A Prototype Implementation of a TTP/C Controller

Hermann Kopetz, René Hexel, Andreas Krüger, Dietmar Millinger, Roman Nossal,
Andreas Steininger, Christopher Temple, Thomas Führer, Roman Pallierer, Markus Krug†

Institut für Technische Informatik, Technische Universität Wien,
Treitlstr. 3/3/182-1, A-1040 Vienna, Austria

†Daimler-Benz F1M/E
T721, D-70546 Stuttgart, Germany

ABSTRACT

The SAE has classified automotive electronics into two major categories, body electronics and system electronics. The latter (SAE class C) comprises safety critical functions that are of vital importance for the movement of the vehicle. This paper presents a prototype implementation of TTP, a time-triggered communication system developed for this type of applications. The main purpose of the prototype is to provide a means for verifying the concepts of TTP.

The paper focuses on a description of the hardware developed for the communication system as well as the protocol software. The TTP controller hardware is a custom made industry pack module which allows the use of TTP in conjunction with a wide variety of existing motherboards and host environments. The protocol software consists of a TTP/C protocol state machine and auxiliary modules to access the various hardware units.

INTRODUCTION

Automotive electronics can be classified into two major application domains. *Body electronics*, such as interior and exterior lights or power window control, comprises all applications not directly linked to the movement of the car (SAE classes A and B [1]). *System Electronics*, on the other hand, comprises functions that directly control the movement of the vehicle (SAE class C), like ABS, engine control, or steer-by-wire applications. These two domains have different requirements on their underlying communication system, especially with regard to their timing requirements and their fault-tolerance capability.

Presently, body electronics is considered a mature field of engineering practice with a number of available network controller devices, whereas system electronics is still an active area of research with none of the currently existing products satisfying the demands for safety critical class C applications [2]. TTP/C is a time-triggered communication protocol for safety-critical distributed control systems that fulfills the stringent demands of SAE class C applications.

This paper presents a discrete component prototype implementation of TTP/C. We will give a rather detailed description of the hardware and the software design and their interaction. Our concept comprises a number of innovative features, above all a strict separation of a communication and a host subsystem by the so called message base interface. Hence the main purpose of this implementation was a feasibility study of the concepts; performance was only of minor concern.

The paper is structured as follows. The next section gives an overview of the time-triggered protocol. We describe both, the system as well as the TTP principle of operation. In section three the hardware architecture of the prototype is explained. Section four is devoted to a description of the TTP/C software. The paper is concluded in section five.

THE TIME-TRIGGERED PROTOCOL — AN OVERVIEW

SYSTEM OVERVIEW – A real-time application can be decomposed into a set of subsystems, called clusters, e.g., a controlled object (i.e., the machine that is to be controlled), and the controlling computer system.

Computational Clusters – The controlling computer system consists of at least one *computational cluster*.

Such a computational cluster comprises a set of self-contained computers (*nodes*), which communicate via a broadcast bus using the TTP Protocol [6]. An approximate *global time base* is established throughout the cluster by synchronizing the clocks located within the nodes. Each node is considered to be *fail-silent*, i.e., only crash failures and omission failures can occur. On the cluster level, node failures and communication failures can be masked by replicating the nodes and grouping them into *fault-tolerant units (FTUs)*. Message transmission is replicated in both the space domain, by using two busses, and the time domain, by sending the messages twice on each bus.

Within a computational cluster, the *communication subsystem* manages the global concern of providing reliable real-time message transmission. The *host subsystem* comprises the host CPUs of each node computer, which execute the local real-time application. The interface between these two subsystems — the communication network interface — is called the *Message Base Interface (MBI)*, providing host CPUs with a memory area for submitting and receiving messages and for obtaining status and control information about the real-time network.

Node Computers – Figure 1 shows the schematic structure of a TTP node computer. The system-wide partitioning into host subsystem and communication subsystem is reflected by the design of the node computer hardware. We find a host subsystem executing the local part of a distributed real-time application. For the TTP/C prototype implementation described in this paper, the host subsystem is of minor relevance, as our work concentrates on the communication subsystem.

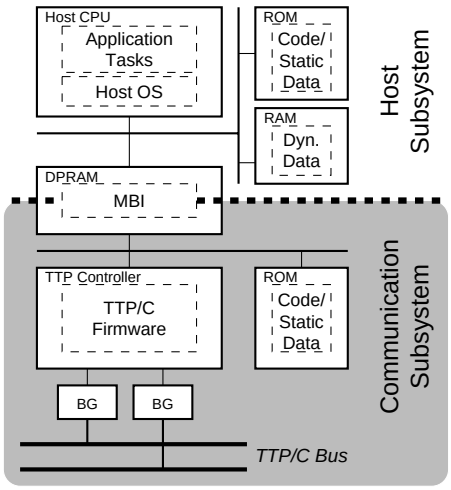

Figure 1: Structure of a Node Computer

The MBI is implemented with a dual-ported memory and represents the interface to the communication subsystem, which executes the real-time communication protocol TTP. The protocol code as well as static configuration data are stored in a ROM device. The TTP controller is supported by two *bus guardians (BGs)*. Each channel is protected by one of these devices, which protect the bus from being monopolized by a faulty node sending at arbitrary points in time (*babbling idiot failure*).

