TosNet: An easy-to-use, real-time communications protocol for modular, distributed robot controllers

Simon Falsig and Anders Stengaard Soerensen, The Maersk Institute, University of Southern Denmark

Abstract—**This paper presents the TosNet network, created for robotics research, education, and prototyping, emphasizing ease of use, robustness, compactness, flexibility and fast hard real-time response, to allow distribution of all levels of the robot control system. The current implementation of TosNet supports up to 15 nodes, with cycle-rates up to 25 kHz, depending on the configuration. The protocol stack is completely specified as VHDL code, implemented in an FPGA. The physical layer is implemented with Toslink fiberoptic links, offering a compact, robust and highly available link technology. The network uses a shared memory model, where a block of memory is mirrored across all attached nodes each cycle, offering a simple, easy-to-use software interface between nodes.**

Index Terms—**Distributed control, Field programmable gate arrays, Modular computer systems**

I. Introduction

WHEN implementing a new robotic system in for instance a factory production line, one of the major costs is the installation and integration with existing systems. The same is true for the controller technology used for robotics research in the laboratory. One of the reasons for this is that many times the system is built more or less from scratch whenever a new setup is needed, simply because it is easier to create a new ad-hoc solution than it is to get to know and adapt a previous ad-hoc system. By taking a different approach, and instead using modular, distributed controllers, it is possible to create the necessary functionality as reusable modules with a well-defined interface. However, the long-run advantages of having a reusable module are more often than not discarded in favor of getting a solution up and running as fast as possible. This is only amplified due to the fact that many of the current technologies for connecting distributed controller modules are complex to use, and may demand a substantial amount of hardware resources compared to the actual functionality of the module. Many research projects also employ short-term team members, such as masters project students, and it is thus important to minimize the project overhead and turn-over time, in favor of more progressive work.

Manuscript submitted February 11, 2009.

Simon Falsig is with the Maersk Institute at the University of Southern Denmark, Odense, Denmark. Phone: +45 26 18 03 82; e-mail: sifa@ mmmi.sdu.dk.

Anders Stengaard Soerensen is with the Maersk Institute at the University of Southern Denmark, Odense, Denmark. Phone: +45 65 50 74 84; e-mail: anss@ mmmi.sdu.dk.

II. The State of Robotic Controllers at the University of Southern Denmark

The University of Southern Denmark (SDU) currently does research into most areas of robotic systems. This includes such aspects as high-level control, vision systems, electronics, and mechanical compositions. Much of the practical work is performed by students in bachelor or master thesis projects, and is thus often restricted to a time frame of one or two semesters. Implementing communication between distributed modules has typically meant first spending time researching the many various network technologies, and then implementing whatever technology is assumed to be best suited for that particular project. This usually results in a more or less fixed amount of work, independent of the scope of the project, and this is thus not so much of a problem for larger projects. For the student projects however, implementing a network interface easily takes up a substantial amount of the available time. Due to this, a generic communication interface is very rarely implemented, resulting in interfaces that vary wildly from project to project. Many interesting implementations are thus never used in a greater context, due to these incompatible interfaces.

The reasoning behind the TosNet project has therefore been to create a communications network and an implementation of this, which is both easy to use, and generic enough to be interfaced and used with most of the systems used in robotics research at the university, such as Field Programmable Gate Arrays (FPGAs), microcontrollers, personal computers, discrete electronics, robots, etc.

There has already been some work at SDU into creating modular, generically usable, distributed controllers using FPGAs [1]. These are well-suited for the task as they allow the implementation of a communications protocol alongside various other advanced functionalities directly in hardware, while also enabling easy and efficient interfacing to most electronic devices. This has resulted in the FPGA-based Generic Embedded Control Node (GEECON) [1], which uses ARC-net to communicate to other GEECONs. The GEECON fulfills the need for a generic, distributed controller, but as the platform consists of several boards and chips, it is too large,

ROBOCOMM 2009, 31st Mar – 2nd Apr 2009, Odense, Denmark.
 ISBN 978-963-9799-51-6
DOI 10.4108/ICST.ROBOCOMM2009.5806

complex and expensive to use in most simple applications.

