The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

An FPGA based approach to increased flexibility, modularity and integration of low level control in robotics research

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

Abstract— **One of the major costs and inhibitors to practical robotics research is the time required for the creation of the embedded systems that implement the discrete event control in experimental robot systems. Using a standardized system architecture can reduce this time, but is often not feasible due to the rapidly varying requirements present in robotics research, and the considerable cost associated with implementing a full-blown standard. This often results in short-term ad-hoc solutions with low reusability and therefore high long-term cost to robotics labs.**

By utilizing the flexibility offered by reconfigurable electronics, in the form of Field Programmable Gate Arrays (FPGAs), we have arrived at an architecture for low-level control, which is flexible enough to bridge the traditional gaps between size, performance, topology, reusability, high-level integration, and ease of use. Several iterations, have led us to our current implementations: TosNet, which is optimized toward rapid prototyping in experimental modular robotics, and µTosNet, which has been optimized for simpler experiments and teaching. In this paper we present the overall concept and its background, as well as the two TosNet frameworks, examples of applications, and thoughts on dissemination and future work.

I. Introduction

Control systems for experimental robots follow the overall trends in the automation industry, and their architectures have undergone the same migration from monolithic to modular and distributed, as seen in other areas of computer control, from factory automation to peripherals of Personal Computers (PCs). Modular and distributed architectures have many potential benefits with regard to aspects such as reusability and the possibility of a tight integration into modular mechanical units. The ultimate vision is to arrive at "plug-and-play" devices that can be aggregated into a complete custom, robotic system, as easily as connecting a keyboard and mouse to a PC.

An abundance of high- and low-level architectures and technologies for interfacing computers to the outside world have been defined over time, each representing a compromise between technical and economical parameters (such as performance, price, ease-of-use, flexibility, physical size, robustness, intended life-span, compatibility, possible topologies, acceptance from others, etc). The more successful of these are those that are optimized with respect to segments of customers with relatively uniform and static overall requirements, e.g.: Personal computers (PCI, USB, etc), Automotive (CAN), Factory automation (DeviceNet), Aerospace (VME), Laboratory and plant control (LabView), etc. In experimental robotics however, we are faced with different requirements from project to project, and no single architecture therefore seems to fit very well. On the contrary, changing technical demands and possibilities, as well as the need for compatibility with external project partners, make it necessary to constantly embrace new technologies and to find ways to deal with the ensuing Babylonian confusion.

As there is a considerable workload associated with adopting a new standardized architecture, this is impractical to do whenever a new project comes along. In our lab, this has led to many ad-hoc solutions over time, with much redundant work being done due to the resulting lack of compatibility between projects. Another problem is that experts from the many different disciplines typically involved in robotics research (electronics, robotics, control, mathematics, etc.), employ different approaches to deal with systems integration. These range from using full-custom, (locally) optimal solutions with low reusability, over sub-optimal standardized solutions imposing arbitrary restrictions on the system, to simply avoiding interacting with the physical world altogether. These approaches are hard to combine and inhibit interdisciplinary robotics research.

The advent of configurable, digital integrated circuits, like Field Programmable Gate Arrays (FPGAs), has allowed us to challenge these conventional dynamics of controller architectures. Today, FPGAs can be used to create complex digital systems containing I/O, processor, memory and communication components all in a single chip, while allowing us to specify, adjust and change any detail in the components or the system without any physical changes to the electronics – hardware can thus be developed using tools and workflows resembling those known from software development. As we have focused on modular control of industrial robotics, we have used this "System on a Chip" (SoC) technology to develop a networked "Controller Node on a Chip" architecture, where much of the conventional controller functionality is specified as separate and reusable modules in a hardware description language (in our case VHDL). The code is then synthesized and implemented into

Manuscript received March 10, 2010.

S. Falsig and A. S. Soerensen are with the The Maersk McKinney Moeller Institute, University of Southern Demark, Odense, Denmark (phone: +45 65 50 35 93; e-mail: {sifa, anss}@mmmi.sdu.dk).

a bit-file that can be loaded into the FPGA, thereby configuring it with the specified modules (we will use the term FPGA "gateware" to describe this configuration).

