DYNAMIC RECONFIGURATION OF MECHATRONIC REAL-TIME SYSTEMS BASED ON CONFIGURATION STATE MACHINES

Roland Kasper; Thomas Reinemann
Institute of Mechatronics and Drive Systems
University Magdeburg, Germany
(thomas.reineman;roland.kasper)@mb.uni-magdeburg.de

ABSTRACT

Data flow and FSMs are used intensively to specify real-time systems in the field of mechatronics. Their implementation in FPGAs is discussed against the background of dynamic reconfiguration which can be used to reduce the amount of logic resources of the FPGA. Bit serial algorithms come into operation for data flow implementation. Particular attention has been paid for its synchronization in dynamic reconfigurable systems. Typically, a control system uses different functions depending on its operating state; dynamic reconfiguration is used to switch between them. This process is controlled by FSMs which are extension of the FSMs already used for specification of the real-time system. Dynamic loading of new logic is triggered in a distributed way by the active states of all FSMs and executed in a central way by a general control module. Synchronization and message passing is guaranteed by a distributed communication system.

KEYWORDS

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

1. INTRODUCTION

This paper presents a new type of adaptable real-time system using dynamic reconfiguration of FPGAs. It's focus is on 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. It relies on the fact that 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].

Control systems step through different operating states, e.g. self test and operation. Since these will not run simultaneously the function's logic may be replaced. If software is used for implementation, then only another function has to be called. Realizing controllers and signal processing algorithms directly in hardware, classical approaches implement all functions needed on the FPGA 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. Partial reconfiguration requires a certain area or slot reserved for reconfiguration and introduces boundaries within the FPGA. These boundaries have to be crossed via special lines at known fixed communication end points, accessible by all slot's configurations. Complexity can be reduced by serial communication based on busses [3] and networks [4].

Implementation takes place using Xilinx FPGAs. The design flow of partial reconfiguration [5] is based on the Xilinx Modular Design methodology [6]. 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.

2. SPECIFICATION OF MECHATRONIC CONTROL SYSTEMS

Mechatronic control systems usually are specified as a combination of data flow and control flow parts that interact closely. Well known tools like Matlab/Simulink and ASCET/SD [7] support specification of both parts separately. Operation control signals generated by finite state machines (FSM) are used to affect the data flow that typically is defined by a signal diagram. Originally dedicated to the specification of control software, recent development of these tools, e.g. System Generator for DSP [8], also allow to specify control systems implemented directly by hardware on an FPGA. On the other hand modern languages like UML offer the possibility to specify controllers for mechatronic systems in a similar way [9].

2.1 Data flow specification

Data flow in control systems can be specified very well by signal diagrams. They are composed of signal lines, representing data flow, and distinct blocks, representing functions for signal processing. Blocks are working in parallel while being synchronized by their input signals associated with a specific clock. Figure 1 shows a cascade controller as an example of a signal diagram. Blocks themselves can contain either signal diagrams (in this case they are referred as structure blocks) or elementary processing elements like adders, subtractors, gains and characteristic tables or other operators etc. A block has well-defined interfaces, divided into inputs and outputs. Thus the signal diagram given in figure 1 represents a structure block

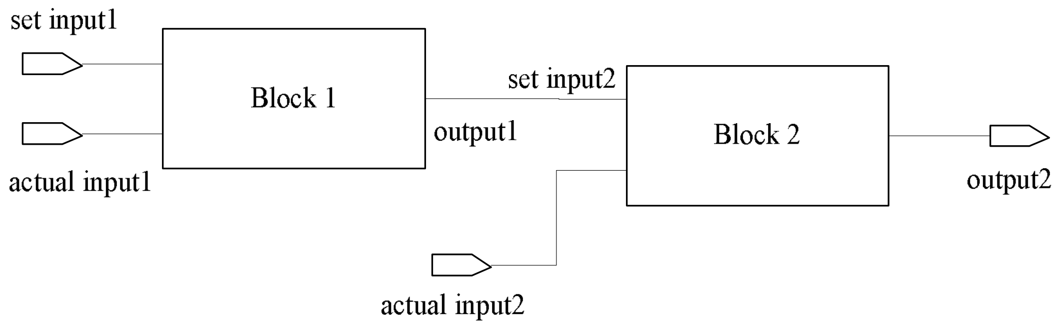

Figure 1 Cascade controller signal diagram

with three inputs and one output. All blocks together built a hierarchy with the top level structure block on the highest level. The lowest level is always built by elementary processing elements representing the algorithms that have to be implemented in logic