The idea of using FPGAs is sound though. Not only due to their flexibility, but also because FPGAs are in widespread use at SDU, and many students already are using these in both courses and semester projects. Implementing the communications protocol in an FPGA, would thus give the students a familiar environment and technology to work with, while also providing the possibility of combining some of the student projects and the communications protocol stack in a single FPGA chip.

Flexibility and ease of use has not been the only design requirement though. To be able to use the network with actual robotic systems, a certain level of performance needs to be guaranteed. This includes real-time, isochronous transfer of data, and network cycle-rates (how often the nodes receive new sets of data) above 2 kHz. This is enough to control virtually any industrial robotic system, according to [1].

The requirements for the communications network have thus been:

- Must be very easy to use, with regard to software interface, necessary hardware components and design skills
- Real-time operation with isochronous data transfer
- FPGA implementation
- Must be able to achieve cycle-rates above 2 kHz
- Must be usable for many different types of nodes: controllers, sensors, motors, etc.
- Must have a reasonably low implementation cost per node
- Possibility of interfacing to a standard Personal Computer

III. Existing network technologies

A lot of network technologies for connecting distributed controllers already exist, and a number of these have been investigated, regarding their ability to fulfill the specified requirements.

A plethora of Ethernet-based real-time protocols are available, such as Ethernet PowerLink [2], Ethernet IP [3] and EtherCat [4]. These all provide at least 100 Mbps operation and various synchronization features, but all require an external Ethernet physical layer chip coupled with either further specialized protocol chips, and/or a TCP/IP stack implementation. Even though the necessary chips can be easily interfaced to an FPGA, and a TCP/IP stack can also be implemented, this will almost certainly require advanced skills in Printed Circuit Board (PCB) design and FPGA coding, even when just using an existing implementation with a custom design. These protocols thus fulfill the performance requirements, but not those regarding ease of use.

CAN-bus [5] and serial protocols like RS485 [6] and others, typically do at most require a single transceiver, and a UART implementation in a microcontroller or FPGA. Both chip and implementation are much simpler than their Ethernet counterparts, but sadly the performance is lacking. CAN-bus can only manage 1 Mbps, while serial protocols typically need special care to be taken with regard to signal integrity, if speeds above a few Mbps are to be obtained.

More specialized networks like ARC-net [7] and Sercos [8] could also be a possibility, but both these require special, proprietary controller chips or FPGA cores. This results in relatively high implementation costs per node, in addition to their complexity.

IV. TosNet Protocol

An existing network technology that fulfills all the stated requirements has thus not been found. Instead, it was decided to implement a custom protocol, using fiberoptic Toslink components [9]. These are used widely for digital audio transfer in consumer electronics, and are thus both cheap and very accessible, and provide an adequate transfer rate of up to 15 Mbps. They are easy to use, as both Toslink transmitters and receivers implement the necessary signal conditioning, and thus simply feature a single, digital, raw serial data pin, which can be connected directly to an FPGA input/output pin. Additionally, the optical fibers are immune to noise, meaning that students will not need to consider EMC issues when cabling the network.

The rest of the network stack has been written completely in VHDL code, and implemented in a Xilinx Spartan3 FPGA [10]. This section describes the further details of the protocol, which has been named TosNet.

A. Network operation

TosNet employs a ring topology with an automatically assigned master node to setup and control the network operation. A shared memory model is used, where a block of memory is distributed between the nodes, and updated during each cycle.

