A Dynamically Reconfigurable Computing Model for Video Processing Applications

G. Alonzo Vera, Daniel Llamocca, Marios S. Pattichis
The University of New Mexico, Department of Electrical and Computer Engineering,
Albuquerque, New Mexico, USA
alonzo@ieee.org, dllamocca@ieee.org,
pattichis@ece.unm.edu

James Lyke
Space Electronics Branch, Space Vehicles Directorate, Air Force Research Laboratory,
Albuquerque, New Mexico USA
james.lyke@kirtland.af.mil

Abstract—**We introduce an idealized dynamically reconfigurable computing model that is suitable for applications in video processing applications. Dynamically reconfigurable computing is characterized by a dynamic data path which has been made possible with the partial reconfiguration feature available in modern FPGA devices. Dynamically reconfigurable computing design leads to a multi-objective optimization model with constraints on power, performance and resources. We provide a review of recent reconfigurable computing applications reported by different groups and propose a new model for dynamically reconfigurable video processing applications. We provide model measurements for reconfiguration time overhead, static and reconfiguration power consumption.**

Keywords-component; Dynamic Reconfiguration; DSP; DIP; FPGAs

I. Introduction

Reconfigurable computing is defined by the ability of modifying a processing unit's data path in response to variations in the computation's constraints. When the idea of "dynamic hardware" was first enunciated by Estrin in [1], devices with the required flexibility were not available yet. It is with the introduction of reconfigurable devices that the first hardware platforms for reconfigurable computing were possible.

The first devices used for this purpose had the ability of holding different configuration images at the same time, and were able to switch between them in few clock cycles. This kind of devices was labeled *multi context* devices. As devices grew denser, holding multiple configuration contexts became impractical. Later devices are *single-context*, meaning that only one configuration image is held at a point in time as shown in Figure 1. Single-context devices are limited by the time it takes to load a new configuration every time a change – reconfiguration- is required. This time is referred to in the literature as *reconfiguration time overhead*. Reconfigurable architectures used multiple devices in order to partially hide the reconfiguration time overhead by performing computations in one device while the others were being reconfigured as needed. In this schema, reconfiguration is seldom used, being more of a change of application altogether rather than a change in the application. Modern devices introduced two new key features that have enabled a myriad of applications that use reconfiguration more intensively. First is run-time partial reconfiguration. Partial reconfiguration reduces the reconfiguration time overhead by allowing partial configurations – thus smaller bitstreams – to be uploaded. Run-time refers to the ability to perform partial reconfiguration while keeping the unchanged parts of the device running. The second feature is access to the configuration memory from within the device's fabric. This means that a device can reconfigure itself. These two features have moved the paradigm of reconfigurable computing one step forward, towards system-on-a-chip (SoC), self-reconfigurable computing systems.

Different deployment flows for programmable devices are possible given all these features (see Figure 2). The traditional flow consists of creating a single configuration file, programming the device and processing data until system shutdown. With partial run-time reconfiguration, a different flow is possible. In this case one can create a set of bitstreams for different sections of a device and different application's conditions. An initial configuration is loaded and processing takes place until a change in the computation conditions force a partial reconfiguration. Processing keeps going on with partial changes in the device configuration as needed. A final alternative is to create partial configurations, compile them in real-time, and use them to reconfigure the device as needed.

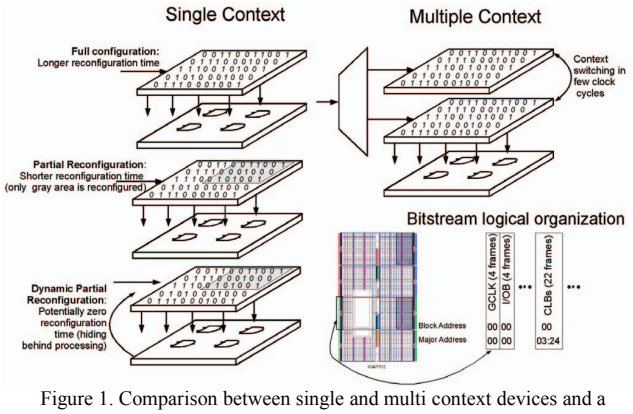

Figure 1. Comparison between single and multi context devices and a configuration file (bitstream) logical organization for a Xilinx's Virtex 4 device.

