Enhanced TP-UART interface board

Andreas Fernbach

Project supervised by

Georg Neugschwandtner

A-Lab @ Automation Systems Group
Institute of Computer Aided Automation
Vienna University of Technology

November 25, 2009

afernbach@auto.tuwien.ac.at

Contents

1 Introduction

KNX [2] is an open system standard for home and building automation. The KNX standard defines multiple network media. The most commonly used among these is a free topology twisted pair medium referred to as "TP1" [3]. TP1 provides link power and uses balanced baseband signal encoding with asynchronous start-stop transmission at 9600 bit/s.

The Siemens TP-UART-IC [4] – in the following simply referred to as "TP-UART" – is designed for interfacing microcontrollers that operate KNX devices to the TP1 medium. However, it can also be used to build a simple yet highly versatile TP1 network interface for PCs. For example, such an interface has benefits in a lab environment since the TP-UART host protocol places very few restrictions on the frame format. While the TP-UART handles most of the KNX protocol stack up to the data link layer (and thus, the most critical timing requirements), higher level protocol aspects remain with the host controller. This leaves ample leeway for, e.g., testing protocol extensions.

However, the TP-UART host protocol – being designed for microcontroller interfacing – still contains some relatively tight timing constraints. While these are not a serious problem for a microcontroller to handle and can be considered a fair price to pay for the flexibility of the protocol, a PC without a real-time operating system will hardly be able to meet them.

Linux kernel drivers have been developed to address this problem [11, 12, 14]. However, a kernel driver is operating system dependent and requires significant maintenance effort (cf. [13]). Moreover, it introduces an additional proprietary protocol layer (accessed via *ioctl* calls), which can be an obstacle if program code is to be transferred to an environment where such a driver is not necessary. Operating system independent user mode solutions are possible [15], but require assumptions about the frame format and special attention to recovery from error conditions.

The goal of the project described in this report was enabling access to the full flexibility of the TP-UART host protocol through a generic, operating system and programming language neutral serial port abstraction. For this purpose, a microcontroller was to be placed between TP-UART and PC to monitor the timing of the critical TP-UART services and convert them to out-of-band communication where applicable. However, any modifications to the protocol were to be stateless and as small as possible.

The approach chosen and the desired behaviour of the interface are discussed in detail in a paper that was presented at the 2008 KNX Scientific Conference [5]. This paper also presents the results of the detailed analyses of the TP-UART host protocol that were made during the project and significantly extend the information available in the data sheet ([4]).

While the following report shares some content with this paper, it focuses on the implementation of the functional requirements described there. The KNX TP1 network interface that was designed and built during the course of the project is presented. In addition to putting the enhancements described in [5] into practice, the hardware platform created is versatile enough to be useful far beyond this task. Its design is available as Open Hardware; all design documents and source code are available on the A-Lab website [1].

2 Requirements

2.1 Hardware

The design of the interface hardware should follow the overall goals of being simple to build and easy to handle. No particularly expensive software or hardware tools should be required for creating and/or modifying the design. Size and complexity (in particular trace width and spacing) of the PCB (printed circuit board) should allow low-cost production. Pin-through-hole components should be used wherever possible. Mounting holes were to be included in the layout design for applications where protecting the PCB with an enclosure is necessary. However, the use of an enclosure should be optional. Therefore, it should be possible for all components to be directly mounted on the PCB. All terminals, jumpers, switches, push buttons and LEDs should also be labeled there (within the limitations of available space).

In-system programming and debugging is standard on current microcontrollers and was to be included. Also, a minimal user interface should be provided to allow modification of program behavior as well as to trigger actions (such as sending a test network message) and show status information (e.g., error conditions) – without reprogramming, requiring a debug interface connection or using the RS-232 interface for this purpose. Also, access to the TP-UART status signals should be possible.

The KNX TP1 medium is electrically isolated to ground. On the other hand, the RS-232 interface on the PC is grounded. Therefore, galvanic isolation between the KNX and the PC side of the interface should be provided. Although the interface is primarily intended for laboratory use, this was made a requirement to avoid problems in case the interface should be connected to a larger KNX installation.

Since any wire that is not strictly necessary is a nuisance on a lab desk, the interface should not require an external power supply. Still, galvanic separation was not to be compromised.

2.2 Firmware

There are two tight timing constraints in particular in the TP-UART host protocol a PC application is not always able to handle. One concerns the end-of-packet (EOP) recognition for frames propagated from the TP1 network to the host. The end of such a received frame is indicated by the TP-UART as a certain (minimum) period of silence on the host interface only. A reasonable condition for an end of packet is the observation of 7 bit times (of 104 μs each) of silence on the TP1 bus or, likewise, on the TP-UART host interface. This follows from the analysis and discussion in [5], pp. 6-8. The task of monitoring such a time interval is very well suited to a microcontroller, since its hardware timer and compare units make accurate time measurements easy. The occurrence of this condition should be signalized to the PC application by inserting explicit symbols into the TP-UART host data stream.

The other time critical matter is sending the U_AckInformation service. According to the TP-UART data sheet, the “U_AckInformation-Service is to indicate if the device is addressed. This service must be send latest 1,7 ms (9600 Baud) after receiving the address type octet of an addressed frame” ([4], p. 12). Measurements conducted