We have already demonstrated the use of FPGAs for flexible I/O in the EU project DockWelder [1]-[4], and a similar approach is also described in [5]. Our first implementation of the FPGA-based TosNet embedded communication network is reported in [6]. Our approach is similar to the wireless sensor network node architecture described in [7] and [8], with regard to the use of FPGAs for abstracting different devices to a common interface, and the modular board design employed, which allows a number of standard modules to be combined with application specific sensor and network modules.

II. Goals

The main goals of our work are three-fold:

1) **Reduce the development time** of experimental robotic controllers to arrive faster and cheaper at fully working demonstrations of new technology and concepts.
2) **Increase the reusability** of experimental systems and components, thus increasing the life-span and utilization of these, and reducing the amount of redundant work.
3) **Ease the use** of interacting with experimental low-level controllers, to open experimental robotics up to a wider audience, and to allow high-level developers a tighter integration with the physical robots, without involving them in the low-level particulars of embedded systems.

One of the means of arriving at goals 1 and 2 in particular, will be the creation of a library containing template circuit layouts and associated VHDL code that can be reused across projects. With such a library, many regular nodes could be created by copy-pasting and combining existing entries.

The stated goals are hard to formalize and evaluate quantitatively, but a place where we expect to see a definite effect, is in the quality and usefulness of master thesis projects. Here, even small reductions in development time have a high ratio to the typical Danish project timeframe of four months. We thus hope that our master thesis students will be able to contribute more to long term research projects, while still arriving at something that actually works on its own during their project.

In the case of the third goal, we hope to see an increased amount of cooperation internally at our department, where there has traditionally been a severe mental and practical barrier between the groups working on actual hardware systems through low-level programming (VHDL/C/C++), the more abstract software researchers working in Java, C# and similar, and the robot motion and kinematics researches working with mathematics.

III. Definition and Structure of a Node

We define a node as a device that abstracts and maps the functionalities and interfaces of one or more application specific device(s) (actuators, sensors, robots, processing units, etc) in a modular, robotic system into a general form, and presents these to other nodes in the system over a common network.

On a conceptualized level, a node in a distributed system will need to perform the following tasks:

1) **Processing:** The necessary operations involved in the data flow of the node. This might simply be moving data between I/O systems and the system network interface, but can also involve more high-level functions.
2) **I/O system:** Interfaces the signals to and from the outside world to the processing domain.
3) **Signal conditioning:** Electronics that will interface the I/O system to the sensors and actuators of the robot.
4) **System network interface:** Will make the node a part of the network chosen for the system. Strictly, this is a special case of I/O and signal conditioning, but for clarity, it will be treated separately and as a single block.

A block diagram of these tasks can be seen in fig. 1.

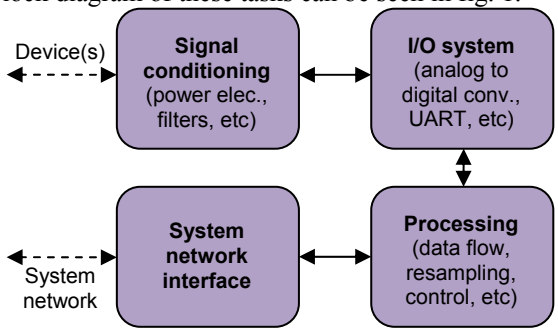

Fig. 1. Conceptualized block diagram of the tasks in a node.

IV. The Node-on-Chip Concept

FPGA technology has allowed us to develop a node architecture based on a single FPGA chip, where any combination of network stacks, processing units, and I/O components can be implemented independently of each other, and simply run in parallel. The only limitation comes from the amount of resources available in the used FPGA chip. This is in contrast to a microcontroller-based solution, where new software modules will almost always have direct influence on the possible timing performance of existing modules, as only one module (or thread) can be executing at any one point in time. Implementing for instance numerous timing-critical device interfaces is thus much easier to do on an FPGA, where they simply run in parallel, than on a microcontroller, where they will need to share the processor. As we need to create flexible I/O, another advantage of FPGAs is the very fine granularity of the timing. External pins can be controlled directly on each clock edge, typically at speeds up to 50-100 MHz. Doing this on a microcontroller would require a faster clock speed (to allow for necessary additional instructions in between I/O operations), and most probably some mucking about with

low-level assembly code.

