A Network Agent for Diagnosis and Analysis of Real-time Ethernet Networks

Hans-Peter Löb
Infineon Technologies AG
Communications, Access System Engineering
81726 Munich, Germany
hanspeter.loeb@infineon.com

Rainer Buchty, Wolfgang Karl
Universität Karlsruhe (TH)
Institut für Technische Informatik
76128 Karlsruhe, Germany
[buchty|karl]@ira.uka.de

ABSTRACT

Within the field of automation technology the use of Industrial Ethernet is rising. This in turn demands devices capable of precisely recording, analyzing, and manipulating communication data for diagnostic purposes. Existing solutions so far lack required flexibility or are unable to cope with sustained Gigabit-per-second data streams. This is especially true for general-purpose approaches employing ordinary network adapters and plain software-based analysis.

In this paper we describe a flexible and lightweight network agent for real-time, high-performance networks. This agent is capable of handling sustained data rates up to 2x 1GBit/s while offering real-time event-triggers, 10ns-resolution timestamps, real-time filtering, and statistics functions. An auxiliary processing unit as well as a modular software environment allow customization for a variety of tasks. The agent is realized as a dual processor SoC design on a Xilinx Virtex-II Pro FPGA.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Diagnosis, Analysis

General Terms

Design, Measurement

Keywords

System-on-Chip, Real-time, Industrial Ethernet, Monitoring

1. INTRODUCTION AND MOTIVATION

With the rising use of Industrial Ethernet within the field of automation technology, increased demand exists for capable monitoring and analysis devices. Such systems must be able to precisely record, analyze, and manipulate communication data for diagnostic purposes. This, in turn, requires a high amount of flexibility, and – in particular – the ability to cope with sustained Gigabit data streams.

Application scenarios range from simple general purpose network analysis to highly protocol-specific tasks within the real-time-capable Industrial Ethernet protocol PROFInet [21] as it is deployed in industrial automation contexts. This includes the control and verification of correct system-wide behavior within tight timing constraints according to protocol specifications. In addition to passive monitoring the device also has to actively interact with its environment: in case of a detected malfunction appropriate reporting e.g. via e-mail is necessary in addition to a precisely-timed recording of network traffic for later proof and analysis. Further, active test modes for device-under-test latency and throughput tests are needed.

To give a more detailed application, a large system such as e.g. a manufacturing site can be considered: during development or in a debug setting the network agent is attached to the system and starts to analyze connected subsystems and identify their addresses and functions on-the-fly. If cyclic communication occurs – as it is the case in a real-time setting – the reliability of single subscribers is monitored with respect to jitter or complete absence of messages. As the agent is rather small in size and portable, several devices can be placed at central or de-central places within the system to gain an exhaustive overview of the overall online status from a central workstation.

To meet all of the above demands, the device must satisfy a number of properties which are central to embedded networking analysis systems specialized on Industrial Ethernet:

- The device must be able to cope with two individual 1 GBit/s data streams and also support 100 MBit/s mode. Overall system memory is very limited compared to incoming traffic and needs to be considered in every aspect of system design.
- Because it is used in automation control and therefore sensitive environments, it furthermore must be very reliable.
- Reception functionality must support two synchronized channels with a timestamp resolution of 10ns, and provide exhaustive filtering and online data aggregation functions. Channel merging allows passive monitoring of two-way communication using a network tap.
- Transmission functionality must include precise scheduling and ways to embed additional fields like counters or timestamps into packets to be sent.
- Adaptive capabilities are needed; the device must especially be able to automatically detect network addresses, the underlying network standard and infrastructure, and used protocols.

- Flexibility is crucial in the context of the intended commercial use. The device must support an easy-to-use update functionality and be adaptable to customers' specialized demands.

- To allow for a broad scope of application scenarios versatile user interfaces are mandatory. These must range from local or physical access to remote availability of both device control and current status in a distributed environment.

General purpose approaches are not able to fulfill these requirements, but also commercially available products and existing academic solutions fall short when it comes to the desired flexibility and specialized functionality in combination with sustained data rate. The intended use of the network agent will be within a professional environment and in small to medium number of pieces. Providing complete, specialized functionality and tailor-made solutions to end-users for an out-of-the-box use of the equipment ensures commercial success.

Having all that in mind, an FPGA-based System-on-Chip design is an appropriate and tractable solution. Allowing a small team of developers to create powerful and highly specialized designs, this approach and todays available advanced FPGAs negate the need for cost-intensive ASIC development in application scenarios like the one presented here.

The work described herein has been developed in cooperation with Siemens AG, Department I&S. Siemens provided the application scenario and an already developed hardware platform. The network agent presented within this paper was specifically developed with the given application scenario in mind and was tailored to the provided hardware platform.

This paper will focus on fundamental aspects of the overall design and is organized as follows: Sections 2 and 3 provide an overview on PROFInet and related work from the field of network and packet analysis. In Section 4 we will introduce the hardware platform followed by an extensive presentation of our network agent architecture. Results are presented in Section 6 whereas Section 7 will conclude this paper and also give an outlook on currently ongoing extensions to the presented network agent.

2. PROFINET PROTOCOL

PROFInet is an open and real-time-capable extension to the Ethernet protocol to implement Industrial Ethernet in automation scenarios [21]. A more comprehensive view on Industrial Ethernet and related protocols is given in [9].