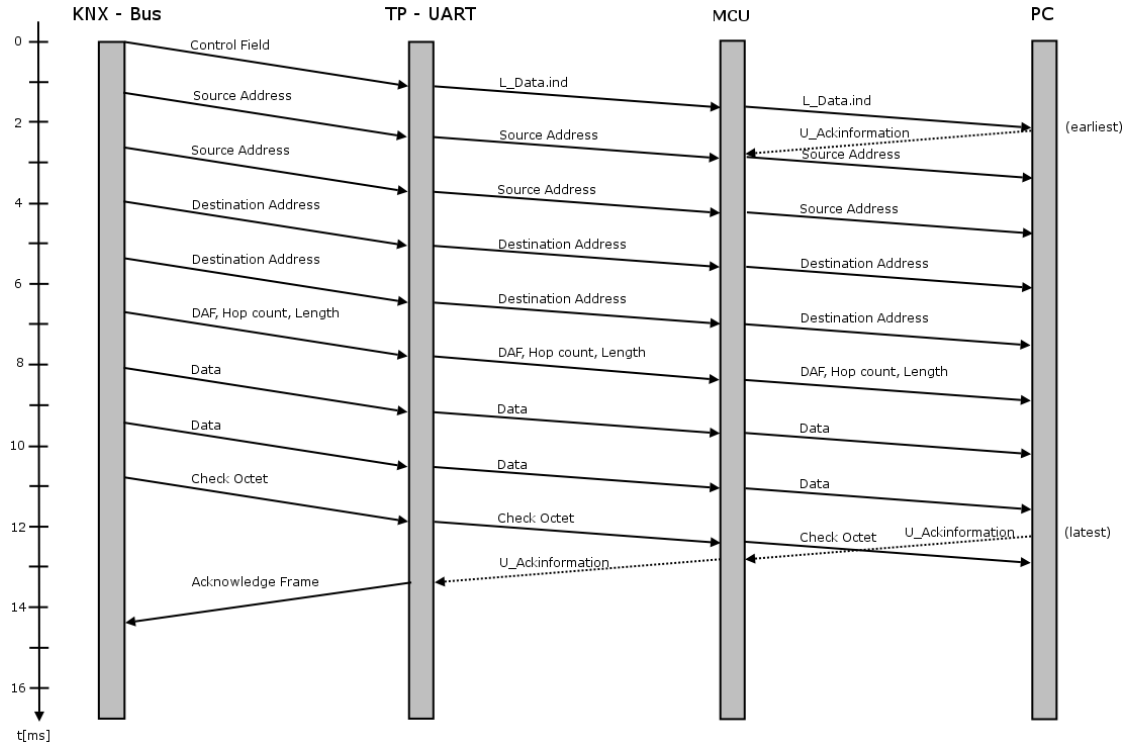

Figure 1: Time frame for U_AckInformation service

during this project show that the transmission of the U_AckInformation-Service (to the TP-UART) must be completed not later than 1.54 ms after the TP-UART has started transmitting the check octet (to the host); otherwise, the TP-UART does not generate an Acknowledge frame. This corresponds exactly with the TP-UART datasheet (if the cited sentence is completed with the missing information that the complete reception of the sixth frame octet by the host, the start of the transmission of the U_AckInformation character and the host interface data rate are referred to); it means that the stop bit of the U_AckInformation-Service and the start bit of the Acknowledge frame lie edge to edge. It must be considered that when a microcontroller is placed between the TP-UART and the PC, the constraints described apply to its TP-UART side. The maximum allowable response time for the PC is shorter due to the transmission delays between microcontroller and PC and processing delay in the microcontroller (the latter is not significant and can be neglected in our case). The resulting time slot for the PC application to send the U_AckInformation-Service to the microcontroller (MCU, microcontroller unit) is illustrated in Fig. 1. In this diagram, a communication speed of 19,200 bps is assumed between the TP-UART, the MCU and the PC.

As this is a relatively short time for a PC application to trigger specific actions, timeouts can happen once in a while. These timing conditions should therefore be monitored by the microcontroller firmware. The user should be informed about the occurrence and the current count of timeouts via the user interface on the board.

3 Hardware

3.1 Microcontroller selection

The microcontroller is a key design component. Within this project, its task is not resource intensive with regard to processing power, memory size or I/O. It mainly consists of communicating via two relatively low speed UARTs (Universal Asynchronous Receiver Transmitter) and checking the timing of incoming messages. Thus, it can be optimized for low power consumption and easy manual soldering. In line with the overall project goals, the price of the programming adapter and the toolchain necessary for developing and debugging the microcontroller software also have considerable impact. First, microcontroller types available in packages with low pin count were selected from popular product families (Table 1 shows the considered types).

Evaluating the types on this short list according to the above criteria, the TI (Texas Instruments) MSP430-F123 [17] (and its F1232 sibling [18], which can be considered identical for the purposes of this project) emerged as ideally suited for the project. It provides a 16 bit RISC CPU, 8 kB Flash memory, 256 Bytes RAM, and can operate at clock rates of up to 8 MHz. Even at 8 MHz, it consumes less than 3 mA in active mode. In-system programming and debugging are possible via a standard JTAG (Joint Test Action Group) interface. Low-cost JTAG adapters are available. Entry level versions of two commercial toolchains are offered free of charge. In addition, a GCC based open source toolchain exists.

The MSP430-F123 is available in a 28 pin small-outline package with 1.27 mm pin spacing (the same as the TP-UART-IC has). The MSP430-F123 has only one hardware UART (no microcontrollers in this class appear to be available that would have two; rather, many have none at all, such as the entire ST7 family). This means that the UART function has to be implemented in software, using a hardware timer. While the MSP430-F123 has only one 16 bit hardware timer, it is equipped with three capture/compare units. Using one of these units for software UART transmission and one for reception, full duplex operation with minimum CPU load is possible. The third capture/compare unit remains free for application timing tasks.

Manufacturer/Type	UARTs	Maximum supply current	Debugging
ATMEL ATMega48	1	12 mA @ 8 MHz, Vcc = 5 V	debugWIRE
ATMEL ATmega88P	1	9 mA @ 8 MHz, Vcc = 5 V; 2.5 mA @ 4 MHz, Vcc = 3 V	debugWIRE
NEC 78K0S/KB1+	1	14 mA @ 6 MHz, Vdd = 5 V	MINICUBE
TI MSP430F1232	**1**	**2.9 mA @ 8 MHz, Vcc = 3.3 V**	**JTAG**
ST ST7LITE19B	0	9 mA @ 8 MHz, Vcc = 5.5 V	ICC

Table 1: Some of the considered microcontrollers [20, 21, 22, 17, 18, 23]

3.2 Interface board

In accordance with the goal that no expensive software should be required for modifying the design, the free Light Edition of Cadsoft EAGLE [16] was chosen as the CAD

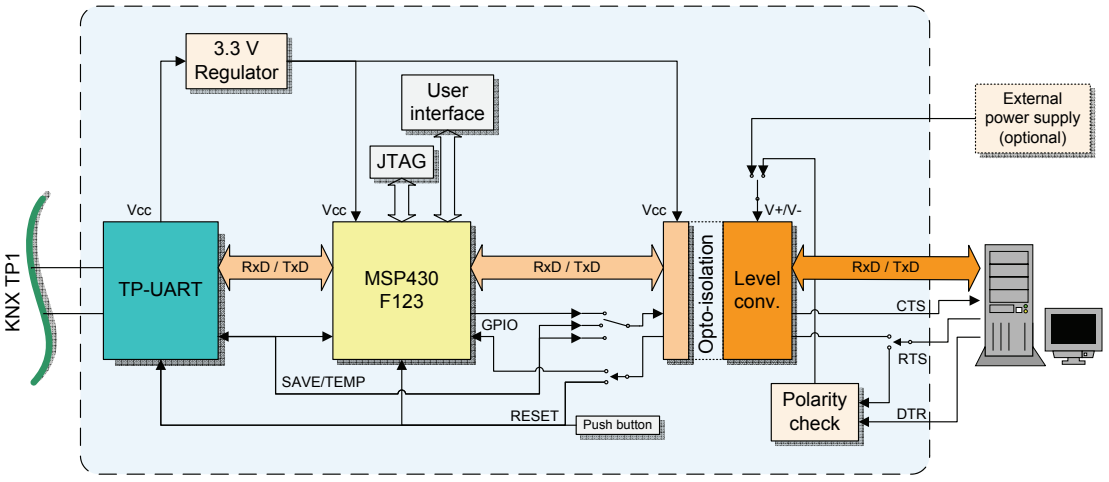

Figure 2: Block diagram

software to be used. EAGLE Light Edition supports PCBs with a size of up to 80×100 mm and two layers. The PCB design goes to the limit set by these constraints. It is shown in Figure 5.

Figure 2 gives a schematic overview of the interface board. The full schematic is shown in Figure 4. Following the signal flow from the KNX bus lines to the RS-232 interface, the first part of interface board is the TP-UART with its external circuitry.

