# EMS <> Adapter <> NetIO <> Raspi

Wer zu den bedauernswerten Menschen gehört, die von IngoF kein Gateway abbekommen haben, muss dennoch nicht verzweifeln. 😉
Mit Hilfe von Pollins AVR NetIO lässt sich für kleines Geld eine passable Schnittstelle zwischen EMS und Ethernet erstellen. Was im Einzelnen zu tun ist, soll hier kurz vorgestellt werden.

N.b.: Selbst Ingos EMS-GW kann mit der Firmware 2.1.1 und einem ENC28J60 Netzwerkmodul verwendet werden. So kann man auch über das EMS-GW das neue Frontend von Moosy benutzen.

## Schema

Vorab eine Schemazeichnung zum besseren Verständnis:

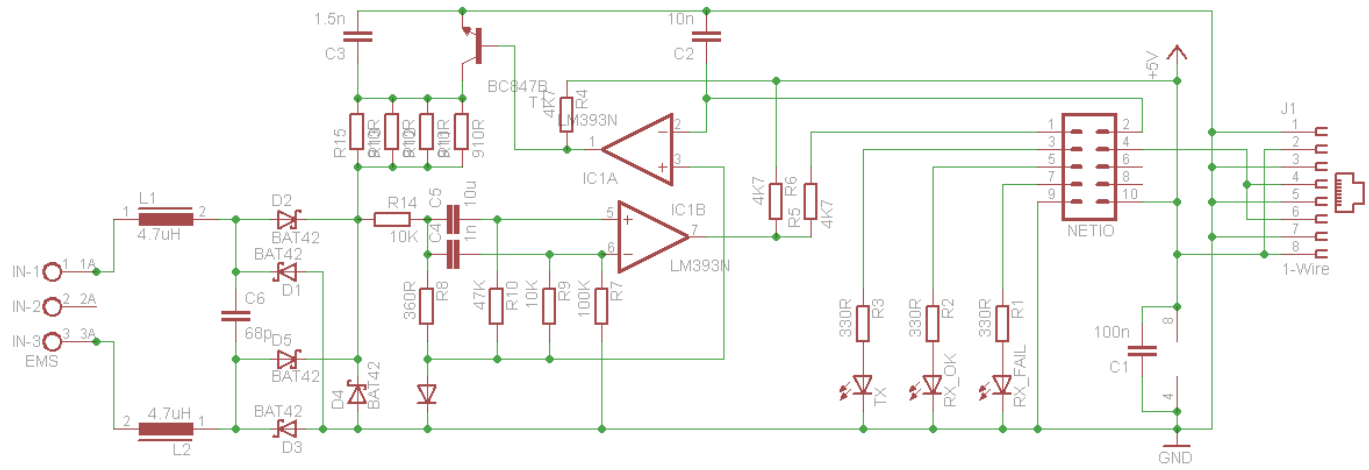
```
+---------+                   +---------+             +-------+
| Buderus |---EMS-RC-(2pol)---| Adapter |---(10pol)---| NetIO |---LAN-->
+---------+                   +---------+             +-------+

+---------+
|  Raspi  |---LAN-->
+---------+
```

## Adapter

### Hardware

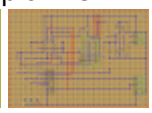
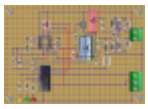
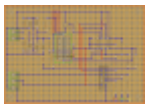
Wie oben bereits angedeutet, braucht es zunächst einen Adapter, der die EMS-Signale für den UART des NetIO aufbereitet. Wir bedienen uns hier Niffkos Schaltplan. Hinzu kommen noch einige Leds zur Signalisierung der empfangenen / gesendeten Telegramme, ein Wannenstecker für die Verbindung zum NetIO und unser neuer Schaltplan liest sich wie folgt:



