Analysis of a Heterogeneous Multi-Core, Multi-HW-Accelerator-Based System Designed Using PREESM and SDSoC

Leonardo Suriano*, Alfonso Rodriguez*, Karol Desnos**, Maxime Pelcat**, Eduardo de la Torre*

*Centro de Electrónica Industrial
Universidad Politécnica de Madrid, Spain
{leonardo.suriano; alfonso.rodriguezm; eduardo.delatorre}@upm.es

**IETR, UMR CNRS 6164, UEB
INSA Rennes. France
{Karol.Desnos; Maxime.Pelcat}@insa-rennes.fr

***Abstract*— Nowadays, new heterogeneous system technologies are flooding the market: through the past years, it is possible to observe the move from single CPUs to multi-core devices featuring CPUs, GPUs and large FPGAs, such as Xilinx Zynq-7000 or Zynq UltraScale+ MPSoC architectures. In this context, providing developers with transparent deployment capabilities to efficiently execute different applications on such complex devices is important. In this paper, a design flow that combines, on one side, PREESM, a dataflow-based prototyping framework and, on the other side, Xilinx SDSoC, an HLS-based framework to automatically generate and manage hardware accelerators, is presented. This integration leverages the automatic, static task scheduling obtained from PREESM with asynchronous invocations that trigger the parallel execution of multiple hardware accelerators from some of their associated sequential software threads. An image processing application is used as a proof of concept, showing the interoperability possibilities of both tools, the level of design automation achieved and, for the resulting computing architecture, the good performance scalability according to the number of accelerators and sw threads.**

***Keywords*— Heterogeneous computing, scheduling, reconfigurable MPSoC, SDSoC, PREESM,**

I. Introduction

The advent of reconfigurable Multiprocessor Systems on Chip (MPSoCs) equipped with a combination of several processing cores with programmable logic, such as the Xilinx Zynq-7000 family or, more recently, the Zynq-UltraScale+, enables the possibility of building large systems with unprecedented complexity and integration levels. However, developing efficient implementations requires the execution of multiple tasks simultaneously, relying on the use of both software (SW) and hardware (HW) computing fabrics at the same time, with relatively complex interdependencies among them.

This type of devices may be considered as high-end target platforms within the domain of high performance embedded systems, but at the same time they are becoming more and more important in the high performance computing domain itself, with significant developments in big data or cloud computing applications, among others. However, their inherent heterogeneity and the associated complexity (which forces the use of embedded operating systems, often with real-time constraints), poses important challenges from the point of view of the design methodologies, where a minimum level of automation is required.

High Level Synthesis (HLS) tools have finally paved the path from high-level languages to HW, allowing some flexibility in selecting the levels of parallelism that HW accelerators should have. However, this is not for free, since performance has to be traded against resource utilization and interface complexity. With this idea in mind, FPGA vendors are offering SW-oriented solutions that, starting from SW programs, let a relatively unexperienced HW designer to implement and use HW accelerators. However, it does not directly address the issue of exploiting the concurrency or parallelism levels that the multiple HW and SW cores may offer.

A good work on providing SDSoC with Dynamic and Partial Reconfiguration (DPR) capabilities was recently presented in [30], allowing HW accelerators to be exchanged by other ones. Another important issue is the possibility, for some reconfigurable architectures, of providing scalability and adaptable fault tolerance levels, deciding at runtime the number of HW accelerators and the depth of HW redundancy to be used in the execution of an accelerated task.

By using these such features (DPR and HW scalability) reconfigurable heterogeneous systems would offer an excellent flexibility in the sense that a set of tasks could be mapped into an arbitrary number and combination of SW cores and HW accelerators, especially when the latter have been obtained from HLS processes. In this context, task/thread mapping and the choosing of right scheduling may benefit from tools that (i): explicitly express the parallelism at function level, (ii): provide parameterized specification, (iii): support various models of architecture (MoA) to adapt to different platforms (and number of HW accelerators) and (iv) are able to produce the convenient task scheduling.

PREESM precisely does this. It provides a graphical framework to support the specification of dataflow applications, where actors are implemented as tasks and are linked with parameterizable communication links. The framework contains a mapper that, according to a specified architecture, provides the corresponding task/thread mapping to cores and the corresponding scheduling (shared memories and semaphores for shared memory structures, FIFOs for dataflow-like linked actors, etc…).

