Developing and Implementing Dynamic Partial Reconfiguration for Pre-Emptible Context Switching and Continuous End-To-End Dataflow Applications

Alex Beasley[1], Luke Walker, Dr. Chris Clarke

[1] A.E.Beasley@bath.ac.uk

Department of Electrical and Electronic Engineering, University of Bath

Abstract:

This study explores the benefits of the Dynamic Partial Reconfiguration (DPR) on Field Programmable Gate Array (FPGA) based System on Chip (SoC) architectures. Consideration is given to the constraints imposed by the implementation of partial reconfiguration on both pre-emptible context switching and continuous end-to-end dataflow applications. Skeleton structure systems that permit the insertion and removal of 'blocks' into the overall FPGA floorplan have been developed. These can be reconfigured dynamically by the on chip system host even during data processing. In pre-emptible context switching maintaining the execution state of a design before switching away from it becomes of paramount importance; this work presents a new Pre-emptible Flip Flop (PFF) design that is used as a basis for a Task Specific Access Structure (TSAS) for FPGA designs and then proposes an algorithm to automate the insertion of these PFF's into a synthesised design. Expanding into continuous dataflow application design allows the system host to re-route signals during the partial reconfiguration process and then re-establish the processing chain with the new configuration hence maintaining a continuous uninterrupted dataflow.

The design flow used in this work makes use of non-standard IP and tools that are not supported by Altera for the Cyclone V SoC. However, as this paper shows, partial reconfiguration is possible inside the Cyclone V SoC device. For further information please contact the authors.

i. Introduction

The development of System on Chip Field Programmable Gate Arrays (FPGA-SoC) have brought about significant increase in functionality from the traditional FPGA. The combination of the dedicated silicon processor alongside the FPGA fabric on the same chip has superseded the previous method of instantiating "soft core" processors out of the FPGA fabric to provide support for software routines that can interact with the FPGA along a low latency bus. Embedding a dedicated processor (Hard Processor System — HPS) provides greater system flexibility such as the opportunity to dynamically reconfigure the FPGA at run-time. Dynamic reconfiguration of the FPGA fabric allows the realisation of adaptable, software-like, designs to be instantiated and executed on the FPGA. Thus making full use of the advantages of using dedicated hardware without the massive overhead that comes with implementing all possible algorithms at once that a system may need in its lifecycle.

This paper explores and develops two potential applications of this technology: pre-emptive switching between complete systems including hardware accelerators and the on-the-fly modification of data or signal pipelines. Modern CPUs achieve multitasking by switching between software tasks pre-emptively; requiring saving the context ('state') of the current task so that it may be restored later. Based on this methodology a similar system can be achieved on an FPGA by saving and restoring the context of the registers [1]. Unlike the context of a software program (which consists of a small number of registers) the context of an FPGA design can be large; including the values stored by all flip-flops and registers, as well as any memory. This is the first type of system to be explored. The second type of system brings about even more challenges as it requires the system output to remain functioning even while the partial reconfiguration of the pipeline is occurring.

On some FPGA devices context saving or loading may be performed via a Configuration Port [2],[3],[4]; however this is not present on all devices and can result in a large amount of redundant context data being saved or loaded [5]. Task Specific Access Structures (TSAS) provide a more generic solution to this problem which can provide the frame work to be able to save and load data as required for a pre-emptible context switching design. They trade of an increase in the amount of resources needed to implement them with a reduction in system redundancy.

Dynamic Partial Reconfiguration (DPR) is the basis for implementing these new structures on FPGAs. Since this technology allows for on-the-fly reconfiguration of the FPGA it is possible to dynamically alter the systems behaviour or even change the system running all together while the device is active in the field. In order to achieve this skeleton structures had to be developed that allow the implementation of both context switchable designs and partially reconfigurable pipelines without the need for the FPGA to be reconfigured by the manufacturing process. For the context switchable designs this entailed the need to design a new pre-emptible flip flop (PFF) design to build a TSAS from and in the case of the task pipeline the structure had to be able to provide a continuous output that isn't disturbed the reconfiguration process.

The contributions of this paper are the development of dynamically reconfigurable systems for FPGAs in which their benefits and implications are explored. From the development of these systems a new PFF design has been created based on the system explored in [6] and a method to convert non-context switchable implementations to context switchable implementations automatically. Further to this, systems are created that dynamically alter the behaviour of a pipeline that has been implemented on an FPGA while maintaining a constant output.

