A Framework for Hardware-Accelerated Services Using Partially Reconfigurable SoCs

Octavian Mihai MACHIDON[1], Sorin HINTEA[2], Florin SANDU[1]
[1] *Transilvania University of Brasov, 500036, Romania*
[2] *Technical University of Cluj-Napoca, 400027, Romania*
octavian.machidon@unitbv.ro, sorin.hintea@bel.utcluj.ro, sandu@unitbv.ro

***Abstract*—The current trend towards "Everything as a Service" fosters a new approach on reconfigurable hardware resources. This innovative, service-oriented approach has the potential of bringing a series of benefits for both reconfigurable and distributed computing fields by favoring a hardware-based acceleration of web services and increasing service performance. This paper proposes a framework for accelerating web services by offloading the compute-intensive tasks to reconfigurable System-on-Chip (SoC) devices, as integrated IP (Intellectual Property) cores. The framework provides a scalable, dynamic management of the tasks and hardware processing cores, based on dynamic partial reconfiguration of the SoC. We have enhanced security of the entire system by making use of the built-in detection features of the hardware device and also by implementing active counter-measures that protect the sensitive data.**

***Index Terms*—System-on-Chip, reconfigurable architectures, web services, reconfigurable logic**

I. Introduction

Reconfigurable hardware is a technology that has known a rapid development in the recent years, with a growing number of applications in different fields. The key strongpoints that reconfigurable devices bring, adaptability and scalability, reduce the requirements for dedicated hardware and optimize power consumption. Due to the re-configurability features, these devices are ideal in providing flexible solutions for the design of embedded systems [1].

The "Everything-as-a-Service" paradigm, which emerged with the expansion of cloud computing, enables a service-oriented approach of reconfigurable hardware resources. Such an approach – based on the synergy between the two technologies: reconfigurable and distributed computing – has the potential of bringing a series of benefits for both fields [2]. It can help minimize the design complexity and improve the flexibility in the management of IP cores, by making them available as services by using neutral, web-based, technologies. Also, this approach favours hardware acceleration of certain services that require high computational resources. By offloading these services to reconfigurable devices, they can benefit from the computing parallelism and hardware-based acceleration brought on by the FPGA/SoC architectures, resulting in improved performance [3] and also enhanced security [4] - due to active and passive built-in security features of such devices.

This paper proposes a service-oriented architectural framework that implements services as IP Cores in reconfigurable hardware with a dynamic management that uses partial reconfiguration.

This hybrid approach – the web services themselves reside on a dedicated web server, but their implementations are offloaded on high-performance reconfigurable SoCs – enables a performance gain due to the hardware acceleration "behind the scenes", while maintaining the standard, easy-to-use web interface for these services.

The main contributions of this paper are:

- Providing an architectural model for offloading the implementation of web services to reconfigurable hardware devices as IP Cores.
- Implementing an efficient and dynamic management of the IP Cores (installation/removal during run-time) in order to maintain the flexibility of the web services.
- The proposed model increases the security of the IP and user data by enhancing attack detection and active counter-measures in the case of a tampering event targeting the SoC devices.

II. BACKGROUND AND RELATED WORK

While Service-Oriented Architectures (SOA) and System-on-Chip (SoC) are mature technologies being used for more than a decade, with a multitude of applications, the synergy between the two is a more recent research subject.

This synergy has the benefit of bringing together the advantages and solving some of the downsides: by applying the SOA model with its unified and standardized approach on computing resources, SoC architectures gain in programmability while the web services gain in performance, being dynamically integrated in the device's programmable logic (PL) and thus hardware-accelerated.

The hardware-acceleration of time/resource-consuming modules or services is a current research concern in a variety of areas like: image and signal processing, communication networks, cloud computing and so on. One of the most interesting and recent approaches in this field regards the integration of reconfigurable hardware devices (FPGAs) in cloud computing datacentres for enabling hardware accelerated services [2]. This advances a new cloud service model: Hardware-Acceleration-as-a-Service (HAaaS) that offers clients "premium", high-performance services, that also have the potential of reducing overall costs by lowering the energy consumption (due to the underlying reconfigurable hardware).

In this context, the novelty of our proposed model is the provisioning of a generic reconfigurable architecture for the acceleration of virtually any type of web services that require computational resources.

Digital Object Identifier 10.4316/AECE.2016.02008

This is done by launching their execution in a synergetic compilation/deployment- into reconfigurable SoCs as IP cores, integrating a dynamic core management by leveraging partial reconfiguration.

