A System on Chip approach to enhanced learning in interdisciplinary robotics

Anders Stengaard Sørensen

Simon Falsig

***Abstract*— To sustain advanced interdisciplinary teaching and learning in the rapidly growing and diversifying field of robotics, we have successfully employed FPGA based System on Chip (SoC) technology to provide abstraction between high level software and low level I/O- and control hardware. Our approach is to provides students with a simple FPGA based framework for hardware access, and hardware I/O development, which is independent of computer platform and programming language, and enable the students to add to, or change I/O hardware in accordance with their skills. We have tested the framework in an embedded systems course and various student projects, and have found that it greatly enhance the students abilities to control hardware from software, and dramatically reduce the time spent on software ↔ hardware interfacing. As the framework is also scalable, it can support projects from controlling a single LED, to complex modular and aggregated robots with demands for high bandwidths and low jitter in the control loop.**

I. INTRODUCTION

When we view "robotics" as the art of using computers to interact with the physical world through sensors and actuators, the field is obviously highly concerned with interdisciplinary connections between the scientific and engineering fields of:

1) The physical processes we want to interact with
2) The relevant mathematics and computer sciences allowing us to develop algorithms and programs for the application
3) The interface between 1) and 2) — typically low-level software, embedded computers, I/O electronics, and signal conditioning.

Naturally, our students specialize in certain established fields like mechanical-, electronic-, and software-engineering, but by applying problem based and project oriented learning, with robotics as the main theme, we encourage the students to apply everything we teach to robotics, and hence lay a good foundation for interdisciplinary cooperation and projects along their study path and later careers.

A. Inhibitors

One of the major inhibitors we encounter in interdisciplinary learning, is the increasing complexity of the individual fields as the students progress, illustrated in figure 1.

A. S. Sørensen is with the Faculty of Engineering, the Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark anders-s@stengaard.net

S. Falsig is with the Faculty of Engineering, the Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark sifa@mmm.sdu.dk

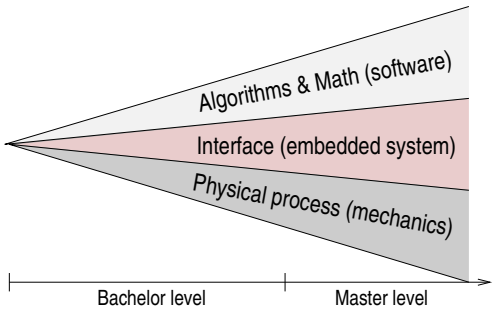


Fig. 1. Complexity progression in robotics educations

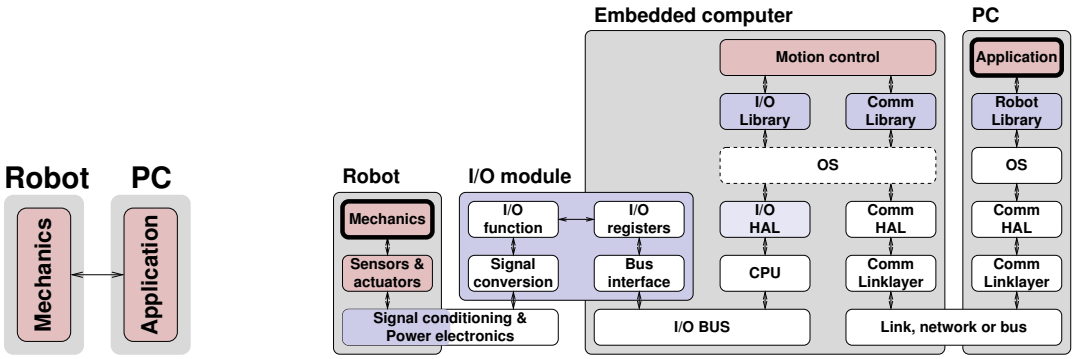
Early in the education, the courses are quite similar across different engineering fields, but as the educations progress, students have to specialize within certain subsets, defined by the educational profiles available, creating divergence and the well known dilemma between specialization and generalization. As our educations are research based, there is an underlying bias toward specialization, reducing the tolerance for the complexities involved in interdisciplinary projects at higher levels in the education, as these complexities are not relevant from an academic point of view, but merely represents unwelcome hassle.