While standard networking technology becomes more widespread and easily available, in automation technology there is a trend towards more complex and unified networks. Thus the PROFInet protocol as developed by Siemens to succeed the well-known PROFIBUS protocol [20] seeks to combine the advantages of Ethernet with the demands in industrial automation control and field-bus applications.

Although TCP/IP protocol functions are well suited for unified management and configuration tasks, the responsiveness of standard Ethernet is insufficient to support even soft real-time requirements [15]. So there are two distinct restriction imposed by the protocol to overcome these limitations:

- To achieve latencies of 10 ms and below, a switched network topology with high-performance network hardware and a priority system must be deployed.

- To achieve latencies of 1 ms, an additional temporal decoupling of the medium called *time slicing* is needed.

The first point is rather straightforward and will not be explained here. As far as the time slicing approach is concerned, this means that network traffic is divided into deterministic and open communication. Figure 1 shows this deterministic part with devices `A` through `E` performing time-critical communication. These devices get exclusively assigned a distinct offset for cyclic sending relative to a recurring basic timing signal or multiples of it. Thus collisions are impossible to occur, and immediate access to the medium is possible, enabling so-called isochronous real-time applications in the sense that a timing signal is common to all devices. Open communication can take place in a designated period after cyclic sending has finished (not shown in the figure).

The downside of this approach is the high amount of recurring traffic, as every device must respond within the assigned period of time even if no new data is available. This lends itself to specialized hardware support to offload the overall system and will be discussed in Section 5.4.

In addition to the above but out of the focus of this paper, PROFInet provides rich protocol functionality to enable company-wide automation and integrated communication.

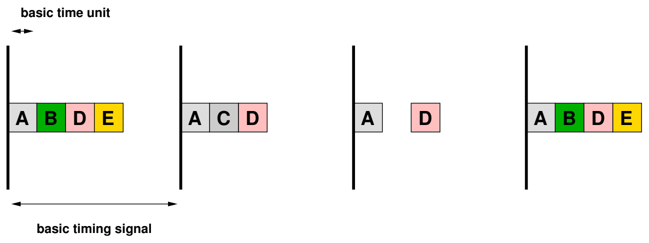

Figure 1: Time-slicing as used within the PROFInet protocol

3. RELATED WORK

Several software-packages exist for network and packet analysis. These range from ordinary protocol analyzers like *Ethereal* [8] to complex solutions like the *BSD Packet Filter (BPF)* [19], which includes a packet capture architecture for UNIX-like operating systems in addition to introducing an influential register-based filtering language. The WinPcap project [7] has ported these ideas to the Windows OS whereas the *Fairly Fast Packet Filter* [4] combines several filtering languages and can be used on multiple platforms.

Numerous commercial network analysis devices exist such as Spirent's *SmartBit*, WildPackets' *Gigabit Analyzer Cards*, or the Shomiti line of products. These solutions, however are rather costly, and do not provide the functionality and flexibility desired for the designated field of use. An overview (from a commercial point of view) over such architectures is presented in [24]. Only very recently commercial FPGA-based analyzer cards have been developed such as the products of Napatech [12] which address many of the issues presented in this paper.

In the scope of this work are FPGA-based System-on-Chip embedded networking solutions. Although FPGAs are being deployed in many application areas most related projects have a different focus, such as the programmable network interface card (NIC) developed by Jeff Shafer [22] or the Field Programmable Port Extender (FPX) project [16]. Techniques for hardware acceleration in real-time Systems-on-Chip can be found in [3], and in [5] a paradigm shift from processor-centered to logic-centered architectures is proposed.

Several techniques exist for implementing filtering algorithms in hardware. These are often employed in high-performance servers and network routers, such as algorithms for IP-table lookups [1], and fully-associative memories such as *Content Addressable Memories* [2]. An Ethernet Statistics IP Core is provided by Xilinx [25]. Also methods exist for classification of several chunks within a packet [11]. A comprehensive overview on packet classification can be found in [23].

Most existing solutions have in common that these are either mainly designed for processing a huge amount of filtering rules (>100,000) or too heavyweight (such as e.g. the well-known SCAMPI [6]), whereas the target scenario of our approach is more in the range of up to several hundred filtering rules.

The technique used within our architecture is basically comparable to the *Parallel Bit-Vector Method* [13], although developed independently and specifically tailored towards the present hardware resources. These resources were defined by our industry partner for which a dedicated architecture framework was developed [10].

The resulting filter engine is despite its small footprint well in the range of current commercial products like [12] as far as filtering functionality and performance is concerned.

4. HARDWARE PLATFORM

The hardware platform was developed by Siemens and provided for this work. Core of the platform is a Xilinx Virtex-II Pro FPGA (XC2VP50). Among reconfigurable logic resources, this chip contains two PPC405 cores, and advanced clock management facilities such as 8 PLL-like digital clock managers (DCMs) and 16 global clock distribution networks (BUFG), which are vital for this work.

The platform is designed for generic use in the fields of network analysis and simulation. It provides up to 2GB SDRAM, standard interfaces like USB and RS232, and offers 6 network ports, 4 of which are low-level-accessible via Broadcom PHYs for analysis purposes. It contains non-volatile memory for program and data storage.

Required functionality needs to be integrated into the FPGA by using IP blocks implementing a complex System-on-Chip. Although an external power supply is needed, the overall device is small, portable and does not need additional cooling.

5. ARCHITECTURE DETAILS

The network agent is realized as a System-on-Chip (SoC) and therefore fully integrated into the FPGA. Within the FPGA, the system divides into the reconfigurable part and the two PPC hard-cores embedded into the FPGA fabric. In this section we will describe, how these resources were used to implement the required functionality.

