Partial Reconfiguration Applied in an On-line Evolvable Pattern Recognition System

Jim Torresen, Geir Aarstad Senland and Kyrre Glette
Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
E-mail: jimtoer@ifi.uio.no
Web: http://www.ifi.uio.no/~jimtoer

Abstract—One of the main challenges with autonomous adaptable systems is the lack of hardware flexibility. However, reconfigurable logic is a promising technology for run-time adaptable systems – often called reconfigurable computing. The paper outlines how reconfiguration can be applied at run-time for an on-line evolvable system to improve flexibility in the hardware. The challenge of the latter is to include flexibility without re-synthesis and avoid having a too large logic gate overhead. An architecture based on system-on-chip and partial reconfiguration is presented in the paper. Results from implementation show that reconfiguration can be undertaken in a few milliseconds for one category detection module of our classification system.

I. INTRODUCTION

Autonomous adaptable systems are expected to be important in the future. Evolvable systems have the potential of becoming an important part of such systems. This could imply that both software and hardware are adaptable. However, commercial dynamic computer systems have so far mainly been based on context switching software – i.e. switching software processes on a processor. However, with the introduction of Field Programmable Gate Arrays (FPGAs) also hardware is able to be modified at run-time. Few embedded systems are designed today without containing one or more FPGAs. The technology has progressed from earlier being used only as glue logic to now also being much applied for fast data processing. However, substituting the configuration at run-time has so far not been much applied. Like software many years ago, the same code/configuration remains static in the device when a system is put in operation. Swapping processes has not yet reached the FPGA application designers. There are naturally some reasons for this including long reconfiguration time and reliability issues. The recent progress of the technology has however reduced these problems and dynamic FPGAs are now more applicable than earlier, as will be described in this paper.

When an FPGA-based system is able to change its configuration, we often talk about reconfigurable computing. The goal of the architecture developed in this paper is to have a set of configurations which the FPGA can switch between at run-time. That is, the system can switch configuration without stopping or interrupting the system for a long time. Since much (i.e. the signal routing) of the FPGA configuration bitstream coding is secret, it is difficult for an evolvable system to directly change the FPGA content unless re-synthesis is undertaken. Re-synthesis is not normally applicable because of the time consuming process. Thus, evolvable hardware architectures often end up being regular and with limited flexibility. The challenge is addressed in this paper by introducing an architecture with inherent reconfigurability.

In the next section, an overview of the status of alternatives for run-time reconfigurable systems is included. This is continued by introducing how this technology can be applied in evolvable systems in section III, followed by results in section IV. Finally, section V concludes the paper.

II. APPROACHES TO RECONFIGURABLE COMPUTING WITH FPGAS

There are several challenges of run-time reconfiguration in FPGA:

• Long time required for reconfiguration
• Avoiding the system from being inactive during reconfiguration (safe and robust reconfiguration)
• Interfacing between modules belonging to different configurations
• Predictability (reliability and testability) of system operation

Since the configuration bitstream is serially loaded into the device, the main problem with switching configurations is the long reconfiguration time. At the moment there seem to be three possible approaches; smaller devices, virtual FPGAs and partial reconfiguration.

1) Smaller Devices: Since the full reconfiguration time is less for smaller devices, reconfiguration time can be reduced by applying smaller devices. Moreover, by applying context switching, we may be able to implement a full system in a smaller device with the benefit of reduced cost and power consumption. The drawback would be that the system would have to be inactive during reconfiguration.

2) Virtual FPGAs: Virtual FPGA is based on designing a “virtual” FPGA inside an ordinary FPGA [1]. We have so far introduced an architecture for context switching based on a multi-context “virtual” FPGA [2], [3]. The architecture provides switching between 16 different configurations in a single clock cycle. Such a system would never achieve as high clock frequency as a leading edge processor. However, by applying massive parallel processing, the execution time...
can still be less than for software on a processor [4]. Even though a fast processing can be achieved, the context switching architecture requires much reconfigurable resources (in that way, this architecture is prioritising speed before cost and power consumption).

