Seamless Extension of a Robot Control Framework to Bare Metal Embedded Nodes

Steffen Schütz, Max Reichardt, Michael Arndt, Karsten Berns

Robotics Research Lab, Department of Computer Science
University of Kaiserslautern
Gottlieb-Daimler-Straße
67663 Kaiserslautern, Germany
{schuetz, reichardt, arndt, berns}@cs.uni-kl.de

Abstract: Engineering complex robot control systems typically suffers from a major break in development process between different platforms – powerful computers running an operating systems and bare metal embedded computing nodes. This work presents an approach to close this gap by running an existing state-of-the-art software framework with only minor modifications on all platforms. In this context, we discuss relevant design principles of robotic frameworks and validate their importance.

1 Motivation

Robotic software frameworks have major impact on development effort and quality of robot control systems. Complex systems typically involve embedded hardware in order to interact with the magnitude of sensors and actuators. It is often advantageous to shift low level tasks to dedicated hardware – which has the potential to significantly reduce energy consumption, weight, and size. These are critical factors for many robotic systems.

A lot of research on robotic frameworks targets powerful computing platforms in order to achieve the many relevant quality attributes in robotics systems [RFB13a]. Framework support for small embedded nodes is, however, limited and there is typically a big gap in the development process. Developers are often only familiar with one of the two *worlds*.

As current compilers mostly support embedded architectures, the software development process can be more unified. Obviously, there is a break where interaction with dedicated hardware is required. Hence, the presented work targets at extending the FINROC framework[1] [RFB13b] to support *bare metal* (no operating system) embedded processors – in the case of this work a soft core run in an FPGA.

Overall, to better the situation, we pursue to achieve the following goals:

- Provide similar software development process
- Retain as many quality attributes as possible

[1] http://finroc.org/

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

- Achieve transparent integration – standard FINROC applications as well as tooling
- Light-weight implementation performing with low cycle times (in the range of kHz)
- Allow component reuse across the two *worlds*

In this paper, we present an approach to reach these goals. This is achieved by focusing on efficient HW/SW-codesign. FPGAs provide the necessary flexibility in this respect.

2 Related Work

Overall, we are not aware of any component-based software framework for complex robot control systems that can also be executed *unmodified* bare metal on embedded platforms.

The closest are model-based approaches in solutions such as OpenRTM-aist [ASK08] or SmartSoft [SHLS09]. Platform-independent models of components and applications can be transformed to platform-specific executables. All target platforms include an operating system. Notably, SmartSoft has been used with an 8-bit microcontroller (ATmega128) with only 128 kB of flash memory and 4 kB of SRAM running the FreeRTOS[2] operating system [SHLS09]. Model-based approaches typically require a complex toolchain.

The Player Project is a well-known, discontinued framework for Linux with a ”small and fast“ implementation [GVH03] – mostly in C++. It also runs on embedded Linux systems. Without device drivers, the binary size is around 64 kB.

Other solutions were specifically designed for embedded platforms. Unity [LSS13] is a notable open source robotics framework for FPGAs (only), including ”HW designs, gateware (GW, VHDL) and SW libraries“ and model-based code generation for integration with ”a high-level software framework“ running on a PC via Unity-Link.

Hartmann et al. [HSM13] propose a framework for a CPU with an FPGA coprocessor with a focus on dynamic reconfiguration. This includes automatically distributing *atomic algorithms* ”to the CPU and FPGA to ensure the best runtime with the current set of algorithms“. Atomic algorithms are implemented inside ”Basic control units“ – connectible building blocks that run on both architectures. It is still in an early stage of development.

Robot Makers GmbH developed FINROC-lite – ”a light-weight control framework specially designed for embedded systems“ [HHLF14]. It is an independent implementation in C++ suitable for FPGA soft cores – currently without a component model. It has sophisticated integration with FINROC.

PEIS-ecologies [SB05] envision to create ambient intelligence using connected, highly heterogeneous [SBG+08] devices, ranging from simple radio frequency tags to PCs. The architecture deals with this heterogeneity by keeping the kernel highly portable. According to [SBG+08], the kernel has been tested “on hardware platforms ranging from multi-core 64bit processors down to 8-bit microcontrollers, and on software environments such as Linux, MacOS, Windows, TinyOS, OpenR and embedded Linux” [SBG+08, III.].