5.1 Architecture Overview

The traditional approach to building embedded network analyzers is typically a single-processor design employing one or more standard network interface cards (NIC). Incoming data needs to be copied by the CPU on an interrupt-based communication scheme, which is a major performance bottleneck. In addition to that, standard NICs lack flexibility and intelligence to alleviate the CPU and system buses from irrelevant traffic, and to perform simple tasks and computations on their own. Thus, even with DMA support, executing all the specialized tasks required for the network agent on a general purpose CPU is not feasible in a small and portable device. Finally, the responsiveness to distinct events and the exactness crucial for time-stamping in a real-time setting can not be achieved with standard components.

The system partitioning chosen for our approach, in term, provides real-time support, frees the system from unnecessary load and provides powerful computational resources divided on two CPUs. To give an overview over the system's architecture, the three major functional parts of the network agent are introduced:

- Real-time subsystem enabling autonomous network operations
- Auxiliary CPU employment for flexible and complex operations
- Communication, control, and configuration infrastructure

Central to the first part are the two Real-time Media Access Controllers (RTMACs), which autonomously transfer data from the medium to main memory or vice versa, providing additional real-time analysis and testing functionality. This will be subject of Sections 5.2 through 5.4.

To offload the overall system and to provide more flexible capabilities to the platform, one of the CPUs is used as a lightweight auxiliary functional unit. The scope of use is maximized by providing appropriate communication mechanisms and optimal system integration, as described in Section 5.5.

The System Bus is the major communication means within the system. The main CPU controls and configures all system aspects and is responsible for user and other remote interaction. A brief overview over the Software environment is given in Section 5.6.

This section concludes with a description of important implementational details in 5.7.

5.2 Real-time Media Access Controller Module

Main part of the system are two Real-time Media Access Controllers (RTMACs) within the reconfigurable part which are responsible for controlling network components. They interface with external network components and contain all necessary logic to receive and transmit network data. The concepts providing this basic functionality were developed by Gorden Griem in a previous work [10].

The RTMACs therefore divide into four main parts which are *bus interface*, *buffering*, *data receive* and *transmit* modules. The bus interface provides necessary interfacing to configure the RTMAC's sub-units such as the filtering and triggering within the receive module, and also enables access to internal registers. Furthermore, a DMA engine ensures independent access to external memory to fetch or store data packets received or to be sent over the network interface.

The RTMAC module further contains all necessary logic for time-stamping, filtering, address detection, cycle control, packet slicing, and statistics aggregation which will be described in the following subsections. In order to achieve reliable processing of all data *at line rate*, all these components are closely coupled with the MAC layer within the reconfigurable part. Figure 2 gives an overview over the RTMAC.

5.3 Data Transmission

Data transmission will take place according to an explicit timing schedule as described below. According to this schedule, the DMA engine will fetch data packets from main memory and output them using the designated physical interface. As an additional decoupling, fetched data is first placed into local buffers prior to transmission to account for delays on the system bus.

The scheduler shown in Figure 2 is responsible to initiate DMA data transfer and therefore keeping the local TX buffers filled. It

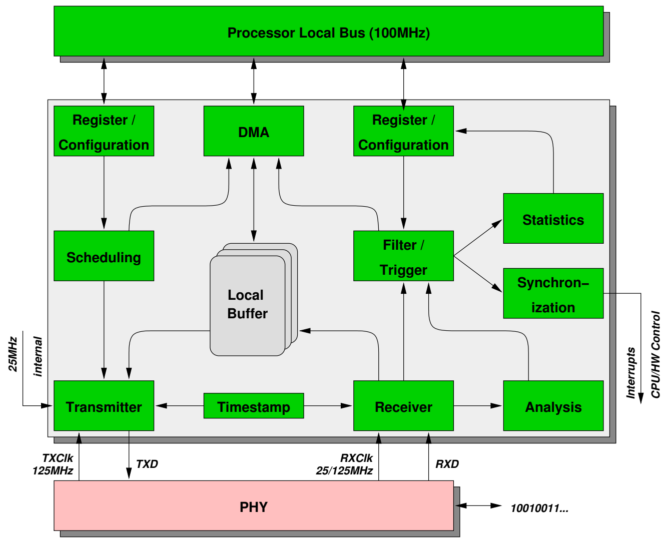

Figure 2: RTMAC Overview

Method of Operation	**cyclic transmission of up to 256 virtual devices**	**CPU-controlled queue w/ delay**
Pro	autonomous, good for device simulation	flexible, exact timing, low demand for logic resources
Contra	high use of logic resources, not timing-exact, not flexible	CPU usage, not appropriate for cyclic transmission

Table 1: Types of Transmission Scheduling

therefore must regard memory access latencies as well as the local buffer fill ratio. When a data packet is ready to be sent, the transmission logic is informed about the packet's address within a designated local buffer.

The transmission logic will then fetch the packet, apply the 8-byte preamble required for the Ethernet frame format, and directs the packet to the addressed physical interface. Depending on protocol and interface, data will be sent out in the desired format (e.g. byte-wise) and using the appropriate clock rate. Finally, the CRC checksum will be computed and transmitted as well.

Additional data can be dynamically applied such as an associated MAC address for simulation or device based operation as well as packet counters and a timestamp for pro-active testing applications.