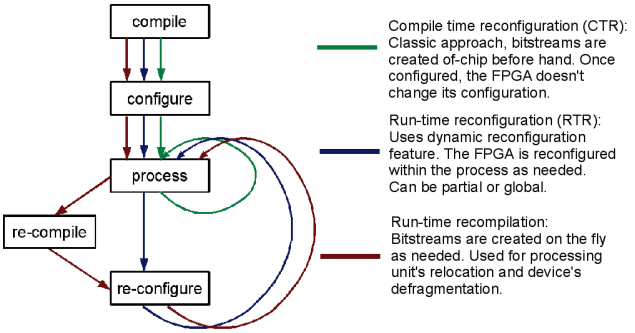

Figure 2. Different deployment flows possible given some of the features available in moder FPGAs.

This is a significantly more complicated flow since it requires the device to have the intelligence to select code and compilation parameters in real-time. Thus, this particular flow requires a deep knowledge of a device architecture and manufacturer tools. In this paper, this flow is not studied since not enough information is provided by FPGA manufacturers for a third party to execute it, unless a non-disclosure agreement is in place [2].

In the field of image and video processing, reconfigurable computing has been used for many purposes, from boosting performance by adapting the hardware to variations in the data's characteristics, to emulating infinite resources by multiplexing hardware in time. However, in image processing, and especially in video processing, we have very tight performance constraints. Increased performance is generally obtained as a trade-off between power consumption and logical resources usage. In this paper, we formulate run-time reconfigurable system design as a problem of meeting objectives in performance, power consumption and logical resources usage.

To motivate the model, we first provide a literature review of representative cases of reconfigurable computing applications reported in the field of image and video processing. The selection of these example cases is based on the different approaches to partial reconfiguration that the authors chose. The cases listed by no means represent an exhaustive collection of reported applications. We then provide a mathematical model that summarizes all the factors and parameters that play a significant role in the multi objective optimization problem of dynamically reconfigurable computing. We then present some measurements that provide actual values for some of the model's key parameters.

II. Background

Research in image and video processing applications implemented in reconfigurable computing platforms can be classified in three groups. The first and earliest group consisted of applications that used reconfiguration in a moderate manner, generally built upon reconfigurable fabric and a static host used as a reconfiguration controller and general processing unit. Platforms such as ARDOISE [3], GARP [4], Chimera [5] and OneChip [6-7] fall into this category. Reconfiguration time overhead in these architectures is either not a problem because it is small enough not to disturb the overall system performance (some of these architecture supported partial reconfiguration), or it is dealt with by parallel processing: reconfiguration is done in one device while useful processing is being done in other devices. Many other techniques to reduce reconfiguration time overhead have been reported in the literature, such as configuration compression [8], caching [9] and pre-fetching [10]. In these early platforms, power consumption was not a constraint. Reconfiguration in this case was mainly used as a way to emulate infinite resources by swapping in and out processing elements from the reconfigurable devices as needed.

An example of a reported application in this group is [3], where the authors describe an approach to implement real time image processing using the ARDOISE architecture. In this case the devices used are ATMEL 40K40 and the reconfiguration time is small enough to allow a device reconfiguration within the time it takes for a new frame to arrive (~40ms). This paper also presents a model used to explore the performance limits of this architecture, based on the complexity of an algorithm (number of gates) and the reconfiguration speed (thus, reconfiguration time overhead). The complex trade-off between performance, power consumption and logical resources is not addressed.

The second and third groups consist of applications that used intensively reconfiguration in its latest form: run-time partial reconfiguration. Partial reconfiguration is used in this case to swap functional or processing units in and out the programmable device without putting on hold the computation or processing being done in the rest of the device. The main function of reconfiguration is still an emulation of infinite resources by multiplexing hardware in time. However, performance management is also being done, either by modifying a functional unit to respond to changes in the data being processed, or by modifying a process parallelism by either incrementing or reducing the number of processing units as needed. Although power is still not a constraint, researchers already observed that it is possible to manage power consumption using the same means used to manage performance.

