Hausbus for everyone

Ein Gemeinschaftsprojekt

1. Vorwort

Im Zeitalter des Netzes, wo jeder mit jedem und alles mit allem kommuniziert sollte auch das eigene Heim nicht ausgeschlossen werden. Enerie wird teurer und der Mensch immer fauler und komfortabler. Darum brauchen wir noch mehr Elektrogeräte im Haus. Ein Wiederspruch? Nein, diese Geräte sollen uns helfen Energie zu sparen und dabei den alltäglichen Komfort erhöhen.

Wer schaltet den Fernseher, die Stereoanlage, die Set-Top-Box, den Computer, und all die Energiefresser aus, wenn er außer Haus geht? Richtig, niemand. Oft wird auf brennende Lichter vergessen. Die Heizung läuft 24 Stunden, obwohl keiner zuhause ist. Es wird dauernd unnötige Energie verschwendet! Dies belastet nicht nur unsere Geldttasche, sondern auch die Umwelt.

„**Machen wir die Welt ein bisschen smater**“, heißt es in einem Werbespot eines Computer- und Elektronikgiganten. Dieses Projekt soll genau das bewirken. Mit einem Hausbus soll Energie gespart aber dabei trotzdem der tägliche komfort erhöht werden.

Table of Contents

1. Vorwort 1
2. Table of Contents 2
3. Anforderungen 4
   - 3.1. Billig 4
   - 3.2. Energieeffizient 4
   - 3.3. Einfache Installation 4
   - 3.4. Benutzerfreundlichkeit 4
   - 3.5. Erweiterbarkeit 4
   - 3.6. Gemeinsame Dokumentation 4
4. Schematischer Aufbau und Funktionsweise 5
   - 4.1. BUS 5
   - 4.2. BUS Protokoll 5
   - 4.3. Clients 5
     - 4.3.1. Stromsparen an den Clients 5
   - 4.4. Stromversorgung 5
   - 4.5. RS485 <-> Ethernet 5
   - 4.6. Anwender 6
5. Details 7
   - 5.1. RS485 BUS Clients 7
     - 5.1.1. Software 7
       - 5.1.1.1. RS485 Bootloader 7
       - 5.1.1.2. Client Initialisieren 7
         - Header Datei: init.h 7
       - 5.1.1.3. UART 7
         - Anforderungen 7
         - Header Datei: uart.h 7
       - 5.1.1.4. RS485 Bus Protokoll 9
         - Paket Aufbau 9
         - Header Datei: rs485.h 9
       - 5.1.1.5. RS485 Packet Controller 10
         - Header Datei: rs485controller.h 10
       - 5.1.1.6. Vom Bus benötigte Libraries 11
         - Ping 11
         - Pong 11
         - ACK 11
         - NACK 11
         - RESET 11
       - 5.1.1.7. Sensoren und Aktoren Libraries 11
         - Taster und Relay über INT0 11
           - Header Datei: client_switch_int0.h 11
         - Taster und Relay über INT0 11
         - Taster und Relay ohne externen Interrupt 11
         - Phasenschnittdimmer 12
         - PWM Mortorsteuerung 12
         - Schrittmotorsteuerung 12
       - 5.1.1.8. Main Prog 12
     - 5.1.2. Hardwarespezifikationen der Clients 12

5.1.2.1. Stromversorgung 12
5.1.2.2. Mikrocontroller 12
5.1.2.3. Peripherie 12
5.1.3. Hardwarekomponenten 12
5.1.3.1. Relay + Taster 12
Schaltplan 12
Boarddesign 13
5.1.3.2. Phasenschnittdimmer 13
5.1.4. RS485 BUS (Hardware) 13
5.1.4.1. Kabel 13
5.1.4.2. Stecker 13
5.2. Hardware BUS Sniffer 13
5.2.1. Schaltplan 13
5.2.2. Board Layout 13
5.3. NET-IO Board = Schnittstelle und RS485 Master 13
5.3.1. Software 13
5.3.1.1. UART / RS485 Bus Protokoll 13
5.3.1.2. Webserver 14
5.3.2. Hardware 14
5.4. Linux Server, Logic 14

2. Anforderungen

an den Hausbus.

