A step toward 'plug and play' robotics with SoC technology

Anders Stengaard Soerensen, Simon Falsig & Rolf Ugilt

The Maersk Mc-Kinney Moeller Institute, University of Southern Denmark, 5230 Odense M, Denmark

** E-mail: anders-s@stengaard.net, sifa@mmmi.sdu.dk, ugilten@gmail.com*

www.sdu.dk/mmmi

This article describes our progress toward simplifying and streamlining the low level systems integration of experimental robots, combining a System on Chip (SoC) approach with conventional modular approaches. The combined approach has increased flexibility, improved the embedded integration, and decreased the complexity of programming, compared to conventional modular approaches. We show the impact of the SoC approach in a simple demonstration and teaching model of a walking robot.

Keywords: System on Chip (SoC), Field Programmable Gate Array, I/O Interfacing, Low Level Systems Integration, Embedded Systems

1. Introduction

Lab models play a vital role in experimental robotics, and the quality, speed and cost of our research is often defined by our aptness for prototyping mechatronic systems. As the platform technologies of electronics, computers and software change rapidly, researchers are constantly placed in a dilemma between the latest technologies that can optimize performance, complexity, size, weight and flexibility, against older technologies that can provide re-use of hardware and software from previous work. When we include technology requests from external research partners, the result is often a Babylonian confusion, where major system parts have to be re-implemented in each new project, adding substantial cost in money and time.

As embedded systems technologies are rarely the object of robotics research, we often defer to clunky and well proven solutions, while wishing for a small, flexible, fast, stable, re-usable and cheap plug-and-play like technology that will allow us to integrate our algorithms with the mechanics, without all that tedious mucking about with micro-controllers, device drivers, endianness, electric noise, incompatible networks, etc.

2. System On Chip (SoC) technology

The primary way to provide flexible hardware architectures is through modularity. The CPU, memory, storage and I/O modules comprising an embedded computer system have shrunk over time, so in a few decades, we have gone from "system in a building" to "system in a rack" to "System on a board" to the current state "System On a Chip" (SoC) that denotes the situation where all the major components of a computer system is placed on the same integrated circuit chip.

Microcontrollers were the first instance of SoC's to become commercially available. They have revolutionized automation and robotics, by offering very small and cheap computers that can be physically embedded into almost anything. Their implementation on an IC is also their Achilles heel, as we can not exchange or update their internal modules when new projects introduce new demands. We either have to compromise performance and functionality to keep using well-known microcontrollers, or pay a heavy penalty of maintaining knowledge and (often incompatible) source-code for a wider range of microcontrollers.

A more modern technology is the Field Programmable Gate Array IC (FPGA), which offers the user an array of basic logic gates, with a network of configurable on-chip interconnections. By configuring the interconnections, any digital circuit can be realized within the limits of the number of basic gates. Modern FPGA's have millions of gates, and using development tools adapted from conventional chip-design, users can specify the intended functionality in a hardware description language, for example Verilog or VHDL,[1] which is similar to software languages. Using this technology, it is relatively simple for electronics designers to specify and implement an entire computer system with CPU, memory and I/O in an FPGA IC, with the benefit of being able to change and update the system inside the IC on the fly. The FPGA thus offers the ability to design and maintain your own "microcontroller", which can be tailored to the exact needs of a project. As the design languages are modular, popular components like CPU's, communication interfaces, etc. can be reused or bought as 'libraries' from third parties, and simply be 'linked' into a design, the same way a software component might be reused or bought.

3. TosNet: A node on Chip I/O architecture

As our work involves modular robots, we subscribe to a distributed architecture, based on embedded control 'nodes', where each 'node' is embedded

