Improving the performance of Real-Time Event Processing based on Preemptive Scheduler FPGA Implementation

Ionel Zagan[1,2] and Vasile Gheorghiță Găitan[1,2]
[1] Faculty of Electrical Engineering and Computer Science, Stefan cel Mare University, Suceava, Romania
[2] Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Stefan cel Mare University, Suceava, Romania
zagan@eed.usv.ro

***Abstract*—The design of an RTOS (real-time operation system) kernel is mostly based on a preemptive scheduler that responds to events and interrupts according to the priorities. The hardware RTOS (HW-RTOS) concept includes a real-time scheduler which ensures a deterministic execution of a set of tasks by implementing in hardware the appropriate scheduling algorithms, similar to those of the RTOS. The contributions of this paper consist in presenting original methods that contribute to increasing the performance of the RTOS, to reduce the jitter effect by implementing a unified space of task and event priorities, to increase the execution factor and controlling the RTS (real-time system) scheduling limit, by implementing the real-time event handling unit.**

Keywords—Field programmable gate arrays, Pipeline processing, Scheduling, Computer architecture

I. INTRODUCTION

With the unprecedented development of microcontroller circuits, also called embedded processors, digital computing has penetrated all fields, and it can be said that there is no field where at least one microcontroller is not present. Real-time requirements additionally introduce predictability of program execution and fast context switching. Under the circumstances of costs, times to market, and the competitive environment, most of the embedded applications running on microcontrollers or FPGAs (Field-Programmable Gate Array) tend to be real-time, as a consequence of interacting with the real world. A digital system is primarily a multitude of combinational and sequential connected circuits. All resources used by a FPGA for the flexibility of the development platform [1], decrease the performances of the entire circuit [2], [3]. Nevertheless, it will never be able to generate a clock signal at speeds equal to those of a dedicated circuit. In comparison, an ASIC (Application-Specific Integrated Circuit) can reach speeds higher than 4GHz, while a FPGA runs in very good conditions at only 450MHz. Although FPGA uses more power in comparison to the ASIC circuits, it has a major advantage because it is suitable for small and medium size low-cost implementations with unlimited reconfiguration possibility, at least in theory.

This work is supported by the project ANTREPRENORDOC, in the framework of Human Resources Development Operational Programme 2014-2020, financed from the European Social Fund under the contract number 36355/23.05.2019 HRD OP /380/6/13 – SMIS Code: 123847.

The scheduler is an essential part of any operating system, being in direct contact with the hardware part represented by the machine level. The implementation of a real-time event handling block allows embedding the hardware scheduler deeply in the processor pipeline, in order to schedule on an instruction-per-instruction basis due to a hardware-implemented real-time scheduling scheme.

This paper presents a custom CPU architecture based on the replication of resources, such as program counter, general purpose registers and pipeline registers. The implementation of this new processor concept, based on a real-time scheduler engine implemented in hardware, tries to meet the current rigorous requirements imposed by critical RTSs. The originality of the approach resides in the predictable execution of real-time tasks on custom architecture based on replicated resources [4], [5]. This processor, implemented and validated using the xc7a100tcsg324-1 FPGA device produced by Xilinx [6], is designed especially for small real-time applications, due to its resource multiplication technique. The innovative real-time scheduler of the proposed architecture, used for obtaining experimental results, is an integral part of the processor and directly responsible for remapping the set of pipeline registers and the register file.

The first section of the paper contains a brief introduction, and section II shows a description of the hardware accelerated RTOS architecture. Section III is dedicated to the real-time event handling block, and section IV describes the validation of the inter-task synchronization and communication mechanisms, including the hardware block for handling multiple events. Section V contains the conclusions and future directions of research.