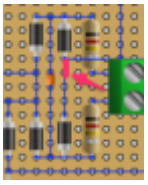

EMS-NetIO-Adapter

Software: Ethersex and Danny Baumanns AddOn

Alle Verbindungen zwischen Adapter und NetIO (Ext) verlaufen über die 2x 5 poligen Pfostenbuchsen. Außerdem wurde die 1-Wire-Schnittstelle des NetIO herausgeführt. Bei diskretem (through hole) Aufbau können die vier parallelen Widerstände im Sendeteil mit jeweils 910R problemlos gegen einen 1-Watt Widerstand mit 220R getauscht werden. Wer Strom sparen will, wählt höhere Vorwiderstände für die Leds.
Eine Eagle-Schaltplanvorlage gibt es hier, eine "Blackboard Breadboard Designer" Vorlage für den diskreten Aufbau auf Lochrasterplatine findet sich hier. Ein Reichelt Warenkorb (Stand: Juni 2014) verbirgt sich hier. Wer den AVR-Net-IO bei Pollin bestellt, bekommt die Bauteile in der Regel auch dort. Die Platinenmaße entsprechen denen des NetIO, so daß die beiden Platinen „gesandwiched“ werden können.
Hier die Vorderseit, Rückseite (nicht gespiegelt!), Vorderseite (Werte) und Rückseite (gespiegelt) der Lochrasterplatine:

(Achtung Fehlerteufel: Die Anoden von D1 und D3 müssen beide mit GND verbunden sein. Siehe Detailbild. Danke für den Hinweis, Lorgru!)




So, oder so ähnlich sieht die  fertige Platine aus. Genügend Platz für überdimensionierte Bauteile aus der Bastelkiste ist jedenfalls vorhanden und die drei Leds haben sich auf dem Photo hinter dem Kabel versteckt.

Maciej Piliński hat sich die Mühe gemacht und eine Eagle-Vorlage entworfen. Näheres samt Download findet Ihr im Thread unter: http://www.mikrocontroller.net/topic/318364#3610939.

# NetIO

## Hardware

Den NetIO gibt es bei Pollin. Der auf dem Bord befindliche ATmega32 wird gegen einen ATmega644P-20PU getauscht. Der hat eine zweite, freie serielle Schnittstelle, die nicht wie USART0 durch einen MAX232 versperrt ist. Die Stromversorgung erfolgt energie- und kostensparend über ein Stecker-Schaltnetzteil mit 5V/1A an der Buchse „J6“ des NetIO. Sonstige Um- und Ausbauten am NetIO, für die es in den einschlägigen Foren unzählige Vorschläge gibt, sind bei einer stabilen Stromversorgung des Boards nach den Erfahrungen des Autors nicht erforderlich.

## Software

Als Software dient Ethersex, den Download gibt's auf GitHub. Im Ethersex-Wiki findet sich die Installations- und Konfigurationsanleitung, so dass wir uns hier kurz fassen können. Danny Baumann, im Folgenden „Danny“ genannt, hat seinen „ems-framer“ bereits in Ethersex integriert, also haben wir es recht einfach. Danke Danny!

Nach Aufruf des Konfigurationsmenüs ***„make menuconfig“*** sollte wie folgt ausgewählt werden:

```
General Setup  --->
  (AVR) Target Architecture
  (ATmega644p) Target MCU
  (16000000) MCU frequency
  (Netio) Hardware/Periphery Class
  [*] Status LEDs  --->
    [*] Status LED (Transmitted)
    [*]    EMS TX
    [*] Status LED (Received)
    [*]    EMS RX OK
    [*] Status LED (Error)
    [*]    EMS RX Failure
  [*] VFS (Virtual File System) support  --->
    [*] VFS File Inlining  --->
      [*] Inline IO
      [*] Inline ADC
      [*] Inline OneWire
      [*] Support Inline SVG
```

```
Network  --->
  Hostname: "EMS"
  (500) Network Buffer Size
  [*] Ethernet (ENC28J60) support  --->
    MAC address: "hier die MAC-Adresse vom Aufkleber des ATmega32 eintragen"
    --- Static IPv4 configuration
      - IP address: "192.168.xxx.xxx"
      - Netmask: "255.255.255.0"
  [*] TCP support
```

```
  [*] ICMP support
```

```
I/O  --->
  (Simple) I/O abstraction model (Port I/O)
  [*] Onewire support  --->
    [*] Onewire device detection support
```

```
Protocols  --->
  [*] ECMD (Ethersex Command) support  --->
    [*] TCP/Telnet
    (2701)  TCP Port
  [*] EMS Support  --->
    (64) EMS Buffer Length
    (7950) EMS TCP Port
    [*] Statistics ECMD
```

```
Applications  --->
  [*] System clock support  --->
    [*] Date and Time support
    Timezone  --->
      (60) Local time offset to UTC (minutes)
      --- Daylight saving
      (60) Time offset (minutes)
      --- Begin of daylight saving
      (3) Month (1-12)
      (5) Week (1-5)
      (0) Day of week (0-6)
      (2) Hour (0-23)
      --- End of daylight saving
      (10) Month (1-12)
      (5) Week (1-5)
      (0) Day of week (0-6)
      (3) Hour (0-23)
    [*] HTTP Server  --->
      (80) HTTP port (default 80)
      (8000) HTTP alternative port (default 8000)
```

```
AVRDUDE configuration  --->
  Falls Ihr mit AVRDUDE arbeitet:
  Euren Programmer auswählen.
  Die Fuses werden wie folgt belegt:
  (e7) Fuse Low Byte (FLB)
  (dc) Fuse High Byte (FHB)
  (ff) Extended Fuse Byte (EFB)
```