3) Partial Reconfiguration: As FPGA devices are getting bigger, the configuration bitstream becomes longer and programming time increases. Thus, run-time reconfigurable designs would benefit from having only a limited part of the FPGA being context switched by partial reconfiguration. This feature is available in some FPGAs where a selected number of neighbouring columns are programmed. This requires detailed considerations for having no interruption at context switching [5]. There has been undertaken some work on real-time partial reconfigurable systems like e.g. [6], [7], [8].

Another challenge is to limit the inter partition data transfer. That is, efficient communication between context switched tasks. While the first FPGAs offering partial reconfiguration required complete columns of the device being programmed, the more recent ones – including Virtex-4/5, require only a part of each column being programmed – see figure 1. This makes interfacing between tasks and having uninterrupted operation easier since some rows can be used for permanent configurations. The smallest Virtex-4 device (LX30) consists of 4 rows while the largest (LX330) consists of 12 rows. Further, there has been introduced tools like PlanAhead that makes partial reconfiguration easier. It is possible to reconfigure the Virtex devices internally for instance by using an on-chip (soft or hard) processor core accessing the Internal Configuration Access Port (ICAP). This will be applied in the architecture presented in the next section where virtual FPGA and partial reconfiguration are combined. This will be targeted at an earlier proposed classification architecture.

III. A Flexible Classifier Architecture

To be able to provide run-time adaptation, some scheme for automatic design is necessary. So far evolution has been a much explored method. However, to provide not too slow evolution, many systems have been based on virtual FPGA implementation rather than FPGA reconfiguration. One of the problems related to virtual FPGA is the large gate overhead especially for routing signals. A common way to implement routing is by using multiplexers. However, these become large as the signal resolution increases and the size of the architecture increases. To reduce this problem to make systems more scalable, we would in this paper introduce an alternative architecture with less need for multiplexers.

A. The Original Classification Module

We have earlier developed an architecture that has shown to give a high classification performances both for an image application [9] and a signal processing application [10]. The system consists of three main parts – a classification module, an evaluation module, and a processor. The complete system is implemented in a single FPGA. The processor is running the evolution algorithm and configures the other modules. The evaluation module is used for fitness computation and is based on evaluating only a small part of the classifier at a time since incremental evolution is applied. The classification module, however, would have to be complete unless time multiplexing is applied. Therefore this module with its typically large structure is difficult to make online adaptable without a large logic gate overhead (mainly for routing). Thus, in this paper the focus is on the classification module rather than the evaluation module.

The classification module operates stand-alone except for its reconfiguration which is carried out by the processor. The classification module consists of one category detection module (CDM) for each category to be classified – see figure 2. The input data to be classified is presented to each CDM concurrently on a common input bus. The CDM with the highest output value will be detected by a maximum detector, and the identifying number of this category will be output from the system.

Each CDM consists of M “rules” or functional unit (FU) rows – see figure 3. Each FU row consists of N FUs. The inputs to the circuit are passed on to the inputs of each FU. The 1-bit outputs from the FUs in a row are fed into an N-input AND gate. This means that all outputs from the FUs must be 1 in order for a rule to be activated. The outputs from the AND gates are connected to an input counter which counts the number of activated FU rows. The FUs are the reconfigurable elements of the architecture. Each FU behaviour is controlled by connected configuration lines (not shown in figure 3). Each FU has all input bits to the system available at its inputs but only one data element (e.g. one byte) of these bits is
chosen. One data element is thus selected from the input bits, depending on the configuration lines.

B. A Flexible Classification Module

Evolution is undertaken for one or a few FUs at a time, thus, the flexibility would only be needed in the classification module to be applied after evolution. Including flexibility could either be undertaken by including flexibility in the design (virtual FPGA) or by changing the design itself either by partial re-synthesis or having a number of pre-synthesized configurations. Reconfiguration of both the latter approaches could be undertaken with partial reconfiguration. For most applications, re-synthesis would not be applicable due to the long time needed. Further, resource planning could be difficult as the flexibility increases. Thus, we would like to look into how flexibility in an architecture can be increased with pre-synthesized configurations.

