TOWARDS RUN-TIME PARTITIONING OF A REAL TIME OPERATING SYSTEM FOR RECONFIGURABLE SYSTEMS ON CHIP *

Marcelo Götz, ¹ Achim Rettberg ² and Carlos Eduardo Pereira ³

¹*Heinz Nixdorf Institute
University of Paderborn, Germany
mgoetz@uni-paderborn.de*

²*C-LAB
University of Paderborn, Germany
achim@c-lab.de*

³*Departamento de Engenharia Eletrica
UFRGS - Universidade Federal do Rio Grande do Sul - Brazil
cpereira@eletro.ufrgs.br*

Abstract: Reconfigurable computing have successfully been designed taking into advantage the supporting of architectures based on the FPGAs and CPU. Moreover, the new hybrid FPGAs (e.g. Virtex-II ProTM), provides a hardcore general-purpose processor (GPP) embedded into a field of programmable gate arrays. Together with the ability to be partially reconfigured, those chips are very attractive for implementation of run-time reconfigurable embedded systems. However, most of the efforts in this field were made in order to apply these capabilities at application level, leaving to the Operating System (OS) the provision of the necessary mechanisms to support these applications. This paper present an approach for run-time reconfigurable Operating System, which take advantage of the new hybrid FPGA chips to reconfigure itself based on online estimation of application demands. The paper focus on run-time assignment and reconfiguration of OS services over a hybrid architecture. The proposed model uses a 0-1 Integer programming strategy for assigning OS components over a hybrid architecture, as well as an alternative heuristic algorithm for it. In addition, the evaluation of the reconfiguration costs are presented and discussed.

Keywords: Reconfigurable Computing, System-on-Chip, Real Time Operating System

*This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

1. INTRODUCTION

Nowadays, the usage of Field Programmable Gate Array (FPGA) in the field of Reconfigurable Computing (RC) has become widely used. In particular the capability of a FPGA to be run-time reprogrammed makes its use for reconfigurable systems very attractive. Even more attractive is the emerging hybrid FPGAs, which has a hardcore or softcore general purpose processor (GPP) surrounded by a large field of reprogrammable logic. These new components open several interesting possibilities to design reconfigurable architectures for Systems on Chip (SoC). [2].

One of the challenges of our research is to provide support for run-time reconfigurable architectures, which may be used for self-optimizing systems. Such systems are used to realize and implement the control of transportation system like that one described in [1]. In dynamic environments, where application requirements may dynamically change, the concept of reconfigurable operating systems appears, which is emerging as new research field.

Differently from the normal approach where the design of such operating system (OS) is done offline, the proposed approach suggests the use of new partial reconfigurable architectures in order to support the development of a hardware/software reconfigurable operating system [6]. In this proposed architecture, the RTOS is capable to adapt itself to current application requirements, tailoring the RTOS components for this purpose. Therefore, the system continuously analysis the requirements and reconfigure the RTOS components at the hybrid architecture optimizing the use of system resources. Hence, the system is capable to decide on-the-fly which RTOS components are needed and also to which execution environment (CPU or FPGA) they will be assigned.

The paper focuses on an online partitioning algorithm for a real-time operating system services, which tries to minimize the whole resource utilization and the reconfiguration costs of the components.

The remaining of the paper is organized as follows: Section 2 presents a brief state-of-the-art analysis regarding hardware implementation of OS services their flexibilities. Then, section 3 shortly presents the architecture used. Section 4 presents the system formulation using 0-1 Integer Programming (BIP) and the reconfiguration costs evaluation. An analysis of the run-time execution of this evaluations, with an heuristic algorithm for the components assignment problem are presented in section 5. Section 6 presents some evaluation results using MATLAB to compare the proposed heuristic algorithm with the optimal solution provided by BIP. Finally, some conclusions and future work are shown in section 7.

Figure 1. Proposed architecture based on a Virtex-II Pro™ FPGA.

