1.8 Dynamic Reconfiguration of FPGAs

Patrick Lysaght and John Dunlop
Department of Electronic and Electrical Engineering
University of Strathclyde, Glasgow G1 1XW

Abstract

This paper considers the dynamic reconfiguration of those cellular Field Programmable Gate Arrays (FPGAs) that employ static memory to store their device configuration data. A FPGA is classified as dynamically reconfigurable if it can be partially reconfigured while active. The circuits on the device that are not included in the selective reconfiguration must continue to operate without interruption.

Dynamically reconfigurable FPGAs form a new class of logic which suggests new methods of digital system synthesis and realisation with the potential for significant advantages relative to current systems. This paper investigates the importance of dynamic reconfiguration and introduces self-controlling, dynamically reconfigurable systems and the concept of Logic Caching.

INTRODUCTION

One of the most significant contributions that reconfigurable FPGAs have made to PLD technology is to stimulate new areas of application. While PLAs, PALS, programmable sequencers and those FPGAs which are essentially derived from these PLD architectures are used for traditional logic replacement, completely new markets have been created by the arrival of reconfigurable cellular FPGAs.

In particular, whereas traditional PLDs were only used as logic replacement parts, reconfigurable FPGAs are also regarded as an exciting new programmable medium on to which to compile algorithms. For certain programmable applications, they offer an alternative to the von Neumann architecture: they are sufficiently complex and cost-effective enough to have become the focus for many experiments ranging over logic design, timeshared hardware, compiler design, hardware verification, automated VLSI synthesis, systolic processing, and general algorithm acceleration, (Kean 1989, Taylor et al 1991, Page and Luk 1991 and Gokhale 1991).

This paper presents definitions for the dynamic and partial reconfiguration of FPGAs and establishes the consistency of these definitions with the terminology used to describe reconfiguration techniques elsewhere in digital electronics. It traces the evolution of the deployment of FPGAs in programmable systems. The importance of partial reconfigurability for FPGAs deployed as programmable co-processors is developed. The paper proposes extending the use of FPGAs as a programmable medium through the exploitation of dynamically reconfigurable FPGAs. It introduces a new method of digital system synthesis called Logic Caching that exploits dynamically reconfigurable logic and gives an example of its use. A short review of previous applications of dynamically reconfigurable systems, including systems not based on FPGAs, is undertaken. Finally, consideration is given to the limitations of current FPGA technology when implementing dynamically reconfigurable systems and suggestions for improvements and future work are presented.

NOMENCLATURE

FPGAs may be classified according to their configurability as shown in Figure 1. All devices are by definition programmable, e.g. Actel devices. A smaller subset may be reconfigured by terminating operation and re-loading a complete device configuration, e.g. Xilinx devices. Note that the Actel device is not reconfigurable. In general, any subset in Figure 1 inherits the properties of its parent set, so that, for example, any reconfigurable device is programmable but the reverse does not hold.

Figure 1 Classification of FPGAs according to configurability

Dynamic Reconfiguration

FPGAs are classified as dynamically reconfigurable if their embedded configuration storage circuitry can be updated selectively, i.e. nominated configuration storage circuits (and the corresponding logic functions and interconnections which they control) can be changed, without disturbing the operation of the remaining logic. These devices can thus
be selectively reconfigured while still active. The resultant logic is referred to as *Dynamically Reconfigurable Logic (DRL).*

At present, only the FPGAs produced by Algotrix and Atmel are dynamically reconfigurable. These FPGAs share common characteristics: they are cellular arrays of relatively fine-grained cells, whose configuration is controlled by writing data to static memory locations. (Further devices, manufactured under licence from Pilkington Microelectronics Ltd., are expected in the near future.) For a particular FPGA to be dynamically reconfigurable, the definition implies that it must be capable of partial reconfiguration while active. At the system level a module containing multiple FPGAs may be classified as dynamically reconfigurable if the component FPGAs are individually reconfigurable.