[2] http://www.freertos.org/

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

3 Conceptual Approach

3.1 Software Framework – Design Considerations

In order to be suitable for running bare metal on embedded platforms, a software framework must not depend on any functionality provided by an operating system – such as a file system, processes or even threads. Cross-compiling and static linking must be possible. Furthermore, it is highly advantageous if there are no dependencies on external libraries (e.g. boost), as they often depend on operating system functionality and would need to be cross-compiled as well. Resource consumption is critical, too. In particular, computational overhead should be minimal. Memory consumption and the size of compiled binaries is also relevant. The most efficient way of exchanging data between different processing units on an FPGA is typically via some kind of shared memory. Thus, it is advantageous if a framework's ports can be set up to store their current data at specific memory locations. From a broader view, all these requirements are closely related to the *portability* of robotics software – an important quality attribute. A framework suitable for bare metal embedded platforms is arguably highly portable.

A central question when creating a bare metal variant of a software framework is, whether to use the same code base as the full PC version or whether to create an independent (re)implementation. This decision has an impact on development and maintenance effort – as well as efficiency. When using the same code base, it is less effort to keep both variants aligned and behaving in the same way. However, the code will inevitably contain `#ifdef` preprocessor directives and care needs to be taken that code changes do not break other framework configurations. With a reimplementation, arguably more optimized realizations are possible and likely – at least if the PC version is not to be cluttered with `#ifdefs`.

With most frameworks for robotics used on PCs, it is difficult to reach all the requirements for embedded platforms mentioned above. Thus, separate light-weight solutions are typically created. Notably, due to missing C++11 support for the Altera NIOS®II soft core until only recently, FINROC was not suitable for our hardware platform either.

Evaluating all these factors, we opted to adapt the existing FINROC framework [RFB13b] that we use and develop for implementation of complex robot control systems on embedded PCs to also run without an operating system. We believe that FINROC has some key features that make it particularly suitable for this project:

- A highly modular framework implementation: Framework functionality can be tailored to application and platform requirements (see [RFB13b] and Fig. 4).
- Relevant features do not depend on external libraries, middleware, or operating system functionality
- Multiple components can be instantiated in the same executable
- An efficient implementation

Notably, we see FINROC as a proof-of-concept for the proposed approach. Any other framework with these properties should be suitable as well.

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

3.2 FPGA-based Embedded Nodes

To provide a high degree of flexibility on the embedded side, the targeted embedded nodes are FPGA-based. Using the elaborated toolchains provided by the FPGA-manufacturers, it has become feasible to implement complex systems incorporating multiple soft cores that can be extended by custom IP-cores. Thereby it has become feasible to implement high-performance dedicated embedded systems.

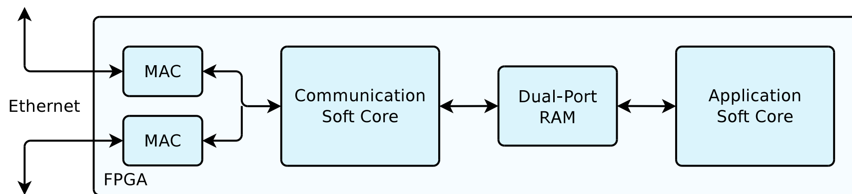

Figure 1: Concept for the FPGA SoC

In order to execute the FINROC framework with good performance, the underlying design approach is to distribute application and communication on two separate soft cores (Fig. 1). The communication soft core interfaces the Ethernet MAC, handles incoming Ethernet frames, and executes the handling of the higher-level communication protocol (see 3.3). For inter-processor communication, the most efficient mechanism is shared memory. Most modern FPGAs feature dual-ported RAM entities allowing simultaneous read/write access by two masters, which is especially suitable for this scenario.

The more detailed design of the system should be guided by two requirements: On *communication side*, to keep bus cycle times low, packets must be handled efficiently. On *application side*, the overhead due to the handling of process data must be kept minimal.

By following a consequent HW/SW co-design approach, a system can be designed that provides the required infrastructure for the performant deployment of FINROC.

