FUNCTION REPLACEMENT OF HARD REAL-TIME SYSTEMS USING PARTIAL RECONFIGURATION

Thomas Reinemann, Roland Kasper
IMAT, Otto-von-Guericke-University Magdeburg
Germany
(thomas.reinemann;roland.kasper)@mb.uni-magdeburg.de

ABSTRACT

This paper describes a method of function replacement in hard real-time systems implemented with FPGAs using dynamic partial reconfiguration. Prefetching of functions is used to avoid violations of real time conditions in cases where reconfiguration time is large compared to sampling time. A finite state machine determined from the specification of the real-time system is used to control reconfiguration and prefetching. Implementations for a distributed finite state machine of each module and a distributed communication system that supports loading and activation of functions are presented.

KEYWORDS

Partial reconfiguration, hard real-time systems, distributed real-time communication system, signal processing

1. INTRODUCTION

The work described in this paper is part of the development of a new type of adaptable hard real time system using dynamic reconfiguration of FPGAs. It is related to signal flow oriented systems commonly used in feed back and feed forward control systems, for example in Electronic Control Units (ECU) in the field of automotive systems. Signal flow is a natural way to specify control systems, where physical data is taken from process inputs representing physical quantities, processed in distinct blocks connected by signal lines and then fed back to process outputs to convert them back to the physical world. Some dedicated considerations emphasize the parallelizing at block level and serializing at signal level, which leads to very efficient solutions [1].

It is common technology to replace a specific function depending on the operating points of a control system. That means that structure, parameters or the resolution and sampling rate of a control algorithm is changed depending on specific condition, e.g. an engine's revolutions per minute (rpm). If software is used for implementation, then only another function has to be called or another set of parameters will be used. Implementing controllers and signal processing algorithms directly in hardware, e.g. using FPGAs, classical approaches implement all functions needed and switch between them as necessary [2]. This results in large logic needs, which can be reduced significantly, if only the logic of the currently used function is loaded and the unused logic is stored in external memory. This can be achieved by partial reconfiguration of the FPGA's logic at runtime. In case of hard real-time systems the time T_R needed for reconfiguration has to be considered, because during reconfiguration the function's logic is not active. T_R is governed by the function's size and reconfiguration speed (FPGA clock and type). Today, reconfigurable FPGAs like the Virtex-II family offer reconfiguration times in the range of some milliseconds. T_R has to be seen in relation to the sampling time T_S, which is given by the real time needs of the control problem, and the processing time T_P, which is determined by the functions logic and the clock. The relation between T_R, T_P and T_S separates two cases.

On the one hand T_S is equal or larger than T_P+T_R. In this situation, function replacement can take place within one sampling period without loss of any data. On the other hand T_P+T_R exceed T_S. In this case one or several samples could not be processed during reconfiguration, which cannot be accepted for hard real-time systems. To avoid this, the new function has to be prefetched. An implementation of this method of function replacement will be presented in this paper.

Implementation takes place using a Xilinx Virtex-II FPGA. The design flow of partial reconfiguration [3] is based on the Xilinx Modular Design methodology [4]. Reconfigurable functions are referred as modules. Each module can have different types, which are loaded to reconfigure the area/slots assigned to this module. All types (TI, TII, TIII …) of a module must have the same interface, which means the same size and location of the ports. Generally, the interface signals can be divided in two groups: control signals and process signals. Process signals carry the information related to signal processing. Control signals are supposed to control the loading and activation of functions/types. This is necessary, because multiple sources for a process signal can exist, if more than one type is loaded at the same time, but only one is allowed to drive a process signal. Signals crossing module boundaries need bus macros, which are used to control driving of process signals.

Bit serial processing and transmission comes into operation for signal processing and implementation of control algorithms [5]. This results in low logic effort and a small number of lines needed to implement process signals. Therefore it is not necessary to use busses [6] or networks [7] for communication purposes of the control system or the signal lines.