In this paper, it is presented a prospection on the use of PREESM and SDSoC for building heterogeneous, dynamically scalable systems, making use of the actor specification (including the parameterization features) and the mapping and scheduling generated from PREESM, combined with HLS-based HW accelerator design as well as all the integration provided by SDSoC. The main highlights of this paper are:

- A proof of concept that shows the possibility of combining dataflow oriented specifications and their associated mapping and scheduling for a multicore-oriented device with HW acceleration using available tools.
- Changes in the HW handling strategy to more efficiently use HW parallelization capabilities may be used to improve performance, by using asynchronous HW invocation calls from within a thread-based multiple-task scheme.

The rest of the paper is structured as follows: a brief description of the state of the art is given in the second section, and in the third and fourth Xilinx SDSoC and PREESM are introduced, respectively. Section V contains the description of the application used for testing the integration of the tools, as well as the experimental results. In the conclusions, the main achievements are highlighted together with the future lines of the work.

II. Background

As predicted by Hartestein in 1997 [1], Reconfigurable Platforms are becoming crucial devices for developing data-intensive applications [2] since they leverage both high-performance and flexibility at the same time, in the same package. However, not only performance and flexibility are important: energy-efficiency of the devices plays a decisive role especially in growing fields such as Internet of Things (IoT), sensor networks, wearable devices [5], and even in space applications [6].

In this context of embedded high computational demand and low power consumption, MPSoCs such as Zynq-7000 and Zynq Ultrascale+ families, designed by Xilinx, are being intensively used [6-10] due to the combination of two ARM processors Cortex-A53 surrounded by 28-nm programmable logic in the case of Zynq 7000 family and, for the Ultrascale+, by four ARM Cortex-A53, two ARM Cortex-R5, a Mali GPU and a 16nm FinFET+ programmable logic fabric [11]. However, an application running on such heterogeneous devices, should guarantee not only the logical correctness of the data output but, also, that the data is “provided on a timely basis” [3]. Therefore, an efficient task scheduling is needed in addition to all the tools already provided to develop applications on these complex systems. In the next paragraphs, a brief overview of the task scheduling methodologies is given, focusing mainly on heuristic methods and on the tools available for programming such systems.

According to the nature where they are applied, task scheduling methodologies can be classified into (i) homogeneous computing systems, (ii) heterogeneous computing systems and (iii) reconfigurable computing systems, as explained by Vucha and Rajawat in [12].

A. Homogeneous Computing Systems

Homogeneous computing systems include multiple processors, all with the same characteristics. Different scheduling techniques were proposed during the 70’s and 80’s in really well known works such as [13-15]. One of the most popular ones is the *Rate Monotonic* Algorithm (RM) [14]. This is a static priority based algorithm. It assigns a high priority to the most recurring task and a low priority to the one that is less executed. Due to the limitations of this scheduling, the *Earliest Deadline First* algorithm (EDF) was proposed in [13], where the priority of execution is assigned based on the deadline of the tasks. The main advantage of this approach is the dynamism of the algorithm, compared to the previous one. Some of the most important scheduling algorithms are the *Task Duplication Based scheduling* [18], the *Maximum Urgency First Algorithm* [15], the *Dynamic Critical Path scheduling* [16] and the *Selective Duplication* algorithm proposed in [17]: all of them are targeted upon a set of homogeneous processors.

B. Heterogeneous Computing Systems

A heterogeneous computing system is, normally, a set of processing elements different among them. Therefore, the application is spread upon it, taking into account the priorities of all the tasks and the time necessary for every processing element of the system to process them [12]. The milestone strategies of all the heterogeneous system scheduling algorithms developed so far are presented in [19]. The *Heterogeneous Earliest Finish Time* (HEFT) and *Critical-Path-On-a-Processor* (CPOP) there proposed, are the bases of countless recent works like, for example, the *Heterogeneous Scheduling Algorithm with Improved Task Priority* (HSIP) proposed in 2015 [20].

C. Reconfigurable Computing Systems

Reconfigurable Computing Systems are “an emerging paradigm” [12] where both flexibility and performance are met. It gives the programmers the possibility of customizing and changing, dynamically, the HW architecture to execute concurrently applications and tasks. The problem was studied in [4][21][22][23][24] and [25], where different solutions for an online scheduler of real-time tasks were also proposed. All these solutions have the common aim of “reducing configuration overheads through resource reuse and minimizing the total execution time in addition to decrease task rejection ratio” [12].