There are a few other research papers on applying the SOA model to SoC architectures: [5] implements hardware services internally on a reconfigurable SoC platform – in other words, organizes the embedded system's internal architecture based on a service-oriented model. The proof of concept presented in [6] represents a hybrid model – with a hardware-accelerated backend and a web service-based interface, providing security services in mobile environments.

Other related implementations are described in [7] and [8], with the latter being an example of using XML-specified web services for remotely exposing reconfigurable hardware appliances.

Our proposed model brings an innovative, key feature: the partial reconfiguration of the hardware platform, which compared to other existing hardware-acceleration solutions, allows for a more dynamic approach to installing/removing services and functions during run-time. Also, this extends the possibilities of building FPGA-based self-reconfigurable systems [9], with the ability of the embedded system to re-configure a part of its own infrastructure.

Last but not least, our approach increases the flexibility and scalability of the hardware component, making it able to cope with the dynamic nature and availability requirements of web services (services can be often updated, modified, new services can be added easily depending on the request).

III. ARCHITECTURE AND IMPLEMENTATION

In our implementation (Figure 1) we have used a dedicated quad-core Intel i7 server for deploying the web application and web service, and a XC7Z020 AP SoC from the Xilinx Zynq-7000 family on an Avnet ZedBoard 7020, for offloading the web service's methods implementations. The Zynq SoC integrates a dual-core ARM Cortex-A9 MPCore based processing system (PS) and Xilinx programmable logic (PL), thus providing high performance and complete/partial re-configuration features.

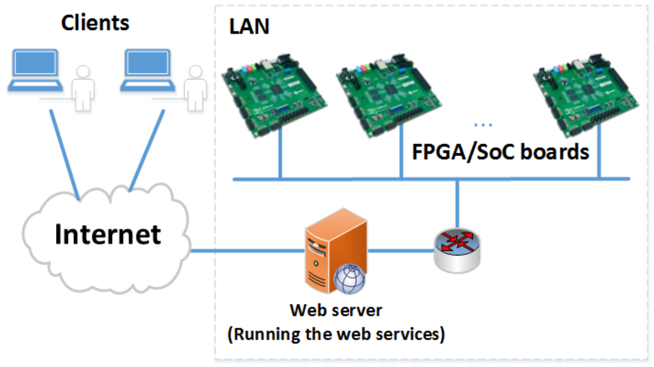

Figure 1. Overview of the implementation

The communication between the server (running the web service) and the SoC has been implemented using the dedicated on-board peripherals that the Zedboard provides: Gigabit Ethernet (used for the actual bitstream/data transfer) and UART (for debug and monitoring). The SoC is accessible over the Ethernet via TCP/IP and UDP protocols, having a unique IP address configured at start-up.

This enables several such devices to coexist, being network-connected and thus enabling a scalable approach on the hardware available for implementing the web services.

A. Web Service and Applications

The service-oriented component of our implementation consists of the HwAccelWS web service (a Java implementation running on a Glassfish 3.0.1 server instance) and a web application (integrating a JSP – JavaServer Pages interface and a Java servlet) used for exposing the functionality of the entire platform online, via an easy-to-use, web-based interface (Figure 2).

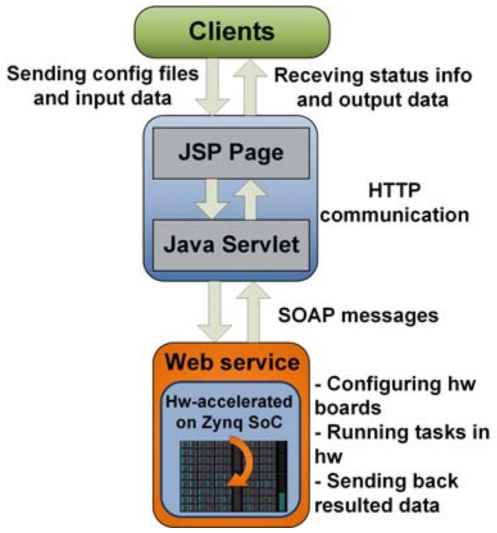
Clients
Sending config files and input data
Receving status info and output data
JSP Page
Java Servlet
HTTP communication
SOAP messages
Web service
Hw-accelerated on Zynq SoC
- Configuring hw boards
- Running tasks in hw
- Sending back resulted data

Figure 2. Web service and application flowchart

The user is offered a graphical interface – the JSP page – for accessing the desired web service method and sending/receiving data. The JSP technology was chosen for implementing the user interface since it can generate dynamic web content by combining HTML, XML and embedded Java code [10]. The Java servlet is used as an intermediary agent between the JSP page and the web service, processing HTTP requests and replies to/from the JSP page and managing the data flow to/from the web service using SOAP (Simple Object Access Protocol) messages [11]. The servlet and JSP page were thus used for providing an interface framework for the web service by encapsulating input data and operating with different Internet protocols.