FPGAs thus have some clear advantages with regard to I/O. The projects in [1]-[5], [7], [8] also all use the FPGA for I/O, although with an additional microcontroller for certain processing and communication tasks. However, with recent advances in the available resources on FPGA chips, it is perfectly possible to implement most of these tasks directly in the FPGA. This may or may not include a softcore processor, such as the highly configurable Xilinx MicroBlaze, for those tasks that are simply better suited for a processor than for an FPGA. By keeping as much as possible inside the FPGA, the boards can also be made simpler, and developers only need to use a single set of software tools (in our case Xilinx ISE/EDK).

Compared to microcontrollers, the availability of FPGAs with on-chip support for signal conversion and conditioning is very limited though. To our knowledge, the Actel Fusion [9] is the only mixed-signal FPGA-series available currently, and is therefore hardly of practical use for a generic system, when considering the broad array of I/O solutions used in our field. The signal conditioning, and often part of the node I/O system (e.g. an A/D converter chip) will thus need to be placed as external components. The logical interfaces to these will then be placed in an FPGA part of the I/O system.

A node in a Node-on-Chip system will therefore typically consist of an FPGA chip attached to some device-dependent electronics, and some static support electronics (power supply, network components, status LEDs, etc). Due to the required extra electronics, the Node-on-Chip name aims more at being an architectural term, describing the implemented functionality of the on-chip system, rather than the actual placement of the hardware contents.

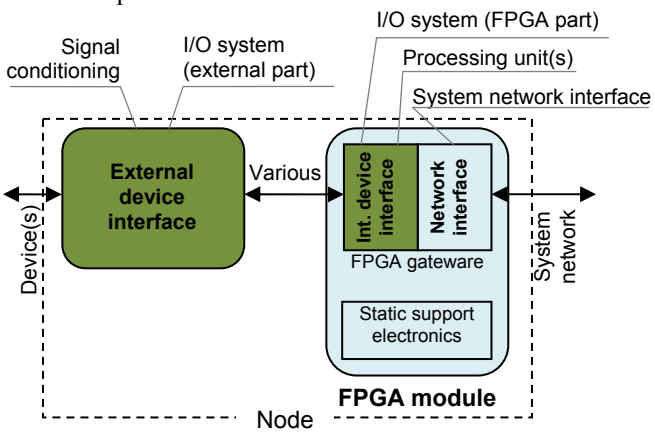

Fig. 2. The various node parts employed in a Node-on-Chip, including how the conceptualized blocks of fig. 1 fit in.

As seen on fig. 2, the various node parts are grouped into four separate categories, described in the following sections.

A. FPGA module

The heart of our node architecture is a generic, reusable FPGA module, containing the FPGA chip, some form of I/O connectors, and a power supply. By providing only the most necessary components it is possible to create rather compact modules, usable for a wide range of applications.

The major reason for having a generic, separate module for the FPGA and its power supply is to enable students and developers without insight into high-speed electronics and advanced soldering, to take advantage of the architecture by concentrating on the application specific interfaces. It also allows us to more or less "mass-produce" the rather advanced multi-layer FPGA boards, while developers and students will only need to create boards for the, often simpler, device-specific external interface electronics.

The external components of the network interface can be deferred to a separate module in order to create an even more generic FPGA board. This will allow experiments with different networks and ease reuse of the board across projects that are not using the Node-on-Chip concept.

B. External device interface

The parts of the interfaces to application specific devices that cannot be implemented inside the FPGA chip will need to be attached as external interface electronics. This could include power electronics and analog signal and signal conditioning components. These are typically implemented on application-specific, custom-made printed circuit boards (PCBs), on to which the FPGA module can be mounted.

Even though the specific combination of these components is unique for each different application, a lot of subsystem circuit designs and PCB layouts for the individual support components can be reused. It is thus possible to create a library of these to go along with the library of associated VHDL modules. With a reasonable library of common I/O functions, application specific I/O boards can be designed and created fast and efficiently.

If space is not critical, a generic I/O board with a broad selection of commonly used I/O and power components can be used, e.g. in teaching or simple test setups.

C. Internal device interface

The main purpose of a node is to interface to one or more application specific devices. As much as possible of this functionality is created by specifying it in VHDL, and implementing it as gateware modules inside the FPGA.

Possible hardware interfaces could include simple data-conversion or -transfer functionality such as PWM signal generators for motor control, and Serial Peripheral Interfaces (SPI) for interfacing to external analog support components. However, more advanced modules, such as complete high-level network protocol stacks, high-level motor drive control systems, and even soft-core processors are also possible.