Achtung: Die „Status LEDs“ im „General Setup“ sind erst vollständig aktivierbar, nachdem der „EMS Support“ unter „Protocols“ aktiviert wurde.

Nach Abschluß der Konfiguration sollte ein ***„make show-config“*** das Folgende zeigen:

```
MCU: atmega644p
```

```
Hardware: netio

These modules are currently enabled:
====================================
 * ADC * ADC_INLINE * CLOCK * CLOCK_DATETIME * ECMD_PARSER * ECMD_TCP * EMS
* ENC28J60 * ETHERNET
 * HTTPD * ICMP * IPV4 * NET * ONEWIRE * ONEWIRE_DETECT *
ONEWIRE_DETECT_ECMD * ONEWIRE_INLINE
 * PORTIO_SIMPLE * STATUSLED_EMS_RX_FAIL * STATUSLED_EMS_RX_OK *
STATUSLED_EMS_TX * STATUSLED_ERROR
 * STATUSLED_RX * STATUSLED_TX * TCP * UIP * VFS * VFS_INLINE *
VFS_INLINE_INLINESVG * VFS_IO_INLINE
```

Selbstverständlich können weitere Module hinzukonfiguriert werden. ADC, der Webserver (HTTPD), OneWire und VFS sind für EMS nicht erforderlich und können abgewählt werden. Alles ganz nach Belieben.

Vor dem Kompilieren sind noch folgende Änderungen in **„./pinning/hardware/netio.m4“** vorzunehmen:

```
ifdef(`conf_ONEWIRE', `dnl
  /* onewire port range */
  ONEWIRE_PORT_RANGE(PD5, PD5)
')dnl

ifdef(`conf_EMS', `
  pin(EMS_UART_TX, PD3)
')
ifdef(`conf_STATUSLED_EMS_TX', `
  pin(STATUSLED_EMS_TX, PD4, OUTPUT)
')
ifdef(`conf_STATUSLED_EMS_RX_OK', `
  pin(STATUSLED_EMS_RX_OK, PD6, OUTPUT)
')
ifdef(`conf_STATUSLED_EMS_RX_FAIL', `
  pin(STATUSLED_EMS_RX_FAIL, PB0, OUTPUT)
')
```

Die Portbelegung des ATmega644p stellt sich jetzt wie folgt dar:
PD2 = RX
PD3 = TX
PD4 = Led TX
PD5 = 1-Wire
PD6 = Led RX ok
PD7 = frei
PB0 = Led RX fail

Jetzt noch ein ***„make“*** absetzen und fertig ist der Hexfile *„ethersex.hex“*. Der wird jetzt in den ATmega644p gebrannt.

### HexFile

Einen fertigen Hexfile gibt es hier.

**Bitte beachten!** Die einkompilierte IP-Adresse ist 192.168.0.0, die MAC ist FF:FF:FF:FF:FF:FF (Broadcast).

Das solltet Ihr entsprechend ändern! Bei laufendem NetIO geht das per Browser wie folgt:

- http://192.168.0.0/ecmd?mac „neue MAC“ (ohne „“, die MAC Eures NetIO findet Ihr auf dem Aufkleber des mitgelieferten ATMega32)
- http://192.168.0.0/ecmd?ip „neue IP“ (ohne „“).
- http://192.168.0.0/ecmd?gw „Eure Gateway IP“ (ohne „“).