3.3 Communication Protocol

The protocol to communicate with the embedded nodes is driven by several requirements:

- Provide a high bandwidth
- Use of standard hardware
- Light-weight regarding communication and computation
- Flexible network topology
- Reliable data exchange
- Tailored to the framework communication

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

- Support dynamic initialization during run-time
- Programming of embedded nodes via the bus system

The following design decisions emerged. For physical and data link layer of the OSI model, Ethernet is the most suitable solution – hardware is widely available, it is well standardized, and offers bandwidths in the gigabit range. Moreover, using Ethernet on the data link layer allows a flexible network topology. If the embedded nodes feature two Ethernet interfaces and support frame forwarding, any mixture between star and line topologies can be realized by using standard layer two switches. On the network and transport layer of the OSI model, UDP/IP is chosen. Thus, on PC side, the protocol can be executed from user space using UDP sockets. Further on, both – IP as well as UDP – include a checksum to test frame integrity. While TCP guarantees reliable, ordered, and error-checked streams of packets on the transport layer, it causes overhead in the form of additional protocol logic, as well as additional bus load. This makes it unsuitable for systems where computational resources are limited. Package loss, fragmentation, and reordering must be handled at a higher layer – if needed.

An alternative would have been to use one of the industrial Ethernet-based fieldbus systems like EtherCAT[3] or Ethernet Powerlink[4]. Not building on top of a rather strictly defined fieldbus system allows for the protocol to be tailored to the framework communication and the requirements of the typical application scenario. Additional features, such as the dynamic initialization during run-time or the possibility to program the embedded nodes via the bus, are easy to realize. This of course comes at the cost of increased latency and jitter – hard real-time cannot be guaranteed running the bus master on top of a UDP socket on a standard OS.

The *Framework-integrated Embedded Protocol* (FINEMBP) is located at the session layer of the OSI reference model. Inspired by Ethernet Powerlink, FINEMBP separates the communication into cycles. As illustrated in Fig. 2, the cycle is split into an *isochronous*, an *asynchronous*, and an optional *idle* phase.

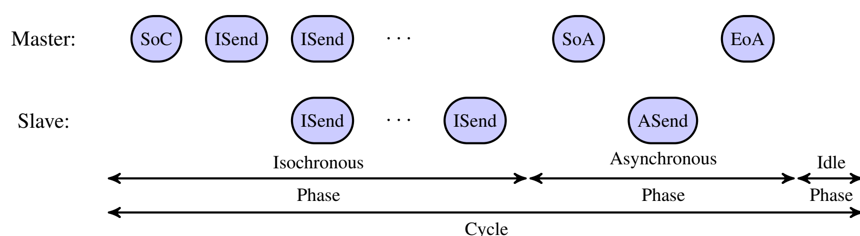

Figure 2: Basic segmentation of a full cycle in FinEmbP.

The *SoC*, *SoA* and *EoA* frames structure the FINEMBP cycle. They have to be received by every node and therefore are sent as a broadcast/multicast. In the isochronous phase,

[3] http://www.ethercat.org/
[4] http://www.ethernet-powerlink.org/

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

cyclic process data is exchanged by a poll mechanism. The *ISend* packets, sent by the master, contain the process data bound for the corresponding node, while the *ISend* response contains the most recent process data, produced by the node. The master sends an *ISend* packet to each of the nodes containing process data. During the asynchronous phase the master can either send or receive any data that does not require cyclic transmission. The asynchronous phase can be used – e.g. for network initialization and data that does not require cyclic transmission.

The underlying state machine of the FINEMBP protocol is shown in Fig. 3. Each state encapsulates a master/node-specific sub state machine.

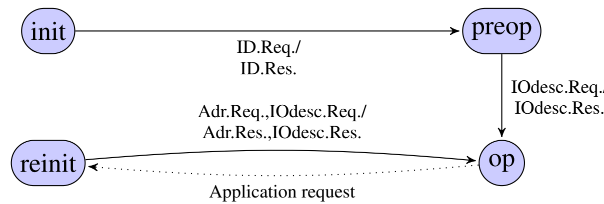

Figure 3: Basic states machine for FINEMBP master and nodes

In the *init* state the master checks the network for FINEMBP nodes. If one or more nodes are available, the master assigns each of them an IP address and thereby manipulates OSI layer 3. Once all available nodes got assigned an IP, the master and all nodes are in the *preop* state. In the *preop* state, the master queries each node for its structure information. The structure information includes all information describing the FINROC application deployed on the node. After the structure information of all nodes has been exchanged, the system is in the *op* state. From now on, cyclic data is exchanged as described above. With the initialization mechanism being allocated in the asynchronous phase of the FINEMBP cycle, additional nodes can be initialized while the network is running.