The shared memory block is divided between the attached nodes, with each node being assigned an equally-sized part, which is used to handle communication between the master and that particular node (in the case of the master node, its part of the memory block can be used for communication with all slaves). These parts are then further divided into a number of segments that can be individually enabled or disabled for each node, see figure 1. The segments have an 'in area' to carry information from master to slave, and an 'out area' to carry information from slave to master. Every node can read

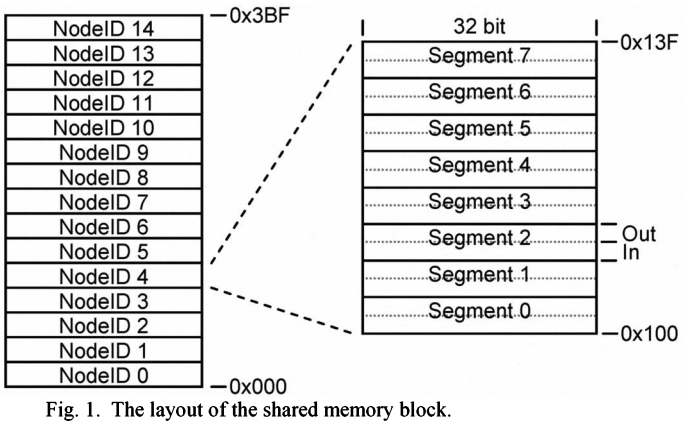

Fig. 1. The layout of the shared memory block.

Digital Object Identifier: 10.4108/ICST.ROBOCOMM2009.5806
http://dx.doi.org/10.4108/ICST.ROBOCOMM2009.5806

the entire block, but only the master is allowed to write to the 'in areas' and each slave can only write to its own allocated 'out areas'. Every node can thus tap into all communication, thereby making it possible for slaves to communicate directly with each other, while also preventing them from interfering with communication that do not regard them. Only those segments that are enabled are transmitted over the network, thereby increasing performance.

The master node initiates the memory block update by sending out a packet containing its own current, local version of the shared memory block. The first slave node in the ring then receives this, and uses it to update its own local copy of the memory block. The received packet is then updated with the current state of the slave node, and transmitted to the next node in the ring, and so forth. To obtain better performance, the slave nodes start transmitting the packet as soon as it is received, updating the contents on the fly.

When the master receives the packet after it has traversed the ring, it uses the packet contents to update its own shared memory block. The next cycle is started as soon as the packet is completely received.

The network operation can be seen in figure 2.

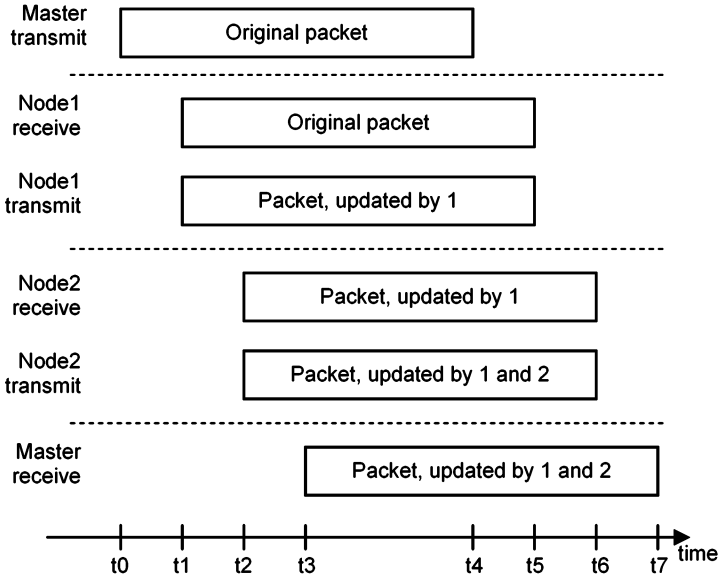

Fig. 2. The network operation shown for a single cycle.

B. *Architectural overview*

The protocol is structured after the OSI model [11], with a physical-, a datalink-, and an application-layer. Except for the network interface hardware (the Toslink transmitter and receiver), all functionality of the protocol stack is implemented as VHDL code in a Xilinx Spartan3 FPGA. This structure is depicted in figure 3.

Each of the layers is implemented as one or more VHDL modules, using synchronous state machines.

