

Communication Concept for Adaptive Intelligent Run-Time Systems Supporting Distributed Reconfigurable Embedded Systems

Michael Ullmann and Jürgen Becker

Universität Karlsruhe (TH), Germany
{ullmann, becker}@itiv.uni-karlsruhe.de

Abstract

Reconfigurable computing systems have already shown their abilities to accelerate embedded hardware/ software systems. Since standard processor-based embedded applications have come to their limits we need new concepts for controlling and managing embedded, possibly distributed, reconfigurable hardware/ software computing systems. Succeeding to previous papers which dealt with management aspects of run-time reconfigurable systems and related AI-approaches this contribution describes an approach and proof of concept of a transparent communication mechanism between the application layer and its possibly distributed and reconfigurable hardware/ software sub-function modules.

Keywords: management of embedded reconfigurable systems, communication concepts

1. Introduction

During the last years the deployment of reconfigurable hardware has become a growing trend in the academic and commercial domains of electrical and computer engineering. Because of the increasing risks and cost of development ASIC prototypes are hardly affordable to many academic institutions and small and medium-sized enterprises. So despite of their higher device cost and inefficient area use, reconfigurable devices have become an interesting and competitive alternative for prototyping compared to ASIC prototype designs. For many tasks, ASICs are preferred over General Purpose Processors (GPP), since ASICs perform significantly better. However, they lack the possibility to adapt themselves to varying environments. In contrast to this, GPPs can cope with branched control flows including recursion etc. so they adapt easily to new requirements. As next evolutionary step reconfigurable computing can combine the performance of ASICs and the flexibility of GPPs. Many commercial vendors nowadays already include reconfigurable building blocks like Field-Programmable

Gate Arrays (FPGAs) as core components in their products since post-production reconfigurability significantly reduces design risks and possible maintenance costs so that hardware with reconfigurable features is on its way to conquer the markets of tomorrow. Additionally this influences the traditional hardware design of circuits used in devices, like mobile phones or PDAs. Another aspect is that state-of-the-art microprocessor-based embedded solutions can no longer overcome the growing computational demands of future control and communications applications. Since modern FPGAs are higher integrated than their previous generations offering new features like partial run-time reconfiguration and powerful hard-wired on-chip processors (e.g. on Xilinx Virtex II pro FPGAs [26]) they are highly predestined for a wide range of applications making them attractive for a new class of future embedded applications. Reconfigurable computing systems have already shown the ability to greatly accelerate program execution, thereby providing a high-performance alternative to pure software-based implementations and a programmable alternative to expensive ASICs. The development of new architectural hardware/ software system concepts [3, 10, 13, 19, 24] by exploiting these powerful features of flexible and adaptive hardware-accelerated coprocessors in combination with a design paradigm shift is an adequate approach to adapt to the market's requirements. Nevertheless the availability of the best superior reconfigurable architectures will not guarantee their success, if no sophisticated control and management mechanisms are provided by the system developers dealing with fault tolerance aspects and application scenarios with dynamically changing power/performance constraints. While prior researchers have addressed architecture design, programming and compilation issues [7, 12], there is still not much consensus on what kind of operating system (OS) support should be provided for reconfigurable architectures. Recent academic approaches already implemented complete reconfigurable system-on-chip supporting run-time reconfiguration of dedicated functions and their management at run-time [6, 10, 18, 22, 23, 24]. Some of these first approaches have already included low budget

embedded operating systems running on soft-core or hard-wired on-chip processors on FPGA (e.g. uClinux on a Xilinx MicroBlaze, [4, 16, 25]). Combining these approaches using one or several low-cost reconfigurable devices plus dedicated hardware like ASICs or DSPs will create flexible and highly adaptive multi-purpose systems which can be applied in a variety of application domains (e.g. automotive infotainment, multimedia, control-oriented applications etc.) [8, 10, 18].

