MODELLING VECTOR CONTROL SYSTEM RECONFIGURATION

József VÁSÁRHELYI[1], Maria IMECS[2], Ioan I. INCZE[2], Csaba SZABÓ[2]
[1] Department of Automation University of Miskolc
H-3515 Miskolc Egyetemváros, Hungary
vajo@mazsola.iit.uni-miskolc.hu
[2] Technical University of Cluj-Napoca
P.O. 1, Box 99, RO-3400, Cluj-Napoca, Romania
{imecs, Ioan.Incze, Csaba.Szabo}@edr.utcluj.ro

Abstract: To apply run-time reconfiguration there is need for an appropriate model to understand reconfiguration. System on chip technology allowed the use of complex low gate/price new type of FPGA. Modelling reconfiguration and using Matlab tools is possible to simulate and rapid prototype vector control systems. The paper presents the reconfigurable models as introduced by Luk, Cheung, Shirazi and Athanas and will present the reconfiguration model for vector control systems.

Keywords: run-time reconfiguration, rapid prototyping, simulation, reconfigurable logic, field-orientation principle, vector control, variable speed drives.

1. Introduction

Configurable computing achieving high performance improvements found its wide range of applications like image processing, fuzzy controllers, compression, morphology, feature extraction, object tracking, computational chemistry, AC drive control, etc.[1.], [2.]. In this paper, we will consider some general aspects of the reconfiguration and the run-time reconfiguration modelling aspects will be treated.

The performance improvement results in magnitude improvement over microprocessors. The so called final performance approved by an application implemented in reconfigurable support depends not only the hardware itself, but also the availability of strong parallel resources as well as on the development environments (CAD support, compiler). Therefore, two main aspects have to be considered when customized reconfigurable computing systems are built [3.]:

- Control Structures and the data dependency of future applications have to be reconsidered in order to identify simultaneously occurring operations for higher performances [2.].
- The availability and possibility of such 'programmable platforms' structure is essential. Obviously, there exists a so called 'software gap' between software compilers and compilers for custom computing machines [4.].

One of the limitations of contemporary FPGAs is the reconfiguration mechanism. There are primary approaches commonly adopted:

- *serial configuration*: In devices using serial configuration, the configuration storage elements are connected as a large scan chain around the entire chip. During configuration, the programming information is loaded into the device and shifted throughout in a bit wise manner.
- *parallel configuration*: parallel loading of the configuration data is possible with some devices, close examination of the timing employed by these devices reveals an internally serial architecture. Most often, the entire chip must be programmed in such fashion before any part of it may be used to perform computation.
- *context switching configuration*: several configuration data is stored in the configuration memory of the chip or outside the chip and the reconfiguration of the chip is made by switching between the stored configuration contexts.
- *random access configuration*: this type of configuration mode can be applied to any of previous configurations at any time not necessarily as part of the power up process.

To allow run-time reconfiguration it is necessary to construct corresponding reconfiguration models. In particular, an appropriate model is necessary to provide the basis for techniques involved in producing efficient reconfigurable systems and in analysing their tradeoffs. Luk introduced some reconfiguration models in [5.]. An overview of the introduced reconfiguration models will be given in the following section.

2. Modelling Reconfiguration

An appropriate model is often the key to understand a new technology and to exploiting it effectively. The main difficulty in understanding reconfiguration is its dynamic nature. First, a simple model was proposed by Luk, which uses a static network to capture this dynamic behaviour [5.].

The basic idea is straightforward. A block that can be configured to behaved either as *P* or as *Q* function is described by a network with *P* and *Q* sandwiched between two control blocks C and C'. *C* and *C'* are responsible for routing the data and results from the external ports *x* and *y* to either *P* or *Q* at the desired instant; the choice can be determined by run-time conditions. Possible implementations of *C* and *C'* may be *a demultiplexer* and *a multiplexer*, but one should note that, while these components may indeed be possible implementations, *C* and *C'* are intended to be abstract entities, which need not be implementable [5.].