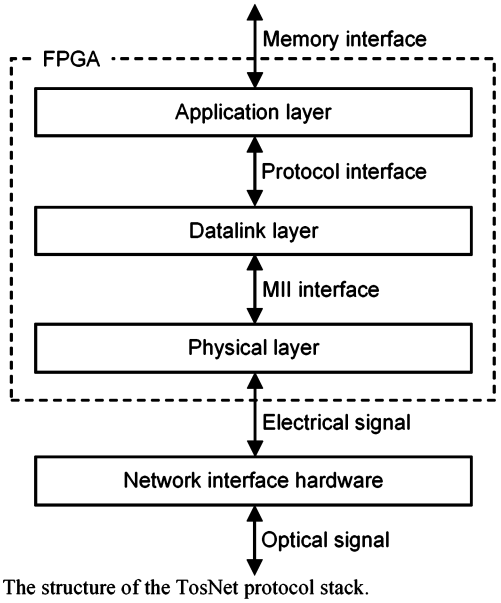

Fig. 3. The structure of the TosNet protocol stack.

C. *Physical layer*

The physical layer handles the low-level en- and decoding of data, and transmission between two directly connected nodes. It communicates with upper layers using a Medium Independent Interface (MII), as specified in IEEE802.3 [12]. This allows both the physical and the upper layers to be exchanged with other layers using this interface, for instance Ethernet physical layer ICs. The interface specification restricts the data rate of the physical layer to either 10 Mbps or 100 Mbps. As the used Toslink components have a maximum transfer rate of 15 Mbps, 10 Mbps is chosen.

Data to be transmitted over the network are first scrambled using a linear feedback shift register (LFSR), and then encoded first using 4B5B encoding (increasing the actual data rate to 12.5 Mbps) and then with non-return-to-zero-inverted (NRZI) encoding (giving a data clock of 6.25 MHz). This encoding ensures that at most nine consecutive bit cells without a level change will occur on the electrical interface between the physical layer and the Toslink transmission components. This is important for the ability of the receiver to perform clock extraction on the received signal, and thus correctly decode the data.

Clock extraction is performed using oversampling, sampling the 6.25 MHz data signal at 50 MHz.

D. *Datalink layer*

The datalink layer handles the CRC error detection, synchronization between nodes, watchdog timer, and setup of the TosNet network.

The implemented CRC check uses an 8 bit CRC polynomial to calculate a checksum for every packet. This checksum is appended to the packet, and the receiver can then use it to detect transmission errors. As the isochronous nature of the network makes it impractical to retransmit erroneous data, the data in the packet is simply discarded if an error is discovered, and an error is signaled to the application attached to the protocol stack. The erroneous packet is still transmitted on to the next node though, to maintain network operation. As each node recalculates the CRC checksum for each packet it transmits, the last four bits of the checksum are inverted if an error is detected, thus forcing a CRC error in the next node, and thereby preventing the remaining nodes in the ring from

Digital Object Identifier: 10.4108/ICST.ROBOCOMM2009.5806
http://dx.doi.org/10.4108/ICST.ROBOCOMM2009.5806

falsely accepting the data as being valid.

The node synchronization is performed using a simple timer and a static delay depending on the position of the node in the ring. This delay is automatically calculated for each node during network setup. The timers in the slave nodes are started whenever a packet is completely transmitted, and due to the individually calculated delay, they fire at approximately the same time in all nodes (looking at figure 2, the delay for node 1 is t7-t5, the delay for node 2 is t7-t6, while the master node synchronization signal fires when it is done receiving a packet).

The watchdog timer is reset whenever a new packet is received. If no packets are received for a set amount of time, the watchdog times out, and restarts the network setup procedure. This is for instance used when new nodes are added to the ring during operation.

The setup process includes determining which node to use as master, finding out how many nodes are connected, and finally what registers are enabled in what nodes. This is done through a series of configuration packets.