B. The SW ↔ HW gap

One of the major contributors to unwanted complexity is the necessary interface between the physical world and the algorithms we want to control it. In junior projects this complexity can easily be hidden inside standardised equipment like LEGO-mindstorms, LabView or other commercially available products. But in senior projects, it is common to encounter situations where such solutions no longer live up to demands for performance, interfaces, programming methodology, size, power consumption etc.

As students — or professors, interested in applying algorithms to the physical world are hardly also interested in the embedded systems domain of electronics, low-level programming and computer architecture; the gap of interfacing between algorithms and the physical world become a serious inhibitor, unless the complexity is well hidden.

Imagine a simple system, where the position of a motor has to be controlled by high-level software (Application). The high-level software reside on a PC, and the motor is controlled by an embedded computer, connected to the PC through a field-bus. Conceptually, we imagine that the application software just commands the mechanics, as shown in Fig. 2(a), but as shown in Fig. 2(b), the details are somewhat



(a) Conceptual path

(b) Typical path

Fig. 2. The complexity imposed by the electronic/computer platform

more involved. The structure of figure 2(b) is typical of most current interface technologies, and is consistent with acknowledged commercial and academic interface systems like PMAC GENERIS [1] LEGO mindstorms and LabView.

The path from application SW to robot mechanics passes through numerous layers of hardware and software that provide abstraction from that hardware, like operating systems, libraries, and motion control. In Fig 2, the red (dark) boxes indicate the parts that typically change from project to project due to the intended application — the components we like to focus on. The white boxes indicate parts that are usually hidden inside the chosen platform technology, and does not need to be understood by individual developers. The blue (light) boxes indicate components it is often necessary to change, adapt or create to achieve the application goals. On top of the complexity of the embedded system itself, comes the programming languages and programming paradigms that impose themselves when developing software for the PC application, and the embedded software. Typically an object oriented paradigm, implemented in Java, C# or C++ is used for high-level programs, while the Hardware Abstraction Layer (HAL) parts are usually developed from a procedural programming paradigm, using plain C or C++.

An additional source of complexity, is the issue of the signal delays that are inherent in all CPU based systems, as the CPU has to share it's attention sequentially between the various software components. Junior projects can be designed so this factor can be ignored, but in advanced projects it creates a whole new dimension of complexity, in terms of real-time software architectures or multiprocessor solutions that ensure that latency and jitter can be bounded.

C. Hiding interfacing complexity

The universal way to hide complexity, is for the designers of the embedded system to make generalizations and compromises in accordance to the intended application area, and present the resulting functionality through a number of well documented interfaces. This approach operate on a scale, from a complete "black box", where everything inside is hidden and unchangeable, to a complete "white box", where the user is empowered to change everything inside the embedded system — provided he has the knowledge and experience to do so. For example, most commercial robot arms come with "black box" controllers, that allow you to control the arm from a C++ program under a certain operating system, but not to change or add anything inside the robot controller, or to write applications under a different operating system. The usability is high, but flexibility is low. Contrary to this, many educational robots come with complete white box interfaces, based on fully documented computer platforms, equipped with open-source software. Anyone who has the proper skill can change anything he wants. Here flexibility is high, but usability is low if you are not an expert in embedded systems technology.

In our experience, a claim that:

$$\text{Flexibility} \times \text{Usability} \times \text{Performance} = \text{Constant}$$

Would be a reasonable summary of the dynamics involved in using software to hide interface complexity, as descibed above.

D. Flexible hardware

One of the core reasons for system complexity and performance limitations, is the inflexibility of traditional electronics, that require all flexibility and abstraction to be created by software. With the advent and maturity of reconfigurable hardware like Field Programmable Gate Arrays (FPGA's) it is possible to lower the overall complexity of a system, by using reconfigurable hardware to provide flexibility and performance combined.