Run-time reconfiguration is done while the system executes the designated task. To improve system performances pipeline reconfiguration technique was suggested by Bittner and Athanas in [6.].

2.1. Pipeline morphing and Virtual Pipelines

Pipeline morphing is simple but effective technique for reconfiguring pipelined FPGA designs at run-time. By overlapping computation and reconfiguration, the latency associated with emptying and refilling a pipeline can be avoided.

Implementing pipeline architectures using reconfigurable devices is attractive for several reasons. Partial reconfiguration for a system operating in an unpredictable environment such as vector control for AC drives this possibility enables the selection of functions adaptively. Partial reconfiguration is a powerful method of exploiting the flexibility of an FPGA, which can be reconfigured while other parts are continuing to function. Pipelines provide a simple but effective scheme for partial reconfiguration, since pipeline registers isolate one pipeline stage from another so that computation and reconfiguration can take place at the same time without interference. The regular structure of pipelines also simplifies the development of hardware operators, which can be relocated to different regions of a pipeline, maximizing the reuse of design effort.

Luk proposed an obvious method for reconfiguring an *n - stage* pipeline; this involves three steps [7.]:

- First, one needs to complete the current computation and clear the pipeline this takes n cycles.
- Then reconfiguration can take place.
- Finally, one has to wait for n cycles for the result to flow through the newly configured pipeline.

This method of reconfiguring a pipeline leads to a latency of *2n* cycles, in addition to the time for reconfiguring all the pipeline stages. In highly pipelined systems when n is large, the pipeline latency cycles and reconfiguration time will have a significant impact on performance. The basic idea is to overlap computation and reconfiguration: the first few pipeline stages are being reconfigured to implement new functions so that data can start flowing into the newly configured stages of the pipeline, while the rest of the pipeline stages are completing the current computation. Instead of changing the entire pipeline at once, this method involves morphing one pipeline to another. The pipeline registers isolate one pipeline stage from its neighbour, enabling computation and reconfiguration to take place concurrently in different stages.

Fig.1 shows how a three-stage pipeline F can be morphed into a pipeline G in three steps. It should be clear from the example that during morphing, the flow of reconfiguration is synchronous with the flow of data, and hence the pipeline latency cycles are eliminated. If the time for reconfiguration is longer than the pipeline processing time, the pipeline will need to include flow control mechanisms to slow down the rate of data flow while morphing is taking place. Whether morphing is used or not, a designer's task is to ensure that,

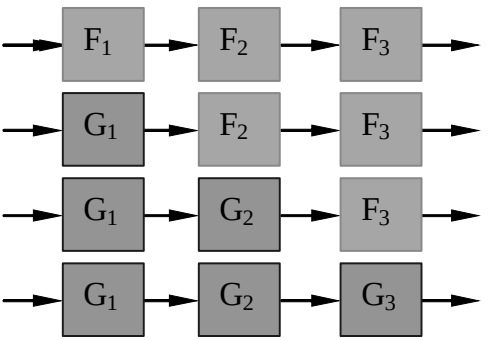


Fig.1. Pipeline Reconfiguration

the slowing down due to run-time reconfiguration will not affect the system performance.

Because of the elimination of latency cycles, pipeline morphing will improve the performance of systems that reconfigure at run-time. It is particularly suitable for devices supporting rapid reconfiguration, and it works best when reconfiguration time is comparable to the pipeline computation time. To meet this condition, the user can build single cycle reconfigurable structures in an FPGA. Morphing can also be applied to systems with multiple FPGAs arranged as a pipeline.

The method proposed by Athanas and later by Luk is not defined to linear pipelines. It can be applied to pipelines of other shapes, such as two dimensional meshes or tree shaped designs.