II. RELATED WORK

The hthread project, proposed by Andrews et al in [7], is a hardware/software implementation of a multi-thread architecture that is an embedded operating system. The core is preemptive and has 256 software execution threads and 256 active hardware execution threads. To provide the services required for the functioning of the RTS, the hthread operation

978-1-7281-6870-8/20/$31.00 ©2020 IEEE

system consists of four different hardware cores. The SPEAR concept proposed by Delvai and others in [8], based on a single execution path and a three-stage pipeline, simplifies the problems created by WCET (Worst-Case Execution Time), at the same time guaranteeing the predictability of the RTS. In some cases, when the application tasks have slow execution times and severe time limitations, the overhead introduced by the execution of RTOS core functions cannot be neglected and can generate important interferences in the execution of tasks. In these situations, the predictability of the system can only be guaranteed if the effects of the overhead of the operating system are taken into account in the feasibility analysis for establishing the scheduling scheme. In software operating systems, the operation of saving and eventually restoring the current task contexts inserts significantly longer time delays (from a few µs to tens of µs), and it also increases the unpredictability degree generated by various search operations in lists or tables based on the task identifier. Even RTLinux [9], a commercial real-time operating system, has a 32µs jitter for the scheduling operation (worst case jitter for a Compaq iPAQ PDA based on a 200MHz StrongArm).

At the hardware micro-architecture level, there is a continuous research effort on precision-timed (PRET) architectures. Firstly, we need to point out that Heckmann and others in [10] and Berg and others in [11] have identified the architectural properties that complicate the WCET analysis and propose certain facilities that make it easier to calculate this coefficient. One solution to guarantee the performance and predictability of the system could be to move the real-time operating system primitives and mechanisms, including the scheduler, into hardware, thus, on the one hand, the over-control of the basic functions implemented by the software is eliminated and, on the other, the jitter effect is significantly reduced.

III. HARDWARE RTOS ARCHITECTURE AND REAL-TIME SCHEDULER

To guarantee the real-time features, the hardware scheduler [12] uses a unified space for tasks and interrupts, and the interrupts inherit the priority of the task to which it is attached. The result is a structure representing the *n*-multiplexed resources. Together with the memory and datapath functional blocks, this structure forms a typical RISC (Reduced Instruction Set Computer) architecture [13] which we will call semi CPU (sCPU). An *i* instance of this semiprocessor will be called semiprocessor *i* (sCPUi).

Fig. 1 presents the hardware block diagram for the real-time preemptive scheduler, including the connections with the CPU implemented in Nexys4 DDR (xc7a100tcsg324-1) development kit. Our processor is designed for a high context switch, not for high frequency. The tests performed until now, have not influenced the operating frequency of the CPU (33MHz). The static scheduler implements the priority preemptive scheduling method considering a CPU version with four sCPUi (sCPU0, sCPU1, sCPU2 and sCPU3). Any sCPUi can be disabled by *cr0MSTOP* = 32'h0000000F scheduler control register. By parameterizing the Verilog HDL implementation, it was possible to analyze the FPGA resources needed for these extensions, thus performing a comparative analysis with other similar CPU implementations. To implement this scheduling model, it was necessary to design the real-time event handling unit with a group of monitoring registers named *mrTEVi[0:3][31:0]* to define the reload timer value for each sCPUi. To synchronize with the on-chip program memory, the signals used in the case of reading and writing operations are modified on the positive edge of the *clock_mem* signal, various frequencies for this signal being used during the implementation of the proposed processor.

