Actas das

III Jornadas sobre Sistemas Reconfiguráveis

8 e 9 de Fevereiro de 2007

INSTITUTO SUPERIOR TÉCNICO

inesc id lisboa

Editores:

Horácio Neto
António Esteves

FPGA Architectures for Reconfigurable Computing

Manuel G. Gericota, Gustavo R. Alves
DEE/ISEP
Rua Dr. Antº Bernardino de Almeida
4200-072 Porto
PORTUGAL
{mgg, gca}@isep.ipp.pt

José M. Ferreira
DEEC/FEUP
Rua Dr. Roberto Frias,
4200-465 Porto
PORTUGAL
jmf@fe.up.pt

Abstract

To accelerate the execution of an application, repetitive logic and arithmetic computation tasks may be mapped to reconfigurable hardware, since dedicated hardware can deliver much higher speeds than those of a general-purpose processor. However, this is only feasible if the run-time reconfiguration of new tasks is fast enough, so as not to delay application execution. Currently, this is opposed by architectural constraints intrinsic to current Field-Programmable Logic Array (FPGA) architectures. Despite all new features exhibited by current FPGAs, architecturally they are still largely based on general-purpose architectures that are inadequate for the demands of reconfigurable computing. Large configuration file sizes and poor hardware and software support for partial and dynamic reconfiguration limits the acceleration that reconfigurable computing may bring to applications.

The objective of this work is the identification of the architectural limitations exhibited by current FPGAs that prevent reconfigurable computing systems to achieve a high efficiency and performance and the proposal of alternatives to its resolution.

1. The Problem

In recent years, reconfigurable computing attracted a lot of attention due to its promise to deliver the high performance provided by reconfigurable hardware along with the flexibility of general purpose processors. This perception is based upon a number of technological advancements that led to current generation of mega-gate chips and to a diverse range of features. One particular feature is the ability to partially reconfigure the FPGA enabling the configuration of application-specific hardware at run-time. Portions of an application, usually referred to as tasks, are mapped to the reconfigurable hardware, while the general-purpose processor handles other tasks. The aim is to accelerate application execution, by transferring the execution of computing-intensive highly-repetitive tasks to application-specific hardware implemented in reconfigurable devices. This means that the reconfigurable device attached to the system is reused for the implementation of different logic and arithmetic tasks, being reconfigured frequently during run-time.

However, despite all the new features, current FPGAs are still largely based on general-purpose architectures that are inadequate for the demands of reconfigurable computing. Some architectural limitations hamper firstly the placement of new tasks, and secondly their fast relocation into the FPGA, preventing the efficient management of the logic space available, and, as a result, leading to a delay on task execution, compromising the benefits of the reconfigurable computing concept. Large configuration file sizes and poor hardware and software support makes partial and dynamic reconfiguration difficult and inefficient. Furthermore, originally designed for rapid prototyping, FPGAs do not emphasize rapid reconfiguration, which limits the acceleration reconfigurable computing may add to applications.

Depending on the application flow, configuration loadings may be carried out along with other computational tasks performed by the host processor, thereby reducing reconfiguration overhead. But this is only true if the application flow, and therefore the sequence of tasks to be configured, is known in advance. Furthermore, execution delays will only be avoided if enough reconfigurable resources are available when required to implement incoming tasks, meaning that tasks have to be pre-synthesised. Otherwise, the time it takes to synthesise them at runtime may lead to an application halt. Hence, there is no flexibility on the placement of tasks and extra resources may be needed just to try to ensure that all tasks will be available when required, decreasing system efficiency.

In addition, since different tasks have specific spatial and temporal requirements, when resources are allocated to enable their implementation and later released, many small areas of free resources are created. These portions of unallocated resources tend to become so small that they fail to satisfy any allocation request due to insufficient adjacent free resources, remaining unused – the FPGA logic space gets fragmented. Yet, a suitable relocation of a subset of the executing tasks might solve it [1].

