UART Module Specification for the SPEAR Architecture

W. Elmenreich

Institut für Technische Informatik
TU Vienna, Austria
wilfried@vmars.tuwien.ac.at

October 19, 2001

Abstract

This paper depicts the specification for a communication I/O module for a custom microcontroller that supports the Time-Triggered Protocol/Class A. An example for such a communication controller with special purpose already exists for TTP/C. Based on the Scalable Processor for Embedded Applications in Real-Time environments (SPEAR) this paper defines an I/O module for the TTP/A communication. This communication is based on a UART transmission, but analysis of the timing has shown, that a custom UART would be much more appropriate for TTP/A communication than a commercial-off-the-shelf UART module.

Keywords: *SPEAR, TTP/A, microcontroller,architecture, UART.*

Introduction

With the advent of systems-on-a-chip solutions for embedded systems the concept of a sensor with a MEMS sensor, a microcontroller unit and a network interface on a single silicon die came up. In contrast, commercial-off-the-shelf hardware offers a low price, but often do not provide some necessary features like small size or low power consumption.

We decided to build a microcontroller system for a sensor/actuator network called TTP/A based on the processor architecture SPEAR (Scalable Processor for Embedded Applications in Real-Time Environments) [1]. TTP/A is a multimaster UART-based sensor network that provides temporal predictable communication to sensor and actuator nodes in a distributed real-time system. It provides multiple views of a sensor node, such as a periodic real-time service view and an aperiodic diagnostic view.

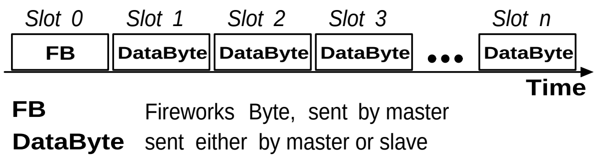
Although the protocol was originally designed for COTS hardware, analysis of the required timing [2] has shown, that standard hardware UARTs refuse to work with most on-chip clocking units.

It is the objective of this paper to define the requirements for a UART I/O-module that complies to the SPEAR architecture.

The rest of the paper is organized as follows: Section 1 examines the timing requirements of the TTP/A fieldbus protocol. Section 2 describes the properties of the scalable processor for embedded applications in real-time environments. Section 3 defines the interface of an appropriate SPEAR I/O module for UART communication. In the following Section 4 a method for handling spike interferences is proposed. The paper is concluded and an outlook is given in section 5.

1 Fieldbus Protocol Properties

TTP/A is a time-triggered master/slave communication protocol for fieldbus applications that uses a time division multiple access (TDMA) bus arbitration scheme [3]. It is possible to address up to 255 nodes on a bus. One single node is the active master. This master provides the time base for the slave nodes. The communication is organized into rounds. Bus access conflicts are avoided by a strict TDMA schedule for each round. A round consists of several slots. A slot is a unit for transmission of one byte of data. Data bytes are transmitted in a standard UART format. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of round.



Slot 0
Slot 1
Slot 2
Slot 3
Slot n
FB
DataByte
DataByte
DataByte
DataByte
Time
FB Fireworks Byte, sent by master
DataByte sent either by master or slave

Figure 1: A TTP/A Round

A TTP/A round (see Figure 1) consists of a configuration dependent number of slots and an assigned sender node for each slot. TTP/A distinguishes between two types of rounds, master/slave rounds, which have a fixed layout and multipartner rounds, which are configured in a data structure called RODL (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the semantics of each individual slot, and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round.