Within our system, transmission scheduling can be done using two different approaches as summarized in Table 1. The device-based *cyclic scheduling* on a round-robin basis can take place autonomously in hardware but has difficulties achieving very tight timing requirements. On the other hand, the combined use of *timed transmission queues* in both main memory and hardware allows for exact timing while using only scarce logic resources.

In this latter scenario, the scheduler fetches a pointer to the next packet together with a delay annotation from main memory whenever buffer space is available. It then transfers the referenced packet to local buffer memory. The actual sending is scheduled in hardware using a FIFO and according to the transferred per-packet delay information which specifies the offset to the previous packet in cycles. This can be realized using a simple hardware counter. After sending a packet, the sender signals the CPU to refill the main memory queue using either interrupts or a hardware-based communication scheme like the one described in Section 5.5. A drawback of this approach is that the queue in main memory has to be maintained and filled, thus relying on either the main CPU or providing an application scenario for the auxiliary CPU (as also outlined in Section 5.5).

5.4 Data Reception

Within the data reception process, incoming data is buffered and post-processed, requiring real-time filtering and analysis capabilities.

Depending on the used physical interface, the start of an incoming data packet needs to be detected, to then byte-wise translate this data into an internal representation as shown in Figure 3 which is stored in local buffers. Upon detecting the packet's end, the internal representation is finalized. In parallel to data reception, information about the packet (type, length, protocol, etc.) are generated and possible errors are detected, which are forwarded to additional modules for post-processing.

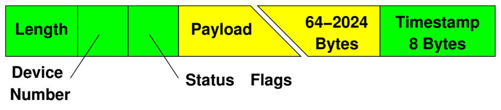

Figure 3: Internal Packet Representation

An exact timestamp is associated with each packet upon arrival and stored within the internal representation, thus being available for immediate and downstream analysis. Specifically, synchronized time-stamping on both channels allows for merging data into one virtual channel as it is needed when two-way network traffic is aggregated using a passive network tap.

To enable fast data transfer, a DMA engine ensures independent writing of buffers into the main memory. It is important to note, however, that local buffering is only used to account for delays on the system bus and to allow operating on complete packets: all parts of the reception process must be capable of dealing with worst-case incoming traffic without creating any backlog, thus further increasing reliability.

Apart from the basic conversion, filter rules can be applied to the incoming data implementing a packet filter, and also recording of a consecutive data stream and statistical values can be triggered with the same facility. By introducing intelligence at the lowest possible level, total amount of data to be processed is reduced thus relieving other system components such as system bus and CPUs. Precisely triggered recordings and on-the-fly analysis are especially key to efficiently using the scarce overall memory resources.

Applicable filter and trigger conditions are general logical formulae derived from basic byte-wise conditions, packet meta data such as length and error conditions, external signals, and also the results of additional analysis modules, so called system filter conditions. This integration of results is a powerful, modular and expandable way to include complex analysis into all reception related decisions such as packet filtering.

These analysis modules are as simple as a CRC check but also include more complex functionality such as address identification of up to 256 devices using binary search, and the computation and comparison of a hash-value of each incoming packet. By combining the latter two, a change-only processing of device-based cyclic data is possible providing an important functionality within the previously described application scenario [10].

It is furthermore possible, to assign packets to a virtual device ID which eases further analysis such as arrival or absence of expected cyclic occurring packets. The virtual device ID provides in turn also a robust interface to the filter module.

The output of the filter module is used for all reception related decisions as stated above as well as for system synchronization. Especially the statistical module benefits from this high flexibility: In addition to aggregating fixed values such as overall number of packets or bytes received, it maintains 256 counters to capture freely programmable events detected by the filtering process providing a powerful means for on-the-fly analysis.

As a summary, Table 2 lists all components which are part of the receive process within the reconfigurable part.

Receive Logic	PHY communication, buffer access, sequence control, error handling, basic analysis
Basic Analysis	Address detection, Cycle control
Filtering	Central processing of all available information, trigger control, packet filtering, statistics & aggregation, CPU synchronization
Trigger Logic	Conditional start/stop of recording
DMA	Transfer of linear and ring buffers into main memory
Statistics	Event counters, system statistics

Table 2: Receive Process Components

5.5 Multiprocessor Operation

The system contains two identical hard-core CPUs which are used as follows: the main CPU runs an OS such as Linux and is responsible for system configuration, user interaction and other complex tasks like downstream analysis or exception handling. Interaction with the hardware takes place through system bus, shared memory, and interrupts. The other CPU is used as an auxiliary processing engine closely coupled to hardware operations and running simple standalone applications. Communication mechanisms and application scenarios for this so-called *dedicated* CPU are presented here.

For control and synchronization tasks an additional hardware module called CPU/HW-Control (CHC) exists. It starts & stops the CPU, provides for configuration of local instruction and data storage and mediates between hardware modules and CPU as depicted in Figure 4. Hardwired links provide tokens from e.g. the RTMACs after packet reception or transmission. These are arbited by CHC, stored in a FIFO and can be fetched by the CPU when it is idle. The tokens usually would contain at least a memory address which can be accessed by the dedicated CPU via system bus and thus providing the next chunk of data or packet to be processed.

CHC can further provide information e.g. on FIFO usage and mode configuration to the CPU and also offload it by providing DMA functionality. Integration of the auxiliary CPU into the overall system is subject to Section 6.