2.2 Control flow specification

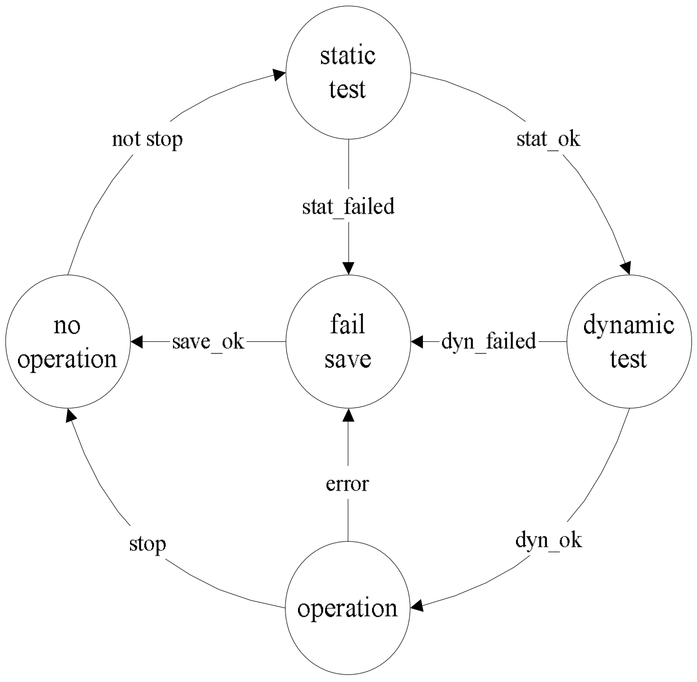

Figure 2 Run-up and operation FSM

Control flow specification can be done in a structured way by FSMs. As a typical example figure 2 shows an FSM which defines a typical system run-up and operation. After the stop signal becomes inactive, first a static test takes place followed by a dynamic test before the system comes into operation. If a test fails or an error occurs during operation the system switches to fail save and puts the outputs in known safe states. A single state of an FSM can contain either another FSM, thus building up a hierarchy, or some function specification represented by its state action. Especially, against the background of mechatronic control systems, actions and conditions are related closely to signal processing. Therefore signal diagrams can be used to specify state actions or transitions, which give a method to integrate data flow into state machine specification. As an example the controller presented by figure 1 can be used as a state action of state “operation”. On the other hand an FSM can be used as part of a signal diagram, if it is encapsulated within a block with well defined data interfaces. In this situation an FSM is working in parallel like any other block or parallel FSM and will be synchronized by its signal inputs.

In the case of a single FSM or an FSM hierarchy it is important to know that there can be only one active state in the complete FSM at a given time. From this point of view, an FSM not only can be seen as flexible method of specification but also as an intelligent way to determine parts of a complex system that have to be loaded and ready for operation under conditions defined by the FSM.

3. IMPLEMENTING DYNAMIC RECONFIGURABLE SYSTEMS

3.1 Implementation and reconfiguration of algorithms/blocks and data flow

Implementation of a system specified by a signal diagram on an FPGA can be done straightforward, using FPGAs logic to implement the algorithms defined by blocks and FPGAs lines to realize the data flow defined by signals. Bit serial data processing and transmission reduces the amount of logic and lines to a minimum compared to parallel approaches and it allows the implementation of large systems using only a small part of an FPGA. Implementation is based on a library containing frequently needed processing elements [10]. All operand bits are processed and transmitted in time division on one line, LSB first. Each processing element needs one clock cycle to process one bit of input data. A control bus is used to mark the operand's bit significance. This bus has as many signals as the length of the operand. Each signal is only active during one clock cycle. By means of the operand's delay the appropriate control signal is chosen to determine significance and operand boundaries. Depending on the signal chain's length, data may arrive with different delays at a processing element. To keep the bit streams on different paths synchronized the needed number of registers is implemented in the paths with less delayed data. This task can be done automatically at compile time [11]. This method assumes constant delays on structure block level. That means block 1 of figure 1 generates always the same delay on its output.

Following this approach a block or a structure block can be compiled and implemented as a module in one (or several) slots of the FPGA. Data lines inside the block can be kept local and implemented with local lines. External inputs and outputs of the block have to be implemented using long lines in order to be connected to modules implemented in different slots. Each signal uses a dedicated line. That avoids any kind of real time problems, which arise extremely by using shared resources. Blocks can be reconfigured if they offer the same interface. This was already a precondition for specification and allows different blocks to use the same long lines. In the strict sense these blocks represent different types of the same module. However against the background of bit serial implementation the types of a module may generate different signal delays for one and the same module output signal. To ensure synchronization across module boundaries, block or structure block outputs are supplemented by synchronization ports as shown in figure 3. Synchronization ports insert as many registers as needed to compensate the delay difference between the actual delay of this type and the highest delay of all types, feeding this signal. In this way a structure block becomes a type and can be used for reconfiguration.