An example of this group of applications is [11], where the authors used a Xilinx's multi context device (XC6200) to implement a motion estimation algorithm. Dynamic reconfiguration was used with the goal of saving power. This is accomplished by modifying the search regions according to frame to frame statistics, being able in some cases to reduce computation and thus reduce power. Another power saving strategy was to re-utilize unused sections of the reconfigurable device to implement local memory, reducing power costly off-chip communications. Reconfiguration time overhead wasn't significant due to the multi context nature of the device used.

The third and latest group consists of applications based on run-time partial self-reconfiguration and usually contained within a single device. In this case, the availability of logical resources is the most important constraint, while power and performance are traded-off depending on an application's needs or power availability. It is this group that this paper focuses on. It is important to mention that the model presented

in this paper, although targeted towards this group, can be adapted to describe applications in the previous two groups by setting some of the parameters accordingly.

To this group belong applications such as the ones described in [12] and [13]. In [12], the authors describe a dynamically reconfigurable image processing system able to perform pixel-based operations using small processing units. These processing units can be swapped at run-time to perform different operations. Their small size reduces the reconfiguration time overhead. The overall architecture is described as a reconfigurable SoC. In [13], the authors describe a multiprocessor SoC that uses run-time partial reconfiguration to generate hardware threads seamlessly intermix with software threads for the execution of the JPEG2000 image compression algorithm. In this case, reconfiguration time overhead is hidden behind the execution of parallel software threads. This architecture trades-off the number of accelerators allocated (hardware processing units) and the number of software threads executed based on performance requirements and logical resources limitations.

III. Multiobjective optimizaton model

This work focuses on a run-time self reconfigurable system for image and video processing. As such, the following assumptions are made:

1.- For simplicity, homogeneous processing elements (PE) are assumed, represented in the model by the number of hardware frames required for their configuration. This measure of size is independent of the logical resources used by the PE due to a practical limitation in the PR design flow.

2.- Power available to the system is a function of time (e.g. a satellite depending on solar power or a battery application, see Figure 3).

3.- An application/algorithm is subdivided into tasks. The smallest atomic data unit is a (video) frame. A single task may be processed in a single PE. Alternatively, multiple PEs implementing the same task can be configured in order to speed up the process by means of parallelism (see Figure 3).

4.- The model is built upon data gathered using Xilinx's FPGA, as they are currently the only devices supporting run-time self reconfiguration (see next section).

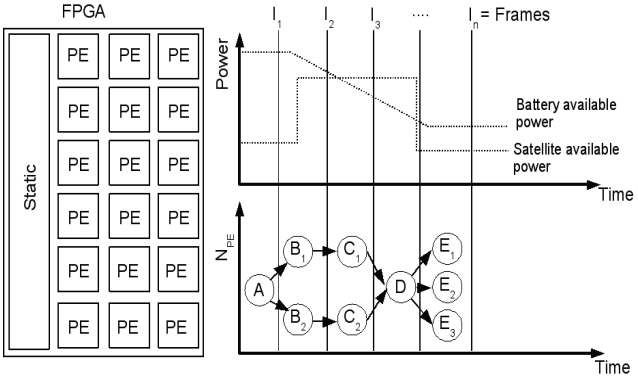

Figure 3. Key assumptions in the model's definition.

5.- The complete system must fit in a single device (logical resources constraint).

6.- It is assumed that all required bitstreams are available at running time. Real-time compilation and relocation is not supported.

7.- Run-time partial reconfiguration is done through the internal configuration access port (ICAP). Thus, there is hardware overhead related with the configuration controller. This overhead is not significant when compared with hardware required to control the overall system (see [1, 14] for typical resource usage).

8.- Performance (measured in frames per second (fps) in the case of video processing) is ideally assumed to be directly proportional to the number of PEs at a point in time.

