Snake: An Efficient Strategy for the Reuse of Circuitry and Partial Computation Results in High-Performance Reconfigurable Computing

Xabier Iturbe†, Khaled Benkrid‡, Ali Ebrahim*, Chuan Hong*, Tughrul Arslan* and Imanol Martinez†

*System Level Integration Group, The University of Edinburgh, Edinburgh EH9 3JL, Scotland, UK
{v.iturbe, k.benkrid, a.ebrahim, c.hong-2, t.arslan}@ed.ac.uk
†Embedded System-on-Chip Group, IKERLAN-IK4 Research Alliance, Mondragón 20500, Basque Country (Spain)
{xiturbe, imartinez}@ikerlan.es

Abstract—In this paper we present “Snake”, a novel technique for allocating and executing hardware tasks onto partially reconfigurable Xilinx FPGAs. Snake permits to alleviate the bottleneck introduced by the Internal Configuration Access Port (ICAP) in Xilinx FPGAs, by reusing both intermediate partial results and previously allocated pieces of circuitry. Moreover, Snake considers often neglected aspects in previous approaches when making allocation decisions, such as the technological constraints introduced by reconfigurable technology and inter-task communication issues. As a result of being a realistic solution its implementation using real FPGA hardware has been successful. We have checked its ability to reduce not only the overall execution time of a wide range of synthetic reconfigurable applications, but also time overheads in making allocation decisions in the first place.

Keywords—High-Performance, Partial Dynamic Reconfiguration, Reconfigurable Computing, Inter-task Communications

I. INTRODUCTION

Dynamic Partial Reconfiguration (DPR) permits to reconfigure a portion of an FPGA device while the remaining portion is still carrying out active computation and hence, FPGA's resources can be shared along time among different swappable circuits. As a result, a larger virtual hardware resource can be implemented using limited physical reconfigurable FPGA hardware. We denote such swappable circuits as hardware tasks to reflect their dynamic and hardware-based nature. DPR enables each task to specialize in a specific type of computation, as the resulting increase in resource demand can be dealt with hardware reuse. Moreover, note that several tasks can run at the same time in different regions of the device. These are the basis for Reconfigurable Computing (RC), which targets a continuous stream of input operands, computation and output results as the data processing hardware tasks are allocated on the FPGA, executed and finally replaced by other tasks.

Scheduling and allocating hardware tasks onto reconfigurable hardware have been widely studied subjects in the technical literature. However, most researchers simply model the FPGA reconfigurable surface as a 2-dimensional array and treat the problem of allocating the tasks as a conventional bin-packing problem. Indeed, very few researchers have considered inter-task data communication issues or FPGAs imposing technological constraints. Yet the point is that hardware tasks are likely to be linked by means of the data they process and DPR-enabled FPGAs present a series of particularities and limitations that must necessarily be addressed. Not considering these facts may involve serious resource conflict when implementing the scheduling and allocation algorithms on real hardware.

In effect, the big gap that exists between currently proposed scheduling and allocating algorithms for RC and the real capabilities of current FPGAs is one of the reasons why nowadays RC still remains only as a potential computing paradigm. Aimed at filling this gap, in this paper we present “Snake”, a novel strategy for efficiently allocating and executing hardware tasks which has been tested using commercially available partially reconfigurable Xilinx Virtex-4 FPGAs. Furthermore, Snake is compatible with the current trend of making more heterogeneous FPGAs, which can be currently seen in newer Virtex-6 and Virtex-7 devices. With the objective of speeding-up the computation, Snake promotes the reuse of both previously configured circuitry in the FPGA and intermediate partial results between different computation stages. The name is inspired by the chains of hardware tasks that are formed in the FPGA as they are linked together by means of the memory elements where the partial results are temporally stored.