An FPGA is an integrated circuit, containing an array of thousands to millions of "logic cells" that each perform a very simple, configurable digital function — like a gate or flip-flop. The logic cells are connected internally by a massive array of crisscrossing wires with configurable connections. By configuring and combining a number of cells, more complex digital functions like registers, counters or state machines are realised. An FPGA can thus be configured into any digital function imaginable, from AND-gate to microprocessor. Large FPGA's can easily contain all the components of a standard computer system, and thus,

a completely flexible embedded computer can be created within a single IC — much like having your own IC factory.

The configuration of individual logic cells and connections are derived by synthesizing software, from a description of the desired functionality. This description is written in a hardware description language like VHDL, which is syntactically similar to common programming languages. The development process is very similar to software development, involving a source-file hierarchy, that is processed into a binary file that can be downloaded into the FPGA to configure it. No universal name for the configuration data has yet evolved, but we chose to refer to it as *gateware*.

Designing a system around an FPGA gives a whole new approach to flexibility as we can now create the hardware platform exactly as we like it, and thus reduce the need for software to make up for inadequate or incompatible hardware. A very important observation, is that a variable number of information processing units (state machines) can exist and operate in parallel, each with bandwidths exceeding a 100 million operations per second, in a 10€ FPGA, allowing even simple FPGA's to outperform microcontrollers in terms of I/O flexibility and performance.

E. Previous work

Convinced that the flexibility of FPGA's can upset the traditional balance between flexibility and abstraction, we have created a FPGA based framework for interfacing software to robots.

The framework has evolved over many stages, starting with FPGA based flexible I/O, as part of a generic embedded control node [2], [3] and [4]. This work was concerned with distributed network nodes, where the FPGA was used to implement a flexible I/O interface that allowed the node to assimilate common robotics mechanisms into a common control network.

We have then integrated the whole distributed node into the FPGA, and created our own fiber optical network to connect the nodes into a closely coupled system intended for rapid systems prototyping in research projects[5]. This resulting framework, which we refer to as TosNet, has been tested with a number of master thesis projects, which encouraged us to make a simplified version of the framework (μTosNet) for smaller student projects and teaching. [6]

Others have reported on the learning benefits of FPGA's for rapid prototyping [7], but we have not yet encountered other teaching efforts with an integrated communications framework.

II. THE FRAMEWORK AND IT'S COMPONENTS

TosNet is designed to allow continuous time data to be exchanged transparently between a number of distributed I/O nodes and one or more computers controlling the system using the distributed I/O, as shown in Fig. 3. Basing it on an FPGA allow most of the abstraction to be done in gateware, simplifying the software and the entire system. The following is a short introduction. A full description of the frameworks can be found in [5] and [6]. If you would like to use TosNet, we have made it available as free gateware on the FPGA and SoC community Opencores.org

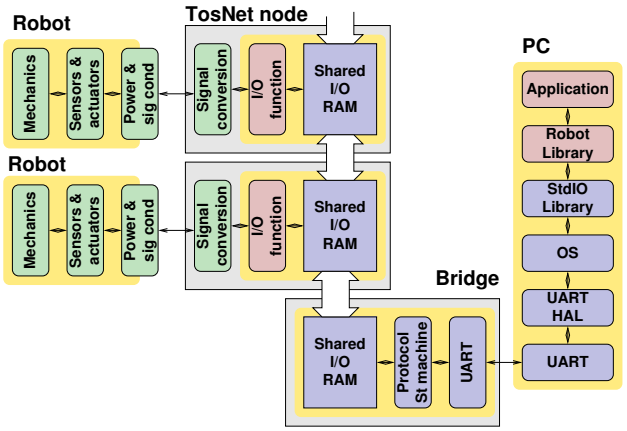


Fig. 3. The structure of TosNet

A. Overall

The TosNet framework is a set of interconnected nodes, where each node contain the following gateware modules specified in VHDL:

- A network interface binding it together with the other nodes through a ring architecture network with point to point optical connections.
- A RAM, that is mirrored across the network. All nodes have a designated area of the RAM that only they can modify, but all nodes can read the entire content. The RAM thus has the role of global distributed I/O register.
- A number of I/O components, that interact with a subset of the RAM registers, to perform their I/O function, in cooperation with necessary (analog) circuitry outside the FPGA.