**Partial Reconfiguration**

A device is defined as partially reconfigurable if it is possible to selectively reconfigure it, while the rest of the device remains inactive *but retains its configuration information.* As yet, there do not appear to be any devices on the market that are partially reconfigurable but not also dynamically reconfigurable.

**Consistent Terminology**

Reconfiguration techniques are widely used in digital electronics for fault and defect tolerance (Johnson 1989). They are also applied in parallel processing, particularly in applications employing processor arrays such as systolic or wave front processors (Chen 1990). Kung (1988) introduces the terms "run-time" and "compile-time" to qualify the reconfiguration of VLSI processor arrays. According to the definitions presented above, reconfigurable FPGAs are compile-time reconfigurable and dynamically reconfigurable FPGAs are run-time reconfigurable. It follows from this that a FPGA that is reconfigurable at run-time is also reconfigurable at compile-time. In this paper, Kung's terminology is used as an adjunct to the two previous definitions.

In Chen (1990), the term "static reconfiguration" is preferred to compile-time reconfiguration; Apel (1990) uses the term "on-line reconfiguration" in place of dynamic reconfiguration; and Johnson (1989) refers to dynamic reconfiguration as "real-time reconfiguration".

**PROGRAMMABLE SYSTEMS**

The evolution in the deployment of FPGAs in programmable systems is depicted in Figure 2. Initially, FPGAs were targeted at traditional "glue logic" applications replacing other types of PLDs. They were used to perform such functions as address decoding, wait state generation or bus demultiplexing as represented in Figure 2 (a). Most reconfigurable, cellular FPGAs are designed to be used as memory-mapped peripherals in conventional programmable systems. The FPGA is then primarily a reconfigurable interface device for implementing custom hardware interfaces as shown in Figure 2 (b).

As the size and speed of FPGAs continued to increase they quickly outgrew this role. It was recognised that the reconfigurable FPGA (or groups of FPGAs) could be used as a programmable co-processor to accelerate those tasks that were better suited to its architecture than that of the von Neumann processor. The subtle change in the role of the device is reflected in Figure 2 (c) where the re-positioning of the FPGA with respect to the processor emphasises the change in its deployment.

**The Importance of Partial Reconfiguration**

FPGAs that are not dynamically reconfigurable must be off-line before they can be either partially or completely reconfigured. The need to suspend device operation, for even small changes of function, has a critical impact on the use of these devices if they are to be used as programmable algorithm accelerators in applications where the nature of the tasks that they will execute is changing. Compile-time reconfiguration, for currently available reconfigurable FPGAs, is restricted to reconfiguration of the entire device. For example, the flowchart for the reconfiguration cycle of the Xilinx LCA parts (Xilinx 1991) shows that an LCA is always reset before reconfiguration. The previous configuration data is thus lost and partial reconfiguration, to implement a partial design change, is not possible.

![Figure 2 Developments in the deployment of FPGAs: (a) Conventional "glue logic", (b) Memory-mapped interface logic, (c) Memory-mapped, programmable co-processor, and (d) Stand-alone programmable system](Image)

If a reconfigurable FPGA is to perform a variety of different tasks as part of its overall function, it is necessary for the array to be configured by an external controller, such as a microprocessor. Another alternative is to employ multiple reconfigurable FPGAs with one or more devices assuming responsibility for loading the others.

The overhead in suspending operation to implement functional changes makes it unattractive to execute certain algorithms, or combinations of algorithms, on the FPGA. In some cases, the reconfiguration latency may offset any acceleration achieved via the
array. It may also influence the nature of the task organisation when multiple tasks are being performed concurrently; certain tasks are likely to complete before others, thus making it more difficult to optimise the ensemble for the maximum speed advantage. If the reconfigurable FPGA only supports complete reconfiguration, the disadvantages multiply.