E. Application layer

The application layer handles the shared memory model, and the transmission and receiving of the register contents. The shared memory is implemented in a dual-port BlockRAM [13] in the FPGA, and is double buffered to prevent problems with both the application layer and an external application accessing the same locations simultaneously.

Additionally, the application layer holds a few status counters that count the number of detected errors, resets and packets sent since powerup.

The interface presented to external applications is simply one of the ports to the BlockRAM, enhanced with a few commit signals to control the double-buffer functionality. External applications assert the commit signals (one for the in area and one for the out area), when they have finished accessing the specified area for that cycle. The application layer detects this, and switches the buffers if new data is available.

F. Deployment and use

To ensure that people wanting to use TosNet are not burdened with such things as having to setup internal FPGA time constraints for the protocol implementation, the protocol has been compiled into so-called Relationally Placed Macros (RPM), using the Floorplanner tool from Xilinx [14]. This makes it possible to create a single, completely placed and routed module in a binary file, which can then be distributed and instantiated as a black-box in new designs. This also prevents access to and accidental modification of the source code of the protocol.

To provide a direct interface to a PC, the TosNet RPMs have been implemented on a Xilinx Spartan3 PCI Express Starter Kit [15]. This starter kit comes with a Spartan3 1000 FPGA, and a 1x PCI Express interface. Simple device drivers have been created for both Windows and Linux, which allow access directly to the TosNet shared memory block from usermode applications on either operating system.

The PCI-express interface allows a PC to act as a (master) node in TosNet, thus including the PC in the real-time domain of the network. By using real-time software on the PC, real-time performance of a complete robotic system can be guaranteed.

The major drawback of the PCI-express solution is that it requires a relatively expensive hardware-board, along with device drivers and APIs for the relevant operating systems and programming languages. As these factors will inhibit the usefulness of TosNet, a TosNet to Ethernet bridge has also been developed, using an Ethernet-enabled microcontroller, the Digi Connect ME [16]. This is connected to a Spartan3 TosNet node through an RS232 serial link running at 115200 bps, and allows an Ethernet enabled computer to access a TosNet network, using for instance UDP packets to communicate the contents of the shared memory block. The low speed of the RS232 link may be an issue for more demanding applications, but in many cases it will be perfectly adequate. In the future the Connect ME may be exchanged with the more powerful Connect ME 9210, which has a 16 Mbps Serial Peripheral Interface (SPI), to overcome this limitation.

The received data is asynchronously injected into a dedicated port of the dual port shared memory block, and does thus not interfere with the isochronous, real-time operation of TosNet itself. However, by using this bridge, real-time operation of the complete system cannot be guaranteed. For most simpler applications this is not an issue though, compared to the advantages of allowing any PC application to access TosNet, using only a few lines of code.

G. Features and performance

The main features of the network are listed below:

- 10 Mbps data rate
- Ring topology with up to 15 nodes
- Each node can use up to eight 32 byte memory segments (each with 16 bytes in, 16 bytes out)
- Cycle rates from ~320 Hz (15 nodes, all segments enabled) to ~25 kHz (2 nodes, 1 segment enabled)
- 8 bit CRC error detection
- Protocol stack completely implemented in an FPGA
- Uses a single 50 MHz clock
- Synchronization between nodes with microsecond accuracy
- Optical Toslink transmission lines for noise immunity
- Automatic network setup and master node assignment

The cycle rate of the network varies with the number of attached nodes, and enabled memory segments. It can be found using the formula:

$$f_{cycle} = \frac{1}{\left(\frac{32bit + (256bit \times r)}{10^{Mbit/s}} + i \times 5.46\mu s \right)}$$

Digital Object Identifier: 10.4108/ICST.ROBOCOMM2009.5806
http://dx.doi.org/10.4108/ICST.ROBOCOMM2009.5806

In the formula, r is the total number of enabled segments (each 256 bits large), and i is the total number of nodes, which is multiplied by the node-to-node delay of 5.46 µs (this value has been experimentally measured, and takes into account both the delay in the protocol stack, and the time taken to physically transmit the data). The added 32 bit are due to overhead (CRC checksum, and packet headers and trailers).