B. Connection to host computer

In order to connect a TosNet system to a computer, one of the nodes has to take on the role of "bridge" between TosNet, and whatever means of communication used by the Computer. The interface toward the computer can be exchanged without changes to the rest of the system. We have developed bridges for PCI-express, Ethernet, and for asynchronous serial communication via a USB UART or RS-232.

C. Shared memory

The heart of the framework is the shared memory that implement the distributed I/O registers. We chose this model, as it represents a simple and understandable abstraction of continuous-time signals, as global variables. If TosNet is interfaced to a computer with a memory bus like PCI or PCI-express, the variables will simply be directly available in the computers address space, to be used in any programming language allowing direct memory access. Using an UART or Ethernet bridge to TosNet, require the memory registers to be accessed through a protocol. For practical reasons, the shared

memory is organized as 32-bit wide, but there is no need for a 1:1 mapping between memory cells, and I/O functions, so it is easy to write the I/O functions in a way that utilize both shorter and longer widths.

D. I/O functions

As common FPGA's can only implement digital logic, not analog circuitry, the I/O functions must be divided between the digital logic that reside in the FPGA, and the analog electronics that reside outside the FPGA. The analog electronics needed for most applications is quite simple, so the problem can be solved either by having a selection of standard boards for experiments and teaching, or by the user designing a "signal conditioning circuit" specifically for the application at hand.

Over time, we have accumulated a small library of commonly used I/O functions, so we can quickly configure a system for a specific combination of I/O functions. Each entry in the library is the VHDL source files for the digital I/O, along with design files for the matching external analog circuitry.

E. Performance

The system performance is defined by the 10Mbps TosLink transceivers used to implement the distribution of the RAM, and the number of exchanged variables. In a system with a total of 64 32-bit variables shared, the cycle frequency will be $10MHz/(64 \times 32) \simeq 4.8kHz$. The cyclic nature of the network ensure the cycle frequency is jitter free. Enabling high performance control to take place across the system. The high cycle rates and absence of jitter exceed the performance demands of all our previous motion control projects.

F. The view as I/O designer

If you plan to develop a TosNet I/O function, your interface will be the address- and data-bus of the shared memory, along with signals to control the double buffering. Your job is then to develop a state-machine that performs the desired I/O function, exchanging relevant information with the RAM. This interface is completely invariant to what other nodes are on the network, and which I/O functions and bridges they contain.

G. The view as application programmer

If you plan to write application programs using TosNet, you first need to pick the bridge between the host computer and the TosNet.

a) UART: Interfacing to TosNet through a UART bridge is very easy, as we use a simple clear text protocol. Commands and responses can literally be typed and viewed in a terminal program like Putty, or be handled by string oriented I/O functions that can access the serial port.

b) Ethernet: Using Ethernet is equally easy, you open a channel to a socket on the TosNet bridge, and then issue commands using a simple protocol based on UDP datagrams. This mode is available to all programming languages that support socket communication over Ethernet.

c) PCI express: Using PCI-express gives the tightest integration between the application program and the real world application. Unfortunately it is more complicated than the above methods. You must first install the appropriate device driver between the PCI-express FPGA board and your operating system, in order to map the TosNet address space into the computers address space. We have drivers for certain versions of Linux and Windows. Once that is accomplished, you can directly read and write the content of the shared memory in your program, provided your programming language has this ability. If using a language without direct memory access, like Java, you must provide a software component that can interface to the application.