However, the most recent hot topic concerns the Heterogeneous Reconfigurable Computing Systems, in which the Zynq-7000 and Zynq Ultrascale+ devices can be included. In the [26], researchers from Politecnico di Milano show how an efficient resource-aware scheduling can improve performance on a ZedBoard using DPR. Nevertheless, as Vucha and Rajawat highlight in [12], there is still a “need of developing scheduling methodologies for Heterogeneous Reconfigurable Computing Systems (HRCS), which is an emerging high speed computing platform for real time applications”.

The preliminary work presented in this paper should be intended as a proof-of-concept in which a manual integration of PREESM and SDSoC is proposed, to provide a static schedule of a given application together with the automatic generation of hardware accelerators by SDSoC, in charge of driving the “translation” of C code into IP cores embedded in Vivado.

III. Xilinx SDSoC

The SDSoC development environment is a tool introduced by Xilinx [27] in the 2015 for the MPSoCs previously mentioned where, using as inputs programming languages such C/C++, it is possible to generate applications (standalone or running upon an Operating System) that can offload part of the computationally-intensive tasks to the reconfigurable fabric of the target device. The hardware accelerators are generated using High Level Synthesis techniques, and the bitstreams are produced by invoking Vivado. In addition, SDSoC provides system level profiling techniques for performance estimation, as well as estimations on the overheads introduced by data communication between Processing System (PS) and Programmable Logic (PL).
Another reason for the choice of such tools lies in future evolution of this paper. In fact, Kalb and Göhringer, in [30], demonstrate the integration of the DPR in SDSoC achieving further acceleration in time performance and, at the same time, a reduction of power consumption. This is, clearly, another aspect that needs to be considered in the development of applications that make use of hw acceleration.

A. Workflow

The first step for designer is to select the board (featuring a Zynq device) where to run the test applications. In our case a Zedboard by AVNET was selected, equipped with a Xilinx Zynq-7000 AP SoC XC7Z020-CLG484 which encloses two ARM processor A53.
The second step is the choice of the OS: SDSoC can, in fact, create application running on an embedded Linux OS or on a FreeRTOS real time OS or in bare-metal enviroment.
The most important element in SDSoC is the SDS compiler that is the responsible, after, of calling the tools of Vivado for the HLS “translation” of C/C++ code in HDL, the creation of the whole system in RTL (including the communication infrastructure) and the generation of the bitstreams.
The final step is the generation of a bootable image (by invoking Bootgen tool) that contains the kernel of the eventual OS, the bitstrams, the devicetree blob, the application and all the other necessary elements for correctly booting the system.

IV. PREESM

PREESM is an open-source framework developed by researchers of INSA in France and Texas Instruments. It is a dataflow-based tool for rapid prototyping and simplifying multi-core Digital Signal Processing (DSP) programming [28].

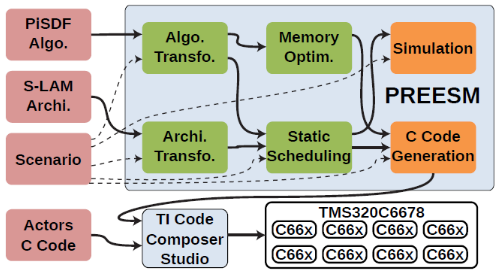

Fig. 1. Rapid Prototyping Process: Workflow example for TMS320C6678 [28]

It provides automatic generation of deadlock-free code, and features memory and time analysis thanks to the Parameterized and Interfaced Synchronous Dataflow (PiSDF) Model of Computation (MoC). In [29], the authors describe how an application can be divided into actors that communicate through FIFOs (i.e. *First In First Out data queues*) using PiSDF: the generation of the code is then performed taking into account the System-Level Architecture Model (S-LAM) that provides a high-level description of the device. In figure 2 an example of PiSDF is shown, whereas in figure 3 an example of S-LAM, related with the DSP TMS320C6678 by Texas Instrument, can be seen.

