A Dynamically Reconfigurable Platform for High-Performance and Low-Power On-Board Processing

Andrea Guerrieri
EPFL, Processor Architecture Laboratory
Lausanne, Switzerland
andrea.guerrieri@epfl.ch

Sahand Kashani-Akhavan
EPFL, Processor Architecture Laboratory
Lausanne, Switzerland
sahand.kashani-akhavan@epfl.ch

Pasquale Lombardi
Syderal SA
Gals, Switzerland
pasquale.lombardi@syderal.ch

Bilel Belhadj
Syderal SA
Gals, Switzerland
bilel.beladj@syderal.ch

Paolo Ienne
EPFL, Processor Architecture Laboratory
Lausanne, Switzerland
paolo.ienne@epfl.ch

***Abstract*—FPGAs (Field Programmable Gate Array) are an attractive technology for high-speed data processing in space missions due to their unbeatable flexibility and best performance-to-power ratio in comparison to software. However FPGAs suffer from 3 major drawbacks: (1) higher programming effort is required with respect to software; (2) hardware resources need to be allocated for each implemented function in contrast to software functions which can be executed on the same processing hardware; and (3) FPGAs are required to adopt radiation hardening techniques when deployed in a space environment. This paper presents a reconfigurable platform that demonstrates how modern FPGAs can be considered as computing resources like any other, suitable for emerging spatial applications and not subjected to the above-mentioned drawbacks. In particular, we show that large FPGAs can be split in different regions containing concurrently-running accelerators which can support the execution of a single or multiple applications. Then, in the same way as software-based multiprogrammed and multithreaded systems can dynamically create, schedule and execute threads, FPGA-based accelerators can be swapped in and out according to scheduling needs by exploiting their dynamic partial reconfiguration capability. A proof of concept cloud detection algorithm for Sentinel-2 multispectral images has been implemented and tested on our platform to validate the system's design principles and performance.**

***Index Terms*—SoC, FPGA, COTS, dynamic partial reconfiguration, software thread, hardware accelerator.**

I. Introduction

European satellite projects today rely on Leon-based cores as the most advanced space-qualified general-purpose processor. The performance of such cores, however, does not match the growing computational needs of space missions, particularly for data processing applications. When higher computational power is needed, one therefore needs to either select a COTS or a rad-hard processor/DSP mostly from US companies (which are more likely to be subjected to export control regulations). The following section discusses the motivations for developing a European technology providing processing power, energy efficiency and performance scalability, as needed by the computational needs of future space missions.

A. Motivation

The necessity for a high-performance European-qualified DSP for payload data processing has been formalized by ESA since October 2007 during a round table discussion with industry [1]. This round table brought the development of a multi-core DSP platform based on a Leon3-FT CPU and two Xentium DSPs [2] (known as the Scalable Sensor Data Processor SSDP) to an ESA roadmap. Although SSDP's development is a significant step forward for its processing performance with respect to previous European space CPUs, it is still not adequate for most demanding space applications, e.g. vision-based navigation as assessed by Lentaris et al. [3].

Yet, a recent report prepared by the DLR [4] (the German Aerospace Center) about on-board computing processing power needed for future missions states that *in the upcoming years, the demand of on-board computing power for spacecraft is expected to grow steadily. This plus in computing performance is requested due to the increasing amount of payload data, which needs to be processed. A special focus resides on sub-domains like earth observation and space robotics.* The same report identifies two main applications requiring significantly more processing power than what is available in the current state of the art in space technologies. The first is autonomous navigation where a large amount of imaging sensor data needs to be processed in order to react to the environment. The second consists in on-board pre-processing of acquired sensor data in order to reduce its volume and overcome limited satellite download bandwidth.

978-1-5386-7753-7/18/$31.00 © 2018 IEEE

A cloud detection application falls in this second category and allows reducing data volume by excluding clouds in Earth Observation images prior to downloading. Furthermore, a NASA assessment (as a result of a series of workshops held among scientists and engineers of JSC, GSFS and JPL in 2012) about space flight computing performance reported that *a high-performance spaceflight computer will indeed be game changing because the capability is needed for many planned space missions across the agency, and will enable new and dramatic mission applications that are strongly desired by advanced mission planners.* [5]

An emerging space application needing high-performance computing is payload instruments for small satellites such as CubeSats. Small satellites contain complex payload instruments which generate increasingly large data volumes, but their download bandwidth and power capabilities remain limited due to their small size. FPGAs (Field Programmable Gate Array) are an attractive technology for high-speed data processing in space missions due to their unbeatable flexibility and best performance-to-power ratio in comparison to software. However FPGAs suffer from three major drawbacks. First, higher programming effort is required with respect to software. Second, hardware resources need to be allocated for each implemented function in contrast to software functions which can be executed on the same processing hardware. Finally, FPGAs need to adopt radiation hardening techniques if deployed in a space environment. In response to the high-performance computational needs introduced above, the dynamically reconfigurable computing platform described in this paper shows to be a promising solution not only for its computational performance, but also for its energy efficiency, optimized use of silicon and, particularly, for its potential flexibility in scaling performance and adapting to different missions or future growing needs.