The *HwAccelWS* web service is the centre of our platform, being implemented using Java and having several methods accessible either directly – via its URL – or by using the JSP interface that was developed especially for easing access and communication with the web service.

Transferring configuration files, offloading the input data, running the task on the Zynq SoC and returning the results to the web service are accomplished using the Ethernet-based communication interface with the SoC. The full and partial bitstreams are transferred via the UDP (User Datagram Protocol) – based TFTP (Trivial File Transfer Protocol) to the board, while the input/output data, control and status commands are sent/received using TCP (Transmission Control Protocol) communication. The networking capabilities on the web service side were implemented using Java UDP and TCP socket communication and embedded into the web service as dedicated methods.

B. Reconfigurable Hardware Platform

As mentioned above, the reconfigurable hardware platform is based on the Xilinx Zynq XC7Z020 SoC. The Programming System (PS) is running at 667MHz on one of the two ARM Cortex-A9 cores and it integrates a DDR controller for interfacing the on-board 512MB DDR3 memory chip. The following I/O peripherals are also configured within the PS: Ethernet (ENET0), UART (UART1), USB, and GPIO (General-Purpose I/O). The processing core, I/O, IP Cores and other system components are interconnected using an architecture based on the AXI (Advanced eXtensible Interface) protocol. Both interface and programmable logic (PL) are operating at 100MHz.

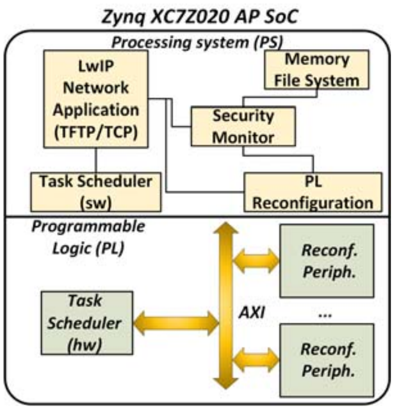

Figure 3. Zynq SoC internal architecture

The ARM PS runs a C application that consists of the following main parts (Figure 3): communication interface (TFTP and TCP servers), creating and handling a file system resident in the DDR3 memory, managing the complete/ partial reconfiguration of the PL, monitoring SoC operation and detecting security attacks using the on-board XADC.

The network communication capability of the SoC was implemented based on the on-board Gigabit Ethernet controller and the TCP/IP LwIP (Lightweight IP) Stack 1.4.0 [12]. The PS is running a TFTP Server – for transferring and storing full/partial bitstreams – and a TCP Server – for transferring data and communicating control and status commands with the web service. Both network applications are implemented in LwIP RAW mode API.

A 128MB file system (FS) has been implemented for storing and handling the configuration files (total and partial bitstreams) residing in the on-board DDR3 memory. The FS was implemented using the LibXil MFS (Memory FS) component, and is accessed by the C application running on the PS by means of specific function calls and libraries. This file system is an important part of the embedded system since it is acting as a local bitstream cache by allowing a dynamic management of the reconfigurable peripherals by means of storing several partial bitstreams and thus enabling a fast reconfiguration – with version-management and regression support.

The partial reconfiguration is the key component of our solution. Complete and partial reconfiguration of the embedded FPGA PL is performed through the device configuration (DevC) / Processor Configuration Access Port (PCAP) interface by the C application (the *config_PL()* method) running on the ARM PS. A complete reconfiguration of the FPGA is executed only at the initial set-up of the SoC, or when a critical update of the entire embedded architecture is required. Partial reconfiguration of the peripherals accelerating the web service's methods can be executed at any given moment during run-time when a new web service method requires installing a new peripheral for offloading its data and performing specific tasks.