A major application scenario is the additional use of software filtering during the reception process. While not as powerful in terms of throughput, software approaches can employ more complex rules and are better programmable. After the data has been

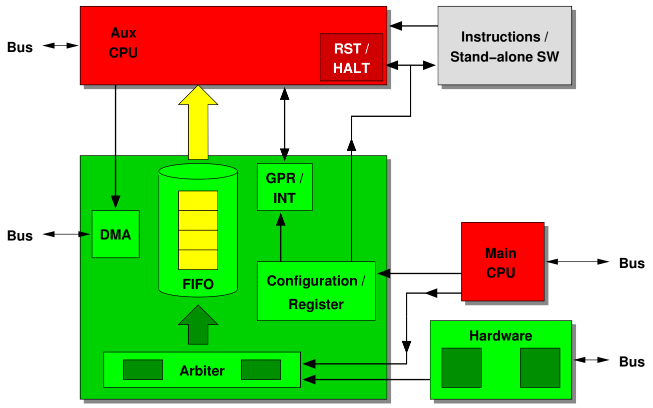

Figure 4: CPU/HW Control

pre-filtered in hardware, the remaining or disputable packets are transferred to main memory and a token is forwarded to CHC. The CPU runs a lightweight management software and e.g. the BPF packet filter interpreter [19], fetching tokens from the CHC's FIFO and accessing packet data in main memory. Eventually, packets can be kept or discarded using the DMA engine.

Another application is the timely-exact replay of formerly recorded network traffic stored in a capture file. The dedicated CPU is used to process the recording and timing information in the file and fill the timed transmission queue (cf. Section 5.3) in main memory with pointers to packets and exact timing information for the hardware transmission unit. A token can be generated by hardware whenever a packet has been sent, making the need for polling obsolete and thus relieving the system bus.

In both cases however, the scheduling and processing decisions when to access the FIFO are made by the CPU. In future scenarios, several dedicated CPUs and several FIFOs allowing for priorization of inputs are possible.

5.6 Modular Software Environment

Embedded software as described in [14] lacks abstraction while being deployed in complex and often critical environments. Therefore, the agent's main system software tries to abstract from underlying hardware by encapsulating functionality in the operating system, kernel driver, robust access mechanisms such as APIs and file systems, and client-server based modular applications.

These application modules work service based and use a uniform protocol on a socket interface for communication. This common mechanism is used for local clients (e.g. the web interface) as well as for distributed applications over a management network link, allowing a system-abstraction in a multi-agent environment.

A central controller manages the modules and provides dictionary services on availability of modules and resources as well as an update environment. Resource management is done within the driver (as for memory buffers) and distributed within the modules itself using a locking and information mechanism.

Security issues are so far not considered within the network agent's software architecture. This is due to the fact that it is designed to operate in friendly environments which will most likely not be aware of its presence. The strict distinction between analysis and management ports in the system's architecture makes it hard for recorded data to interfere with device operation. On the management side, however, the broad range of security measurements provided by the Linux operating system could be used for user authentication and encrypted control communication to ensure the integrity of the agent in any case.

5.7 Implementation

Following the architecture description, we will now highlight two specific implementation details. First we will present the flexible filtering and monitoring infrastructure, followed by an overview over the various clock domains used within the SoC, how they were generated and distributed.

5.7.1 Filtering

The filter unit is part of the receive process and closely coupled with the MAC layer within the reconfigurable part. It enables the analysis and filtering of data on-the-fly directly while it is received.

Data is arriving byte-wise in a linear fashion and is typically stored in a RAM-based buffer. For this reason, in a usual setup only one distinct position within a data packet can be analyzed, i.e. several consecutive load&compare cycles are required. This is problematic in a setup with tight timing constraints. On the other hand, a completely parallel register-based buffering of a complete packet uses to many hardware resources for a small-footprint solution as the one described here.

Thus, our concept combines both approaches using single parallel filters which are controlled by a linear VLIW-style program. These basic and customizable filters allow different byte-wise operations and process incoming data on the fly and synchronous with the receive data clock: instead of buffering the complete data for later analysis, only the results of these basic filters are stored. Thus, parallel analysis is achieved and the outcome is then post-processed. This processing is controlled by the control words, the so-called *filter rules*. Figure 5 represents the filtering process with some example basic filters (marked as **F**) and **n** filter rules.

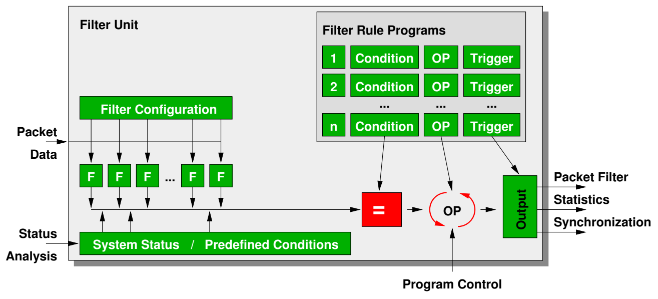

Figure 5: Filter Unit Internals

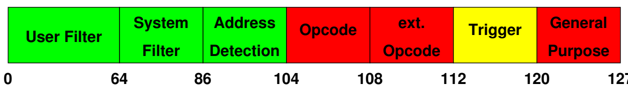

Figure 6: Filter Rule

Those filters are configured to analyze exactly one position within the byte stream and change their output once the preprogrammed conditions hold. This allows for both broad analysis of many fields as well as for deep analysis by applying many conditions to few fields at the same time. Filters have been implemented to conduct byte-wise comparison, bit-masked comparison and range checks, providing a wide range of protocol analysis capabilities.

