Architecture Template with Dynamic Buffering for Runtime Reconfiguration of Adaptive Embedded Communication Systems

Dirk Eilers, Helmut Steckenbiller, Rudi Knorr
Fraunhofer Institute for Communication Systems
Hansastr.32, 80686 Munich, GERMANY
eilers@esk.fraunhofer.de

Abstract

This contribution introduces a distributed dynamic buffering scheme for runtime reconfiguration in adaptive processing architectures, e.g., for streaming media applications. With dynamic reconfiguration, area-cost of field-programmable logic (FPL) can be reduced by reuse and potential for adaptive signal processing techniques can be enabled. The challenge with runtime reconfiguration is the reconfiguration latency. Given the limitations with traditional approaches, we propose a distributed dynamic buffering scheme to hide latency. The simulation results show that our approach enables potential for runtime reconfiguration for adaptive signal processing under real-time constraints. Finally, we derive an architecture template with implementation details of the dynamic buffer scheme.

1 Introduction

Streaming media applications are characterised by high performance demands in the lower layer signal processing. The use of FPL can provide great benefits over software with regard to performance [3] and over application specific integrated circuits (ASIC) with regard to flexibility, time-to-market and product-life-time. Unfortunately, these benefits are gained by the cost of area. The re-use of area resources by dynamically reconfiguring parts of the custom logic can significantly reduce the cost. Furthermore, field-programmable logic with dynamic reconfiguration capabilities can enable adaptive methodology for digital signal processing, to satisfy time-variant environments.

Signal processing for streaming media application is characterised by a series of processes with high performance needs in high bandwidth applications. These processes are: data processing; source processing; channel processing; symbol-mapping and modulation; transformation and filtering. The constraints for streaming media applications with adaptive signal/data processing are: broadband digital signal processing; variant environment and/or variant functionality; (scalable) count of function-variants and real-time operation. For real-time operation the main constraints for quality of service (QoS) have to be met, namely, throughput, variance/jitter, delay and loss-rate. Adaptive hardware can deliver the needed performance in combination with the necessary flexibility at runtime. The dependencies regarding the dynamic range of adaptive signal processing functions are: data-dependent; channel/noise-dependent; functional/user-dependent and movement-dependent. The data-dependency can be, for instance, the data-rate adaption in rate controlled CODECs (Coder/Decoder). The channel-dependency can be, e.g., channel-adaptive CODECs, which adapt the redundancy regarding a time-variant noisy environment. The functional-dependency can be based on user inputs such as horizontal/vertical handover. Finally, movement-dependency can be adapting the modulation scheme in reference to speed (e.g. wireless gateways in vehicles).

The challenge with runtime reconfigurable hardware is the reconfiguration latency. For Example: with the Xilinx XCV1000 one CLB (Complex Logic Block) column with the configuration bit-stream size of 7488Byte may hold logic of about 10KGE (gate equivalent) and might be configured with 50MHz in about 150µs of time [8]. The reconfiguration latency disrupts the timing of the data stream. To guarantee each aspect of the QoS, the reconfiguration latency has to be hidden in respect to the inputs and outputs. Traditionally, this can be done by implementing input buffers as in [2]. Techniques like configuration caching [5], multi-context CPLDs (Complex Programmable Logic Device) and compile-time scheduling via CFG (Control Flow Graph) [6] can also be used.

This work introduces a novel distributed and dynamic buffer scheme for hiding latency. The challenge with runtime reconfiguration using a traditional single input buffer approach is investigated in the next section. We introduce a new buffer scheme based on distributed and dynamic buffering in section 3, and in section 4 we derive an architecture template with implementation details. We will conclude with a view on future work in section 5.

2 Buffering of Runtime Reconfiguration for Streaming Media Application

Systems with burst natured dataflow disruptions are normally buffered to hide the latency. Over a mean period of time the buffers have to be large enough, to collect all data from the disruption periods. Additionally the process elements (PE) have to work down the built up buffer in time. Figure 1 shows a system with 4 process elements and a single input buffer.