To meet real-time conditions, a constant sampling frequency f_s has to be guaranteed for all signals in all situations, even during reconfiguration. Bit serial methods use a fixed ratio between clock frequency f_c and word length n given by

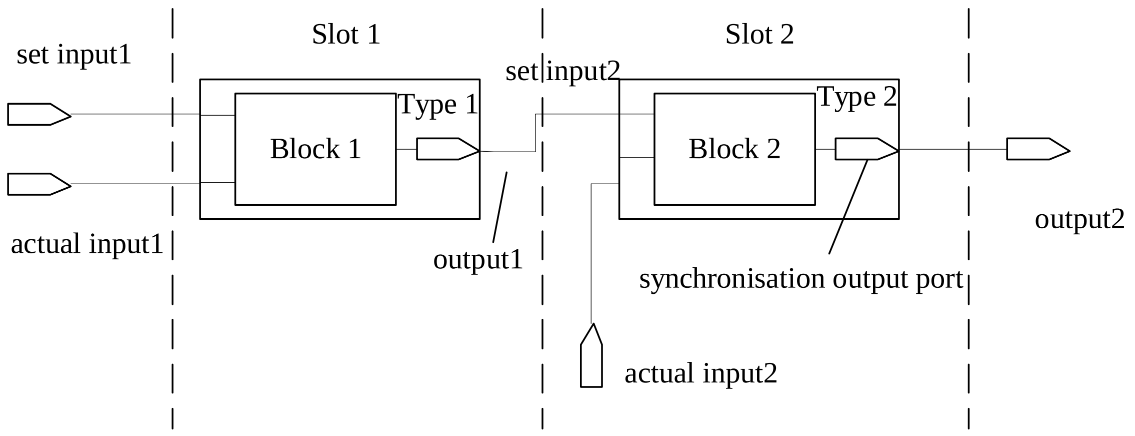

Figure 3 Block diagram for a partial reconfigurable system

$$f_c=f_s\cdot n.$$

This results from non-slip transmission, which means there is no gap between two succeeding operands. This leads to a low clock frequency (some MHz) as the sample frequency of mechatronic systems is not too high (<100 kHz) and helps to reduce the total power consumption of the FPGA, but the continuous data stream leaves no time for reconfiguration. This problem can be overcome by a burst mode. Here a gap is introduced between operands of one signal and all bits of an operand are transmitted and processed in a burst. This can be realized simply with the control bus already mentioned. At a clock frequency of 40 MHz a 16 bit operand takes 400 ns for processing or transmission. A sampling frequency of 100 kHz results in a sampling period of 10 µs. Thus the time gap available for reconfiguration is 9.6 µs. In this way the burst mode breaks the fixed ration between the clocks and offers a time gap which can be used for operations necessary during reconfiguration.

3.2 Implementation and reconfiguration of state machines

Implementation of FSMs can be done straightforward using classical methods. In the context of signal processing switching of signal lines is an important additional action following a transition from one state to another. This can be realized by multiplexers to manipulate the data flow as shown in figure 5a). As an alternative to multiplexers, tri state drivers, as given in figure 5b), come into operation. In both cases all states of the FSM and all logic elements processing data being switched have to be implemented completely on the FPGA, not exploiting the fact that at one time there is only one active state in an FSM. Partial reconfiguration offers the possibility to load only the part of logic needed by the active state and to use a new kind of data flow manipulation as indicated in figure 5c). To achieve this, an FSM is treated as a module and the blocks defined in its states are considered to be its types. Loading of the type of the active state is supervised by a type FSM (T-FSM), which is a distributed implementation of the FSM defined in specification, extended by special actions to control partial reconfiguration as shown in figure 5. Each type implements its own part of the FSM. Transition from state “not loaded” to “loaded” is initiated by the active state and executed by a general control module (GCM). If type loading is finished and the slot local reset becomes inactive, the FSM switches to ready and waits for activation. So far no signal processing occurs and all outputs of the type are deactivated. Activation is signaled by a T-marker, which is transmitted via a communication system from the GCM. Now the type’s signal processing