2.1. Billig

Die einzelnen Komponenten sollen für jeden leistbar sein. 10€ für das Material einer Komponente soll nicht überschritten werden.

2.2. Energieeffizient

Das Bussystem im Haus soll Energie sparen. Das heißt, die Elektronikkomponenten dürfen nicht mehr Energie verbrauchen, als dadurch eingespart wird.

2.3. Einfache Installation

Auch ein interessierter Leihe sollte das System installieren können. Sowohl Hardware als auch Software.

2.4. Benutzerfreundlichkeit

Jeder Hausbewohner, ob jung oder alt, muss das intelligente Eigenheim bedienen können. (Smartphone, PC, Userpanel)

2.5. Erweiterbarkeit

Ein einfaches und leistungsfähiges Protokoll sollen jeden Hobbybastler die Gelegenheit bieten ganz einfach verschiedenste Komponenten selbst zu bauen.

Der Hardwarespezialist kann vorhandene Komponenten weiterentwickeln und neue Sensoren und Aktoren hinzufügen.

Die Software kann ganz einfach angepasst, erweitert und individualisiert werden. Dies wird durch definierte Softwareschnittstellen ermöglicht.

2.6. Gemeinsame Dokumentation

Mit einer gemeinsamen Dokumentation und Bedienungsanleitung können viele Leute an dem Projekt Mitarbeiten.

3. Schematischer Aufbau und Funktionsweise

3.1. BUS

RS-485 eignet sich meiner Meinung am besten für die Vernetzung der einzelnen Clients. Multimasterfähig + 32 Clients + via UART vom uC ansprechbar + billige Komponenten + 2 Draht + lange Leitungslängen.

Sollten mehr als 32 Clients benötigt werden kann man die Übertragungsgeschwindigkeit herabsetzen und bis zu 128 Clients dranhängen, oder man nimmt einen uC mit zwei UART's und baut eine Bridge.

3.2. BUS Protokoll

Ein einfaches selbstprogrammiertes Kommunikationsprotokoll würde voll und ganz ausreichen.

Protokoll Paket:
1Byte Typ + 1Byte Dest. Adr. + 1Byte Src. Adr. + 2Byte Length + Data + 2Byte CRC16

3.3. Clients

Jeder Client besteht aus einem MAX485/ADM483 Bustreiber und einem ATMega mit Hardware UART.

- schaltbare Steckdose
- Lichtschalter (Relay + beliebig viele Taster)
- Dimmer
- Rollosteuerung
- Heizungssteuerung
- etc.

3.3.1. Stromsparen an den Clients

Die Clients schlafen grundsätzlich. Das heißt, der ATMega befindet sich im sleep Zustand. Er wird nur geweckt, wenn er gebraucht wird. Mit einem Interrupt an der UART oder mit einem Interrupt von einem Sensor (Taster, ...).

3.4. Stromversorgung

Die Stromversorgung erfolgt Zentral. Zusätzlich zu den 2 Twisted Pair Leitungen vom RS485 Bus kommen noch +13,8 und GND.

Die Busleitung besteht also insgesamt aus 4 Adern.

3.5. RS485 <-> Ethernet

Eine Verbindung vom RS485 Hausbus zum IP Netz ist nötige, damit man einfach und von Überall sein Haus unter Kontrolle hat.

Am einfachsten geht das am Anfang mit einem NET-IO Board von Pollin. Dies sollte aber noch nicht die Schnittstelle zum Anwender (Smartphone, PC) sein. Mit dem NET-IO Board sollten die Clients im RS485 Bussystem angesprochen und ausgelesen werden. Ein Linux Server kommuniziert mit dem NET-IO Board. Es würde sich ein Linksys WLAN Router eigenen, da diese ganz einfach mit Linux geflashed werden können.