3.2.1 TP-UART circuitry

The support circuitry for the TP-UART-IC follows the "typical application circuit" shown in its data sheet [4]. Directly connected to the bus terminal, the diode D1 protects the circuit against overvoltage; another diode (D2) protects it against polarity reversal. A jumper block (JP1) determines whether the TP-UART operates in normal mode or in analog mode and connects the required passive components. In normal mode, the TP-UART host interface is always running at 19200 bps. All status pins of the TP-UART are also permanently connected to the microcontroller. The firmware can pass their state to the PC via UART communication (this possibility is not used by the current firmware). To make connecting an oscilloscope or logic analyzer for monitoring easier, all TP-UART data and status I/Os (TSTOUT, TxD, RxD, RESn, SAVE) are broken out on a pin header (JP2), as is the ground potential (X2). The RESn signal is additionally connected to a pushbutton which allows the user to trigger a hardware reset.

To satisfy the claim of an interface board without an external power supply, the stable 5 V supply the TP-UART provides is used to supply the KNX side of the interface board. The power is derived from the KNX TP1 network and provided at the VCC pin of the TP-UART. However, according to the TP-UART data sheet, the maximum load must not exceed 10 mA. Due to careful choice of the components, this is sufficient to power the microcontroller, the user interface, one half of the optocouplers, and the required support circuitry. The HCPL2530 optocouplers consume about 0.2 mA together. Since the MSP430 requires a 3.3 V supply, a voltage regulator (IC1) is required, whose ground pin current also factors into the equation. However, regulators with a ground pin current

as low as 0.15 mA at 10 mA load current are readily available. On this board, a LE33A low drop regulator is used.

3.2.2 Microcontroller

The center part of the interface board, the microcontroller, is looped into the serial data connection between TP-UART and PC. Since the TP-UART and the microcontroller are powered from the KNX network, they are automatically reset when link power returns after a failure or after being disconnected. Due to its integrated brownout detection, the MSP430 reliably starts up once its power supply is stable. Therefore, a diode (D6) protects it from being reset by the TP-UART RESn signal. Since the power supply (TP-UART VCC) is stable for a considerable time before the TP-UART releases RESn, this gives the microcontroller additional time to initialize. Therefore, it can correctly receive the Reset.indication service, which is sent by the TP-UART almost simultaneously with releasing RESn. An additional diode (D5) can be mounted in case something similar should be necessary in the opposite direction; however, it will typically be replaced by a wire jumper since the MSP430 RST pin is not bidirectional. In addition, a push button is provided that allows to manually reset both TP-UART and microcontroller if necessary. If this button is used, the MSP430 cannot start its initialization before the TP-UART and therefore cannot finish it in time to correctly receive the Reset.indication service.

As mentioned in Section 3.1, the MSP430F123 only provides one hardware UART. Since the microcontroller is looped into a bidirectional serial communication link, a second UART is required. This one is implemented in software using the hardware timer and two of its capture/compare units. The software UART is placed at the TP-UART side to allow the usage of this timer hardware to perform the accurate bit timings necessary (and still be able to use the hardware UART for communication with the PC) when the TP-UART is operated in analog mode.

For in system programming and debugging, the JTAG interface of the MSP430 is accessible via a 14 way pin header (PL1) which fits the programming interface of the TI MSP430 flash emulation tool and compatibles.

For clock generation, the MSP430 uses an 8 MHz crystal oscillator (Q2). The value of the two capacitances between the oscillator pins and ground (C7, C9) are taken from the oscillator data sheet.

If the enhanced functionality provided by the microcontroller is not needed, the board can also be assembled without the microcontroller. This allows immediate use with existing TP-UART related PC software. The PCB layout includes the necessary optional pass-through tracks and jumpers (BP1, BP2) required for the RXD and TXD lines.

3.2.3 User interface

A minimal user interface (UI) is included to allow interacting with the microcontroller without requiring a debug interface connection or using the RS-232 interface. It consists of four LEDs, two push buttons and two DIP switches, which are all connected to the general purpose I/O (GPIO) pins of the microcontroller. The port allocation is shown in Table 2. Pull down resistors are used to prevent voltage levels at the microcontroller input pins where the switches and pushbuttons are connected from floating. In case these

UI element	MCU port	Port direction
LED1	3.0	Out
LED2	3.1	Out
LED3	3.2	Out
LED4	3.3	Out
SW1.1	2.0	In
SW1.2	2.1	In
PB1	2.2	In
PB2	2.3	In

Table 2: User interface

pins are erroneously configured as outputs, the current limiting resistor R13 prevents a short circuit situation. For the LEDs, an ultra bright type was chosen that provides reasonable brightness already at 0.3 mA.

3.2.4 RS-232 level conversion

On the PC side, the interface needs to generate RS-232 (EIA-232) signal levels. An RS-232 receiver considers a line voltage of 3 to 15 V ("space state") as a logic '0' and the corresponding negative voltage range ("mark state") as a logic '1'. Transmitters are required to output at least 5 V. To further increase the noise margin, levels around 10 V are typical (see, for example, [24]).

Since most equipment today operates on +5 V or lower, ICs that generate these voltages from a single +5 V supply and perform the necessary signal level conversion are popular (a very common type being the MAX232 [25]). However, the power supplied by the TP-UART cannot be used since galvanic separation has to be observed. Also, there is no such thing as a +5 V power supply pin on an RS-232 interface. Power can only be drawn from the signal lines (see, for example, [28] and [27]). The exact amount depends on the driver circuits in the PC. Short circuit currents of about 10 mA are typical (with a single output shorted). The output voltage rapidly decreases with increased load (see also the technical data for the 1488 series of RS-232 drivers that was once the de facto standard in PCs, e.g., [26]).

In theory, it should be possible to pass the DTR signal through a voltage regulator to obtain a stable 5 V supply that is independent of the voltage the line is actually driven to by the PC, and use a standard RS-232 transceiver with a built in DC-DC converter for level conversion. However, this method is not very efficient. In the experiments conducted during this project, it was not possible to obtain stable operating conditions using DTR alone, despite using low power ICs. The optocouplers could not be further optimized for power consumption without increasing the signal rise time to unacceptable levels.