2. Requirements and proposals

Therefore, efficient reconfigurable computing systems require both the fast allocation and the fast relocation of tasks, but these are features that current FPGA architectures are unable to support. To tackle with these problems it becomes imperative to study and define the requirements that an FPGA targeted at improving reconfigurable computing must fulfil to enable fast dynamic reconfiguration, and to propose a new architecture able to cope with such demands.

Performance degradation that occurs when tasks are relocated is presently a major obstacle to the efficient management of the logic space in an FPGA, as it limits the feasibility of relocating tasks. To be able to fit a new incoming task into the free space available on the FPGA, instead of having a fixed placement it is necessary to have relocatable mappings and a symmetric architecture (i. e. the resources in the FPGA are the same across the entire FPGA, though some local heterogeneity is possible) [2].

Current FPGA present some architectural features that aim to improve the performance of certain types of tasks. As an example, counters benefit from the existence of fast dedicated carry interconnections between adjacent logic blocks. The propagation delay of the carry signal is significantly lower because no pass transistors (usually used to route signals along the interconnect resources) exist in this path. The parasitic capacity of the pass transistors is the main cause of introduction of propagation delays in routed signals in FPGAs. Their inexistence on the dedicated carry paths greatly contributes to improve the maximum allowable frequency of operation, enabling the implementation of faster counters. Paradoxically, the existence of carry paths is also an obstacle to the achievement of a higher efficiency in current reconfigurable computing systems, because in current commercial FPGAs these dedicated carry resources enable only to interconnect vertically adjacent logic blocks. If a counter needs to be relocated from a vertical to a horizontal placement, or if the resources available to be allocated to an incoming counter span an horizontal area rather than a vertical one, carry signals have to be propagated through generic interconnection resources (since no dedicated carry paths exist to interconnect horizontally adjacent logic blocks). Past experiments performed using a 24-bit counter indicated an 80% decrease on performance when a vertically placed counter was relocated horizontally [3].

One of the earlier examples of a simple, symmetrical, hierarchical and regular architecture were the XC6200 FPGAs from Xilinx, composed of a large array of simple, configurable cells. Each basic cell contained a computation unit capable of simultaneously implementing one of a set of logic level functions and a routing area through which inter-cell communication could take place [4].

However, more and more, architectures are becoming heterogeneous with the introduction of memory blocks, multiplier blocks and of dedicated Digital Signal Processing (DSP) blocks distributed among the FPGA array meant to accelerate the execution of specific types of tasks. In sum, FPGAs evolved from being just a fine grained architecture to more of a mixed grain architecture [5]. The challenge is to retain those features (or, if necessary, to reinforce them or even to introduce others) without compromising the homogeneity and symmetry of the new architecture and therefore the possibility of relocating tasks anywhere throughout the configuration space.

However, to easily relocate a task the symmetry of the logic and routing resources is not enough. It is also indispensable to take into account the addressing structure of the configurable memory cells that will support FPGA logic configuration. The relocation of a task from one position to another involves the displacement of each one of the bits present in its configuration bitstream, which defines the configuration of the task into the FPGA.

The earlier XC6200 FPGAs had a full parallel CPU interface, referred to as ‘FastMAP‘, which made all the configuration SRAM and logic cells appeared as conventional memory mapped SRAM. This built-in, memory-like interface simplified system design and directly interfaced to most embedded processors without consuming any FPGA resources, providing high-speed access to all internal registers in the logic cells. Any register could be mapped into the memory address space of the host processor, allowing for simple hardware and fast data transfers. These capabilities allowed XC6200 FPGAs to support virtual hardware in which circuits that run on the FPGA could be saved (`swapped out’) to allow the FPGA resources to be assigned to a different task, then restored (`swapped in’) at a later time with the same internal state in their registers [4].

This option may be a solution, considering that most recent FPGAs possess embedded processors, which may mean faster access interfaces. Despite of

that, it may also be a bit cumbersome, taking into account the amount of reconfiguration memory to be mapped and the heavy load a recalculation of configuration addresses in case of relocation might mean for the processor. Furthermore, the XC6200 is not on the market anymore and its addressing structure has not been reused commercially ever since, which may be a warning of its inefficiency.