1.1. Reconfigurable Layered System Concept

The development and proof of concept of such a versatile approach is a main research topic of our research group. In the following we want to give a short overview on our related previous work and the system concept. One of our previous successful approaches consisted in the development and implementation of a first run-time reconfigurable system-on-chip, supporting flexible on-demand hardware-task switching and a sophisticated run-time reconfiguration and task management mechanisms on Xilinx Virtex II FPGAs [9, 18, 19]. By exploiting the column-wise reconfiguration possibilities of Xilinx Virtex II FPGAs it offers at run-time a set of functionalities which are switched on demand during operation in a time-multiplexed way, so that a larger set of functionalities appears to be available to the user [18, 19]. Furthermore the internal bus-system was modified, so that local bandwidth- and topology-adaptive Networks-on-Chip (NoC) [2] can be created at run-time, allowing optimized energy and resource saving communication and operation modes between the instantiated modules [9]. Although the tested application domain in our previous work targets currently at automotive control applications with soft time and security constraints we intend to extend our approach to other fields of application as well. Figure 1 (middle/bottom section) above shows how this previous approach can be re-used as one of multiple-instances of run-time reconfigurable hardware/software sub-systems based on FPGA, DSP or standard processor technology. As can be seen from figure 1, the system is logically divided into different layers. On top at the application level different applications are executed depending on their location and mode of operation (as parallel hardware or sequential software tasks). Most applications are conceived to have major parts in software and some dedicated parts accelerated in reconfigurable hardware or DSP. An application programming interface layer is introduced separating the application level from lower sub-system levels [22, 23]. This API offers dedicated services for inter-layer communication and Quality of Service (QoS) negotiation mechanisms which can be used for sub-function calls. Another system level which can be settled in the middle of the system hierarchy is responsible

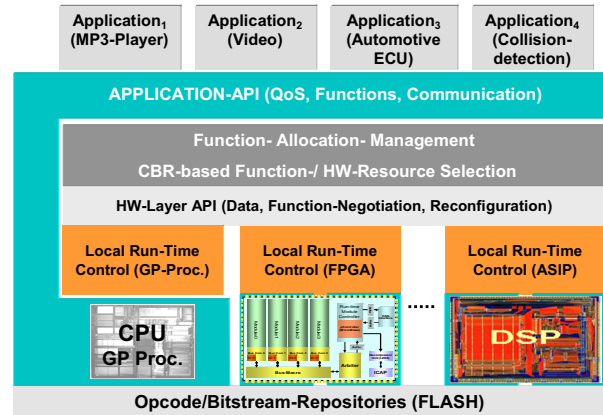


Figure 1. Reconfigurable system layers

for the proper allocation of sub-functions requested by the application layer. Depending on the QoS demands, given by a calling application an appropriate implementation of the desired sub-function has to be found from a run-time function repository. The retrieval of suitable implementation variants of a requested function type based on given QoS-parameters can be performed by applying a hardware-accelerated Case-Based-Reasoning (CBR) approach [1, 20]. Apart from the needed information about available function implementations and their QoS-features the system will need information about its current load and power consumption status, which are provided by the HW-Layer API one level below. This HW-Layer API is the responsible interface concerning all hardware relevant aspects like resource consumption, low-level communication, fault tolerance aspects and reconfiguration of system parts. It connects the high level system layers to the local system controllers, which can be located on different devices (e.g. standard CPU, FPGA (soft-core CPU) or DSP) or on the same chip as well. The local sub-controllers are responsible for the management of local run-time reconfiguration and other sub-tasks like local task/ resource management and communication issues [18, 19]. It should be noted that the system as shown in figure 1 can be comprehended as distributed system built of discrete devices. Nevertheless the proposed concept can be realized as system-on-chip as well. Since we have already described detailed aspects of the proposed system's resource-allocation- management strategies and CBR-approach in earlier papers [18, 19, 20] we will focus in this contribution on another important aspect, the system's communications architecture providing transparent communication between applications and their allocated low-level sub-systems.

Our contribution is structured as follows. In section 2 we give an overview on the demonstrator's current implementation and its different components. Section 3 outlines the internal hardware/software system structure

and operating system adaptations. In section 4 we introduce the Universal Multiplexing Multi-channel Message Transceiver (UM³T) concept and an example scenario. Current FPGA synthesis results are presented in section 5. The paper concludes with a summary and perspectives on our future work.

2. System Platform Overview

In the following we want to give an overview on the system demonstrator and its used components. As figure 2 shows, the complete demonstrator consists of three main components. The first bigger component is a Xilinx ML 310 board featuring a Xilinx Virtex II pro XC2VP30 FPGA including two on-chip PowerPC processors (on-board clock 100 MHz). The second main component is a low cost Digilent FPGA-board equipped with a Xilinx Spartan-3 XC3S400 low-power FPGA (on-board clock 50 MHz). The third main component is a standard of-the-shelf 2 GHz Pentium M laptop computer which is used for control and debugging purposes. As can be seen from figure 2 both boards are connected through a Controller Area Network-bus (CAN) [17] as shared communications medium. The CAN-bus protocol is known as a robust high-speed real-time capable serial-bus communication standard in the automotive domain where it is used to interconnect engine control units and other electronic automotive devices. The laptop computer is connected to the CAN-bus through a PCMCIA-CAN-connector card, which enables for communication and bus sniffing purposes. The laptop's software functionalities for accessing the CAN-bus are provided by the CANoe software toolkit, which offers all needed means for programming interactive CAN-aware applications [21]. The laptop can be alternately connected to both boards through a JTAG-interface for configuration programming and run-time debugging purposes. Additionally the laptop is connected via a hyper-terminal on its COM RS-232 ports to both boards enabling for text I/O and ASCII-data streaming. The ML 310 board features an attached LCD-display for text output and a simple self-made audio output device. As mentioned above both boards are connected to CAN-bus through special CAN-transceiver cards which provide an analog-digital interface between CAN-bus and digital on-chip CAN-IP-core functionalities. The used Verilog CAN-IP-core originates from an open-source hardware project where it was developed on the basis of the original CAN-bus system's specifications [17] for providing a free licensed CAN-IP-core to the community. The interconnecting CAN-bus main cable and the described CAN-transceiver cards are because of cost reasons self-designed based on the given CAN-bus system's specifications. Unfortunately the deployed cable has a lower performance compared to commercial expensive high-quality CAN-connector bus-cables. The