TTP/C: PRINCIPLE OF OPERATION – In a time-triggered architecture all information about the behaviour of the system, e.g., which node has to send what type of message at a particular point in time, is known a priori — at design time — to all nodes of the ensemble. TTP makes best use of this a priori information to reduce the number and size of messages, for example, by retrieving the message identification from the a priori known time of message reception.

TTP is an integrated time-triggered protocol that provides prompt transmission of messages with high data efficiency, a responsive membership service, a fault-tolerant clock synchronization service, mode change support, error detection with short latency, and distributed redundancy management. In the following an overview of the main features of TTP is provided. The interested reader is referred to [6] to find a comprehensive description of the protocol.

Frame Formats – TTP distinguishes two frame types. *I-frames* (initialization frames) are used for system initialization. They contain the internal state of the TTP controller in their data field. This allows integrating nodes to participate in the protocol when they receive an I-frame. I-frames are sent by the communication subsystem

1. during the startup phase of the protocol (cold start after power-up), and
2. at predefined intervals during normal operation of the protocol to facilitate re-integration of failed nodes.

N-frames (normal frames) are used during normal operation and contain application data. The header byte of an N-frame contains three fields: the first bit identifies the message type, then a three bit mode change field is used to request system-wide mode changes, and a four bit acknowledgment field that contains acknowledgment information about the receipt of the messages from the predecessor and the pre-predecessor.

Bus Access Scheme – Access to the transmission medium is controlled by a static TDMA scheme. Each node is allowed to send messages only during a predetermined time span, called its *TDMA slot*. The nodes of an FTU send in subsequent TDMA slots, their *FTU slot*. In the implementation described in this paper an FTU always consists of two nodes, so two TDMA slots form an FTU slot. The sequence of the periodic TDMA slots is called a *TDMA cycle*. With regard to the duration of the TDMA slots and to the sending sequence of the nodes, all TDMA cycles are equal. However, the length

and contents of the messages (the application data) may differ. The set of periodically recurring TDMA cycles with possibly different message length and contents is called a *cluster cycle*.

The Message Descriptor List – The attributes of the messages sent and received by the protocol are described in a static configuration data structure, the *Message Descriptor List (MEDL)* that resides in the ROM within the communication subsystem. According to this list the TTP controller periodically and autonomously reads the messages to be transmitted from the MBI and writes received messages to the MBI. The most important information contained in the MEDL is therefore the address of each message in the message base interface MBI and the length of the message.

The Controller State – In TTP all nodes are forced to implicitly agree on their *controller states (C-states)*. The controller state consists of three fields: the MEDL position, the time, and the membership. The MEDL position field is a pointer to the current entry in the MEDL, i.e., it identifies the current mode and TDMA slot. The time field contains the global time at the beginning of the current FTU slot. The membership field indicates which FTUs have been active and which FTUs have been inactive at their last membership point. To enforce C-state agreement between a sender and a receiver the CRC of a normal message is calculated over the message contents concatenated with the local C-state. A receiver can only interpret the frame if sender and receiver agree about the controller state at the time of sending and receiving. In case the C-state of the sender differs from the C-state of the receiver, the message will be discarded by the receiver due to the different CRC.

THE TTP IP BOARD

Physical Appearance of the Hardware – The prototype of the TTP/C hardware module [14] is implemented as an "IP-Module" according to the Industry Pack Logic Interface Specification from GreenSpring Computers [4]. This facilitates its use together with any off-the-shelf computer board as host. Version 1 of the prototype, which is described in this paper, is a double-sized IP-module (9.2 cm × 10 cm). A single-sized version is currently under development. To facilitate hardware debugging most components on the board are mounted on the side of the PCB that is accessible during operation. This does not comply with the IP specification. Therefore, when using this prototype module together with VME boards in a standard rack, the slot next to the carrier board must be left empty.

Figure 2 shows an outline of the IP-module. The main functional unit is a Motorola 68332 *microcontroller*. Additionally, the following peripheral devices are provided:

- a *Flash EPROM* for the TTP/C protocol code and configuration data,

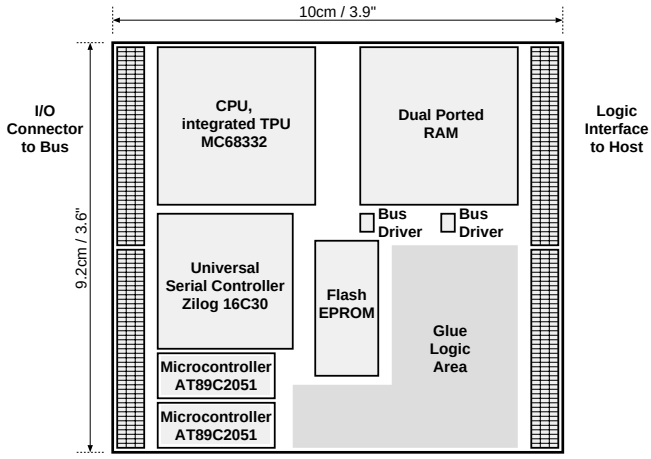

Figure 2: Outline of the TTP IP Board

- a *Dual Ported RAM*, partly serving as
  (a) interface to the host CPU (*MBI*) and
  (b) local RAM section for the MC68332
- a dual channel *Serial Communication Controller* for the physical layer management of the two TTP channels,
- two *Bus Transceivers* for signal conditioning
- two autonomous microcontrollers used as bus access monitors (*Bus Guardians*),
- glue logic, mainly implemented in programmable logic devices

The IP-module has two external 50-pin interfaces, one *Logic Connector* to the host CPU and one *I/O Connector* to the TTP bus and external debug devices.

