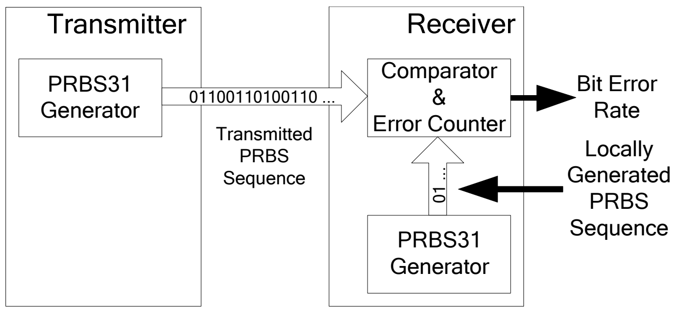

Figure 4. PRBS31-based Bit Error Rate Tester (BERT) (based on [20])

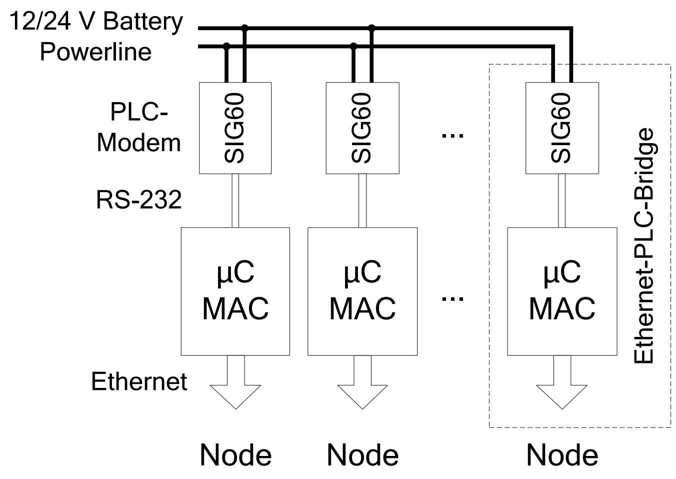

Figure 5. Distributed hub concept of the Ethernet-PLC-bridges

`app/Makedef` and `app/Makerules` to be created. These files should be checked in when using a local repository.

In a final step, Nut/OS has to be compiled as shown below.

```
cd lib
gmake clean all install
cd ..
```

Care must be taken to repeat this step when modifying the Nut/OS kernel and to also link the applications again as the `Makefiles` of the latter do not detect changes to it.

Now, applications in the subdirectory `app` may be compiled in the following way.

```
cd app/<application>
gmake
```

Provided the necessary permissions for `avrdude` are set, the application binary then can be flashed to an Ethernut board using the command below.

```
gmake burn
```

## III. Examples

### *A. Bit error rate tester*

A set of devices that have been built using Ethernut 1 boards and Nut/OS are Bit Error Rate Testers (BERTs) for SIG60 [21] from Yamar Electronics Ltd. The latter are narrowband Powerline Communication (PLC) transceivers for use with the in-vehicle battery powerline. They provide digital communication at data rates up to 115.2 kbps and an RS-232 interface towards the host, in this case an Ethernut board.

In order to qualify these for use in an automotive network, the Bit Error Rate (BER), which is the unitless ratio of the number of incorrect bits divided by the total number of bits received within the studied time interval, had to be determined. For the implementation of the BERT, the Pseudo-Random Bit Sequence (PRBS) test method recommended in IEEE 802.3 [1, section four, p. 269 ff.] for Ethernet link verification has been chosen. This method uses the pseudo-random bit sequence generator of order 31 (PRBS31) shown in Figure 3, which corresponds to the generator polynomial in Equation (1) with the pattern output *inverted.*

$$G(x) = 1 + x^{28} + x^{31} \tag{1}$$

PRBS31 can be implemented using a Linear Feedback Shift Register (LFSR) in hardware or – as in this case – software.

In general, using a PRBS for bit error detection has the advantage, that for the communication channel the resulting stream has the properties of white-noise, i.e. that of a random signal [20]. However, both the transmitter and the receiver can calculate the deterministic PRBS and the latter then may compare the received sequence to the locally generated one for determining the BER. In order do so, the receiver uses the first bit sequence sent by the transmitter to seed its local PRBS31 generator. Provided that this first sequence has been received correctly, the local PRBS31 generators of both the transmitter and the receiver then are synchronized. The general concept of PRBS31-based BER testing is shown in Figure 4.