As these hardware interfaces are specified as VHDL code, there is an obvious possibility of reuse. We hope to arrive at a library of VHDL modules, containing highly-optimized versions of often-used device interfaces. This is similar to the work described in [8], although not limited to just low-level interfaces to sensors and actuators. With such a library, creating the device interface part of a specific node could in

many cases be reduced to choosing and connecting the right blocks in a VHDL editor.

D. Network

The network needs to be able to easily connect the nodes to each other, while also providing an easy-to-use interface to the hardware modules inside the FPGA.

For the Node-on-Chip concept, we have settled on using a shared memory model (fig. 3), where each node is assigned some memory in a common, shared memory space. The system network is then used to provide the implementation of this memory block, and to keep it updated across all attached nodes. The complete shared memory space is available to all nodes, and can be used for process variables, various commands, and, with a higher level protocol, even transfer of larger blocks of data. It will be up to the chosen network system to manage access control and similar.

Another important aspect is the need for an easy-to-use interface between the network nodes and applications on a PC. The minimum requirement is that the PC interface needs to provide direct access to the full shared memory block of the system. It will be up to the specific Node-on-Chip implementation to provide options for this, as it will more or less boil down to which interfaces (e.g. USB- or Ethernet-enabled components) are made available to the FPGA.

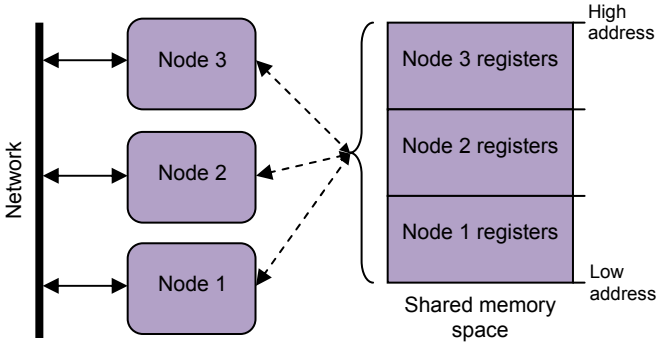

Fig. 3. A conceptual diagram of the shared memory model employed in the Node-on-Chip architecture. All nodes operate on the same memory space, through the network. Each node is assigned a memory space for its own registers, and has read access to the registers of the other nodes.

V. Node-on-Chip frameworks

We have created two implementations of the Node-on-Chip concept. These implementations, or frameworks, each provide a standard FPGA module, a system network implementation, and various applications and components to help interface to a standard PC.

The two created frameworks are named after the networks they use, and are called the TosNet framework and the µTosNet framework. When the frameworks are referenced, the full name (e.g. "The TosNet framework") will be used, while the network name for itself, or appended with "network", (e.g. "The TosNet network" or just "TosNet") will always reference the network itself.

The reason for choosing these networks over other, more established networks such as CAN-bus, Ethernet Powerlink and similar is due to TosNet being focused on ease of use, a small footprint (both physical and in the FPGA), and with a completely FPGA-based implementation of the protocol stack. Also, TosNet directly implements the shared memory model specified in the architecture, and can thus be used without further modification. Other networks would need to be implemented as FPGA gateware, with an extra layer of abstraction implementing the shared memory model.

We realize that having two different frameworks may seem slightly contradictory to the stated goals and context; however, the two frameworks are so similar, that components and applications created for one framework can be ported to the other with only a minimum of modification, and both will thus be able to use components from the same VHDL and circuit libraries. Being able to support multiple compatible frameworks, each with its own advantages, has proven to be an asset in our practical work. µTosNet is very attractive for small projects with short timeframes, e.g. small educational projects, whereas TosNet is better suited for more advanced projects with longer timeframes.

A. The TosNet framework

As implied by the name, the TosNet framework is built around a revised version of the TosNet network, developed in [6]. It provides a completely FPGA-based, isochronous protocol, operating at 10 Mbps, with support for up to 15 nodes connected in a ring. It directly implements the shared memory model of the architecture, by giving each node a local copy of the memory block, and can keep these updated with cycle-frequencies typically in the area of 4-10 kHz (depending on the number of attached nodes and allocated registers). The physical layer uses optical Toslink components, providing a cheap, noise-immune, and easy-to-use interconnect. The revised version features a redesigned physical layer, addressing some of the problems discovered with the original version described in [6].