The remainder of this paper is organized as follows. After stating the problem of executing hardware tasks onto DPR-enabled Xilinx FPGAs and introducing the preliminary knowledge in Section II, we summarize the related work in the field in Section III. Next, in Section IV, we describe our novel Snake strategy for allocating and executing the hardware tasks. In Section V, we characterize the performance of our approach by means of both synthetic simulations and a set of experiments using Xilinx Virtex-4 FPGAs, and finally, concluding remarks are pointed in Section VI.

II. PROBLEM STATEMENT AND PRELIMINARIES

A. Xilinx Partially Reconfigurable FPGAs

An FPGA can be modeled as an architecture with two layers: the functional layer, which contains the physical
resources used to perform the computation, and the configuration layer, which controls the configuration of the functional layer.

We focus on Xilinx Virtex-4 FPGAs, but we also target state-of-the-art Virtex-6 and Virtex-7 FPGAs, whose architecture is not fundamentally different from Virtex-4 for the purpose of our study.

The functional layer of Virtex-4 devices includes a regular array of Configurable Logic Blocks (CLBs), Input-Output Blocks (IOBs), clock resources and some special resources such as Block-RAM memories and DSP48s. CLBs are the most abundant resources and provide logic, arithmetic, data storage and data shifting functions. They are organized in a regular array. On the other hand, IOBs, DSP48s and BRAMs can be seen as the heterogeneous resources embedded in the middle of the homogeneous CLB array, being organized in columns which span the whole height of the device. Newer devices accentuate the trend for including more columns of heterogeneous resources, as can be seen in Fig. 1.

All FPGA resources are one-to-one connected by means of a vast amount of programmable wires, allowing for an incredible capability of data movement among registers and memory elements. This is the key for the advanced computational capacity offered by modern FPGAs.

![Virtex-6 family layout (XC6VLX760 part)](image)

**Figure 1:** Virtex-6 family layout (XC6VLX760 part)

The configuration layer consists in a memory, which stores the bitstream defining the functionality implemented by the physical resources in the functional layer as well as their interconnections. The configuration memory is organized in configuration frames of 1312 bits each and the frame addressing scheme includes information related to the position of the physical resources they configure. Specifically, each frame configures a resource column spanning the whole height of a fabric clock region within the chip as shown in Fig. 1. For Virtex-4 devices, each clock region is 16 CLBs, 4 BRAMs, 4 DSP48s or 32 IOBs high. The CLB, DSP48 and IOB frames include configuration information related to both the resources themselves and the associated routing wires to these resources. BRAMs however include separate configuration frames for the memory content data and the associated wires; i.e. while 64 frames are necessary for configuring the 72 Kb memory content data of 4 BRAMs, 20 frames are used for configuring their associated routing resources. Each BRAM content frame stores 256 bits of information and 32 bits of parity of each of the 4 memories mapped to that frame.

The use of SRAM technology in the configuration memory of Xilinx FPGAs permits to download new configuration data at runtime, which is the key for DPR. Modern Xilinx FPGAs ease the access to this memory by including an Internal Configuration Access Port (ICAP), which makes up the interface between both layers of the FPGA. However, the existence of a single ICAP makes the reconfiguration process sequential, although the circuitry previously configured on the FPGA work simultaneously. Virtex-4 ICAP is 32 bit-width and is able to operate up to 100 MHz, allowing a theoretical bandwidth of 400 MB/s.

In summary, an FPGA can be seen as a two-layered device, with the capability of performing on-demand computation, true multitasking and huge internal bandwidth in its functional layer, yet sequential and limited communication bandwidth with its underlying configuration layer. Therefore, the efficacy of RC depends on the capability for masking sequential transfers to the FPGA-based “computation cache” with the advanced parallel computation of its functional layer.

**B. Our Reconfigurable Multitasking Scenario**

In our approach, hardware tasks are managed as if they were processes or threads in a traditional software multitasking operating system because we think that preserving the software programming style is, after decades of prevalence, the only way to guarantee the success of any new computer platform. Hence, the execution of the reconfigurable application is driven by a software program, which is executed in the main system processor. The program includes task definitions but the body of these tasks is defined in hardware.