The structure of the remaining part of this article is as follows: Section 2 details how dynamic reconfiguration of an FPGA is accomplished and the two types: full and partial reconfiguration. Section 3 discusses the new pre-emptive flip-flop for the context switchable systems and proposes a system for the automatic implementation of this system. Section 4 describes a dynamically reconfigurable pipeline that maintains the systems output at all times. Section 5 demonstrates the feasibility by comparing the save/restore performance as well as the resources used with and without scan-path structures in two HAS tasks on an Altera Cyclone V SoC. Finally Section 6 presents the Conclusions and Future Work.

ii. Dynamic Reconfiguration

Dynamic Reconfiguration of an FPGA is the re-assignment of the FPGAs configuration at run time. This means the device remains operational before, during and after the process and there is no need for power cycling to bring the new configuration online. This takes two forms: Full Dynamic Reconfiguration (FDR), used to change the entire system, and Partial Dynamic Reconfiguration (PDR), acts upon a portion of the FPGA lay out while allowing the rest of the device to function as normal. FDR prevents the FPGA from providing the user with an output while the reconfiguration is taking place however it is easier to implement and has a reduced risk of causing Single Event Upsets (SEU) as the entire device configuration is scrubbed prior to loading the new configuration. PDR requires the design of a wrapper around the reconfigurable region that holds the input signals during the reconfiguration process to prevent operation in that region and limit the potential for SEUs.

Both full and partial reconfiguration requires the synthesis of the design files prior to having the system operate. The design environment is used to create the configurations files required for the FPGA for both full and partial reconfiguration. The types of files required for the operations differ: full reconfiguration replaces the entire configuration of the FPGA it uses a standard SRAM object file (.sof) as would usually be used to configure an FPGA, which is then converted to an object file which is usable by the internal host to configure the device. By contrast the partial reconfiguration is only concerned with a section of the FPGA fabric and requires a raw bit stream (.rbf) generated from partial mask SRAM object files (.pmsf). The design flow for partially reconfigurable designs differs from a standard design as the partial reconfigurability utilises design partitioning for the reconfigurable regions and logic locking which prevents the compiler from using the space in the reconfigurable design region for other parts of the design and confines the reconfigurable part of the design to a specific region. Once the logic locked design regions are created reconfigurable design revisions must be created in which the design files may be manipulated to reflect the changes made to the design when partial reconfiguration is initialised. Another type of design revision that may be used is the aggregate revision which is used for timing analysis of the design with the different revisions in place [7]. The design flow is detailed in Figure 1.

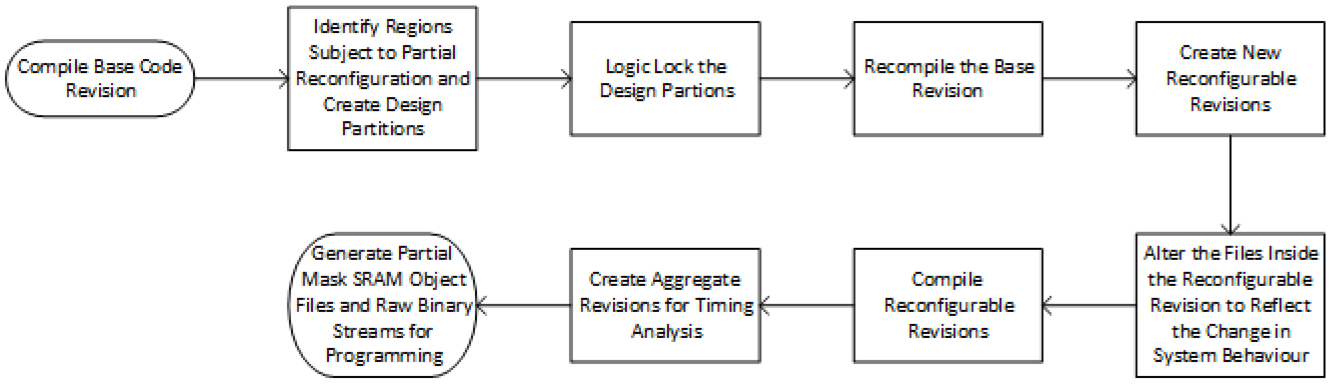
Compile Base Code Revision
Identify Regions Subject to Partial Reconfiguration and Create Design Partitions
Logic Lock the Design Partions
Recompile the Base Revision
Create New Reconfigurable Revisions
Alter the Files Inside the Reconfigurable Revision to Reflect the Change in System Behaviour
Compile Reconfigurable Revisions
Create Aggregate Revisions for Timing Analysis
Generate Partial Mask SRAM Object Files and Raw Binary Streams for Programming

Figure 1 - Design flow for partially reconfigurable systems

Partial Reconfiguration creates regions of 'reconfigurable' logic that sit in pre-designated areas within the FPGA floorplan, these must then be connected to the static logic surrounding them Figure 2. The most practical way to do so is by means of a standard interface that is coherent for each configuration as it important to remember the ports in the static logic cannot be altered once configured. By keeping a consistent interface to each re-configuration there is a reduction in unnecessary port declarations leading to wasted interconnect usage.