2. FUNCTION REPLACEMENT

2.1 Module FSM

The dependency which module type has to be loaded is controlled by a finite state machine for each module (M-FSM). Figure 1 shows an example of a FSM representing a module having three types (TI, TII and TIII). The transition conditions typically are related to physical items (e.g. revolution speed of a motor). They are part of the controller’s specification and define its operating points. In this example the FSM covers three different operating points, each implemented by a type. In this example, defined by the structure of the FSM, only one additional type needs to be prefetched in each state to switch to the next state without delay. That means only two of three types have to be loaded at the same time and the module needs two slots within the FPGA, where its types are loaded. Prefetching a type can be seen as a transition action. It is important that loading TIII is a transition action of state A that is executed in parallel to the activation of state B.

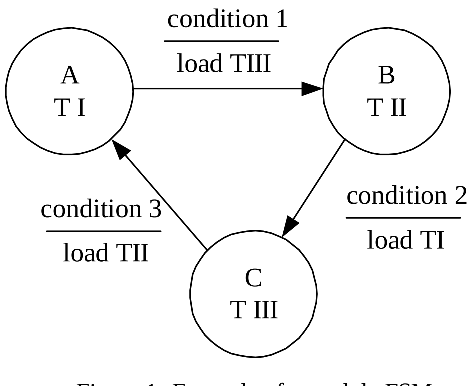

Figure 1 Example of a module FSM

Extending the example of figure 1 with an additional transition from A to C, means that in state A type TI has to be loaded and active and types TII and TIII have to be prefetched and will require logic too. In this situation dynamic reconfiguration will have no advantages, whereas the situation where state B or C is active will be as before.

Typically a control system implements more than one FSM or module. To analyze the usability of dynamic reconfiguration for a specific application the relation between the maximum number of accessible states N_R and the total number of states N_S of a module is relevant. Only in cases where $N_R+1<N_S$ module prefetching is useful and results in logic savings.

In general, each transition disposes the activation of one type and loading of zero (if already prefetched), one or more other types of the module it belongs to, depending on the structure of the M-FSM that is known at compile time. Consequently the M-FSM can be broken down to parts implemented by the types of the module. This distribution requires a method to transfer the active state of each M-FSM from one type to the next active state. For this purpose a global marker for each type (T-marker) is used.

2.2 Type-FSM

Each type of a module can have different states

- Not loaded: stored in external memory, needs no logic,
- Ready: prefetched, listening on control input, but not processing data or driving outputs,
- Active: loaded, listening on control input, processing data and driving outputs,

which can be represented by a FSM (T-FSM) given in figure 2. After a type is prefetched and the reset (local to each slot) becomes inactive, the T-FSM switches to "ready". Now the type listens for its T-marker but keeps the signal outputs inactive. To each type belongs exactly one T-Marker, which can is defined by the static structure of the M-FSM. After receiving its T-marker the T-FSM switches to "active" and activates its signal outputs. From now on it processes information and sends data to signal outputs. If "condition x" of the M-FSM fires the T-FSM switches back to "ready", two actions take place:

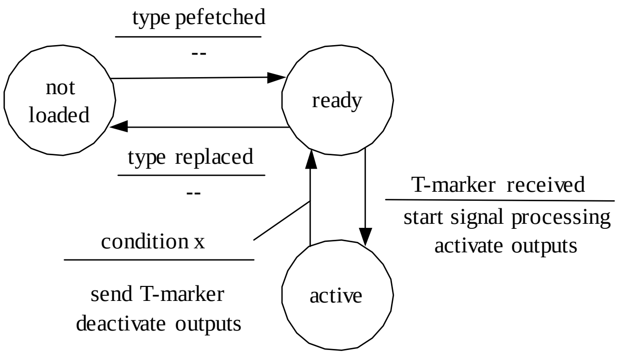

Figure 2 Type-FSM

- Deactivate process signal outputs,
- Send the T-marker of the next active type.

In cases where the type is replaced by a prefetched type its state changes to "not loaded". To facilitate implementation M-FSM and T-FSM are packed together. This is straightforward, because the "active" to "ready" transition and action of the T-FSM has to be implemented only once and actions of all transitions of the M-FSM have only to be augmented by the actions of the corresponding transitions of the T-FSM.

2.3 Communication and Control System

T-markers are exchanged via a very efficient communication system. Since communication takes place across module boundaries, bus macros come into operation. Xilinx bus macros utilize long lines, which are driven by tristate buffers. To avoid a complex arbitration method, a distributed shift register has been used as base of a communication system.