The absence of partial reconfigurability, even at compile time, is an important disadvantage of reconfigurable FPGAs that can be expected to become more problematic as device densities increase. The overhead resulting from the need to store unique bit maps for every computational permutation is significant; consider, for example, the Xilinx 4008 device (an approximately mid-range entry in the Xilinx 4000 range, in terms of gate complexity) which requires 140,597 bits of programming data for each configuration. If this seems manageable now, consider a future device with sixteen times this capacity and multiply the new figure by the number of unique device reconfigurations that a particular application may require: the time and storage penalties accumulate rapidly. It may be possible, in some cases, to exploit data dependencies within the configuration bit maps by using special procedures, performed externally, to reduce the data storage requirements while, at the same time, maintaining transparent access for the FPGA. Partial, compile-time reconfigurability, however, represents a much better solution.

Dynamically Reconfigurable Logic

The unique capability of Dynamically Reconfigurable Logic, i.e. selective reconfiguration of active devices, suggests that a new system model is possible. If the controlling von Neumann host of Figure 2(c) is removed, the dynamically reconfigurable FPGA becomes a stand-alone host, as shown in Figure 2(d). The FPGA, used in this way, yields an example of a self-controlling, dynamically reconfigurable system. Those functions, that were previously performed by the von Neumann computer and that are still required, are absorbed into the FPGA itself.

The presence of read/write memory is crucial to the model. Note that the FPGA can be reconfigured to emulate the functions of a microprocessor. It is, therefore, as generic a model of computation as the von Neumann system. (It cannot, of course, compete with a dedicated implementation for the speed of its emulation, but neither can the microprocessor match the speed of the array for many algorithms.) The RAM block is critical to this capability since, without it, the model is simply that of the FPGA in "master mode".

The system model can be adapted to include a von Neumann co-processor, where special functions that are difficult to synthesise efficiently on FPGAs, such as floating point arithmetic, are accelerated by the co-processor. The von Neumann co-processor would essentially be an ALU (Arithmetic and Logical Unit); those of the Central Processing Unit's other functions, which were deemed necessary, would still be absorbed by the FPGA.

Models, such as these, of a single FPGA at the centre of a fully programmable system are not possible with single FPGAs that are not dynamically reconfigurable. If the task controlling reconfiguration is resident on the array, dynamic reconfiguration is implicit, since the controlling device must initiate and control reconfiguration and simultaneously be the object of that reconfiguration.

For applications where there is no need for a microprocessor, self-reconfiguring systems have an immediate cost advantage. Moreover, their operation does not rely on continuous execution of the von Neumann interpretive (fetch, decode, execute) cycle since they are reconfigured much less frequently, which represents a potentially significant speed advantage. Note also that the regular building-block architecture of microprocessor systems, which has become so popular with digital designers, is preserved in these new FPGA systems.

Dynamically Reconfigurable Logic combines properties normally associated only with software or hardware. It offers the flexibility of programmable solutions and fast operating speeds approaching those of hardware systems. This is represented graphically in Figure 3. Although it cannot compete with full or semi-custom techniques for either the highest speed or highest density applications, Dynamically Reconfigurable Logic is already available with usable gate densities of approximately 10,000 gates and maximum toggle frequencies of 150 MHz.

The new logic is faster than software (when executed on a microprocessor implemented in comparable technology) for many applications. This advantage results from the high clock frequencies, the inherent parallelism of hardwired solutions and the concurrency offered by executing multiple tasks on a single array simultaneously. For classes of programmable systems, it offers an alternative to the ubiquitous microprocessor and one that is ideally suited to VLSI because of its highly regular architecture.