H. μTosNet

In order to reduce cost we have created a reduced version for single-node systems, where the network for distributing the RAM has been replaced by a UART bridge, so the entire system reside on a single node. The RAM has a smaller address space, but is organized the same way as the full TosNet, so the I/O functions developed for TosNet can be used with μTosNet and vice-versa.

III. TEACHING REQUIREMENTS

Our ability to develop TosNet for teaching was based on the fact, that we we already had robotics, FPGA's and software on the curriculum, so it was very easy for most of our students to put these aspects together, but the students do not need to be FPGA or VHDL experts to begin with, as they can start with these skills at a relatively low level. Below we discuss some of the more practical requirements for problem based learning in this field.

A. FPGA boards

The TosNet framework was developed on the Xilinx Spartan-3 family of FPGA's, and can be used with all evaluation/demonstration/development-boards containing a Spartan-3. We have also successfully used TosNet with the Xilinx Spartan-6 family. If you want to use the UART bridge, your board should have an RS232 interface, or a UART to USB bridge. We use Toshiba TosLink optical transceivers and cables for Inter-node connections in the full TosNet system, but any tranceiver able to transfer a digital signal at 10Mbps will suffice. The inter node connection require 1 input and 1 output on each FPGA board.

For our introductory course, we have used the smallest possible 50k Spartan-3 FPGA (XC3S50AN) on the simplest possible board, shown in Fig. 4, that allow the students to use the FPGA in a breadboard. This somewhat unorthodox approach has the pedagogical advantage of demystifying the FPGA, forcing the students to design signal conditioning circuits for the FPGA, as there is no I/O device on the FPGA board except two LED's. The board is also so cheap $\simeq 20$€ and simple to assemble, that we can allow eager students to make and take home their own boards, for increased learning in their spare time.

B. Development software

As we are using Xilinx FPGA's, we also rely on Xilinx development tool: "ISE". Which is available for Linux and Windows on commercial and educational license, as well as a reduced license available for free download (ISE WebPack). The TosNet framework itself is compatible with the free WebPack, and there is no need for a full version of ISE.

C. Application programming

As the 50k FPGA is too small for the full TosNet, we have used μTosNet throughout the introductory course, so the only requirement to the application software platform is that it can read and write to a serial port. If the students are reasonably experienced with application programming, they can easily cope, but if they are inexperienced, a beginners programming language and a knowledgeable teacher would be beneficial.

D. I/O circuitry

In our course, one of the objectives was to create simple I/O circuitry, that would enable the digital FPGA to interact with the analog world. This require both students and teacher to be experienced with, and interested in, basic analog and digital electronic design, which is not always the case.

As our students have limited experience with analog and mixed-signal electronics, we have found it very helpful to start out with the *breadboard* compatible FPGA board in figure 4, which allow inexperienced students a fast way to combine the strictly digital domain of the FPGA with simple filters, signal converters, sensors and actuators.

Alternatively, if a typical FPGA demo/evaluation-board is used, it will contain different I/O elements which can then be integrated with TosNet. Some evaluation boards are even modular, with the option of buying or building different I/O functions on daughter boards.

IV. THE ROBOTRONIX COURSE

We have tested our framework on 30 students attending the introductory course in embedded system on the masters of engineering education in robotics. The course, referred to as "Robotronix" is 5 ECTS ($1/6$ semester), and run over 6 weeks, with two blocks of 4×45 minutes per week. One Tuesday, one Wednesday. 1 additional week is available for unsupervised project work in the end.