Therefore, it was decided to use RTS as well as a negative power supply (requiring the signal to be deasserted by the PC). This way, a (low power) RS-232 transmitter can directly be provided with positive and negative input voltages. DC-DC conversion is eliminated entirely. A low power type from the well known 1488 series of RS232 line drivers, the MAX1488ECPD, was chosen. It loads the status lines lightly enough that

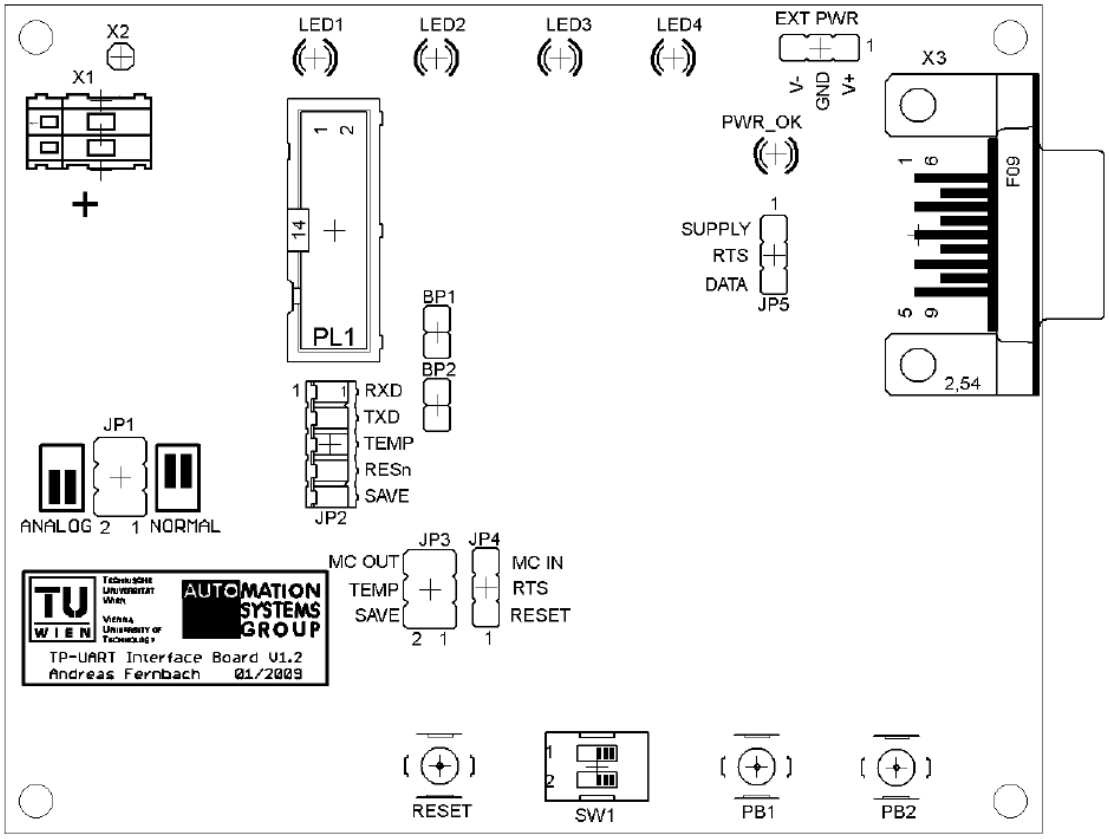

Figure 3: Jumpers, connectors and UI elements

the RS-232 output levels are in the non-critical range of +/- 8 to 10 V.

Correct DTR and RTS polarity is ensured via diodes and visually confirmed using a LED (PWR_OK). If the PC does not provide an RTS output or if the signal has to remain available for communication (selectable via JP5, see next section), a +/- 9..12 V external power supply can be connected (EXT PWR).

It would also be possible to derive power from signal lines irrespective of their polarity, leaving communication entirely undisturbed ([29]). This possibility was not pursued since it would significantly increase circuit complexity. Rather, the goal was a simple design that would eliminate the need for an external power supply in the majority of use cases.

3.2.5 Jumpers and connectors

Figure 3 shows all jumpers and connectors placed on the interface board. An overview of the functions and the various settings are given in Table 3. A detailed description of each element follows.

JP1 This jumper block is used to determine the operation mode of the TP-UART IC. To select *normal mode*, place one jumper over pins 4 and 6; in addition, place another over pins 5 and 6. For *analog mode*, connect pins 1-3 and 2-4. In normal operation mode both parts of the TP-UART, digital and analog, are enabled; the idle-level on its

Name	Function	Description
JP1	TP-UART mode	4-6/3-5 Normal, 1-3/2-4 Analog
JP2	TP-UART breakout	RxD, TxD, TEMP, RESn, SAVE
JP3	CTS select	1-2 SAVE, 3-4 TEMP, 5-6 MCU Output
JP4	RTS select	1-2 RESET, 2-3 MCU Input
JP5	RTS select	1-2 Supply, 2-3 Data
EXT PWR	External supply	1: +12 V, 2: GND, 3: -12 V
X1	KNX TP1 connector	1: TP1 -, 2: TP1 +
X2	Ground pin	Signal common for test purposes
X3	Serial interface	RS-232 pinout on DB9 female
BP1, BP2	MCU bypass	Must be connected if no MCU is placed

Table 3: Jumpers

host UART interface is '1' (high). In analog operation mode only the analog part is enabled; the idle-level on the host UART interface is '0' (low) ([4], pp. 8-9).

JP2 For test purposes, the TP-UART I/O signals are broken out. RxD, TxD, TEMP, RESn and SAVE are provided. GND is available on X2.

JP3 The RS-232 CTS status line (which is connected to the right column of pins) can be fed with one of the following signals: SAVE (jumper position 1-2), TEMP (jumper position 3-4) and MCU port 1.1 (jumper position 5-6), which has to be configured as output in this case. This way, additional status information can be propagated to the PC.

JP4 Via this jumper, the RS-232 RTS status line can be used to reset the microcontroller and the TP-UART (jumper position 1-2) or as additional data path from the PC to the microcontroller (jumper position 2-3). The signal is connected to port 1.0 of the MSP430.

JP5 This jumper determines if the RS-232 RTS status line is used as the power supply for the level conversion part of the interface board (jumper position 1-2) or for data (jumper position 2-3). In the latter case, the function of RTS is determined by JP4 and an external power supply has to be connected to EXT PWR.

EXT PWR If the host device connected via RS-232 does not provide status lines on this interface where power can be drawn from or if JP5 is set to "Data" (jumper position 2-3), an external symmetric 9..12 V DC power supply must be connected to these pins.

X1 The bus connector has a dual footprint. On the one hand, spring terminals can be fitted on the inner pads (X1) to directly connect the TP1 wire pair. On the other hand, two pins can be soldered to the outer pads (TP1, TP2). This is intended to obtain a standard "type 5.1" connector that is usually found on KNX devices; however, the outer pads do not have the correct distance in the current layout.

X2 This ground pin for test purposes is connected to the TP1 ground wire.

X3 RS-232 serial interface with standard pin assignment.

BP1, BP2 These jumpers connect the TP-UART RxD and TxD pins to the TxD and RxD pins of the galvanic isolation and level conversion block (RS-232 side). These connections need to be made if no microcontroller is installed on the board.