Figure 3 The characteristics of Dynamically Reconfigurable Logic

LOGIC CACHING

FPGAs have been described as "programmable active memories" (Bertin 1989). If viewed from this perspective, the operation of a self-controlling dynamically reconfigurable system can be seen to be analogous to that of a memory cache in a conventional computer system. The "programmable active memory" is characterised by low capacity and high cost, while the external RAM has the opposite properties, i.e. it is relatively cheap and dense. The overall system, therefore, is an example of a memory hierarchy. When operating like a cache, only those circuits that are currently active are held in the "programmable active memory". The remainder reside in external RAM and are loaded as required. In this application, spatial and temporal locality are interpreted
slightly differently from the meaning ascribed to them in conventional von Neumann environments. Nonetheless, overall performance remains dependent on a high cache "hit-to-miss ratio" which, in turn, requires that the principle of locality of reference must apply, (Hennessy and Patterson 1990 and Lysaght 1991).

The operation of the logic cache optimises the ratio of active logic with respect to silicon area. This is very important for SRAM based FPGAs because of their logic density limitations. Despite the highly regular architectures which make them ideally suited to VLSI fabrication techniques, and the fact that densities and speed can be expected to continue to improve as process geometries decrease, the enormous overhead in additional circuitry imposed by the requirement for programmability remains a dominant problem.

![Diagram of Logic Caching](image)

**Figure 4 Logic Caching**

SRAM programming elements are typically twenty times larger than, for example, antifuse programming elements and a large FPGA will easily use more than 100,000 such elements. The size of a typical FPGA is dominated by the programmability of the wiring resources rather than that of the logic resources and the pitch (inter-track spacing) of the wiring channels is dictated by the square root of the area of the programming element (Rose et al 1989).

Note that Logic Caching in itself does not imply that the underlying system must be self-controlling. It is equally valid to deploy Logic Caching in systems where reconfiguration is controlled by the use of a von Neumann processor.

**System Example**

Consider a hypothetical data test set, employing dynamically reconfigurable logic, to monitor in real-time a 64 kbps serial communications link. The instrument employs a multi-phase algorithm which begins with a parallel search of the incoming data for one of five possible framing structures. The operation of the instrument, implemented using Logic Caching, is depicted in Figure 5. The diagram combines the floorplan perspective with a complementary Gantt chart representation.

At time equal to zero, the array is programmed to commence operation. This compile-time configuration will often, but not always, be the longest configuration. Each circuit resident on the array is represented by a task number. After initialisation, six tasks, denoted T1 to T6, are present on the array. Only one of the five frame sequence detectors (represented by T1 to T5 in Figure 11) can achieve frame synchronisation with any given data stream. Furthermore, as soon as one circuit detects frame lock (T2 in this case), the other four circuits (T1, T3, T4, and T5) are immediately redundant. An additional circuit T6 is activated to perform a partial, run-time reconfiguration of the array for phase two of the algorithm, while T2 continues to maintain frame lock.

Since only tasks T2 and T6 are not reconfigured, the lines on the Gantt chart representing these tasks continue uninterrupted through the partial, reconfiguration interval. The essential difference between run-time reconfiguration and compile-time reconfiguration is represented on the chart by tasks surviving the interval of reconfiguration as represented by the continuous task lines during the reconfiguration intervals. The duration of this reconfiguration interval is proportional to the amount of new circuitry, represented by T7 and T8 in this example, being loaded on to the array. (Note that extra, functionally redundant data may need to be loaded on to the array to make redundant cells inert, by re-initialising them to the default state. This is done to remove unwanted circuit paths which have the potential to oscillate, cause crosstalk or consume unnecessary power.)

![Combined floorplan and Gantt chart](image)

**Figure 5 Combined floorplan and Gantt chart perspectives**
Tasks T7 and T8 extract a particular channel from the incoming data stream and check the channel contents to see whether it contains live traffic or control data. If the most significant bit of the channel byte is set, the channel contains control data and additional circuitry (T9, T10, and T11) replace T8 (the data or control detection circuitry) to decode the control information. The new tasks take whatever appropriate action is required, such as activation of local loopback or the comparison of incoming control data to a locally generated reference such as a CCITT 16 pseudo random binary random sequence. In the event of loss of frame synchronisation, the data test set suspends current activity and operation reverts to phase one, via a dynamic reconfiguration controlled by T6.