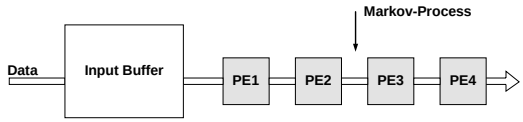

Figure 1 Static Buffer Scheme

We used the discrete event model method [7] to simulate arriving reconfiguration requests, which are expected to be Poisson distributed as they are assumed to be independent [4]. No two reconfigurations can be conducted at the same time and the FIFOs (first in first out buffers) do have blocking nature. For the simulation, we used the approach of O. Diesel et al, who assume stochastic distributions for the area-size and inter-arrival time of configurations and conducted simulations, to draw conclusions on buffer overflows and throughput with dynamic reconfiguration of functions in heavily loaded co-processing systems [2]. We use the OMNet++ simulation framework from the Budapest University of Technology and Economics [7]. As in [2] we focus on the buffer sizes and statistically generated Poisson distributed inter-arrival times of reconfigurations with generally estimated application profiles. It must be stated that the results we derive are of a general nature, to depict the potential for function variants in stream-based real-time communication systems. For analysis of concrete applications, dedicated modelling has to be done.

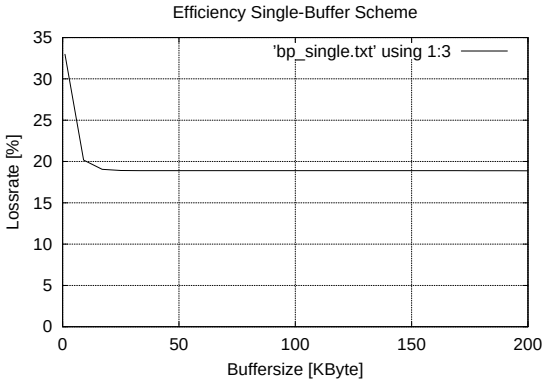

Figure 2 Loss-Rate vs. Static Buffer Size

We conducted the simulations with mean expected system parameters of $rate_{input}$ = 20MByte/s, $rate_{processing}$ = 25MByte/s, 5 process elements with 5 function variants each of generally distributed size $area_{funcvar}$ = 1KGate to 30KGate. This leads to reconfiguration times $latency_{funcvar}$ = 15µs to 450µs. The mean inter-arrival time is set to $inter\text{-}arrival\text{-}time_{funcvar}$ = 3ms. The simulation results are depicted in Figure 2. The desired loss-rate for the QoS is not met, because one stall of a process element will stall the whole chain. With the given parameters the process elements are not able to catch up with the built-up buffer, even with an arbitrary amount of buffer.

3 Distributed Buffering Schemes for Runtime Reconfiguration

To overcome the limitations of the single input buffer scheme, we propose to decouple the process elements by using distributed buffers. The first approach uses distributed fixed-size buffers and the second approach uses dynamic distributed buffers.

Two effects would make local buffering more advantageous than global buffering at the input. Firstly, local buffering will erase some dataflow disruption in a statistical manner. For example: the local buffered dataflow after reconfiguration of process element k is worked down, when process element k-1 is reconfigured. Therefore, both stalls will only result in a built-up buffer from one stall. Secondly, the process elements are normally not designed with 100% utilisation and can work down a buffer built-up by reconfiguration stalling. For example: when a processing element is stalled over the first 20% of a dedicated period and the mean utilisation is about 80%, it can work down the buffer in about 8/10 of the dedicated period and can catch up with normal buffer utilisation in time – before the next reconfiguration starts. In the first approach the FIFOs are organised with fixed size and in the second approach the size of each FIFO is scheduled dynamically up to a given total amount.

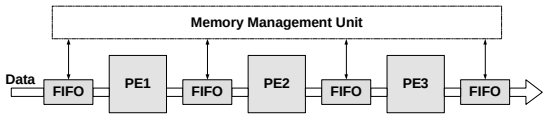

Figure 3 Dynamic Buffer Scheme

Figure 3 depicts the system with a memory management unit (MMU) for dynamic buffers. In the case of fixed-size buffers, the MMU is left out. For the simulation of the different buffer schemes, the discrete event model based simulator OMNet++ has been used, as described in section 2. The dynamic reconfiguration events have a statistical arrival-rate and form a markovian process. If an

event is in service, the arrival event is queued. For the depicted graphics the parameters of section 2 were used.