2. RELATED WORK

The idea of implementing OS services in hardware is not new. Several works in the literature, [12],[11],[10],[13],[15] and [14] show that hardware implementation may significantly improve performance and determinism of RTOS functionalities. The overhead imposed by the operating system, which is carefully considered in embedded systems design due to usual lack of resources, is considerably decreased by having RTOS services implemented in hardware. However, up to now all approaches have been based on implementations that are static in nature, that means, they do not change during run-time even when application requirements may significantly change.

In the field of reconfigurable computing, reconfiguration aspects have been concentrated at application level (see [7], [18] and [16]). At the OS level the research are limited to provide run-time support for those applications (see [10], [3], [20] and [19]).

In the approach presented in this paper we expand the usage of those concepts and the hardware support to the OS level. Additionally, based on the state-of-the-art analysis, for a self-optimized reconfigurable RTOS none such similar approach has been proposed yet.

3. ARCHITECTURE

Our target architecture is composed of one CPU, configurable logic elements (FPGA), memory and a bus connecting the components. Most of these elements are provided in a single CHIP, such as the Virtex II Pro™, from Xilinx company. The RTOS services that are able to be reconfigured are stored on an external SDRAM chip in two different implementations: as software object and as FPGA configuration bitstream (see Figure 1).

The Run-Time Reconfiguration (RTR) block (see [5]) provides the necessary services to the RTOS to manage the FPGA resources: module placement, loading and allocation. More detailed information about this architecture can be seen in [9] and [8].

4. PROBLEM DEFINITION

The problem of assigning RTOS components over the two execution environments can be seen as a typical assignment problem. Therefore, we decided to model the problem using Binary Integer Programming (BIP) [17] and [21]. A set of available services is represented as $S = \{s_{i,j}\}$, where every service i has its implementation for CPU ($j = 1$) or FPGA ($j = 2$). Every component has an estimated cost $c_{i,j}$, which represents the percentage of resource from the execution environment used by this component. On the FPGA it represents the circuit area needed by the component and at CPU it represents the processor used by it. Note that these costs are not static, since the application demands are considered to be dynamic. This topic will be addressed later on in subsection 4.2.

4.1 OS Service Assignment

The assignment of a component to either CPU or FPGA is represented by the variable $x_{i,j}$. We say that $x_{i,j} = 1$ if the component i is assigned to the execution environment j , and $x_{i,j} = 0$ otherwise. As some of the components may not necessary be needed by the current application, they should neither be assigned to the CPU ($j = 1$) nor to the FPGA ($j = 2$). Therefore, to proper represent this situation we consider that this component should stay at memory pool ($j = 3$). As we are focusing on the resource utilization optimization between CPU and FPGA we define that a component i placed on the memory pool ($j = 3$) does not consume any resource ($c_{i,3} = 0$). The definition of a third assignment place for an OS component is more useful for reconfiguration costs estimation, that will be seen in section 4.2.

The resources are limited, which derive two constraints for our BIP formulation: the maximum FPGA area available (A_{max}) and the maximum CPU workload (U_{max}) reserved for the operating system. Thus, the total FPGA area (A) and total CPU workload (U) used by the hardware components and the software components, respectively can not exceed their maximums. These constraints are represented by

$$U = \sum_{i=1}^n x_{i,1}c_{i,1} \leq U_{max}, \quad A = \sum_{i=1}^n x_{i,2}c_{i,2} \leq A_{max}$$

We also consider that a component i can be assigned just to one of the execution environment: $\sum_{j=1}^3 x_{i,j} = 1$ for every $i = 1, \dots, n$.

To avoid that one of the execution environment would have its usage near to the maximum, we specify a constraint to keep a balanced resource utilization (B) between the two execution environments: $B = |w_1U - w_2A| \leq \delta$. Where δ is the maximum allowed unbalanced resource utilization between CPU and FPGA. The weights w_1 and w_2 are used to proper compare the resource utilization between two different execution environments. If the resource used from an execution environment are not near to its maximum, it will have the capability to absorb some variation of the application demands. This characteristic are useful for real-time system in order to avoid the application to miss its deadlines due to workload transients. Note that this approach cannot guarantee hard real time constraints. However, for soft real-time systems it is useful.