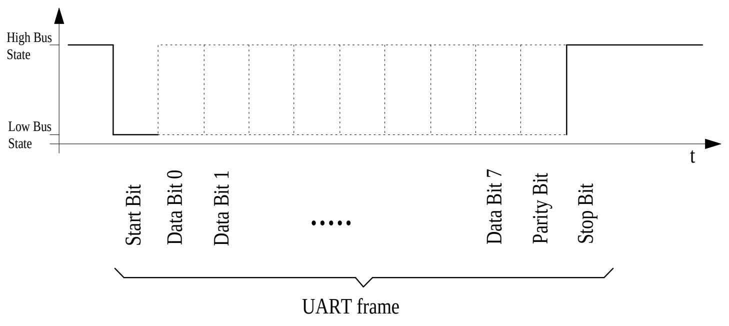


Figure 2: TTP/A UART Frame

A TTP/A slot consists of a UART frame as depicted in Figure 2. This UART frame transmits a message of one byte and a parity bit. For the first byte of a round, the fireworks byte, odd parity is used, thus the number of set bits in message and parity is an odd number. All other bytes are sent with even parity, thus the number of set bits in message and parity is an even number. Each UART frame is sent in a time window of 13 bits, so that a UART frame has a timing tolerance of $+/-1$ bitcell time.

One of the fireworks bytes sent by the master contain a regular bit pattern, in order that slaves with imprecise on-chip oscillators can synchronize their local clocks. Figure 3 depicts the synchronization pattern. The pattern is preceded by at least 12 bit times of bus silence and contains each 5 transitions from high to low bus state and vice versa. Interpreted as a UART frame the synchronization pattern reads as message 0x55 with an odd parity bit.

The protocol implementation needs at least one 16 bit timer unit. If the number of timer ticks which corresponds to the longest timeout in the protocol specification is below 2^{16}, a clock prescaler is not needed. This denotes the *lower* bound of transmission speed. E.g. for a system with a timer clocked by 10 MHz the transmission rate must be 3500 Baud or higher, if the timer is clocked with 1 MHz the minimal transmission rate is 350 Baud.

For evaluation purposes, the TTP/A Basic protocol has been implemented on an 8-bit ATMEL microcontroller [4]. The UART was implemented in software to provide compatibility to smallest (and therefore cheapest) MCUs. Furthermore standard UART implementations do not offer arbitrary bitcell lengths. This is important, because of the impreciseness of the internal RC oscillator, which was choosen for economical reasons. The UART-speed has to be adjusted each TDMA round to the master's reference speed.

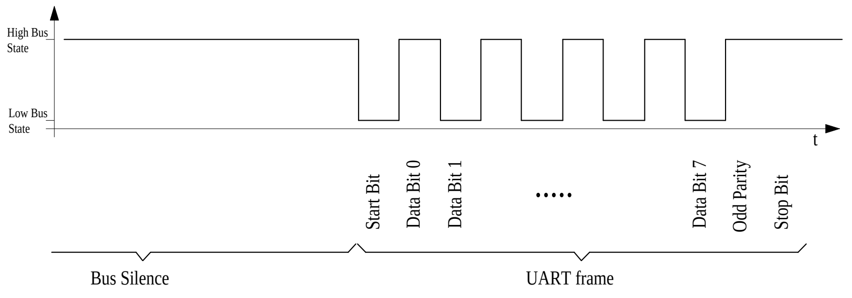


Figure 3: Synchronization Pattern

The evaluation of this implementation lined up, that the CPU utilization has its maximum during UART operations. An appropriate hardware UART would disburden the CPU.

2 The SPEAR Architecture

Motivated by the increasing size of embedded systems, an architecture for a customable microprocessor architecture was devised at the VLSI design group of the Vienna University of Technology [1].

The scalable processor for embedded applications in real-time environments is optimized for middle-performance, fast responding microsystems with deterministic timing behavior. All components are accessed over well-defined interfaces, thus a system design can be extended by I/O modules to fit a special application. Although each SPEAR hardware system can be compiled individually, a set of tools can easily be adopted to each SPEAR system.

Figure 4 depicts the architecture of the central SPEAR component.