B. Paper Organization

This paper is organized as follows: Section II provides an overview of our reconfigurable platform and why it targets heterogeneous SoCs; Section III shows how the system is internally constructed; Section IV introduces the cloud detection algorithm for multispectral images and its hardware implementation; Section V provides an evaluation of our platform; finally, Section VI concludes.

II. Dynamically Reconfigurable Platform for High-Performance and Low-Power On-Board Processing

A. High Level Overview

The main idea behind our system is to create a reconfigurable platform capable of merging the convenient programming paradigm followed by software with the unbeatable flexibility and performance-to-power ratio of FPGAs. In particular, we would like to allow *multiple* traditional software applications running on a CPU to offload computational tasks to FPGA-based hardware accelerators running on a *single* FPGA. From a high-level perspective, the main actors in our system are (1) a Host Machine (HM) on which application code is executed, (2) an FPGA, (3) an external memory, and (4) an Execution Controller (EC) which abstracts control of the FPGA by the Host Machine.

B. Novelty of our system architecture

Naturally, one would wonder about the benefits of our proposed architecture. After all, the idea of using FPGAs as an acceleration platform is not new and the litterature contains numerous papers where platforms similar to ours exist. The main limitation of existing platforms originates from the seemingly unharmful design decision of using a softcore processor embedded within the FPGA as their runtime manager (as presented in some recent work [6]). This design choice limits platforms flexibility since it prevents the runtime manager from performing a *full* reconfiguration of the FPGA without destroying its own software state. The resuling systems are therefore restricted to using configurations where the number of accelerators present in the FPGA and their allocated resources are decided at compile-time and cannot be changed without rebooting the system. One of the primary functions of any platform is the management of system resources, so it is essential the software stack performing this function be non-volatile. In order to support full *and* partial FPGA reconfigurability, we propose to move the runtime manager *outside* the FPGA by using a traditional *hard* processor instead. FPGA platforms which include hard processors exist since a while; however, a promising new family of *SoC-FPGAs* have emerged since 2011 which incorporate hard multi-core ARM processors with a tightly-coupled FPGA fabric in a single chip, such as Xilinx's *Zynq*, Intel's *Cyclone V SoC* and Microsemi's *SmartFusion*. NanoXplore is also expected to ship similar devices such as the *NG-LARGE* and the *NG-ULTRA* [7]. In particular, the processors in these devices have dedicated hard peripherals for managing the FPGA. For example, Intel's *Cyclone V SoC* family of devices have a dedicated *FPGA Manager* peripheral and Xilinx's *Zynq* devices have a *Processor Configuration Access Port* (PCAP) for performing FPGA reconfiguration.

C. Features Relevant to Space Applications

We designed our architecture on Xilinx's *Zynq* family of devices, a COTS component based on the "All Programmable" SoC architecture [8]. Several radiation tests have been done on this device and the results show that it is an attractive solution for non-critical missions with shorter lifetime and less quality constraints [9], [10]. The *Zynq-7045* integrates a feature-rich dualcore ARM Cortex A9-based processing system (PS) and a 28nm FPGA (programmable logic, PL) in a single chip. The device supports Dynamic Partial Reconfiguration (DPR) which allows it to dynamically change the configuration of internal FPGA regions while the rest of the chip continues to operate uninterrupted [11]. The main idea behind our system is to place the main software processor (Host Machine) as well as the unit in charge of the reconfiguration process (Execution Controller) in the hard-wired programmable part of the chip.