The objective function used to minimize the whole resource utilization is defined as

$$\min \left\{ \sum_{j=1}^3 \sum_{i=1}^n c_{i,j} x_{i,j} \right\}$$

The solution of this BIP are the assignment variables $x_{i,j}$, which we define as being a specific system configuration: $\Gamma = \{x_{i,j}\}$.

4.2 Reconfiguration Costs

As is has been said in the section 4, the application requirements are considered to change over system life time. These modifications are represented by changes of the component costs $c_{i,j}$. This leads to the fact that a certain system configuration Γ may no longer be valid after application changes. Therefore, a continuously evaluation of the components partitioning is necessary. Whenever the systems reaches a unbalanced situation ($|w_1U - w_2A| > \delta$), the RTOS components should be reallocated in order to bring the system again in the desired configuration. In this situation, not just the new assignment problem need to be solved (Γ') again, but also the costs (time) necessary to reconfigure the system from Γ to Γ' need to be evaluated. This evaluation is necessary since we are dealing with real-time system. Thus, we have a limited time available for reconfiguration activities.

The reconfiguration cost of every component represents the time necessary to migrate a component from one execution environment to the other one. Therefore, we need to specify for every possible migration of a component its correspondent cost. As it was shown in section 4.1, our model assumes three different environments ($j = \{1, 2, 3\}$). The definition of the environment $j = 3$ (memory pool) is necessary to proper represent the case where a new OS service arrives in the system. This happens when the application requires a service that is neither at CPU nor at FPGA available, but it is stored at the memory pool. The same is valid for a service that leaves the system (it is not more needed by the application). So, we define for a component i a 3×3 size

migration costs matrix R_i . Let $R_i = \{r_{j,j'}^i\}$, where j and j' are the current and new execution environment of component i .

If $x_i = \{x_{i,1}, x_{i,2}, x_{i,3}\}$ and $x'_i = \{x'_{i,1}, x'_{i,2}, x'_{i,3}\}$ are the current and new assignment of the component i , then the complete reconfiguration cost K (total reconfiguration time) of the system is defined as:

$$K = \sum_{i=1}^n x_i^T R_i x'_i$$

In our current approach the migration costs associated with a component includes all necessary steps to remove a component from one execution environment to the other one. These steps represent the time to program the FPGA with a component or link the software component with in the CPU program, translate the context between different execution environments (when necessary), and also read the component instance from memory pool.

5. RUN-TIME ANALYSIS

As our operating system is being designed to support real-time applications, a deterministic behavior for service assignment and system reconfiguration need to be used in order to handle application time constraints.

5.1 Heuristic Algorithm for Assignment Problem

The solution of an BIP finds an optimal solution for the assignment problem. For a small set of components this approach is very suitable. However, it is too computationally complex to solve all problem sizes. Therefore, we are currently using an heuristic greedy based algorithm for this problem.

The algorithm starts selecting the component that has the smallest cost among them needed by the current application and it assigns this component to the correspondent execution environment. It then selects the component that has the smallest cost among the remaining unassigned ones so that it tries to keep the usage of CPU resource U equal to the FPGA resource A . This process is repeated until all components have been assigned. The algorithm terminates by checking if the CPU and FPGA resources usage constraint are fulfilled: $U \leq U_{max}$; $A \leq A_{max}$. If so, the algorithm returns a valid assignment solution, or an error otherwise. It can be seen that this algorithm has a polynomial complexity of $O(n^2)$, since we have just one *for* loop which produce n searches in a list of (maximum) n elements. This algorithm is also shown in Algorithm 1.