A straightforward solution is to use a two-part base address to identify each logic block (its x and y coordinates inside the FPGA), as illustrated in figure 1. The word length would be equal to the logic block height, in terms of configuration bits, and the number of words equal to the logic block length. Since all logic blocks have the same size, a task may be relocated just by adding or subtracting the two--part address by the amount of logic blocks to be displaced. In current architectures the minimum configuration area spans more than one logic block, meaning that when a logic block is configured other logic blocks may be adversely affected by that.

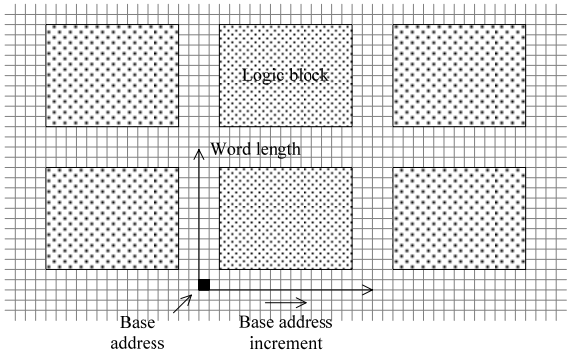

Figure 1: Proposal for the configuration memory addressing scheme

To keep the homogeneity of the configuration bitstream and to make it simple to relocate tasks just by recalculating the base addresses of the logic blocks that they occupy, the interconnection configuration bits, responsible by the routing of the internal signals of each task, have also to be associated to each logic block.

Additionally, it is necessary to preserve and replicate the current state condition of the relocated task, avoiding data incoherencies due to transfer failures or loss of state updates that may have taken place in the meanwhile. Just by enabling a fast and reliable relocation of tasks it will be possible to improve the performance and efficiency of reconfigurable computing systems.

To replicate a task that implements a purely combinational function, a two-phase replication procedure is enough to assure that the relocation occurs without disturbing its operation. In the first phase the internal configuration of the logic blocks and the interconnections among them are copied into their new location and input data is applied to both tasks simultaneously. In the second phase, the output data feeder is switched from the original task to its copy. The gap between first and second phase assures the stabilization of output data and the synchronization of both implementations.

To replicate a task that implements a sequential function the relocation mechanism has to do more than just copying the functional specification of the logic blocks to be replicated: the internal state information must also be copied. In current architectures it is only possible to perform direct read operations of the flip-flops contents. No specific mechanism is provided to enable the transference of the current logic state of a flip-flop while assuring data coherency. The possibility of directly accessing the memory cell of a flip-flop to read and write its content, while adequate to enable the pre-emption of tasks, is not sufficient to ensure a successful replication because the circuit state may be updated between the two operations. When dealing with synchronous free-running clock circuits, the two-phase relocation procedure described previously is a good solution. Between the first and the second phase the replicated logic blocks receive the same input data as the original logic blocks, and all their flip-flops acquire the same state information. When using synchronous gated-clock circuits, where input acquisition by the flip-flops is controlled by the state of the clock enable signal, the previous method does not ensure that the replicated logic blocks capture the correct state information, because the clock enable signal may not be active during the relocation procedure. Besides, it is not feasible to simply set this signal as part of the relocation procedure, because the value present at the input of the replica flip-flops may change in the meantime, and a coherency problem will occur. A solution to manage the transference of the state information from the original flip-flops to their replicas, while enabling state update by the circuit at any instant, without delaying the relocation procedure and assuring coherency, needs to be developed.

The communication between tasks and the host processor is an issue of its own that needs to be addressed as well. A possible approach for the implementation of a communication mechanism is the use of a DyNoC (Dynamic Network-on-Chip) in which a fixed, non-configurable, grid of routers and interconnections is established separately from the resources available in the configurable logic space for the implementation of tasks, using a three--dimensional approach, illustrated in figure 2. Specific configurable routing interconnections are available only to establish communication between routers and configurable logic. The implementation

of a task overlaps, in spatial terms, with one or more routers. However, since this occurs in different plans, corresponding to different layers, they do not interfere with each other. The interconnection between a task and the communication network may be established through one or more routers. Due to the fixed nature of the DyNoC structure and of its symmetry, it does not prevent task relocation. The original task will ensure the correct operation until both, original and replica, were perfectly synchronized, and the output data feeder could be switched from one to the other.

