Implementation of a Reconfigurable Hard Real-Time Control System for Mechatronic and Automotive Applications

Steffen Toscher; Roland Kasper; Thomas Reinemann
Institute of Mobile Systems
University Magdeburg, Germany
(steffen.toscher; roland.kasper; thomas.reinemann)@mb.uni-magdeburg.de

Abstract

Control algorithms implemented directly in hardware take advantage of parallel signal processing. Furthermore, implementing controller functionality in reconfigurable hardware facilitates modification of controller structure and parameters during run-time. In this paper, we introduce an implemented and tested reconfigurable hard real-time control system based on an FPGA device. It supports dynamic partial reconfiguration of controller functionality by self-reconfiguration mechanisms. Self-reconfiguration is performed using an internal configuration interface. We also present sophisticated on-chip and off-chip communication solutions. Specification of controller functionality involves Finite State Machines (FSMs) and comprises parts of the distributed communication and reconfiguration solution.[1]

1. Introduction

With the advances in reconfigurable hardware, functionality of mechatronic and automotive control systems can be implemented in FPGAs. The use of reconfigurable devices in mechatronic and automotive applications, such as controllers for electrical drives and engine control units, enables the replacement of functionalities, algorithms and parameters in hardware during run-time. This decreases the amount of utilized silicon area and increases system flexibility. The implementation of a control system which benefits from reconfigurable hardware is still highly problematic since facts like on-chip and off-chip communication, reconfiguration management and specification of functionality have to be taken into consideration.

In this paper, we present an implemented and tested reconfigurable hard real-time control system supporting the modification of controller functionality and parameters while the system is running. Controller functionality is implemented directly in hardware. The target device is a Xilinx Virtex-II FPGA. No micro-processor on or off the FPGA device is used. Modification of functionality and parameters is achieved by dynamic partial reconfiguration of the device through its Internal Configuration Access Port (ICAP) enabling self-reconfiguration of the system. Implementation of the system follows the Xilinx modular design flow [1]. This design flow implies that reconfigurable portions of the design are referred to as modules. A dedicated bit serial [2] communication system ensures data transfer and integrity regarding the reconfiguration process. Thus, the hard real-time control system we present in this paper assures continuity in system behavior and output signals.

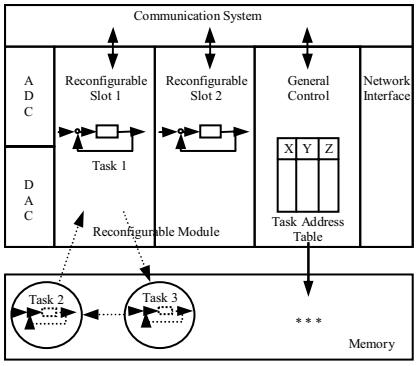

Figure 1: System structure

Figure 1 shows the structure of the hard real-time control system. It consists basically of two or more reconfigurable slots and a static general control module. The dedicated communication system connects both slots and the general control module and is responsible for data transfer with regard to reconfiguration control. The general control module implements common monitoring and control functions and initiates the reconfiguration process. The reconfigurable slots implement signal processing (controller functionalities) organized in modules and tasks as described in the next paragraph of this paper. Aside from reconfigurable slots the hard real-time control system comprises communication solutions for off-chip communication with a higher-level control system network and an interface to analog process signals. The analog-digital converter is split into an on-chip and an off-chip part.

[1] *This work was supported by the Deutsche Forschungsgemeinschaft (DFG).*

2. Specification and reconfiguration control

The structure of the hard real-time control system depends on specification and implementation issues. Specification follows the rules of data flow and control flow used in common tools like Matlab/Simulink or ASCET/SD. Controller functionality is specified and implemented using modules and tasks. A module is a fixed part of the target device whereas tasks represent the different signal processing functionalities of a module. Although there is only one active task in a module at the same time, there can be more than one task loaded in a module, thus consuming more chip area. The complexity of a global reconfiguration management [3] is reduced by a distributed reconfiguration control mechanism. Implementation of modules and tasks considers real-time behavior regarding signal processing as well as reconfiguration.

