

Adaptive Traffic Scheduling Techniques for Mixed Real-Time and Streaming Applications on Reconfigurable Hardware

Tobias Ziermann, Juergen Teich

Hardware/Software Co-Design, Department of Computer Science, University of Erlangen-Nuremberg

Abstract—With the ongoing development of new FPGA generations, the reconfiguration time decreases and therefore the benefit of runtime reconfiguration increases. In this paper, we describe how to use runtime reconfiguration to improve the efficiency of transmitting streaming data on a communication channel shared with real-time applications. This means, the bandwidth that the streaming data has available is dynamically changing. To use the bandwidth effectively, different modules can be loaded on the reconfigurable hardware. These modules have a tradeoff between bandwidth and area requirements. The target now is to find an optimal reconfiguration schedule that minimizes an objective function consisting of two conflicting objectives: reducing the average area needed and providing a certain quality of transmission. In this paper, a model for this scheduling problem is presented and an Integer Linear Programming (ILP) formulation is introduced to calculate an optimal offline solution for benchmarking.

In addition, an online scheduling system is presented. It uses the current delay of the streaming application to calculate the schedule. Extensive simulations have been made to show the benefits of the proposed solution.

I. INTRODUCTION

Heterogenous communication systems, where real-time and streaming applications share the same medium, are becoming increasingly important. The main challenge is to provide a certain Quality-of-Service for the streaming applications with dynamically changing available bandwidth due to the sending behavior of the residual applications. FPGAs are one of the most promising technologies to make dynamic context-aware hardware possible. Several architectures have been presented to adapt the hardware to the current needs. In [1], partial reconfiguration is used for embedded applications, or the Erlangen Slot Machine [2], which is an FPGA board that enables the designer to easily exchange hardware modules. In [3], a software reconfigurable multiprocessor system is described and an automotive application is implemented.

On the one hand, combining reconfigurable hardware with a heterogenous communication system has the potential to provide a very effective system, in terms of cost, energy efficiency and high performance. On the other hand, the complexity makes the design of such systems difficult. Therefore, we provide a mathematical model of such systems to simplify further development on scheduling mechanisms.

An example of such an application is the integration of a rear view camera system into an existing car communication

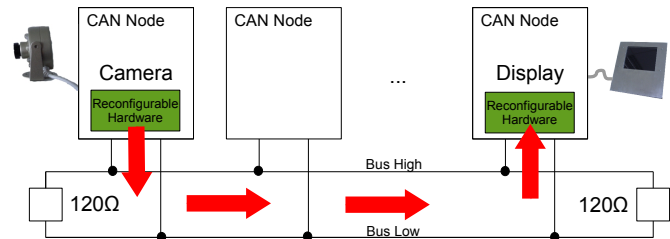


Fig. 1. Application architecture

structure as depicted in Figure 1. A camera and a display node are connected to the communication network, here the Controller Area Network (CAN) [4]. These two CAN nodes use the bandwidth, which is not consumed by real-time applications with higher priorities, to transmit a video stream. Therefore, the available bandwidth is changing. The camera node is able to adapt to that by loading different modules. For example loading a lossless compression module would allow to transmit the same data with less bandwidth, but would also require additional computation. In general, using a more efficient module means that additional computation has to be done.

On FPGAs, additional computational blocks may be accommodated by means of hardware (re-)configuration. The used area should however be minimized for several reasons: First, unused FPGA-area can be disconnected from either the clock signal or the power supply to save energy. Second, in a multi-task system, the reconfigurable area can be used by other applications, e.g., in our automotive example, the freed computation resources can accelerate a navigation calculation.

The rest of this paper is organized as follows. Section II describes the underlying scheduling problem. Section III presents the architecture used. A solution to the special case of having two reconfigurable modules is introduced in Section IV. In Section V, results of this approach are presented, before concluding in Section VI.

II. SCHEDULING MODEL

In the following section, we provide a verbal and a mathematical description of our scheduling problem. Then, an example is presented to clarify the model.

A. Problem Description

The scheduling problem can be described as follows. For a given communication path, a node wants to stream data.

By streaming we mean that a constant amount of data per time unit needs to be transmitted. The data doesn't have to be transmitted immediately. However, the requirement is that the delay of the data stays within a certain limit. The communication path is also used by other real-time or streaming applications, and we are assuming our streaming data has the lowest priority.

Therefore, only the remaining bandwidth can be used. To be able to adapt to the varying remaining bandwidths, the reconfiguration system of the node has the possibility of using different modules for the transmission. These modules allow to transmit different amounts of data per available bandwidth, but the use of more efficient modules comes with the cost of extra computational effort.

Furthermore, the switching between modules induces times where no data is transmitted. Therefore, the question is how to find an optimal schedule where the least computation is done while still fulfilling the constraints of the data.