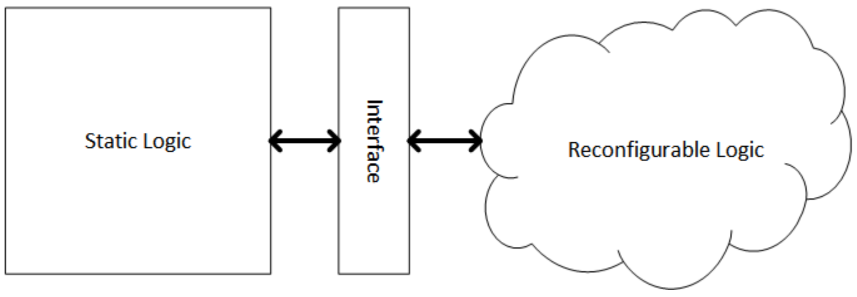

Figure 2 - Connection between static and reconfigurable FPGA regions

Currently the process of dynamic reconfiguration takes milliseconds to achieve which may limit its potential applications. The advantages of on the fly reconfiguration outweigh this current limitation. The device becomes infinitely more flexible than it used to be allowing them to become a feasible alternative to a processor to perform traditional processor tasks. They have been proven to be significantly faster than GPPs, CPUs or DSPs to perform certain tasks (i.e. take fewer clock cycles) [8], making them more efficient and quicker to complete a task leading to higher system throughput. There is the potential for less system overhead compared to designs that have previously been implemented on more than one FPGA device to provide the necessary floor space. FPGA designs also tend to be more deterministic than equivalent processor designs meaning the arrival of the system output is more consistent and less likely to cause a system to 'hang'.

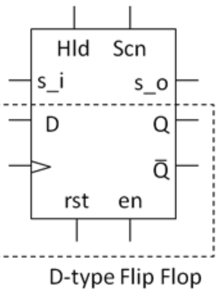

Figure 3 - Pre-emptible Flip-Flop Connections

iii. Context switching design

The specific details of the PFFs used in the context switchable HAS differ slightly from those presented by others [6] The PFFs implemented in this design require two new signals: HOLD and SCAN. The HOLD signal is shared with each flip-flop, when it is held high the operation of the flip-flop is stopped. It must be noted that as this is a shared single bus for this signal that proper system timing considerations must be considered when designing the system so that the HOLD signal for every flip flop is the same on each clock edge. This method only considers single clock domain systems close to the HPS fabric as asynchronous systems by their very nature cannot be stopped on a single clock edge. The SCAN signal is controlled by the HPS and used to control the propagation of the state of each PFF along the scan chain.

The developed PFF operates as follows: while the HOLD and SCAN lines are low the flip-flop operates as normal; if the SCAN line is high during this time, then this does not affect the flip-flop behaviour. When the HOLD line goes high the flip-flop is halted, and no longer under the control of the clock, reset and enable lines. If the SCAN line is high during this time, then positive clock edges cause the value on the PFF to shift its value along to the next PFF in the scan-path.

The implementation could be achieved using similar methods to scan chain insertion for ASIC testing [9], permitting automation in the FPGA design flow. This will allow designers to take existing FPGA designs and convert them to context switchable designs to work as part of HAS. It is therefore important that the design of the PFF only enhances the behaviour of the original flip-flop without compromising functionality. Based on these requirements, the PFF is modelled upon a standard D-type flip-flop with 4 additional ports: scan-in; scan-out; HOLD and SCAN Figure 4 . Scan paths are constructed by connecting the scan-in and scan-out ports on neighbouring flip-flops Figure 5 . Internally the scan-in port and the D port may be multiplexed together to the original flip-flop's D port; the scan-out port and Q port may both be internally connected to the Q output of the original flip-flop, however for ease of demonstration a port has been added to represent the behaviour while the design is halted.

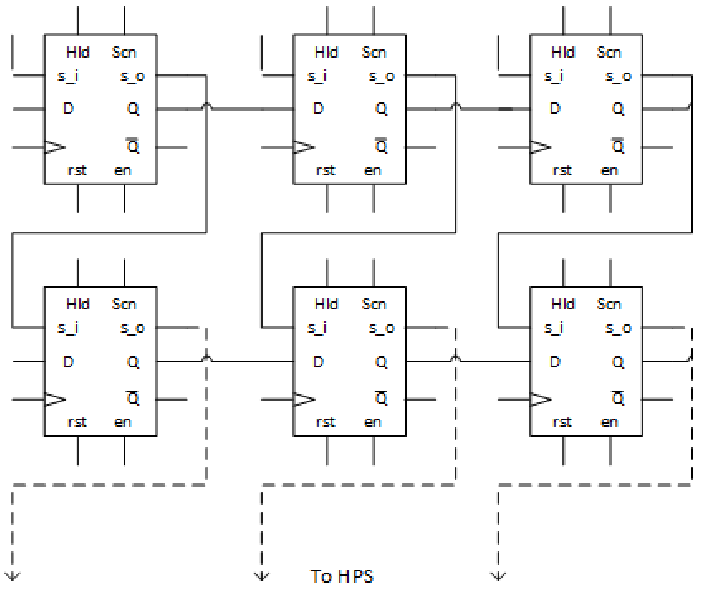

Figure 4 - Scan-chain connected to ports on HPS [6]