When the *config_PL()* method is called, the following operations are executed:

- The *XDcfg* (Xilinx Device Configuration Interface) is initialized: PCAP is enabled and the control register is configured for the corresponding PCAP mode (complete or partial PL configuration).
- *DMA* and *PCAP Done* interrupts are cleared.
- The bitstream is transferred from the DDR3 memory to the PL fabric.
- *PCAP* and *AXI Done* interrupts are polled, the function call returns when the transfer is complete.

Consequently, the bitstream stored in the on-board DDR3 memory is transferred via the AXI-PCAP Bridge to the PL, the data being synchronized between the two interfaces using a FIFO buffer.

Since the reconfigurable logic is being modified while the rest of the FPGA device is operating, ensuring that the data from the module to be reconfigured is being ignored at its destination during partial reconfiguration (interface decoupling) is mandatory. This has been accomplished by sending a signal from the C application to the static logic inside the reconfigurable peripheral that determines the invalidation of the reconfigurable module's outputs.

Also, to ensure that after reconfiguration the RM is in a defined state, a reset after reconfiguration feature has been applied in the implementation stage of the design, by setting the RESET_AFTER_RECONFIG=TRUE property to the reconfigurable partition block.

In order to make things easier for the engineering of IP cores for this particular framework, the Zynq SoC is pre-partitioned, having a certain number of partial reconfigurable regions, of certain size and available resources, according to the applications and the nature and number of the services to be offloaded. Thus, the FPGA design engineers that would potentially design IP cores implementing in hardware different services are being given the exact available resources in terms of logic cells and bRAMs. In order to ensure a correct overlay of modules, they all have the same generic I/O ports, so the IP cores need to be constrained, using a top-level HDL wrapper, to this interface. The standard interface together with the resource availability information ensure a feasible IP core design, easing the instantiation of these cores in the system.

C. Securing the system

In this proposed framework the SoC device is situated at a remote location while operating with client data, and thus vulnerable especially to physical tampering security attacks. An attacker could attempt, by modifying the operating voltages and/or temperature of a SoC, to force the device to behave abnormally in order to extract data or bypass its security features [13].

Consequently, the Zynq SoC was chosen as the target development system for this service-oriented framework also because of its unique built-in security features [14]-[15]. We have implemented additional security features based on the Xilinx Anti-Tamper technology for detection and response to physical tampering events. This has been accomplished by using the on-chip hard IP block analog-to-digital converter (XADC) available on Zynq for monitoring the on-chip power supply voltage and die temperature.

The first step, at board power-up, was to configure the Zynq SoC generic interrupt controller in order to use the XADC interrupt as part of this interrupt service routine. Next, the XADC configuration has been initialized, the channel sequencer was stopped (set into safe mode) and all alarms have been disabled. Further on, readings of the temperature and voltage values are performed, being considered as normal operation reference values. Based on these readings, with the appropriate tolerance margins, the upper and lower temperature and voltage alarms are being assigned and the corresponding interrupts are enabled.

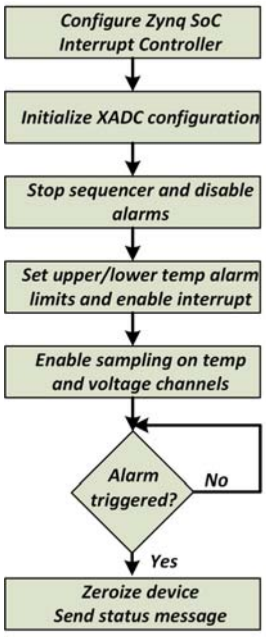
Configure Zynq SoC Interrupt Controller
Initialize XADC configuration
Stop sequencer and disable alarms
Set upper/lower temp alarm limits and enable interrupt
Enable sampling on temp and voltage channels
Alarm triggered?
No
Yes
Zeroize device Send status message

Figure 4. Security mechanism flowchart

In this operating mode, any temperature or voltage readings outside the established range will trigger the designated interrupt service routine. This routine clears the interrupt status register, disables other interrupts and reads the temperature/voltage values which triggered the alarm. Next, if these readings are consistent with security attack patterns, it takes reactive measures for preventing any potential unauthorized access to the data stored on the SoC by loading the partially reconfigurable modules with black-box versions and deleting the entire content of both the MFS containing configuration files and the user data memory space (storing input/output user data). Also, a TCP message is sent to the web service in order to signal that the board has detected a possible malfunction or tampering event and has taken the appropriate counter-measures.

Such prevention, detection and reaction measures increase the overall security of the SoC device, making it a trusted platform for secure computations [16]. Also, the design concept that makes use of partial reconfiguration adds an extra security level, enabling IP cores loading only when needed and unloading them in case of a security breach.