Main Functional Units – The above-mentioned main functional units of the IP-module are described in more detail in the following.

Motorola MC68332 Microcontroller – The Motorola MC68332 [11] is a modular microcontroller unit (MCU) consisting of the following components:

CPU32: The CPU32 is based on a 32 bit architecture and has many features of the MC68010 and MC68020. A *background debug mode* (BDM) is provided that allows read and write access to registers and memory over a dedicated BDM interface. The CPU32 executes the main protocol tasks of TTP/C.

TPU: The time processing unit (TPU) is an autonomous microcontroller with sixteen independent, programmable timer channels. In addition to some ROM-based pre-programmed primitives customized time functions can be micro-programmed and executed in the *TPU emulation mode*. Since

the micro-programs directly access the TPU hardware, TPU operation is very fast and time resolution is very good. Consequently, the TPU was used for the implementation of all timing-related functions required for TTP/C.

Internal RAM: In the TTP implementation the internal static RAM (2 KBytes) is used as microcode RAM for the TPU functions and therefore not available to the CPU32.

Queued Serial Module: This module contains two serial interfaces, the queued serial peripheral interface (QSPI) and the serial communication interface (SCI). The QSPI is a full-duplex, synchronous serial interface intended for communication with peripherals and other MCUs, while the SCI is a full-duplex universal asynchronous receiver/transmitter (UART). Both interfaces are provided at the I/O connector. The TTP/C software uses the SCI for debug outputs.

System Integration Module: This module integrates the following glue logic functions on the MC68332:

- An *external bus interface* provides functional support for the access to peripheral devices. Data bus width is 16 bit.
- A *system configuration and protection block* controls configuration parameters, preserves reset status and provides bus and software watchdog monitors.
- A *system clock unit* generates a programmable system clock of up to 20 MHz from a 32.768 kHz crystal (in the current implementation a 16 MHz version of the MC68332 is used).

The internal modules of the MCU communicate via an *Inter Module Bus* with 24 address and 16 data lines.

Universal Serial Controller – The Universal Serial Controller (USC), a Zilog Z16C30 [16], has two independent full-duplex channels with data rates ranging from 0 to 10 Mbit/sec (currently, 1 Mbit/sec is used). Each receiver and transmitter is buffered with a 32 byte data FIFO, which is large enough to hold a complete TTP message. The USC supports several coding techniques. In the current implementation *FM0* is used, because this code allows synchronization of the receiver without bit stuffing (each bit cell starts with a transition, a logical 0 is coded with an additional transition in the middle of the bit cell). Data transfer is controlled by the CPU via several USC registers, status pins and control pins.

Bus Transceivers – Each of the two USC-channels is connected to the TTP bus over one PCA82C250 [13] bus transceiver that performs physical signal conditioning. Electromagnetic radiation behaviour associated with the bus signals can be improved by wave-shaping.

Microcontroller AT89C2051 – The AT89C2051 is an 8-bit, MCS-51 compatible microcontroller with a 2 KByte flash memory, two 16-bit timer/counters and a 128×8-bit internal RAM [3]. It is capable of handling two external interrupts. The high clock rate of 24MHz facilitates fast operation. In the prototype implementation the AT89C2051 is used as a Bus Guardian. To ensure independent functionality of the two redundant TTP channels, one autonomous bus guardian per channel is provided, each with a separate crystal.

Flash EPROM – The AB28F400 Flash EPROM [5] has a size of $256K \times 16$ bit. Its array blocking architecture allows blocks of predefined size to be programmed separately. The protocol code located in the boot block can remain write protected during maintenance operations for other blocks (e.g., MEDL update). A jumper facilitates write protection for the complete Flash EPROM. The Flash EPROM can be programmed on-board via the BDM interface.

Dual Ported RAM – The IDT7024 DPRAM used on the TTP IP board has a size of $4K \times 16$ bits, which may be upgraded to $8K \times 16$ bits by a pin-compatible chip. It comprises the following logical memory areas:

Message Base Interface (MBI) The MBI is the Communication Network Interface of the TTP protocol. In order to achieve the required operational decoupling of host and communication controller, both CPUs may access the MBI simultaneously. The DPRAM is capable of resolving access conflicts to the same memory cell by delaying the second access until the first is finished. The maximum delay is limited to 2 cycles of the IP clock.

Local RAM of CPU32 Since the internal memory of the MC68332 is used by the TPU, the CPU uses part of the DPRAM as local RAM. Host CPU access to this memory area can be inhibited through a jumper.