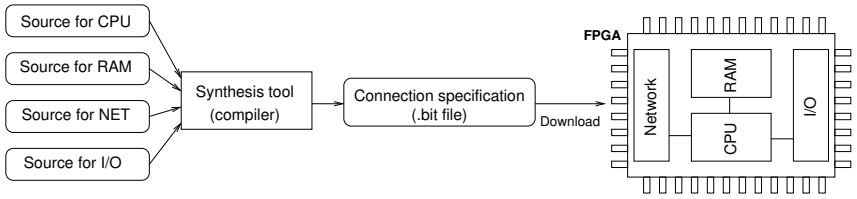

Fig. 1. From source to chip

into, and controls a mechanical module, e.g. a robot arm or mobile platform. When we participated in the EU project DockWelder,[2] these control nodes were based on a commercial network, a conventional CPU, and a small FPGA as a configurable and generic I/O system.[3] Following the progress of FPGA's we have advanced our distributed architecture into a framework, named TosNet, where all digital functions are embedded in the FPGA, and specified in the hardware description language VHDL.[4] The framework enables us to adapt the core functionality of our control nodes to any communication system and any I/O functionality needed in a specific project, with a minimum of effort.

The core of TosNet is a distributed memory, where relevant system states and I/O variables are shared between the nodes as distributed global variables. Physically, this memory is implemented as a dual-port RAM inside the FPGA of each node.

In each node, an I/O controller continuously copies the relevant variables between the memory cells and the external world, using the appropriate I/O interfaces. For instance, the value of an up/down counter that keeps track of a motor position is copied to the memory cell(s) assigned to hold information about that motors position. Simultaneously, the content of the memory cells assigned to hold information about the applied motor voltage is copied to the I/O component responsible for this function. More advanced I/O controllers implementing e.g. local feedback control and interpolation can also be created.

A communication component uses the other port of the RAM to synchronize the content of each modules memory across a network. As this is an independent controller, no resources are shared between the I/O and communication components, apart from the RAM itself, resolving most of the complexity experienced in CPU based systems. In the original TosNet, we use the optical 10 Mbps TosLink® physical layer to synchronize up to 15 nodes, with a completely isochronous rate from 20 kHz to 200 Hz, depending on the amount of nodes and shared variables.

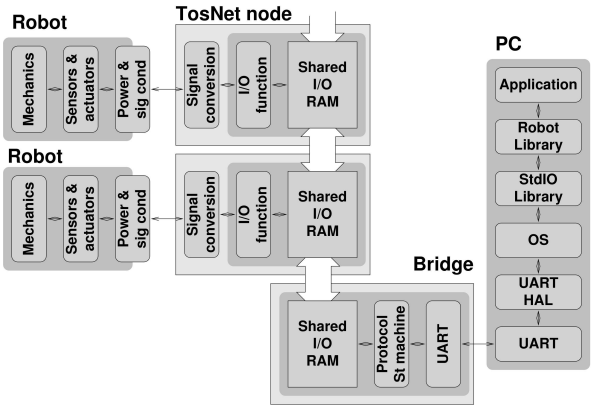

Fig. 2. Example TosNet architecture

Access to the TosNet distributed memory from e.g. a PC/workstation is done through a dedicated "master node" working as a bridge. Currently, we have bridges that communicate with the PC over PCI-Express, USB or Ethernet. For projects where a single node suffices, we have developed a reduced framework — called μTosNet, where the network is collapsed and removed, and the I/O functions are implemented directly in the "bridge" component. μ-TosNet can thus be implemented in the smallest and cheapest FPGA's, while having interfaces that are completely compatible with the full framework.

4. Development methodology

The TosNet "Node on Chip" framework has made integration of new mechatronic devices very easy. Basically the methodology is as follows:

(1) Implement/buy and integrate, the necessary power electronics and signal conditioning to connect the device to the digital I/O signals of the FPGA. (*This task is very similar to creating signal conditioning for a conventional computer/controller. FPGA's do not have analog I/O though, so digital ↔ analog conversion is a part of the signal conditioning.)*
(2) Re-use, adapt or create the specification for the digital part of the I/O in the VHDL language. (*By now, many I/O functions have been used in previous projects, and the VHDL specification can thus be reused from the library of previous work.)*
(3) Choose how to map I/O registers to memory addresses, and modify the template I/O controller VHDL specification accordingly.
(4) Include the source-code of the (μ)TosNet dual-port memory and communication interface in the VHDL project.
(5) Let the development tool synthesize (compile) the source files into a

configuration file.
(6) Download the configuration file to the FPGA.

Obviously, the primary work lies in step (1) and (2), and if e.g. local feedback control is required, also (3). The rest of the steps are simple routine.