The standard FPGA module for this framework is based on a Xilinx Spartan3AN 400 kilo-gate FPGA, which provides plenty of space for both the implementation of the TosNet network (uses about 30% of the available resources), and for device-specific hardware interfaces. It is mounted on a small board (45 mm x 49 mm) and equipped with I/O connectors on both top and bottom, thus making it stackable. In addition to this, two different standard expansion modules for the top connectors, using the same form factor as the FPGA module, are also available. The simpler of the two top modules carries a USB-to-UART converter, while the other, more advanced top module, carries an Ethernet-enabled, and user-programmable microcontroller unit (the ARM-9 based Digi Connect ME 9210), connected to the FPGA through an SPI interface. Both also provide the necessary Toslink components and some pin-headers for easily accessible I/O.

The top modules can be used to create so-called gateway nodes, allowing a PC to communicate with nodes on the TosNet network over a serial COM port (through USB) or Ethernet, respectively. ASCII and binary-based generic

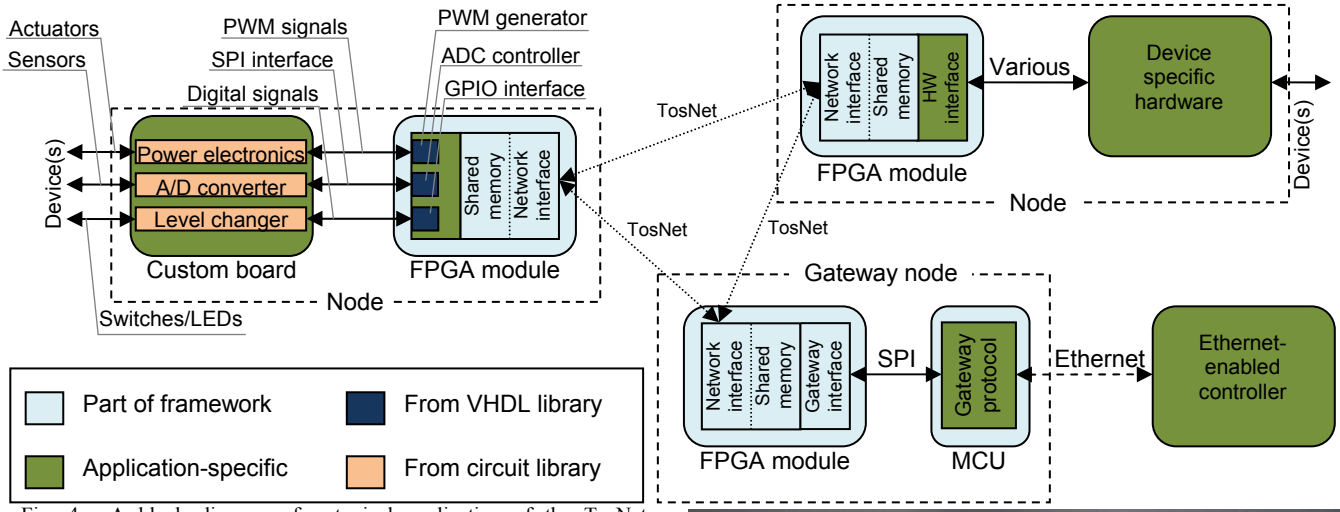

Fig. 4. A block diagram of a typical application of the TosNet framework. For simplicity, the FPGA modules only show the FPGA gateware parts, whereas the static support electronics are implied.

protocols are available for this, making communication with high-level software very simple. In the case of the Ethernet unit, creating an application specific gateway protocol is possible, as this can rather easily be programmed into the microcontroller in the C or C++ languages. A PC-mountable PCI Express based gateway (based on the Xilinx Spartan3PCIe Starter kit) is also available, allowing direct access to the shared memory block of the gateway node, through a simple memory array in user-space applications.

A block diagram of an example application of the TosNet framework can be seen in fig. 4, and the FPGA module and the two top modules can be seen in fig. 5.

B. *The µTosNet framework*

Inspired by the ease with which the TosNet framework allows communication between low-level hardware devices and high-level PC applications, the µTosNet network was created. It is meant to be used in simple, cost-critical applications that do not need to be distributed across several nodes, and which have rather basic requirements with regard to FPGA resources. This includes being used in introductory FPGA courses, to provide students with an easy-to-use PC interface for their hardware designs.