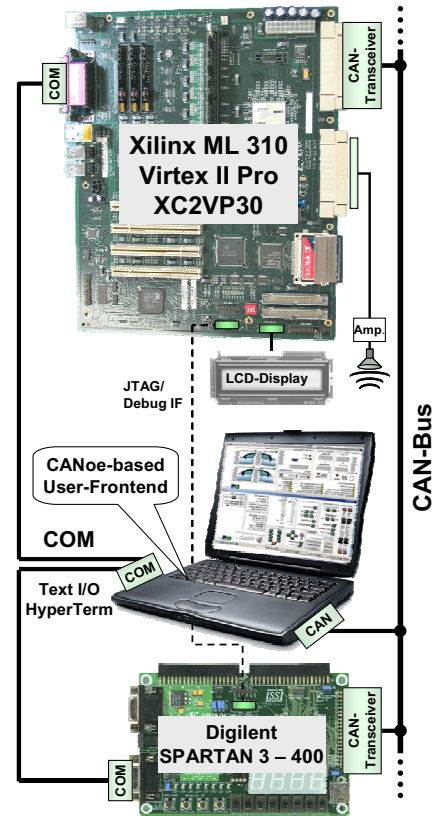


Figure 2. Hardware system-setup

main problem was to correct the cable's wave impedance adjustment to a value of 120 Ohms which was not such successful so that our best efforts resulted in a 280 kbit/s error free peak performance compared to a specified theoretical value of 1 Mbit/s. Although it is possible to operate the bus in the range of 1 Mbit/s the transmission error rate was much too high for being tolerated.

3. Hardware/Software System Structure

As mentioned above the ML 310 features a Virtex II pro FPGA with two hard-wired on-chip PowerPC processors. These processors are used to set up an on-chip multi processor system which is driven by a real-time operating system in a master/slave fashion on both processor cores. Figure 3 gives an overview on the internal on-chip system structure. On every processor we implemented an instance of the μ C/OS-II real-time operating system [11, 14]. On top of each OS implementation a set of several dedicated system-service and application tasks is provided. The service tasks are responsible for the local resource management, peripheral driver support and communications handling (UM³T, section 4). A superordinate centralized function is inherent in the allocation management function which is located on

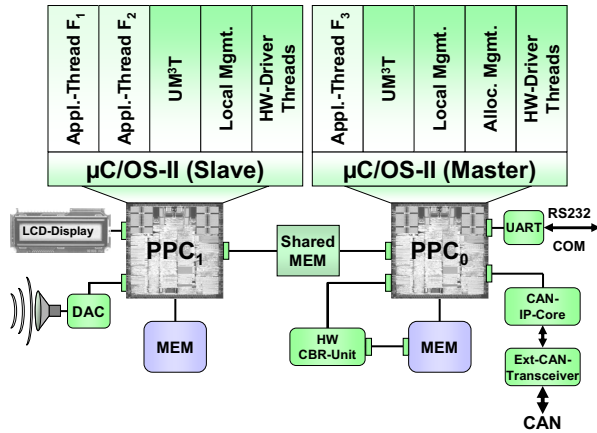


Figure 3. Virtex II pro platform

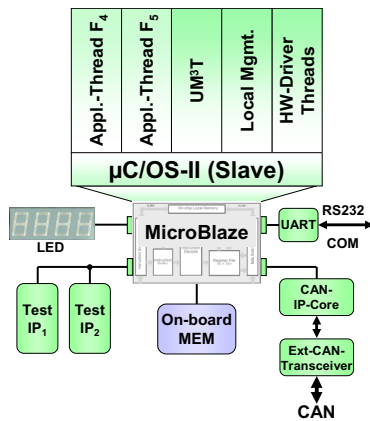


Figure 4. SPARTAN-3 platform

the master-processor (PPC_0). As mentioned in section 1 it is responsible for handling function requests from local (and possibly non-local) application tasks [20].