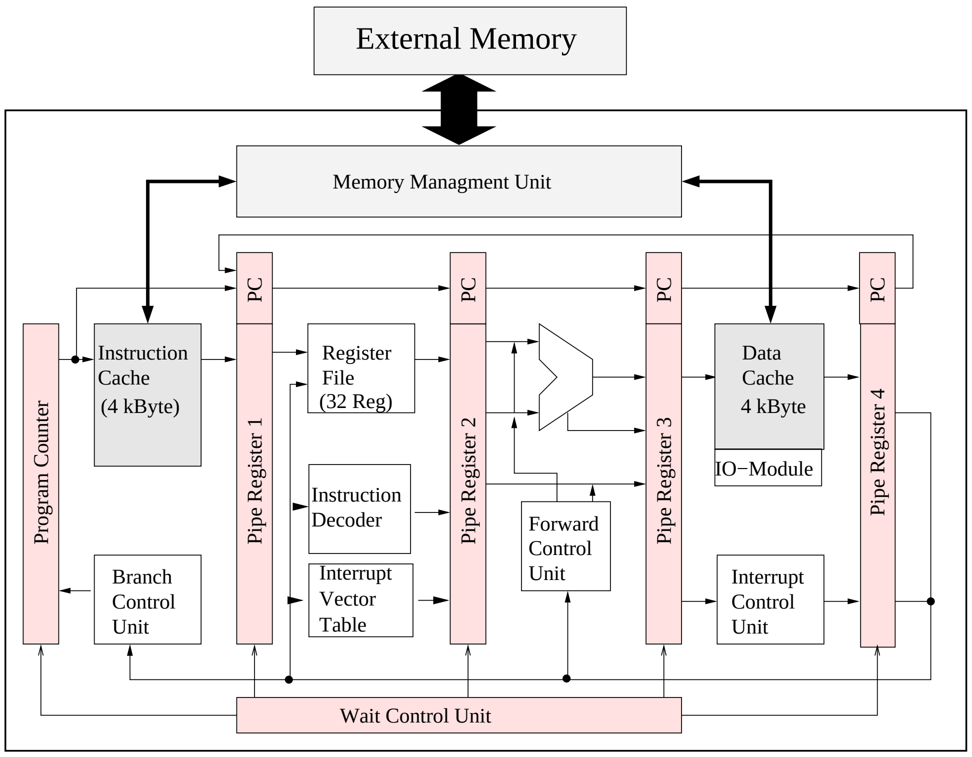


Figure 4: SPEAR Architecture

SPEAR features a 16 bit processor that executes instructions via a 4 stage pipeline. All processor parts are synchronized by a Wait Control Unit which resolves control and data hazards.

There are 20 general purpose registers and 12 control, pointer, and configuration registers.

SPEAR comprises a 4 kB instruction cache and a 4 kB data cache. Usually these values suffice for many embedded applications, but it is also possible to add up to 128 kB external memory for instructions and 127 kB external data memory.

SPEAR supports up to 64 I/O modules which are mapped into the data memory. Examples for I/O modules are

- Internal units such as additional timers or floating point units.
- External units such as sensors and actuators or network interfaces.

The UART module described in the next section will be an I/O module belonging to the latter type.

3 UART Module Interface Description

The UART is realized as generic I/O module. The interface between such modules and processor is defined in [5]. The interface contains signals for reset, addressing, data I/O, and an interrupt. Additionally every interface comprises 8 16 bit registers. The first two registers are the *status* and the *config* register. These registers have standard assignments for all I/O modules, the remaining registers are module-specific.