Note that this algorithm do not take into consideration the balancing constraint δ . So, there is no guarantee that it will provide a solution which fulfills this constraint. Therefore, a second algorithm is proposed that improves the balancing B in order to meet the δ constraint. It is based on Kernighan-Lin

Algorithm 1 Heuristic for service assignment

```

 $C_1 = \{c_{i,1}\}$  Set of components available for CPU ( $j = 1$ )
 $C_2 = \{c_{i,2}\}$  Set of components available for FPGA ( $j = 2$ )
 $X_1 = \{x_{i,1}\}$  Assignment of CPU components
 $X_2 = \{x_{i,2}\}$  Assignment of FPGA components
 $C = C_1 \cup C_2$ ;  $X = X_1 \cup X_2$ ;  $U = A = 0$ 
 $n =$  number of components
for  $k = 1$  to  $n$  do
  if  $U \leq A$  then
    Find an unassigned component  $i$  among  $\{c_{i,1}\}$  so that it has the smallest
    cost.
    Assign this component to CPU:  $x_{i,1} = 1$ ;  $x_{i,2} = 0$ 
  else
    Find an unassigned component  $i$  among  $\{c_{i,2}\}$  so that it has the smallest
    cost.
    Assign this component to FPGA:  $x_{i,2} = 1$ ;  $x_{i,1} = 0$ 
  end if
   $U = C_1 X_1^T$ 
   $A = C_2 X_2^T$ 
  if  $U > U_{max}$  or  $A > A_{max}$  then
    Exit with error: "Not feasible"
  end if
end for
return  $X$ 

```

algorithms [4] and it aims to obtain a better balancing B than the first one by swapping pairs of components between CPU and FPGA. The algorithm receives as input the assignment solution X provided by the first algorithm which has $nc_1 = \sum_{i=1}^n x_{i,1}$ components assigned to CPU and $nc_2 = \sum_{i=1}^n x_{i,2}$ components assigned to FPGA. The maximum number of pairs that are possible to be swapped is defined as: $max_pairs = \min(nc_1, nc_2)$.

By moving a component i , previously assigned to the CPU, to the FPGA ($\{x_{i,1} = 1; x_{i,2} = 0\} \Rightarrow \{x_{i,1} = 0; x_{i,2} = 1\}$), we have a new balancing B : $B_{new} = |B_{current} - s_i|$, where $s_i = \{c_{i,1} + c_{i,2}\}$. Similarly, by moving a component i from FPGA to CPU, the new balancing B will be: $B_{new} = |B_{current} + s_i|$. Thus, swapping a pair of components o, p ($\{x_{o,1} = 1; x_{o,2} = 0\}; \{x_{p,1} = 0; x_{p,2} = 1\}$), the new balancing B is defined as: $B_{new} = |B_{current} - s_o + s_p|$. Similarly, $B_{new} = |B_{current} + s_o - s_p|$ if $\{x_{o,1} = 0; x_{o,2} = 1\}; \{x_{p,1} = 1; x_{p,2} = 0\}$. Additionally, we define G_{op} as the gain obtained in the balancing B by swapping a pair o and p of components:

$G_{op} = B_{new} - B_{current}$. A gain below 0 means an improvement obtained in the balancing B .

Algorithm 2 Heuristic for balancing B improvement

```

 $X_1^{init} = \{x_{i,1}\}$  Initial assignment of CPU components
 $X_2^{init} = \{x_{i,2}\}$  Initial assignment of FPGA components
 $X^{init} = X_1^{init} \cup X_2^{init}$ 
 $X^{new} = X^{init}$ 
 $B^{init} = |U^{init} - A^{init}|$ 
 $B^{new} = B^{init}$ 
 $m = max\_pairs$  maximum number of pairs
repeat
  for  $k = 1$  to  $m$  do
    Find the pair  $o, p$  ( $\{x_{o,1} = 1; x_{o,2} = 0\}; \{x_{p,1} = 0; x_{p,2} = 1\}$  or
     $\{x_{o,1} = 0; x_{o,2} = 1\}; \{x_{p,1} = 1; x_{p,2} = 0\}$ ) so that  $o$  and  $p$  are un-
    locked and  $G_{op}$  is minimal
    if  $G_{op} < 0$  then
      Swap  $o$  and  $p$  changing the current assignment  $\Rightarrow X^{new} = (X^{new}$ 
      with  $o$  and  $p$  swapped)
       $B^{new} = B^{new} + G_{op}$ 
      Lock  $o$  and  $p$ .
    end if
  end for
until  $G_{op} \geq 0$  OR  $B^{new} < \delta$  OR all pairs are locked
return  $X^{new}$ 