Ein „…/ecmd?help“ zeigt alle verfügbaren Befehle, weitere Hinweise zur „ecmd“ Syntax findet Ihr hier.

Abschließend bitte noch mal die „Fuses“ kontrollieren. Sie sollten wie folgt eingestellt sein:
Fuse Low Byte (FLB) = e7, Fuse High Byte (FHB) = dc, Extended Fuse Byte (EFB) = ff.

Testtool

### Testtool für collector

Es gibt auch ein kleines Testtool um die Verbindung zum EMS-Bus über den NetIO zu testen.

## Raspberry Pi

### Hardware

Ja klar, es muss nicht unbedingt ein Raspberry sein. Aber, der ist klein, stromsparend und bringt alles mit, was man für einen HomeServer so braucht. Wie man den Raspi auf Touren bringt, das findet sich im Netz, z.B. hier. Meine Himbeertorte läuft „headless“, mit WinSCP kann man auch von einer Windows Umgebung problemlos auf den Raspi zugreifen.

### Software

Hier werkelt ein **Raspbian**. Das hat den Vorteil, dass nicht viel konfiguriert und gebastelt werden muss und man sich aus den Wheezy-Quellen bedienen kann. Den Download gibt's hier.

Wenn der Raspi mit der Standardsoftware läuft, holen wir uns noch folgende Bibliotheken:

```
root> apt-get update
root> apt-get upgrade
root> apt-get install build-essential libboost-all-dev mysql-server mysql-
client libmysql++ php5-mysql
```

```
root> apt-get install git telnet php5 php5-cgi gnuplot
```

Als Webserver empfehle ich Lighttpd, der ist klein und dennoch performant. Wie der auf dem Raspi installiert wird, findet sich hier. Die Einrichtung von lighttpd ist in Lighttpd PHP fastcgi configuration beschrieben.

Zum Einsatz für EMS kommt wieder mal Dannys Software, die gibt es per git bei GitHub,

```
git clone git://github.com/maniac103/ems-collector
```

oder als Download.(zip).

Im Verzeichnis „ems-collector[-master]“ findet Ihr die Ordner „collector“, „framer“, „tools“ und „webpage“. Den „framer“ brauchen wir hier nicht, den haben wir mit Ethersex auf den NetIO schon abgefrühstückt.

Im Ordner „collector“ liegen die ems-collector Quelldateien. Um die Datenbankunterstützung mit einzubauen, bearbeiten wir das Makefile, suchen die Stelle, die 'Uncomment the following lines to build the collector with MySQL database support' sagt, und entfernen die Raute am Anfang der 3 Zeilen danach. Danach sollte Ein ***„make“*** die ausführbare Datei „collectord“ erzeugen. Bis der Raspberry das erledigt hat, könnt Ihr Euch eine Tasse Kaffee genehmigen.

Der gerade kompilierte collector muss nach /usr/local/sbin kopiert werden.

## MySql Benutzer anlegen

```
pi> sudo mysql -u root -p
mysql> select password('password');
+-------------------------------------------+
| password('geheim')                        |
+-------------------------------------------+
| *462366917EEDD1970A48E87D8EF59EB67D2CA26F |
+-------------------------------------------+
1 row in set (0.00 sec)

mysql> create user ems identified by password
'*462366917EEDD1970A48E87D8EF59EB67D2CA26F';
mysql> GRANT ALL ON ems_data.* TO 'ems'@'%';
```

hier wird erst der Hash für das angegeben Passwort erstellt. Danach wird dann der Benutzer erzeugt. Die letzte Zeile vergibt die Rechte für die Datenbank.

Nachdem der collectord einmalig erfolgreich gestartet ist, wurden die nötigen Tabellen angelegt. Nun könnte man dem Datenbank-Benutzer ems die Rechte wieder teilweise entziehen.

**Hinweis: Die mysql-Datenbank findet sich im Verzeichnis "/var/lib/mysql/ems_data".**

Falls alles ordnungsgemäß geklappt hat, könnt Ihr auf der Konsole mit einem ***„collectord -h“*** die Startoptionen anschauen.