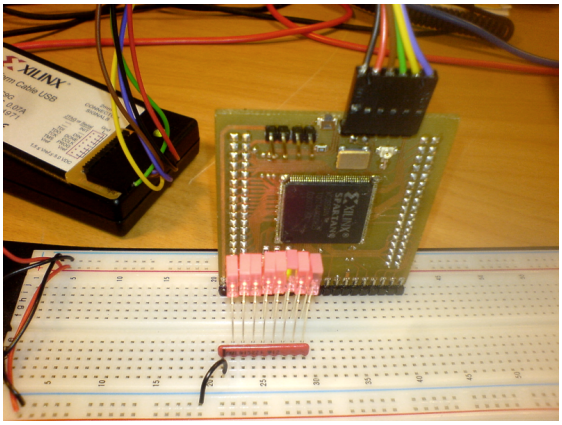


Fig. 4. The FPGA board used for introduction - mounted in a breadboard where 8 LED's and a resistor array have been added

The course is organized with Tuesdays in class, and Wednesday in the lab. Tuesdays we cover necessary theory for the coming lab exercise, Wednesday we spend in the lab. The lab report must be submitted during the weekend. The students peer review the lab reports before Tuesday, where we evaluate the reports on class, and go on to theory on the next lab exercise.

A. First 3 weeks

As we recruit students with a wide range of bachelors degrees, from a wide range of countries, with varying traditions. We spend some time on boosting/refreshing basic electronics skills. Generally, our students are quite strong in math and software engineering, but have limited knowledge of electronics and computer architecture. Before we get to working with FPGA's, we spend 3 weeks working with oscillators, flip-flops and sequential logic. These topics allow the inexperienced a chance to get up to speed with lab work, while the experienced get a chance to absorb themselves in a advanced electronics topics outside the mandatory scope. During these 3 weeks, we also offer a parallel voluntary 2×4 lesson crash-course in VHDL and FPGA's, as these topics are not mandatory with the foreign bachelors we accept for the course, and some have never worked with them before.

B. In the 4. week

The students are given the FPGA board, along with a users guide, containing examples of simple I/O related functions in VHDL.

We ask the students to implement the examples, and to change them to suit different design criterion. The culmination of this, is to build a control system for temperature control using: 1): A 12V fan controlled through a transistor by pulse width modulation (PWM). 2) A NTC thermistor and a capacitor, configured as a low-pass filter, which time-constant is read using a square generator and a timer implemented in the FPGA. 3): A heat source placed near the thermistor. Although simple, this exercise give the students a good sense of the potential for I/O applications.

C. In the 5. week

The students are given the source code for μTosNet, along with an undocumented application example, implementing some simple timers, PWM generators, and digital inputs controlled by μTosNet. First we ask them to analyze the source code and document the I/O functions. Then we ask them to merge some of the I/O examples from the 4. week into μTosNet, in order to familiarize them with writing TosNet I/O functions. The culmination of this, is the ability to read temperature and control the speed of a fan, from an application program on their laptops. At this point, the students became quite excited, as they realized how easy it

was to access the physical world from a high level application program.

D. The final project

The students have two weeks available for a project, and are asked to select a more advanced I/O application to implement in μTosNet, combined with an application program. In order to demonstrate the ease and flexibility of using the framework, we suggested some rather different I/O interfaces, as described in the following section.

E. Course conclusions

Although there was only two weeks at part time available — giving an estimated 5 to 7 full days for the students to complete the final project, the students were able to arrive at quite impressive results.

- Two groups successfully implemented the digital part of 3-phase AC inverters for controlling AC servomotors. Demonstrating open loop control of both syncronous and asyncronous AC motors (at reduced voltages for safety reasons), using a PC program with GUI as user interface.
- Two groups had implemented RC servomotor controllers, also with their own GUI PC application, allowing them to control a Lynx 5DOF demo robot arm, and a RoboNova 18DOF android using sliders in their GUI.
- One group had created both an I/O interface for the PlayStation control pad, and a VGA interface displaying the Pong video-game. Running the Pong game-play in a PC application, they had created a powerful demonstration of a real-time feedback system over μTosNet.
- One group had interfaced to a MIDI controller featuring 16 sliders, allowing musicians to control synthesizer parameters over the MIDI network. Using the FPGA to intercept the MIDI commands, they could display the slider positions in a GUI on the PC. They had also implemented a VGA controller on another FPGA, and using a separate USB cable, the application could display the slider position through the second FPGA. A video of this is available at [8]
- One group interfaced the DC motor of the robot gripper, and were able to control it from a PC application. Additionally they also integrated a SONAR rangefinder with the system, becoming able to measure the distance from gripper to the nearest surface.