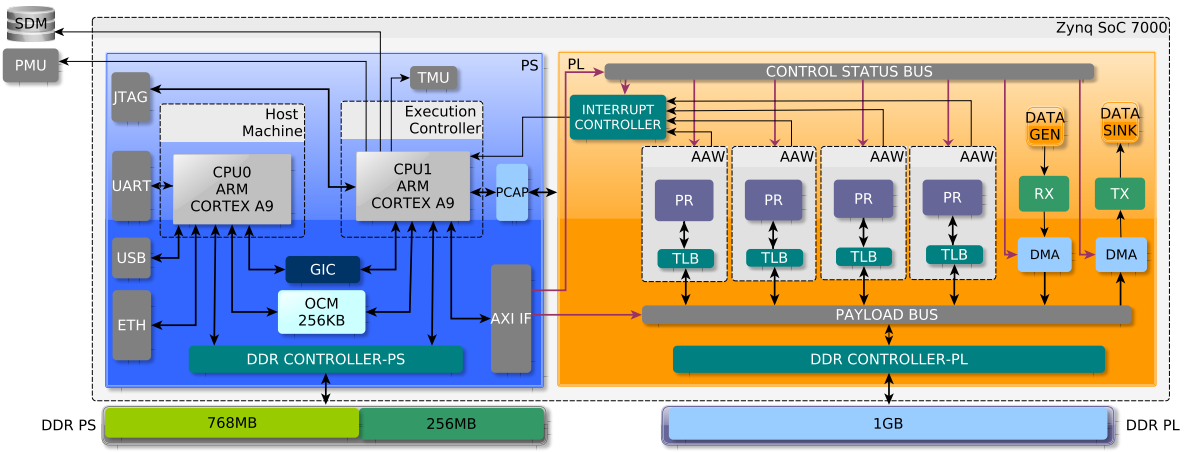

Fig. 1. Block Diagram of the Reconfigurable Platform

Such a design frees the Programmable Logic from being involved in system-level control decisions regarding its own configuration and can therefore be reserved for computational purposes. The Host Machine and Execution Controller continue to operate during full and partial FPGA reconfiguration since the FPGA reconfiguration controller is part of Processing System (instead of the Programmable Logic) and is therefore not affected by the reconfiguration process. This enables the system to automatically perform Single Event Effect (SEE) mitigation techniques [11] presented in some recent work [12] such as partial and full FPGA configuration scrubbing. Furthermore, this allows the platform to change the size and the number of concurrent hardware accelerators allocated at runtime. This dynamic resource context switching allows the system to adapt itself to the computational needs of applications on-the-fly.

III. System Architecture

Although our system was developed on a Xilinx *Zynq*-based device, our architecture has been designed to be portable to platforms from multiple vendors and can scale to support multiple applications. An application's main code runs on the HM which, in turn, runs an embedded Linux operating system and therefore natively supports multi-task management. The application code initiates, allocates/deallocates, and controls tasks intended to run on the FPGA as accelerators at runtime through the use of *partial bitstreams* which are dynamically loaded into Partial Rconfigurable Regions (PRRs) on the FPGA. The internal block diagram of our architecture is shown in Figure 1.

A. Processing System

In a *traditional* execution model, the two CPUs of PS would be used and managed by the operating system (Linux). However this method cannot guarantee to meet deadlines present in a real-time environment since the Linux kernel is preemptive and not hard real-time ready. Instead, our system uses the two CPUs as independent machines through the use of *Asymmetric Multiprocessing* where CPU0 is the HM (which runs embedded Linux & applications), whereas CPU1 is exclusively used to run a bare-metal application capable of satisfying hard real-time deadlines (an Execution Controller, EC). This design choice enables us to use the flexibility of Linux's existing multi-threading infrastructure, while at the same time guaranteeing real-time constraints using a bare-metal application. Furthermore, this separation allows to easily export the HM off-chip to interface multiple FPGA simultaneously.

Host Machine (HM): The HM runs embedded Linux and its main function is to execute user application code as any traditional embedded processor would. In addition to user application code, the HM exposes a set of predefined APIs which allow applications to manage FPGA resources without requiring that a user have any hardware skills or knowledge about the platform's internal architecture. The design of the FPGA management APIs were inspired by the programming paradigm used to manage POSIX threads which allows application designers to manage hardware threads (accelerators) in a similar way as they would for software threads, i.e. with complete abstraction about how the FPGA works [13]. In the same way as software threads are instantiated, executed and synchronized in software applications, APIs have been implemented in our platform allowing a host program to instantiate, execute and synchronize hardware threads with applications that can collect and use their results. The APIs collectively form the interface between the HM, the EC, and the FPGA. Table I lists the APIs and Figure 2 shows their execution flow.

Execution Controller (EC): The EC is a single-threaded software control loop which executes a set of predefined operations that manage the FPGA subsystem. It is internally architected as a finite state machine and can manage the FPGA in real-time. The EC follows directives provided by the HM while simultaneosuly guaranteeing the safe functioning of the system. Safety is achieved by adding each asynchronous request coming from the HM to an execution

Method	Description
`swift_initialize(uint32_t threads)`	Sets a new static design on the FPGA
`swift_create(uint32_t threads, HWTHREAD_T* thread, ...)`	Creates a new hardware accelerator instance
`swift_cancel(uint32_t threads, HWTHREAD_T* thread, ... )`	Cancel an existing hardware accelerator instance
`swift_join(uint32_t threads, HWTHREAD_T *thread, ...)`	Wait until the specified hardware accelerator has terminated
`swift_set_sched_param(uint32_t param)`	Set scheduling parameters
`swift_malloc(HWTHREAD_T* thread, uint32_t size)`	Allocate a virtual memory space for the accelerator
`swift_free(HWTHREAD_T* thread)`	Free the allocated virtual memory for the accelerator
`swift_mutex(HWTHREAD_T* thread)`	Mutual exclusion for the accelerator