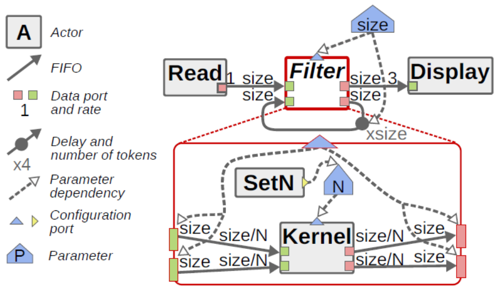

Fig. 2. An Example of a PiSDF Model [28]

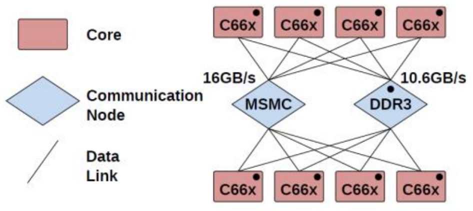

Fig. 3. example of S-LAM for TMS320C6678 [28]

The PREESM framework does not provide yet a code generation optimized for HLS techniques and the code itself is not intended to be exported directly into SDSoC for hardware accelerator generation: in the S-LAM is not possible to specify the HLS nature of the cores (i.e. it is not possible to specify that some specific functions should be executed on the FPGA). Therefore, one of the main contributions of this work consist of manually adapting, for an image-processing/video-stream application, the generated deadlock-free code for Synchronous Data Flow computation in SDSoC, applying, when necessary, pragma directives to guide the hardware optimization process.

V. Video-Image processing application (dataflow)

In this section, the different experimental setups are presented. Notice that all of them share the same functionality (the same high-level definition in PREESM), even though the hardware/software partitioning or certain configuration parameters (e.g. number of stripes in the image) might be different.

In order to establish a baseline solution, the unmodified output of PREESM is executed in the target platform (dual-core ARM Cortex A9). Every implementation alternative is then compared with this reference solution.

For this first version, where only software is executed, it is not possible to perform automatic code instrumentation to track the events of the function calls (this feature of SDSoC is available only when moving functions to hardware). However, it is possible to measure the execution time by manually instrumenting the code or by using software-profiling functions available in Linux.

The other versions will contain different number or hardware accelerators, different number of threads invoking the hardware functions and different strategies of performing these hardware calls.

A. SW version

With the software version, it was possible to reach 150 frames per second using a video 352 pixels wide and 288 pixels high. In this case, the video was divided in 8 slices in order to exploit data-level parallelism (each image stripe can be independently processed by a different instance of a Sobel actor). So, to execute a Sobel function with a slice of 352x38 pixels upon an ARM processor clocked at 667MHz, 300us are needed (average over three hundred executions). For this "baseline-set" test, only one threads was used in the execution.

B. SW accelerated by HW execution

The second version of the program uses four hardware accelerators generated by SDSoC. In this case, SDSoC is left free to handle hardware function calls. In this case, a 115 frame-per-second (fps) rate is reached when executing the whole image pipeline, and it takes 650us to execute every instance of the Sobel function in hardware. The frequency at which hardware accelerators operate was intentionally left to the default value of 142.86 MHz in order to show that, with a modified and parallelism-aware manual invocation strategy, execution performance can be improved significantly.

In Fig. 4, a graph reporting the experimental "event tracing" is shown. Notice that, by default, SDSoC implements hardware-function handling using a blocking approach, i.e. each accelerator has to finish before the software can proceed executing the rest of the application code. Of course, this leads to an *inefficient* use of FPGA resources (it is possible to obtain similar performance values with just one hardware accelerator instead of four).

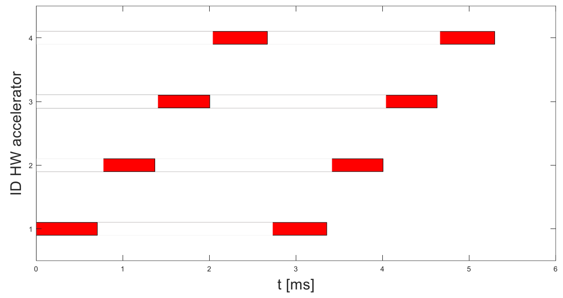

Fig. 4. Execution of HW function in the case of automatic handling of hardware function calls by SDSoC

C. Indroduction of "pragmas"

In the third version, the use of `#pragma SDS async()` was introduced. Whit these compiler instructions it is possible to call a hardware function in a non-blocking way, i.e. without waiting for the outputs to be available. Every "`async`" call is associated with a `#pragma SDS wait()` that has to be placed just before the program needs to gather the outputs. In this case it is possible to reach 200 fps on the same video elaboration and, in Fig. 5, it is shown that the strategy of the asynchronous function call was useful: inside only a sequential software thread, another level of parallelism can be exploited. In fact, it is easy to note that the execution of the image processing functions is overlapped in time. Therefore, and although the performance of a single call to Sobel accelerator is worse than the one of its software counterpart, it is possible to reach a speedup of 25%.