The current implementation of the PFF has been achieved by manually inserting the modified flip-flop at a behavioural level using a Hardware Descriptive Language (HDL). SystemVerilog provides a new *interface* feature that can be used to connect modules together in a quick and consistent manner. This was used to connect the new HOLD, SCAN, scan-in and scan-out ports which could then be passed through the module port-list to the next level in the design. This process is repeated until the top-level module, where the *Interface* structures are instantiated and connected together. At the top level of the design the flip-flop chain are connected to the save and load ports of the HPS, and additional controller modules are instantiated to connect the SCAN and HOLD controls to the HPS. Flip-flop chains must be instantiated as multiples of 32 parallel chains to enable the context to be saved as quickly as possible through a 32-bit Lightweight AXI bridge to the HPS. When a design is unable to divisible the number of flip-flops by 32, additional padding flip-flops are added to create the complete scan-path.

Manual insertion at the behavioural level is naturally a slow and laborious implementation: human nature could cause context storage elements can be overlooked as well as leading to more verbose code. An automated method of conversion between non-context switchable and context switchable designs is therefore proposed. The PFF has been designed so that its connections simply augment the original flip-flop design, therefore after synthesis the gate-level netlist could be parsed to search for flip-flops / structures which will hold the state of an FPGA design. Essentially this creates an automated process operating at the gate-level. A parser is able to identify the flip-flops that need converting and replace them with the PFF then automatically link the additional scan- ports together. The automated process would also add-in any controlling modules to the top-level module. This process could even be included as part of the synthesis engine.

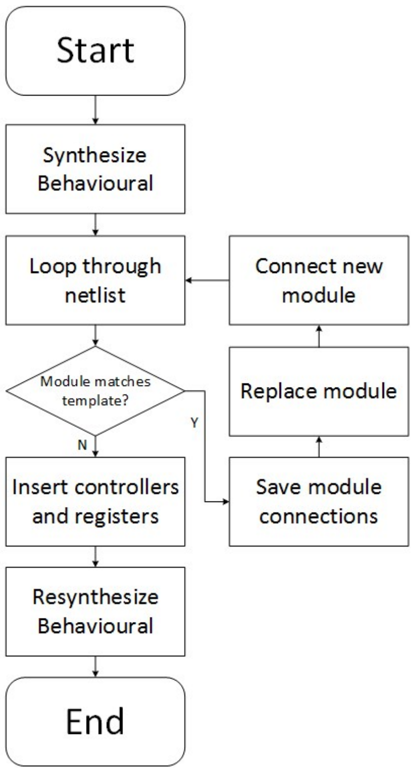

Figure 5 - Flow chart potential automatic insertion system for new PFF's

Figure 6 demonstrates the automated process as a flow chart — a program parses the net list and checks each module instantiation against a list of context storage elements. If a match is found, the old flip-flop is replaced with a new PFF maintaining the original connections. Once the end of the file has been reached, the total number of PFFs is known, padding may be added as required with the inputs connected to ground. Next the SCAN and HOLD ports can be connected to the required drivers and the scan-in and out ports can be connected together. In order to make this process independent of the specific FPGAs architecture the new file will then undergo re-synthesis for the target device.

iv. Pipelined Data Processing Systems

Pipeline processing breaks down a large task into a series of sub-tasks [10], each one fed by its predecessor, such that the output of the system is the same as performing the operation as one. Data pipelining is traditionally applied to streamed data, for instance in the case of multimedia, networking, digital security, scientific processing etc. by embedded processor. Nawinne *et al.* looked at applying this in a Multiple-Processor System on Chip (MPSoC) architecture in [10] but with the advancements in FPGA-SoC devices and the ability to perform partial reconfiguration on the fly, it is now possible to implement this technology on FPGA's. The dynamic partial reconfiguration capability of the FPGA device means it can be viewed as a more "software algorithm" friendly device where hardware implementations of software processes can be realised, thus making use of the FPGA's speed at performing tasks and more deterministic output while maintaining the flexibility to change the pipeline based on the needs of the user.

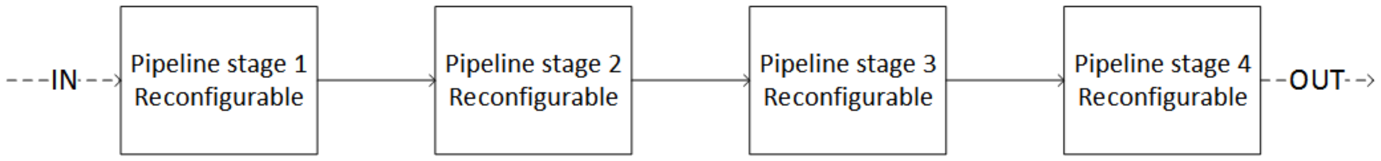
IN
Pipeline stage 1 Reconfigurable
Pipeline stage 2 Reconfigurable
Pipeline stage 3 Reconfigurable
Pipeline stage 4 Reconfigurable
OUT

Figure 6 - Pipeline system

The ability to keep reconfigure the pipeline while keeping the system output constant affords greater opportunities for where this could be implemented, for example in the case of live audio or video processing, such that the end user does not notice the transition from one system to another. A frame work has been created that works in conjunction with the systems embedded host to manipulate the dataflow around the area undergoing reconfiguration and then re-establish the data flow path with the new configuration, which has been demonstrated pictorially in Figure 8.