Debug Information Space Upon a failure, the communication controller logs debug information in this memory area. A diagnostic task running on the host computer may read this data for off-line debugging.

Hardware Architecture – Figure 3 shows a block diagram of the TTP module.

It is important to note that the connection between communication controller (MC68332) and host (IP-interface, respectively) is restricted to the DPRAM and a single interrupt line for a programmable time interrupt from the TTP controller to the host. This strict decoupling is a key requirement of the TTP concept on the module hardware. There is no flow of control information across this "MBI-firewall". The host subsystem is provided with all information by this well behaved memory interface, while the "real world" activities of

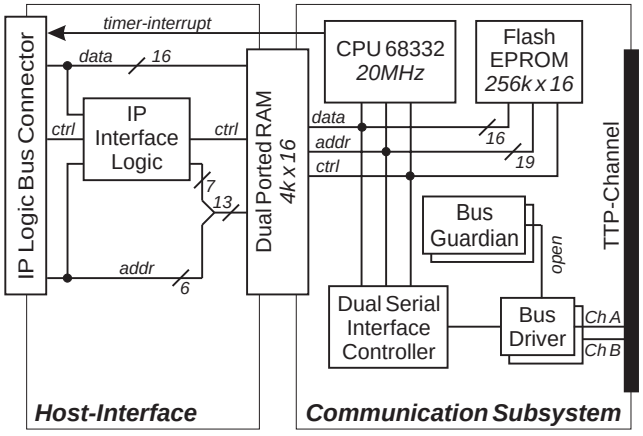

Figure 3: Architecture of the TTP IP Module

the TTP channel are handled by the controller subsystem, invisible to the host.

Another important requirement of the TTP concept on the hardware is high precision of time stamping. Since clock synchronization in TTP [7] relies on time stamping of messages, uncertainties introduced by buffers and delays limit the achievable time granularity. For this reason all critical functions relevant for time stamping have been implemented in the TPU and the respective TPU pins have been directly connected to corresponding USC control and status signals. The TPU can directly trigger the start of a data transmission on a channel via the corresponding *clear to send* ($\overline{CTS}$) input. Similarly, a *receive character* signal activated by the USC for each channel upon the start of a message reception is used by the TPU for time stamping incoming messages.

The bus guardians play a key role in ensuring the required fail silent behaviour of the TTP module. Due to the deterministic bus access scheme of TTP, a measure against transmission failures in the time domain can be introduced: the bus guardian checks the bus access behaviour of the communication controller by observing the *clear to send* signal. Only if $\overline{CTS}$ activations follow the regular pattern of the TDMA scheme, the bus guardian permits access to the TTP channel via an *open* signal, otherwise it blocks the transmission path. The TTP controller uses a *ready* signal to inform the bus guardians whether the module is actively sending on the bus. Care was taken during the design that the bus guardian operation is self-supporting. There is no data or control path to the bus guardian except for the two above-mentioned status bits.

To support fast re-integration a flag is provided by the glue logic that indicates whether a cold start or a warm start is being performed.

CONNECTORS – IP-modules are mechanically and electrically connected to the carrier board by two 50-pin connectors. These are dedicated as follows:

IP Logic Connector The IP specification gives a complete description of this interface between the host CPU and the carrier board. While the TTP module meets all requirements for the basic 8 MHz-operation with 16 bit data width, it does not support options like DMA, 32 MHz-operation, or 32 bit mode.

The following address spaces are distinguished:

- *memory space:* allows host read/write access to the DPRAM.
- *I/O space:* provides read/write access to a semaphore logic (currently not supported by the TTP software).
- *ID space:* 32 byte of identification information can be read by the host in this space.
- *interrupt vector:* Although TTP does not require interrupt vectoring, a fixed interrupt vector is provided in this space to ensure compliance with the IP specification.

Access to all address spaces is possible without wait states, i.e., with a data rate of 8MB/sec.

I/O Connector The use of the I/O connector is specific to the functionality of the IP-board. On the TTP module it comprises the following signal groups [15]:

- the TTP/C signals of both communication channels
- all signals required for BDM operation, including the reset line for the module logic
- all TPU signal lines
- both serial interfaces of the MC68332 (QSPI and SCI)
- one parallel port of the MC68332
- status lines of the bus guardians

While the signal group described in the first item represents the important connection of the module to the TTP bus, all other signal groups are supplied for maintenance, debugging and evaluation procedures.

THE TTP/C SOFTWARE ARCHITECTURE

OVERVIEW – The TTP/C controller prototype consists of several modules. Figure 4 gives an overview over the protocol software modules and their interaction.

The central module of the TTP/C prototype software is the module called *TTP/C core functions*. This module is executed on the CPU of the MC68332 and contains all protocol tasks. It accesses the MEDL and MBI data structures and interacts with the USC, TPU, and Clock Sync software modules.