B. Mathematical Model Description

A reconfigurable system with a set of possible modules M is given. Only one module can be loaded at the same time. A reconfigurable module m is described by its bandwidth requirement b_m to transmit the video data for one timeslot k :

$$0 < b_m \leq 1 \in \mathbb{R},$$

and by its cost (i.e., FPGA area) a_m , which is necessary to run the module on the system:

$$0 \leq a_m \leq 1 \in \mathbb{R}.$$

To represent the time necessary to reconfigure between different modules, an additional module r is introduced to the set of possible modules. This means, when module r is loaded the reconfiguration system is reconfiguring the next module. It has the following properties:

$$b_r = \infty; a_r = \max\{a_x | x \in M \setminus \{r\}\}.$$

If a reconfiguration takes place, i.e., the reconfiguration module r is scheduled, no data can be sent. Therefore, the bandwidth requirement b_r is infinity. Technically, the reconfiguration module r would need the maximum area of the module before and after reconfiguration. For the sake of simplicity we use the worst case assumption which is to use the maximum area requirement of all modules.

Time is divided into timeslots k in such a way, that one timeslot has the length of the reconfiguration time. For

b_m	Bandwidth requirement
a_r	Area requirement
$s(k)$	Schedule
$c(k)$	Capacity
$d_p(k)$	Data, the loaded module could possibly send
$d_a(k)$	Data, available to transmit
$d_n(k)$	Data, not send (i.e. delay)
o_{total}	Objective

TABLE I
OVERVIEW OVER THE USED VARIABLES AND FUNCTIONS

the mathematical definition, we assume a finite number of timeslots which are numbered from 0 to k_{max} :

$$K = \{0, 1, 2, \dots, k_{max}\}.$$

The schedule, which is the function that has to be found, assigns one module to each timeslot:

$$s : K \rightarrow M.$$

Since we are sharing the communication path with other applications, we have a varying busload. It is described by a capacity function, which describes the available bandwidth during timeslot k :

$$c : K \rightarrow [0, 1].$$

Consequently, the streaming data $d_p(k)$, which the module $s(k)$ can maximally transmit during timeslot k , can be calculated as follows:

$$d_p(k) = \begin{cases} 0, & s(k) = r \\ \frac{c(k)}{b_{s(k)}}, & else \end{cases}.$$

Assuming the amount of streaming data is constant, we normalize the arriving streaming data in each timeslot to one. Therefore, the streaming data $d_a(k)$ that is available for transmitting at the beginning of the timeslot k is described by the following function:

$$d_a(k) = \begin{cases} 1, & k = 0 \\ d_n(t-1) + 1, & else \end{cases},$$

where $d_n(k)$ is the data not sent at the end of the timeslot k :

$$d_n(k) = \max(0, d_a(k) - d_p(k)).$$

The data not sent at the end of timeslot k is the amount of data delayed and therefore we will simply call it *delay* in the following of this paper.

The constraint that between two successive different modules the module r needs to be scheduled, is formulated as:

$$s(k) = r \Leftrightarrow s(t-1) \neq s(t+1) \\ \wedge s(t-1) \neq r \wedge s(t+1) \neq r.$$

C. Objective Function

The goal is to find a schedule that minimize the total cost (i.e., area) while keeping a certain quality of service. This means we try to minimize the objective function o_{total} , which can be expressed as a weighted function of the average cost of the modules and the maximum delay:

$$o_{total} = w_0 \cdot \frac{\sum_{k=0}^{k_{max}} a_{s(k)}}{k_{max} + 1} + w_1 \cdot \max(d_n(0), d_n(1), \dots, d_n(k_{max})) \quad (1)$$

Still, the weights for the objective function o_{total} have to be chosen. We like to scale the weights w_0 and w_1 in such a way, that the maximum delay of 100 is equally weighted to using the full reconfigurable area. As the maximum cost is normalized to 1 the delay measure $d_{measure}$ has to be divided by 100: $w_0 = 1$; $w_1 = 0.01$

k	0	1	2	3	4	5	6	7
$c(k)$	0.5	0.5	0.2	0.2	0.2	0.2	0.6	0.2
$s_i(k)$	l	l	l	l	l	l	l	l
$d_p(k)$	1	1	0.4	0.4	0.4	0.4	1.2	0.4
$d_n(k)$	1	1	1	1.6	2.2	2.8	3.4	3.2
$d_n(k)$	0	0	0.6	1.2	1.8	2.4	2.2	2.8
$s_i(k)$	l	r	h	h	h	h	h	h
$d_p(k)$	1	0	1	1	1	1	3	1
$d_n(k)$	1	1	2	2	2	2	2	1
$d_n(k)$	0	1	1	1	1	1	0	0