The used operating system offers various advantages. It is highly scalable and can be run from read-only memory if needed. It features a preemptive and deterministic multitasking kernel for microprocessors and micro-controllers. Up to 63 application tasks (+1 kernel task) can be handled and most common OS services like semaphores, message queues and time/task management are supported. The execution time for most of the provided kernel services is constant and deterministic and does not depend on the number of running application tasks. The source code is completely available as ANSI C code and is very well documented [11]. Furthermore it is completely royalty free for non-commercial use. It can be ported to various processor-based target architectures and only some minor assembler level adaptations have to be done for porting the operating system to a new hardware platform. $\mu C/OS-II$ is certified by FAA [5] and MISRA [15] for compliance showing that $\mu C/OS-II$ is a very robust and reliable piece of software. Since the actual on-chip

structure on the Virtex II pro FPGA is rather complex, figure 3 shows only a simplified view of the modular hardware/software on-chip interdependencies. Even though it was possible to interconnect both processors via a common On-chip-Peripheral Bus (OPB), we decided to implement both processors with their own separate buses, since we wanted to realize two more or less independent subsystems. Nevertheless we had to provide a fast communication channel between both processors. Although there possibly exist better solutions we implemented a shared-memory by using on-chip dual-ported Block-RAM (BRAM) resources enabling both processors to access the RAM-block via their own Processor-Local-Bus (PLB) interconnections. The available on-chip RAM resources were not sufficient for providing enough program-memory for both operating systems and their application executables, since we needed the BRAM resources for other hardware functions (like the CAN-IP core and CBR-unit) also. So we used the plentiful available on-board 256 MB DDR RAM for storing there all operating system and application executables in disjoint memory regions. The Xilinx Embedded Development Kit (EDK 6.3) [26] which was used for the development, synthesis and testing did not allow common memory areas for both processors. So we used a little trick by mapping a block of 16 KB dual-ported BRAMs as shared memory into both memory areas. The shared-memory communication channel is set up by low-level driver threads which synchronize by a simple handshake protocol. Apart from the shared-memory the on-chip implementation features some peripherals like UART/COM, CAN-IP, LCD-Display and a simple DAC for sound output. Since these peripherals are connected separately to their local processor's OPB they are treated by their processors as private peripherals, which is important for the resource management approach as it will be briefly outlined in section 4.

Figure 4 gives a simplified overview on the Spartan-3 FPGA secondary slave system. Compared to the previous PowerPC platform the Spartan-3 does not come with a hard-wired CPU so we mapped the Xilinx MicroBlaze 32-bit soft-core RISC processor [26] together with its OPB-connected UART, CAN-IP and some small test-IP peripherals on FPGA. Like on the Virtex II pro platform the available on-chip memory resources were not sufficient for the executables so that we moved all executables on the 1M-byte fast asynchronous on-board SRAM which offered enough resources for our purposes. The Spartan-slave platform uses nearly the same $\mu C/OS-II$ implementation like on PowerPC. Apart from some minor low-level modifications concerning timer & interrupt handling we could migrate the OS-kernel and management source codes without any extensive adaptations, which is one of the great benefits of $\mu C/OS-II$. Since we had to use on-board memory in both cases for storing the run-time

executables it was not possible to embed the opcodes inside the FPGAs' bitstreams. So we had to upload subsequently all executables after FPGA configuration through the processors' JTAG-debugger interface. Albeit we intend to replace in near future the resource limited Spartan-board by a modified $\mu\text{C}/\text{OS-II}$ adapted run-time reconfigurable FPGA-sub-system, we wanted to get some first motivating results on the feasibility of our sub-system spanning concept.

4. Universal Multiplexing Multi-channel Message Transceiver Concept (UM³T)

As previously mentioned in section 1 the resource and allocation management mechanism is responsible for handling the application's reservation requests on sub-functions calls. The central allocation manager's main tasks are the retrieval of suitable function-implementation variants and a feasibility check of the found functions in the context of the callers priority by considering the resource consumption and assignment. At run-time the manager stores administrative data on the current set of active instantiated functions in a function resource allocation table (FRAT) (figure 6 bottom). It contains relevant allocation and reference informations like the application handler (Alloc ID) and on which hardware-unit the function was instantiated. The table contains as many entries as functions implementations can be provided by the sub-systems. Each implementation variant offered by a sub-system is identified by its unique identifier (UID). Furthermore the table stores the calling application's priority and its reservation status. Other table-fields store information on each implementation's power consumption (PC) and required bandwidth (BWR) which have been pre-estimated by previous simulation and testing of the functions' models. These information can be evaluated in future versions for run-time optimizations. It should be noted that each table is characterized by a realization identifier (RID). This RID is provided by the CBR-retrieval unit [20] which checks a function-database for suitable function realization variants which match best to the application's request description. Figure 5 and 6 give a brief overview on the needed steps that occur during a function allocation. Figure 5 shows the decision steps to be taken if a requested (and unique) resource is already in use by an other application. Depending on its priority the calling application may preempt the application of lower priority. In that case the FRAT gets updated and the preempted application either may request another resource providing the same function type with lower quality or it can try to request the resource until it gets a new grant. Figure 6 gives a different view on the allocation steps. An example application (UID 68) demands an FIR-filter function (1). The allocation manager forwards the relevant parts of the request to the