For testing the BER of PLC modems built using the SIG60 and the influence of the battery powerline attenuation on the quality of the communication, their RS-232 interfaces have been connected to the signal pins of the secondary USART available on the expansion port of Ethernut boards. The BERT application including the LFSR-based PRBS31 generator, the comparator, the BER calculator with data reporting using the primary USART and the SIG60 device initialization have been implemented completely in software on top of Nut/OS.

Given that a rather static test configuration using one Ethernut 1 board as the transmitter and another one as the receiver was envisioned, the BERT has been written to determine the actual role of an Ethernut board – transmitter or receiver – based on its unique Ethernet MAC address. This allowed the same application binary to be used on both Ethernut boards. In total, this application takes about 400 lines of C code.

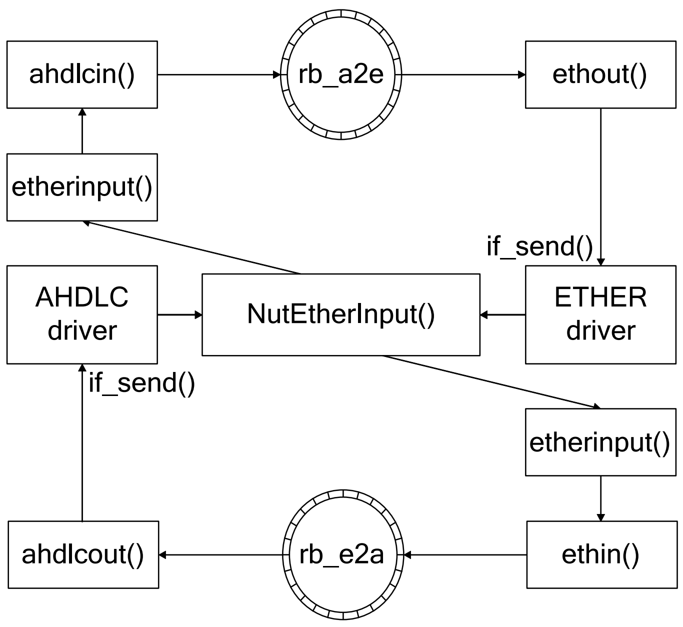

Figure 6. Ethernet-PLC-Bridging NETBUF flow

```
struct _NETBUF {
        NETBUF *nb_next;
        uint8_t nb_flags;
        NBDATA nb_dl;
        NBDATA nb_nw;
        NBDATA nb_tp;
        NBDATA nb_ap;
};
```

Figure 7. Nut/OS NETBUF structure

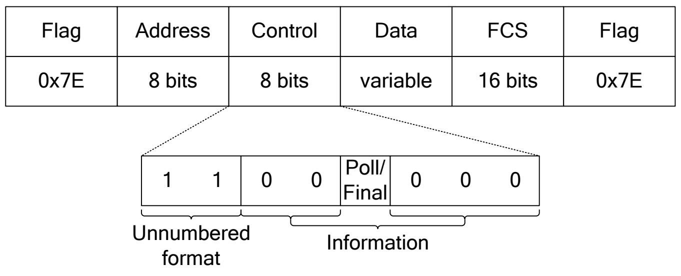

Figure 8. HDLC Unnumbered Information frame format

### B. *Ethernet-PLC-bridges*

Using the aforementioned SIG60-based PLC modems, as part of a research project investigating the possibilities for providing the low speed automotive network in an “All IP/Ethernet Car”, a set of four prototypes of Ethernet-PLC-bridges further described in [22] have been built using Nut/OS and one Ethernut 1 board each. Together, for nodes connected to the Ethernet interfaces of Ethernut boards, these act as a distributed hub on the 12/24 V battery powerline. This general concept of the Ethernet-PLC-bridges is shown in Figure 5.

While for the nodes attached to the Ethernet-PLC-bridges these are transparent, the latter use the “Unbalanced connectionless operation (point-to-point and multipoint)” master/slave principle of the High-level Data Link Control (HDLC) protocol, as standardized by ISO/IEC 13239 [23], in order to communicate with each other and for tunneling the Ethernet payload. Therefore, from the point of view of the Open Systems Interconnection (OSI) model, as standardized by ISO 7637-1 [24], the Ethernet-PLC-bridges as seen by the Ethernet side operate at layer 1 while in fact they communicate on layer 2 over the powerline.