4 Implementation Details

4.1 FINROC

As introduced in [RFB13b], FINROC is implemented in a highly modular way – composed from slim software entities with a specific concern. By selecting a suitable set of these entities, it can be tailored to application and platform requirements. For the bare metal version of FINROC, we targeted a minimal configuration for executing the common FINROC component types – defined in the *structure* and *ib2c* plugins. The configuration is illustrated in Fig. 4[5].

Although, we generally try to minimize use of the C++ preprocessor for maintenance rea-

[5] RRLIBs are framework-independent libraries, whereas plugins contain optional functionality for the FINROC core. Lines of code were counted using David A. Wheeler's 'SLOCCount'.

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

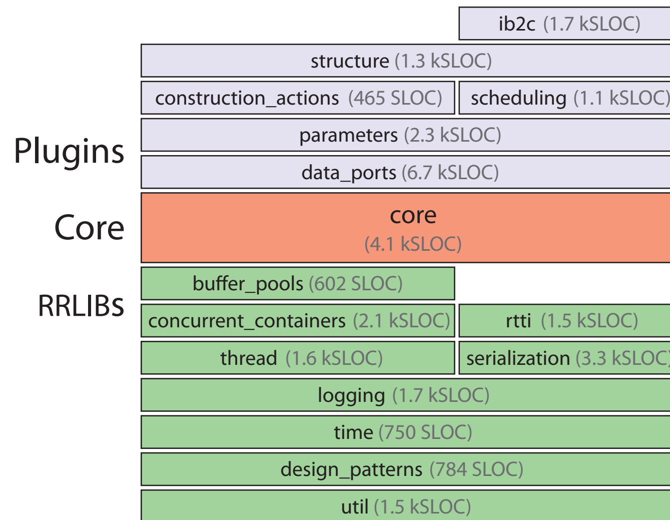

Figure 4: FINROC's highly modular core in the configuration running bare metal on the FPGA soft core

sons, there are futhermore few `#defines` that can be set for specific target platforms – most importantly `RRLIB_SINGLE_THREADED` which will compile FINROC in single-threaded mode with e.g. a major performance increase for data ports. Furthermore, pre-processing directives can check whether certain software entities (RRLIBs and plugins) are present[6].

Without a network protocol, the default FINROC configuration only depends on the platform-independent *libxml2* library[7]. It is mainly used for processing configuration files in the *parameters* plugin and *rrlib_logging*, as well as serializing application structure in the *runtime_construction* plugin. As this is not relevant for FINROC's bare metal version, the dependency is now optional – availability of XML-related functionality depending on availability of *libxml2*.

Notably, there is a single-threaded variant of *rrlib_thread*. This was preferred to removing it from the configuration, as the latter would require cluttering the code with `#ifdefs`. If `RRLIB_SINGLE_THREADED` is defined, many classes such as mutexes and locks are empty, resulting in nops wherever they are used. Thread objects still exist and could be used for basic handling of virtual threads. The RRLIBs *concurrent_containers* and *buffer_pools* employ policy-based design [Ale01] – e.g. using different implementations (policies) depending on the number of producers and consumers. This always includes a single-threaded variant.

Similarly, FINROC's data ports use different implementations depending on the type of data they transfer (more precisely, the type traits of T). This is illustrated in Fig. 5. For

[6] FINROC's build system creates respective `#defines`

[7] `http://xmlsoft.org/`

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

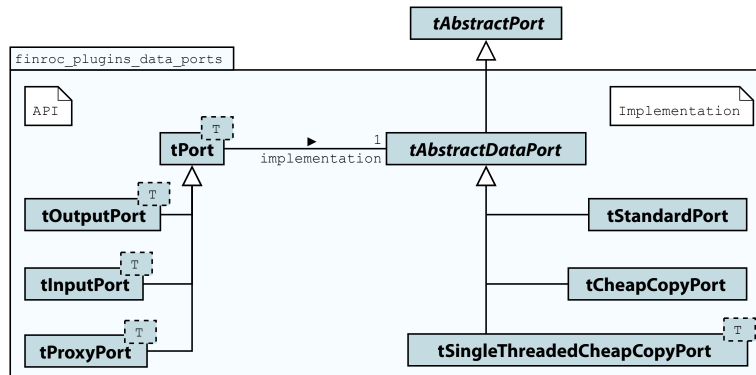

Figure 5: Port classes in FINROC's *data_ports* plugin (simplified)