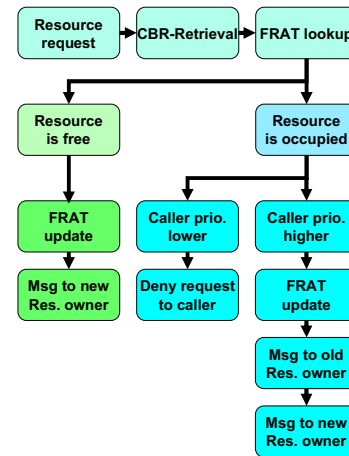


Figure 5. Preemptive priority controlled Resource allocation mechanism

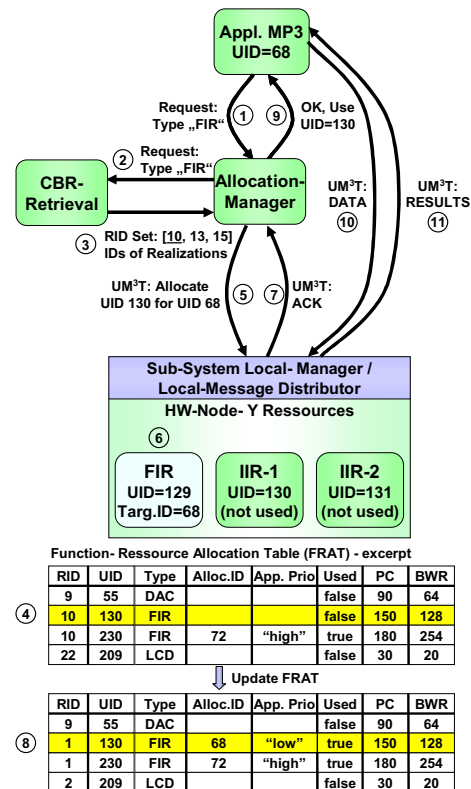


Figure 6. Resource -allocation by using UM³T

CBR-retrieval unit (2) which attempts to find a set of best matching realizations and their related RIDs (3). Depending on the found RIDs the FRAT is checked for matching non-used table entries (4). Allocated resources get only de-allocated if no free resources were found and the requesting application's priority overrides the allocated resource's application priority. In the next step the allocation manager will allocate the function module by

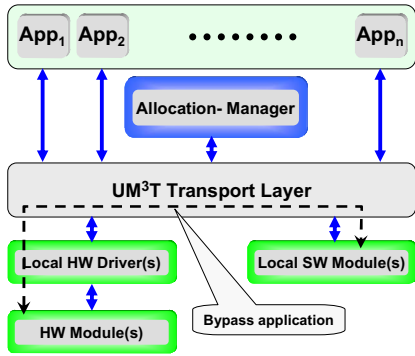


Figure 7. System view - UM³T-layer

sending an allocation command through UM³T to the local sub-system manager where the resource gets configured and prepared to run (5)(6). The local sub-system manager responds by sending a positive acknowledge via UM³T to the allocation manager (7) who updates its FRAT (8). Finally the successful allocation will be reported back to the calling application (9) which in return can communicate directly with its allocated function (10)(11).

The universal multiplexing multi-channel message transceiver (UM³T) concept is basically a node-bridging packet oriented routing protocol mechanism. It is a central system layer which multiplexes the inter-module and application \leftrightarrow system \leftrightarrow module communication (see figure 7). The used data frame format was derived from the CAN message data payload format, since CAN is a central bus of the demonstrator so that data-frame transformations between different physical ports can be simplified due to this fact. An UM³T-data-frame consists of a target UID, source UID, data length and up to 8 data bytes. Since CAN furnishes this UM³T-frame with its own headers, and provides low-level error-detection and retransmit-services this does not need to be handled by UM³T, although corresponding high-level services are conceivable for future versions. UM³T performs the conversion and forwarding of incoming buffered data packets to other communication ports (e.g. shared memory, UART and other peripherals) which are connected to the same local device node. The packet forwarding is performed by the corresponding low-level port drivers, bypassing communication without involving the application layer (see figure 8 top). Local UM³T port tables are used containing UID-ranges and their corresponding output ports (see figure 8 top). The UID-range identifies all function-modules and sub-system managers which can be (in-)directly reached through the given output port. This way, devices which don't have a direct connection to a central system bus, get the possibility for sending their data to their destinations without having to know about the route to be taken. This is an interesting aspects for integrated systems communicating through bridged networks-on-chip [2].