Hardware tasks turn the fine-grained and generic Xilinx FPGAs into coarse-grained and specific computing machines. Without loss of generality, we consider that hardware tasks transform an input operand stream into an output result stream, being triggered when all their input operands are ready to be processed. That is, we are targeting an event-driven data-dependant computation model. Likewise, in our
RC environment, data exchanges among tasks (inter-task communications) are carried out only prior to task execution start, with the computation thereafter performed atomically. Note that this scheme avoids most of communication related problems in RC, such as deadlocks or race conditions, without significantly constraining the achievable performance. In a general way, we model a reconfigurable application as a graph where vertices represent hardware tasks and edges represent inter-task communication channels (See Fig. 6a). When partitioning a reconfigurable application, related circuitry must be grouped together to exploit the high bandwidth communications among their internal components, and thus, reduce inter-task communication requirements. In a second phase, loosely coupled pieces of circuitry in time or content must be extracted from the tasks until the desired granularity is achieved. The criteria for setting the size and shape of the hardware tasks in our RC system is described in Section IV.B.1 of this paper.

The hardware tasks’ internal architecture is defined by means of a partial bitstream. Again, without loss of generality, we assume that this architecture must include an input data buffer, where the input operands are stored, and an output data buffer, where the output results are to be written to. When the amount of data to be processed is high, both the input and the output data buffers are implemented using BRAMs. We call these tasks high-bandwidth communication tasks. Otherwise the buffers are implemented using the storage capabilities of the CLBs (LUT-RAMs). We name these tasks as low-bandwidth communication tasks. As shown in Fig. 2, both buffers define the spatial boundaries of the tasks, making up a very versatile Communication Interface (CIF) for them [2]. This enables the traditional software programmers to benefit from the advanced computation capabilities delivered by the hardware tasks without having to know hardware-related low-level details; i.e. they only have to write the inputs to certain positions mapped into the memory (Output data buffers of the main processor) and retrieve the results after some time from other positions (Input data buffers of the main processor). Additionally, in a true multitasking environment, a mechanism for synchronizing the execution of the concurrent tasks must be provided. To this end, we have proposed a Hardware Semaphore (HWS) in [2]. The HWS is a special register included in the CIF which acts as the reset signal for the task’s hardware core. The task starts computing only when its HWS is enabled and once it finishes its execution, the task itself disables the HWS. Therefore, the HWS is active only while the task is performing active computation and hence, polling can be done on it when the execution time of the tasks is not known or cannot be predicted with precision.

While a partial bitstream defines the internal architecture of the hardware tasks, the configuration frames used to store this bitstream determine the position of the tasks within the FPGA. Therefore, as occurs with their software counterparts, hardware tasks can be relocated to different positions online by simply changing the frame addresses when their bitstreams are loaded into the configuration memory [3]. However, due to the hardware nature of these tasks, their relocation must be carefully treated. First, the target position must be identical to the original one in the type and arrangement of resources it contains. To meet this condition the presence of heterogeneous resources, such as IOBs, DSP48s and BRAMs, are a major issue but not the only one. The routing signals that connect other parts of the system must be preserved in the target position. Online merging the partial bitstream of the relocating task with already existing routes in the target position is a tedious and long time consuming process which is not always successful; i.e. in the worst-case, a task relocation can be prevented if the task needs to use the same routing resources already assigned to the existing signals in the target position. Second, the communication interfaces must be preserved in the relocated position, otherwise, the task will be isolated and will not be able to process data.