As the circumstances of teaching changes from semester to semester, it is impossible to draw definite conclusions from a single course, especially as 7 weeks is a short time for evaluating the learning effect of a new approach. Some differences from previous micro-controller based courses are however so significant that we feel safe in concluding the following:

- TosNet has made it feasible to replace CPU/micro-controller based interfaces with FPGA based in the course.
- The students have been able to utilize the flexibility and performance of the FPGA, to create combinations of I/O interfaces that exceed the abilities of normal micro-controllers.
- The students have been able to write high-level application programs in JAVA and C# that interact with their I/O applications through μTosNet.
- The TosNet/FPGA approach have ensured that the students have avoided the conventional timing- and resource sharing challenges of CPU based interfacing.
- Considering the two week (part time) time-frame for the final project, the students have been able to come up with impressive and diverse applications.
- The unanimous feedback from the students is that they are impressed and pleased with the frameworks ability to offer them focus on the SW application and I/O interface, without dragging them into the particulars of communication electronics and protocols.
- Approximately $1/3$ of our students had no prior experience with FPGA's and VHDL. Our strategy of a 2×4 lesson crash-course made it possible for those students to participate reasonably well, but there is no doubt that more time should be allocated for learning VHDL.
- The exam revealed that some students had immersed themselves completely in the application software, and had been able to write high quality application programs for I/O functions implemented by other members of their team. This was contrary to the course description, but indicated that TosNet can be used by programmers without insight into the implementation of the I/O functions.

To summarize, the students have been able to let high-level software on a PC interact with various aspects of the physical world, at a relatively high quality, considering the available time-frame. It is our clear impression that this is due to the reduced hassle of moving information across the SW $\leftrightarrow$ HW gap, offered by the TosNet/FPGA framework.

V. OTHER STUDENT PROJECTS

After the Robotronix course, we have also used TosNet in a small number of other students projects. The projects have been started in the wake of our initial TosNet experiments, by simply letting students take over our demonstration platforms after we were finished with them. One of the projects is particularly interesting, as it was done by a single student as follow up on the Robotronix course.

A. The 18 DOF Walker

One of our students, Rolf Ugilt, wanted to work further with TosNet at the same time we were contemplating resurrecting an obsolete 18DOF walking demonstration robot. We thus defined an individual course, with the following goals:

- Interface a μTosNet node to the 18DOF walking robot.
- Write a C# high-level program, with a GUI that allow manual programming of the robot gait.

As the robot uses RC servomotors, controlled by simple PWM signals, the task is not very complicated. Even so, Rolf was able to complete the project from scratch, in approximately 30 hours of work, to a level where he can begin focusing on kinematics and gait control, without worrying about the embedded system [9]. In the project originally creating the walker robot, hundreds of hours were spend on the 68HC11 based micro-controller system interfacing a PC to the 18 servomotors [unpublished]

VI. CONCLUSIONS AND FUTURE WORKS

All our experience with the TosNet framework indicate that it provides a very transparent conduit between high level application software and I/O functions. As the major parts of the framework is implemented in reconfigurable hardware (gateware), there are no performance issues, or derived issues about resource sharing, timing, deadlocks etc. which is a major reason for the transparency of the abstraction our framework provides.

By utilizing popular communication technology like UART, USB and Ethernet as bridge between our framework and the application, we have bypassed many common problems of compatibility between I/O devices, operating systems and programming languages, and have arrived at a method which is completely platform and language independent, as our framework can be accessed by any application that can access a serial port or an Ethernet socket.

The price of this approach over memory mapping is a dependency on the operating system and programming language to handle the communication. We have no guarantees about the latency and jitter when we do not handle the information ourselves all the way.