Name	Code	Description
apply_filter	1000	Check filter conditions, apply trigger if true
skip	0100	Suspend execution for a given number of steps
skip_all	0010	Suspend execution until complete packet arrived
jump	0001	Resume execution at designated program address
done	0000	Quit execution

Table 3: Filter Program Opcodes

Name	Code	Description
packet_filter	00100000	Forward to packet filter
start_trigger	00010000	Start recording
stop_trigger	00001000	Stop recording
user_interrupt	01000000	Trigger User Interrupt
statistics	10000000	Trigger Event Counter

Table 4: Filter Program Opcodes

In each step, the current control word then basically selects which filter rules are checked, and which actions are triggered if a filter rule fires. Such a filter rule is depicted in Figure 6. The first 86 positions determine, which of the basic user-defined and fixed system filters are selected for this rule, followed by flags to select whether source or destination addresses (or both) have to be tested, and their device IDs (8 Bits each) are given. The opcode defines what will happen on a successful test, i.e. when all selected filters and the device ID(s) match. Table 3 shows the currently implemented opcodes, which can be further extended by the extended opcode field allowing for more complex logic operations.

It must be noted, that the filter program allows rudimentary program control, i.e. rules can be skipped for a certain amount of steps, until the successful reception of a packet, or it can be jumped to another position within the filter program. This is to exploit locality of important data in the packet header and to provide a more diverse set of execution possibilities.

As an example, consider the reception of a packet. The rule program starts upon the reception of the first byte of the packet. All basic filters are still empty (showing a '0' that is) and are waiting for the pre-programmed position in the packet stream to execute their pre-programmed comparison (showing a '1' if successful). Thus it makes sense to skip the execution until more information is available, e.g. until the complete header or parts of it have been received. Now execution can continue, evaluating the current status of the basic filters and system information against the current filter rule to be matched. When a preliminary result is achieved, execution can be conditionally continued using the branch instruction, e.g. based on the recognized protocol. Finally, execution can be skipped until the complete reception of the packet, resuming then with post-processing which may now include per-packet information such as length, CRC validity or the result of the device-based cyclic real-time traffic test.

In addition, it can be specified how the output of the filter rule can be used, i.e. to start and stop recording triggers, throw a user interrupt or discard the packet. This allows a very effective integration into the overall system. Especially, in combination with the general purpose field, a distinct event with an 8-bit ID can be triggered to address up to 256 statistical values, whereas the other outputs are usually summed up to allow more complex logical conditions. All outputs are listed in Table 4.

5.7.2 *Clock Domains*

The entire internal system timing is derived from 25MHz clock source. From this clock source, the required clock signals as listed in Table 5 are generated by using DCMs. With the RTMAC supporting operation in both 100 MBit/s and 1 GBit/s mode however, there is the need to multiplex the outgoing sending clock between the external 25MHz signal (100MBit/s) and the internally created 125MHz signal (1GBit/s). Reception and related logic is clocked mode-independent with the external signal coming from the PHY. For optimal synchronization with the external PHY components, double data rate output registers are used.

The PPC cores are clocked with 300MHz and 100MHz. The decision was made to run the Main CPU with 300MHz, providing enough computation power for OS functions, network- and user-I/O. The stand-alone Auxiliary CPU is clocked by 100MHz to simplify synchronization with connected interfaces but can as well be sped up to 300MHz in future scenarios.

100MHz is also used as the clock frequency for the Processor Local Bus, On-Chip Peripheral Bus and the memory controller. For external interfacing, a 33MHz clock is provided according to PCI specification with the need to drive the clock signal off-board before internal distribution to achieve better synchronization.

These clock domains are distributed within the FPGA using dedicated clock nets, ensuring minimal clock skew. Because of access restrictions to the different quadrants of the FPGA for the global clock nets, a carefully handcrafted partitioning of clock domains has to be chosen. This is especially important for the RTMACs where up to four clock domains converge as depicted in Figure 7.

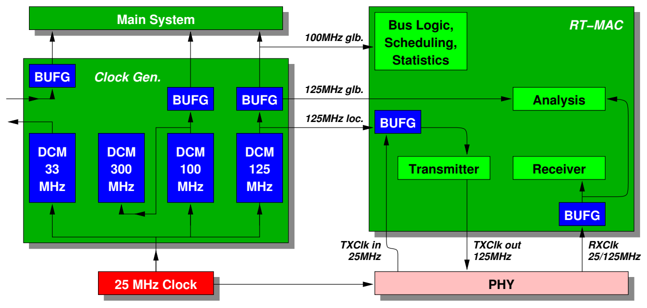

Figure 7: Clock Generation and Domains

Network	Clock Rate
Main CPU	300MHz
Aux CPU	100MHz
PLB	100MHz
PCI	33MHz
RTMAC	12.5, 25, 100, and 125MHz
TX Clock	125MHz internal, 25MHz external
RX Clock	125MHz and 25MHz external

Table 5: Clock Domains

6. IMPLEMENTATION RESULTS

Figure 8 gives an overview over the implemented SoC. Here you can see how this system partites into the real-time system, which is depicted in detail, and the high-level Linux system. To not overload the picture and because the high-level system is not in the focus of this work, the high-level system is only represented by the Main CPU.