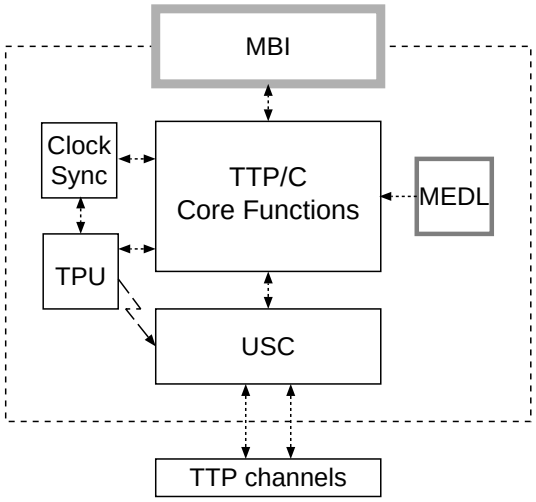

Figure 4: Software Architecture Overview

The *USC* module contains interface functions to the Zilog Universal Serial Communication Controller (USC). It handles autonomous message transmission and reception at the physical layer.

All timing related functions are handled by the *TPU* software module. The TPU module consists of two parts:

- Custom-made microcode is executed by the TPU of the MC68332. It deals with all low-level timing functions, i.e., the generation of a globally synchronized time base, triggering the message transmission at certain points of time, and time-stamping the received frames.
- CPU functions act as an interface between the micro-code and the TTP/C core functions.

In order to establish a global time base, the internal clock must be synchronized with the other active nodes. Therefore, a high level clock synchronization algorithm — contained in the *Clock Sync* software module — is required. This module continuously calculates a correction factor for the clock tick generated by the TPU.

DATA STRUCTURES – Data and control flow information for TTP/C is contained in two major data structures. The MBI — placed in the DPRAM — is the memory interface to the host CPU. The MEDL data structure contains the static message schedule and is stored in the flash EPROM.

Message Base Interface – As already mentioned in the previous section the MBI [10] is the communication network interface of TTP. It is structured as follows:

The *Status/Control Area* (SCA) contains system related information. The memory layout of the SCA registers is shown in figure 5. The arrows indicate the direction of the data transfer of the corresponding field.

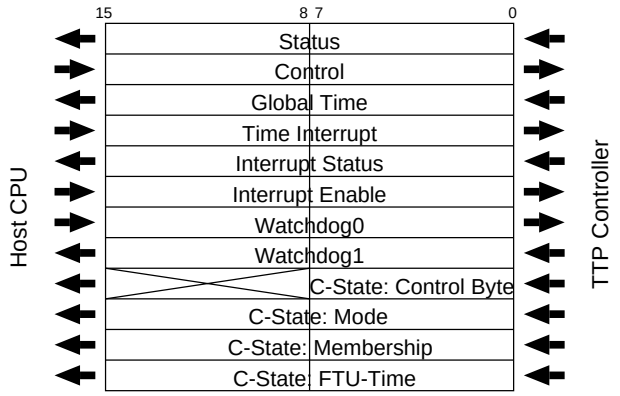

Figure 5: MBI Status/Control Area

Status: The TTP controller writes information about the current status of the protocol execution into this register, for instance, whether the last received message was valid, and/or was an I-frame, or whether a mode change was requested.

Control: A host may request the communication protocol to change the current operational mode of the system (a *mode change*) by writing the number of the requested mode into this register.

Global Time: Via this register the TTP controller provides the cluster-wide synchronized time base to the host CPU.

Time Interrupt: The host subsystem may request an interrupt at certain points in time depending on the value in this register. Once the synchronized global time reaches this value an interrupt is raised to the host subsystem.

Interrupt Status: This register contains information about the source of an interrupt raised to the host CPU by the TTP controller.

Interrupt Enable: Using this register, a host CPU can enable/disable several types of interrupts. Interrupts can be raised upon message reception, mode changes, or membership changes.

Watchdog: Through these registers the host subsystem and the communication subsystem perform mutual activity checks using a *challenge–response protocol*.

C-State: These registers contain the current C-state, consisting of the current MEDL position (indicating current mode and TDMA slot), the current membership, and the start time of the current FTU slot.

The *Message Area* contains the TTP message data sent or received by the node. Access to the message area is bi-directional; both the host subsystem and the communication subsystem read and write messages from/to

this area. For each message a control byte indicates the reception status of the message.

The message control byte is also used for resolving concurrency conflicts between the host and the TTP controller. Since the DPRAM supports atomic 16-bit memory access operations all data fields longer than 16 bits, like the C-state, must be protected by an additional synchronization protocol in order to avoid inconsistent data. Therefore, the Non-Blocking Write (NBW) protocol [9] is used, which requires a dedicated control field in the control byte.

In the NBW protocol the writer is never blocked, while the reader may be delayed until the writer has finished. It can be shown that an upper bound for the delay exists if the time between write operations is significantly longer than the duration of a write or read operation.

In TTP, the NBW protocol is only used in one direction of the data transfer, namely for data transfer from the communication controller to the host CPU. This ensures that the communication controller never is blocked. The integrity of the data transferred from the host CPU to the communication controller must be asserted by synchronizing any write access of the host CPU with the protocol TDMA sequence (e.g., by using the programmable time interrupt).