```

The algorithm starts trying to swap all possible pairs and storing the gain obtained by every try. It then chooses the one which provides the smallest gain. If this gain is bigger than or equal to zero, no swap is able to provide an improvement in the balancing B and the algorithm stops. Otherwise, the pair that provides the smallest gain are swapped and locked (no longer a candidate to be swapped). This process is repeated until all pairs have been locked or no improvement can be obtained by any interchange or if the δ constraint has been fulfilled. The algorithm terminates by returning the new assignment solution X that provides a better (or at least an equal) balancing B than the solution provided by the first one. The algorithm for balancing B improvement is shown in Algorithm 2.

5.2 System Reconfiguration

For real-time applications, the reconfiguration time must also be taken under consideration. Depending on the decision of the assignment algorithm (which generates a new system configuration Γ') the total system reconfiguration time

K required may be bigger than the current time allowed by the application. Therefore, the reconfiguration of the OS components need to be done separately in time in order to still able the application to meet its time constraints. It means that a schedule for components reconfiguration needs to be found. Some technics for component reconfiguration scheduling are currently being investigated. However, this topic will not be treated in the scope of this paper.

6. EXPERIMENTAL RESULTS

For system evaluation of the run-time assignment problem, we made some simulations using MATLAB tool. We generated a number of 100 different systems having randomly costs: $1\% \leq c_{i,1} \leq 15\%$ and $5\% \leq c_{i,2} \leq 25\%$; and fixed size: $n = 20$ components. The maximum FPGA resource was defined to be 100% ($A_{max} = 100$), as well as for the CPU ($U_{max} = 100$). The components assignment were calculated for every system using the 0-1 Integer Programming (optimal solution) and the heuristic algorithm (first and second one). The average value of total cost ($U + A$) and the absolute difference cost ($|w_1 U - w_2 A|$) were compared for different values of δ (the resource usage balancing restriction): (0.5, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50 and 60).

The solutions provided by the first heuristic algorithm were very similar to the optimal one, if the δ constraint has values around ($\delta \approx 10\%$), concerning the fulfil of this constraint (see Figure 2, Heuristic-1). The smaller the δ the poor the results given by the first heuristic algorithm. This was expected to be so, since the first algorithm do not consider the balancing restriction.

The appliace of the second algorithm over the solution provided by the first one deliver a better balancing B . However, an increase in the total cost

Figure 2. Unbalance average for different δ constraints.

Figure 3. Total cost assignment average for different δ constraints.

assignment was verified for the cases where the second algorithm achieved an improvement in the balancing B ($\delta < \approx 10\%$). Nevertheless, the total cost assignment achieved by this heuristic algorithm were quite satisfactory: maximum of 15% bigger if compared with the optimal case (see Figure 3, Heuristic-2).

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented our investigation towards a run-time reconfigurable RTOS running over a hybrid platform, focusing in the OS service assignment and system reconfiguration. Looking at the related work, we are quite convinced that this is a novel approach for a self-optimized RTOS.

A shortly presentation of the hardware architecture based on a Virtex-II ProTMFPGA was also presented. The 0-1 Integer Programming model of the system and the reconfiguration cost evaluation have been presented. Additionally, considerations of a run-time execution of this technics, in order to support real-time applications have been discussed.

As a future work, the investigation of a proper OS components assignment algorithm which takes into consideration the application time constraints and the integration of the communication costs among the components are going to be made. Technics to able the run-time reallocation of the components over an hybrid architecture will be investigated. Moreover, a reconfiguration scheduling of the components, necessary to bring the system to a new configuration ($\Gamma \Rightarrow \Gamma'$) are going to be made.