Another benefit of UM³T that we figured out is its potential use for bus-diagnosis purposes. Instead of buying expensive CAN-diagnosis hardware we can use for example the cheap Spartan-board and its UART/ UM³T connection for diagnostic measurements on the connected CAN-bus. The use of UID ranges offers potentials concerning an optimization of the port mapping tables' size and access speed by organizing the heterogeneous network as a hierarchical spanning tree. This way every sub-tree gets its own unique UID-range. Additionally it is possible to subdivide the UID into a node-ID and a sub-ID for addressing the local node's manager and sub-modules. The UM³T port table mapping and forwarding procedure is rather simple and can be potentially moved from software implementation into a dedicated interface IP-block if the network node does not provide processor resources. Every legal packet which arrives at its final destination node will be forwarded to the node's local message distributor, a sub-unit of UM³T. There a local table is used for assigning the packet to its destination function-module (see figure 8).

It should be noted here that the CAN-bus protocol does a one-to-many communication where the sender and not its destination is identified in the CAN-packet header, so

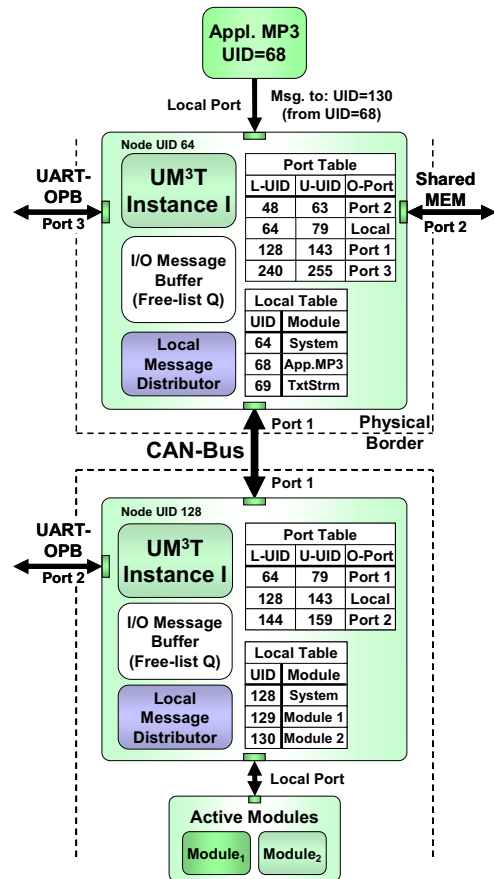


Figure 8. UM³T Data forwarding after allocation

each node connected to CAN-bus has to accept all arriving packets from the bus. This overhead can be easily handled by UM³T which checks only the UM³T-headers target UID on its local port map. If input port and output port are equal the incoming packet can be dropped for preventing from packet duplication. Broadcast communication is currently not supported by UM³T since most data dependencies between applications and their sub-functions are simple single layered trees than complex hierarchic structures. Nevertheless we conceive as a future extension sub-modules that may allocate other sub-functions by themselves, creating this way chains and trees. As shown in figures 2, 3, 4 we implemented the demonstrator as a heterogeneous network. The system was tested by applying different application request scenarios. We used the laptop computer for inserting text (1) and audio (2) data streams into the connected RS232-COM (1) (via Spartan-board) and CAN channels (2). Both PowerPC processor's provided the handling applications that reserved the needed peripheral output and display resources by using the allocation management service. After their reservation the incoming streaming data were directly sent via UM³T to their output destinations (LCD-display, low-rate 16 kHz WAV-out). We programmed on the laptop a high priority CANoe-based application that could alternately request and release the already allocated resources for its own data streams. Each time we claimed through that application one or both output peripherals we could observe the immediate switching of the output streams. As soon as we released the seized resources the previous owners regained the output devices and continued their job. A buffering of interrupted streams, although possible, was not implemented in this version. Just to test and show that communication of third party applications is not interfered an additional application (on PPC₁) performed distributed calculations by using a reserved hardware-IP on the Spartan-board. The results were sent back to their calling application and their sequential order and correctness were verified. Exclusive measurements of the shared memory channel between PPC₀ and PPC₁ showed a possible peak data rate of 195 kB/s. Although table 3 shows that much faster CAN-data rates are possible in principle we had to reduce the bus-speed to 280 kBit/s (35 kB/s), since during our tests we identified our self-designed CAN-bus connector-cable as bottleneck, which prevented us from doing other stress tests on the system. Apart from this problem we could prove the operativeness and feasibility of our concept.