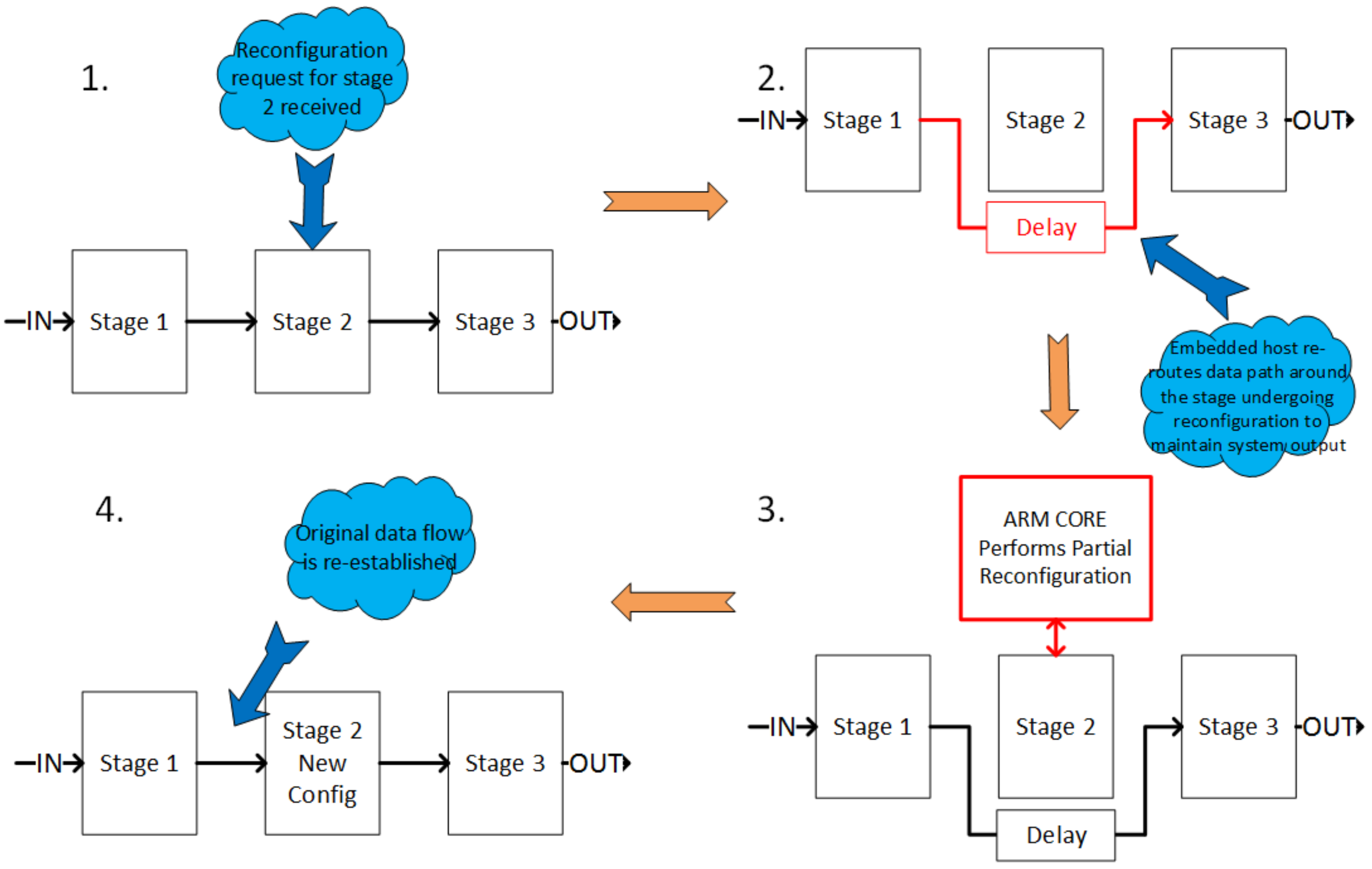
1.
Reconfiguration request for stage 2 received
IN
Stage 1
Stage 2
Stage 3
OUT
2.
IN
Stage 1
Stage 2
Stage 3
OUT
Delay
Embedded host re-routes data path around the stage undergoing reconfiguration to maintain system output
3.
ARM CORE Performs Partial Reconfiguration
IN
Stage 1
Stage 2
Stage 3
OUT
Delay
4.
Original data flow is re-established
IN
Stage 1
Stage 2 New Config
Stage 3
OUT