Specification of modules and tasks combines data flow and control flow specification methods. Data flow is usually specified by signal diagrams which consist of signals and blocks. Control flow, which is switching between functionalities and states of operation, is specified using Finite State Machines (FSMs) [4]. State actions and transition conditions are an important connection between data and control flow, since they can be triggered or represented by signal diagrams. Specification of tasks employs both signal diagrams and FSMs whereas module specification requires only control flow methods to switch between tasks or load different tasks.

Reconfiguration of a module means basically to load a new task on the modules chip area. This is specified by a Module-FSM, which is usually dependent from physical process values, e.g. the revolution speed of an electrical drive. Figure 2 gives an example of a Module-FSM. This special Module-FSM consists of 3 states and each of the states represents a single task of the module that has to be loaded on a certain condition. If the time needed for reconfiguration is longer than the process signal sample period, then the new task has to be loaded some time before. This is called prefetching of tasks and enables the system to switch between tasks without loosing samples.

The structure of the Module-FSM is known at compilation time, and it solely initiates loading and prefetching of one or more tasks. Due to these facts, the Module-FSM can be split in its single states. This enables the distributed implementation of the Module-FSM in the single corresponding tasks of the module, where each task implements the state action and the out bound transition of the Module-FSM. In order to mark the active state (task) at a certain time, the reconfiguration control system uses global identifiers for all tasks. This global identifier is called T-Marker. The T-Marker is exchanged via the global communication system which connects all modules.

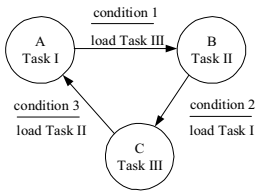

Figure 2: Module-FSM example

As only one task per module can be active, it is necessary to establish a control mechanism that reports and triggers the states of the specific task. These states are identical for all tasks in all modules and are comprised in a Task-FSM which is again identical for all tasks. Figure 3 shows the Task-FSM. It controls the following states of a task: **not loaded**, **ready** and **active**. Each Task-FSM represents a state and the out bound transitions of the Module-FSM.

Modules and tasks including Task-FSM and Module-FSM are implemented directly in hardware logic. This improves performance, enables real-time signal processing and reduces hardware area in contrast to a processor-based implementation.

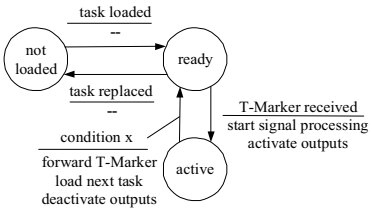

Figure 3: Task-FSM

A module can comprise one or more slots depending on the need to prefetch tasks. A task is directly mapped onto a slot of its module. The process signal inputs of a specific task are therefore attached to the slot the task is loaded in. Furthermore, each task implements a part of the distributed communication system that realizes transfer of reconfiguration related data like the T-Marker. The actual functionality of a task depends on application requirements and is only limited by the allocated chip area. The static general control module is also implemented directly in hardware including a part of the distributed communication solution. Aside monitoring and controlling activation and reconfiguration of tasks, the general control module implements the connection between a higher-level control system network and the hard real-time control system. In order to initiate and control reconfiguration of single slots, the general control module also implements memory accesses, handles configuration data and realizes the reconfiguration mechanism itself.

3. On-chip and off-chip communication

On-chip communication comprises basically communication regarding reconfiguration and signal processing. A dedicated global communication system provides reconfiguration related communication, especially

figuration related communication, especially exchanging the T-Markers mentioned above. The communication system connects all modules. It consists of three functional units, a persistent hard macro and two different subscriber connections. The subscriber connection of the general control module is different from the others, because it contains an additional watchdog to ensure run-up and rerun-up if communication has been disturbed.

All subscribers build a shift register, each subscriber stores one bit of the message frame and shifts it to the next subscriber on each clock. The difference between the frame length and the number of subscribers is balanced by a shift register that is implemented in the general control module. If a task wants to activate another task, it simply sends a T-Marker. This T-Marker is received by all other slots. Each Task-FSM compares the received and the expected T-Marker (hard coded). If this comparison is true, its task is activated. When a slot is under reconfiguration, the communications system has to continue to work without disturbance. Therefore its multiplexers and registers are implemented in kind of a hard macro. Placement is done in a way that these components are located within the same columns as the reconfigurable area to which they belong to. This avoids a module boundary crossing routing.