TABLE I
LIST OF APIS

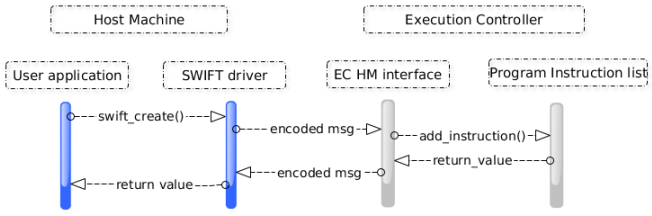

Fig. 2. Host Machine & Execution Controller Execution Flow

list, and consequently processing the list in a synchronous way. This approach also allows one to compile a sequence of instructions for the EC offline and to send it at a later time to prevent data loss in case of a communication fault between the HM and the EC (especially if the HM is off-chip). The EC loads the FPGA bitstreams (partial and full) from the external memory and programs the PL using the PCAP controller. The software running on the HM is completely abstracted from the structure of the FPGA as the EC is responsible for controlling its resources. To manage the assignement of a hardware accelerator to the PRRs, the EC uses various non-preemptivie scheduling algorithms such as First-Come-First-Served, Priority-Scheduling, Shorter-Job-First, and Dependency-Scheduling. The scheduling algorithm can be selected by the user through the exposed APIs and can also be updated at runtime. Due to the dynamic nature of FPGA resource utilization, the platform could encounter periods where power is overconsumed with respect to the thermal budget available in space applications. The EC uses a Power Monitoring Unit (PMU) and a Temperature Monitoring Unit (TMU) to handle this situation by adapting its scheduling algorithm in order to keep power and temperature under a certain predefined threshold.

Inter-Machine Communication: The two processors in the PS do not share the same address space since they run independently in our asymmetric multiprocessing environment. Therefore, communcation between the processors in done by means of an internal on-chip memory (OCM) which is part of the PS. A custom Inter-Machine Communication Protocol (IMCP) was been developed to encapsulate the messages between the two machines. The IMCP is an asynchronous, bidirectional, hybrid master-slave type protocol: depending on the transaction source, both the HM an EC could be the master. The protocol supports CRCs, ACKs, ARQ, TSN, and piggybacking. Furthermore, the protocol includes support for FPGA addressing in order to support control for multiple FPGAs in a larger system. In particular, the ICMP has been designed to be ported over other communication mediums such as SpaceWire, SpaceFibre or TCP/IP.

B. Programmable Logic

The PL structure has been specifically designed to host multiple independent hardware accelerators (HA) with different sizes and functionalities. The PL design can be divided in two parts: the *static* design and the *dynamic* design. The static design represents all the infrastructure needed to host the HAs and support HA $\leftrightarrow$ EC and HA $\leftrightarrow$ external memory communication. The SoC includes several AXI4-based interconnects to allow communication between the PS and the PL and all communication between the CPU and the HAs is done through these interconnects. However, the HAs would achieve very low performance if they were to ask the host CPU to perform memory accesses for them as a middleman and communicate the result to them through a simple control/status bus. To get around this issue, we decouple the static design's communication infrastructure into two independent interconnects: the control/status bus and the payload bus so the HAs can have a high-bandwith DMA pathway to the external memory. To allow the EC to control the events coming from the HAs, we introduce a generic interrupt controller managed exclusively by the EC and whose goal is to intercept requests directed to the processor from the HAs. An example of such a request is the termination of a HA or a hardware error. The interrupt controller is connected to the control/status bus and the number of HAs it supports is related to the number of HA slots present in the static design. Finally, an AXI Accelerator Wrapper (AAW) surrounds each accelerator and allows software to dynamically configure a given application's reconfigurable logic. The module loaded into the reconfigurable region represent the dynamic design.

AXI Accelerator Wrapper (AAW): The AAW is a key element for platform functionality providing a standard interface to the custom accelerator, accelerator isolation during partial reconfiguration, and the accelerator's memory virtualization. Figure 3 shows the architecture of the AAW. It is it composed of (1) a Wrapper Control Unit (WCU), (2) a PR Region (Accelerator slot), (3) isolators, and (4) a Translation Lookaside

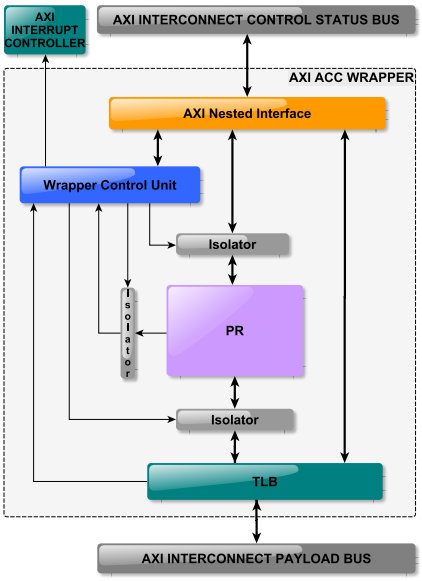

Fig. 3. AXI Accelerator Wrapper

Buffer (TLB). The WCU provides a set of predefined registers for controlling the subsystem's components. An accelerator kernel can be loaded dynamically into each PRR depending on user/system requests. Isolators provide a safe interface between the static design and dynamic design during partial reconfiguration. Finally, a TLB is added to each AAW to enable memory protection and virtualization. If disabled, the TLB will transparently allow direct access to memory.