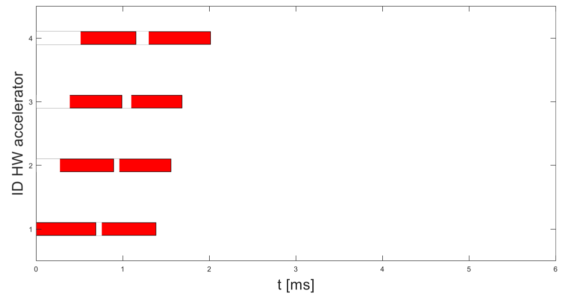

Fig. 5. Execution of HW function in the case of manual hw handling using one thread

Following, it is shown that it is still possible to improve the performance adding other optimization to the code and to the HLS synthesis (by increasing the clock frequency of the HW functions and using more threads to perform the HW calls). In addition, as it was described, the frequency of the communication bus and the clock signal of the RTL block is increased giving better performances in executing the same functionalities.

D. Modification of HW accelerators clock frequency

Bringing the frequency of the clock of the HW accelerators from 142.86MHz to 166.67MHz, in the case of the automatic hardware-call handling in SDSoC, the execution of the Sobel actor is reduced from 650us to 550us and the frame rate of the video goes from 115fps to 125fps. Anyhow, in the case of manual strategy optimization, it is possible to achieve an improvement of 5 fps (from 200 fps to 205fps).

In table 1, all the results obtained with 1 thread and 4 hardware accelerators are shown with the different values of frequency used in both manual and automatic accelerator handling.

Table 1.Performance with 1 thread and 4 HW accelerators

	1 Thread - 8 Slices 4 Accelerators		
HW freq. [MHz]	142.86	166.67	200
SDSoC hw handling	115 fps	125 fps	135 fps
Manual hw handling	200 fps	205 fps	207 fps

E. Improving performance by using two Pthreads

Now, using again PREESM to generate more threads (Pthreads), it is possible to use the two ARM cores available in the Zedboard for the concurrent execution of software and, for every generated Pthread, offload the computation of the different instances of the Sobel actor to one or more hardware accelerators. In this case, again, for every Pthread, it is necessary to take care of manually performing the asynchronous handling of the hardware calls in order to better exploit the computational power enabled by the parallel resources of the FPGA.

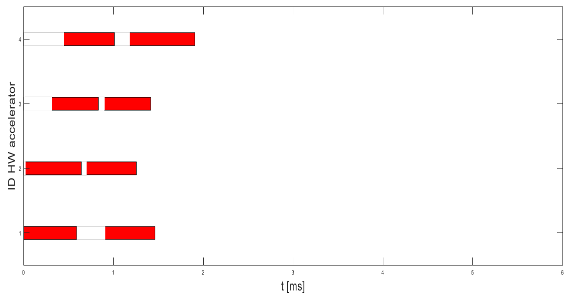

Fig. 6. Execution of HW function in the case of manual hw handling using two threads

In Fig 6, a graph obtained by manual handling of HW calls is shown: as expected, the level of parallelism of the execution of the functions is even higher due the independence of the call inside Pthreads. In contrast, in Fig 4, a graph of the event tracing for the case of the default SDSoC HW calls management is shown.

In table 2 al the results obtained using two Pthreads and four hardware accelerators are shown.

Table 2.Performance with 2 threads and 4 HW accelerators

	2 Thread - 8 Slices 4 Accelerators		
HW freq. [MHz]	142.86	166.67	200
SDSoC hw handling	150 fps	160 fps	171 fps
Manual hw handling	211 fps	218 fps	222 fps

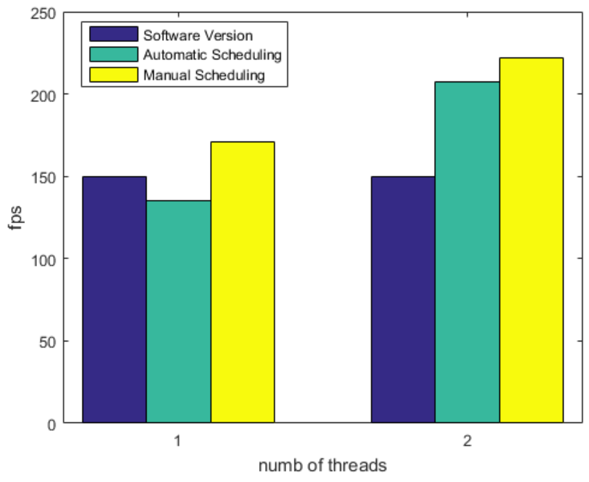

Fig. 7 . Frame per second obtained using 1 or 2 threads in case of manual and automatic hw handling