Fig. 2. Scheduling example

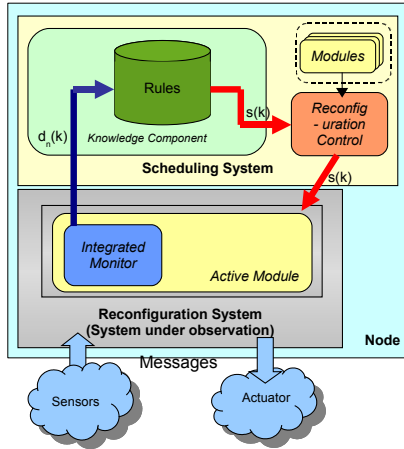


Fig. 3. Reconfiguration architecture used for the experiments.

D. Example

The example setup has the possibility to reconfigure between a *low* and a *high* speed module: $M = \{l, h, r\}$. Throughout the paper, we use the following module properties as an example:

- 1) Module “high”: $b_h = 0.2$; $a_h = 1$
- 2) Module “low”: $b_l = 0.5$; $a_l = 0.5$
- 3) Module “reconfiguration”: $b_r = \infty$; $a_r = 1$

In Figure 2, two schedules for a given capacity function $c(k)$ are described as an example. The first schedule produces the lowest average area used of 0.5, but generates a maximum delay of 2.8. The second schedule reconfigures at timeslot 1 from module low to high. This increases the average area used to 0.9375, but also reduces the maximum delay to 1.

E. ILP Formulation

The ILP formulation is skipped for space restrictions. However, it is analogous to the mathematical description in Section II-B. It will serve as comparison for the runtime algorithms.

III. RECONFIGURATION ARCHITECTURE

The architecture used in this paper is described in Figure 3. It consists of two subsystems: the *reconfiguration subsystem*

and the *scheduling subsystem*. The reconfiguration subsystem is running the modules. The scheduling subsystem is responsible for deciding which modules are loaded to the reconfiguration subsystem. The *knowledge component* contains the intelligence of the system. It can consist of simple rules or a complex learning system. Our first approach to a knowledge component is described in section IV. The *reconfiguration control* puts the commands from the knowledge component into action, by loading the appropriate module to the reconfiguration system.

IV. THE TWO MODULE PROBLEM

The first approach to solve the scheduling problem during runtime is by simplifying it to two modules, l and h as described in Section II-D. The architecture described in Section III is used. As knowledge component we are using a rule set. The rule set is responsible to control which module has to be loaded at each timeslot.

For the two module problem the scheduler only has to decide whether to switch modules during a timeslot, or not. This can be done by using two rules. The rule considered is depending on which module is currently loaded:

- 1) if $s(k) = l$:

$$if \{ (d_n(k) > d_{n,lh}) \} then s(k+1) = r, s(k+2) = h$$

- 2) if $s(k) = h$:

$$if \{ (d_n(k) < d_{n,hl}) \} then s(k+1) = r, s(k+2) = l$$

The decision whether to switch modules is only dependent on the current delay $d_n(k)$. This value can also be seen as the fill level of the buffer.

The behavior of the rules is best explained by the hysteresis diagram in Figure 4. In the following, we denote this approach as *two rules approach*.

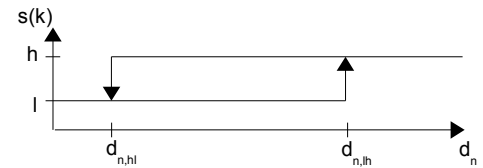


Fig. 4. Hysteresis diagram to describe two rule approach.

An offline evaluation is done to find the parameters for the rules. Simulation is done for different kinds of capacity functions that represent possible characteristics. Then the best parameter set is chosen. A detailed analysis is presented in Section V-C.

V. EXPERIMENTAL ANALYSIS

In the following, the two rule approach will be analyzed. Therefore, different capacity functions are generated to represent different scenarios. They are used to generate the optimal solution with the ILP formulation and compared with the two rules approach. At the end of this section, we try to find the best static parameter set to receive good results in all scenarios.