REFERENCES

- [1] <http://nbp-www.upb.de/>.
- [2] David Andrews, Douglas Niehaus, and Peter Ashenden. Programming models for hybrid cpu/fpga chips. *Computer - Inovative Thecnology for Computer Professionals*, 37:118–120, January 2004. IEEE Computer Society.
- [3] Krishnamoorthy Baskaran, Wu Jigang, and Thamipillai Srikanthan. Hardware partitioning algorithm for reconfigurable operating system in embedded systems. In *Sixth Real-Time Linux Workshop*, pages 117–123, November 2004. Singapore.
- [4] Petru Eles, Krzysztof Kuchcinski, and Zebo Peng. *System Synthesis with VHDL: A Transformational Approach*, chapter 4, pages 114–119. Kluwer Academic Publishers, 1998.
- [5] Björn Griese. Eingebettete prozessoren in rekonfigurierbaren strukturen. Studienarbeit, Schaltungstechnik, Heinz Nixdorf Institut, Universitat Paderborn, feb 2003.
- [6] Marcelo Götz. Dynamic hardware-software codesign of a reconfigurable real-time operating system. In *International Conference on Reconfigurable Computing and FPGAs 2004 (ReConFig04)*, pages 330–339. Mexican Society of Computer Science, SMCC, 20-21, September 2004.
- [7] J. Harkin, T. M. McGinnity, and L. P. Maguire. Modeling and optimizing run-time reconfiguration using evolutionary computation. *Trans. on Embedded Computing Sys.*, 3(4):661–685, 2004.
- [8] Heiko Kalte, Mario Porrmann, and Ulrich Rückert. A prototyping platform for dynamically reconfigurable system on chip designs. In *Proceedings of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC)*, Hamburg, Germany, 2002.
- [9] Boris Kettelhoit, Alexander Klassen, Carlos Paiz, Mario Porrmann, and Ulrich Rückert. Rekonfigurierbare hardware zur regelung mechatronischer systeme. In *3. Paderborner Workshop: Intelligente mechatronische Systeme*, HNI-Verlagsschriftenreihe, 2005.
- [10] Paul Kohout, Brinda Ganesh, and Bruce Jacob. Hardware support for real-time operating systems. In *International Symposium on Systems Synthesis*, pages 45–51. Proceedings of the 1st IEEE/ACM/IFIP International conference on HW/SW codesign and system synthesis, 2003.
- [11] P. Kuacharoen, M. Shalan, and V. Mooney. A configurable hardware scheduler for real-time systems. In *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pages 96–101, June 2003.
- [12] Jaehwan Lee, Vicent John Mooney III, K. Ingström, Anders Daleby, T. Klevin, and Lennart Lindh. A comparison of the rtu hardware rtos with a hardware/software rtos. In *ASP-DAC2003, (Asia and South Pacific Design Automation Conference)*, page 6, 2003. Japan.
- [13] Jaehwan Lee, Kyeong Ryu, and Vincent John Mooney III. A framework for automatic generation of configuration files for a custom hardware/software rtos. In *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, June 2002.
- [14] Lennart Lindh. Fasthard - a fast time deterministic hardware based real-time kernel. In *IV Euromicro Workshop on Real-Time Systems*, pages 21–25, June 1992.
- [15] Lennart Lindh and Frank Stanischewski. Fastchart - a fast time deterministic cpu and hardware based real-time-kernel. In *EUROMICRO'91*, pages 12–19, 1991. Paris, France.

- [16] Jean-Yves Mignolet, Vincent Nollet, Paul Coene, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *DATE*, pages 10986–10993, 2003.
- [17] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, 1988.
- [18] Heather Quinn, Laurie A. Smith King, Miriam Leeser, and Waleed Meleis. Runtime assignment of reconfigurable hardware components for image processing pipelines. In *FCCM*, pages 173–, 2003.
- [19] Herbert Walder and Marco Platzner. A runtime environment for reconfigurable hardware operating systems. In *FPL*, pages 831–835, 2004.
- [20] Grant Wigley and David Kearney. The development of an operating system for reconfigurable computing. In *FCCM*, April 2001.
- [21] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.