5. Implementation Results

The tables 1-4 give an overview on the needed FPGA resources on Virtex II pro and Spartan-3. Because of the larger size of Virtex II pro we still have plenty of free CLB resources left (approx. 67 %), that we can use for

Table 1. XC2VP30 synthesis results

External IOBs	94 out of 556	16%
PPC405s	2 out of 2	100%
RAMB16s	91 out of 136	66%
SLICES	4532 out of 13696	33%
BUFGMUXs	7 out of 16	43%
DCMs	2 out of 8	25%
JTAGPPCs	1 out of 1	100%
TBUFs	8 out of 6848	1%
Max frequency	101 MHz	

Table 2. Spartan-3-400 synthesis results

External IOBs	85 out of 173	49%
RAMB16s	9 out of 16	56%
SLICES	3300 out of 3584	92%
SLICEMs	356 out of 1792	19%
BUFGMUXs	2 out of 8	25%
MULT18X18s	3 out of 16	18%
Max frequency	52 MHz	

Table 3. CAN IP resource usage

SLICES (<i>XC2VP30</i>)	1000 out of 13696	7%
SLICES (<i>Spartan 3-400</i>)	855 out of 3584	23%
External IOBs	3 on both FPGA types	
RAMB16s	3	"
BUFGMUXs	2	"
Max frequency (<i>XC2VP30</i>)	124 MHz (standalone)	
Max frequency (<i>Spartan 3-400</i>)	89 MHz (standalone)	

Table 4. Overall size of executables (KB)

ML 310 Virtex II pro PPC0	295 kb (Kernel approx. 60 %)	
ML 310 Virtex II pro PPC1	294 kb	"
Spartan-3 MicroBlaze	150 kb	"

other purposes. On the other hand the reached maximum frequency of 101 MHz is very close to the on-board clock. Table 2 shows that the resources of Spartan-3 are completely exhausted. Since we mapped a MicroBlaze processor, CAN-IP and small test-peripherals on that chip we had to spend some efforts until synthesis and place & route were successful. The standalone synthesis of our adapted CAN-IP cores (see also table 3) showed that they can be operated at even higher frequencies. Table 4 shows the size of the different executables on PowerPC and MicroBlaze. Although PPC₀ and PPC₁ are configured with different applications and system services the code sizes are only slightly different, since both use the same OS- and UM³T-implementation. Compared to that results the MicroBlaze-implementation consumes only the half size of code which might be caused by other compiler settings.

6. Conclusions and Outlook

This contribution presented a system concept of interconnected heterogeneous hardware/ software components that can dynamically allocate and release system resources depending on priorities and QoS demands. We introduced and demonstrated a first version of the UM³T-protocol which allows indirect

communication between the system's functional components regardless of location and local physical layer properties. It should be noted that UM³T can be flexibly extended by other bridge modules supporting other standards like LIN, USB or Ethernet. Additionally it is conceivable to move later the UM³T functionality into HW-accelerated IP-blocks, including message buffering, translation and forwarding. We plan to extend the current resource reservation process in a way that sub-module chains and trees can be created, which implies changes to the allocation/ de-allocation mechanism concerning aspects like locality/ neighborhood relations of resources and garbage collection issues. To get closer to our goals it is furthermore intended to include partial run-time reconfigurable hardware modules on our Virtex II pro platform which are interconnected by a packet oriented local network on-chip as a part of UM³T. Finally the project shall be driven into the direction of the highly topical organic computing paradigm providing new features like self-adaptation, self-healing and self-optimization.