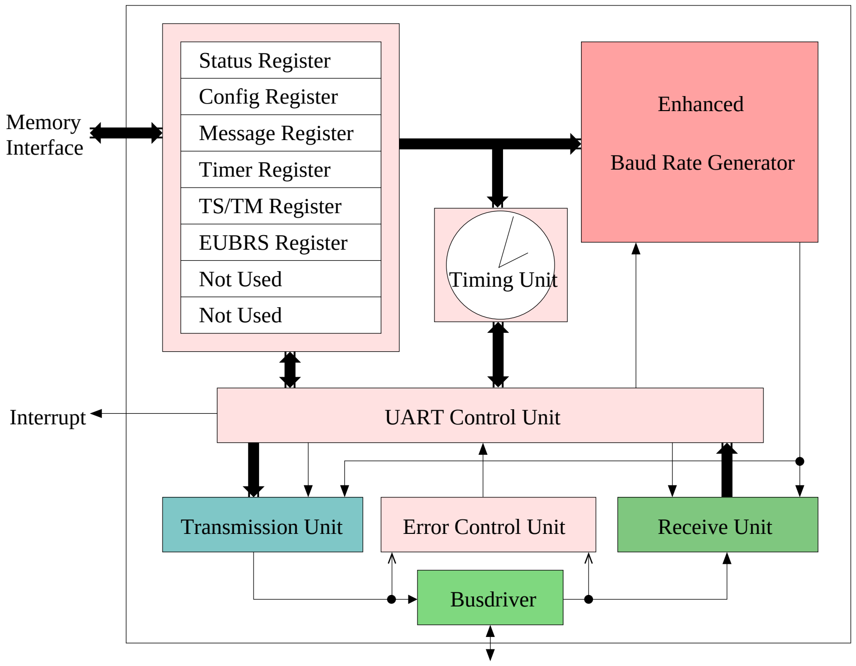


Figure 5: Block Diagram of the UART I/O Module

Elmenreich and Delvai [2] lined out some deficiencies of standard UART modules:

Send Jitter: Due to its internal clocking, a UART transmission will be only initiated at certain instants. These instants are set periodically and the period time is a bit time at the given baud rate or an integer multiple of this baud rate. Thus the start of a UART transmission can be delayed by an unpredictable duration of up to one bit time in conventional UART implementations.

Baud Rate Setting: The baud rate setting in conventional UART implementations can only be done as an integer fraction of the system clock. Thus conventional UARTs need clock oszillators calibrated

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Not Used			EvF	Ovf	RBR	TBR	SncR	LooR	Res		FSS	Busy	Err	Rdy	INT

Figure 6: Status Register Layout

for a special frequency (e.g. 4.9152 MHz instead of 5.000 MHz) to achieve standard baud rate settings such as 19,200 bit/sec.

The UART described in this section will overcome the above disadvantages. It starts immediately after receiving the transmit signal with the message transmission. In this way the send jitter is eliminated totaly.

The enhanced UART baud rate setting (EUBRS) is the key element of the UART. The EUBRS calculates the duration of a bit for transmission by using a fixed comma value EUBRS according to the following formula:

$$t_{bit} = \frac{EUBRS}{16} * \frac{1}{f_{clk}} \tag{1}$$

Because EUBRS is a 16 bit register the maximum bit duration calculates to $(2^{12} - 2^{-4})$ timer ticks. The duration of a transmitted bit can be defined with a granularity of $\frac{1}{16}$ of a system clock period.

Figure 5 lists the block diagram and the register layout for the UART I/O module.

The registers have the following assignments:

Status: This register contains the standard status bits for all user modules and 8 bit for module-specific information. This register is read-only. The layout of the status bits is depicted in Figure 6. The standard status bits are:

INT: The interrupt flag (INT) is set when the interrupt line is activated. This flag is cleared when the processor sets the INTA bit in the config register.

Rdy: The ready bit (Rdy) shows when then module is ready-to-operate.

Err: Denotes an unexpected error.

Busy: If set, the module is not ready for new commands. This flag is never set in the UART module.

FSS: This bit tells if the module is in Fail Safe State (FSS). The Fail Safe State of the UART module is equal to the output disable function by setting the OUTD bit in the config register.

Res: These bits are reserved for future use.

LooR: For module detection, this bit shows the value of the LooW bit in the config register for each active I/O module.