9.- A maximum time delay constraint of 33 ms (~30 frames/second for real-time video) is used to account for data reading, reconfiguration and data writing.

10.- The model is described in terms of a multi-objective constrained optimization that searches to minimize the energy consumed to execute an application while maximizing the system's performance.

With these assumptions at hand the model can be simply defined as the minimization of energy consumption and maximization of performance, given constraints in both performance and total energy available:

$$\min(E, -S) \quad \text{subject to} \quad S \geq 30fps \quad \text{and} \quad E \leq E_T$$

where S is the system's performance that can be expressed in terms of frames per second (*fps*). E is the total amount of energy consumed for the execution of a task and E_T is the total energy available. This is an important parameter that could affect the number of PEs involved in executing a task. It is calculated by integrating the amount of power available over a time frame (see Figure 3).

To model performance we start by estimating the time *D(fr)* it takes to process a frame *fr* as:

$$D(fr) = S_{PE} * N_p(fr)/N_{PE}(fr)$$

where N_{PE} represents the number of processing elements (PEs) available to process a frame, S_{PE} represents the time it takes for a PE to process a single pixel within the frame, and N_p represents the number of pixels for the frame *fr*.

Note that *D(fr)* does not include the time it takes to move data in and out of the processing units nor does it include reconfiguration time overhead. Including reconfiguration, *D(fr)* becomes $D_{RC}(fr)$:

$$D_{RC}(fr) = D(fr) + \alpha(fr) * \beta(fr) * R_T * N_{RPE}$$

where $\alpha(fr)$ is a reconfiguration cost factor for possibly pre-configuring PEs for video frames ahead subject to power availability. This factor accounts for the use of configuration pre-fetching techniques to reduce the reconfiguration time overhead. $\beta(fr)$ is a factor to account for using parallel processing while partially reconfiguring sections of the device at run-time. It is effectively the percentage of reconfiguration

overhead that can be hidden behind parallel processing. Note that the factors α and β have natural constraints:

$$\alpha(fr) \geq 1, \quad \beta(fr) \leq 1$$

R_T represents the time needed to reconfigure a PE. This parameter depends on the device being used (characterized by different configuration memory ports with different speeds) and the size of the PE. The larger the PE, the more logical resources it uses, thus the larger the partial bitstream required to configure it. $N_{RPE}(fr)$ represents the number of PEs reconfigured per frame processed.

Thus, the performance S can be defined as $1/D_{fRC}$ or:

$$S = (S_{PE} * N_p(fr)/N_{PE} + \alpha(fr) * \beta(fr) * R_T * N_{RPE}(fr))^{-1}$$

The application's energy E can be modeled as:

$$E = \sum_{fr=0}^{N_f}(N_{PE}(fr) * P_{PE} + N_{RPE}(fr) * P_R)$$

where P_{PE} is the power consumed per PE and P_R is the power required to reconfigure a PE, which is device/technology dependant. The power consumed by each PE is generally divided in static power and active power (see [15] for a definition and measurement's setups or estimation). Static power consumption is linearly dependent on the amount of logical resources being used. Dynamic power consumption is related to frequency of operation, data path width and voltage of operation. Since the 3rd factor is normally not manageable (there are unconventional techniques reported to manage this voltage in order to save power [16]), dynamic power can only be finely controlled by changing the frequency of operation and the level of parallelism of a task. Note that energy is the integral of power over time. For simplicity we assume time discrete measured in frames.

In order to use this model some of the parameters ought to be estimated or measured.

IV. Physical constraints and measurements

In applications such as the ones we are trying to model, reconfiguration time overhead is usually the largest limiting factor in performance. Thus a significant amount of research has been focused on reducing this overhead by either manipulating the bitstream (compression) or by adequately scheduling loading times (prefetching). Other research has focused on accelerating loading times by either exploiting the configuration port's bandwidth or by implementing some sort of configuration caching. Ultimately, all methods related to how fast one can load a configuration bitstream into memory. For a variety of different FPGA devices, we present experimental results for measuring the reconfiguration time overhead, reconfiguration power, and static power. For comparison purposes, we normalize our results in terms of the percentage of the device that is being reconfigured.