3.2.6 Schematics

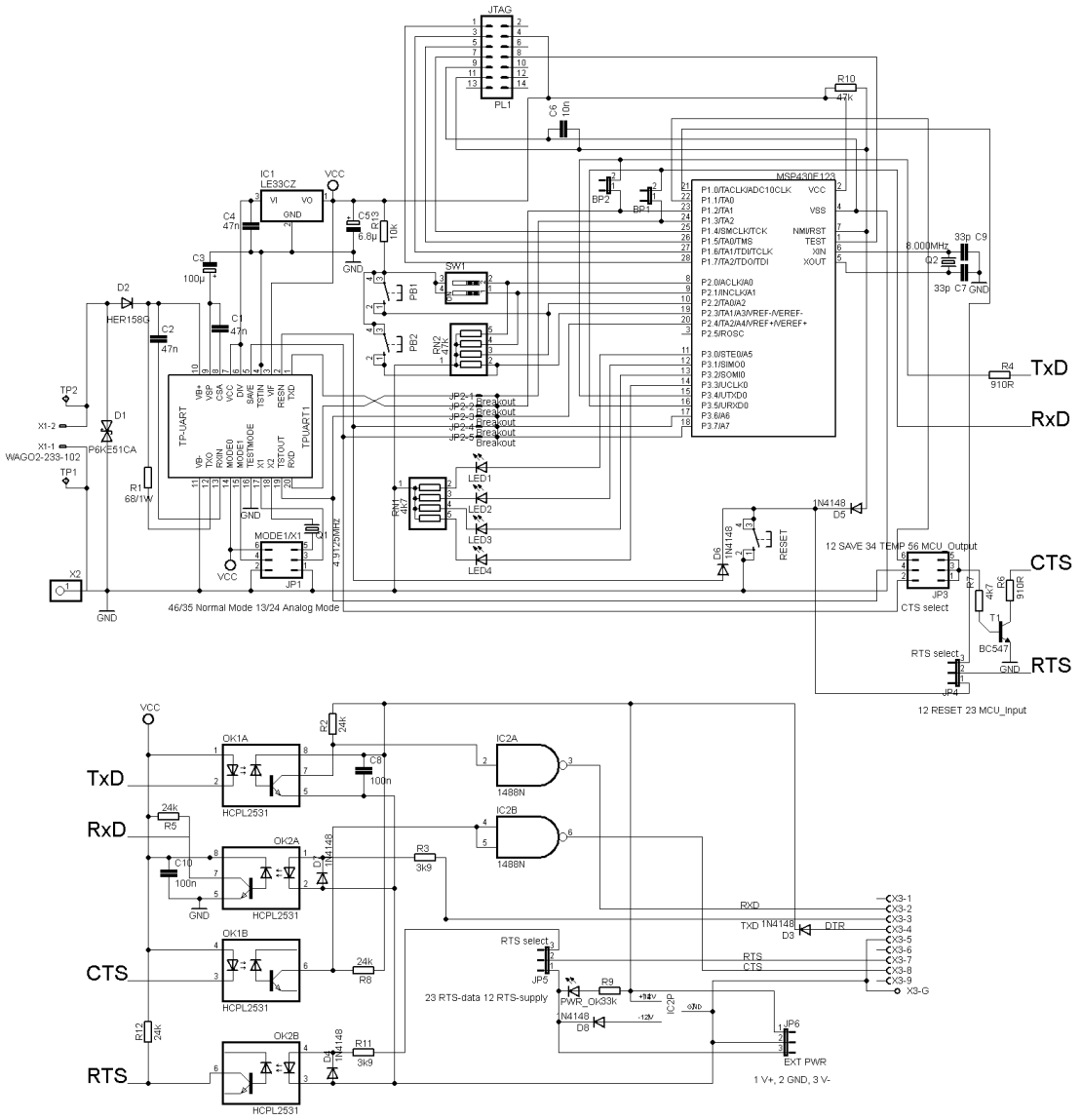

Figure 4: Interface board: Schematic

3.2.7 Board layout

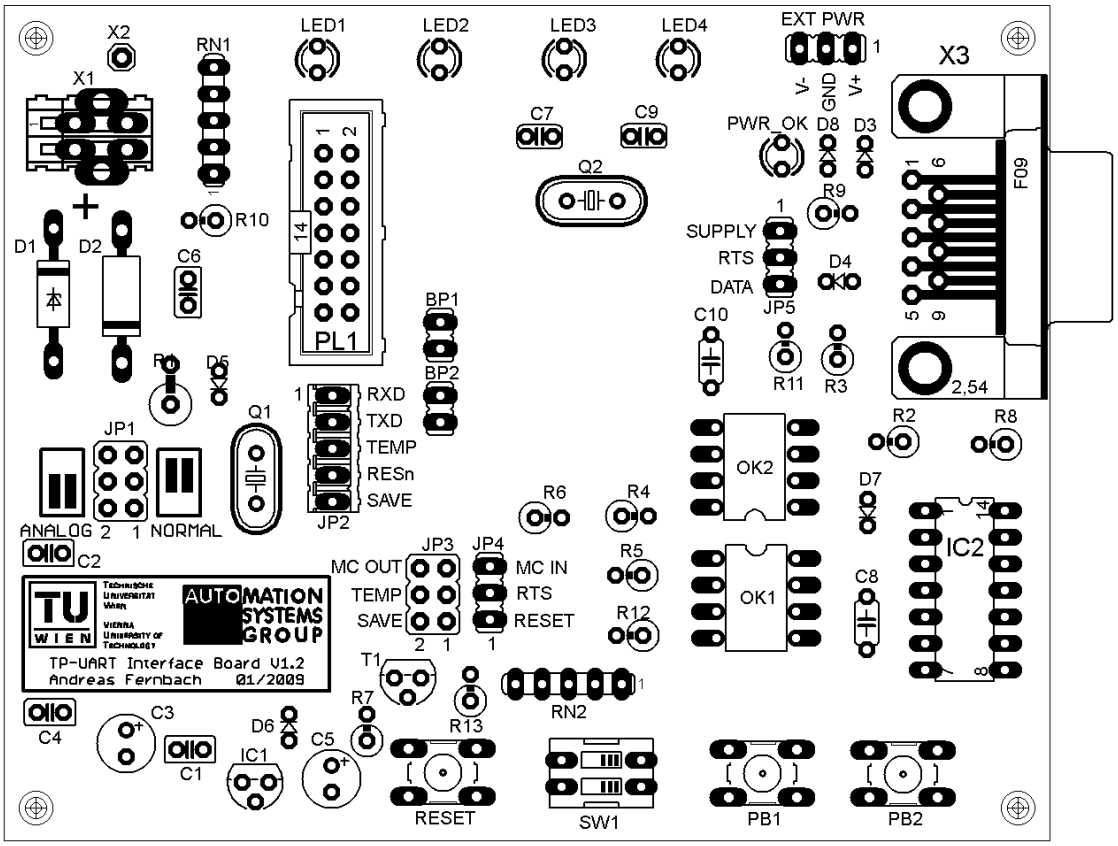

Figure 5: Interface board: Layout (not to scale)

3.3 External power supply board

If for any reason no power can be drawn from the RS-232 interface where the interface board is connected (no status lines available, status lines used for data communication), an external power supply must be used. It supplies the level conversion part (and half of the optocouplers) of the interface board with the necessary +/- 9...12V. A small board was built that connects to a stock AC (mains) power adapter via a standard low voltage connector and provides these voltages. The power supply board accepts input voltages from 12 V–35 V DC or 12 V–25 V AC. For mobile use, it is also possible to use a 9 V block battery as the power source. PP3 connectors are fitted on the power supply board where the battery can be attached. To avoid damage, an AC adapter and a battery should not be connected simultaneously. Figure 6 shows the schematic of the power supply board. The board layout is shown in Figure 7. Table 4 gives an overview of the connectors placed on the board.