On the other hand, when concerned with teaching, potential latency and jitter can be a small price to pay for the boost in efficiency and focus on the relevant subject matter.

A. Conclusions

It was evident that the μTosNet framework worked very well, and that it enabled the Robotronix students to focus on the subject matter of application programming and low-level I/O interfaces, without any distractions from intermediate layers.

This focus is a reasonable explanation for the impressive progress they made in the small final projects, and is a strong indication that our framework does indeed boost problem oriented learning in robotics.

B. Future Works

The preliminary success in both teaching and research projects, prompt us to continue development and dissemination of the TosNet Framework.

a) More projects: Is the way to generate more experience and ideas. We will start up a batch of master thesis projects using TosNet in Septermber 2010, and by making the Framework available at OpenCores.org, we hope others will use, comment and contribute to the framework.

b) The library of I/O functions: is still very informal and ad-hoc. We see a formalized and well kept library as a major way of saving students, researchers and technicians time, so we will increase our focus on this part.

c) The effect of the OS on latency and jitter: Is a very relevant issue, as it is very tempting to let the high level application handle closed loop feedback, at least in teaching and during experimental project phases. We have actually been surprised about the lack of reported problems with OS generated jitter. We have a suspicion, that hyper-threading and dual-core processors are responsible for reducing the OS generated jitter, and we would like to investigate this question further.

d) Real-time Ethernet: As we are working with the real-time Ethernet derivative "powerlink" in other projects, it will be very natural to create a powerlink based bridge to support easy real-time communication between the PC and the Bridge. Powerlink is also well suited as a node to node medium to keep the RAM syncronized, so we are considering Powerlink as an alternative to our own protocol.

VII. ACKNOWLEDGMENTS

The authors will like to thank our skilled and efficient technician Carsten Albertsen and the students of Robotronics 2009, for their invaluable help with developing and testing the framework.

REFERENCES

[1] Emilio Ruiz Morales, *GENERIS: the EC-JRC generalised software control system for industrial robots*, Industrial Robot: An International Journal, Vol. 26 Iss: 1, pp.26 - 32

[2] A. S. Sørensen, *Modular control of industrial mechanics*, Ph.D. dissertation, 2003, University of Southern Denmark http://www.stengaard.net/anders-s/Thesis/phd.pdf.gz

[3] A. S. Sørensen, O. G. Jakobsen, P. Favrholdt, H. G. Petersen,*Implementation of a practical reconfigurable manipulator system based on hybrid parallel and sequential elements*, 2004, Proceedings from the "Intelligent manipulation and grasping international conference" (IMG04), Genoa

[4] A. S. Sørensen, H. G. Petersen, *A development of modular robots for flexible robotic manufacturing units*, 2002, in: "Proceedings of the 33nd ISR (International Symposium on Robotics)", Stockholm

[5] S. Falsig, A. S. Sørensen *"TosNet" an Easy-to-use, real-time communication protocol for modular, distributed robot controllers"* , 2009, in "Proceedings from the 2. international Conference on Robot Communication and Coordination" Odense

[6] S. Falsig, A. S. Sørensen *An FPGA based approach to increased flexibility, modularity and integration of low level control in robotics research*, 2010, to appear in "2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2010)" Taipei

[7] K.C. Aw, S.Q. Xie, E. Haemmerle *A FPGA-based rapid prototyping approach for teaching of Mechatronics Engineering* Mechatronics Volume 17, Issue 8, October 2007, Pages 457-461

[8] M. Green, R. R. Christensen, T. Haastrup *Midi on TosNet* YouTube video, 2009 http://www.youtube.com/watch?v=PNwYi--gjuE

[9] Anders S. Sørensen, Simon Falsig, Rolf Ugilt, *A step toward plug and play robotics with SoC technology*, 2010, To appear in Proceedings of the 13. international conference on climbing and walking robots. video1: http://www.youtube.com/watch?v=Gr43OqVB7_4 video2: http://www.youtube.com/watch?v=JX0XHqHAySU