Message Descriptor List – The MEDL contains the message schedule for the TTP controller. It identifies the sequence of senders and transmitted data within the TDMA sequence. A detailed specification of the MEDL is given in [12].

Figure 6 gives an example of a MEDL structure used in this prototype implementation. The operational ensemble consists of 6 SRUs forming 3 FTUs. Within an FTU slot both SRUs of an FTU send frames in the two TDMA slots of the FTU.

In this example there are two different operational modes: the *join mode* where the cluster cycle consists of a single TDMA cycle, and one *application mode* where the cluster cycle consists of two TDMA cycles. A mode change from the join mode to the application mode is only possible at the end of the TDMA cycle.

In the join mode only I-frames are transmitted. In the application mode N-frames (containing application data) are transmitted, but FTU 0 and FTU 1 are also sending I-frames. This allows failed nodes to synchronize and re-integrate into the operational ensemble.

For each SRU slot there is a MEDL entry containing the following information:

Mode Identifier: This field indicates which mode is currently being executed.

Sending FTU: This field contains the number of the currently sending FTU, i.e., the current FTU slot.

Slot Count: Since an FTU slot consists of two TDMA slots, this field is needed to distinguish between the first and second TDMA slot of the current FTU slot.

I-Frame: This field determines whether an I-frame or an N-frame is sent in the current slot.

Interrupt Condition: This bit field determines which protocol conditions may cause a time interrupt. In the current implementation, a change of the current membership, the reception of a particular message, and a mode change can cause an interrupt at the beginning of the next FTU slot.

Message Length: The length of the message that is sent in the current slot is stored in this field.

Channel Assignment (CAS): This bit field indicates which node will be sending on which channel in the two SRU slots of an FTU slot.

Message Base Address: This field specifies the address of the message in the MBI.

If a mode change is allowed in a slot, the corresponding MEDL entry additionally contains a list of *Successor Mode Pointers* (SMOPs) pointing to the first FTU slot of the new mode. Whenever a mode change is requested, the SMOP corresponding to the requested operational mode is used to perform the mode change.

AUXILIARY MODULES – The TTP/C implementation includes several auxiliary modules that provide basic services required by the core protocol functions, such as clock synchronization and interfacing to the IP board hardware devices.

Clock Synchronization – This module covers the high level part of the clock synchronization routines. Its input are the time differences captured by the TPU that are used to calculate a correction factor for the local clock to synchronize with the rest of the ensemble. In our implementation, a fault tolerant clock synchronization algorithm similar to [8] is used.

USC – The USC module contains the low level interface to the Zilog USC hardware. It provides the functionality for autonomously transmitting and receiving frames on both TTP-channels.

Right before its sending slot, the TTP/C controller assembles the message to be transmitted. The USC module then calculates the CRC over the data and places the frame into the USC FIFO. Transmission is started by the TPU through the Clear To Send ($\overline{CTS}$) line when the sending slot time is reached, allowing data transmissions to be triggered with a minimal jitter.

At the beginning of a receive slot, the TTP/C controller activates the USC module and switches the communication hardware to "receive mode". A received frame is then copied from the USC-FIFO to the receive buffer of the TTP/C state machine, where a CRC check is performed.

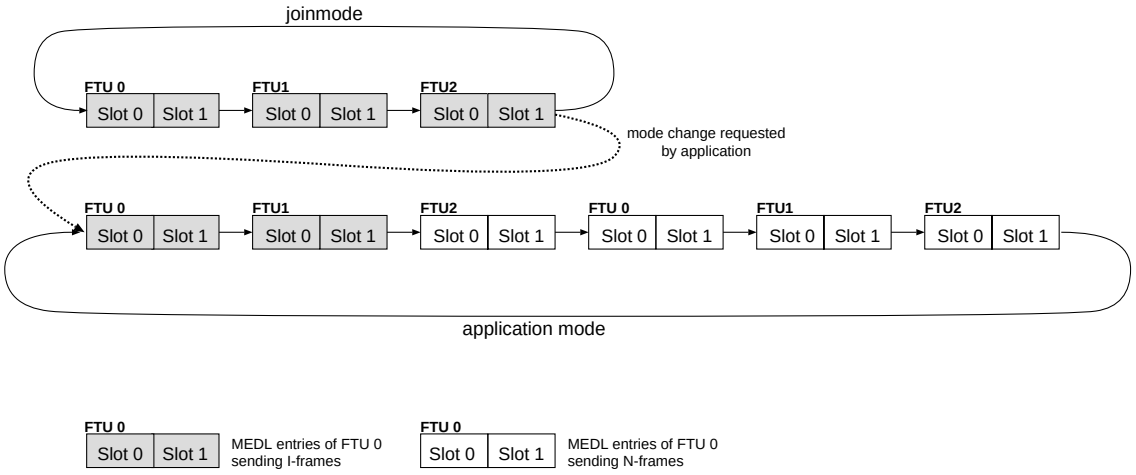

Figure 6: MEDL Structure

Bus Guardian – A node that violates its transmission schedule can disrupt any concurrent data transmission on the bus. As outlined in the previous section an independent device, the bus guardian, is included on each node to control the periodicity of the transmission activity of the respective node.