large data types such as images that are expensive to copy, a zero-copy implementation with buffer pools and reference counting is used [RFB13b] – *tStandardPort*. It is also used for types with internal memory management such as *std::string* – as memory allocation is to be avoided in real-time applications and also relatively expensive. For cheaply copied types such as scalars and vectors, there is an optimized implementation using thread-local buffer pools – *tCheapCopyPort*. In single-threaded mode, however, there is no necessity for buffer pools at all. Instead, a port's current value can simply be stored at a fixed location in memory, as there is no concurrent access. Thus, a simple and very different single-threaded implementation for cheaply-copied types was created. The implementation allows to set the memory location the port data is stored to – possibly shared memory that processing units on an FPGA use to exchange data.

In theory, connected *tStandardPort*s could operate on the same data buffer. However, e.g. the smart pointer classes in the API allow to reference potentially multiple buffers of the same port. As copying is not a viable option for complex types, the standard implementation is used on single-threaded platforms, too.

An issue during compilation was that the *libstdc++* shipped with Altera's gcc 4.7 compiler does not fully support the C++11 standard – e.g. *std::chrono* is missing. Furthermore, some functions defined in headers cannot be found when linking. Thus, we added simple implementations of the relevant missing parts to a small target-platform-specific library. This was added to the target platform's compiler and linker flags. In future compiler versions this will likely be obsolete.

Overall, using a highly modular framework core combined with techniques such as policy-based design, the bare metal version of FINROC could be created adding only a limited amount of `#ifdefs` (no nested ones in particular).

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

4.2 FPGA HW/SW-System

The system outlined in 3.2 has been implemented on the eSM Board developed and produced by elrest Automationssysteme GmbH. At the core of the board, there is an Altera[8] Cyclone IV FPGA. Due to the external peripherals connected to the FPGA, the board is designed to optimally support the deployment of two soft cores. As it also features two Ethernet interfaces, it is suitable for the use in networks with line topology.

A detailed illustration of the implemented logic system in the FPGA is provided in Fig. 6. Only the components and bus connections that are relevant for the data flow of incoming Ethernet frames are illustrated.

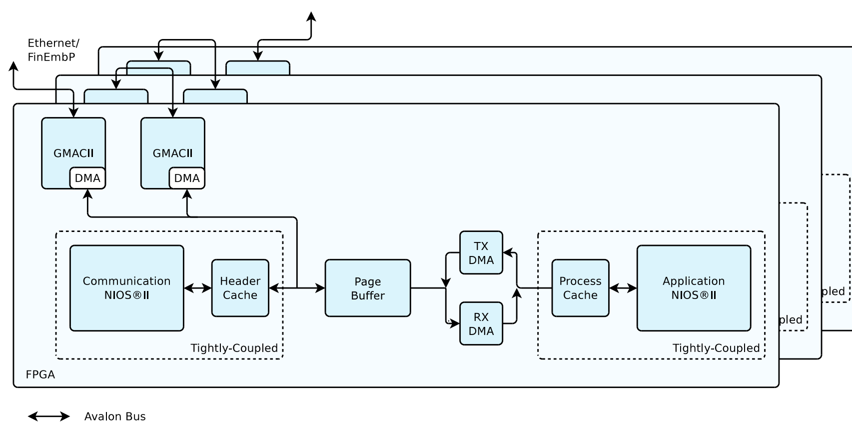

Figure 6: Detailed illustration of the FPGA system.

The implemented soft core is the NIOS®II provided by Altera. Ethernet medium access is managed by two GMACIIs, a high performance Gigabit Ethernet MAC developed by IFI[9], which are interfaced by the *Communication NIOS®II*. A prominent feature of the GMACII is the integrated DMA, which is alignment-aware, byte-exact and provides on-the-fly checksum calculation. Both soft cores have dual-port RAM attached as tightly-coupled memory, which allows for fast one-cycle read/write access.

On communication side, the tightly-coupled memory is mainly used for header caching, while on application side, the process data is located in the memory. The two processors are coupled through a third dual-port RAM, the *Page Buffer*. Besides holding a structure for the inter-processor communication, once the node is initialized, it serves for the exchange of coherent pages of process data. Therefore, the memory is segmented into several areas, buffering the different process data pages. Using two pages for cyclic process data and some page flipping logic, delays in case of concurrent access from application and communication side can be avoided.