Figure 3 shows the structure of the communication system, straddling all slots of the FPGA. Each module represents a subscriber. All together build a shift register, where each subscriber stores one bit of the message

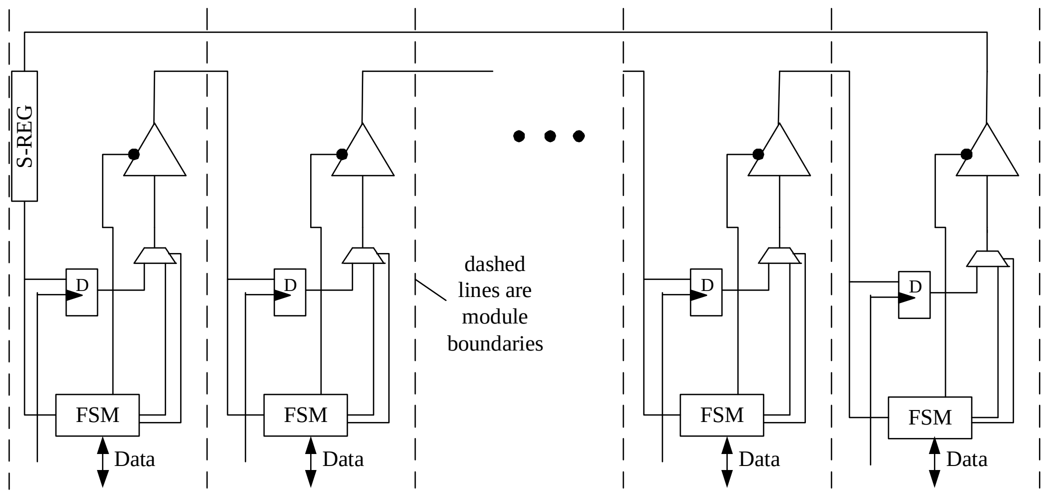

Figure 3 Basic structure of the communication system

and shifts it to its neighbor on each clock. An additional shift register (S-REG) is required to balance the difference between the message length and the number of subscribers. The message has the following structure:

- Start delimiter: 8 bits
- Used flag: 1 bit,
- Data field: fixed length to represent T-markers.

A subscriber is permitted to write data only if the used flag is not set. If the used flag is set and the subscriber detects its T-marker, the used flag will be cleared. It has to be guaranteed that the message is shifted through the complete register within sampling period T_S. With an operation frequency of 40 MHz and a message length of 16 bits, sampling periods smaller than 0.4 µs are possible.

Loading of module types is organized by a general control module GCM, which is a static module that receives all messages and implements the additional shift register. The GCM has access to a table containing all possible T-markers together with their location and size of bit streams in external memory. Furthermore, the table stores information which types have to be prefetched. After receiving a T-marker the GCM reads the bit streams to be loaded and sends them to the internal reconfiguration access port (ICAP). The type table represents static data that can be determined at compile time from the FSM structure. GCM also adapts lines between the distributed registers of the communication system to bypass a slot during reconfiguration.

2.4 Implementation and Slots

A reconfigurable system based on the method described in this paper has been implemented on a Virtex-II. The exchange of messages via the communication system, reading of a configuration and sending it to ICAP have been simulated and validated on functional level. The complete testbench is described in VHDL. Prefetching of types influences implementation, because the modular design flow [3] is planed to implement different types of a module always into the same slot. But the proposed method expects multiple types of a module to be loaded simultaneously. The allocation of a slot by a certain type can be static or variable, depending on the structure of the M-FSM. In case of variable allocation a type may be loaded into different slots. To enable this, the type can be implemented for all possible slots, which results in multiple bit streams and a large amount of memory to store them. Another way is to implement the type only once and to manipulate the bit stream before loading in the GCM. This is possible using tools like PARBIT [8]. In this case a module needs unified slots with the bus macros placed on the same relative slot position, which is guaranteed for this application. Special bus macros have been developed, which straddle more than two slots,