Figure 7 - Flow diagram of the partial reconfiguration process of a single stage in a pipeline

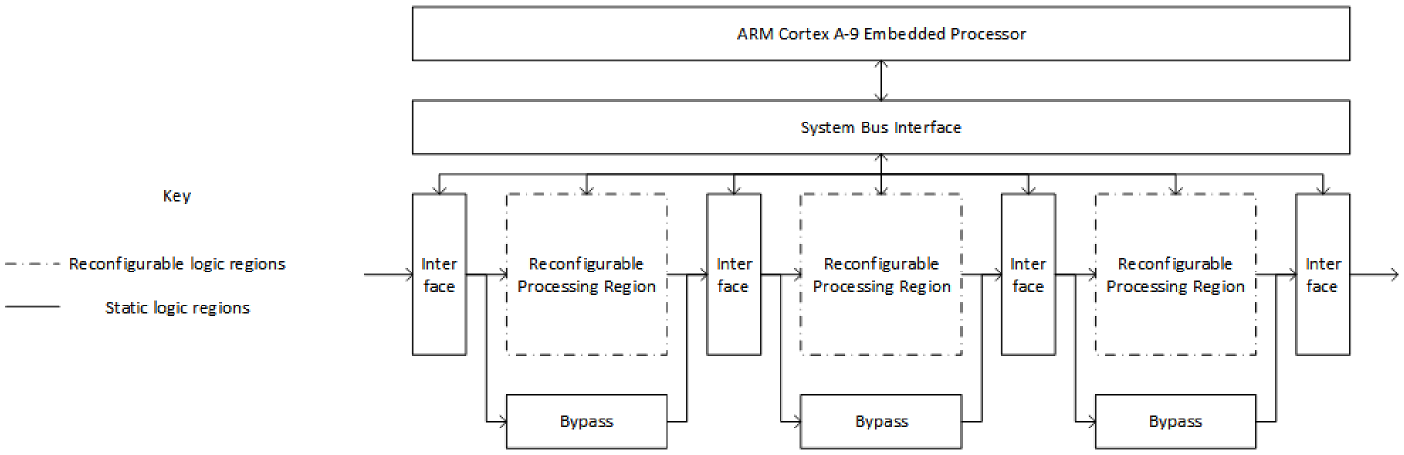
ARM Cortex A-9 Embedded Processor
System Bus Interface
Key
Reconfigurable logic regions
Static logic regions
Inter face
Reconfigurable Processing Region
Inter face
Reconfigurable Processing Region
Inter face
Reconfigurable Processing Region
Inter face
Bypass
Bypass
Bypass

Figure 8 - Example skeleton structure for a partially reconfigurable pipeline with interfacing and bypass chains

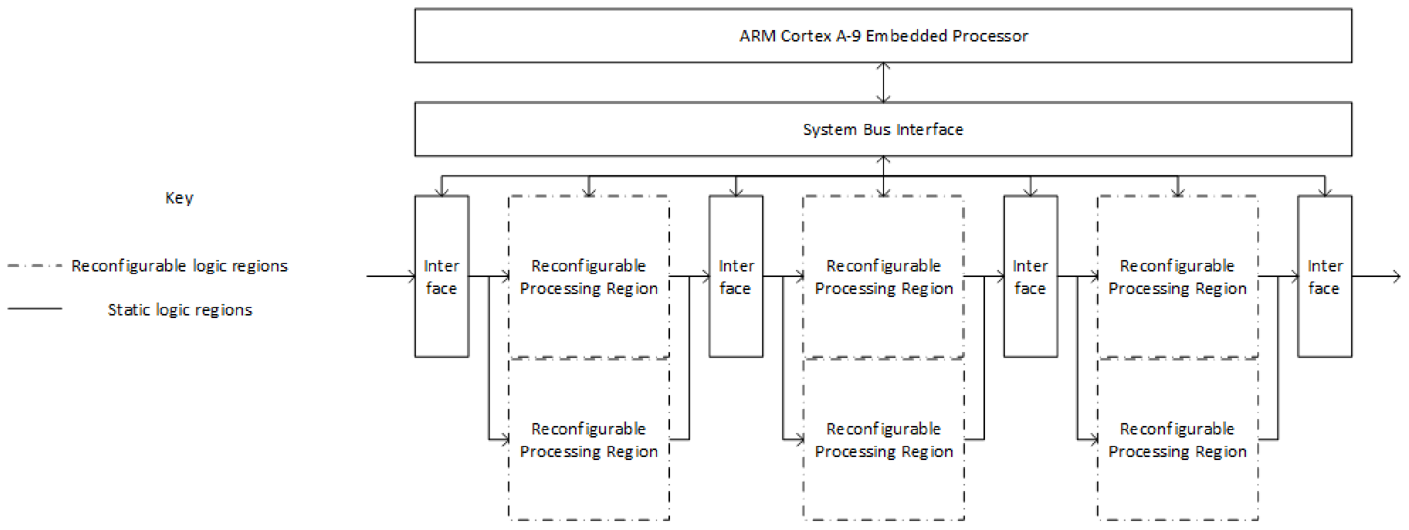
ARM Cortex A-9 Embedded Processor
System Bus Interface
Key
Reconfigurable logic regions
Static logic regions
Inter face
Reconfigurable Processing Region
Inter face
Reconfigurable Processing Region
Inter face
Reconfigurable Processing Region
Inter face
Reconfigurable Processing Region
Reconfigurable Processing Region
Reconfigurable Processing Region