Off-chip communication includes interfaces to a higher level control system or network and to process signals. The selected network interface is the Universal Serial Bus (USB) 2.0 which is a current and widely used standard. Other standards like Profibus or Ethernet can be implemented alternatively. The USB interface is implemented as a part of the general control module and is therefore not dynamically reconfigurable. However, the network interface can be replaced by a reconfiguration of the whole FPGA device depending on the application's requirements. The hard real-time control system uses an external USB controller, the EZ-USB FX2 device by Cypress. This high speed USB peripheral controller implements the physical layer, the serialization of data and several other tasks. It connects to the control system's network interface by using special FIFO memories on the USB controller. The network interface on the FPGA is responsible for controlling the FIFOs. Furthermore, the network interface checks and caches any data that is received or is to be sent. Further processing of received data depends on its properties. Configuration data (bit streams) can be used for instant reconfiguration of a slot or can be stored in internal respectively external memory to be used for a later reconfiguration.

The process interface of the reconfigurable hard real-time control system consists basically of a simple pulse width modulation for digital-analog conversion and an analog-digital converter (ADC). However, today's FPGAs lack any on-chip analog-digital conversion. Furthermore, the sampling frequencies are usually in a range of some ten kHz regarding mechatronic and automotive applications. Fulfilling the sampling theorem requires a low-pass filter with a very small transition band. An analog design and implementation of such a filter is very costly. Modification of the filter or an internal down sampling is necessary when the sample frequency changes. Due to these problems and the high resource requirements of parallel analog-digital converters Delta-Sigma-ADCs [5] were developed. They consist of a simple off-chip analog front-end, the Delta-Sigma-Modulator, including a subtraction unit (delta), integrator (sigma), comparator and quantizer and a digital filter. The Delta-Sigma-Modulator converts the analog input signal into a binary pulse-code modulated (PCM) signal representing the average value of the analog signal. The sampling frequency of the PCM signal is considerably higher (minimum: x32) than the actual sampling frequency of the control system. This ratio is called Over Sampling Ratio (OSR). Obtaining the actual sampling frequency of the control system with a corresponding resolution involves filters and decimations. The resolution logarithmically increases depending on the OSR. Thus, resolution and sampling frequency can be adjusted by modification of the filter and decimation algorithms without changing the analog off-chip design. The digital filter of the Delta-Sigma-ADC is implemented in the FPGA. Therefore, modification of filter structure and parameters can be accomplished using (static) reconfiguration.

4. Reconfiguration process

Since controller functionality is implemented in the tasks of a module, reconfiguration means basically the replacement of a task with another. In hardware this implies the dynamic partial reconfiguration of the slot the task is implemented in. The hard real-time system itself determines when a new task is to be loaded. Together with reconfiguration through the Internal Configuration Access Port (ICAP) this founds the self-reconfiguration feature of the hard real-time control system.

The USB interface receives configuration data (bit streams), distributes and stores it in memory. Configuration data is stored in internal memory on the FPGA device or in external memory. Internal memory (BlockRAM) enables very fast memory accesses but its amount is device specific. Upon receiving a corresponding request and T-Marker, the general control module initiates the loading/reconfiguration sequence for a specific task/slot. Configuration data is then read from its memory location and is send to the internal configuration interface ICAP.

ICAP is accessed directly by the FSMs of the general control module which are implemented in hardware. These FSMs form the Hardware-ICAP module inside the general control module. Thus, no microprocessor and API [6], IP core solution, functional block provided by Xilinx or embedded microprocessor control system is used. The Hardware-ICAP module emulates the SelectMap protocol

that is required to talk to ICAP and handles the configuration data. As just a few hardware FSMs organize the direct access to ICAP, this reconfiguration solution is very effective, fast and requires only a small amount of logic resources. Furthermore, an implementation of ICAP access directly in hardware ensures a deterministic time behavior and the real-time characteristics of the whole system.