To summarize, in Fig. 7, the software version of the video application executed on the ARM processors of the Zynq device (dark-blue bar) is compared with other two versions where four hardware accelerators are used in each. On the left side, only one thread is used and, on the right side, two threads.

The light-green bars show that, leaving SDSoC free to call the hardware accelerators, the performance decreases when one thread is used while, using two, an improvement is obtained. However, the improvement is achieved always if a manual managing of the hardware function is performed (yellow bar) by using asynchronous calls.

VI. Conclusion and future work

In this paper it is proposed, as a proof-of-concept, the integration of two existing tool for the scheduling and the generation of hardware accelerators: PREESM, a data-flow based framework for rapid prototyping, and SDSoC, a commercial tools developed by Xilinx that generates hardware accelerators exploiting HLS techniques. It is demonstrated that a careful asynchronous invocation of hardware accelerators inside threads (already generated by PREESM) can bring benefits, pushing the performance of a video streaming application from 150 fps up to 220 fps with an improvement of 48% on the execution time.

However, this is a first step that will lead to an evolution of PREESM where it is possible to automatic generate deadlock-free and scheduled code ready to be compiled by SDSoC with the right pragmas in order to generate and call the hardware function execution.

For future work it is possible to consider, also, the energy and power consumption

Acknowledgments

Leonardo Suriano holds a predoctoral contract under RR01/2015 (Programa Propio) by Universidad Politécnica de Madrid**.**

References

[1] R. Hartestein, "Microprocessor Is No More General Purpose: Why Future Reconfigurable Platforms Will Win" Invited Paper, Of The International Conference On Innovative Systems In Silicon.Isis'97, Texas, Usa, Pp 1- 10, October 8-10, 1997.

[2] D. Wang, S. Li and Y. Dou, Loop Kernel Pipelining Mapping Onto Coarse-Grained Reconfigurable Architecture For Data-Intensive Applications, In Journal Of Software, Volume 4, No-1,P81-89, 2009.

[3] J. Singh and N. Auluck, "Real time scheduling on heterogeneous multiprocessor systems — A survey," 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, 2016, pp. 73-78.

[4] J. Spasic, D. Liu and T. Stefanov, "Energy-efficient mapping of real-time applications on heterogeneous MPSoCs using task replication," 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Pittsburgh, PA, 2016, pp. 1-10.

[5] H. Cherupalli, R. Kumar and J. Sartori, "Exploiting Dynamic Timing Slack for Energy Efficiency in Ultra-Low-Power Embedded Systems," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, 2016, pp. 671-681.

[6] D. Rudolph, C. Wilson, J. Stewart, P. Gauvin, A. George, H. Lam, G. Crum, M. Wirthlin, A. Wilson and A. Stoddard, CHREC space processor: a multifaceted hybrid architecture for space computing, Proc. of the AIAAIUSU Conference on Small Satellites, 2014.

[7] K. Railis, V. Tsoutsouras, S. Xydis and D. Soudris, "Energy profile analysis of Zynq-7000 programmable SoC for embedded medical processing: Study on ECG arrhythmia detection," 2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Bremen, 2016, pp. 275-282.

[8] S. Shaikh and S. Pujari, "Migration from microcontroller to FPGA based SoPC design: Case study: LMS adaptive filter design on Xilinx Zynq FPGA with embedded ARM controller," 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), Pune, 2016, pp. 129-134.

[9] K. Railis, V. Tsoutsouras, S. Xydis and D. Soudris, "Energy profile analysis of Zynq-7000 programmable SoC for embedded medical processing: Study on ECG arrhythmia detection," 2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Bremen, 2016, pp. 275-282.

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

[11] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan and R. Wittig, "UltraScale+ MPSoC and FPGA families," 2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, 2015, pp. 1-37. doi: 10.1109/HOTCHIPS.2015.7477457.