Figure 4 shows estimated loading times for partial configurations of different sizes for the larger devices in Xilinx FPGA's representative families. The graph's numbers were calculated with the maximum theoretical bandwidth and some of the most important and faster configuration controllers reported to date. This graph provides bounds for the reconfiguration time overhead R_T, for processing elements that occupy different percentages of the device. Here, for real-time performance at 30 frames per second, we note the delay bound of 33 msecs for reconfiguration and processing.

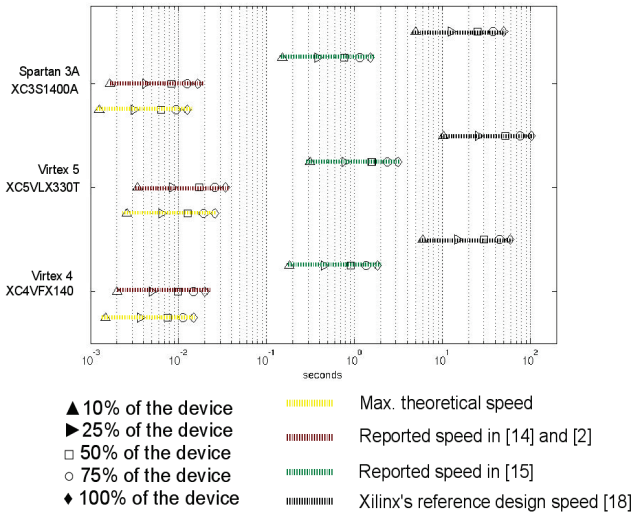

Figure 4. Reconfiguration times for different device percentages at different configuration bandwidths.

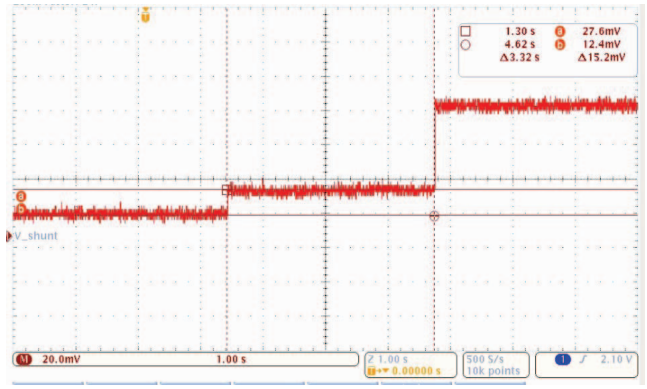

Figure 5. Power drawn during reconfiguration for a Xilinx's XC2VP30 device.

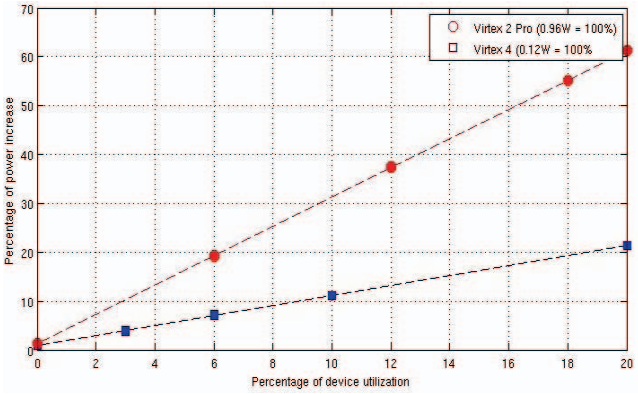

Figure 6. Linear relationship between the amount of logical resources being used the static power consumption for Virtex 2 and Virtex 4 devices.

We also measure FPGA reconfiguration power during reconfiguration P_R .Reconfiguration power is largely dependent on the family's device and technology. Figure 5 shows an indirect measurement of the current (15.2 mVolts/ 0.1 Ohms) being drawn by an Virtex 2-Pro (XC2P30) FPGA while configuration is taking place. A description of the measurement setup can be found in [15]. In this case, the amount of power can be calculated as $15.2^2/0.1$ watts.