IV. Cloud Detection Algorithm for Multispectral Image

A. Sentinel-2 Multi-Spectral Images

The Sentinel-2 satellite mission will monitor variability in land surface conditions and support monitoring changes in vegetation during the growing season [14]. The Sentinel-2 Multi-Spectral Instrument (MSI) samples 13 spectral bands to build a multi-spectral image. Each image covers 100×100 km^2 of earth surface and has a size of 1830×1830 pixels. We use Sentinel-2 MSI Level-1C data available in the Copernicus Open Access Hub [15]. Level-1C provides per-pixel radiometric measurements and parameters to transform them into radiances. Level-1C products are resampled with a constant Ground Sampling Distance (GSD) of 10m, 20m, and 60m. At this level, algorithms like scene classification and atmospheric corrections may be applied to images to provide the next product level, Level-2A. Scene classification usually includes cloud detection algorithms which we select as an application example for the SWIFT platform.

B. On-board Processing of Cloud Detection Algorithm

On average, 66% of image pixels are contaminated with clouds and users interested in monitoring land surface conditions like desertification or water cover such cloudy pixels as useless information [16]. On-board cloud detection would prevent to store and downlink meaningless data; cloudy pixels may be detected and enumerated for every image. Therefore, an on-board decision may be taken to whether keep or discard a pixel. Using this technique, it becomes possible to save on-board storage resources and downlink capacities.

C. Cloud Detection Algorithm Description

The algorithm aims to find clouds in multispectral images by applying a sequence of threshold comparisons on reflectance data. This approach assigns pixels to pre-defined classes "Clouds" or "Not-Clouds" [17]. Potential cloudy pixels are identified by a first filtering in the red region of the solar spectrum. Then all these potentially cloudy pixels undergo a sequence of filtering based on spectral band thresholds, ratios, and index computations. The result of each pixel test is a cloud probability ranging from 0 (for high confidence of clear sky) to 1 (for high confidence of cloud). After each step the cloud probability of a potentially cloudy pixel is updated by multiplying the current pixel cloud probability by the result of the test. Finally, the cloud probability of a pixel is the product of all the individual test results [17]. The sequential filtering of a potentially cloudy pixel stops when a test result returns a cloud probability equal to zero. The pixel is then considered to be high confidence clear sky in the cloud probability map. A clear sky pixel may be further classified as bare soil, water, vegetation, rocks, dark features or snow. The latter presents almost similar thermal and reflectance properties as the cloud (they are both cold and bright). Advanced and refined filtering is therefore used to unravel snow and cloud reflectance properties. However, this filtering is not required on images of earth regions where snowfall never happens. Figure 4 shows the block diagram of the cloud detection algorithm. The algorithm's filtering sequence can be divided into two parts: the basic cloud detection sequence (CDS) and the snow detection sequence (SDS). The SDS is enabled if and only if the image contains snowy pixels. The SDS sequence may not be instantiated into the hardware thread that will detect clouds in snow-free regions to further save FPGA resources and power consumption. However, the snow sequence can be added whenever the image represents a snowy region thanks to the platform's dynamic reconfiguration. This can be simply done by swapping hardware threads implementing CDS and CDS+SDS sequences using the platform's APIs. The selected hardware thread matches as much as possible the current processing need of the cloud detection algorithm and, thus, optimizes hardware resources utilization and global algorithm runtime and power consumption. However, this flexibility comes at a price; swapping hardware threads induces a reconfiguration time overhead. Reconfiguration time is constant at each sequence swapping operation and increases proportionally to the swapping frequency. In our case, swapping between hardware threads is not frequent since the satellite processes many images before crossing snowy and snow-free region borders.

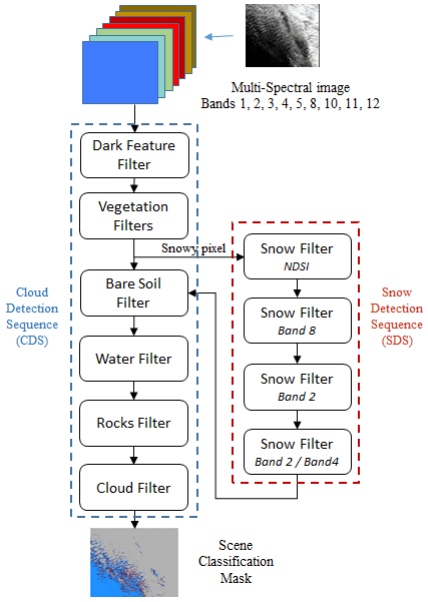

Fig. 4. Cloud Detection Algorithm steps. Algorithm input: multi-spectral images. Algorithm output: scene classification mask.

D. Hardware Implementation

We have implemented two hardware threads to map the cloud detection algorithm onto the FPGA: one hardware thread implements the CDS sequence and a second thread implements both the CDS and SDS sequences. The design is directly interfaced with the AXI4-lite bus for control and configuration, and AXI4-full bus for data transfer. The data bus size is configurable and ranges from 32 bits to 512 bits allowing up to 2 pixel transfers per communication cycle. The algorithm's steps are implemented in a processing element (PE). The PE module may be instantiated as many times as the PR region size allows it. Having a configurable number of PEs in the system gives more flexibility to fit the application's processing needs and the PR size. In the same time, it allows to reach the best tradeoff between data transfer throughput and processing power. High data transfer throughput and low processing power leads to wasting communication bandwidth. In the opposite situation, high processing power and low data transfer throughput leads to communication bottlenecks and unused computation resources. Synchronization among communicating PEs inside a hardware thread and among hardware threads is assured using shared memory accesses. Figure 5 summarizes the design modules used to implement the cloud detection algorithm.