[12] M. Vucha and A. Rajawat, "A Case Study: Task Scheduling Methodologies for High Speed Computing Systems", International Journal of Embedded systems and Applications (IJESA) Vol.4, No.4, December 2014

[13] C. L. Liu And James W. Layland, Scheduling Algorithms For Multiprogramming In A Hard Real Time Environment, Journal Of Acm, Vol. 20, No. 1, Pp. 46-61, 1973.

[14] J. Lehoczky, L. Sha And Y. Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior, Proceedings Of Real Time Systems Symposium, Pp. 166-171, Dec. 1989.

[15] Wei Zhao, K. Ramamritham and J. A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," in IEEE Transactions on Software Engineering, vol. SE-13, no. 5, pp. 564-577, May 1987.

[16] Y.K. Kwok And I. Ahmad, "Dynamic Critical Path Scheduling: An Effective Technique For Allocating Task Graphs To Multiprocessors", Ieee Trans. Parallel Distributed Systems, Vol. 7, No. 5, Pp. 506-521, May 1996.

[17] An Improved Duplication Strategy For Scheduling Precedence Constrained Graphs In Multiprocessor Systems, Ieee Trans. Parallel And Distribution Systems, Vol. 14, No. 6, June 2003.

[18] S. Darba And D.P. Agarwal, "Optimal Scheduling Algorithm For Distributed Memory Machines", Ieee Trans. Parallel And Distributed Systems, Vol. 9, No. 1, Pp. 87-95, Jan. 1998.

[19] H. Topcuoglu, S. Hariri And Min-You Wu, Performance Effective And Low-Complexity Task Scheduling For Heterogeneous Computing, Ieee Transactions On Parallel And Distributed Systems, Vol. 13, No. 3, Pp. 260 – 274, March 2002.

[20] G. Wang, H. Guo and Y. Wang, "A Novel Heterogeneous Scheduling Algorithm with Improved Task Priority," 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, 2015, pp. 1826-1831. doi: 10.1109/HPCC-CSS-ICESS.2015.48.

[21] Xue-Gong Zhou, Ying Wang, Xun-Zhang Haung And Cheng-Lian Peng, On-Line Scheduling Of Real Time Tasks For Reconfigurable Computing System, International Conference On Computer Engineering And Technology, Pp 59-64, 2010.

[22] M. M. Bassiri And H. S. Shahhoseini, A New Approach In On-Line Task Scheduling For Reconfigurable Computing Systems, In Proceedings Of 2nd International Conference On Computer Engineering And Technology, Pp. 321-324, April 2010.

[23] K. Dane And M. Platzner, A Heuristic Approach To Schedule Real-Time Tasks On Reconfigurable Hardware, In Proceedings Of International Conference On Field Programmable Logic And Applications, Pp 568 – 578, 2005.

[24] S. Raju Kota, C. Shekhar, A. Kokkula, D. Toshniwal, M. V. Kartikeyan And R. C. Joshi, "Parameterized Module Scheduling Algorithm For Reconfigurable Computing Systems" In 15th International Conference On Advanced Computing And Communications, Pp 473-478,.2007.

[25] Ali Ahmadinia, Christophe Bodda And Jurgen Teich, A Dynamic Scheduling And Placement Algorithm For Reconfigurable Hardware, Arcs 2004, Lncs 2981, Pp. 125 – 139, 2004.

[26] A. Purgato, D. Tantillo, M. Rabozzi, D. Sciuto and M. D. Santambrogio, "Resource-Efficient Scheduling for Partially-Reconfigurable FPGA-Based Systems," 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, 2016, pp. 189-197.

[27] Xilinx Inc., “SDSoC Environment User Guide (UG1027)”, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug1027-sdsoc-user-guide.pdf

[28] Pelcat, Maxime; Desnos, Karol; Heulot, Julien; Guy, Clément; Nezan, Jean-François; Aridhi Slaheddine (2014) "PREESM: A Dataflow-Based Rapid Prototyping Framework for Simplifying Multicore DSP Programming". EDERC 2014, Milan, Italy.

[29] Desnos, K., Pelcat, M., Nezan, J.F., Bhattacharyya, S.S., Aridhi, S.: Pimm: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration. In: SAMOS XIII (2013)

[30] T. Kalb and D. Göhringer, "Enabling dynamic and partial reconfiguration in Xilinx SDSoC," 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2016, pp. 1-7.