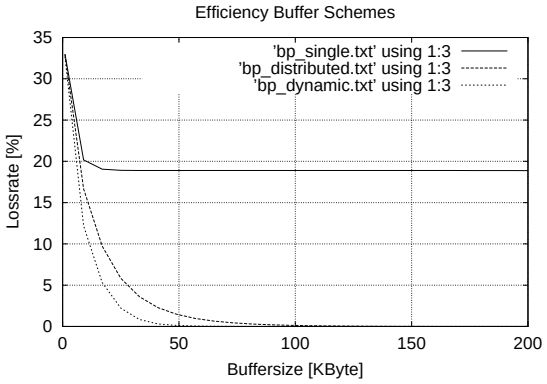

Figure 4 Loss-Rate vs. Buffer Size

Figure 4 depicts the loss-rate vs. the total buffer size. The distributed fixed size scheme will reach the desired loss-rate with buffer sizes above 50KByte. The total amount of used buffer of the chain has to be worked down again, before reaching the sink. Therefore, the amount of used buffer is proportional to the additional delay. Furthermore, the amount of buffer used is analog to the jitter of the dataflow seen at the end of the chain. When assuming a sink with constant bit-rate without backpressure, this jitter has to be buffered to prevent additional data losses. In this example, the additional delay for the fixed distributed buffer scheme results from 50KByte built up buffer divided by 20Mbytes/s to about 2,5ms. Assuming e.g. a constant sink rate λ the total buffer is two times 50Kbyte (one for the chain buffer and one for the jitter buffer). Finally, with the dynamic distributed scheme all dedicated loss-rates are reached with less total buffer. In this example the desired loss-rate (near zero) is reached with 50% of buffer size compared to the fixed distributed buffer scheme. Therefore, the delay is about 1,25ms and the total amount of buffer adds up to 50KByte in size. For a complete transmission system, these considerations have to be made for sender and receiver. Additionally, if the jitter buffer is integrated in the chain buffer, further reduction can be achieved.

When looking at the dynamic range in respect to the arrival rate of reconfigurations depicted in Figure 5 (with total buffer-size of $size_{buffer}$ = 20KByte) one can see that the loss-rate of the fixed distributed scheme is not developing as well with decreasing dynamic as that of the single input buffer scheme. This is not convenient, as the dynamic of the reconfiguration arrivals may change during runtime. This is because the first buffer becomes more important, if the dynamic of the arrival of reconfigurations decreases and it is not able to handle a single disruption. At some point this preponderates to the benefits of the distributed buffer scheme. This is not the case with the dynamic distributed buffer scheme. The loss-rate for the dynamic distributed scheme is superior to the other schemes over the whole range, with the exception of the overloaded dynamic range, where the buffers are mostly blocked and the stochastic benefits from the first effect from the distributed scheme can not take effect as well as with the fixed distributed buffer scheme. In any case - in this range the QoS is not satisfying for streaming media application and therefore is not interesting in this context.

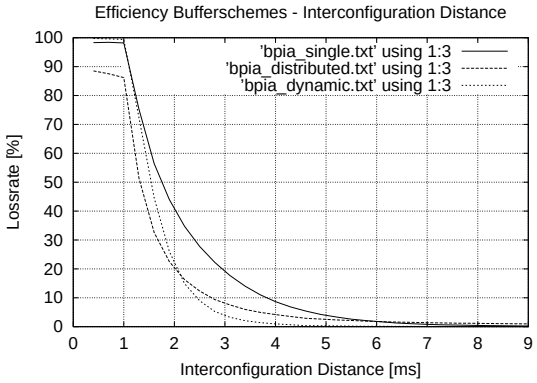

Figure 5 Loss-Rate vs. Dynamic Range

The delay can not be decreased by adding an additional amount of buffer, because the delay is dependent on the desired loss-rate, which itself is correlated with a fixed minimum buffer size. To further decrease the delay, additional methodology such as configuration caching can be used to archive scalability. The fixed distributed scheme will reduce the loss-rate to the desired range with a buffer build-up as additional delay. This can be further reduced with the dynamic distributed buffer scheme.

4 Architecture Template with dynamic Buffering

Given the results from section 3 we derive an implementation for the dynamic distributed buffer scheme. The basic idea is to have a bank of FIFO elements which can be dynamically allocated.

Each FIFO element has an input and an output port with registers for the read and the write pointers respectively. In addition there is a token-in and token-out gate for the input and the output port. To make connections between different FIFO elements, there are dynamically reconfigurable interconnections for the input ports and the output ports. There is one token for the input and one for the output port in each system of connected FIFO elements. The token represents the active state for the input and the output respectively.