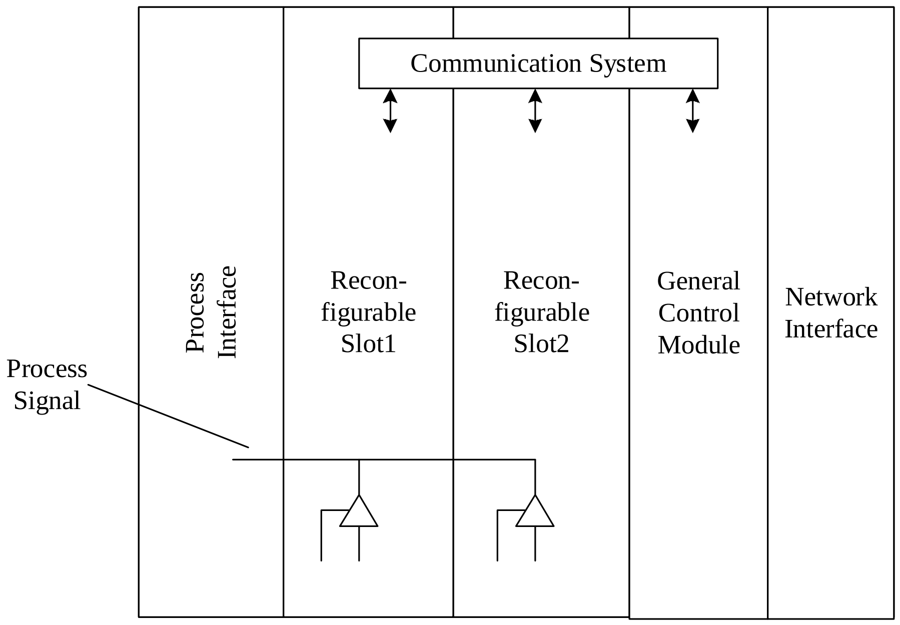

Figure 4 Partitioning and implementation of a dynamic reconfigurable system on a Virtex-II

to enable access to one process signal destination from multiple slots. The dynamic reconfiguration of the communication system and its bus macros have been implemented and tested.

Figure 4 shows the implementation example for the reconfigurable system, built from a fixed part (General Control Module, Network Interface, Reconfiguration Controller and Process Interface) and two reconfigurable slots. Both slots can host either an active or prefetched type mutually exclusive. Furthermore an example of a process signal is shown, which can be driven from both slots, depending on which slot is active. The activation information is exchanged as a T-Marker via the communication system. It has three subscribers, General Control Module and both reconfigurable slots.

3. CONCLUSION

The presented method allows function replacement in hard real-time systems implemented directly in FPGA hardware. The needed logic amount is reduced by dynamic reconfiguration and enables the implementation of more complex algorithm into smaller FPGAs. Prefetching of function logic is used to guarantee hard real-time conditions, even in cases where reconfiguration times are large compared to sampling times. The distributed communication system used to synchronize the distributed state machines is very fast and needs a minimum of logic. Future work will be focused on questions of dynamic slot allocation in situations, where static concepts will not offer a satisfying solution.

REFERENCES

[1] Andre DeHon, John Wawrzynek: Reconfigurable computing: what, why, and implications for design automation, *Proceedings of the 36th ACM/IEEE conference on Design automation conference,* 1999, ISBN 1-58133-109-7, New Orleans, Louisiana, United States, ACM Press, p. 610 – 615.

[2] Gand, G.; Kasper, R.; 2004, A Power Drive Control for Piezoelectric Actuators, *Proceedings of the IEEE-ISIE 2004*, Ajaccio, France, pp. 963-968

[3] Xilinx: 2004; Two flow for partial reconfiguration: Module based or difference based, XAPP290

[4] Xilinx: 2004; Xilinx Development Systems Reference Guide;

[5] Kasper, R.; Reinemann, Th.: 2000, Gate level implementation of high speed controllers and filters for mechatronic systems, *Mechatronic Workshop 2000*; Krakau, Poland

[6] Huebner, M.; et. al.: 2004, Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems, *FPL 2004,* Leuven, Belgium, pp. 1037 – 1041.

[7] Marescaux, T et. al; 2002, Interconnection networks enable fine-grain dynamic multitasking on FPGAs, *FPL 2002;* FPL 2002, Montpellier, France, p. 795

[8] Edson L. Horta: 2002; Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration; *Design Automation Conference (DAC);* New Orleans, LA, http://www.arl.wustl.edu/projects/fpx/parbit/