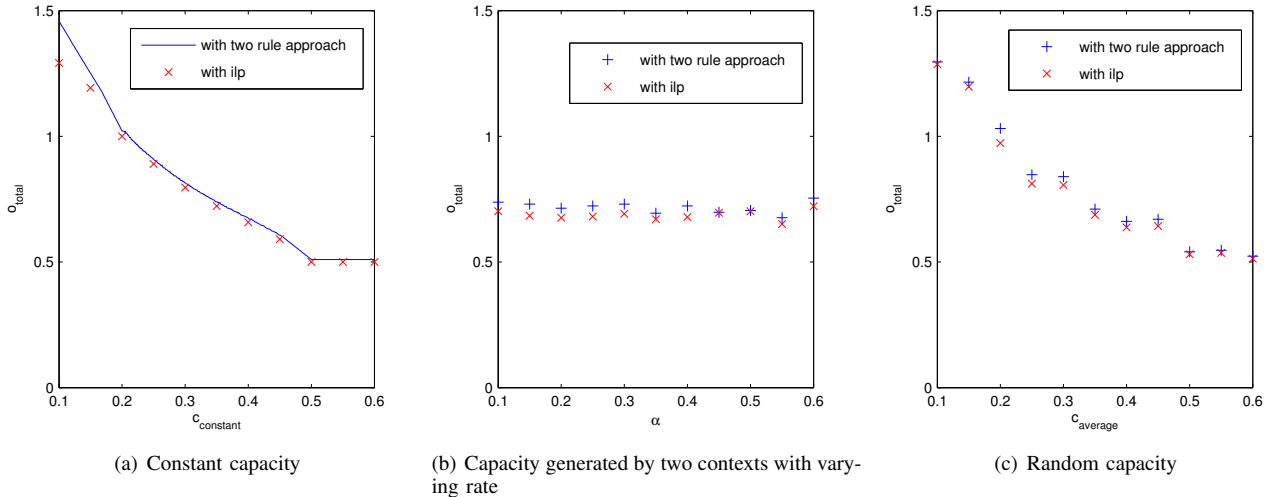


Fig. 5. The value of the objective function of the ILP results and the two rules approach for different capacity functions

A. Generation of the Capacity Function

Beside the set of modules, we need to define the capacity function for the problem. We choose three different approaches to generate the function:

The first and simplest one is to assume a constant capacity:

$$c(k) = c_{constant}$$

The second approach assumes that the communication path can be in different contexts γ , where the capacity is constant. A context γ is described by its characteristic capacity c_γ and its termination rate α_γ . As common for network traffic modeling [5], the termination of a context is modeled by using a Poisson process. The next context after one context terminates is chosen based on a uniform distribution.

The last approach to generate the capacity function is by assigning each timeslot a random capacity.

B. Comparing the Two Rules Approach to the ILP Solution

The two rules approach was chosen because in most cases it is very close to the optimal solution. In contrast to extensive optimization, like ILP, it has a relatively low computational complexity and can be run every timeslot concurrently to the loaded streaming application.

For the following experiments three different capacity sequences, as described in V-A, were tested with $k_{max} = 99$. For analysis, we always ran the two rules approach with the following possible parameters and chose the setting with the lowest objective function value:

$$d_{n,lh,test} \in \{0, 0.1, 0.2, 0.3, \dots, 100\}$$

$$d_{n,hl,test} \in \{0, 0.1, 0.2, 0.3, \dots, 10\}$$

The objective function was calculated according to Equation 1. The results in Figure 5 show that the two rules approach is very close to the optimal ILP solution.

C. Choosing a Parameter Set

An evaluation of the objective space shows, that optimal parameters are not dependant on the character of the capacity function. Therefore, we use static parameters.

O_{total} :	high variance	rare low c(k)	rare big c(k)	$c(k) = 25$	$c(k) = 45$
static	0.793	0.677	0.927	0.953	0.644
optimal	0.790	0.674	0.926	0.952	0.643

TABLE II
COMPARING THE STATIC TO THE OPTIMAL PARAMETERS

To show that the chosen static parameters give very good results, we compared the resulting objective function from the two rule approach using the static parameters and using the best parameters, chosen as described in Section V-B for different scenarios. The results from these tests are shown in Table II. It can be read that the results with the best parameters for each capacity function are almost similar to our static ones.

VI. CONCLUSION AND OUTLOOK

In this paper, we presented a model and an ILP formulation for a scheduling problem that occurs when using runtime reconfiguration to improve efficiency of transmitting streaming data over a medium shared with real-time applications. For the case of two available modules an online scheduling system has been developed. Experiments to compare the system to the optimal solution and to evaluate different parameter settings have been made.

Currently, the scheduling parameters have to be setup manually. It is desirably that a design flow can find these in an automatic offline evaluation. Going forward, the scheduling system has to be extended for multi-module systems.

REFERENCES

- [1] M. Hasan and S. Zivarras, "Runtime partial reconfiguration for embedded vector processors," in *Information Technology, 2007. ITNG'07. Fourth International Conference on*, 2007, pp. 983–988.
- [2] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer," *J. VLSI Signal Process. Syst.*, vol. 47, no. 1, pp. 15–31, 2007.
- [3] M. Ullmann, M. Huebner, B. Grimm, and J. Becker, "An FPGA run-time system for dynamical on-demand reconfiguration," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004.
- [4] *CAN Specification Version 2.0*, Robert Bosch GmbH, 1991.
- [5] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience, 2006.