On the application side the data is transferred to and from the *Page Buffer* using the

[8] http://www.altera.com/
[9] http://www.ifi-pld.de/

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

GMACII-integrated DMAs. The application system is extended by two DMAs as well – the *TX DMA* for data transfer from the *Process Cache* to the *Page Buffer*, the *RX DMA* vice versa.

Software-wise, the *Communication NIOS®II* runs a UDP stack handling layer two, three, and four of the OSI model. To allow for Ethernet line topologies, the stack features frame forwarding on OSI layer two. The implementation of the UDP stack makes extensive use of the *GMACII-DMA*s for fast frame handling and checksum calculation. On top of the UDP stack runs the client-FSM handling the FINEMBP protocol and the inter-processor communication. A new Ethernet frame arriving at one of the Ethernet interfaces is, depending on the the frame and the state of the FINEMBP FSM, either forwarded to the second Ethernet interface, copied to the *Page Buffer*, or both. If data is copied to the *Page Buffer*, this will be indicated to the *Application NIOS®II* by setting the according flag. Again, depending on the context, the FINEMBP client-FSM might respond to the incoming frame by sending an adequate frame. In most cases, the response contains process data from the *Page Buffer*. On application side, at the beginning of a FINROC cycle, the *Page Buffer* is checked for new incoming process data. If data is available, it is copied to the *Process Cache* using the *RX-DMA*. As mentioned above, the FINROC ports are mapped to constant memory locations, located in the *Process Cache*. Thus, this two-staged buffer setup relieves the *Application NIOS®II* of the need for any pointer arithmetic. At the end of the FINROC cycle, outgoing data is transferred to the *Page Buffer* via the *TX-DMA*.

By handling data transfers with the DMAs and avoiding pointer arithmetic due to the two-staged buffer setup, the resulting overhead for the *Application NIOS®II* is kept minimal.

4.3 Build System

The build system used to build FINROC needed only minor modifications. So-called *targets* contain definitions for the compiler and compile flags to use. For this work, the NIOS®II compiler provided by Altera is specified and some soft-core-specific compiler/linker flags are added. Two modifications to the build system itself were needed:

Running the code on a bare metal embedded processor requires static linkage of the final binary. This is in contrast to the usual procedure for running with an operating system. There, dynamic linking is highly advantageous to save space. The build system was thus extended to not only produce shared library objects , but also to produce archives to enable static linkage of the binaries, when requested.

Additionally, the build-process requires a so-called *board support package* containing important definitions of the hardware (including a linker script), prior to the linking process. Also, the binaries that result from compiling and linking need some post-processing of the ELF binaries so that they can be run on the NIOS®II soft core. This led to the extension of the build system by two hooks that can be overridden by targets (called *pre-build-hook* and *post-build-hook*). For this work, the *pre-build* hook builds the board support package and the *post-build* hook post-processes the ELF binaries so that they can directly be uploaded to the embedded soft core.

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

5 Benchmark

The benchmark for evaluating the performance of the proposed system is running a simple PID closed-loop controller within a FINROC module on the soft core. To make the benchmark as transparent as possible, an existing implementation of a PID controller for Atmel AVR microcontrollers has been used instead of implementing a proprietary solution. The documentation and the implementation of this controller are given in Atmel's application note *AVR221: Discrete PID Controller*[10].

This application note provides an implementation in plain C. The FINROC module is just a wrapper around the C implementation. It instantiates a controller and periodically feeds it with sensor data to obtain the control-signal output. The resulting statically linked FINROC binary has a size of about 2.5 MB. Using this setup, it was possible to run the FINROC framework with an update rate of 25kHz. Thus also the control loop of the PID controller runs with this frequency, which is more than enough for most typical closed-loop control applications [ASHO+07]. For comparison, the same (single threaded) FINROC-implementation of this controller runs with an update rate of 3.75 MHz on an Intel Core i7-920 CPU clocked at 2.67 GHz.

6 Conclusion and Outlook

To conclude, an approach was presented to deploy a state-of-the-art component-based robotics framework on an FPGA soft core. Using a highly modular framework core combined with techniques such as policy-based design, a bare metal version of robotic software frameworks can be created with only minor code changes. Results show negligible computational overhead from using a full-featured framework. In fact, a performance level significantly above the initial expectations and goals of the authors was reached. In particular, closed-loop controls can be executed at frequencies above 10 kHz. However, binary size of several megabytes and the dependency on C++11 is a limiting factor regarding portability to some other embedded architectures. Without an operating system and minimal I/O latency, the developed systems seems to be an ideal platform for the development of real-time applications – without the typical gap in the development process. Furthermore, this facilitates the realization of highly distributed robot control systems with dedicated hardware nodes.