For a network consisting of 4 nodes, each with 2 segments enabled, the result is thus:

$$f_{cycle} = \frac{1}{\left(\frac{32bit + (256bit \times 8)}{10^{Mbit/s}} + 4 \times 5.46\mu s\right)} \approx 4.4\text{kHz}$$

The formula has been experimentally verified to give results within about 5% of what is experienced in practice. The deviation is due to the used value for the node-to-node delay, as the measured value is not completely accurate, and as the delay also fluctuates during operation, because of clock variations between the nodes.

The node-to-node delay is also used in the synchronization timers. The delay used for a particular node is found by multiplying the node-to-node delay by the number of cable segments a packet will need to traverse between the node and the master node. The synchronization accuracy has been measured to be close to 1 µs for a four-node network.

H. Results

TosNet has initially been tested in a few simple motor controller applications, where an application on a PC accessed the network using a Xilinx Spartan3 PCI Express Starter Kit. Through TosNet, this was connected to a Simple-Solutions Zefant XS3 [17] board with a Xilinx Spartan3 1500 FPGA. The PC application could then set the speed of a motor connected to the Zefant board, and read back the motor current.

A full suite of exhaustive tests for all possible conditions has not yet been completed. Instead, a more pragmatic approach with focus on testing with practical applications has been used. Under all experiments TosNet has performed as expected, and without any repeating errors. However, it has been seen that it is possible to provoke continuous errors by repeatedly transmitting certain bit patterns resulting in DC offsets at the transmitter components. Due to the scrambler employed in the protocol, the risk of accidentally creating such a pattern is extremely remote though. The problem can be corrected in a future version, by updating the bit-encoding to an offset-free algorithm, such as 8b/10b.

TosNet fulfills all the stated requirements. This includes being easy to use, and it is assumed that a reasonably skilled student can get a TosNet node up and running in a custom design in a matter of hours, if just provided with the necessary code and components.

V. Application example: MiniVGT

To demonstrate the advantages of using distributed controllers in general, and the functionality of TosNet in particular, it has been used to implement a distributed controller for a Variable Geometry Truss (VGT) robot.

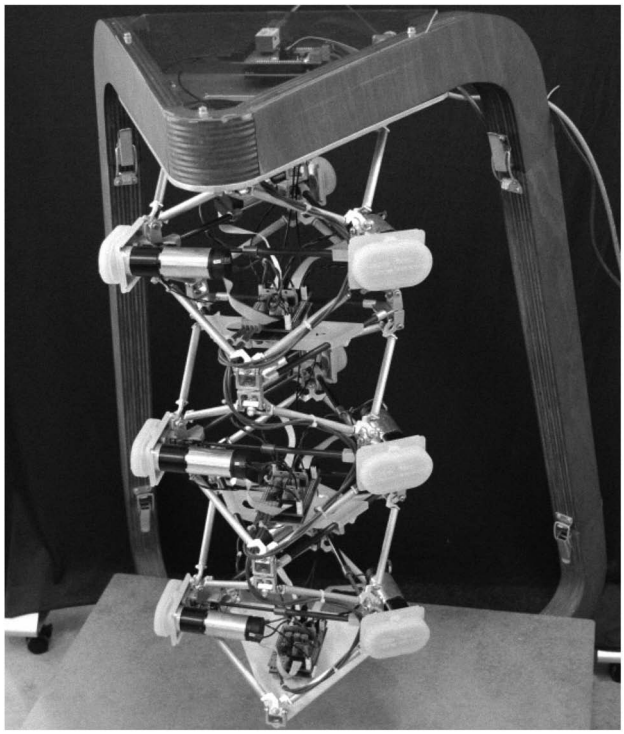

Fig. 4. The MiniVGT in its current state, with the TosNet based distributed controller.