When the configuration is tested and downloaded and the FPGA is connected to the network, the I/O functions implemented by the FPGA are on-line, and can be used.

Once the designer masters the development tools and VHDL programming, there are substantial benefits to this method:

- The I/O functionality can be tailored for optimal performance, without compromises due to performance, resolution etc.
- FPGA's can implement algorithms in hardware with a large potential for parallelisation, allowing very high performance data analysis of e.g. video, RADAR or SONAR signals.
- As the functionality is completely specified as source code in a specification language, the functionality can easily migrate to newer FPGA IC's as the technology progresses.
- The mechatronic designer can focus on the I/O interface and the necessary I/O components, and simply reuse the rest of the framework.
- High-level software using TosNet to access sensors and actuators can largely be re-used between various I/O applications. Only the register model needs to be changed, while the communication method remains constant.
- The option of using USB and Ethernet as a bridge to TosNet makes high-level software development independent of operating systems and programming languages.

5. Application example: 18 DOF walker

After evaluating the full TosNet in a number of distributed applications,[4] we wanted to evaluate μTosNet in students projects, and defined a master-thesis project, where a vintage 18DOF six legged walker platform, based on hobby servomotors, should be resurrected with μTosNet as a low-level interface. One of our 8. semester students, Rolf Ugilt, has begun working on this project, but as he is still taking courses, the work is primarily done in his spare time, until after the summer break of 2010.

Using the methodology described in section 4 the first step was to implement the platform specific electronics for making the FPGA control the servomotors. The signal conditioning to the motors is straight-forward as

the motors accept 3.3 V logic signals on their control inputs, and can thus be connected directly to the FPGA outputs. In order to get force-feedback from the motors, we include a current sensor for each motor, and read the currents one at a time, using a single A/D converter and a set of analog multiplexers connected to form a 18-1 analog multiplexer.

The μTosNet to PC bridge is based on a 115 kbps full duplex asyncronous link. The stream is transferred wireless, using an XBee module, and interfaced to the PC with a USB to serial interface. The serial stream is made available through the PC file system, as a "virtual com port" that can be opened, closed, read or written as an ordinary file. The protocol used to read/write the I/O memory block from the PC, is based on simple text commands. For instance, "`w01 000000ff`" will write the value 255 into address 1. This allows anyone to control the I/O by opening, and then writing and reading to a file.

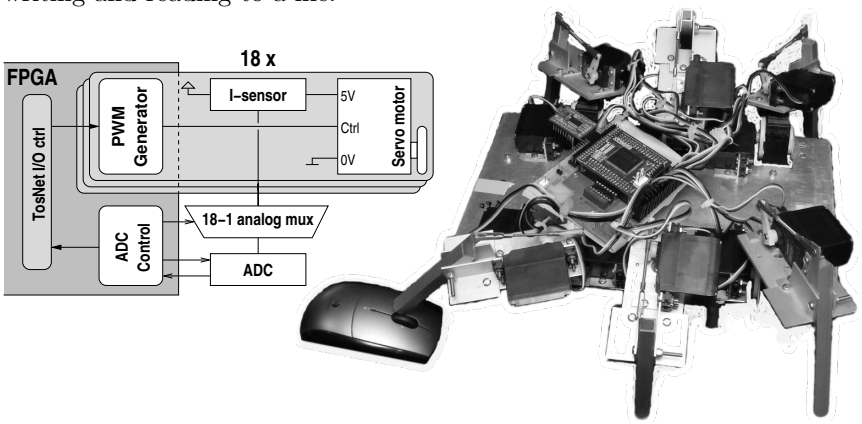

Fig. 3. Block diagram of the interface (left) for the six-legged walker (right)