V. Evaluation and Tests

A. Experimental Setup

The design has been implemented on Xilinx's ZC706 development board. The cloud detection algorithm has been implemented in C, cross-compiled for the ARM Cortex A9, and executed on the HM. Then, we modified the software-only version by adding calls to our APIs to move the computation from the HM to the FPGA. The PL design runs at

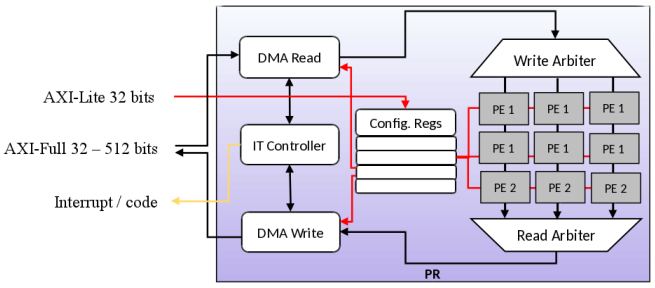

Fig. 5. System view of cloud detection hardware implementation. White boxes represent reusable modules. Gray boxes represent custom modules.

		Xilinx XC7Z045-FFG900	
	PRRs	Slices [#]	Slices [%]
Control / Status Interconnect	1	179	0.06
	2	247	0.11
	4	494	0.22
Interrupt Controller	1	78	0.04
	2	78	0.04
	4	78	0.04
Accelerator Wrappers	1	1702	0.67
	2	3404	1.34
	4	6808	2.68
Payload Interconnect	1	272	0.08
	2	514	0.24
	4	1028	0.48
Total	1	2231	0.85
	2	4243	1.73
	4	8408	3.42

TABLE II
Resource utilization of the static design

200 MHz using a single clock domain. Table II and Table III respectively report the resources used for the static design and the dynamic design. We experimented different hardware-accelerated versions of the algorithm with different data bus widths to the external memory (32, 64, and 128 bits) and varying number of concurrent accelerators (1, 2, and 4). For example, `HW-32-2` corresponds to a platform with a 32-bit data bus and 2 concurrent hardware accelerators. During the execution of the tests, power consumption and temperature were monitored from the EC to collect data about energy consumption.

	Xilinx XC7Z045-FFG900				
	Slices LUTs	Slices Registers	Muxes	Slice	DSP
Acc. type 1	5524	5575	145	2197	4
Acc. type 2	6441	6087	344	2395	4
PE type1	750	959	17	364	1
PE type2	990	1087	66	417	1

TABLE III
Resource utilization of the accelerators. Type 1 units only implement the CDS, whereas type 2 units implement the CDS+SDS sequence.

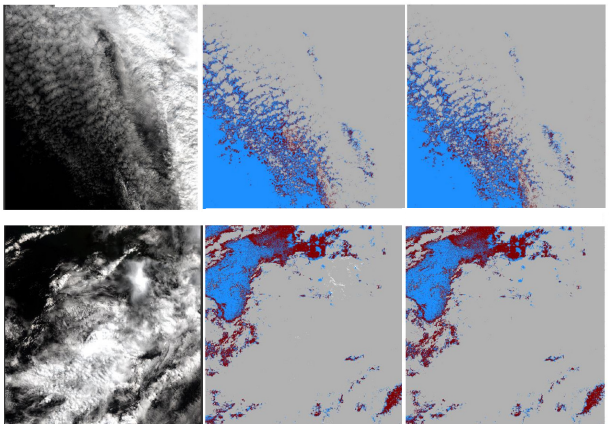

Fig. 6. Image Processing Results. From left to right: satellite multispectral image (RGB); Classification Mask SW output w/o FPGA threading; with FPGA threading.