If the system is self-controlling, then some permutation of tasks T2 and T6 must control the reconfiguration process. After the first, partial reconfiguration tasks T2, T6, T7, and T8 are resident on the array. Then a partial reconfiguration takes place during which tasks T2, T10, and T11 are substituted for task T8. At any time, the maximum number of tasks which may be present simultaneously on the array is determined by a potentially complex relationship between array size, the combined task sizes and any special requirements, such as access to specific input/output pins which may be necessary for certain tasks to run.

**Application Characteristics**

The example quoted above illustrates many of the properties which are important for algorithms to be suitable for implementation in dynamically reconfigurable logic. The algorithm is multi-phase and each phase has several component tasks. Many of the tasks, required within different phases, are mutually exclusive to each other with respect to time.

Consider the frame synchronisation phase. For maximum speed of operation, it is desirable that all the frame synchronisation detectors execute in parallel until one circuit reports frame lock. At this point, all the other frame synchronisation circuits are redundant and phase two can commence. Search algorithms of this kind are ideally suited to hardware multitasking systems. For any data stream, \( n-1 \) of the \( n \) frame synchronisation circuits and the tasks that are needed in subsequent phases are mutually exclusive to each other with respect to time.

The system is a real-time application and requires several tasks to be executed simultaneously which suits the speed, concurrency and parallelism of dynamically reconfigurable systems. Note also that in this case, the reconfiguration latency associated with the hardware multitasking may not present any difficulties. For example, once synchronisation is achieved, the subsequent operation of the instrument is determined by the user entering the requirements for the next stage of the test operation. The delay in the user entering this information will typically be several orders of magnitude greater than any delays associated with device reconfiguration. In general, the effect of reconfiguration latency will matter and must be considered in detail. The application does not require any numerically intensive computations, such as multiplication or square root calculations; this is an advantage since FPGAs are a less than optimal choice for numerically intensive applications due to inherent delay characteristics of multi-level logic.

Applications with logic blocks which appear to be well matched to synthesis on cellular FPGAs include string matching, error coding and decoding, bit-level signal processing and bit-level systolic algorithms. In general, it is difficult to find examples of such applications that simultaneously require programmability and exhibit the multiphase, multi-tasking characteristics with mutual-exclusion between tasks over time. The technique appears to be most suitable to complex digital systems where this kind of redundancy is more likely to occur. Consider, for example, that by the end of the decade dynamically reconfigurable FPGAs with perhaps as many as 200,000 gates will be available. The complexity of the digital systems that will be accommodated on such devices makes it more likely that the application characteristics identified here will be evident in these designs. In general, as digital designs increase in size and complexity, it becomes less likely that all of the components of the design will be operational simultaneously. Note that to implement complex systems using conventional techniques (i.e. without exploiting dynamic reconfiguration), more than one FPGA may be required. This commonly results in difficulties such as partitioning the design across device packages with possible reductions in system speeds. Logic caching, if applicable, could reduce the number of packages saving space, power and ultimately money.

The search for suitable applications resembles the task of converting programs to parallel processing environments. Hennessy and Patterson (1990) point out that this is not a trivial activity because of the need to address the complex interaction of three elements: these are the application, the algorithm chosen for implementing it and the architecture of the target system. It is a comparatively easy task to produce a new parallel processing system, but it is considerably more difficult to convert existing applications to the new systems such that they exploit the parallelism offered. One further complication in the case of FPGAs is that the target architecture is no longer a von Neumann architecture.

**PRECEDENTS**

The programming elements in a FPGA may be abstracted to a memory plane which lies above the logic and routing plane but controls the latter's operation. The overhead imposed by this layer is large given that, without dynamic reconfiguration, its function is entirely static. Manufacturers of reconfigurable FPGAs have begun to recognise the importance of recouping more 'functionality' from the logic in the memory plane. Xilinx and AT&T, in particular, have introduced new modes of operation for their logic blocks in which the configuration memory is directly accessible and can be used for data storage (Hsieh 1990, AT&T 1992). This overcomes the problem, described in Kean (1989), of implementing large storage registers with logic blocks that, at best, only provide two or four storage elements.

