Intelligent UART Module for Real-Time Applications

Martin Delvai[1], Ulrike Eisenmann[1], Wilfried Elmenreich[1]

[1]Institute of Embedded Computing Systems,
Technical University of Vienna, Vienna, Austria
{delvai, eisenmann}@ecs.tuwien.ac.at
wil@vmars.tuwien.ac.at

Abstract — *More and more fieldbus applications require a communication with real-time properties, while still being economically feasible. The fieldbuses LIN and TTP/A take this requirement into account by providing a temporally deterministic communication protocol that uses a common UART (Universal Asynchronous Receiver/Transmitter) as communication interface. Due to the fact, that UARTs originally have not been designed for this kind of application, some problems arise that require increased software and processing effort or reduce the achievable bandwidth. This paper presents a UART module that has been adopted to be used in real-time applications: A synchronization mechanism is provided, which reduces the software complexity and facilitates the periodical synchronization required by the mentioned real time communication protocols. Further the UART is able to initialize actions on occurring events. In this way, a big part of the communication process can be handled autonomously by the UART module. Send jitter has been eliminated completely and the arithmetic error in baud rate setting has been significantly reduced. Therefore the UART module is able to work with unprecise clock sources with a high drift rate, as it is the case with low-cost RC oscillators.*

1 Introduction

A UART (Universal Asynchronous Receiver and Transmitter) is a standard communication component that is provided by most of the available microcontrollers. In order to supply a low-cost solution, two novel field-bus protocols, TTP/A [1] and LIN [2], specify a common UART as communication interface to the network. Both protocols are central-master UART protocols for low-cost single-chip smart sensor and actuator nodes, which enable a temporal predictable communication [3].

Case studies [4, 5, 6] have shown that an implementation with COTS (commercial-off-the-shelf) hardware is feasible. However, an in-deep analysis of the behavior of standard hardware UARTs has shown that they are hardly suitable for real-time communication [7]. Moreover, both LIN and TTP/A specify a synchronization message that enables a slave node with an imprecise low-cost on-chip oscillator to synchronize with a running network. As a consequence, implementations for LIN [8] and TTP/A [9] exist that prefer

a software-implemented UART to a COTS hardware UART component, leading to increased software complexity for the implementation of the protocol.

The objective of this paper is to present an improved UART design that overcomes the problems of standard hardware UART implementations. The UART contains local intelligence supporting the implementation of real-time fieldbus protocols like LIN or TTP/A. The UART can be configured to react automatically on events or on interrupts. If the indicated event occurs, an action or interrupt is triggered. The combination of events and so-called assigned actions make it possible to perform the time-critical parts automatically without interaction from the microprocessor. The UART is able to recognize the synchronization pattern of LIN and TTP/A on the communication line and to calculate and set the appropriate baud rate. The UART is part of a modular construction system [10] that enables a developer to build an application specific network node with minimal effort.

The remainder of the paper is structured as follows: Section 2 describes the problems with standard UART implementations and states the requirements for an improved UART module. Section 3 presents the architecture and functionality of our proposed UART module. Section 4 gives an evaluation on our experiences with an FPGA implementation of the UART module. Section 5 discusses related approaches for our problem. The paper is concluded with Section 6.

2 Motivation and Problem Statement

As the market segment for low-cost microcontrollers is continually growing, many components get integrated into a system-on-a-chip (SoC). A crucial reduction of costs can be achieved by integrating a complete network node, comprising a microcontroller, a sensor/actuator, a communication unit and an oscillator on a single silicon die. However, state-of-the-art technologies do not support the integration of quartz oscillators, so SoCs are typically equipped with RC-oscillators. The disadvantage of RC oscillators is that their frequency is very sensitive to voltage and temperature changes, therefore a typical RC oscillator shows a rated frequency of 1 MHz $\pm$ 50% and a frequency-drift of up to $\pm$ 10% per second.