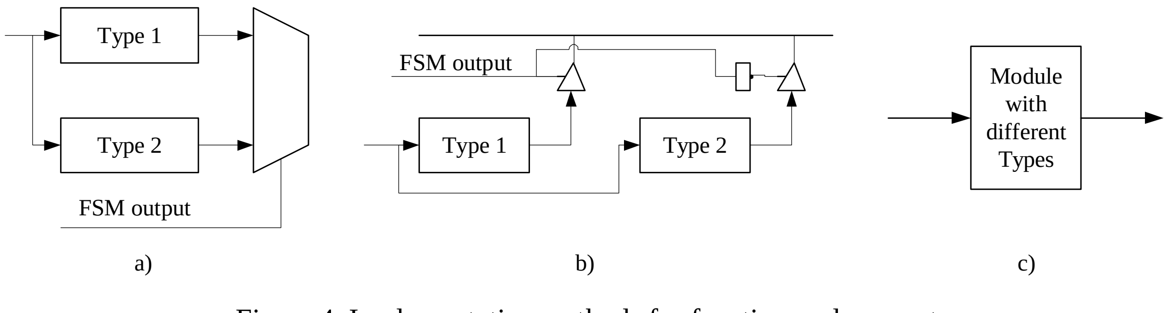

Figure 4 Implementation methods for function replacement

starts and its outputs are activated. If transition x becomes true the T-marker of the next state is forwarded to GCM, where loading of the associated type is disposed.

The time T_R needed for reconfiguration is governed by the size of the function that has to be loaded and the reconfiguration speed (FPGA clock and type). Today, reconfigurable FPGAs offer reconfiguration times in the range of some milliseconds. If sampling times are small enough to realize complete reconfiguration step in one sampling period, loading and activation of the next type can be accomplished in the time gap offered by burst mode. This will not cover all high frequency applications, but offers the advantage that the communication system and the GCM can have a very simple structure. A module type can be replaced in the same slot by another type having the same connectivity but yielding another function, depending on a state of its FSM. In situations where the time gap of the burst mode is to narrow, preloading of all reachable types will help to meet real-time conditions [13].

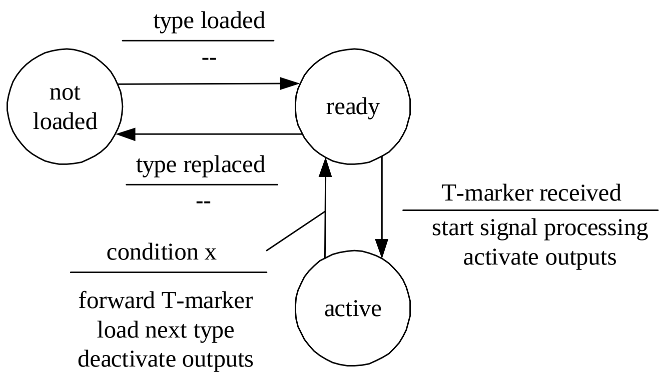

Figure 5 T-FSM

3.3 Communication system

A communication system that connects all reconfigurable slots with the GCM is needed for exchanging the T-Marker. Some major requirements arise:

- Low communication needs, only few bits have to be transmitted
- Low logic needs, all subscribers are implemented within one FPGA
- An addressed subscriber is listening always
- A medium access method appropriate for FPGAs

Three medium access methods are considered for implementation: central arbitration, multiple access (CSMA) and token passing. A central arbitration requires request and acknowledge lines for each subscriber that increase the number of bus macros needed significantly. Multiple access methods allow subscribers simultaneous write cycles and can destroy the FPGA. Therefore a kind of token passing methods was chosen for implementation; usually these methods forward a token to each subscriber in kind of a logical ring. This needs an address field to address each subscriber. Token passing on a bus requires at least receiving the complete token frame before transmitting it, and introduces a delay of one token transmission duration per subscriber. Both disadvantages, the need of an address field and the large delay, can be overcome by replacing the bus by a point-to-point-connection, similar to INTERBUS [12], where the delay can be limited to only one bit. Figure 6 shows the structure of the communication system. As communication takes place across module boundaries special developed bus macros come into operation. They extend Xilinx' bus macros that utilize long lines which are driven by tri state buffers. All subscribers build a shift register, each stores one bit of the frame and shifts it to the next subscriber on each clock. An additional shift register (S-REG) is implemented in the GCM to balance the difference between the frame length and the