2.2. Virtual Pipelines

An advantage of adopting a pipeline structure is the ease of mapping a large virtual pipeline onto a small physical pipeline. The approach involves feeding back partial results to the physical pipeline, which morphs between different sections of the virtual pipeline. The performance of such a system can often be enhanced by a temporary storage ***Fig. 2***.The mapping of a six stage virtual pipeline is implemented onto a three stage physical pipeline. The first three stages of the virtual pipeline are time multiplexed with the last three stages. Note that the physical pipeline operates in two modes: the *"fill mode"* and the *"feedback mode."*

In *the fill mode*, the first pipeline stage is connected to the external input and data start filling up the pipeline. Once the pipeline is filled up, partial results will emerge and will be stored in the temporary storage.

When all input data have been processed by the first three stages of the virtual pipeline or when the temporary storage is full, the pipeline will operate in the feedback mode.

When an n stage physical pipeline first starts in the feedback mode, its first stage will be reconfigured to become the (n+1)-th stage of the virtual pipeline. Temporary storage can be implemented in different ways. If a large amount of temporary storage is required, then external memory can be used; otherwise onchip registers or embedded memories within the FPGA may be sufficient. As explained above, pipelines supporting rapid reconfiguration can afford a small temporary storage. When this happens, the feedback connections can be made entirely onchip, possibly using global connections in the FPGA so that output data from the last stage can feedback to the first stage. Global connections are provided in most of FPGAs; such connections can themselves be pipelined to ensure high performance.

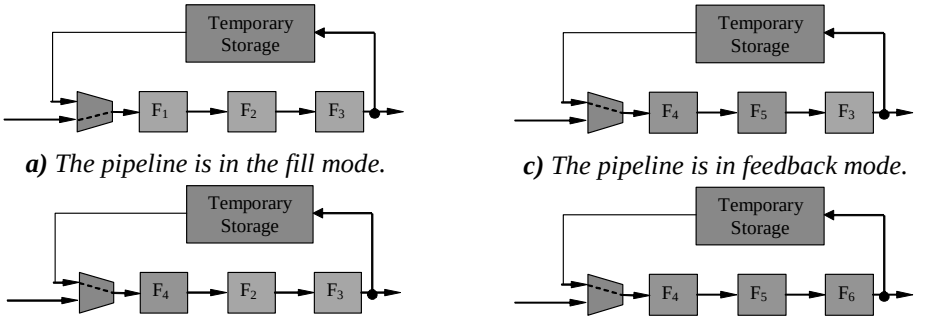
The pipeline morphing method with temporary storage elements presented by Luk **[7.]** is an effective technique for reconfiguring systems, which can be organised in pipeline structure.

3. Reconfiguring Control Systems

The reconfiguration methods presented by Bittner and Athanas **[6.]** and Luk **[7.]** is applicable in the case were no feedback is in the system. If one consider control systems such as motor control, which has feedback from the plant, and it is characterised with high

dynamic, then in the reconfiguration there should be considered the response of the plant, and the actual output variable values have to be stored in temporary registers.

Considering the general vector control structure presented in [1.] run-time reconfiguration of a vector control system can be modelled as shown in ***Fig. 3***.



a) *The pipeline is in the fill mode.*

c) *The pipeline is in feedback mode.*

b) *The pipeline is in feedback mode.*

d) *The pipeline is in feedback mode*

Fig. 2. Emulation of a six stage virtual pipeline F1 … F6 using a three stage physical pipeline. The control of the switch that selects the external or the feedback data is not shown.