One advantage of using HDLC for this is that Nut/OS already includes an implementation of the Point-to-Point Protocol (PPP) as specified in RFC 1661 [25]. In turn, PPP uses an HDLC-like framing as specified in RFC 1662 [26]. In Nut/OS, the PPP stack is split into the Asynchronous High-Level Data Link Control (AHDLC) driver, which again is based on the UART driver, implementing the framing and a PPP state machine accordingly.

For the implementation of the Ethernet-PLC-bridges, the AHDLC driver has been modified to not depend on the PPP state machine but instead act as a stand-alone network device driver similar to the Ethernet device driver (ETHER). This allows the bridging application to pass the Ethernet and HDLC packets represented by the Nut/OS network buffer structure `NETBUF` shown in Figure 7 around between these two drivers without copying them. Basically, the bridging application, whose concept is shown in Figure 6, works by taking off both the incoming Ethernet and HDLC packets before the Nut/OS TCP/IP stack using its own “Ethernet” input handler `etherinput()`. This hook is registered to the Nut/OS packet dispatcher `NutEtherInput()`. The latter is directly called by the receive methods of the network device drivers. Then, based on the type – HDLC or Ethernet – of the received packets represented by the `NETBUF`s, the application calls either `ahdlcin()` or `ethin()`, which buffer them in one of two ring buffers (`rb_a2e` for AHDLC to ETHER and `rb_e2a` for vice versa). When adding a `NETBUF` to `rb_a2e`, the `ahdlcin()` input handler additionally triggers an event. Likewise for sending payload in the opposite direction, i.e. from the Ethernet to the HDLC side, `ahdlcin()` triggers an event when a slave is polled by the master and therefore may communicate on the powerline. These events cause the output thread, `ahdlcout()` or `ethout()` respectively, to process the packet on the other end by taking it off its ring buffer and sending it via the `if_send()` transmission method of the corresponding network device driver.

The processing of Ethernet packets takes place in `ethin()` and `ethout()`, while the HDLC master/slave handling as well as the encapsulation and decapsulation of the Ethernet payload is implemented in `ahdlcin()` and `ahdlcout()`. Both the master/slave communication and the Ethernet payload transport is done using HDLC Unnumbered Information (UI) elements, whose frame format is shown in Figure 8, over the PLC. For the latter, both the role of the bridge, i.e. the one master, which polls all of the slaves, or slave, and the HDLC address used again is determined based on the Ethernet MAC address of the Ethernut Board. Thus, a single firmware image is sufficient for both types of the Ethernet-PLC-bridges.

The whole bridging application built on top of Nut/OS and consisting of the HDLC master/slave implementation, the packet handling and buffering as well as the initialization of the SIG60 takes about 780 lines of C code, debugging hooks included.

## IV. Conclusion

In this paper, the Ethernut family of microcontroller reference design boards and their real-time operating system Nut/OS have been presented. Due to the fact that these boards include all components typically needed for embedded systems and the maturity of the latter, these allow to rapidly build complex prototypes with minimal effort as shown by the real-world examples given. Moreover, as both the layouts and the schematics of the boards as well as the source code of Nut/OS are provided under a liberal BSD style license, these allow to build actual products based on them without the requirement to pay royalties or to publish the sources needed for manufacturing the hard- and software.

## Acknowledgment

The research leading to the results presented in this paper has been partly funded by the project "Ausbau der Kompetenzpartnerschaft zum Themenschwerpunkt 'IT-Sicherheit' an den Standorten Passau und Regensburg" which is co-funded by the European Union's European Regional Development Funds - Regional Competitiveness and Employment (in German: Europäischer Fonds für regionale Entwicklung - Regionale Wettbewerbsfähigkeit und Beschäftigung).

## References

[1] *Information technology, Telecommunications and information exchange between systems, Local and metropolitan area networks, Specific requirements Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, Institute of Electrical and Electronics Engineers Computer Society Std. 802.3, 2008.

[2] *Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange*, Electronic Industries Association, Engineering Dept. Std. RS-232-C, 1969.