A portion (static power) of the overall power consumption P_{PE} have been found to be linearly dependent with the percentage of the device being used. Figure 6 shows this linear relationship for Virtex 2 and Virtex 4 devices.

V. Conclusions

A dynamic computer simulator is being implemented using the model described. All parameters are being evaluated in Xilinx's Virtex 5 and Virtex 4 families. The model has applications in run-time reconfigurable intensive applications, where power and performance are constrained. An example of such applications is developed in [15] and [17].

The model does not account for data dependencies. This early model intended for use by the image and video processing community for deciding on the usefulness of dynamic run-time partial reconfiguration solutions.

Acknowledgment

The research presented in this poster has been funded by the Air Force Research Laboratory under grant number QA9453-060C-0211.

References

[1] G. Estrin, B. Bussell, R. Turn, and J. Bibb., "Parallel Processing in a Restructurable Computer System", IEEE Trans. Electronic Computers, vol. EC-12, pp 747-755, 1963.

[2] Colby J., "High-Speed Dynamic Partial Reconfiguration for Field programmable gate arrays", Master thesis, July 2009.

[3] Demigny, D., Kessal, L., Bourguiba, R. and Boudouani, N., "How to Use High Speed Reconfigurable FPGA for Real Time Image Processing?", International Workshop on Computer Architectures for Machine Perception, IEEE Computer Society, 2000, 0, 240

[4] Hauser, J.R., Wawrzynek, J., "GARP: a MIPS processor with a reconfigurable coprocessor",. Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[5] S. Hauck, T. Fry, M. Hosler, J. Kao, "The Chimaera Reconfigurable Functional Unit", IEEE Symposium on FPGAs for Custom Computing Machines , 1997.

[6] R. D. Wittig, P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic", IEEE Symposium on FPGAs for Custom Computing Machines, 1996.

[7] Wittig, R. D., "OneChip: An FPGA Processor With Reconfigurable Logic", University of Toronto, 1995.

[8] S. Hauck, Z. Li and E. Schwabe, "Configuration Compression for the Xilinx XC6200 FPGA", IEEE Symposium on FPGAs for Custom Computing Machines, 1998.

[9] Z.Li, K. Compton and S. Hauck, "Configuration Caching Management Techniques for Reconfigurable Computing". IEEE Symposium on FPGAs for Custom Computing Machines, pp. 87-96, 2000.

[10] S. Hauck, "Configuration Prefetch for Single Context Reconfigurable Coprocessors", ACM/SIGDA International Symposium on Field-Programmable Gate Array s, pp. 65-74, 1998.

[11] Park, S. and Burleson, W., "Reconfiguration for power saving in real-time motion estimation", Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 3037-3040 vol.5

[12] Daniel. Llamocca, Marios. Pattichis, and G. Alonzo. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", Proceedings of FPL'2009, Prague, Czech Republic, Sept. 2009.

[13] Tumeo, A., Borgio, S., Bosisio, D., Monchiero, M., Palermo, G., Ferrandi, F., and Sciuto, D., "A multiprocessor self-reconfigurable JPEG2000 encoder", 2009 IEEE International Symposium on Parallel and Distributed Processing.

[14] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A Multi-Platform Controller allowing for maximum dynamic partial reconfiguration throughput" in Proceedings of FPL'2008, Heidelberg, Germany, Sept. 2008, pp. 535-538.

[15] Guillermo A. Vera, "A Dynamic Arithmetic Architecture: Precision, Power, and Performance Considerations", PhD. Dissertation, University of New Mexico, Albuquerque, NM, USA, May 2008.

[16] Garcia A., Danger, J. and Burleson, W., "Reducing the Power Consumption in FPGAs with Keeping a High Performance Level", Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00), IEEE Computer Society, 2000.

[17] Ackermann, K., Hoffmann, B., Indrusiak, L. and Glesner, M., "Enabling self-reconfiguration on a video processing platform", International Symposium on Industrial Embedded Systems,, 2008, 19-26.

[18] Partial Reconfiguration Early Access software tools for ISE 9.1 SP2. Design examples targeting an ML403 Development board.