To tackle these problems we have proposed to use the ICAP interface to establish on-demand “virtual” channels among the hardware tasks. Since all the physical resources are mapped to the configuration memory, it is possible to access them through the ICAP. More specifically, the content of the data output buffer of a sender task can be read-back from the configuration memory and written into the configuration frames where the input data buffer of the receiver task are mapped (See Fig. 3). Note that this communication mechanism works regardless of the position of both sender and receiver tasks within the FPGA. Analogously, the synchronization of the tasks execution is also carried out through the ICAP by accessing the HWS. Using virtual channels and ICAP-through accessible HWS instead of physical wires in the functional layer leads to a route-free chip and as a result, the task relocatability is improved.

The tasks are fully relocatable computing structures where the clock is the only signal crossing its boundaries; i.e.
all their internal connections are self-contained within their boundaries. This means no circuitry remains in the chip when they are swapped out from the FPGA. As a result, task boundaries are flexible and can change along time, circumventing the internal fragmentation which arises in the traditional reconfiguration flow when fitting small hardware tasks into larger area reconfigurable slots with fixed boundaries [1]. Moreover, in our approach the number of concurrent tasks is limited only by the amount of resources in the device and not by the amount of reconfigurable slots defined at design phase.

Figure 3: Inter-task communication: Data relocation

The process of executing a hardware task \( \theta_i \) onto the FPGA involves five different phases (See Fig. 4). First, the circuitry of the task must be configured into the FPGA; we name this phase as task allocation and requires \( t_{A,i} \) units of time to complete. Then, the input operand stream must be copied to its input data buffer; we name this phase as input data delivery and requires \( t_{D,i} \) units of time. When the data is ready to be processed by the task, the HWS is enabled and the task starts its execution phase. Thanks to the temporal isolation among different tasks execution, it is possible to guarantee that every task \( \theta_i \) will produce the output results \( t_{E,i} \) units of time after it started the computation, regardless of system workload. To acknowledge its ending the task automatically signals its HWS. The fourth phase is the synchronization phase, which prolongs for \( t_{S,i} \) units of time. During this phase the HWS is checked to verify the correct termination of task’s computation. In the affirmative case, the output results are next read from the output data buffer of the task. We name this phase which takes \( t_{R,i} \) units of time output result retrieval. Overall, the computation time of the task \( \theta_i \) is equal to: \( t_{C,i} = t_{A,i} + t_{D,i} + t_{E,i} + t_{R,i} \), and the time the ICAP is occupied is equal to: \( t_{ICAP\_busy} = t_{A,i} + t_{D,i} + t_{S,i} + t_{R,i} \).

Figure 4: Hardware task’s execution phases

Although our approach benefits from true on-demand multitasking (i.e. the number of concurrent tasks at each time varies based on the application demands) and efficient resource exploitation (i.e. the size of the tasks is exclusively determined by their resource requirement), its weakness is the intensive use of the ICAP due to inter-task communications, and in lesser measure, due to inter-task synchronization. While the relocation of low amounts of data among CLBs does not consume much time, this is not the case when relocating high amounts of data among BRAMs. Snake is aimed precisely at reducing the usage of the ICAP in order to improve the performance, being mainly oriented to tasks that process high amounts of data. The fact of more efficiently masking the bottleneck introduced by ICAP enhances the performance delivered by FPGAs.

III. RELATED WORK

This Section gives an overview of the most significant approaches for allocating hardware tasks onto partially reconfigurable FPGAs.

In 2000, Bazargan et al. proposed to Keep the set of All Maximal Empty area Rectangles (KAMERs) in the device [4]. They used several bin packing-based fitting heuristics (e.g. Best-Fit, First-Fit, Bottom-Left-Fit, etc.) to decide whether a MER had to be split vertically or horizontally when allocating a new task. Ahmadiania et al. presented a variant of Bazargan’s algorithm which managed the occupied area instead of the empty area [5]. With the time KAMER became the key reference in the field, and several authors proposed different ways to keep track of the MERs. Walder et al. proposed to use a hash matrix of the same dimensions of the FPGA device in which every entry consisted of a pointer to a list of MERs of the corresponding size [6]. In [7] a binary tree was proposed in which each node represented an occupied location in the device, while each leaf represented a MER. Note that the latter is one of the few works that consider inter-task communication issues; i.e. the authors were aimed at finding an optimal placement for the hardware tasks such that the Manhattan distance to the centers of a subset of previously allocated tasks was minimized.