Warum brauchen wir einen Zusätzlichen Webserver?
Der Linux Server ist die Schnittstelle zum Anwender.

- Firewall
- VPN
- Php für Benutzerverwaltung
- Möglicherweise gibt es nicht nur ein NET-IO Board im Haus. Es können z.B. auch Webcam's etc. angesprochen werden.
- ...

3.6. Anwender

Der Anwender steuert sein Haus über ein Webinterface. Dieses ist angepasst für Smartphones, PC's und alle anderen Webclients.

4. Details

Es folgt eine detaillierte Schnittstellendefinition.

4.1. RS485 BUS Clients

...

4.1.1. Software

- Die Software muss auf allen ATMegas funktionieren.
- Alle Clients sind Multimaster. Sie können auch ohne Aufforderung senden. z.B. HALLO, wenn sie eingeschaltet werden. Oder eine andere wichtige Mitteilung senden (Feueralarm).
- Auf jede RS485 Paketanfage gibt es eine Antwot (ACK, NACK, PING PONG). Außer bei Broadcasts.
- Eine Kollisionserkennung findet nicht statt. Diese kann Softwaretechnisch implementiert werden.
- Die Software wird für jeden Client exakt die Selbe sein. Im Flash speicher des uC stehen Konstanten, welche die Software verwendet um sich anpassen zu können.

4.1.1.1. RS485 Bootloader

Neue Firmware muss über RS485 aufgespielt werden können (kann später implementiert werden)

4.1.1.2. Client Initialisieren

Betroffene Dateien: init.h, init.c

Der Client wird gestartet und initialisiert. Im Flash speicher stehen seine Eigenschaften.

Header Datei: init.h

```
//Öffentliche Methoden
unsigned int init();
```

4.1.1.3. UART

Betroffene Dateien: uart.h, uart.c

Über die Hardware UART wird der MAX485 Bustreiber gesteuert.

Anforderungen

- Transmit und Receiv Ringbuffer (ca. 32 Byte)
- Interruptgesteuert; Notwendig um den uC aus den Sleepmodus zu holen.
- Einstellungen: 8 bit Data + Stopbit
- MAX485 auf senden / empfangen schalten

Header Datei: uart.h

```
//Buffergröße:
```

```
#define UART_RX_BUFFER_SIZE 32
#define UART_TX_BUFFER_SIZE 32

//MAX485:
#define MAX485_DDR          DDRC
#define MAX485_PORT         PORTC
#define MAX485_BIT          PC5

//Öffentliche Methoden:
/*********************************************************************
Function:  uart_init()
Purpose:   method to initialisize the UART
Input:     none
Returns:   return code
*********************************************************************/
unsigned int uart_init  ();

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int uart_put_c (unsigned char c);

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int uart_get_c (); //High byte error code

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int uart_put_s (unsigned char * s);

//Rückgabewerte:
#define UART_INIT_SUCCESS             0x0001
#define UART_INIT_ERROR               0x0002

#define UART_PUT_C_SUCCESS            0x0003
#define UART_PUT_C_ERROR              0x0004
```

```
#define UART_PUT_S_SUCCESS              0x0005
#define UART_PUT_S_ERROR                0x0006

#define UART_GET_C_FRAME_ERROR          0x0800     //high byte error
#define UART_GET_C_OVERRUN_ERROR        0x0400
#define UART_GET_C_BUFFER_OVERFLOW      0x0200
#define UART_GET_C_NO_DATA              0x0100
#define UART_GET_C_SUCCESS              0x0000
```

4.1.1.4. RS485 Bus Protokoll

Betroffene Dateien: rs485.h, rs485.c

Paket Aufbau

- 2 Byte Type
- 1 Byte Destination Address
- 1 Byte Source Adress
- 2 Byte Length
- x Byte Daten
- 2 Byte CRC16