Figure 9 - Example skeleton structure for a partially reconfigurable pipeline with interfacing and paired reconfigurable regions

There are a multitude of different potential ways the system can be configured with the reconfigurable regions depending on the end application. Figure 9 shows each reconfigurable region with a bypass path that adds delay to the data path such that while reconfiguration is in progress the data is passed through the delay channel and samples aren't lost. This works well for a system where having a few milliseconds of samples that aren't processed doesn't matter in the end result, such as live audio processing. However there are also applications where it is important to ensure the data is always subject to being processed at each point of the pipeline, Figure 10 demonstrates a method to ensure this is the case. Here the reconfigurable regions are arranged as pairs but only one in each pair is used at a time, leaving the other to be reconfigured when it is necessary. Once the reconfiguration has been complete the regions are switched so the new configuration is used. It is important to note that if a block or process in the pipeline that is being switched has inherent delay, this delay is compensated for during the switching to achieve a seamless output of the system.

The system is configured with an internal partial reconfiguration manager, based on the Altera Corp. IP for Stratix V devices, used in conjunction with the embedded ARM core to provide reconfigurability. Configuration files for the device are stored on off chip memory in the form of a micro SD card and a FAT32 driver was written to allow the ARM core to interact with the memory and load new configuration files from it. A wrapper ensures that the signals into the partial reconfiguration region remain static during the process to reduce the risk of SEUs.

v. Experimentation

A. Setup

Both the context switching and the pipelined designs were implemented on a DE1-SoC Development Board (TerASIC), consisting of a Cyclone V SoC (Altera) device and peripherals. To test the performance and functionality of the systems the times to achieve both full and partial dynamic reconfiguration from the embedded ARM core where measured as well as the save and load times for the new pre-emptive flip-flop structure. It was deemed sensible to create several different programs to test the context switching design as the configuration size of an implementation effects the time taken to perform a save or load operation: an MD5 hashing system (MD5) and a LED Patterning design (LEDP), which also had software counterparts which that ran on the HPS portion of the SoC, to measure the effect of hardware acceleration.

TABLE I
ALMs Used Before and After Conversion

Design	Original ALMs	Context Switchable ALMs
MD5	6136.7	7397.7
LEDP	89.2	92.0

TABLE II
SAVE AND RESTORE TIMES FOR VARIOUS FPGA DESIGNS

Design	Configuration Size (kB)	Save Time (μs)	Load Time (μs)
MD5	48	13158	7289
LEDP	0.096	356	20
Reference	0	334	7

FPGA designs were created using behavioural level SystemVerilog in Quartus II Design Software (Altera), with the software counterpart created using C in ARM Design Suite 5 (DS5) Altera Edition. Design verification was provided by both ModelSIM logic simulation and real world testing. Once both dual software/hardware tasks had been verified, they were converted to use the PFF and the behaviour was re-evaluated using the same methods. Table I shows the FPGA ALMs used both before and after the conversion to PFFs. In the case of the MD5 hashing design approximately 20% more ALMs were required, but only 3% for LEDP design. The difference in percentage increase may have been because the former design previously used memory blocks, which had to be resynthesized using Logic Array Blocks (LABs). As there are two different types of memory: Random Access Memory (RAM) or Read Only Memory (ROM), methods for dealing with these will differ. RAM will require the scan chain to save the data from the system when a reconfiguration occurs ready to be loaded back when the system is restored. This requires a wrapper to incorporate scan chain removal of the data. ROM cannot be written to during operation by a given configuration; therefore if a system consists of ROM there would be no need to save the context when switching away from the configuration; only to load the fixed configuration back when the system is restored.

B. Performance

Tests were carried out on the ability of both the software and hardware implementations to save and restore their context, Table II, along with a zero context design used as a reference example. It can be seen from Table II that the time taken to load and save the context of the tasks is significant, however despite this the save and restore times are comparable to those reported by Jozwik et al [3] for TSAS per kB. Jozwik et al achieved times per kB of approximately 266.3µs and 175.1µs respectively compared to the 274.1µs and 151.9µs respectively reported for this system.