Last but not least, by this continuous monitoring of the operating parameters a pre-emptive approach is taken regarding the detection of natural occurring problems that might induce failures (e.g. a slowly drifting power supply).

D. Task Management

The tasks are created by the Web service after the client calls one of its methods using the JSP web page interface. Once created, they are sent over the network to the Zynq SoC to be executed on one of the existing processing cores (according to the task's type). The task management is performed on the Zynq device by a hybrid mechanism, involving a dedicated routine in the PS application and a scheduling peripheral in the reconfigurable logic.

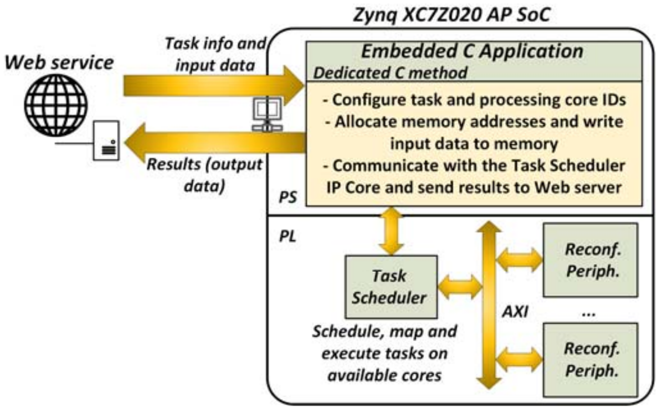
Zynq XC7Z020 AP SoC
Web service
Task info and input data
Results (output data)
Embedded C Application
Dedicated C method
- Configure task and processing core IDs
- Allocate memory addresses and write input data to memory
- Communicate with the Task Scheduler IP Core and send results to Web server
PS
PL
Task Scheduler
Schedule, map and execute tasks on available cores
AXI
Reconf. Periph.
...
Reconf. Periph.

Figure 5. Task management mechanism on the Zynq SoC

This mechanism has been designed and implemented considering the requirements of the present application: the tasks are independent processes, so no inter-task synchronization is needed, however the issue to be solved remains scheduling them and designating a processing core in order for each of them to be executed.

Every new task offloaded to be executed on the hardware platform is defined by a TCP message received from the web service. Following this message, a TCP data transmission transfers the corresponding input data. Both the new task message and input data are handled by a dedicated C method that performs task and data management. This method translates each new task to a task instruction that goes into an execution queue:

$$T_i = (ID_{Task}, ID_{Core}, Din_{Addr}, Din_{Length}, Dout_{Addr}, Dout_{Length})$$

Therefore, each task is assigned a unique task ID and the ID of target peripheral (IP core) on which it will be executed, together with two DDR3 memory spaces: for input (*Din*) and output (*Dout*) data.

The C task management routine transmits the task IDs and memory addresses to the *TaskScheduler* peripheral from the PL using software-accessible registers. This peripheral implements a FIFO-type scheduling of the tasks, enabling their parallel execution on the available IP cores. It launches the tasks by communicating via the AXI interface with the reconfigurable modules and controlling their operation. When each task is finished, the peripheral signals this event to the C application which is then responsible for sending the resulted data back to the web service. This processing flow is being shown in Figure 5.

This hybrid software/hardware approach for task management has been chosen as it offers better performance than a traditional software approach due to the improved execution times [17]. Consequently, the interface has been implemented in software - to maintain flexibility and ease of network communication with the web service - while tasks scheduling and execution have been implemented using a hardware peripheral, reducing time and energy consumption.

IV. Results and Discussion

The validation of our proposed system and methodology involved designing and implementing the embedded system based on the Zynq 7020 SoC on the Avnet Zedboard, implementing the web service and testing the communication and functionality of the setup.

For our proof-of-concept experimental setup, we have developed four web service methods. The first two (SHA_SW and AES_SW) implement SHA-2 (Secure Hash Algorithm-2, 256 bits) and AES (Advanced Encryption Standard) encryption as standard Java implementations running on the server machine, while the last two (SHA_HW and AES_HW) implement the same operations but the computations are offloaded to the Zynq SoC device. The software versions of the methods were implemented in order to benchmark the performances and correctness of the hardware accelerated methods' results.