The classification architecture introduced above is based on using predefined values for \( N \) and \( M \). To be able to change them, re-synthesis is required. Keeping \( N \) and \( M \) predefined lead to the most efficient hardware architecture since flexibility could often not be effectively implemented. However, with a system-on-chip implementation, there will typically be a limit on the total number of FUs that can be implemented in the device. On the other hand, since the data set changes over time, it would often be impossible to predict the optimal selection of \( N \) and \( M \) at design time. The following flexibilities could be explored [11]:

- Variance in the number of FUs in each row,
- The number of FUs in each row \( N \) versus the number of FU rows \( M \).
- The total number of units assigned to each CDM could be different for each category.

Below, we present how an architecture with the ability to select the combination of \( N \) and \( M \) that maximizes the performance can be implemented.

The most intuitive approach to applying pre-synthesised configurations would probably be to store a number of configurations with different values for \( N \) and \( M \). However, all have the same total number \( (N \times M) \) of FUs based on the amount of available logic gate resources in the given device. This could be undertaken by the architecture in figure 4. In this architecture, the CDMs are programmable by partial reconfiguration through the ICAP interface. After the processor (CPU) has evolved a new classifier, CDMs are configured with the configuration corresponding to the best found combination of \( N \) and \( M \). The Input and Output interfaces (as well as the Max Detector) are static but can be updated with data like the chosen value of \( N \) and \( M \).

A more active variant would be to change the LUT (look-up table) content available in the partial configuration bitstream. However, this requires low level study of the design and some reverse engineering could be required to locate the correspondence of LUTs and units in the design. Thus, this approach has not been further explored in the experiments below.

A Virtex-II Pro FPGA of type 2VP30 was applied for the experiments. It contains a PowerPC hard core processor. The partial bitstreams are stored in CompactFlash\(^1\) and are copied to SDRAM memory at start-up.

\(^1\)CompactFlash was for us the most convenient way of storing the bitstreams since it was available on our board.
C. Implementing Partial Reconfiguration

To design the system for partial reconfiguration, the PlanAhead tool from Xilinx was applied. This tool helps in defining and placing the partial reconfigurable regions (PRR) which communicate with the static part through bus macros. The different partial bitstreams are called partial reconfigurable modules (PRM), see figure 5. PlanAhead further helps in placing bus macros and running place-and-route on the various modules. In the end, one bit file is generated – containing the static part and one of the PRMs, which is downloaded to the FPGA. In addition, partial bitstreams for each of the PRMs are generated. The PowerPC interfaces and software are designed with the help of the Embedded Development Toolkit (EDK) provided by Xilinx. The software is also incorporated in the bit file which is downloaded at start-up.

IV. RESULTS

A number of experiments have been undertaken to determine the reconfiguration speed. Initially, it was configured directly from CompactFlash to ICAP but this was very slow. Speed was improved by configuring from copies of the partial bitstreams stored in SDRAM. Other effective optimisations included turning the PowerPC cache on (72% time reduction) and optimising the functions for ICAP access (33% time reduction).

The experiments were limited to reconfiguring one CDM as given by the PRR in figure 6. This PRR (154 KB bitstream length) allows for CDMs of size e.g. 8x8, 12x6, 16x4, 20x6 and 28x4. With the PowerPC running on 100 MHz, we obtained a configuration time of 18.4 ms for one CDM. If the processor had run at the maximum speed of 300 MHz, we expect a reconfiguration time of 6.1 ms. Further, we expect the configuration time to increase linearly. Thus, 10 CDMs should require 61 ms which is still a short time for most pattern recognition applications.

V. CONCLUSION

In this paper, it has been outlined how reconfigurable computing can be applied in an on-line evolvable system to improve flexibility in the hardware. The challenge of the latter is to include flexibility without re-synthesis and avoid having a too large logic gate overhead. An architecture based on system-on-chip and partial reconfiguration is presented in the paper which allows for increased flexibility. Experiments show that reconfiguration can be performed in a few milliseconds.

REFERENCES