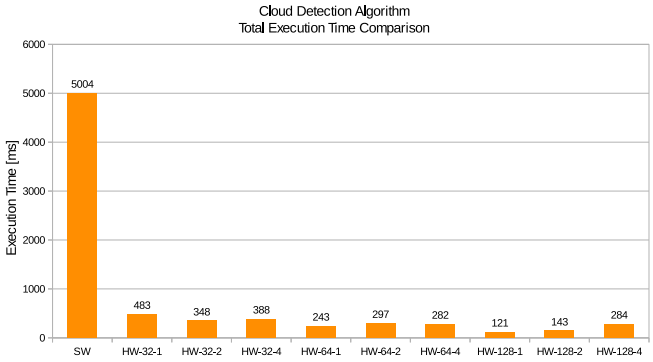

Fig. 7. Total Execution Time for Various Hardware Configurations

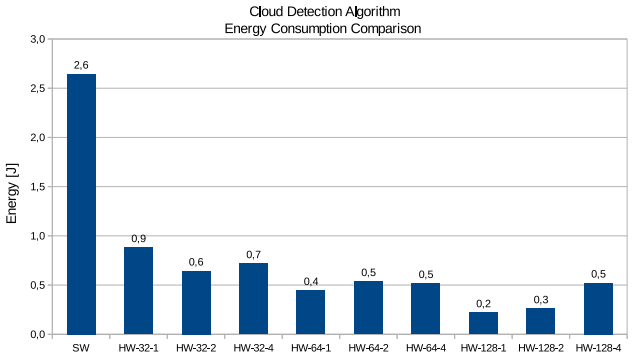

Fig. 8. Energy Consumption for Various Hardware Configurations

B. Experimental Results

Image Processing: To test our reconfigurable platform, we processed different multi-spectral images according to the algorithm described in Section IV. For each image, we compared the output of the reference software model to the output computed by our platform. Figure 6 shows the RGB version of an image followed by the classification mask images including the one generated by the reference implementation and a second one generated by hardware accelerators.

Performance: Figure 7 compares the execution time of the software-based algorithm to the various hardware-accelerated versions used in our reconfigurable platform. The execution time and energy consumption for executing the entire algorithm on the HM (software-only ARM) are of 5004ms and 2.6J, and 1.8s on an Intel Core i7 (i7-5950HQ 3.8 GHz), but with a power consumption not easily estimable albeit reasonably higher. Execution time and energy consumption with FPGA acceleration where the host creates and executes hardware threads on the FPGA to perform image processing tasks are reported in Table IV. Below we report and describe in detail the results for the 128-bit data bus as an example. All numbers includes the overhead due to reconfiguration time and external memory accesses.

One hardware thread processing the whole image: 121ms (2ms HM-EC communication time, 63ms reconfiguration time and 56ms execution time); 221mJ.

Two concurrent hardware threads processing 1/2 image each: 143ms (2ms HM-EC communication time, 126ms reconfiguration time and 15ms execution time); 260mJ.

Four concurrent hardware threads processing 1/4 image each: 284 ms (2ms HM-EC communication time, 252ms reconfiguration time and 30ms execution time); 522mJ.

The best case is obtained with 1 concurrent hardware thread with an overall execution time $40\times$ shorter than the non-accelerated version and energy consumption almost $12\times$ less. To interpret the results, it should be noted that data transfer from/to the external memory takes a large portion of the execution time. This is due to (1) the quantity of data in the multi-spectral images ($1830 \times 1830 \times 13$ pixels), (2) a bus with of 128 bits and (3) sharing of that bus by all accelerators. This causes the execution time not to decrease proportionally to the number of accelerators. Moreover, the overhead of configuring multiple hardware threads also has its influence as it can be seen by observing the execution time with two and four hardware threads being larger than with one hardware thread (see Figure 9). However, in the real application, the reconfiguration time may be considered amortized since the needs for reconfiguration is not for every execution cycle as explained in Section IV, reducing significantly its absolute value. Further improvement of the execution time is possible by using a larger external memory access bus width (up to 512 bits) as it is foreseen by the design and increasing the number of internal PEs.

VI. Conclusion

This paper presented a dynamically reconfigurable platform for high-performance and low-power on-board processing. A functional demonstrator has been developed and implemented on Xilinx Zynq COTS SoC. The architecture has been designed to be portable to platforms from multiple vendors: portability and scalability are fundamental principles of the platform. A proof of concept cloud detection algorithm for Sentinel-2 multispectral images has been implemented and

Execution Model	Tot. Exec. Time [ms]	Comm. Time [ms]	Reconf. Time [ms]	HW Exec. Time [ms]	Power Avg [W]	Energy [J]	Speedup [x]	Energy Efficiency [x]
SW	5004	2	N/A	N/A	0.53	2.6		
HW-32-1	483	2	67	414	1.82	0.9	10	3
HW-32-2	348	2	134	212	1.85	0.6	14	4
HW-32-4	388	2	268	118	1.85	0.7	13	4
HW-64-1	243	2	65	176	1.83	0.4	21	6
HW-64-2	297	2	130	165	1.80	0.5	17	5
HW-64-4	282	2	260	20	1.83	0.5	18	5
HW-128-1	121	2	63	56	1.82	0.2	41	12
HW-128-2	143	2	126	15	1.82	0.3	35	10
HW-128-4	284	2	252	30	1.83	0.5	18	5