[3] *Electrical Characteristics of Generators and Receivers for Use in Balanced Multipoint Systems*, Electronic Industries Association, Engineering Dept. Std. RS-485, 1983.

[4] *IEEE Standard Test Access Port and Boundary-Scan Architecture*, Institute of Electrical and Electronics Engineers Computer Society Std. 1149.1.1, 1990.

[5] "Transmission control protocol," RFC 793, Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA, Sep. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[6] "Internet protocol," RFC 791, Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA, Sep. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[7] Ethernut project. Website. egnite GmbH. Castrop-Rauxel, Germany. [Online]. Available: http://www.ethernut.de/

[8] *Information technology, Telecommunications and information exchange between systems, Local and metropolitan area networks, Specific requirements Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, Amendment: Data Terminal Equipment (DTE) Power via Media Dependent Interface (MDI)*, Institute of Electrical and Electronics Engineers Computer Society Std. 802.3af, 2003.

[9] The Linux Kernel Archives. Website. Linux Kernel Organization, Inc. San Jose, CA, USA. [Online]. Available: http://www.kernel.org/

[10] M. Strobl, "Running BSD-licensed software on BSD-licensed hardware," in *Proc. EuroBSDcon*, Warsaw, Poland, Oct. 2012, submitted for publication.

[11] The FreeBSD Project. Website. The FreeBSD Foundation. Boulder, CO, USA. [Online]. Available: http://www.freebsd.org/

[12] (2006, Nov.) Ethernut version 1.3 revision G board. Figure. egnite Software GmbH. Castrop-Rauxel, Germany. [Online]. Available: http://www.ethernut.de/img/ethernut13g-large.jpg

[13] (2006, Nov.) Ethernut version 1.3 function blocks. Figure. egnite Software GmbH. Castrop-Rauxel, Germany. [Online]. Available: http://www.ethernut.de/img/enut13_blk3d.png

[14] Cadsoft. Website. CadSoft Computer GmbH. Pleiskirchen, Germany. [Online]. Available: http://www.cadsoft.de/

[15] (2004, Jul.) Eagle 4.11 schematics and layout file for Ethernut 1.3 revision G. egnite Software GmbH. Castrop-Rauxel, Germany. [Online]. Available: http://www.ethernut.de/arc/enut130g.zip

[16] Ethernut embedded Ethernet. egnite GmbH. Castrop-Rauxel, Germany. [Online]. Available: http://sourceforge.net/projects/ethernut/

[17] Nut/OS features. Website. egnite GmbH. Castrop-Rauxel, Germany. [Online]. Available: http://www.ethernut.de/en/firmware/nutos.html

[18] *Information technology, Portable Operating System Interface (POSIX®), Base Specifications, Issue 7*, Institute of Electrical and Electronics Engineers Computer Society Std. 1003.1, 2008.

[19] *Nut/OS Software Manual*, egnite GmbH, Castrop-Rauxel, Germany, Jul. 2009. [Online]. Available: http://www.ethernut.de/pdf/enswm28e.pdf

[20] A. Rasmussen *et al.*, "Framed bit error rate testing for 100G Ethernet equipment," in *Proc. IEEE International Conference on High Performance Switching and Routing (HPSR)*, Dallas, TX, USA, Jun. 2010, pp. 165–168.

[21] "SIG60 - UART over powerline, for AC/DC-BUS network," Datasheet, Yamar Electronics Ltd., Tel Aviv, Israel, 2011. [Online]. Available: http://www.yamar.com/datasheet/DS-SIG60.pdf

[22] M. Strobl *et al.*, "Using Ethernet over powerline communication in automotive networks," in *Proc. IEEE Tenth Workshop on Intelligent Solutions in Embedded Systems*, Klagenfurt, Austria, Jul. 2012, to be published.

[23] *Information technology, Telecommunications and information exchange between systems, High-level data link control (HDLC) procedures*, International Organization for Standardization and International Electrotechnical Commission Std. 13 239, 2002.

[24] *Information technology, Open Systems Interconnection, Basic Reference Model: The Basic Model*, International Organization for Standardization Std. 7498-1, 1994.

[25] Daydreamer, "The point-to-point protocol PPP," RFC 1661, Jul. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1661.txt

[26] ——, "PPP in HDLC-like framing," RFC 1662, Jul. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1662.txt