The skeleton of the MiniVGT was originally designed by NASA as a low-cost demonstration of the mechanical concept of VGT robots. In 1998 it was acquired and upgraded by SDU, and several projects have worked on various parts of it since then. It consists of three similar sections, each controlled by three DC motors with encoders, and is thus both easy to control, and also an obvious target for a distributed controller.

The MiniVGT can be seen in figure 4.

A. The previous, monolithic controller

In its previous incarnation a team of bachelor project students equipped it with a single monolithic controller. This connected all three sections together with large ribbon cables, and long power cables running from a single motor driver board at the top of the robot, to the nine motors. The controller was a combination of an Atmel AtMega128 microcontroller [18], and a Xilinx Spartan3 FPGA. Although this worked, it clearly illustrated some of the disadvantages of using a monolithic controller:

- No flexibility with regard to adding new functionality to the robot (tools, extra sections, etc), as the complete controller would need to be redesigned
- Encoder signals need to travel a long way through the ribbon cables, making them susceptible to noise
- Bulky, inelegant cabling

B. The current, distributed controller

The PCBs, along with the controller and the motor drivers, were removed, and instead a distributed system, using TosNet for communication, was installed. The new controller uses one node for each section, and a fourth master node to interface to

Digital Object Identifier: 10.4108/ICST.ROBOCOMM2009.5806
http://dx.doi.org/10.4108/ICST.ROBOCOMM2009.5806

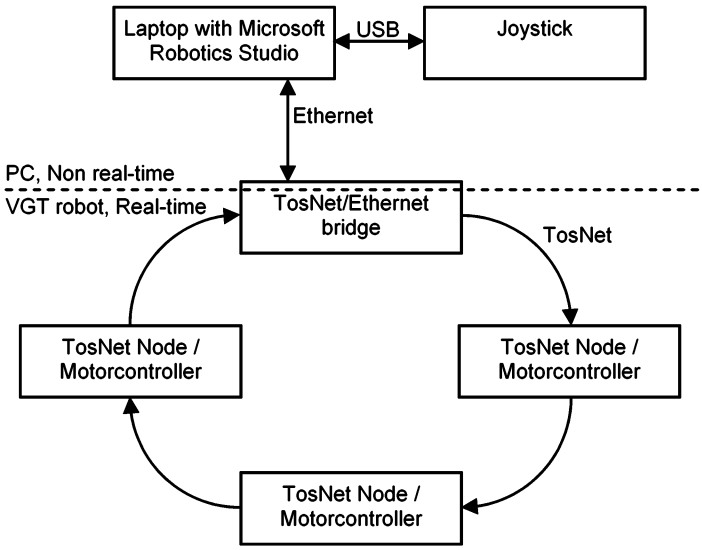

Fig. 5. The MiniVGT controller system.

a Local Area Network (LAN) using an Ethernet interface. A block diagram of the controller system can be seen in figure 5.

Each of the three section nodes consists of:

- Simple-Solutions Zefant LC3E board [19], with a Xilinx Spartan3E 250 FPGA
- Three PWM based motor drivers
- TosNet interface

The FPGA holds the TosNet protocol stack, and a motor controller for each of the three motors. The motor controller takes a desired motion speed and target position as input, and emits a PWM signal for the motor drivers as output. It uses the encoder signals from the motors as feedback, in a simple P-regulation loop.

The master node is a TosNet to Ethernet bridge, and consists of:

- Simple-Solutions Zefant LC3E board, with a Xilinx Spartan3E 250 FPGA
- Digi Connect ME ARM7 based embedded module [16] with an Ethernet and an RS232 interface
- An LCD display for outputting information such as the assigned IP address to the operator
- TosNet interface

The Digi Connect ME simply forwards data received in UDP packets over its Ethernet connection directly to the FPGA, through the RS232 interface. The FPGA receives these and stores the data in the appropriate positions in the TosNet shared memory block.