A bridge rectifier (B1) at the input takes care of the right polarity of the input voltage. The voltage is stabilized by an electrolytic capacitor (C1) and a 12 V linear voltage regulator (IC2) to obtain V+ (and GND). V- is generated from V+ by a voltage inverter IC (IC1) that uses C3 to form a charge pump. The inverted voltage is buffered in C4.

In case a 9 V battery is used, there is no need for stabilisation, so the battery voltage is fed directly into V+ and GND to drive the voltage inverter. The output voltages are provided on JP1, whose pins have to be connected to the corresponding pins (V+, V-, GND) of JP6 on the interface board.

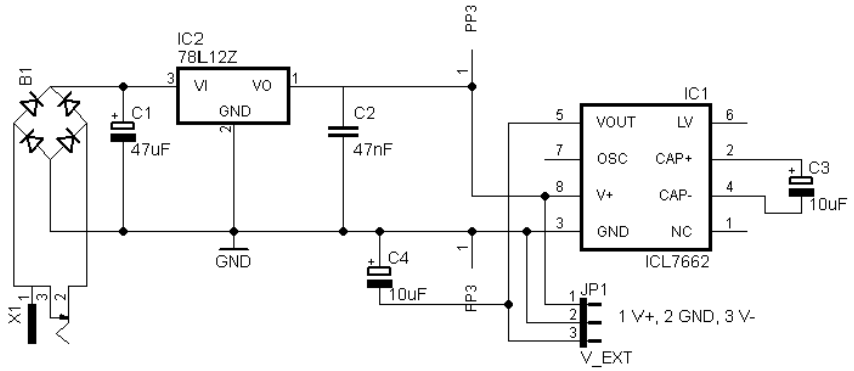

Figure 6: External power supply: Schematic

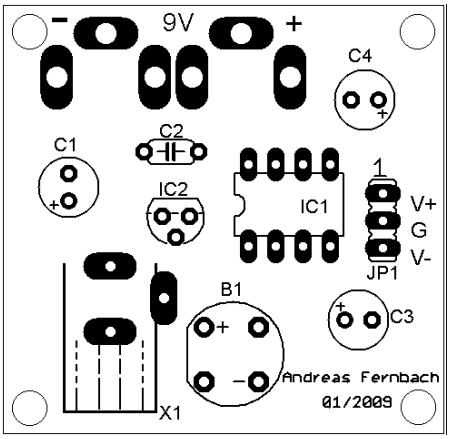

Figure 7: External power supply: Layout (not to scale)

Name	Function	Description
X1	Low voltage connector (input)	12–25 V AC/DC
JP1	Output voltage (to interface board)	1: +12 V, 2: GND, 3: -12 V
9V	9 V battery clip	Standard PP3 connector

Table 4: External power supply: Connectors

3.4 Part lists

Farnell (www.farnell.com) is a large distributor of electronic components. Where applicable, Farnell order numbers are provided for better documentation of part specifications (manufacturer, packaging, characteristics, ...).

3.4.1 External power supply

Part	Value	Device	Farnell order no.
B1	1A, 600V	Bridge rectifier	4077300
C1	47µF, 50V	Electrolytic capacitor	1144660
C2	47nF, 100V	Ceramic capacitor	1141782
C3	10µF, 50V	Electrolytic capacitor	1144604
C4	10µF, 50V	Electrolytic capacitor	1144604
IC1	ICL7662	Voltage inverter	1228203
IC2	78L12Z	Voltage regulator +12V	9490272
JP1		3 pin header	1022248
9V		PP3 connector	723988
X1		Low voltage connector	224960

3.4.2 Interface board

Part	Value	Device	Farnell order no.
C1	47nF, 100V	Ceramic capacitor	1141782
C2	47nF, 100V	Ceramic capacitor	1141782
C3	100µF, 16V	Electrolytic capacitor	1144614
C4	47nF, 100V	Ceramic capacitor	1141782
C5	6.8µF, 50V	Electrolytic capacitor	1144603
C6	10nF, 100V	Ceramic capacitor	1216425
C7	33p, 100V	Ceramic capacitor	1141761
C8	100nF, 100V	Ceramic capacitor	1141784
C9	33p, 100V	Ceramic capacitor	1141761
C10	100nF, 100V	Ceramic capacitor	1141784
D1	P6KE51CA	TVS	1467630
D2	HER158G	Fast Diode	1625088
D3	1N4148	Small signal diode	9843680
D4	1N4148	Small signal diode	9843680
D5	1N4148	Small signal diode	9843680
D6	1N4148	Small signal diode	9843680
D7	1N4148	Small signal diode	9843680
D8	1N4148	Small signal diode	9843680
IC1	LE33CZ	Voltage regulator 3.3V	9755349
IC2	MAX1488E	RS-232 line driver	1188031
JP1		6 pin header	3418492
JP2		5 pin header	588428

JP3		6 pin header	3418492
JP4		3 pin header	588404
JP5		3 pin header	588404
JP6		3 pin header	588404
LED1	TLDR4400	LED3MM ultrabright	1045471
LED2	TLDR4400	LED3MM ultrabright	1045471
LED3	TLDR4400	LED3MM ultrabright	1045471
LED4	TLDR4400	LED3MM ultrabright	1045471
MSP430F123	MSP430F123	TI low power MCU	1471266
OK1	HCPL2530	Optocoupler	1021246
OK2	HCPL2530	Optocoupler	1021246
PB1		Pushbutton	1217775
PB2		Pushbutton	1217775
PL1		JTAG connector	1106785
PWR_OK		LED3MM ultrabright	1045471
Q1	4.9125MHz	Crystal	9712925
Q2	8.000MHz	Crystal	9712844
R1	68/1W	Resistor	1357878
R2	24k/0.25W	Resistor	9341609
R3	3k9/0.25W	Resistor	9341854
R4	910R/0.25W	Resistor	9342311
R5	24k/0.25W	Resistor	9341609
R6	910R/0.25W	Resistor	9342311
R7	4k7/0.25W	Resistor	9341951
R8	24k/0.25W	Resistor	9341609
R9	33k/0.25W	Resistor	9341757
R10	47k/0.25W	Resistor	9341960
R11	3k9/0.25W	Resistor	9341854
R12	24k/0.25W	Resistor	9341609
R13	10k	Resistor	9341110
RESET		Pushbutton	1217775
RN1	4k7	Resistor network	9356100
RN2	47k	Resistor network	9356118
SW1		DIP switch	9479155
T1	BC547	Bipolar transistor	1467869
TPUART1		Siemens TP-UART IC	
X1		Terminal Block	4016129
X2		Ground pin	
X3		DB-9 socket	1207597

4 Microcontroller Firmware

This section gives an implementation description of the main microcontroller firmware. It is split in two parts, the description of how the end of an L_Data frame coming in from the KNX network is determined and the procedure of detecting late U_AckInformation services sent by the PC. For testing TP-UART output in analog mode, an alternative firmware was developed, which is not described here.