The standard FPGA module provided in this framework uses a Xilinx Spartan3AN 50 kilo-gate FPGA, which can use simpler PCBs and is much cheaper than the 400 kilo-gate version, but lacks the necessary resources to hold the full TosNet network implementation. The board is aimed at introductory courses, and thus has a pin-header allowing it to be mounted directly in a bread-board, which can dramatically speed up the creation of simple experimental setups. An add-on board provides access to an optional USB-to-UART converter and a Digi Connect ME 9210.

Inside the FPGA, the µTosNet network provides a reduced, single-node shared memory block, a simplified version of the full TosNet memory interface to this, and either a UART (converted to USB through the USB-to-

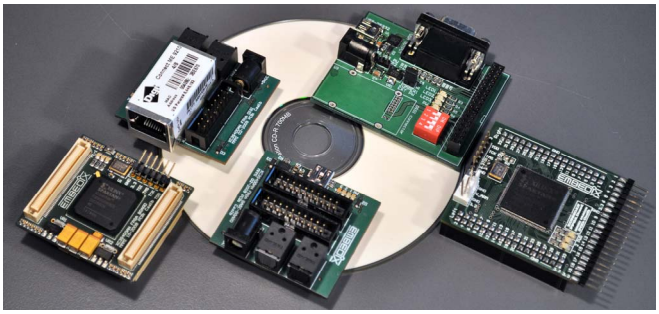

Fig. 5. The hardware modules created for the two frameworks, shown alongside a standard Compact Disc for size reference. The three left-most modules are from the TosNet framework, whereas the two boards to the right are from the µTosNet framework.

UART converter) or SPI (converted to Ethernet through the Digi Connect ME 9120) interface, to use for communicating with an attached PC. The framework only supports a single connection from the board to the PC, and it is thus not directly possible to interconnect multiple nodes. However, several nodes can be connected to and controlled from the same PC if necessary. The simplified memory interface shields new users from some of the more advanced features of the full TosNet, such as double buffering of the shared memory, while still making it easy to transition FPGA components between the two frameworks.

The FPGA module and the add-on board (without the Digi Connect ME 9210 unit though) can be seen in fig. 5.

VI. Results

As it is hard to formalize concepts such as ease-of-use and flexibility, we have employed an informal and pragmatic approach in the evaluation of these important aspects. This has been done so far by focusing on using the Node-on-Chip concept and its associated frameworks for as many projects as possible in order to gather experience and get inputs from colleagues and students outside the embedded world. We are employing a four-phased approach to achieve this:

1) Internal development and evaluation: Demonstration and test within our own group, by using and reinstrumenting old robotic systems out of circulation.

2) Evaluation with fellow non-embedded researchers: Using our framework in research projects, discussing the results with a broader group of robotic researchers.
3) Integration into courses and student projects: By teaching our robotics students to use the architecture and frameworks in their work.
4) Making the technology available to others: By utilizing the FPGA-focused, online, open-source site OpenCores [14], we hope to broaden the use of and contributions to our Node-on-Chip concept and frameworks.

The MiniVGT [6] and CATO [10], [11] robots fall under the internal evaluation category, and mainly served as initial tests and demonstrations of both the TosNet network, and, in retrospect, the application of the Node-on-Chip concept. Both robots are interfaced to Microsoft Robotics Developer Studio, to provide a very high-level environment for controlling them. For CATO, it was also possible to directly reuse several VHDL and software modules from the MiniVGT, showing a potential of fulfilling our goals regarding reduced development time and increased reuse.

The RoboTrainer [12] is a clear example of the ease with which we can now instrument robotic systems, and thus also a demonstration of our goal regarding ease of use. Using µTosNet, all the custom electronics present in the setup (quadrature encoder, motor driver and A/D converter) were interfaced to a small PC command-line utility in about half a day. Similarly, a software engineer from our institute, previously unfamiliar with both concept and frameworks, was able to create a high-level application with a graphical userinterface for the setup in a matter of hours.