The first step in implementing the hardware component was creating the ARM A9-based processing system using Xilinx EDK XPS (Embedded Development Kit Xilinx Platform Studio) 14.7. Using this software tool, the two IP cores were created: *reconf_periph_v1_00_a* (which was instantiated twice in the design and contains in the VHDL static logic the instantiation of the generic *pr_module*) and the above-mentioned *task_scheduler_v1_00_a*.

The two IP cores (SHA-2 and AES) were implemented as PR (partial reconfigurable) modules, being instantiated in the *reconf_periph_0* and *reconf_periph_1* peripherals. The individual synthesis of the two modules was done using ISE 14.7; this operation generated netlist files each having a standard wrapper over the specific top-level, with the role of standardizing the interfaces and module names to coincide with the ones of the generic *pr_module* (instantiated in the EDK peripherals).

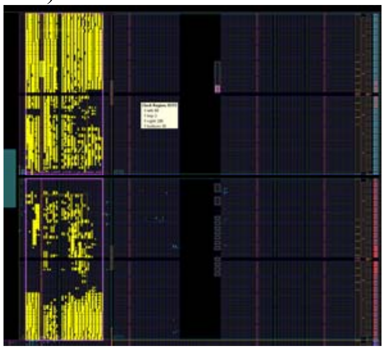
Figure 6. Layout of Zynq chip having defined two reconfigurable partitions with occupied slices highlighted for SHA (up) and AES (down) modules

The rest of the partial reconfiguration design flow: floorplanning, routing and generating the full and partial bitstreams was done with PlanAhead 14.7 [18]. The *pr_module* instances from the two peripherals were declared as reconfigurable partitions, for each of them a function module being added. The modules have been defined based on the two IP core netlists (ngc files) obtained after Xilinx ISE synthesis.

Besides the functional module, for each reconfigurable partition a non-functional black-box type version has been added, for allowing either unloading the peripheral when desired, or configuring the system without loading the peripherals at start-up. A view of the Zynq layout with the two partitions highlighted is shown in Figure 6.

Next, the implementation continued with the map, place&route, formal verification and bitstreams generation. A complete bitstream file was generated for configuring the entire chip, having black-box versions loaded for the reconfigurable partitions. Along with this file, another four partial bitstreams were generated, two for each module (one functional and one black-box

For testing the system, in order to assess the compliance with real-time requirements [19]-[20], the embedded application was deployed on the ARM PS while the PL was configured using the full bitstream with black-box versions of the PR modules. The web service remotely configured the two PR modules with partial bitstreams and the corresponding web service methods were accessed via the JSP page. Two tasks were offloaded to the device, together with two 1MB files of input data. The tasks were processed by the task management mechanisms running on the SoC and launched in parallel on the two IP cores. When each task finished, resulting data was sent back to the web service and checked for consistency with the software-computed results.

TABLE 1. PR REGIONS RESOURCES UTILIZATION

Resource type	**Available per region**	**Occupied**	
		AES-256	**SHA-256**
LUT	4000	648 (17%)	2050 (52%)
FD_LD	8000	717 (9%)	1167 (15%)
SLICEL	650	106 (17%)	334 (52%)
SLICEM	350	57 (17%)	180 (52%)
DSP48E1	20	0	0
FIFO18E1	10	5 (50%)	0
RAMB18E1	10	5 (50%)	0
RAMBFIFO36E1	10	0	0

The available and utilized resources for the PR regions with the two reconfigurable modules loaded are displayed in Table 1. The figures show that the two cores fit easily in the PR regions, AES occupying about 17% of the cells and 50% of the block RAMs, while SHA needing about 50% of the cells, without any RAMs. The PR regions can be chosen with different sizes, according to the needs of the cores to be used for accelerating the web services. For our setup, each PR region consists of less than 2% of the total number of LUTs and SLICEs available on the entire Zynq device.

TABLE 2. RECONFIGURATION AND OPERATION TIMINGS

Action	**Duration**	
	AES-256	**SHA-256**
Loading partial bitstream	2.48ms	
Data transfer Server – SoC	11.2ms	
Peripheral execution in PL	5.65ms	3.13ms
Peripheral execution in PS	41.15ms	20.2ms
Data transfer SoC – Server	12.4ms	
Total (PL with reconf.):	31.73ms	29.21ms
Total (PL without reconf.):	29.25ms	26.73ms
Total (SW on ARM PS):	64.75ms	43.8ms
Total (SW impl. on server):	73.25ms	47.38ms

The timings of the main actions during operation are shown in Table 2 while an overall performance comparison is presented in Figure 7. The size of the partial bitstreams used was 313kB, and each reconfiguration of a peripheral took 2.48ms. It took around 20ms for the input data and results to be transferred to and from the device, given the network conditions (Gigabit Ethernet, with the embedded C application implementing the TCP communication in RAW mode and achieving an average throughput of 90MB/s). The timing for the two peripherals was different: while the AES-256 core encrypted the input file in 5.65ms, it took 3.13ms for the SHA core to compute the hash value.