4.1 End-of-packet recognition

In the chosen implementation, a hardware timer increments a software counter (*bittimes_since_rx*) periodically every medium bit time (1/9600 s). It starts to run immediately after power up. If a UART character is received by the software UART (at the TP-UART side), *bittimes_since_rx* is reset to zero by the receive callback function of the software UART once the character has been fully received. If the threshold of seven bit times is reached, the hardware UART (at the PC side) is triggered to send an end of packet signal. This is either a break signal or the escape sequence '0x1B 0x00', depending on the positions of the DIP switches. If they are identical, a break signal is sent (the host protocol is not altered otherwise). If they are odd, the escape sequence is sent. The threshold can be adjusted by setting the macro PACKET_TIMEOUT to the desired value of bit times.

```
if (bittimes_since_rx == KNX_CHAR_DURATION + PACKET_TIMEOUT)

     // End-of-packet timeout elapsed on the medium?

    {
      if (((P2IN & P2_SW1) == P2_SW1) ^ ((P2IN & P2_SW2) == P2_SW2))
                                        // DIP switches in different positions?
        {
          // user selected "escape mode" -> generate escape sequence
          U0TXBUF = 0x1B;
          esc_status = INSERT_00;
        }
      else
        {
          // user selected "break mode" -> send a UART break signal
          P3SEL &= ~P3_SERIAL_TO_PC;    // detach HW USART module from pad
                                        // to force TXD low
        }

    }
```

The reason why KNX_CHAR_DURATION (the time for the transmission of one character on the TP1 medium, in bit times) is added to PACKET_TIMEOUT in the if statement is the following: Consider the situation that the stop bit of the last character was received seven bit times ago (*bittimes_since_rx == PACKET_TIMEOUT*). If another character has just begun to come in from the network now (which would mean that the timeout has *not* elapsed), we will not see it until we have waited for another KNX_CHAR_DURATION bit times. Since *bittimes_since_rx* only measures the duration between two receive callbacks – that is, from the end of one character to the end of the next –, but we require the duration between the end of one character and the *beginning* of another character to determine the end-of-packet timeout, its value has to be compensated by KNX_CHAR_DURATION.

If ESC signalization is chosen, a '0x1B' character is sent immediately after the end of packet condition becomes true. The following '0x00' character is appended when the next UART transmit interrupt occurs. The variable *esc_status* is used to store the information about this, which can be seen in the UART transmit interrupt service routine (ISR):

```
usart0_tx_isr (void)
{

  if (esc_status == INSERT_ESC)
    {
      U0TXBUF = 0x1b;
      esc_status = INSERT_NOTHING;
    }

  if (esc_status == INSERT_00)
    {
      U0TXBUF = 0x00;
      esc_status = INSERT_NOTHING;
    }
}
```

Depending on *esc_status* (whether it is INSERT_ESC or INSERT_00) the corresponding character is transmitted in the subsequent transmit ISR.

To prevent any misinterpretation of an ESC character by the host controller software it must be taken care that any '0x1B' appearing in a KNX frame is marked somehow. Therefore, a second '0x1B' is added if one is received from the TP-UART.

```
if ((c == 0x1b) &&      // is this an ESC character and are we in ESC mode?
      ((P2IN & P2_SW1) == P2_SW1) ^ ((P2IN & P2_SW2) == P2_SW2))
    esc_status = INSERT_ESC;
```

A single '0x1B' received from the bus is converted to the sequence '0x1B 0x1B' on the PC side.

4.2 Indication of a late U_AckInformation service

When the TP-UART is passing a data indication from the network to the host, the host is expected to answer with a U_AckInformation service in time so the TP-UART can generate the Acknowledge frame on the KNX medium. A timeout is determined by the microcontroller firmware as follows: Whenever the start of a (standard or extended) data frame is detected (i.e., a L_DATA.ind or L_LONG_DATA.ind service is received after a long enough pause), the variable *dataframe_octets* is incremented every on every software UART receive interrupt (at the TP-UART side) to count the number of octets the data frame consists of. Again, *bittimes_since_rx* is used in the calculation.

```
void
sw_uart_rx (unsigned char c)
{

  TACCR0 = TAR + KNX_BITTIME;   // align timer interrupt with end
                                //  of character

  if (dataframe_octets)
    dataframe_octets++;
```

```
  // Is this the start of a new L_Data message cycle?
  if (bittimes_since_rx > (ACK_TIMEOUT + 1))
    {
      if ((c & DATAIND_MASK) == 0x10)   // catch L_DATA, L_LONG_DATA
        {
          dataframe_octets = 1;
          ackinfo_do = 255;     // make certain a timeout will be detected
                                //  if no U_AckInformation is sent at all
        }
      else
        {
          dataframe_octets = 0;
        }
    }
```

From this point in time the reception of an U_AckInformation service is expected. Whenever the host sends a U_AckInformation, the microcontroller program stores the current time relative to the start of frame. This is done in the receive ISR of the PC side (hardware) UART. The variable *ackinfo_do* stores the count of (data) octets already completely received, *ackinfo_bsr* stores the medium bit time offset since the last stop bit.

```
#pragma vector=USART0RX_VECTOR
__interrupt void
usart0_rx_isr (void)
{

  swu_tx_byte (U0RXBUF);        // propagate character to TP-UART

  if (dataframe_octets
      && (U0RXBUF == U_ACKINFO_ACK || U0RXBUF == U_ACKINFO_NACK
          || U0RXBUF == U_ACKINFO_BUSY))
    {
      ackinfo_do = dataframe_octets;
      ackinfo_bsr = bittimes_since_rx;
    }

}
```

Whenever the end of a data frame (L_DATA, L_LONG_DATA) is detected (*if (bittimes_since_rx == ACK_TIMEOUT)*), the microcontroller program evaluates this stored timestamp to determine if the TP-UART has received the U_AckInformation in time or not. For this purpose, the actual point in time when the U_Ackinformation service has been received is compared to the latest possible point in time an U_Ackinformation service must come in to be propagated to the TP-UART in time. Both durations are calculated in medium bit times from the beginning of the frame. The time it takes to send one character between the TP-UART and the microcontroller (TPU2MCU_CHAR_DURATION) has to be included in the calculation twice to account for the fact that it causes the MCU to see the data frame later than it actually occurs on the medium on the one hand and, on the other hand, that the U_Ackinformation service has to be transmitted to the TP-UART before it can start generating the Acknowledge frame.

```
#pragma vector=TIMERA0_VECTOR
__interrupt void
timer_a0_isr (void)
{

  TACCR0 += KNX_BITTIME;
  if (bittimes_since_rx < 255)
    bittimes_since_rx++;
```

```
if (bittimes_since_rx == ACK_TIMEOUT)
  {

    if (dataframe_octets &&   // only data frames need to be acknowledged

        ( ((unsigned int) (ackinfo_do - 1) *
                          (KNX_CHAR_DURATION + KNX_MIN_CHAR_GAP)
                          + ackinfo_bsr + TPU2MCU_CHAR_DURATION) >

          ((unsigned int) (dataframe_octets - 1) *
                          (KNX_CHAR_DURATION + KNX_MIN_CHAR_GAP)
                          + ACK_TIMEOUT - TPU2MCU_CHAR_DURATION) )
       )

      {
        ackinfo_timeouts++;

        if (ackinfo_timeouts < 0x10)  // display on LEDs, holding the
          P3OUT = ackinfo_timeouts;   //  highest value that can be shown
      }
  }
```

If the host fails to meet this timeout, a counter is incremented. LED1–LED4 on the interface board display the current counter value in BCD format. The counter does not wrap, but is frozen when it reaches the value 15. It can be reset by pressing the reset button.