This is done by enabling and disabling the bus driver of the node according to the TDMA scheme. In order to fulfill this task the bus guardian must contain knowledge of two bus access parameters: (1) the duration of the TDMA slot and (2) the duration of the TDMA cycle. In the current implementation these parameters are identified during the first three transmission cycles of the node. Alternatively, it is also possible to store these two parameters statically within the code segment of the bus guardian.

After the communication controller has been reset the bus guardian prohibits any transmission of its own node on the communication channel. Once the node is ready to transmit it notifies the bus guardian by toggling the *ready* signal. In the next phase the bus guardian identifies the bus access parameters.

Once the bus guardian has acquired these parameters the following steps are performed periodically: At the beginning of the respective TDMA slot the bus guardian enables the bus driver. During the transmission phase the deviation of the bus guardian time base to the synchronized time base of the cluster is recorded. At the end of the TDMA slot the bus driver is disabled. The measured deviation is used to maintain a synchronized time base within the bus guardian. At the end of a TDMA cycle the process repeats with the activation of the bus driver.

TPU FUNCTIONS – Since the protocol execution in TTP is controlled by the progression of time, the implementation of the timing-related functions is one of the crucial parts of the prototype implementation. Due to its low jitter the TPU is very well suited to perform all required time functions.

Through its macrotick function the TPU provides a global synchronized time base with a *macrotick* granularity of $15.3\mu s$. Clock correction is done by adjusting the number of *microticks* (internal oscillator ticks) per macrotick according to the correction factor calculated by the clock sync module.

Another important TPU function is time difference capturing. The *Time Difference Capture* function of the TPU measures the time deviation between the expected and the observed arrival time of a received frame. This is closely linked to the start transmission function for sending: The TPU microcode provides the functionality to start a transmission at a certain point in time without jitter by directly triggering the USC via the $\overline{CTS}$ line.

TTP/C CORE FUNCTIONS – This subsection gives an overview of the main protocol functions of TTP/C. A dispatcher handles the sequential execution of the two major protocol functions, the *inter message gap* function and the *transmission phase* function. These functions are invoked at two important points in time, the beginning and the end of a message transmission. These two points in time are called *action times* and are shown in figure 7.

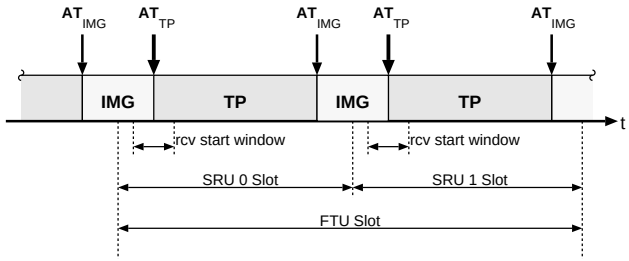

Figure 7: General Protocol Timing

In this figure the start of transmission is labeled AT_{TP} (action time for transmission phase), the end of transmission is denoted as AT_{IMG} (action time for inter message gap). The basic operations performed by the functions are shown in figure 8.

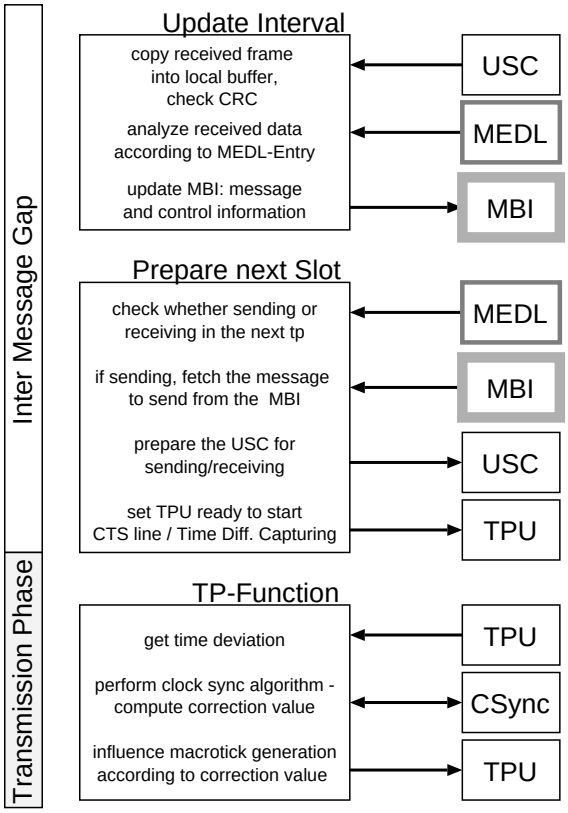

Figure 8: Overview of the protocol execution

Inter Message Gap (IMG) – The IMG comprises the *Update Interval*, where the received frames are analyzed and copied to the message base interface (MBI), and the *Prepare next Slot* interval, in which the controller prepares data transmission and reception for the next TDMA slot.

In the *Update Interval* the controller first checks the validity of the received messages. A frame is only accepted when its CRC is valid and the C-state of the sender is equal to the C-state of the receiver. For N-frames this is ensured implicitly through the CRC, which covers the C-state of the sender in addition to the message data.