Deficiencies arising from the usage of such oscillators are analyzed in [7] and can be summarized as follows:

Arithmetic error in baud rate setting: The baud rate setting of a standard UART usually is configured by integer values of the system clock. The baud rate depends on the frequency of the UART clocking and is set by choosing an UBRS (UART Baud Rate Setting) value as follows:

$$BaudRate = \frac{f_{clk}}{C_1 \cdot (UBRS + C_2)} \tag{1}$$

The integer constants C_1 and C_2 depend on the UART implementation. Typically, C_1 represents the number of samples per bit cell and C_2 is 1.

Thus, conventional UARTs need clock oscillators calibrated for a special frequency (for example a quartz with a frequency of 4.9152 MHz instead of 5.000 MHz) to achieve standard baud rate settings such as 19,200 bit/s.

Clock drift: Another problem evolves when the clock source shows a high frequency drift since this directly affects the baud rate of the UART. For example if the frequency drifts from 1MHz to 1.1MHz, the following error arises:

$$baudrate = \frac{f_0 = 1MHz}{UBRS = 50} = 19200 \qquad baudrate = \frac{f_0 = 1.1MHz}{UBRS = 50} = 20200 \quad (2)$$

In order to deal with the drift rate problem communication protocols specify means of synchronization where a receiving component can derive the baud rate from the data bits on the bus. The LIN and TTP/A protocol provide a regular bit pattern as depicted in figure 1. Both protocols specify an identical pattern which can be used for baud rate detection at startup or for periodic clock synchronization.

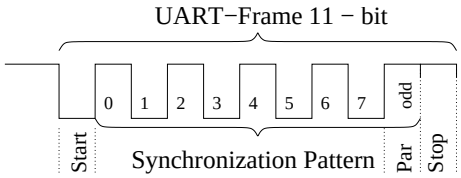

Figure 1: Synchronization Pattern

Send jitter: Another deficiency of standard UARTs used in real-time communication is the fact that the point of time for transmission cannot be adjusted with sufficient precision: we call this problem the *send jitter* problem. Due to the internal clocking, a UART transmission will be only initiated at certain instants. In conventional UART implementations the baud rate generator permanently runs in the background and periodically generates potential transmission points. Thus, the start of a UART transmission may be delayed by an unpredictable duration of up to one bit time. As a result the user cannot recognize from the outside, when the next pulse is produced.

$$t_{jitter} = [0..\frac{1}{BaudRate}) \quad (3)$$

Regarding the considerations above, it can be seen that standard UARTs are not very suitable for the requirements of real-time communication under these boundary conditions.

It is possible to use a software UART implementation but at the cost of node performance. In order to send or receive a single bit a software-implemented UART typically needs to execute about $15...40$ machine instructions. Moreover, since a software UART implementation performs only one sampling per bit cell, this approach is also more vulnerable to short spike interferences on the communication medium.

So the demand for a special UART which overcomes the disadvantages of standard UARTs evolves. Thus, our motivation has been to realize a UART that fulfills the following requirements:

Temporal predictability: A revised UART design should not be susceptible to arithmetic errors in baud rate setting or to send jitter problems.

Support for automatic synchronization: The synchronization pattern specified by the LIN and TTP/A protocol should be supported.

Support for reduced software complexity: The UART should support typical tasks in real-time systems such as providing timestamps or triggering actions on special points of time in order to minimize the required user program. This reduces complexity and increases performance.

High flexibility: The UART should be easily adaptable to many applications and designs.

Note that the requirement for flexibility and reduced software complexity are very difficult to fulfill both at the same time. Thus, special care has to be taken on the design of the programming interface to the UART component.

Another advantage of a customized UART implemented in hardware could be that due to the reduced software complexity, the required memory for the application will decrease so that costs can further be reduced by achieving a smaller silicon die size for the overall system.

3 UART-Module

The UART is realized as generic extension module [10] and communicates with the processor core over a very slim interface. The extensions are mapped to the top address space of the data memory. For the processors the extension modules are only storage positions which can be accessed with simple load and store instructions. Therefore, from the processor's point of view it makes no difference, whether the extension is a simple sensor, actuator, a complex floating-point unit or a UART. Figure 2 shows the generic interface of these extension modules.