Header Datei: rs485.h

```
//Paket Struktur:
struct Packet
{
      unsigned int     errorcode;
      unsigned char    type;
      unsigned char    dest;
      unsigned char    src;
      unsigned int     length;
      unsigned char*   data;
      unsigned int     crc;
};

//Öffentliche Methoden:
/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int rs485_init ();

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int rs485_send_packet(  unsigned int type,
```

```
                                    unsigned char dest,
                                    unsined char* data );

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int rs485_receiv_packet (struct Packet * packet);

//Rückgabewerte:
#define RS485_INIT_SUCCESS                  0x0001
#define RS485_INIT_ERROR                    0x0002

#define RS485_SEND_PACKET_SUCCESS           0x0003
#define RS485_SEND_PACKET_ERROR             0x0004

#define RS485_RECEIV_PACKET_SUCCESS         0x0005
#define RS485_RECEIV_PACKET_UART_ERROR      0x0006
```

4.1.1.5. RS485 Packet Controller

Betroffene Dateien: rs485controller.h, rs485controller.c

Der Packet Controller ließt ein Packet ein, kontrolliert es und leitet es je nach Type weiter. Beispiel: es trifft ein Packet vom Type ping ein. Der Controller leitet es nach Überprufung an eine ping Routine weiter.

Header Datei: rs485controller.h

```
//Öffentliche Methoden:

/*********************************************************************
Function:
Purpose:
Input:
Returns:
*********************************************************************/
unsigned int rs485controller_run ();

//Pakettypen:
#define RS485_HALLO                     0x0000

#define RS485_PING                      0x0001
//Call: rs485_ping_received(struct Packet* packet) in ping.h
#define RS485_PONG                      0x0002
#define RS485_ACK                       0x0003
#define RS485_NACK                      0x0005
#define RS485_RESET                     0x0009
#define RS485_ERROR                     0x000E
#define RS485_GET_STATUS                0x000F
#defien RS485_SET                       0x0010
```

4.1.1.6. Vom Bus benötigte Libraries

Ping
Betroffene Dateien: ping.h, ping.c

Auf ein Ping Packet folgt ein folt ein Pong

Pong
Betroffene Dateien: pong.h, pong.c

Pong empfangen. Fertig.

ACK
Betroffene Dateien: ack.h, ack.c

NACK
Betroffene Dateien: nack.h, nack.c

RESET
Betroffene Dateien: reset.h, reset.c

Der Client wird Hardware Reseted. Am besten mit Watchdog.

4.1.1.7. Sensoren und Aktoren Libraries

Je nach dem Type Feld im RS485 Packet wird die REMOTE methode des Sensors / Aktors aufgerufen. Was genau gemacht wird steht in den Daten des Packets.

Taster und Relay über INT0
Betroffene Dateien: client_switch_int0.h, client_switch_int0.c

Header Datei: client_switch_int0.h

```
//Öffentliche Methoden:
unsigned int client_switch_int0_init();
unsigned int client_switch_int0_status();
unsigned int client_switch_int0_switch(unsigned char type);
unsigned int client_switch_int0_remote(struct Packet *packet);
```

Taster und Relay über INT0
Betroffene Dateien: client_switch_int1.h, client_switch_int1.c

Taster und Relay ohne externen Interrupt
(Problem: uc kann nicht in den Sleep Modus gesetzt werden)

Phasenschnittdimmer

PWM Mortorsteuerung

Schrittmotorsteuerung

...

4.1.1.8. Main Prog

Stromsparen mit Sleepmodus.

4.1.2. Hardwarespezifikationen der Clients

- Die Hardware muss in eine Standard-Unterputzdose passen (Druchmesser ca. 55mm)
- Auf den Stromverbauch der einzelnen Komponenten Achten.

Zentrale Bausteine jedes Clients sind ein ATMEL Mikrocontroller und ein RS485 Bustreiber. Welche Typen ausgewählt werden ist egal. Software funktioniert mit allen kompatiblen Bausteinen.

4.1.2.1. Stromversorgung

Eine Spannung >12 Volt wird an jedem Knoten angelegt. Mittelsspannungswandler wird diese auf +5V gewandlt und der uC damit gespeist.