At the end of the current FTU slot, i.e., the point in time when all four redundant frames should have been received, the controller analyzes the accepted messages and updates its C-state. This includes an update of the membership field, the current MEDL position (which is advanced to the next TDMA slot position, or — if requested — to the start position of another mode), and an update of the C-state time to the start time of the transmission phase of the next slot.

After the MEDL position has been advanced, the node checks whether its membership point has been reached, i.e., whether it is the next to send. In this case the controller checks if its C-state has been in agreement with the majority of nodes during the last TDMA cycle. If the check is negative, the node assumes a disagreement with the majority of nodes and performs a reset. Otherwise it assembles the frame to send according to the current MEDL entry. The frame is then copied to the transmit FIFO of the USC.

Finally, the controller determines the next action time (AT_{TP}) and activates the USC for receiving and/or sending at that time on the corresponding channels.

Transmission Phase (TP) – In this phase two frames are transmitted and received on the bus, one frame on each channel. Since a complete TTP message can be stored in the USC-FIFOs, the USC autonomously performs the send and receive operations without any interaction with the TTP controller. The actual start of transmission is triggered by the TPU via the $\overline{CTS}$ line.

Once the results of the *Time Difference Capture* TPU function are available, the CPU calculates a clock correction term using the high level clock synchronization function described above. This clock correction term is then fed back to the macrotick function of the TPU, which adjusts the local clock accordingly.

CONCLUSION

In this paper a prototype implementation of the time triggered protocol TTP/C has been presented. This implementation has clearly shown the feasibility and usefulness of the concepts associated with TTP.

The prototype was intended to get first practical experience with TTP and to obtain results for fine tuning the concept. Neither performance nor size was a major concern. However field applications will require improvement of these two topics.

We are currently working on a version 2 hardware. One goal for this version is to improve performance by delegating the task of CRC checking to dedicated hardware, since CRC calculation has proven to constitute a considerable performance bottleneck in the current implementation. Version 2 will also provide a higher error detection coverage through additional self-checking mechanisms. Fault-Injection experiments will be performed to validate the fault tolerance mechanisms of TTP/C and to evaluate error detection coverage rates.

Our final vision of hardware implementation is a VLSI chip appearing as a peripheral device, providing full TTP functionality. By applying the MBI specification for the prototype as well as for the VLSI chip, the interface to the host will remain the same, regardless of the actual implementation. This allows decoupling of the application software development and plug-in replacement of a discrete component module by a VLSI module. The low production cost for a single chip in a high

volume production will make a TTP chip attractive not only for the automotive industry, but for other application domains as well.

REFERENCES

[1] Class C Application Requirement Considerations. SAE Recommended Practice J2056/1, SAE, June 1993.

[2] Survey of Known Protocols. SAE Information Report J2056/2, SAE, April 1993.

[3] Atmel Corporation, 2125 O'Nel Drive, San Jose, CA 95131. *Atmel Microcontroller Data Book*, Oct 1995.

[4] GreenSpring Computers, Inc., 1204 O'Brien Drive, Menlo Park, CA 94025. *IndustryPack Logic Interface Specification*, 1995. Revision 0.7.1.

[5] Intel. *Intel A28F400BR-T/B Data Sheet*, 1995.

[6] H. Kopetz and G. Grünsteidl. TTP — A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[7] H. Kopetz, R. Hexel, A. Krüger, D. Millinger, and A. Schedl. A Synchronization Strategy for a TTP/C Controller. In *Application of Multiplexing Technology (SP-1137)*, pages 19–27, Detroit, MI, USA, Feb. 1996. Society of Automotive Engineers, SAE Press. SAE Paper No. 960120.

[8] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 36(8):933–940, Aug. 1987.

[9] H. Kopetz and J. Reisinger. The Non-Blocking Write Protocol NBW: A Solution to a Real-Time Synchronisation Problem. In *Proc. 14th Real-Time Systems Symposium*, Raleigh-Durham, North Carolina, USA, Dec. 1993.

[10] A. Krüger and H. Kopetz. A Network Controller Interface for a Time-Triggered Protocol. In *SAE Symposium on Future Transportation Electronics: Multiplexing and In-Vehicle Networking*. Society of Automotive Engineers, August 1995. SAE Paper No. 952576.

[11] Motorola, Inc. *MC68332 User's Manual*, 1990. MC68332UM/AD.

[12] R. Nossal. MEDL/MODL Specification. Technical Report 25/95, Institut für Technische Informatik, Technische Universität Wien, 1996.

[13] Philips Electronics North America Corporation. *80C51-Based 8-Bit Microcontrollers Data Handbook*, 1994. page 1243.

[14] A. Steininger. The 68332 IP Hardware Module. Research Report 9/96, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, Jan. 1996. Version 1.0.

[15] A. Steininger. The Hardware Interfaces of the IP Module. Research Report 3/96, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, Jan. 1996. Version 1.0.

[16] Zilog, Inc., 210 East Hacienda Ave., Campbell CA 95008-6600. *Zilog 16C30 USC Universal Serial Controller User's Manual*, 1995.