4.3 Software UART

Since the MSP430F123 provides only one hardware UART, it was necessary to implement a software emulation of this interface. The best way to avoid jitter and high CPU loads is to use a hardware timer with its capture/compare interrupts. The basic concept of the software UART implementation follows application notes available from Texas Instruments [30, 31]. However, it was rewritten in C, significant performance improvements were made, and parity support was added.

Initialization After connecting the RX pin (port 1.2) and the TX pin (port 1.3) to the capture/compare units one and two of Timer_A, these units have to be set to the right mode. The capture/compare unit at the input pin runs in capture mode; the one connected to the output pin runs in compare mode and is set to high, representing the UART idle level.

Transmission When the transmit function (*swu_tx_byte(unsigned char c)*) is called, the timer is started and the output pin is set to low to transmit the start bit. The character to be transmitted is buffered. Depending on its LSB, the output unit at the TX pin is programmed to bring the pin to the proper level at the next compare interrupt. The capture/compare register is updated with the value representing one bit time. When the compare event occurs and the ISR is called, this bit time value is added again to the capture/compare register value to trigger the next compare interrupt. The output unit is prepared with the value of the second bit of the character to be transmitted. This procedure repeats with all data bits and the parity bit, which has been calculated before. In the last ISR call, the output pin is set to high again for the stop bit and the following idle level.

Reception When a character is received, the level at the input pin changes from high (idle level) to low (start bit). This edge is used to trigger a capture interrupt. In the ISR, the capture/compare unit is set to compare mode and its capture/compare register is loaded with the current timer value plus an increment corresponding to 1.5 UART bit times. Therefore, the ISR is next called in the middle of the first data bit; its value is sampled and buffered. This time, the capture/compare register is incremented by a value corresponding to one bit time to reach the middle of the second data bit. This continues till all data bits and the parity bit are received. The receive callback function is then called and the capture/compare unit is set to capture mode again, making the software UART ready to receive another byte.

References

[1] Automation Systems Group A-Lab website,
http://www.auto.tuwien.ac.at/a-lab

[2] KNX Association, KNX Specifications (Version 2.0), Diegem, 2009.

[3] KNX Handbook, Vol. 3, Part 2, Ch. 2, System Specifications: Communication Media: Twisted Pair 1, v1.1 AS

[4] Siemens EIB-TP-UART-IC Technical Data,
http://www.automation.siemens.com/et/gamma/download/tpuart.pdf

[5] G. Neugschwandtner and A. Fernbach, "Design of an enhanced TP-UART based KNX PC interface", KNX Scientific Conference 2008, St. Katelijne-Waver,
http://www.auto.tuwien.ac.at/g̃neugsch/knxsci08-tpuart_if.pdf

[6] Schematic of the Siemens-TPUART-interface for the serial port,
http://os-projects.fh-deggendorf.de/images/tpuart-interface.png
(retrieved in November 2001, no longer available online)

[7] TP-UART interface by C. Troger,
https://www.auto.tuwien.ac.at/downloads/eib4linux/tp-uart-interface.zip

[8] Disch Systems TP-UART Interface,
http://disch-systems.de/download/tpuart_interface_en-datasheet.pdf

[9] Siemens Bus Transceiver Module PCB Technical Data,
http://www.opternus.com/uploads/media/BTM_PCB_datasheet_V2.0.pdf

[10] F. Praus, W. Kastner and G. Neugschwandtner, "A versatile networked embedded platform for KNX/EIB", KNX Scientific Conference 2006, Vienna.

[11] R. Stocker and A. Grzemba, "Linux Device Driver for the TP-UART interface", EIB Scientific Conference, October 2001, Munich.

[12] W. Kastner and C. Troger, "Interfacing with the EIB/KNX: A RTLinux device driver for the TPUART", Proc. 5th IFAC Intl. Conference on Fieldbus Systems and their Applications (FeT '03), pp. 29–36, 2003.

[13] TP-UART Linux kernel driver updates by R. Buchinger and M. Kögler,
https://www.auto.tuwien.ac.at/a-lab/eib4linux.html

[14] Disch Systems EIB Driver for Linux,
http://disch-systems.de/download/edrv_en-datasheet.pdf

[15] `eibd` daemon homepage,
http://www.auto.tuwien.ac.at/~mkoegler/index.php/eibd

[16] CadSoft online,
http://www.cadsoft.de

[17] TI MSP430x12x Mixed Signal Microcontroller (Rev. C) Datasheet,
http://www.ti.com/lit/gpn/msp430f123

[18] TI MSP430F11x2, MSP430F12x2 Mixed Signal Microcontroller (Rev. D) Datasheet,
http://www.ti.com/lit/gpn/msp430f1232

[19] TI MSP430x1xx Family User's Guide (Rev. F),
http://www.ti.com/litv/pdf/slau049f

[20] ATmega48/88/168 Datasheet (Rev. R),
http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf

[21] ATmega48P/88P/168P Datasheet (Rev. K),
http://www.atmel.com/dyn/resources/prod_documents/doc8025.pdf

[22] NEC 78K0S/KB1+ 8-bit Single-Chip Microcontrollers User's Manual,
http://www.necel.com/cgi-bin/nesdis/o002_e.cgi?litcode=U17446EJ*

[23] ST7LITE1XB Datasheet,
http://www.st.com/stonline/products/literature/ds/11929/st7lit19bf1.pdf

[24] Maxim Application Note 83: Fundamentals of RS-232 Serial Communications,
http://www.maxim-ic.com/appnotes.cfm/an_pk/83/

[25] MAX232, MAX232I Dual EIA 232 Drivers/Receivers Datasheet,
http://focus.ti.com/lit/ds/symlink/max232.pdf

[26] ON Semiconductor MC1488 Quad Line EIA-232D Driver Datasheet,
http://www.onsemi.com/pub_link/Collateral/MC1488-D.PDF

[27] Microchip Appl. Note AN519: Implementing a Simple Serial Mouse Controller,
http://ww1.microchip.com/downloads/en/AppNotes/00519c.pdf

[28] Tomi Engdahl, "Get power out of PC RS-232 port",
http://www.epanorama.net/circuits/rspower.html

[29] United States Patent 7,291,938, "Power supply apparatus and method based on parasitic power extraction", 2007.

[30] Texas Instruments Application Report SLAA078A, "Implementing a UART function with Timer A3", http://focus.ti.com/lit/an/slaa078a/slaa078a.pdf

[31] TI Application Report SLAA307A, "Using the Timer_A UART Library",
http://focus.ti.com/lit/an/slaa307a/slaa307a.pdf

All URLs last visited 2009-11-25.