4.1.2.2. Mikrocontroller

Der uC muss folgende Kriterien erfüllen:

- ATMEL kompatibel
- Hardware UART
- Externer Taktgeber (Quarz, Oszilator)

Ich werde für den Anfang einen ATMEGA8 verwenden.

4.1.2.3. Peripherie

- Stromspardende Komponenten verwenden
- kleine Komponenten verwenden
- Sicherheit! Achtung beim Umgang mit 230V, Leiterbahnabstand, Leiterquerschnitt

4.1.3. Hardwarekomponenten

Hier kommen die Hardwarekomponenten hin. Jeder kann sich beteiligen. Einfach efinderisch sein!
Vorschläge. Relay, Taster, Temperatursensor

4.1.3.1. Relay + Taster

Beschreibung

Schaltplan
EAGLE Datei

Boarddesign
EAGLE Datei

4.1.3.2. Phasenschnittdimmer

4.1.4. RS485 BUS (Hardware)

Für die serielle Übertragung am RS485 Bus werden 2 Leitungen benötigt (Receiv, Transmit). Twisted Pair Leidtungen sind nicht vorgeschrieben, aber sie senken die Störanfälligkeit, erhöhen die max. Übertragungsgeschwindigkeit und erweitern die max. Buslänge.

Zusätzlich benötigt man noch zwei Leitungen zur Stromversorgung. Ich würd 18 Volt bzw. 13,8 Volt empfehlen.

Jeder knoten hat zwei Stecker und damit Läuft der Bus direkt über die Platine des Kotens.

4.1.4.1. Kabel

Rx und Tx können über dünne Twisted Pair leitungen übertragen werden.
Für + und GND wird wohl ein größerer Querschnitt benötigt. Vielleicht hat wer erfahrung, oder irgendwer kann's berechnen.

4.1.4.2. Stecker

RJ11 bzw. RJ45 Stecker werden wohl kaum für die Stromversorgung reichen, da die leitungen zu dick sind. Was anderes Überlegen.

Anforderungen: Kleine dimensione. evt. nur Schraubklemmen.

4.2. Hardware BUS Sniffer

Um den Bus debuggen und überprüfen zu können wird ein Hardware Sniffer benötigt. Dieser wird wie ein Knoten am Bus installiert.

MAX485->MAX232->Serielle Schnittstelle des PCs.

Mit einem Terminalprogramm (Hyperterm) können die Daten am Bus visualisiert werden.

4.2.1. Schaltplan

hier kommt der schaltplan her (EAGLE)

4.2.2. Board Layout

4.3. NET-IO Board = Schnittstelle und RS485 Master

4.3.1. Software

4.3.1.1. UART / RS485 Bus Protokoll

Exakt das selbe wie bei den Clients.

4.3.1.2. Webserver

Der Webserver bietet HTML-Seiten zur Sensor- und Aktorsteuerung der Clients an. Er Speichert zustände der Clients und bietet diese ebenfalls an. Die Zustände werden auf Bestellung von den Clients abgerufen.

Auf die Sicherheit muss nicht geachtet werden. Wir befinden uns im Intranet. Firewall, Verschlüsselung und Benutzerverwaltung übernimmt der Linux Server.

Die Kommunikation mit dem Linux server könnte natürlich auch über ein eigenes UDP oder TCP Protokoll erfolgen. Aus Gründen der Einfachheit wird aber HTTP bevorzugt.

4.3.2. Hardware

Für den Anfang reicht das NET-IO Board von Pollin.

4.4. Linux Server, Logic

Sicherheit, Benutzerverwaltung, Firewall, VPN.

Die Logiksteuerung des Hauses ist auch hier Zuhause: Temperaturüberwachung, Energiesparmodis, etc....

Es können Natürlich auch mehrere NET-IO Boards und andere Ethernet Clients (Webcam) verwaltet werden.

Der Endanwender greift auch hier auf die Weboberfläche zu.