5. Application

Typical applications of the hard real-time control system we presented in this paper are controllers for electrical drives and engine control units. These applications use generally microcontrollers to implement controller algorithms and functionality. Thus, real-time problems emerge owing to restricted computing performance and timing problems. In contrast to microprocessors, FPGAs offer a high computing performance due to parallel signal processing and a direct hardware implementation of algorithms to overcome real-time problems. By taking advantage of dynamic partial reconfiguration, new features regarding controller implementation on FPGAs can be realized. The modification of controller functionality, structure, parameters and interfaces are some of these features.

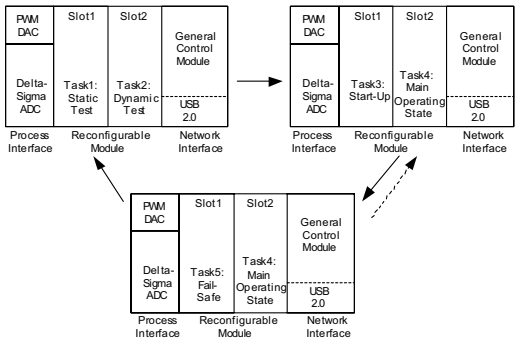

Figure 4: Application example electrical drive system

An example of an application of the hard real-time control system we presented is the run-up of an electrical drive system. Upon power-up of the electrical drive, only the general control module including the USB interface and the process interface including analog-digital and vice versa conversion must be loaded on the FPGA. The system receives now additional configuration data, the tasks, through the USB interface and stores it in memory. The tasks are consecutively loaded into the slots of the system depending on the operation states of the drive system, such as test, start-up, main operation state and fail-safe. Figure 4 illustrates this application example.

Another application example for the reconfigurable hard real-time control system is a drive control for piezo-electric actuators. An FPGA based implementation of drive control algorithms for piezo-electric actuators is described in [7]. Using the reconfigurable hard real-time control system as a platform could lead to new approaches and features.

6. Conclusions

The reconfigurable hard real-time control system we described in this paper is fully implemented and tested on a Xilinx Virtex-II FPGA device. Its main areas of application are automotive and mechatronic systems with real-time conditions. The system supports the modification of controller algorithms and functionality at run-time. All functionality is implemented directly in hardware. Thus, no microprocessor is required and the system can be flexibly implemented with a low amount of logic. Furthermore, real-time conditions are guaranteed.

The system comprises a general control module, one or more functional modules and sophisticated on-chip and off-chip communication solutions. Functional modules consist of different slots on the target device serving as placeholder for the module's tasks. The tasks finally represent the reconfigurable controller algorithms. Both modules and tasks are controlled by specific FSMs connected to the dedicated on-chip communication system. Off-chip communication comprises an USB 2.0 networking interface as well as a Delta-Sigma analog-digital converter. Reconfiguration of controller functionality utilizes the internal configuration interface ICAP and is accomplished by the general control module. Thus, the hard real-time control system is a self-reconfigurable system.

7. References

[1] Xilinx: Two flows for partial reconfiguration: Module based or difference based, XAPP290, 2004

[2] Reinemann, Th.; Kasper, R.: Delay optimization for bit serial algorithms, *International Signal Processing Conference*, Dallas, USA, 2003

[3] B. Griese, E. Vonnahme, M. Porrmann, U. Rückert: Hardware Support for Dynamic Reconfiguration in Reconfigurable SoC Architectures, *Proceedings of the FPL 2004*, Antwerpen, Belgium, 2004

[4] Toscher, S.; Reinemann, Th.; Kasper, R.: Dynamic Reconfiguration of Mechatronic Real-Time Systems Based on Configuration State Machines, *Proceedings of the IPDPS 2005*, Denver, USA, 2005

[5] Medeiro, Fernando: Top-Down Design of High-Performance Sigma-Delta Modulators, Kluwer Academic Publishers, Boston, 1999

[6] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, P. Sundararajan: A Self-reconfiguring Platform, *Proceedings of the FPL 2003*, Lisbon, Portugal, 2003

[7] Gnad, G.; Kasper, R.: A Power Drive Control for Piezo-electric Actuators, *Proceedings of the IEEE-ISIE 2004*, Ajaccio, France, 2004