To give an indication about this second step in the methodology, figure 4 shows a simplified VHDL module, which uses a 50 Mhz clock to create a $\simeq 50Hz$ square wave signal with a pulse-width that can be varied from 0 to $\simeq$ 2.6ms, controlled by the 8-bit "`magnitude`" input. Although simplified, the shown code can control the servomotors used, although including a few extra features will increase the accuracy and efficiency. The VHDL module is simply instantiated (included) 18 times in a higher layer of VHDL code, which also connects the PWM magnitude inputs to corresponding memory cells in the I/O RAM. The VHDL controlling the ADC is more complex, as the chosen ADC has a serial interface (SPI), and needs a small state machine to clock in data. Students with beginner or intermediate skill in VHDL can easily write the necessary modules though.

```
-- This simplified VHDL code illustrates how a square signal with a frequency
-- of 48 Hz, and a pulse width between 0 and 2.6 ms can be generated from an
-- 8-bit setpoint using a 50 MHz clock.
-- (The real code used is a little more detailed to allow better resolution.)

entity PWM_GEN is
port (
      clk_50m : in std_logic;
      magnitude : in std_logic_vector(7 downto 0);
      pwm : out std_logic );
end entity;

architecture Behavioral of PWM_GEN is
   signal counter : std_logic_vector(19 downto 0) := (others => '0');
begin
   process(clk_50m)
   begin
      if (clk_50m'event and clk_50M='1') then
         counter <= counter+1;
      end if;
   end process;

   pwm <= '1' when counter(16 downto 9) > magnitude and
                   counter(19 downto 17) = "000"
              else '0';
end Behavioral;
```

Fig. 4. Simplified VHDL for PWM generator

To integrate the control for the walker into μTosNet, I/O modules generating the 18 PWM signals and a module to control the ADC and MUX must be written in VHDL, and linked and 'compiled' with the predefined TosNet code. Then a PC application program can be written to test the system.

As the master project is still in its early stages, our student has only written a test program, where the user can control the position of each motor, using a graphical user interface with 18 "spinboxes".[5] As the I/O memory is organised in 32-bit words, each word represents four 8-bit position values. The test program, written in C#, basically loops through all the applied values in the GUI, converts them to an ASCII string, concatenates them four by four with the respective TosNet register address, and transmits them over USB/XBee using the protocol discussed above. Although the student was a C# novice, we chose this language to demonstrate compatibility with the current fashion in high-level level programming.

At the point of writing, the ADC functionality is not yet included in VHDL, so the estimated 31 work hours spend on the control part cover only the forward control of the servomotors. Based on previous experience with ADC applications, we estimate that the current feedback will be operational before the total project time exceeds 60 hours.

Student estimate of time spent on low level control	
Platform specific electronics	8 hours
VHDL — components & integration with framework	3 hours
Learning C#, and writing test code	20 hours

6. Conclusion

We have presented our TosNet framework for experimental robotics, and have reported its positive impact on a walking robot student project. Although the walker is rather simple, the example illustrates the reduction in complexity offered in a single-node system. The students time consumption of 31 hours is dramatically shorter than previous microcontroller based projects, where hundreds of hours have been spent on the low-level interfacing, before the walker platform was placed under PC control. Although other factors can influence the time consumption, we see the work presented here as a strong indication that our modular SoC approach to low-level systems integration has distinct advantages when it comes to rapid prototyping of experimental robots at student level.

Future work will include conventional development of educational applications for the walking robot. We will also use the occasion to investigate the performance of the wireless XBee modules further, with respect to reliability and real-time performance. Parallel to this, we are making TosNet and μTosNet available for everyone through the forum `opencores.org`

References

1. J. V. der Spiegel, *VHDL PRIMER*. University of Pennsylvania, Department of electrical engineering, ørsted, dtu edn. `http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html`.
2. A. S. Sørensen, H. G. Petersen and O. G. Jacobsen, Implementation of a practical reconfigurable manipulator system based on hybrid parallel and sequential elements, in *IMG-04 — 1. Intelligent manipulation and grasping international conference*, (Genova, Italy, 2004).
3. A. S. Sørensen, Modular control of industrial mechanics, PhD thesis, University of Southern Denmark, (Campusvej 55, 5230 Odense M, 2003). `http://www.stengaard.net/anders-s/Thesis/phd.pdf.gz`.
4. S. Falsig and A. S. Sørensen, Tosnet: An easy-to-use, real-time communications protocol for modular, distributed robot controllers, in *Proc. ROBOCOMM-2009 2. Int. Workshop on Robot Communication and Coordination*, (Odense, Denmark, 2009).
5. R. Ugilt, S. Falsig and A. S. Sørensen. YouTube video(may, 2010), `http://www.youtube.com/watch?v=Gr43OqVB7_4`.