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

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.

Als zentrales Kommunikationsprotokoll wird das EIB/KNX verwendet.

2.6. Gemeinsame Dokumentation

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

3. Schematischer Aufbau und Funktionsweise

GEHÖRT VERVOLLSTÄNDIGT UND AKTUALISIERT!!!!!!!
BILDER; ZEICHNUNGEN; DIAGRAMME; ANSICHTEN, etc....

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

EIB/KNX Protokoll

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. Hausbus <-> 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. BUS Clients - 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).
- Kollisionserkennung: CSMA/CA nur auf freien BUS schreiben!!! http://de.wikipedia.org/wiki/Carrier_Sense_Multiple_Access
- Im EEPROM des Controllers wird die Logik sitzen. Verbindungen, etc.

4.1. EEPROM Aufteilung

Im EEPROM des uC's stehen die Eigenschaften des Clients.

```
Byte        EIB address high byte
            EIB address low byte
```

4.2. KNX/EIB Bootloader

Neue Firmware muss über EIB/KNX aufgespielt werden können (kann später implementiert werden)

4.3. Client Initialisieren

Betroffene Dateien: init.h, init.c

Der Client wird gestartet und initialisiert. Im EEPROM Speicher stehen seine Eigenschaften.

4.3.1. Header Datei: init.h

```
//Öffentliche Methoden

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

//Returncodes:
```

4.4. UART for RS485 Bus – OSI Layer 1

Betroffene Dateien: uartRS485.h, uartRS485.c

Über die Hardware UART wird der MAX485 Bustreiber gesteuert.

4.4.1. Anforderungen

- Transmit und Receiv Ringbuffer (ca. 32 Byte)

- Interruptgesteuert; Notwendig um den uC aus den Sleepmodus zu holen.
- Einstellungen: 8 bit Data + Stoppbit
- MAX485 auf senden / empfangen schalten

4.4.2. Header Datei: uartRS485.h

```
//Buffergröße:
#define UART_RX_BUFFER_SIZE 32
#define UART_TX_BUFFER_SIZE 32

//e.g. 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:  uart_put_c()
Purpose:   sends a cahracter to the bus
Input:     char to send
Returns:   standard return code
*********************************************************************/
unsigned int uart_put_c (unsigned char c);

/*********************************************************************
Function:  uart_get_c()
Purpose:   get a char from the bus
Input:     none
Returns:   data byte and a high byte standard return code
*********************************************************************/
unsigned int uart_get_c (); //High byte error code

//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_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
```

4.5. Schnittstelle: Protokoll (KNX, EIB) und Physical Layer

Betroffene Dateien: protocol2physical.h

In unserem fall wird als Bus der RS485 verwendet. Je nach belieben können ander Bussysteme zur übertragung verwendet werden (z.B. CAN, Eigenentwicklung, EIB).

Um die Methoden zu Standartisieren müssen einheitliche Empfangs- und Sendemethoden geschaffen werden. Diese können dann jeweils auf die physikalische Methode weitergeleitet werden.

Diese Alias werden in protocol2physical.h eingetragen.

4.5.1. protocol2physical.h

```
//          Alias:                  Methode:
#define     physical_put_c()        uart_put_c()
#define     physical_get_c()        uart_get_c()
```

4.6. Bus Protokoll (KNX, EIB) – OSI Layer 3

Betroffene Dateien: protocol.h, protocol.c

Siehe auch: freebus.org, KNX, EIB, Wikipedia, etc.

Das Protokoll richtet sich an den EIB/KNX Protokol, sodass die Komponenten miteinander kompatibel sind.

Ein Telegramm besteht aus mindesten 7 Byte und Maximal 23 Byte.

4.6.1. Telagramm Aufbau

- 1 Byte Kontrollbyte
- 2 Byte Source Address
- 2 Byte Destination Address
- 1 Byte DRL
- x Byte Daten. Mindestens 2 Byte, Maximal 15 Byte
- 2 Byte Checksum

4.6.1.1. Kontrollbyte

Beihnaltet das Wiederholungsbit und die Priorytöt des Telegramms

7	6	5	4	3	2	2	0
1	0	R	1	p1	p0	0	0

- R ... Wird ein Telegramm das erste Mal gesendet, so ist das Widerholungsbit = 1. Wird ein Telegramm w iederholt, z.B. w eil ein Empfänger es beim ersten Mal nicht verarbeiten konnte, ist dieses Bit = 0. Somit können die Teilnehmer, die das Telegramm bereits beim ersten Mal korrekt empfangen haben, es bei derWiederholung ignorieren.

- p0, p1 ... Priorität
    a. 0 0 Systemfunktion
    b. 0 1 Alarmfunktion
    c. 1 0 hohe Priorität
    d. 1 1 normale Priorität

4.6.1.2. Source Addres

Die Quelladresse besteht aus 2 Byte, wobei als erstes das Most Significant Byte (MSB) und anschließend das LSB übertragen wird.