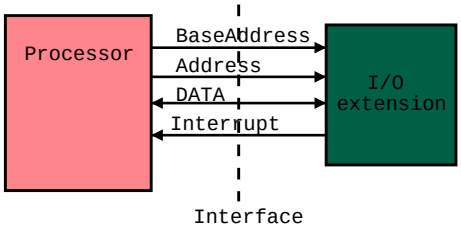

Figure 2: Interface of extension modules

In order to be suitable for real time applications our UART module combines a pure UART with a timer unit. Figure 3 shows the block diagram of the UART module.

The eight 16 bit registers, depicted in the left corner of the schematics provide the interface to the processor core. The UART control unit is the central part of the UART. It reads the values of the interface register and generates the control signals of all other components. The enhanced baud rate generator generates the signals for the receiver and the transmitter. The error control unit checks the communication process and signalizes when an error occurs. The bus interface contains a hardware filter, which preprocesses the input signal from the bus and guides it through to the receiver unit to achieve robustness against spike interferences.

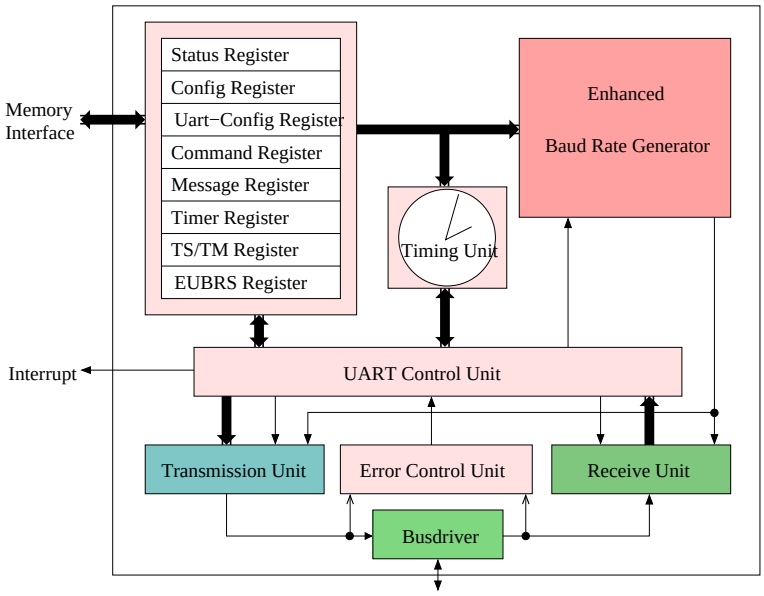

Figure 3: Block diagram of the UART extension module

The UART is controlled over eight 16 bit registers, whereby the first two registers are the *status* and the *config* register.

The semantic of the generic parts of the status and control register are universally defined for all extension modules. The other registers form a module-specific interface that enables configuration and data exchange.

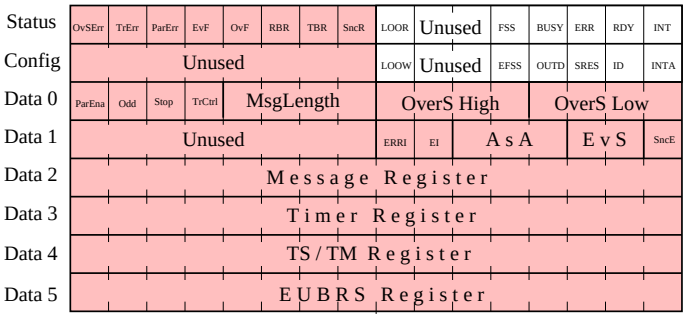

Figure 4: Registers of the UART extension module

The module specific part of the **status register** reflects the state of the UART. *Data 0* is used to define communication parameters like message length, type of parity bit, number of stop bits. The UART performs 32 samples per bit cell - with *OverS_High* and *OverS_Low* the upper and lower bounds can be specified, where a bit is interpreted as '1', undefined, or '0'. Commands to the UART are written into the *Data 1*-register. The assigned action bits *AsA* specify the reaction of an event defined by the event selection bits *EvS*. Table 1 shows the combinations of events and assigned actions that are supported by our UART.