The task was not until 2004 that a completely new allocation strategy was proposed: Adjacency-based heuristics. The 2-D adjacency (2DA) heuristic is reported in [8]. The authors were aimed at placing the tasks at the position with the highest contact in \( x \) and \( y \) directions with previously running tasks or with the device’s boundaries. The same authors proposed the 3-D Adjacency heuristic (3DA) as well [9]. 3DA prolonged in time the contact perimeter of the tasks leading to vertical surfaces. Hence, the tasks were placed at the position with the highest contact in \( x, y \) and \( t \) directions. As a result of including the time domain in the analysis 3DA outperformed 2DA [9]. In fact, 3DA is currently one of the most effective heuristics for efficiently allocating swappable
However, the vast majority of these research efforts are aimed at improving the computation density when allocating the tasks onto the FPGA device, neglecting inter-task communication issues. Furthermore, the 2-dimensional area model assumes are too abstract and can be considered incomplete as it does not account for the existence of heterogeneous resource columns nor fabric clock regions in the chip. In fact, most of these approaches were proposed before the first dynamic bitstream relocation was reported to be successful and some of them even before clock region spanning configuration frames based FPGAs were launched.

IV. THE SNAKE APPROACH

This Section presents our novel Snake approach. We first summarize our previous work, where we basically considered a traditional 2-D homogeneous FPGA area model, and then we describe our extension to target both a heterogeneous (and realistic) FPGA area model and the inter-task communication related issues when allocating the hardware tasks.

A. Our approach for the 2-D FPGA Area Model

Previously we have proposed two novel algorithms for swappable hardware task placement: the Empty Area / Volume Compaction algorithms (EAC and EVC, respectively) [11]. Both are aimed at dealing with fragmentation, trying to keep the empty resources adjacent. Specifically, the MER is preserved as long as possible for future use. Moreover, both EAC and EVC manage the reconfigurable device as a single resource, instead of splitting it into non-realistic independent pieces of area (as KAMER does), or considering only the boundaries of already running tasks (as 2DA and 3DA do).

EAC assigns a value to each FPGA position according to the measure it contributes to form adjacent pieces of empty area in the device. The EAC score for a given task allocation is thus computed as the sum of the values corresponding to the resources to be assigned to the task in that position. The placement quality of the allocation is inversely proportional to the EAC score and hence, a low EAC score means that the empty area in the device is not significantly fragmented when allocating the task in that position.

EVC includes the time domain in the analysis as well. Therefore, it assigns a value to each FPGA position according to not only to the measure it contributes to form adjacent pieces of empty area in the device when the EVC algorithm is executed (area-domain), but also according to the amount of time the adjacent tasks to that position need to complete their computation (time-domain). The allocations with low EVC score are preferably selected.

While EAC and EVC improve the performance of related work in the field, by reducing at maximum the negative effect provoked by fragmentation, they rely on using a non-realistic FPGA area model which does not distinguish between different types of resources. Furthermore, they are not able to exploit FPGA’s internal bandwidth for inter-task communications, incurring high ICAP utilization when moving high amounts of data among related-tasks that are likely to be placed distant from each other. In any case, note that these algorithms continue to be useful for allocating low-bandwidth communication tasks.

B. Snake

1) Sizing and Shaping the Hardware Tasks: In the reconfigurable computing scenario we are targeting the hardware tasks are independently sized and shaped, leading to a better usage of the FPGA computational resources. In order to provide the system with a higher flexibility for dealing with fragmentation, several versions of the same hardware tasks are used. Each of them defines a different direction of the computation and uses different input and output buffer locations (See Fig. 5). Note that in this figure the white arrows represent the data movement as the computation is performed, that is, the direction of the computation.