Two factors may increase the save and restore times recorded: non-optimised code and non-optimised FPGA communication. At present the code used to save and restore the tasks has undergone no manual or compiler optimisations. Furthermore the save and restore is done through PIO cores provided by Altera mapped via a Lightweight AXI Bridge to the HPS, which has been selected for ease of use, rather than low latency performance. Improvements or changes to either of these factors may yield decreased save and restore times.

TABLE III
Full Dynamic Reconfiguration time for compressed MD5 hashing FPGA design

Design	Time without DMA (ms)	Time with DMA (ms)
MD5 Hashing algorithm	476	51

Device reconfiguration times were also measured and show in Tables III (full dynamic reconfiguration) and IV (partial dynamic reconfiguration). For the full dynamic reconfiguration the FPGA image was compressed resulting in a file size of ~2.5MB and the FPGA configuration was set to FPP x16, compression: enabled, encryption: optional and POR delay: fast. The times taken to perform the full dynamic reconfiguration of the cyclone V device using the FPGA manager both with and without the Direct Memory Access have been evaluated. Although the times taken to perform the full reconfiguration are in the order of milliseconds the effect of adding the DMA results in a substantial approximately 10 times increase on the speed of the reconfiguration process.

TABLE IV
Partial Dynamic Reconfiguration times for uncompressed images of FPGA designs

Design	ALMs	Combinatorial LUTs	Dedicated Logic Registers	Time (ms)
5x5 2D Swap Median Filter	5784.5	3206	19621	7064
Audio Delay with LED counter	396.5	704	91	1783
Audio Delay with LED Flasher (On/Off)	177	251	85	1667

The times taken to perform partial dynamic reconfiguration over the AXI/Avalon bus are then detailed in table IV, these FPGA bit streams are uncompressed due to a limitation of the Quartus II environment not supporting partial bitstream compression for the cyclone V devices. As can be seen here increasing the size of the design to be partially reconfigured results in a much greater reconfiguration time, from 1.7 seconds to just over 7 seconds for the median filter. If a system was required to perform multiple reconfiguration tasks in a short period of time this solution would be inappropriate, however for configurations that will be implemented for extended periods of time the few seconds required to perform the reconfiguration are less of an issue.

While currently this technology is in its infancy, this speed increase between a system without DMA and with DMA proves promising that further developments will significantly improve the time taken to dynamically reconfigure an FPGA. Being able to compress partial binary bit streams would also improve performance for the cyclone V partial reconfiguration time.

vi. Conclusions and further work

It has been shown that FPGA-SoCs are very capable and flexible with their embedded process it is possible to design systems that can reconfigure themselves, both fully and partially on the fly. The paper explored the implications of this flexibility and presented two cases: pre-emptive context switching and continuous end-to-end pipelines. The reconfigurable ability of the devices allows hardware implementations of software algorithms without confining the system to one design or the need for massive overhead to accommodate multiple algorithms that can be switched in and out. Further work will explore methods to decrease the reconfiguration time and to perform comparisons of the time taken to perform dynamic reconfiguration on other Altera products.

REFERENCES

1. Levinson, L., et al. *Preemptive multitasking on FPGAs*. in *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*. 2000.
2. Gong, L.K., O. Diessel, and Acm, *Functionally Verifying State Saving and Restoration in Dynamically Reconfigurable Systems*. Fpga 12: Proceedings of the 2012 Acm-Sigda International Symposium on Field Programmable Gate Arrays. 2012, New York: Assoc Computing Machinery. 241-244.
3. Jozwik, K., et al., *Comparison of Preemption Schemes for Partially Reconfigurable FPGAs.* Embedded Systems Letters, IEEE, 2012. **4**(2): p. 45-48.
4. Blodget, B., S. McMillan, and P. Lysaght. *A lightweight approach for embedded reconfiguration of FPGAs*. in *Design, Automation and Test in Europe Conference and Exhibition, 2003*. 2003.
5. Kalte, H. and M. Porrmann. *Context saving and restoring for multitasking in reconfigurable systems*. in *Field Programmable Logic and Applications, 2005. International Conference on*. 2005.
6. Jovanovic, S., C. Tanougast, and S. Weber. *A Hardware Preemptive Multitasking Mechanism Based on Scan-path Register Structure for FPGA-based Reconfigurable Systems*. in *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*. 2007.
7. Altera. *Performing Partial Reconfiguration*. 2013 [cited 2015 25/08/2015]; Available from: http://quartushelp.altera.com/13.1/mergedProjects/comp/comp/comp_pro_part_reconfig.htm.

8. Chauhan, A., A. Rajawat, and R. Patel. *Reconfiguration of FPGA for Domain Specific Applications Using Embedded System Approach*. in *2009 International Conference on Signal Processing Systems*. 2009.
9. Zaourar, L., Y. Kieffer, and C. Aktouf. *An Innovative Methodology for Scan Chain Insertion and Analysis at RTL*. in *Test Symposium (ATS), 2011 20th Asian*. 2011.
10. Nawinne, I.B., et al. *Heterogeneous processor pipeline for a product cipher application*. in *Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on*. 2011.