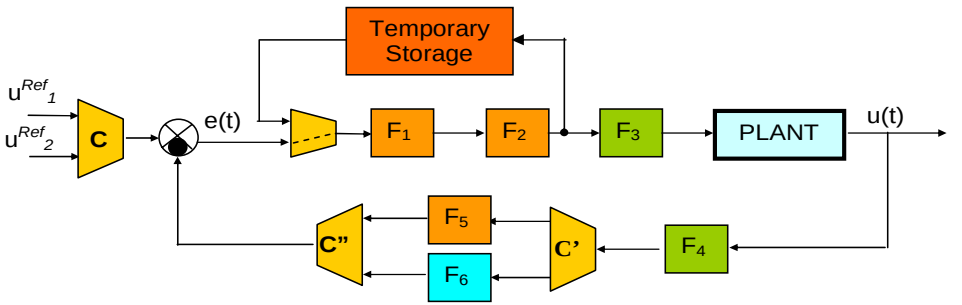


Fig. 3. Control system pipeline

Fig. 3 composed of reconfigurable control blocks (C, C', C"), there are blocks, which are not reconfigured at all (F_3, F_4) and there are blocks, which are reconfigured on demand (F_1, F_2, F_5, and F_6). The run-time reconfiguration is made in two different ways in output variable computation blocks (F_1, F_2) and in the feedback (F5, F6).

In the feedback loop, reconfiguration will be from block F_5 to F_6, which means F_5 executes a different function compared to F_6 and no temporary storage is necessary. F_5 and F_6 may compute different output variable values. In the control loop F_1 and F_2 are reconfigured, conform ***Fig. 2*** and may compute the same/different output variable

depending on the reference of the control structure (stator-field, rotor-field). Under these circumstances the block F3 having the same structure may compute different output variables with different range (voltage, current).

In this case the new control structure is known before reconfiguration and it is downloaded in run-time, and compiled in advance.

Modelling reconfiguration helps rapid prototyping. Usually reconfiguration can be done in run-time, during compilation time or in any other circumstances. In the case of vector control structures, the new control structure is known before reconfiguration and it is downloaded in run-time, and compiled in advance. Using the presented reconfiguration model simulation and rapid prototyping can be made by the use of an intellectual property library [1.] in Matlab Simulink environment.

Acknowledgment

The authors are grateful to Triscend Inc. and Xilinx Inc. for donations, which made possible the research on some aspects of reconfigurable vector control framework.

This publication is subject of the scientific and technological Hungarian-Romanian intergovernmental and sponsored by the Department of Development and Research of the Hungarian Ministry of Education and its contract partner the Romanian Ministry of Education Research and Youth, research is part of the Project TET 16/2003.

4. References

[1.] VÁSÁRHELYI, J., IMECS Mária, INCZE J. J., SZABÓ Cs.: *Module Library for Rapid Prototyping and Hardware Implementation of Vector Control Systems.* In: Proceedings of International Conference on Intelligent Engineering Systems, INES 2002. pp. 447-452. Opatija, Croatia.

[2.] IMECS M., BIKFALVI P, NEDEVSCHI S., VÁSÁRHELYI J.: Implementation of a Configurable Controller for an AC Drive Control a Case Study**,** In: Proceedings of IEEE Symposium on FCCM 2000, Napa, California, USA, 2000 pp. 323-324.

[3.] BREBNER G., *Field-Programmable Logic: Catalyst for New Computing Paradigms,* In*:* Field Programmable Logic and Applications, From FPGAs to Computing Paradigm, 8th International Workshop, Proceedings FPL'98, Tallinn, Estonia, August 31 – September 3, 1998, ISBN 3-540-64948-4, , 1998, pp. 49-58.

[4.] HAUCK S., *The Roles of FPGAs in Re-programmable Systems***,** Proceedings of the IEEE, Vol. 86, No. 4, April 1998, pp. 615639.

[5.] LUK W., SHIRAZI N., CHEUNG P., *Modelling and Optimizing Run-time Reconfigurable Systems,* Proceedings FCCM96, IEEE Computer Society Press, 1996, pp. 167 – 176.

[6.] BITTNER R., ATHANAS P., *Wormhole Run-time Reconfiguration*, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, California, USA, February 1997 pp.79-85.

[7.] LUK W., SHIRAZI N., GUO S. R., CHEUNG P., *Pipeline Morphing and Virtual Pipelines,* Proceedings on 7th International Workshop on Field Programmable Logic and Applications, FPL'97, 1997, pp. 111-120.