Aiming at best exploiting the FPGA resources, the hardware tasks must always span a minimum number of clock regions in height. In the horizontal direction the criterion changes. There are no constraints for low-bandwidth communication tasks which use abundant LUT resources for input and output data buffers. On the other hand, some of the high-bandwidth communication tasks (e to i versions in Fig. 5) must lie between two columns of BRAMs. For the others (a to d), the width is chosen with the only constraint of fitting the necessary amount of resources. This means that a pair a-c or a-d or b-c or b-d hardware task versions can flexibly exploit the computational resources between two BRAM columns.

Again, in order to best exploit FPGA resources, the size of the input and output data buffers of high-bandwidth
communication tasks should be an integer multiple of 4 BRAMs (72 Kb) and multiple of 16 CLBs (1 Kb) for low-bandwidth communication tasks (given the granularity of configuration memory read/write, see Section II).

Note that due to the existence of different zones with different resource patterns in a typical FPGA layout, several implementations of the same hardware task version must be provided. In any case the existing symmetry in the FPGA defines identical zones, reducing the amount of bitstreams to store. For instance, in Fig. 1, the partial bitstreams for the left-side zones A, B and C can be relocated in their right-side counterparts A’, B’ and C’, respectively.

2) Reusing Partial Results (Inter-task Communications): As explained in Section II.B, the hardware tasks read a stream of data from their input data buffer, perform the required operations to it and write the results in their output data buffer. Then, another task will use these results as input data and produce a new set of results. This process is repeated until the final results are computed; i.e. the last task in the reconfigurable application’s flow is executed. Finally, these results are passed to the main processor.

If possible, Snake allocates the receiver task in such a way that the BRAMs of its input data buffer match with the BRAMs of the sender task’s output data buffer. Hence, the partial results are reused by the receiver task, which accesses it in the same position where it was created. In this context, the BRAMs act in a similar way as the FIFO queues commonly inserted in-between pipeline stages. Note that BRAM sharing also applies to the output data buffers of the main processor.

To do this, the BRAM content frames are removed from the receiver task’s partial bitstream (to preserve previously computed results by the sender task) and the BRAM routing frames of the sender task are replaced by those of the receiver task. This process permits to switch the connection of the BRAMs between the two tasks by simply writing the 20 routing frames associated to those memories (See Fig. 6c). The BRAM switch is completed in $t_{SW}$ units of time. Note that the process of relocating the data of the sender task’s BRAMs to the position where the receiver task is placed through the ICAP would involve to read and write 64 content frames, requiring $t_{DR-C}$ units of time to complete. Note that in the same time nearly 6 CLB columns can be fully configured in the device. In this context, the ICAP is used only to relocate the final results to the input data buffers of the main processor.

The linking together of the hardware tasks by means of the memory elements where the partial results are temporarily stored give rise to computation chains within the FPGA, as shown in Fig. 6b. The chains are initiated in the output data buffers of the main processor (Heads). Data propagate between successive links in the chain through the functional layer, and the final results are moved trough the configuration layer to the input data buffers of the main processor (Tails), being then accessible for the software program.

Snake also manages Routing-Exclusive pieces of Circuitry (REC) to perform fast data relocation among adjacent BRAMs through the functional layer (Chequered in Fig. 6). The allocation of RECs (marked as ‘a’ in Fig. 6b) is very fast due to bitstream compression; i.e. since these circuits only include the routing paths among the BRAMs, most of the frames are empty. The amount of time needed by the REC to relocate the data is denoted as $t_{DR-F}$. As shown in Fig. 6, this solution is efficacious for circumventing the small pieces of reconfigurable area in the device due to the heterogenous resources of modern FPGAs. In addition, note that RECs can replicate the content of one BRAM to multiple memory blocks at a time, which is very useful for dealing with computation branches (e.g. $\theta_1 \rightarrow \theta_2, \theta_4$ in Fig. 6).