Using the combination of events and assigned actions makes it possible to specify the

EVENTS	EvS BIT CODE	ASSIGNED ACTIONS	AsA BIT CODE
no event	00	start send transmission	011
no event	00	enable receive mode	100
no event	00	disable receive mode	101
no event	00	timestamp	001
no event	00	timerreset	010
startbitdetection	01	timestamp	001
startbitdetection	01	timerreset	010
startbitdetection	01	disable receive mode	101
receive completion	10	timestamp	001
receive completion	10	timerreset	010
receive completion	10	disable receive mode	101
timermatch	11	timerreset	010
timermatch	11	start send transmission	011
timermatch	11	enable receive mode	100
timermatch	11	disable receive mode	101

Table 1: Possible events and assigned actions

behavior of the UART module. For example if the module should determine the time of the start of the next transmission, the EvS bits are set to 01 and the AsA bits are set to 001.

The message register contains the data to be received or transmitted. The current value of the integrated timer is maintained in the timer register, while the TS/TM contains timestamps or timer match values. The EUBRS register is used to set the baud rate. The UART module is able to automatically detect the baud rate and to set the appropriate value for this register. If the programmer wants to force a particular baud rate, he or she can set the EUBRS register directly.

4 Evaluation

We implemented a prototype of our UART module in an Altera Apex 20ke FPGA. The module requires 699 LE [1] and runs with a peak clock frequency of 42 MHz. Even in extreme cases, for example at a clock frequency of 0.5 Mhz and a reference baud rate of 115,000 baud, the UART is able to synchronize its baud rate with a deviation of 1% from the reference baud rate. Measurements concerning delay and send jitter confirm the theoretical approach presented in the previous section: After a new value has been written into the command register, it takes exactly one clock cycle to start the synchronization, to enter the fail safe state or to process the indicated event (the *no event* event as well). Also, if an event occurs, it takes exactly one clock cycle to process the assigned action and all registers are written with one clock cycle delay.

Our UART provides the following mechanism to handle the problems described in Section 2:

Temporal predictability: The arithmetic error problem is solved by enhancing the func-

[1] LE: Logic Element - is the smallest programmable unit within Altera's FPGA

tionality of the baud rate generator: In the EUBRS-register the first twelve digits are interpreted as integer part of the timer constant and the latter four digits are the fractional part of the timer constant. Four digits for the fractional part are completely sufficient [7]. The UART calculates the duration of a bit for transmission by using a fixed comma value EUBRS according to the following formula:

$$t_{bit} = \frac{EUBRS}{16} \cdot \frac{1}{f_{clk}} \cdot \frac{1}{2} \tag{4}$$

In contrast to standard UARTs, our module starts the enhanced baud rate generator immediately after receiving the transmission command. In this way the send jitter is eliminated completely.

Support for automatic synchronization: Setting the **SncE** flag in the command register causes the module to listen for the synchronization pattern. The synchronization pattern is always preceded by a duration of bus silence. When the synchronization mode is set, the UART is disabled until the module captures a correct synchronization pattern. When the first falling edge of the pattern has been detected, the timer resets its value, counts the time for one bit and compares it with the following, until the end of the synchronization pattern is recognized. A difference of 4 % bit length is allowed. The value of the timer register is then copied directly into the EUBRS register.

$$\mathrm{EUBRS} = \frac{TimerConstant}{16} \tag{5}$$

In this way the considerable software effort to synchronize a node is reduced to setting a flag in the UART module.

Support for reduced software complexity: The fusion of a timer and a UART leads to a powerful communication module for real time application. As described above the UART is able to initialize autonomous actions at given points in time. On the other hand the UART can be used to measure points in time of incoming messages.

High flexibility: Although the design of the UART is optimized for real time applications, the module can be used as a simple timer or like a conventional UART module.

5 Related Work

Our UART is designed to be used as extension module in a modular construction system [10]. This system comprises several processor cores, which can be equipped with different extension modules. Altera offers a similar solution to speed up the development of customized micro controllers, called Nios® Embedded Processor System Development Kit. This development system comprises a UART as well. Similar to our UART, the Altera's UART is mapped into the data memory of the processor core and is controlled over five 16-bit registers.