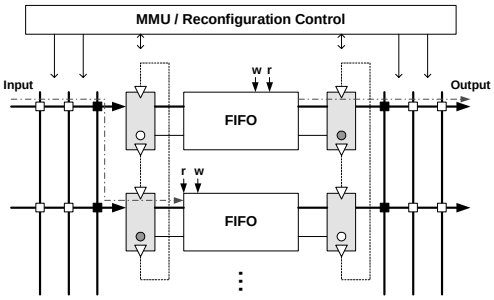

Figure 6 Schematic of Dynamic Buffer Scheme

Figure 6 shows the schematic of the distributed dynamic buffer scheme, just after overflow of FIFO element 1 and allocation of FIFO element 2. If an overrun in the input-active FIFO is detected, this will cause an exception handling as follows: if the next FIFO element is in output-active state, a new FIFO element will be allocated and the input token will be passed to it; otherwise the input token will be passed. If an underflow in the output-active FIFO is detected, this will cause an exception handling as follows: if the output-active port is not on the same FIFO-element as the input-active port, the read token will be passed to the next FIFO element in the chain. The last FIFO element is scheduled for de-allocation.

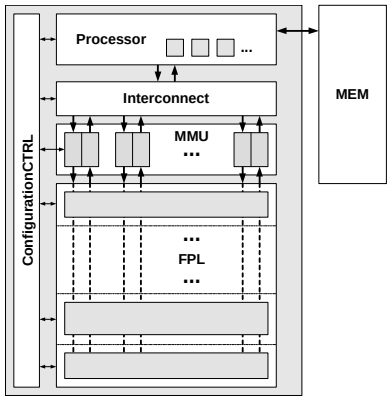

Figure 7 Architecture Template for Adaptive Communication Systems with Dynamic Reconfiguration

We derive a general SoPC architecture template for adaptive signal processing streaming media systems as depicted in Figure 7. The architecture template is comprised of a processor core, an FPL section for customised functions, a flexible interconnection, the dynamic distributed buffer scheme with specialised memory management unit and a configuration control unit. The distribution of the data busses follows a regular structure. This circumstance becomes more important with dynamic placement scheduling, when using a cache-like approach [5] potentially with defragmentation [2]. The software parts of scheduling algorithms and (potentially) the exception handling of the MMU can be handled by the Processor core.

5 Conclusion and Future Work

The proposed distributed buffer schemes can hide the reconfiguration latency more effectively than traditional single input buffers and the constraints for the QoS can rather be met. Compared to the fixed distributed buffer scheme the dynamic distributed buffer scheme can further reduce the mean delay and is beneficial over the whole range of possible inter-arrival times of reconfigurations. With the adherence of the desired QoS, e.g., for streaming media applications, the proposed architecture template enables potential for runtime reconfiguration of adaptive embedded communication systems, e.g., adaptive channel-coding algorithms, to optimise channel efficiency, in conjunction with other adaptive methodology, and can hence reduce the area cost significantly. The architecture template exploits the properties of today's available FPGA architectures and the dynamic allocation is proposed to be implemented using runtime reconfiguration itself.

Further addition of buffer amount cannot reduce the mean delay-time and other approaches in combination with the buffer scheme are needed. Furthermore, results from an architectural implementation are needed to draw the balance between cost-offset, scaling costs and gains as to delay and loss-rate. Finally, further work has to be done to derive application profiles for adaptive communication systems based more on real-life application data.

References

[1] G.I.E. Cancelo, S. Zimmermann: *Modeling and Simulation of a Readout Architecture for Pixel Detectors*, IEEE Nuclear Science Symposium (NSS), 1998

[2] O. Diesel, H. ElGindy, M. Middendorf, B. Schmidt: *Dynamic Scheduling of Tasks on Partially Reconfigurable FPGAs*, 1999

[3] D. Eilers, A. Voglgsang, A. Plankl, G. Körner, H. Steckenbiller, R. Knorr: *A prototype of an AAL for high bit rate real-time data transmission system over ATM networks using a RSE CODEC*, IEEE International Workshop on Rapid System Prototyping, Los Alamitos USA, 2000

[4] B.R. Haverkort: *Performance of Computer Communication Systems: a Model based Approach*, Wiley, Chilchester, England, 1998

[5] Z. Li, K. Compton, S. Hauck: *Configuration Caching Techniques for FPGA*, FPGAs for Custom Computing Machines, 2000

[6] Z. Li, S. Hauck: *Configuration Prefetching Techniques for Partial Reconfigurable Coprocessor with Relocation and Defragmentation*, FPGA'02, Monterey USA, 2002

[7] A. Varga: *OMNeT++ User Manual*, Department of Telecommunications, March 2002

[8] Xilinx: *XAPP151: Virtex Series Configuration Architecture User Guide*, September 2000