The module-specific status bits are:

SncR: This synchronization ready flag is set when the next synchronization pattern has been catched and the EUBRS registers have been rewritten. When SncR is set, the corresponding SncE bit in the config register is cleared.

TBR: This transmit buffer ready flag is set when the next message can be written into the message register. Writing a new message clears this flag.

RBR: This receive buffer ready flag is set when the received message can be retrieved from the message register. Reading a new message clears this flag.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pre	EI	AsA			EvS		SncE	LooW	Res		EFSS	OutD	SRes	Id	INTA

Figure 7: Config Register Layout

- **Ovf:** This receive overflow flag is set when a new message was received while the Receive Buffer Ready flag was still set.
- **EvF:** The event flag is set when the event specified in the config register occurred.

Config: This register contains the standard function for all user modules and 8 bit for module-specific settings. The layout of the config bits is depicted in Figure 7. The standard bits of this register are:

- **INTA:** This interrupt acknowledge (INTA) bit must be set by the interrupt service routine to clear a pending interrupt.
- **Id:** A set identify bit (Id) causes the module to map identification information into its address space.
- **SRes:** The soft reset bit (SRes) enables the software to reset the module.
- **OutD:** If set, the output to the bus is disabled. However the module is able to perform the synchronize function and can receive messages from the bus.
- **EFSS:** Setting this bit causes the module to enter the fail-safe state. For the UART module the fail-safe state is equal to the output disabled mode.
- **Res:** These bits are reserved for future use.
- **LooW:** For module detection, the bit written to this register can be read in the LooP bit of the status register when the module is present.

The module-specific config bits are:

- **SncE:** Setting this synchronization enable flag causes the module to listen for the synchronization pattern (preceded by a slot of bus silence). The UART is disabled, when in synchronization mode. When the module captures such a synchronization pattern it recalculates the values in the EUBRS register, sets the SncR bit in the status register, and terminates the synchronization mode by clearing the SncE bit.
- **EvS:** The event selector (EvS) is a combination of two bits that select a special event according to table 1. Events are used to trigger an action or interrupt.
- **AsA:** The assigned action (AsA) to a certain event is defined by a group of 3 bits according to table 2.
- **EI:** Setting this bit causes the event specified in the event selector to create an event interrupt (EI).
- **Pre:** Setting this bit enables a 1 : 4 prescaler for the timer unit. Otherwise the timer uses the native processor clocking. The prescaler function is useful, when low baud rate speeds ($< 5kBit/sec$) are used with a fast clocked ($> 20MHz$) system.

Message: Reading this register yields the last received message and clears the Receive Buffer Ready flag in the Status Register. Writing this register sets the message to be transmitted and clears the Transmit Buffer Ready flag in the Status Register.

Timer: This register sets the module's clock, when written and gives the actual timer value when read.

Time Stamp/Timer Match (TS/TM): Write access to this register sets the timer value for the specified action, read access yields the time stamp of the last enabled event.

Uart Baud Rate Setting (UBRS): This register contains the integer value of the baud rate setting. It is either set by the synchronization function or written by software.

Enhanced Uart Baud Rate Setting (EUBRS): This register contains the fractional part of the baud rate setting. It is either set by the synchronization function or written by software.

Event	Bit Code
No Event	00
Startbit Detection	01
Receive Completion	10
Timer Match	11

Table 1: Event Codes in the Config Register

Assigned Action	Bit Code
No Action	000
Make Timestamp	001
Reset Timer	010
Start Send Transmission	011
Enable Receive Mode	100
Disable Receive Mode	101

Table 2: Action Codes in the Config Register

4 UART Sampling Points

Most common UARTs perform multiple samples for one bitcell. Typical 16 samples are made within one bitcell. The binary results of these samplings are usually fused to a single value by majority voting. Thus the UART is less vulnerable to spike interferences. However oversampling comes with the cost of reduced maximum UART speed and can be a reason for arithmetic errors when the UART has to be set to a particular communication frequency that is not in an appropriate ratio to the UART clocking unit. Furthermore it is difficult to establish a communication between two UARTs with slightly different baud rates, because the first or last few samples may detect information from a preceding or following bitcell.