7	6	5	4	3	2	2	0	7	6	5	4	3	2	2	0
B3	B2	B1	B0	L3	L2	L1	L0	T7	T6	T5	T4	T3	T2	T1	T0

- B0 bis B3 Bereich 0-15
- L0 bis L3 Linie 0-15
- T0 bis T7 Teilnehmer 0-255

Die Quelladresse ist immer die physikalische Adresse eines Gerätes. Die Schreibweise in dezimalen Zahlensystem ist: <Bereich>.<Linie>.<Teilnehmer>

z.B. 10.0.125

4.6.1.3. Destination Address

Physikalische Adresse des Zielgerätes oder eine Gruppenadresse.

4.6.1.4. Physikalische Adresse

7	6	5	4	3	2	2	0	7	6	5	4	3	2	2	0
B3	B2	B1	B0	L3	L2	L1	L0	T7	T6	T5	T4	T3	T2	T1	T0

- B0 bis B3 Bereich 0-15
- L0 bis L3 Linie 0-15
- T0 bis T7 Teilnehmer 0-255

4.6.1.5. Gruppenadresse

7	6	5	4	3	2	2	0	7	6	5	4	3	2	2	0
0	H3	H2	H1	H0	M2	M1	M0	U7	U6	U5	U4	U3	U2	U1	U0

- 0 Adressierung
- H0 bis H3 Hauptgruppe
- M0 bis M2 Mittelgruppe
- U0 bis U7 Untergruppe

Dies ist die Einteilung in 3 Ebenen, die Schreibweise der Gruppenadresse ist z.B. 1/3/43

Es gibt auch die (selten genutzte) Möglichkeit die Gruppenadresse in nur 2 Ebenen aufzuteilen. Dann gibt es nur die 4 Bit breite Hauptgruppe und die aus 11 Bit bestehende Untergruppe.

Das erste Bit ist immer 0 für die Standard-Adressierung. Es gibt auch Sonderadressierungen, die interessieren uns aber erstmal überhaupt nicht.

4.6.1.6. DRL

DRL kommt von Destination-adress-flag, Routing-counter, Length und das sind bereits die drei Bestandteile dieses Steuerbytes:

7	6	5	4	3	2	2	0
D	R2	R1	R0	L3	L2	L1	L0

- D ... Das Destination-Adress-Flag gibt an ob die Zieladresse eine physikalische Adresse (0) oder eine Gruppenadresse (1) ist.
- R0 bis R2 ... Der Routing-Zähler gibt an w ie oft Telegramme über Linien- und Bereichskoppler w eitergereicht w erden. Der Standardw ert ist 6 und w ird bei jedem Weiterreichen um eins reduziert. Bei 0 w ird das Telegramm gar nicht w eitergeleitet, bei 7 hingegen beliebig oft.
- L0 bis L3 ... Länge der auf das DRL-Byte folgenden Nutzdaten. Der Wertebereich geht von 1 bis 16, d.h. bei Länge=1 folgen 2 Bytes, bei Länge=15 folgen 16.

4.6.1.7. Data, Nutzdaten

Die Nutzdaten bestehen aus mindestens einem, maximal 16 Bytes. Dabei haben die ersten beiden Bytes eine besondere Bedeutung, denn zum Einen ist in ihnen der auszuführende Befehl kodiert, zum Anderen kann man viele Aufgaben schon mit diesen 2-Byte erledigen. Es existieren auch Befehle mit nur einem Byte.

Es gibt, grob gesagt, zwei mögliche Varianten für die Nutzdaten:

- Der so genannte EIS (EIB Interworking Standard) ist ein Standard zur Kommunikation von Geräten unterschiedlicher Hersteller. Der Befehlssatz ist ziemlich mächtig und erlaubt so ziemlich alles an Informationen auf Gruppenadress-Ebene zu übertragen.

Es gibt 15 verschiedene EIS Formate für die folgenden Funktionen:

- EIS 1 Schalten
- EIS 2 Dimmen
- EIS 3 Uhrzeit
- EIS 4 Datum
- EIS 5 Wert, Zahl mit Nachkommastellen
- EIS 6 Relativwert, 0-100%

- EIS 7 Antriebssteuerung
- EIS 8 Zwangssteuerung
- EIS 9 Zahl mit Nachkommastellen nach IEEE
- EIS 10 16-Bit Wert
- EIS 11 32-Bit Wert
- EIS 12 Zugangskontrolle
- EIS 13 ASCII Zeichen
- EIS 14 8-Bit Wert
- EIS 15 Zeichenkette

- Die zweite mögliche Nutzung sind Befehle zur Kommunikation mit nur einem Teilnehmer. Dies sind alle möglichen Befehle zur Programmierung, Parametrierung, zum Lesen und Schreiben des EEPROM eines Teilnehmers, etc.

Wie genau die Inhalte der Telegramme zu interpretieren sind, muss noch erläutert werden.

4.6.1.8. Checksum

Im Anschluss an die Nutzdaten wird immer das Prüfbyte gesendet. Es handelt sich dabei um die invertierte, bitweise EXOR-Verknüpfung aus allen vorher gesendeten Bytes des Telegramms.

Auf der Empfängerseite kann man sich die Prüfung ziemlich einfach machen. Es reicht dabei, alle empfangenen Bytes inklusive des Prüfbytes ohne Übertrag zu addieren. Nur wenn am Ende 0xFF herauskommt ist das empfangene Telegramm OK.