7. References

- [1] A. Aamodt and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *Int'l Journal on Intelligent Automation and Soft Computing*, vol. 7, no. 1, 1994, pp. 39-59.
- [2] L. Benini and G. De Micheli, "Networks on chips: a new SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, 2002, pp. 70-78.
- [3] G.J. Brebner, "A Virtual Hardware Operating System for the Xilinx XC6200," in *Proc. 6th Int'l Conf. Field Programmable Logic Smart Applications, New Paradigms and Compilers (FPL'96)*, LNCS vol. 1142, R.W. Hartenstein and M. Glesner, Edt., Springer-Verlag, 1996, pp. 327-336.
- [4] K. Danne, "Operating Systems for FPGA Based Computers and Their Memory Management," *Proc. Organic and Pervasive Computing, Workshop (ARCS'04)*, Köllen Verlag, 2004, pp. 195-204.
- [5] Federal Aviation Administration, 2005, <http://www.faa.gov/>
- [6] J.C. Ferreira and M.M. Silva, "Run-Time Reconfiguration Support for FPGAs with Embedded CPUs: The Hardware Layer," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS'05) (CD-ROM)*, IEEE CS Press, 2005, pp. 165a-165a.
- [7] S.C. Goldstein et al., "PipeRench: a Reconfigurable Architecture and Compiler," *IEEE Computer*, vol. 33, no. 4, 2000, pp. 70-77.
- [8] B. Griese, S. Oberthür and M. Pormann, "Component Case Study of a Self-optimizing RCOS/RTOS System: A Reconfigurable Network Service," *Proc. Int'l Embedded Systems Symp. (IESS)*, Springer-Verlag, 2005, pp. 267-277.
- [9] M. Hübner et al., "Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems," in *Proc. 14th Int'l Conf. Field Programmable Logic and Applications (FPL'04)*, LNCS vol. 3203, J. Becker, M. Platzner and S. Vernalde, Edt., Springer-Verlag, 2004, pp. 1037-1041.
- [10] M. Hübner, K. Paulsson and J. Becker, "Parallel and Flexible Multiprocessor System-On-Chip for Adaptive Automotive Applications based on Xilinx MicroBlaze Soft-Cores," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp., Reconfigurable Architectures Workshop (IPDPS'05)*, IEEE CS Press, 2005, pp. 149-154.
- [11] J. Labrosse, *MicroC/OS-II: The Real-Time Kernel*, CMP Books, 2002.
- [12] Z. Li, K. Compton and S. Hauck, "Configuration Caching Management Techniques for Reconfigurable Computing," *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, IEEE CS Press, 2000, pp. 22-36.
- [13] T. Marescaux et al., "Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs," in *Proc. 12th Int'l Conf. Field Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream (FPL'02)*, LNCS vol. 2438, M. Glesner, P. Zipf and M. Renovell, Edt., Springer-Verlag, 2002, pp. 795-805.
- [14] Micrium Technologies Corporation, "Micrium - μ C/OS-II RTOS," 2005, <http://www.ucos-ii.com/>.
- [15] MISRA, The Motor Industry Software Reliability Association, 2005, <http://www.misra.org.uk/>
- [16] V. Nollet et al., "Hierarchical Run-Time Reconfiguration Managed by an Operating System for Reconfigurable Systems," *Proc. 3rd Int'l Conf. Eng. Reconfigurable Systems and Algorithms (ERSA'03)*, CSREA Press, 2003, pp. 81-87.
- [17] Robert Bosch GmbH, "CAN Specification," 2003, <http://www.semiconductors.bosch.de/pdf/can.pdf>.
- [18] M. Ullmann et al., "An FPGA Run-Time System for Dynamical On-Demand Reconfiguration," *Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS'04)(CD-ROM)*, IEEE CS Press, 2004, pp. 135-142.
- [19] M. Ullmann et al., "On-Demand FPGA Run-Time System for Dynamical Reconfiguration with Adaptive Priorities," in *Proc. 14th Int'l Conf. Field Programmable Logic and Applications (FPL'04)*, LNCS vol. 3203, J. Becker, M. Platzner and S. Vernalde, Edt., Springer-Verlag, 2004, pp. 454-463.
- [20] M. Ullmann, W. Jin and J. Becker, "Hardware Support for QoS-based Function Allocation in Reconfigurable Systems," *Proc. Conf. Design, Automation and Test in Europe (DATE '05)*, IEEE CS Press, 2005, pp. 259-264.
- [21] Vector CANTech Inc., "CANoe," 2005, <http://www.vector-cantech.com>.
- [22] M. Vuleti et al., "Operating System Support for Interface Virtualisation of Reconfigurable Coprocessors," *Proc. Conf. Design, Automation and Test in Europe (DATE '04)*, IEEE CS Press, 2004, pp. 10748.
- [23] M. Vuleti, L. Pozzi and P. Jenne, "Seamless Hardware-Software Integration in Reconfigurable Computing Systems," *IEEE Design & Test of Computers*, vol. 22, no. 2, 2005, pp. 102-113.
- [24] J.W. Williams and N. Bergmann, "Programmable Parallel Coprocessor Architectures for Reconfigurable System-on-Chip," *Proc. IEEE Int'l Conf. Field-Programmable Technology (FPT'04)*, IEEE CS Press, 2004, pp. 193-200.
- [25] J.W. Williams and N. Bergmann, "Embedded Linux as a Platform for Dynamically Self-Reconfiguring Systems-on-Chip," *Proc. 4th Int'l Conf. Eng. of Reconfigurable Systems and Algorithms (ERSA'04)*, CSREA Press, 2004, pp. 163-169.
- [26] Xilinx, 2005, <http://www.xilinx.com/>.