When contrasted with this approach, the operation of dynamically reconfigurable logic can be seen as an alternative, and more general, method to achieving more "functionality" from the memory plane in SRAM based FPGAs. The two techniques should not present any special difficulties if used simultaneously in the same FPGA. A more radical coupling between the logic and memory planes of a dynamically reconfigurable FPGA would allow logic on the array to directly access the configuration data of other cells. The normal loading mechanism for configuration data would be effectively by-passed. Further research would be required to determine the price to be paid for such flexibility and the overall performance of the resultant system.
To date, there are no reported examples of self-controlling, dynamically reconfigurable systems based on FPGAs. The concept of logic caching is, in some aspects, similar to the ideas behind the design of National Semiconductor's MAPL PLD for state machine synthesis (Hawley 1991). When hardware multitasking is used to implement a single large state machine on a single FPGA, the similarity between the two systems is at its greatest.

Limited attempts at using dynamic reconfiguration with FPGAs have been reported by Kean (1992) and van Daalen et al (1993). There are a number of examples from other disciplines where the concept of dynamically reconfigurable systems has been reported: a dynamically reconfigurable architecture for a "supersystem" for ballistic missile defence was proposed by Vick (1980) and a system for "on-line software extension and modification of stored program computers in digital switching exchanges" was described in Apel (1990) and Esteban (1990).

**LIMITATIONS OF CURRENT TECHNOLOGY**

With the exception in some aspects of the Algotronix CAL device, current FPGAs have not been designed with dynamic reconfiguration in mind. Even those devices capable of dynamic reconfiguration are supported by the most primitive CAD tools when considered from the perspective of designing dynamically reconfigurable systems. This may be attributed to the relative immaturity of these systems and the need for further work to be done to establish their commercial viability. Therein lies a recursive problem, however, since the absence of these tools makes it harder to establish the viability of dynamically reconfigurable logic. Several hardware improvements would be immediately useful: these include

- Technical specifications for dynamic reconfiguration
- FPGAs optimised for speed of reconfiguration
- The smallest viable unit of reconfigurability
- Access to reconfiguration control circuitry from inside the array
- Orthogonal (i.e. regular in two dimensions) routing and logic resources
- Coarse-grained dynamically reconfigurable FPGAs
- Co-ordinate addressing of individual array cells
- Write to and read back from the storage elements of individual cells

Associated improvements in the software tools provided with the devices would include the following:

- Simulation models for dynamic reconfiguration
- Automatic design partitioning based on temporal specifications
- Support for generation of relocatable bit streams
- Deterministic APR optimised for rectangular areas
- A simulation package for modelling new FPGA architectures
- Debugging tools

**FUTURE WORK**

Work is currently underway to establish practical systems demonstrating the benefits of Logic Caching and to characterise the extent of their performance advantages relative to more conventional design approaches. Hardware and software tools to facilitate this research are being developed. Proposals for enhanced FPGA architectures and new design methods and algorithms are also being investigated.

**CONCLUSIONS**

The availability of dynamically reconfigurable FPGAs extends the use of FPGAs as programmable accelerators in conventional microprocessor systems to new systems where the only programmable component is a FPGA. A description of a self-controlling, dynamically reconfigurable system has been presented. Its operation was shown to be analogous to that of a memory cache and so the term Logic Caching was coined to describe its behaviour. A potential example of its use was given. Some characteristics of algorithms well suited to the method and precedents for the technique were discussed. Suggestions for improvements to current hardware and software were made and the direction of future research outlined.

**ACKNOWLEDGEMENTS**

The support of the Nuffield Foundation for the work reported in this paper is gratefully acknowledged.

**REFERENCES**


Xilinx Inc., *The Programmable Gate Array Data Book*, San Jose, Ca., USA, 1991.