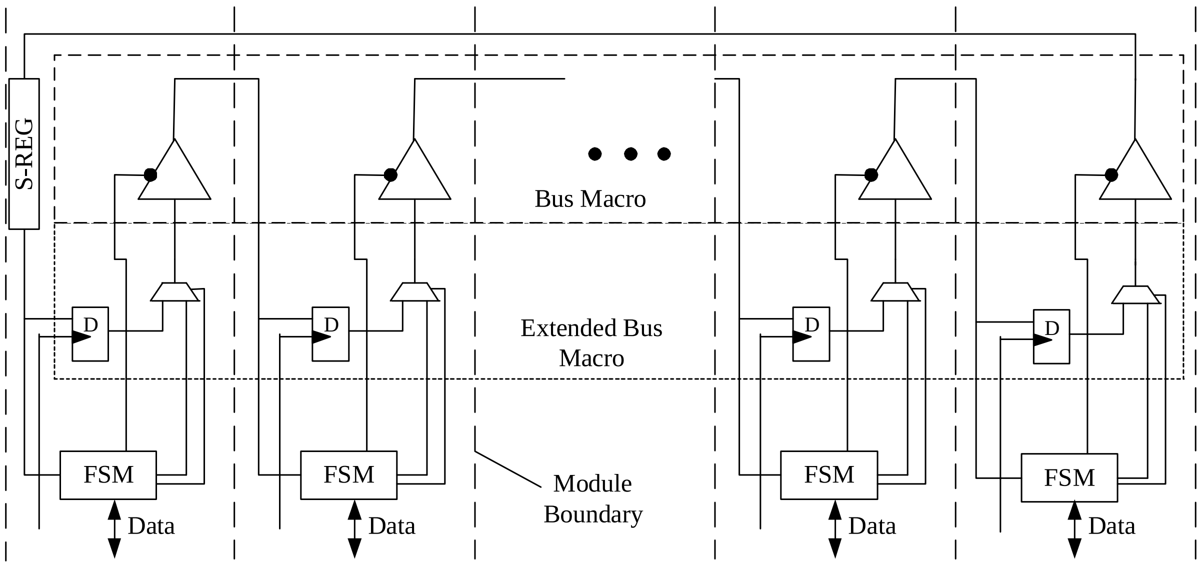

Figure 6 Structure of the communication system

number of subscribers, because the frame has usual more bits as subscribers that join the communication. The frame has the following structure:

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

A subscriber synchronizes to a frame by detecting the start delimiter. The next bit is the data field used flag. A subscriber is permitted to write to the data field only if used flag is not set. If used flag is set data will be passed to the T-FSM. This decides whether data represents the local T-marker and decides if the type is activated. The subscriber that filled the data field finally resets the used flag.

If slots are under reconfiguration, the communications system has to work without disturbances. To guarantee that the communication chain will not be broken during reconfiguration, the local multiplexer and register is implemented inside the bus macro as shown in figure 6. Bus macros are cores that are not affected by reconfiguration. They always use the same logic elements and read only data from the long lines of the communication system. The local multiplexer prevents any interaction of communication system's FSM to the bus macro during reconfiguration.

3.4 Control system

The T-marker is exchanged between a type (active state) and the GCM to load the next type. The GCM has access to a table containing all possible types (T-markers) together with their location and size in external memory. The type table represents static data that can be determined at compile time from the FSMs structure. After receiving a T-marker the GCM reads the bit stream to be loaded and sends it to the internal reconfiguration access port (ICAP). Following, the T-marker is forwarded to the new loaded type to activate its signal processing. The GCM will never be reconfigured and stays in a fixed part of the FPGA.

4. RESULTS AND CONCLUSIONS

Implementation of a run-up FSM and a feed back control system of an electro motor has been done using a XILINX FPGA and development tools. The communication system has been tested and validated. Special bus macros have been developed, which straddle more than two slots, to enable access to one process signal destination from multiple slots. Reconfiguration actually is tested via JTAG. Implementation of the GCM and reconfiguration via ICAP is in progress and will be done shortly. Results can be presented in the final paper.

The presented methods allow direct dynamic reconfiguration of hard real-time and control systems with moderate sampling frequencies. Mixed specification of data flow (signal diagram) and control flow (FSM) is used for direct implementation on an FPGA. The needed logic amount is reduced by dynamic reconfiguration, thus enabling the implementation of more complex algorithms into smaller FPGAs. The distributed communication system, used to dispose type loading, is very fast and needs a minimum of logic.

Future work will focus on dynamic saving and restoring of permanent local data inside blocks. This has to be done in real-time, synchronized to loading of type's logic. Data transfer will be realized by an extension of the communication system to move data between type's registers or RAM and additional RAM blocks controlled by the GCM.

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] 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.

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

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

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

[7] ETAS; 2004, http://en.etasgroup.com/products/ascet_sd/

[8] The Mathworks: System Generator for DSP, http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=dr_dt_system_generator

[9] i-Logix: 2004, http://www.ilogix.com/

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

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

[12] Interbusclub: 2004; http://www.interbusclub.com/

[13] Reinemann, Kasper, ; (to be published); Function replacement of hard real-time systems using partial reconfiguration