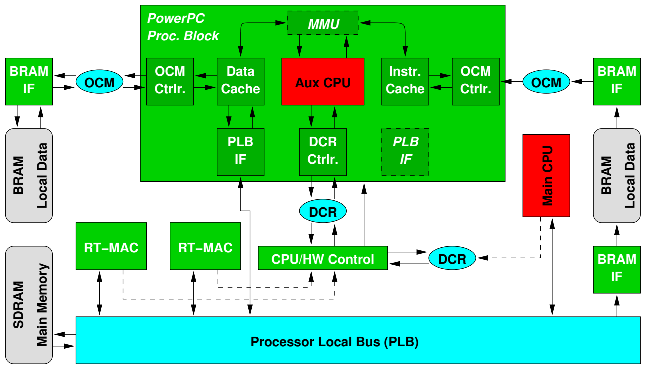

Figure 8: Network Agent System View

The Auxiliary CPU runs in standalone mode. It fetches its instructions from a local instruction memory formed by BRAM cells. This is done using the On-Chip Memory Bus (OCM) providing cache hit performance while offloading the system bus. The current program is easily accessible from within Linux via the Processor Local Bus (PLB) and therefore can be altered or exchanged as required.

Likewise, local data memory is realized by using BRAM cells. In addition, extended data memory is provided via attached SDRAM accessible through the PLB. Although both CPUs share a common address space, the local storages are overlayed within the second CPU, simplifying for example the boot-up process.

Transmission and reception of data is handled by the RTMACs, which contain required functionality as presented in Section 5. The RTMACs' DMA engines allow exchange of buffer contents with main memory regions in SDRAM via the PLB. In addition, control and necessary synchronization signals are created and forwarded to the CPU/HW control (CHC) which also handles incoming signals from the Main CPU.

The central time-stamping facility deployed throughout the system is based on the 100MHz clock and allows thus for 10ns resolution. Because a 32-Bit counter would overflow within less than 43 seconds, a 64-Bit timestamp has been implemented. For statistical values, 64-Bit counters are implemented as well to allow e.g. byte-exact amount of traffic measurement in long-term tests. Because BRAM cells are used to hold these counters, this has no major impact on resource utilization.

Bus usage was examined using an on-chip logic analyzer. The theoretical bandwidth of 6.4GBit/s resulting from the 64Bit Processor Local Bus at 100MHz proved to be sufficient to store two 1Gbit/s data streams via DMA in the main memory.

Table 6 illustrates the resource consumption of the network agent SoC: integrated into the XV2P50 FPGA, the entire system occupies about 68% of logic resources. Biggest part of these resources are the RTMACs which account for about 20% each, i.e. 40% in total. Other major parts are SDRAM and PCI controller as well as the system's Processor Local Bus logic.

The filter unit presented in this paper is part of each RTMAC. It uses BRAM cells for the rules and registers to store the settings of the basic filters. While the processing logic is rather small, the overall resource utilization is dominated by the number of basic filters, which can be customized via VHDL-generics to up to 64. The basic filters implement byte-wise functionality for comparison, range check, and bit-masked comparison´and can be reconfigured at runtime over the system bus. The filtering unit is pre-buffered and pipelined and can run at 125MHz on a speed-grade -5 FPGA.

To meet the tight timing constraints imposed on the system, the overall resource consumption must be considered carefully in order for the place-and-route process to achieve a valid result. By using extensive optimization techniques, timing closure for all implemented components could be achieved.

About 66% of the available BRAM cells are occupied, resulting in an overall used memory area of 346.5kByte used for instruction and data memory as well as buffers.

As previously noted, the Main CPU runs at full 300MHz speed whereas the Auxiliary CPU is only clocked with 100MHz to simplify system integration but can be sped up for future scenarios. The entire system timing is derived from an external 25MHz clock source. Because of the multiple clock domains, 50% of the available DCMs and 75% of the available Clock Nets are used.

Extensive two-channel testing has been conducted using Spirent's *SmartBit* professional networking equipment and by cross-connecting two agents for self-testing. With numerous traffic patterns and various filter and statistic settings as summarized in Table 7 correct and reliable operation of the device could be verified.

Resource	Total	Used	Amount
Logic (total)	23,616	16,300	68%
Logic/RTMAC	./.	4,700	20%
BlockRAM	232	154	66%
DCMs	8	4	50%
Clock Nets	16	12	75%
CPUs	2	2	100%

Table 6: FPGA Usage

Parameter	Tested Items
Packet Size	64 Bytes, 1514 Bytes , random
Mode	100 MBit/s, 1 GBit/s
Error Conditions	alignment, symbol, dribble, CRC, runt, oversized
Filter Settings	simple, combined, system conditions, cyclic data change-only, address check

Table 7: Successfully conducted testing patterns.

7. CONCLUSION

In this paper we presented an architecture for an FPGA-based network agent suitable for Gigabit Ethernet and especially for Industrial Ethernet. This agent consists of a dual-processor System-on-Chip enhanced with dedicated IP blocks based on an externally defined hardware platform described in Section 4.

It provides a flexible filtering and statistics infrastructure required for exhaustive network analysis. Filtering is based on parallel, register-based HW-filters, and so-called filter programs which are stored in the FPGA's BlockRAM and distributed RAM and can therefore be easily replaced. This combination enables powerful at-line-rate filtering patterns while using only very few hardware resources. Further flexibility is reached with the integrated programmable auxiliary processing engine and the extensible and modular software environment. A detailed introduction into the agent's architecture was provided in Section 5. Despite its capabilities, the footprint of the entire architecture is rather lightweight as shown in Section 6, resulting in a small, mobile, and robust device well suited for the intended industrial context.

The architecture as described within this paper is actively used within Siemens I&S for testing and debugging network-based process automation as commonly used within current factories.