Für einen ersten Test starten wir den ems-collector mit ***„collectord -u ems -p geheim -f -d all tcp:192.168.xxx.xxx:7950“***, wobei
„ems“ für den mysql-user,
„geheim“ für das mysql-password steht.
Auf der Konsole tanzen jetzt die EMS-Telegramme und eventuelle Debug-Meldungen.

Zur Konfiguration des ems-collector kopieren wir die Datei „ems-collector-master/tools/ems-collector.default“ nach „/etc/default“, benennen sie um in „ems-collector“ und ändern den Inhalt wie folgt, nicht ohne vorher die mysql-Anmeldeparameter zu ändern:

```
# Defaults file for EMS collector daemon
# This is a POSIX shell fragment

# if you need further configuration
# config file location
CONFIGFILE="/etc/ems-collector.conf"

# Serial device file
# SERIALDEVICE="/dev/ttyUSB0"
SERIALDEVICE="tcp:192.168.XXX.XXX:7950"

# Where to put the PID file
PIDFILE="/var/run/ems-collector.pid"

# Other options -- command-port, data-port, db-user, db-pw, rate-limit (s)
to write to db, target
# For debugging purposes insert "-d all=/var/log/ems-collector.log" before
"tcp:...."
# OPTIONS="-C 7777 -D 7778 -u ems -p geheim-r 60 tcp:192.168.xxx.xxx:7950"
OPTIONS=""
```

In dieser Datei können alle OPTIONS-Parameter übergeben werden. Dennoch wird eine zweite Konfigurationsdatei „/etc/ems-collector.conf“ benötigt. Sinnvollerweise übergibt man die OPTIONS-Parameter hier, wie unten gezeigt.

```
ratelimit = 120
#rc-type = rc35
#db-path = 192.168.XXX.XXX:3306
#db-path = localhost:3306
db-user = ems
db-pass = geheim
command-port = 7777
data-port = 7778
```

Werden die OPTIONS in „/etc/default/ems-collector“ übergeben, sollte die „/etc/ems-collector.conf“ leer bleiben. (Hier könnte man auch die Adresse des MySQL angegeben werden. Moosys Frontend muss aber bisher noch auf dem selben Server wie die MySQL-Datenbank liegen.)

Damit der ems-collector nach einem Neustart des Raspberry auch anläuft oder auf Befehle hört, wie: ***„service ems-collector start|stop|restart|condrestart|status“***,
müssen wir noch die Datei „ems-collector-master/tools/ems-collector.init“ nach „/etc/init.d“ kopieren,

dort in „ems-collector“ umbenennen und ausführbar machen. Gegebenenfalls ist in Zeile 21 noch der Pfad anzupassen. Schließlich bindet ein ***„update-rc.d ems-collector defaults“*** den collectord in die Startscripte ein.

Alternativ könnt Ihr Euch auch mit telnet auf den Port 7777 des Rechners, auf dem der Collector läuft, verbinden ***„telnet localhost 7777“***, und darüber Parameter setzen oder auslesen. Einfach mal ***„help“*** eingeben, um zu sehen, welche Befehle unterstützt werden.

## Webseite

Inzwischen, Mitte März 2014, hat Michael Moosbauer (moosy) Dannys collector „aufgebohrt“ und zusätzlich ein tolles Frontend entwickelt. Das findet Ihr mit allen zugehörigen Dateien auf Github.

Scheinbar wird der Frontend von Moosy nicht mehr weiterentwickelt. IngoF kümmert sich jetzt darum. Die beiden aktuellen Versionen findet Ihr in seinem Repo: emstools Github WebinterfaceGithub.

Es gibt auch eine kleine **Installationsanleitung**.

## Finito

Fragen, Anregungen, Ergänzungen und Korrekturen bitte im Thread absetzen oder am besten gleich hier ändern. Falls Ihr Fehler findet, bitte behaltet sie nicht für Euch, sondern gebt Laut. 😉

From:
https://emswiki.thefischer.net/ -

Permanent link:
**https://emswiki.thefischer.net/doku.php?id=wiki:ems:net_io&rev=1479229319**

Last update: **2016/11/15 18:01**