We are currently evaluating another interdisciplinary project, where the TosNet framework has been interfaced to a Panasonic Scara robot, various tool-devices and the RobWork software framework [13]. The initial impression is rather close to what is shown in fig. 4: framework contents were easily and quickly reused from previous projects, allowing us to focus on the implementation of the FPGA interface to the robot and the RobWork software.

At the time of writing, we have just completed the first master level course incorporating the framework. As hoped, the students found it easy to use, and were generally impressed with how easy low-level hardware could be interfaced to PC applications. Detailed results and other aspects of using in particular µTosNet in teaching are described in [15]. Furthermore, we expect 5-10 students to use the framework during their master thesis projects in 2010.

We have just begun the fourth phase, and are currently in the process of deploying the frameworks to OpenCores. Initially, µTosNet will be deployed to get to know the online environment, whereas TosNet will follow later.

VII. Conclusion

Our FPGA-based Node-on-Chip architecture and frameworks for distributed control have demonstrated many advantages in relation to experimental robotics. The goals of the work have been to reduce the time spent creating hardware- and network interfaces, increase the amount of reuse between projects, and ease the use of low-level hardware systems. To some extent all these goals have been met by the TosNet and µTosNet frameworks, and we are hoping to see further evidence of this as the technologies are employed in future projects.

Apart from increasing the size of our VHDL and circuit component libraries, planned extensions currently include:
1) New FPGA modules using Xilinx Spartan6 devices.
2) A further layer of abstraction on top of the TosNet network, providing better support for easy configuration from high-level applications.

Acknowledgment

The authors thank Carsten Albertsen, University of Southern Denmark, for help with board design and creation.

References

[1] A. S. Soerensen, "Modular control of industrial mechanics," Ph.D. dissertation, The Maersk McKinney Moeller Institute for Production Technology, University of Southern Denmark, Odense, 2003.
[2] A. S. Soerensen, H. G. Petersen, O. G. Jakobsen and N. J. Jacobsen, "A development of parallel robotic modules for long reach applications," in *Proc. of the 32nd International Symposium on Robotics*, Seoul, 2001.
[3] A. S. Soerensen and H. G. Petersen, "A development of modular robots for flexible robotics manufacturing units," in *Proc. of the 33rd International Symposium on Robotics*, Stockholm, 2002.
[4] A. S. Soerensen, *et al.*, "Implementation of a practical reconfigurable manipulator system based on hybrid parallel and sequential elements," in *Proc. International Conference on Intelligent Manipulation and Grasping*, Genova, 2004, pp. 404-409.
[5] Z. Salcic, "PROTOS – A microcontroller/FPGA-based prototyping system for embedded applications," *Microprocessors and Microsystems*, vol. 21, pp. 249-256, 1997.
[6] S. Falsig and A. S. Soerensen, "TosNet: An easy-to-use, real-time communications protocol for modular, distributed robot controllers," in *Proc. 2009 2. International Conference on Robot Communication and Coordination*, Odense, 2009, pp. 1–6.
[7] J. Portilla, A. de Castro, E. de la Torre, T. Riesgo, "A modular architecture for nodes in wireless sensor networks," *J. Universal Comp. Sci.*, vol. 12, pp. 328-339, March 2006.
[8] J. Portilla, A. de Castro, A. Abril, T. Riesgo, "Rapid prototyping for multi-application sensor networking," *SPIE Optoelectronics and optical communications*, [Online]. Available: http://spie.org/x17547.xml?ArticleID=x17547
[9] Actel (2010). Actel Fusion Mixed Signal FPGA [Online]. Available: http://www.actel.com/products/fusion
[10] S. Falsig (2009, October 2nd). Cato [Online]. Available: http://robolab.tek.sdu.dk/mediawiki/index.php/CATO
[11] A. S. Soerensen, "Practical AGV construction and sensor-integration," Master thesis, The Maersk McKinney Moeller Institute for Production Technology, University of Southern Denmark, Odense, 1999.
[12] S. Falsig (2010, February 24th). RoboTrainer [Online]. Available: http://robolab.tek.sdu.dk/mediawiki/index.php/RoboTrainer
[13] The RobWork Team (2010). RobWork.org [Online]. Available: http://www.robwork.org
[14] The OpenCores Team (2010). OpenCores.org [Online]. Available: http://www.opencores.org
[15] A. S. Soerensen and S. Falsig, "A system on chip approach to enhanced learning in interdisciplinary robotics," to appear in *Proc. of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, 2010.