8. ACKNOWLEDGMENTS

This work was conducted by Hans-Peter Löb as an industrial cooperation at Siemens I&S Karlsruhe under supervision by Christian Bergmann, Siemens AG, and Dr. Rainer Buchty. This paper is a summary of the work described in [17] and [18].

9. REFERENCES

[1] Florin Baboescu, Suresh Rajgopal, Lun-Bin Huang, and Nick Richardson. Hardware implementation of a tree based ip lookup algorithm for oc-768 and beyond. Design Con 2005, February 2005.

[2] Florin Baboescu, Sumeet Singh, and George Varghese. Packet classification for core routers: Is there an alternative to cams. *IEEE INFOCOM '03, San Francisco*, April 2003.

[3] Neil W. Bergmann, Peter Waldeck, and John Williams. A catalog of hardware acceleration techniques for real-time reconfigurable system on chip. In *IWSOC*, pages 112–115. IEEE Computer Society, 2003.

[4] Herbert Bos, Willem de Bruijn, Mihai Cristea, Trung Nguyen, and Georgios Portokalidis. Ffpf: Fairly fast packet filters. In *Proceedings of OSDI'04*, 2004.

[5] Gordon J. Brebner. Eccentric soc architectures as the future norm. In *DSD*, pages 2–9. IEEE Computer Society, 2003.

[6] Jan Coppens, Evangelos Markatos, Jiri Novotny, Michalis Polychronakis, Vladimir Smotlacha, and Sven Ubik. Scampi - a scaleable monitoring platform for the internet. In *Proceedings of the 2nd International Workshop on Inter-Domain Performance and Simulation (IPS 2004), Budapest, Hungary*, March 2004.

[7] Loris Degioanni, Mario Baldi, Fulvio Risso, and Gianluca Varenni. Profiling and optimization of software-based network-analysis applications. In *SBAC-PAD '03: Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, page 226, Washington, DC, USA, 2003. IEEE Computer Society.

[8] Ethereal. *Ethereal: The world's most popular network protocol analyzer*, 2006. `http://www.ethereal.com/`.

[9] Max Felser and Thilo Sauter. Standardization of industrial ethernet – the next battlefield? In *IEEE WFCS.*, pages 413–421, Vienna, Austria, September 2004.

[10] Gorden Griem. Design and implementation of a realtime media access controller for simulation and bus analysis on profinet. Master's thesis, Technische Universität Hamburg-Harburg, September 2004.

[11] Pankaj Gupta and Nick McKeown. Packet classification on multiple fields. In *ACM SIGCOMM*, pages 147–160. ACM Press, New York, NY, USA, 1999.

[12] Napatech Inc. *The Napatech Protocol and Traffic Analysis Network Adapter – White Paper*, 2006. http://www.napatech.com/media(35,1033)/-White_paper.pdf.

[13] T. V. Lakshman and Dimitrios Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *ACM SIGCOMM*, pages 203–214. ACM Press, New York, NY, USA, 1998.

[14] Edward A. Lee. Embedded software from concurrent component models. *ACM SIGPLAN Notices*, 36(8), 2001.

[15] Kyung Chang Lee and Suk Lee. Performance evaluation of switched ethernet for networked control systems. In Processings of the 2002 28th Annual Conference of the IEEE Industrial Electronics Society (IECON-2002), Sevilla, Spain, 2002.

[16] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *FPGA*, pages 87–93, 2001.

[17] Hans-Peter Löb. Integration eines prototypischen Realtime-Media-Access-Controllers in eine PowerPC-basierte Hardware-Umgebung. Studienarbeit, Universität Karlsruhe (TH) · Forschungsuniversität, Institut für Technische Informatik, May 2005. http://itec.uka.de/capp/diploma/index.php?lang=d&show=/capp/diploma/sa/loeb-2005.pdf.

[18] Hans-Peter Löb. Konzeption eines Netzwerkagenten auf einer FPGA-basierten Hardware-Plattform für Diagnose und Analyse von Realtime-Ethernet-Netzwerken. Diplomarbeit, Universität Karlsruhe (TH) · Forschungsuniversität, Institut für Technische Informatik, December 2005. http://itec.uka.de/capp/diploma/index.php?lang=d&show=/capp/diploma/da/loeb-2005.pdf.

[19] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *USENIX Winter*, pages 259–270, 1993.

[20] ProfiBus. *Technology and Application—System Description*. ProfiBus International Support Center, Haid-und-Neu-Straße 7, 76131 Karlsruhe, Deutschland, 2002.

[21] PROFIBUS Working Group "'PROFInet"'. *PROFInet Architecture Description and Specification*. PROFIBUS Nutzerorganisation e.V., Haid-und-Neu-Str. 7, 76131 Karlsruhe, 2003.

[22] Jeff Shafer and Scott Rixner. A gigabit reconfigurable programmable network interface card. Annual Affiliates Meeting, Department of Electrical and Computer Engineering, Rice University, September 2005.

[23] David E. Taylor. Survey and taxonomy of packet classification techniques. Tech. Rep. WUCSE-2004-24, Department of Computer Science and Engineering, Washington University in Saint Louis, May 2004.

[24] WildPackets Inc. *A WildPackets Technical Brief: On Gigabit Analysis System Performance and System Requirements*, 2003. http://www.wildpackets.com/elements/technical_briefs/gigabit_performance.pdf, Dezember 2005.

[25] Xilinx Inc. *Xilinx LogiCORE Ethernet Statistics*, 2005.