To feed the VGT with information, a PC with Microsoft Robotics Studio (MSRS) [20] was used (2.5 GHz Intel Core 2 Duo, 2 GB RAM, Windows XP SP3). A custom MSRS service was developed, to turn input from a standard joystick (a Saitek Cyborg Evo Wireless [21]) attached through the Universal Serial Bus (USB) interface, into target positions for the robot, and transmit these over the LAN connection. The kinematics model implemented is very simplified, but still manages to demonstrate the functionality of the TosNet protocol.

The robot reacts as good as instantly to the movement of the joystick.

VI. Conclusion and Future Work

The TosNet protocol works as intended, and obvious advantages over previous monolithic controllers are evident from the MiniVGT application example. In its current state the protocol fulfills all the listed requirements, making TosNet a valuable tool for research and student projects.

Even though the TosNet protocol is more or less complete, SDU has further plans for research into real-time communications for distributed controllers. This includes creating a generic FPGA implementation of the Decentralized Software Services Protocol (DSSP) [22] used in MSRS, and attempts at mapping this onto a real-time Ethernet network. This is not intended to have the same accessibility as TosNet, but instead for use as a general interface between all kinds of nodes in a loosely-coupled, distributed system, that can be used for industrial applications and long-term research projects, outside the scope of TosNet.

References

[1] A. Soerensen, "Modular control of industrial mechanics" Ph.D. dissertation, The Maersk Mc-Kinney Moeller Institute for Production Technology, University of Southern Denmark, Odense, Denmark, 2003, pp. 57-68 and pp. 49-54. Available: http://www.stengaard.net/anders-s/Research/publications.html

[2] Ethernet Powerlink [Online]. Available: http://www.ethernet-powerlink.org

[3] Ethernet I/P [Online]. Available: http://www.odva.org/

[4] EtherCat [Online]. Available: http://www.ethercat.org/

[5] CAN-bus [Online]. Available: http://www.can-cia.org/

[6] *Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems*, ANSI/TIA/EIA Standard 485-A, 1998.

[7] ARC-Net [Online]. Available: http://www.arcnet.com/

[8] Sercos [Online]. Available: http://www.sercos.com/

[9] Toshiba Fiberoptic Module TOTX147PL / TORX147PL datasheets

[10] Xilinx datasheet DS099, "Spartan-3 FPGA Family Data Sheet", 2008

[11] *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, ISO/IEC Standard 7498-1, 1994, pp 28-31.

[12] *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method and physical layer specifications,*IEEE Standard 802.3, 2005, Section 2, pp 1-58.

[13] Xilinx datasheet DS444, "Block RAM (BRAM) Block(v1.00a)", 2004

[14] Xilinx Application note XAPP422, "Creating RPMs Using 6.2i Floorplanner", 2004

[15] Xilinx Spartan-3 PCI Express Starter Kit [Online]. Available: http://www.xilinx.com/s3pcie

[16] Digi Connect ME [Online]. Available: http://www.digi.com/products/embeddedsolutions/digiconnectme.jsp

[17] Simple Solutions Zefant XS3 [Online]. Available: http://simple-solutions.de/shop/product_info.php?info=p1_Zefant-XS3-Micromodule.html

[18] Atmel ATmega128 microcontroller [Online]. Available: http://www.atmel.com/dyn/products/product_card.asp?part_id=2018

[19] Zefant LC3E [Online]. Available: http://www.simple-solutions.de/catalog/product_info.php?products_id=51

[20] Microsoft Robotics Studio [Online]. Available: http://www.microsoft.com/robotics

[21] Saitek Cyborg EVO Wireless [Online]. Available: http://www.saitek.com/uk/prod/evowireless.htm

[22] H. F. Nielsen and G. Chrysanthakopoulos, "Decentralized Software Services Protocol – DSSP/1.0"

Digital Object Identifier: 10.4108/ICST.ROBOCOMM2009.5806
http://dx.doi.org/10.4108/ICST.ROBOCOMM2009.5806