Next steps will be to improve integration with FINROC's tooling and to gain experience from use in projects. If required, further optimizations are possible – e.g. reduction of binary size. Notably, we observed that including certain C++ standard headers have a significant impact in this respect already.

[10] http://www.atmel.com/webdoc/atmel.docs/atmel.docs.33085.19841.html

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany

References

[Ale01] Andrei Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 2001.

[ASHO+07] Alin Albu-Schäffer, Sami Haddadin, Christian Ott, Andreas Stemmer, Thomas Wimböck, and Gerd Hirzinger. The DLR lightweight robot: design and control concepts for robots in human environments. *Industrial Robot: An International Journal*, 34(5):376–385, 2007.

[ASK08] Noriaki Ando, Takashi Suehiro, and Tetsuo Kotoku. A Software Platform for Component Based RT-System Development: OpenRTM-Aist. In Stefano Carpin, Itsuki Noda, Enrico Pagello, Monica Reggiani, and Oskar von Stryk, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 5325 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin / Heidelberg, 2008.

[GVH03] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, Coimbra, Portugal, June 30 - July 3 2003.

[HHLF14] Carsten Hillenbrand, Jochen Hirth, Bernd Helge Leroch, and Martin Frank. Closed-Loop Joint Angle Control for a Multi-Axes Hydraulics Arms - Towards Autonomous Construction Machines. In *Commercial Vehicle Technology 2014 - Proceedings of the Commercial Vehicle Technology Symposium (CVT)*, pages 37–46, Kaiserslautern, Germany, March 11-13 2014.

[HSM13] Jan Hartmann, Walter Stechele, and Erik Maehle. Self-reconfigurable Control Architecture for Complex Robots. In *Informatik 2013*, Lecture Notes in Informatics (LNI), Koblenz, Germany, September 16-20 2013. Springer.

[LSS13] Anders B. Lange, Ulrik P. Schultz, and Anders S. Soerensen. Unity: A Unified Software/Hardware Framework for Rapid Prototyping of Experimental Robot Controllers using FPGAs. In Davide Brugali, editor, *Proc. of the 8th full-day Workshop on Software Development and Integration in Robotics*, Karlsruhe, Germany, May 2013.

[RFB13a] Max Reichardt, Tobias Föhst, and Karsten Berns. Design Principles in Robot Control Frameworks. In *Informatik 2013*, Lecture Notes in Informatics (LNI), Koblenz, Germany, September 16-20 2013. Springer.

[RFB13b] Max Reichardt, Tobias Föhst, and Karsten Berns. On Software Quality-motivated Design of a Real-time Framework for Complex Robot Control Systems. *Electronic Communications of the EASST*, Software Quality and Maintainability(60), August 2013. http://journal.ub.tu-berlin.de/eceasst/article/view/855.

[SB05] A. Saffiotti and M. Broxvall. PEIS Ecologies: Ambient Intelligence meets Autonomous Robotics. In *Proc of the Int Conf on Smart Objects and Ambient Intelligence (sOc-EUSAI)*, pages 275–280, Grenoble, France, 2005.

[SBG+08] A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, J. Rashid, B.S. Seo, and Y.J. Cho. The PEIS-Ecology Project: Vision and Results. In *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*, pages 2329–2335, Nice, France, September 2008.

[SHLS09] Christian Schlegel, Thomas Haßler, Alex Lotz, and Andreas Steck. Robotic Software Systems: From Code-Driven to Model-Driven Designs. In *14th International Conference on Advanced Robotics (ICAR)*, Munich, Germany, June 22-26 2009.

Copyright GESELLSCHAFT FÜR INFORMATIK E.V. (GI) · AHRSTRAßE 45 · 53175 BONN
In: E. Plödereder, L. Grunske, E. Schneider, and D. Ull (eds.)
Informatik 2014 - Lecture Notes in Informatics (LNI)
ISSN 1617-5468 / ISBN 978-3-88579-626-8
Bonner Köllen Verlag (2014)
September 22-26, 2014 - Stuttgart, Germany