TABLE IV
PLATFORM PERFORMANCE FOR VARIOUS HARDWARE CONFIGURATIONS

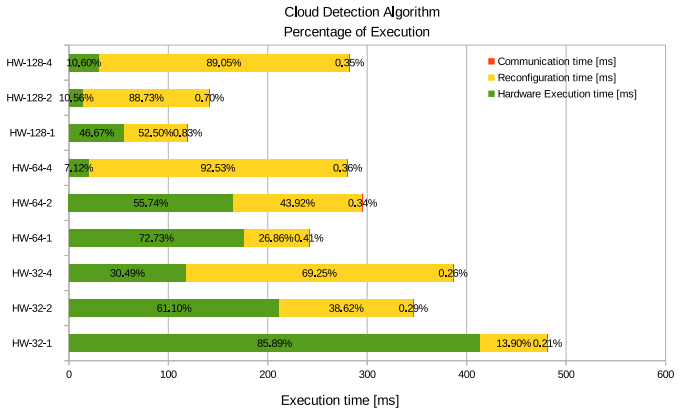

Fig. 9. Execution Time Breakdown for Various Hardware Configurations

tested on our exerimental platform to validate the system's design principles and performance. We showed how our platform can achieve up to $40\times$ speedup over the equivalent software version and reduces energy consumption by up to $12\times$, using a programming model which requires no more effort than software threading. The demonstrator's implementation proved that FPGA resources can be managed by software and effectively used for computational tasks in the same way as a central processing unit can be used by an operating system to execute user applications. In particular, the FPGA can be initialized for a number of dynamically-configurable partitions and make each of them available for running concurrent hardware threads as needed. Additionally, synchronization mechanisms have been implemented and tested to coordinate the execution of the instantiated hardware threads with respect to the main software flow. Finally, the scheduling and allocation of hardware threads to the available FPGA resources is transparently managed in the background and can be configured through software.

We believe this platform is well-suited for space mission deployment in future satellites.

REFERENCES

[1] ESA, "Next generation processor for on-board payload data processing application," ESA, Tech. Rep. TEC-EDP/2007.32/RT, Issue 1, Rev. 0, Oct. 2007.

[2] R. Pinto, L. Berrojo, E. Garcia, R. Trautner, G. Rauwerda, K. Sunesen, S. Redant, S. Habinc, J. Andersson, and J. López, "Scalable Sensor Data Processor: Architecture and Development Status," p. 6, 2016.

[3] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano, "High-Performance Embedded Computing in Space: Evaluation of Platforms for Vision-Based Navigation," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 178–192, Apr. 2018. [Online]. Available: https://arc.aiaa.org/doi/10.2514/1.I010555

[4] H. Benninghoff, K. Borchers, A. Börner, M. Dumke, G. Fey, A. Gerndt, K. Höflinger, J. Langwald, D. Lüdtke, O. Maibaum, T. Peng, K. Schwenk, B. Weps, and K. Westerdorff, "Obc-ng concept and implementation," DLR, Deutsches Zentrum für Luft- und Raumfahrt, Tech. Rep. Version 1.0, Jan. 2016.

[5] R. Doyle, "High performance spaceflight computing (hpsc) an avionics formulation task study report," Jet Propulsion Laboratory, NASA, Tech. Rep., October 2012.

[6] R. Pinto, L. Berrojo, E. Garcia, R. Trautner, G. Rauwerda, K. Sunesen, S. Redant, S. Habinc, J. Andersson, and J. López, "Scalable Sensor Data Processor: Architecture and Development Status," p. 6.

[7] J. Le Mauff, "From efpga cores to rhbd soc fpga," in *SpacE FPGA User Workshop*. ESA-ESTEC, April 9-11 2018.

[8] Xilinx, "Zynq-7000 SoC Data Sheet: Overview (DS190)," 2018.

[9] G. Allen, F. Irom, and M. Amrbar, "Zynq soc radiation test results and plans for the altera max10," in *NASA Electronic Parts and Packaging Program (NEPP) Electronics Technology Workshop*. NASA, June 23-26, 2015.

[10] L. A. Tambara, F. L. Kastensmidt, N. H. Medina, N. Added, V. A. P. Aguiar, F. Aguirre, E. L. A. Macchione, and M. A. G. Silveira, "Heavy Ions Induced Single Event Upsets Testing of the 28 nm Xilinx Zynq-7000 All Programmable SoC," in *2015 IEEE Radiation Effects Data Workshop (REDW)*, Jul. 2015, pp. 1–6.

[11] E. C. for Space Standardization, "ECSS-Q-HB-60-02a – Techniques for radiation effects mitigation in ASICs and FPGAs handbook)." [Online]. Available: http://ecss.nl/hbstms/ecss-q-hb-60-02a-techniques-for-radiation-effects-mitigation-in-asics-and-fpgas-handbook-1-september-2016-published/

[12] A. Stoddard, A. Gruwell, P. Zabriskie, and M. Wirthlin, "High-speed PCAP configuration scrubbing on Zynq-7000 All Programmable SoCs," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 1–8.

[13] D. Andrews, W. Peck, J. Agron, K. Preston, E. Komp, M. Finley, and R. Sass, "hthreads: A Hardware/Software Co-Designed Multithreaded RTOS Kernel," vol. 2. IEEE, 2005, pp. 331–338. [Online]. Available: http://ieeexplore.ieee.org/document/1612697/

[14] "Sentinel-2 - Missions - Sentinel Online." [Online]. Available: https://sentinel.esa.int/web/sentinel/missions/sentinel-2

[15] [Online]. Available: https://scihub.copernicus.eu/dhus/#/home

[16] M. Eder, "Real-time detection of clouds on board of satellites with fpgas," Ph.D. dissertation, Bundeswehr Munich University, 2016.

[17] R. Richter, J. Louis, and B. Berthelot, "Sentinel-2 msi – level 2a products algorithm theoretical basis document," DLR and VEGA Sentinel-2 Document, Tech. Rep., 2011.