In contrast to our UART, the Alteras module does not provide a mechanism to synchronize its baud rate to a reference baud rate – this limits the applicability of this UART

for fieldbus networks when imprecise oscillators are used. The baud rate of the UART is defined by $BaudRate = \frac{f_{clk}}{Divisor+1}$, where *Divisor* can be specified in the respective interface register. Thus, when arbitrary oscillator frequencies are used, a significant arithmetic error in baud rate setting occurs. Furthermore, Altera's UART does not include its own clocking source but needs an additional timer unit instead. Due to this separation of UART and timer an efficient interaction by event and assigned action schemes cannot be achieved. The documentation found on Altera's homepage does not give any information about the send jitter. Altera's UART module requires 293 LE, the timer requires 246 LE, this results in a total amount of 539 LE for both modules which is almost the same size as our customized UART.

As an alternative to a UART implementation, it would be possible to use a microcontroller that features a time programming unit (TPU), where the TPU could be used to act as a customized UART. TPUs are available on most Motorola embedded microcontrollers [11]. A TPU offers a fine resolution for waveform generation and measurement, while it is programmable in a machine language that allows great flexibility. However, since TPU programming can become very complex, the TPU approach does not decrease software complexity, it is a propriety system from Motorola, and its implementation requires more die size than our customable UART approach.

6 Conclusion

We have presented a UART module that supports real-time networks with low-cost nodes with imprecise RC oscillators. The UART supports a synchronization mode that allows it to adjust its baud rate automatically. Furthermore it is able to recognize events (such as timer match, begin or end of a transmission, etc.) and to perform assigned actions (such as timestamping or initiating a transmission, etc.) when a particular event occurs. In this way, a big part of the communication process can be handled autonomously by the UART module.

Timing problems (such as arithmetic error in baud rate setting, send jitter) that exist with standard UART implementations have been fixed. Send jitter were eliminated completely and arithmetic error in baud rate setting was significantly reduced. In that way, the UART module is able to work with unprecise clock sources with a high drift rate, as it is the case with low-cost RC oscillators.

The presented UART is well apt to support implementations of LIN or TTP/A smart transducer networks. Currently, we are porting the TTP/A protocol to a microcontroller equipped with our UART module.

Acknowledgments

This work was supported by the Hochschuljubiläumsstiftung der Stadt Wien via project MOSAIC (H-1147/2002).

References

[1] H. Kopetz et al. Specification of the TTP/A protocol. Technical report, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, March 2000. Available at http://www.ttpforum.org.

[2] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification and LIN press announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.

[3] H. Kopetz, W. Elmenreich, and C. Mack. A comparison of LIN and TTP/A. In *Proceedings of the 3rd IEEE International Workshop on Factory Communication Systems*, pages 99–107, Porto, Portugal, September 2000.

[4] P. Peti and L. Schneider. Implementation of the TTP/A slave protocol on the Atmel ATmega103 MCU. Technical Report 28/2000, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, August 2000.

[5] Atmel Corporation. *LIN Protocol Implementation on the T89C51CC01/02*, March 2003. Application note available at http://www.atmel.com.

[6] R. Obermaisser and A. Kanitsar. Application of TTP/A for the Otto Bock Axon bus. Technical Report 27/2000, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, July 2000.

[7] W. Elmenreich and M. Delvai. Time-triggered communication with UARTs. In *Proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'02)*, Västerås, Sweden, August 2002.

[8] Atmel Corporation. *AVR 308: Software LIN Slave*, May 2002. Application note available at http://www.atmel.com.

[9] C. Trödhandl. Architectural requirements for TTP/A nodes. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[10] M. Delvai, U. Eisenmann, and W. Huber. Modular construction system for embedded real-time applications. In *Tagungsband of Austrochip 2002, Vienna, Austria*, 2002.

[11] R. Soja. Inside Motorola's TPU. *Dr. Dobb's Journal*, December 1996.