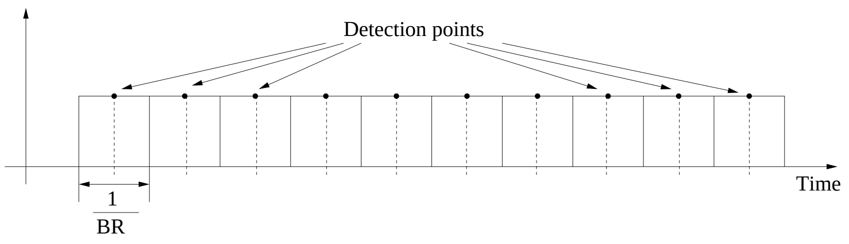


Figure 8: Detection Points of the Proposed UART Design

To avoid these problems, our proposed UART design will have only one detection point in the middle of each bitcell (see Figure 8. To achieve firmness against spike interferences we propose a hardware filter that preprocesses the input signal from the bus. An schematic example for such a filter is given in

Figure 9. The output signal will only change if the input signal is steady for at least two clock cycles, thus interferences shorter than this duration are filtered. The depicted schematic is only a proposal. In the final UART design, the filter should be adaptable to the desired baud rate.

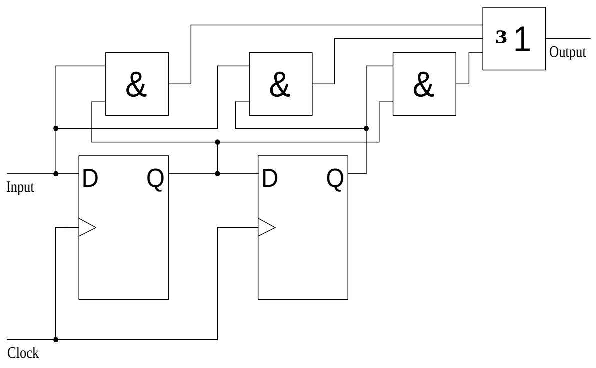


Figure 9: A Filter for Spike Interferences

5 Outlook and Conclusion

The specified UART module overcomes the timing difficulties of standard UARTs in combination with imprecise on-chip oscillators. Both, the send jitter problem and arithmetic error in baud rate setting have been eliminated with the presented design.

Although this UART was customized for TTP/A communication, it is also applicable in other fields. The LIN protocol specifies also a UART communication and aims at the low-cost fieldbus sector. Thus the UART specified in this paper will also work for a LIN environment with imprecise clock oscillators.

Furthermore the usage of special calibrated quartz oscillators (for example 4.9152 MHz or 7.3728 MHz) to avoid the arithmetic error in baud rate setting can be omitted with the new design.

A UART following the concept has already been implemented in an FPGA module and tested [6]. The next step will be the implementation as a SPEAR module according to this specification.

References

[1] M. Delvai, A. Steininger, W. Huber, and B. Rahbaran. SPEAR - Design-Entscheidungen für den “Scalable Processor for Embedded Applications in Real-Time Environments”. Technical report, Technische Universität Wien, Institut für Technische Informatik, 2001.

[2] W. Elmenreich and M. Delvai. Using UARTs for time-triggered communication. Technical Report 19, Technische Universität Wien, Institut für Technische Informatik, 2001.

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

[4] M. Holzmann and W. Elmenreich. Implementation details on the TTP/A slave protocol. Technical Report 4, Technische Universität Wien, Institut für Technische Informatik, July 1999.

[5] W. Huber. Peripherieanbindung an SPEAR. Technical report, Technische Universität Wien, Institut für Technische Informatik, 2001.

[6] M. Delvai. Entwicklung eines Mikrokontrollers für das Echtzeitprokoll TTP/A. Master’s thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2000.