4.6.2. Header Datei: protocol.h

```
//Paket Struktur:
struct Packet
{
      unsigned char    control;
      unsigned int     src;
      unsigned int     dest;
      unsigned char    drl;
      unsigned char*   data;
      unsigned int     checksum;
};

//Öffentliche Methoden:
/*********************************************************************
Function:  protocol_init()
Purpose:   initialisize the protocol
Input:     none
Returns:   standard return code
*********************************************************************/
unsigned int protocol_init();
```

```
/*********************************************************************
Function:  protocol_send_packet()
Purpose:   sends a packet/telegramm to the bus
           use: physical_put_c()
Input:
Returns:   standard return code
*********************************************************************/
unsigned int protocol_send_packet(    unsigned int type,
                                      unsigned char dest,
                                      unsined char* data );

/*********************************************************************
Function:  protocol_receiv_packet()
Purpose:   receives a packet/telegramm from the bus
           use: physical_get_c()
Input:     pointer to a Packet structure
Returns:   standard return code
*********************************************************************/
unsigned int protocol_receiv_packet (struct Packet * packet);

//Rückgabewerte:
#define PROTOCOL_INIT_SUCCESS                    0x0001
#define PROTOCOL_INIT_ERROR                      0x0002

#define PROTOCOL_SEND_PACKET_SUCCESS             0x0003
#define PROTOCOL_SEND_PACKET_ERROR               0x0004

#define PROTOCOL_RECEIV_PACKET_SUCCESS           0x0005
#define PROTOCOL_RECEIV_PACKET_UART_ERROR        0x0006
```

4.7. Packet Controller – OSI Layer 4

Betroffene Dateien: controller.h, controller.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 Überprüfung an eine ping Routine weiter.

4.7.1. Header Datei: controller.h

```
//Öffentliche Methoden:

/*********************************************************************
Function:  controller_run()
Purpose:   call from main to do the EIB/KNX communication
Input:     none
Returns:   standard return code
*********************************************************************/
unsigned int controller_run();
```

4.8. Sensoren und Aktoren Libraries

VORSCHLAG!!!!!!

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.

4.8.1. Taster und Relay über INT0

Betroffene Dateien: client_switch_int0.h, client_switch_int0.c

4.8.1.1. 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);
```

4.8.2. Taster und Relay über INT0

Betroffene Dateien: client_switch_int1.h, client_switch_int1.c

4.8.3. Taster und Relay ohne externen Interrupt

(Problem: uc kann nicht in den Sleep Modus gesetzt werden)

4.8.4. Phasenschnittdimmer

4.8.5. PWM Mortorsteuerung

4.8.6. Schrittmotorsteuerung

4.8.7. ...

4.9. Main Prog

Stromsparen mit Sleepmodus.

5. Hardwarespezifikationen der Clients

- Die Hardware muss in eine Standard-Unterputzdose passen (Durchmesser ca. 55mm)
- Auf den Stromverbrauch 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.

5.1.1. Stromversorgung

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

5.1.2. Mikrocontroller

Der uC muss folgende Kriterien erfüllen:

- ATMEL kompatibel

- Hardware UART
- Externer Taktgeber (Quarz, Oszillator)

Ich werde für den Anfang einen ATMEGA8 verwenden.

5.1.3. Peripherie

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

6. Hardwarekomponenten

Hier kommen die Hardwarekomponenten hin. Jeder kann sich beteiligen. Einfach erfinderisch sein!
Vorschläge. Relay, Taster, Temperatursensor, Mehrfach Relay (Luster), Mehrfach Phasenschnittdimmer(Luster)

6.1. RS485 BUS

Für die serielle Übertragung am RS485 Bus werden 2 Leitungen benötigt (Receiv, Transmit). Twisted Pair Leitungen 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 würde 18 Volt bzw. 13,8 Volt empfehlen.

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

6.1.1. 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.

6.1.2. Stecker

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

Anforderungen: Kleine Dimensionen. evt. nur Schraubklemmen.

6.1.3. Termination Widerstand

Ein kleine Miniplatine, welche an die zwei Endpunkte des Buses angeschlossen wird und die mit dem Abschlusswiderstand bestückt ist

6.1.3.1. Schaltplan

Eagle Datei

6.1.3.2. Board Layout

Eagle Datei

6.2. RS485 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.

6.2.1. Schaltplan

hier kommt der schaltplan her (EAGLE)

6.2.2. Board Layout

7. NET-IO Board

7.1. Bus Connector

Eine kleine Platine, die das NET-IO Board von Pollin mit dem Hausbus verbindet.

7.1.1. Schaltplan

Eagle Datei

7.1.2. Board Layout

Eagle Datei

7.2. Ethernet Schnittstelle

Das NET-IO Board von Pollin dient als Schnittstelle zwischen dem RS485 Hausbus und Ethernet.

7.2.1. Software

Software des Hausbus <-> Ethernet schnittstelle.

7.2.2. 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. (Keine Daueraktualisierung)

Auf die (IP)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.

7.3. Hardware

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

8. 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.