In order to better evaluate the hardware speed-up compared to software implementations, the comparison in Figure 7 contains the timings for the corresponding software methods running on the ARM PS. These results clearly show the speed-up brought by hardware execution, with the AES service being 2x faster and the SHA about 1.5x.

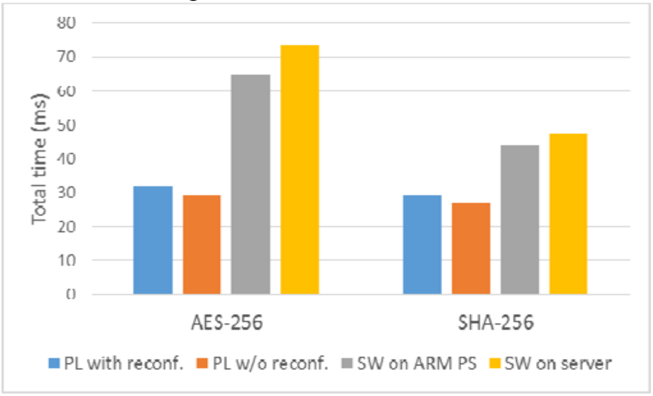

Figure 7. Performance comparison (including data transfer time to/from the web service).

V. Conclusions

This paper described a framework for accelerating web services by offloading their implementations on reconfigurable hardware devices. The framework provides a scalable, dynamic management of the tasks and hardware processing cores, based on dynamic partial reconfiguration. The hybrid approach to the task management: dedicated software method and integrated peripheral in the programmable logic allows for a *fast task scheduling and execution*, while maintaining a *flexible communication interface* with the web service.

The main contribution of our implementation lies in leveraging the partial reconfiguration feature of the latest generation Zynq SoC in order to achieve a *dynamic run-time management of the processing cores* (used for hardware-implementing the services), which is crucial given the existing requirements for web services in terms of response times and availability.

The experimental validation of our solution showed that the overhead induced by partially reconfiguring the peripherals to support different implementations during runtime for enabling the hardware acceleration of different services, is minimal (negligible compared to the overall timing during operation). It can also be noticed a faster execution time of the hardware-deployed tasks compared to their software equivalents.

Being designed to operate with client data while at a remote location (the service provider datacentre), the proposed framework *provides enhanced security features* for securing the user IP. Our approach offers two levels of protection: an advanced, dynamic detection of physical tampering attempts or hardware malfunctions, together with a mechanism for unloading the IP cores and deleting all sensitive data in the case of a security threat.

The *performance, scalability and flexibility* of the implementation extend the application range of this framework, allowing for its integration in distributed systems like cloud computing for accelerating computational intensive tasks, thus potentially increasing service execution speeds and decreasing energy consumption.

Also, the enhanced security allows the hardware implementation of offloading critical tasks, enabling higher security guarantees to be offered to clients by the service providers.

References

[1] A. Hayek, S. Domes, J. Borcsok. “Internet-controlled dynamic reconfiguration for FPGA-based embedded systems”, in *Proc. of 3rd Int. Conf. on Communications and Information Technology (ICCIT)*, Jun.19-21, 2013, Beirut, LB, pp.190-194. doi: 10.1109/ ICCITechnology.2013.6579547

[2] K. Mershad, A.R. Kaitoua, H. Artail, M.A.R. Saghir, H. Hajj, “A framework for multi-cloud cooperation with hardware reconfiguration support”, in *Proc. of IEEE 9th World Congress on Services (SERVICES)*, June 28-July 3, 2013, Santa Clara, CA, pp.52-59. doi: 10.1109/SERVICES.2013.12

[3] S. Pedre, T. Krajnik, E. Todorovich, P. Borensztejn, “A co-design methodology for processor-centric embedded systems with hardware acceleration using FPGA”, in *Proc. 8th Southern Conf. on Programmable Logic (SPL)*, Mar.20-23, 2012, Bento Goncalves, BR, pp.1-8. doi: 10.1109/SPL.2012.6211770