Network structure symmetry is mandatory to facilitate the initial placement and posterior relocation of tasks. Since the interface architecture ensures symmetry and homogeneity and its placement is known in advance, tasks may be pre--synthesized and be ready to be placed anywhere in the configurable logic space.

Of course, it is not feasible to place a number of routers equal to the number of logic blocks. However, if the number of routers were too small, it would lead to constraints in the placement of tasks. On the contrary, too many routers would lead to extreme flexibility, but it would also mean a considerable waste of resources. Experimental work based on simulation of task placement and resource occupation has to be done to find the optimal number of routers to be implemented in the FPGA.

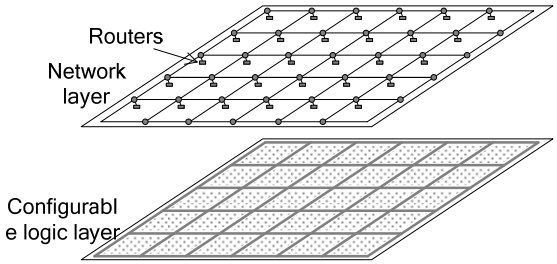

Figure 2: Proposal for the implementation of the communications network in 3-D

Since the interconnection between the FPGA and the host processor is done through a fixed bus, the problem of signal rerouting from tasks to pins, when tasks are relocated, is solved. Pin assignment is fixed between FPGA and host processor and completely independent from the implemented tasks.

3. Conclusions

While possibly accommodating different types of blocks, logic block distribution should guarantee that new allocations or the relocation of already implemented tasks can be done easily. To achieve it, the main features of a new architecture should be symmetry and homogeneity of the logic blocks distribution and of the interconnect resources structure. Within this framework, it will be possible to relocate horizontally or vertically tasks throughout the configuration space or even to rotate them in steps of 90º.

However, to easily relocate a task the symmetry of the resources is not enough. It is also indispensable to take into account the addressing structure of the configurable memory cells that will support FPGA logic configuration. A direct addressing scheme, bit-by-bit, despite its inherent flexibility, would consume too much resources just to address each configuration cell, and lead to lengthy configuration bitstreams and thus other forms of addressing need to be explored.

A mechanism for the relocation of current tasks while preserving the logic state, and without disturbing their operation, and for the pre-emption of tasks enabling the correct holding and restoring of current task state is also mandatory. This has to comprise mechanisms to preserve and replicate the current state condition of the relocated task, avoiding data incoherencies due to transfer failures or loss of state updates that may have taken place in the meanwhile. Just by enabling a fast and reliable relocation of tasks it will be possible to improve the performance and efficiency of reconfigurable computing systems.

In sum, the final goal is to increase applications' performance by taking full advantage of reconfigurable features. To get there, a series of problems have first to be address and solved before to be able to define an adequate architecture for this purpose.

References

[1] Manuel G. Gericota, Gustavo R. Alves, Miguel L. Silva, José M. Ferreira, "Run-Time Management of Logic Resources on Reconfigurable Systems", *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference*, 2003, pp. 974-979.

[2] S. Hauck, "The Future of Reconfigurable Systems", *5th Canadian Conf. on Field Prog. Devices*, 1998.

[3] Manuel G. Gericota, Gustavo R. Alves, Luís F. Lemos, José M. Ferreira, "A New Approach to Assess Defragmentation Strategies in Dynamically Reconfigurable FPGAs", *Revised selected papers of the International Workshop on Applied Reconfigurable Computing*, 2006. pp. 117-129.

[4] XC6200 Field Programmable Gate Arrays, *Data Sheet*, Version 1.10, April 1997.

[5] S. Shukla, N. Bergmann, J. Becker, "QUKU: A Coarse Grained Paradigm for FPGA", *Proc. Dagstuhl Seminar*, 2006.