3) Reusing Previously Configured Circuitry: The above-presented way of sharing BRAMs between a pair of sender-receiver tasks in the computation chain is suitable for promoting the circuit reuse. By only sharing the BRAM routing frames, it is possible to keep both tasks allocated and in case the sender is triggered again, it can be re-executed by simply switching again the connection of the BRAMs. Hence, Snake allows to fully, or at least partially, re-execute repetitive patterns of tasks that were previously executed. This significantly reduces the occupation of the ICAP and accelerates the execution of the reconfigurable application.

4) Dealing with Fragmentation: In order to deal with fragmentation, the low-bandwidth communication hardware tasks are allocated using EAC and EVC heuristics into the ‘holes’ that are created within the high-bandwidth communication tasks chains (E.g. in the region between $\theta_3$ and $\theta_5$ in Fig. 6). Besides this, the RECs presented in Section IV.B.2 are used for dealing with fragmentation as well.

5) Making the Allocation, Relocation (and Deallocation) Decisions: Partial results and circuitry reuse are two efficacious ways for reducing the occupation of the ICAP.

However, when a hardware task is already allocated, but far away from the memories where its input data stream is stored, Snake must decide whether to reuse the circuitry or the partial results. Note that in this situation it is not feasible to reallocate the data by means of a REC. Relocating the data through ICAP to the input buffer of the already configured task consumes $t_{DR-C} + t_{SW}$ units of time. On the other hand, relocating a new instance of the task close to the memories with the input data stream requires $t_{A,i}$ units of time. Consequently, the operation that introduces the minimum time overhead is performed: if $t_{DR-C} + 2 \cdot t_{SW} > t_{A,i}$, the task is relocated and viceversa.

When the receiver task cannot be placed next to the sender task, Snake must decide whether to deallocate some of the
surrounding tasks to free space for it, or to reallocate the data to another position within the device. Despite deallocating the tasks leads to the minimum execution time overhead, the performance would be affected in case the deallocated tasks are triggered in the future. The criteria used by Snake is to compare the potential amount of time that the future reallocation of these tasks would require ($\sum_{i} t_{A,i}$ with $i \in \text{deallocated tasks}$) with the amount of time the data relocation to another position would involve ($t_{DR-C}$ when using ICAP) or $t_{DR-F} + 2 \cdot t_{SW}$ (when using a REC, if possible). As the first term in the comparison represents only a potential source of performance reduction, we weight it with a value of $\frac{1}{W}$. This means that data will be relocated to a different position only when the potential overhead introduced by task reallocation is at least $W$ times greater than $t_{DR-C}$ or $t_{DR-F} + 2 \cdot t_{SW}$.

In this context, Snake relies on EAC and EVC to:

- Select the most suitable version (the one which reduces the fragmentation in the device) of the arriving hardware task when the task is to be linked to the task-chain.
- Find the best allocation for the arriving task when its input data stream is relocated to a new position, or
- Allocate low-bandwidth communication tasks.

V. EVALUATING SNAKE’S PERFORMANCE

We have implemented a proof-of-concept prototype which uses our Snake approach in a Xilinx XC4VFX20 FPGA device. A Xilinx MicroBlaze soft-processor acts as the main processor and the ICAP is controlled by a Finite State Machine (FSM), which itself is driven by a Xilinx PicoBlaze 8-bit microcontroller. The system runs at 100 MHz, which is the highest clock frequency allowed by the ICAP. Table I shows the most significant performance results measured in our system.
we presented in [2] (e.g. ICAP-based inter-task communication and synchronization) and EAC/EVC allocation algorithms [11]. Snake promotes the reuse of both partial results between different computation stages and previously configured circuitry on the device. We have characterized Snake’s performance by means of a wide range of synthetic reconfigurable applications and we have provided a set of real results measured when implementing our solution using commercially available Xilinx Virtex-4 FPGAs.

REFERENCES