[4] T. Huffmire, B. Brotherton, T. Sherwood, R. Kastner, T. Levin, T.D. Nguyen, C. Irvine, “Managing Security in FPGA-Based Embedded Systems”, in IEEE Journal on Design & Test of Computers, vol.25, no.6, Nov.-Dec.2008, pp.590-598, doi: 10.1109/MDT.2008.166

[5] W. Chao, L. Xi, Z. Junneng, C. Peng, Z. Xuehai, “CaaS: Core as a service realizing hardware services on reconfigurable MPSoCS”, in *Proc. 22nd Int. Conf. on Field Programmable Logic and Applications (FPL),* Aug.29-31, 2012, Oslo, NO, pp.495-498. doi: 10.1109/FPL.2012.6339263

[6] A. Cilardo, L. Coppolino, A. Mazzeo, L. Romano, “Combining programmable hardware and web services technologies for delivering high-performance and interoperable security”, in *Proc. 15th Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP’07)*, Feb.7-9, 2007, Napoli, IT, pp.381-386, doi: 10.1109/PDP.2007.30

[7] J. Mao, Q. Xu, Z. Sun, X. Lu, “FSArch: Enables FPGA-Based platforms to provide network services”, in *Proc. 4th World Congress on Software Engineering (WCSE),* Dec.3-4, 2013, Hong Kong, CN, pp.241-245. doi: 10.1109/WCSE.2013.44

[8] D. Rodriguez, J.M. Sanchez, A. Duran. “Distributed reconfigurable computing using XML Web services”, in *Proc. IEEE Workshop on Signal Processing Systems Design and Implementation*, Nov.2-4, 2005, Athens, GR, pp.613-617. doi: 10.1109/SIPS.2005.1579939

[9] A. Melnyk, V. Melnyk, "Self-Configurable FPGA-Based Computer Systems," *Advances in Electrical and Computer Engineering*, vol.13, no.2, pp.33-38, 2013, doi:10.4316/AECE.2013.02005

[10] B. Perry, *Java Servlet & JSP Cookbook*, pp. 119-120, Sebastopol, CA: O’Reilly, 2004.

[11] D. Chappell, T. Jewell, *Java Web Services*, pp. 39-45, Sebastopol, CA: O’Reilly, 2002.

[12] A. Sarangi, S. MacMahon, U. Cherukupaly, “LightWeight IP application examples”, Application Note XAPP1026 v5.1, Xilinx Corp., San Jose, CA, 2014.

[13] M. Schramm, A. Grzemba, “Reconfigurable trust for embedded computing platforms”, in *Proc. IEEE Int.Conf. on Applied Electronics (AE)*, Sep.5-7, 2012, Pilsen, CZ, pp.261-264.

[14] S. McNeil, “Solving today's design security concerns”, White Paper WP365, Xilinx Corp., San Jose, CA, 2012.

[15] J. D. Corbett, “The Xilinx Isolation design flow for fault-tolerant systems”, White Paper WP412, Xilinx Corp., San Jose, CA, 2012.

[16] S.M. Trimberger, J.J. Moore, “FPGA Security: Motivations, features and applications”, *Proceedings of the IEEE*, vol.102, no.8, pp.1248-1265, 2014. doi: 10.1109/JPROC.2014.2331672

[17] R. Pellizzoni, M. Caccamo. “Real-Time Management of Hardware and Software Tasks for FPGA-based Embedded Systems”, in *IEEE Trans. Computers*, vol.56, no.12, pp.1666-1680, 2007. doi: 10.1109/TC.2007.70763

[18] D. Dye, “Partial reconfiguration of Xilinx FPGAs using ISE Design Suite”, White Paper WP374, Xilinx Corp., San Jose, CA, 2012.

[19] Z. Zheng, Y. Zhang, M.R Lyu, “Investigating QoS of Real-World Web Services”, in *IEEE Trans. Services Computing,* vol.7, no.1, pp.32-39, 2014. doi: 10.1109/TSC.2012.34

[20] D. McKee, D. Webster, J. Xu, "Enabling Decision Support for the Delivery of Real-Time Services", in *Proc. of 16th IEEE Int. Symposium on High Assurance Systems Engineering (HASE),* Jan.8-10, 2015, Daytona Beach, FL, pp.60-67. doi: 10.1109/HASE.2015.18