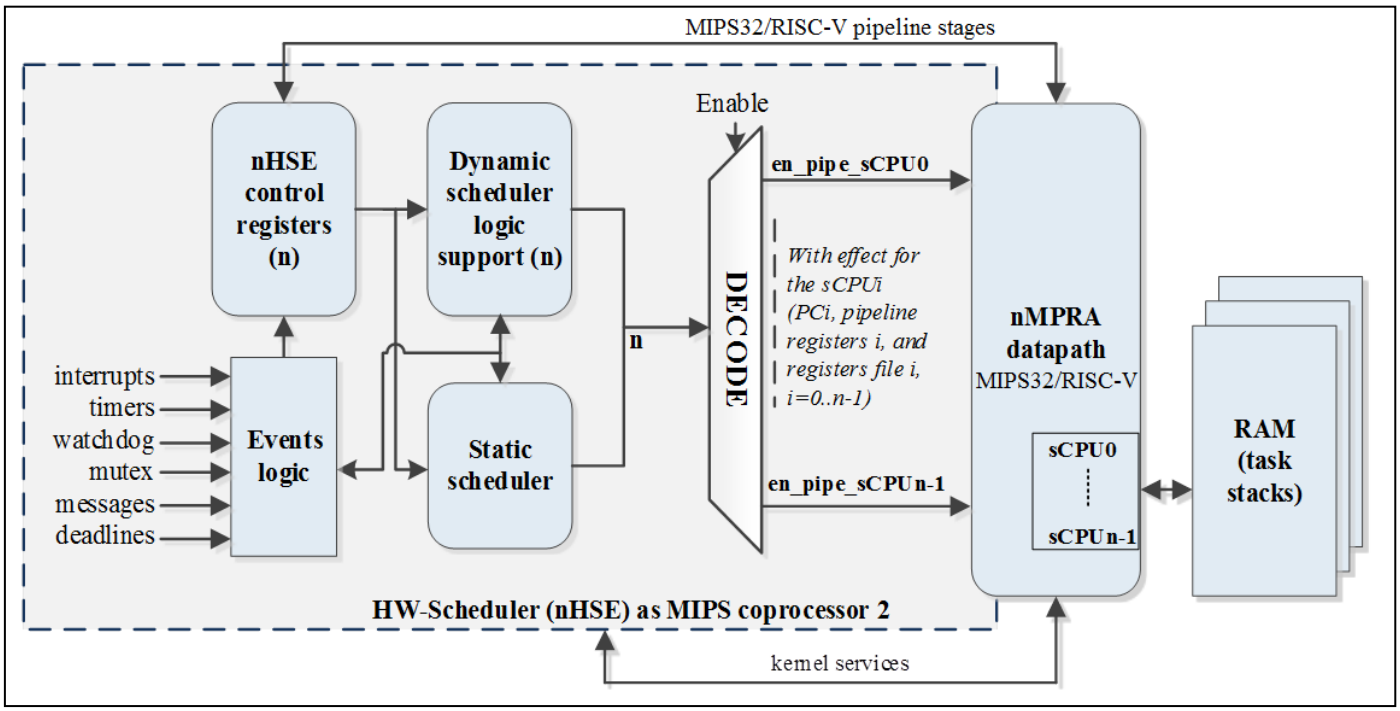

Fig. 1. Hardware real-time scheduler block diagram and MIPS32(Microprocessor without Interlocked Pipelined Stages)/RISC-V datapath.

Being a configurable scheduler, the hardware real-time event handling module enables designers to utilize more efficiently the processor time and modify the task set parameters in order to reach maximum performance for the applied real-time applications. Fig. 2 presents the general design of the static real-time scheduler. The interrupt treating mechanisms can introduce predictable delays for critical tasks, causing the non-compliance with their execution deadlines. For this reason, the interrupt treating mechanism is integrated within the scheduling mechanism. Therefore, the interrupts can be programmed the same way as tasks, thus guaranteeing the feasibility of the preempted scheduling scheme based on priorities.

The preemptive static or dynamic scheduler implemented in hardware allows the activation of interrupts, and also the attachment of these interrupts to tasks (sCPUi) with lower priority. If the priority monitoring registers (*mrPRIsCPUi*) of two different sCPUi, corresponding to the dynamic priority scheduler, have the same value which is not 0, the operation of the hardware scheduler is unpredictable. Because the interrupts adopt the same execution as tasks, the process of activating or deactivating is achieved using an instruction set dedicated to the real-time scheduler and associated with COP2. The activation or deactivation of any sCPUi specific resources can be accomplished with *en_pipe_sCPU0* through *en_pipe_sCPUn-1* signals. This way, the proposed schematic can be used for static scheduling, if each task runs on a sCPUi. In this case, the static priorities are identified by the IDs of the tasks. Regarding the compatibility with the designed architecture, we provide the following clarification in the paper. The notion of TCB (Task Control Block) in the proposed hardware accelerated RTOS is poorly diluted because the proposed architecture is based on resource multiplication (for example a set of registers for each semiprocessors CPUi), access to global resources (interrupts, mutex, signals) is achieved through the implementation of special registers with global atomic access (*reg [31:0] grMutexi [0:NR_MUTEX-1]; reg [31:0] grSSRi [0:NR_EV-1]; reg [31:0] mrCommRegij[0 : NR_REG_INTERTASK_COMM*NR_TASKS-1];*).

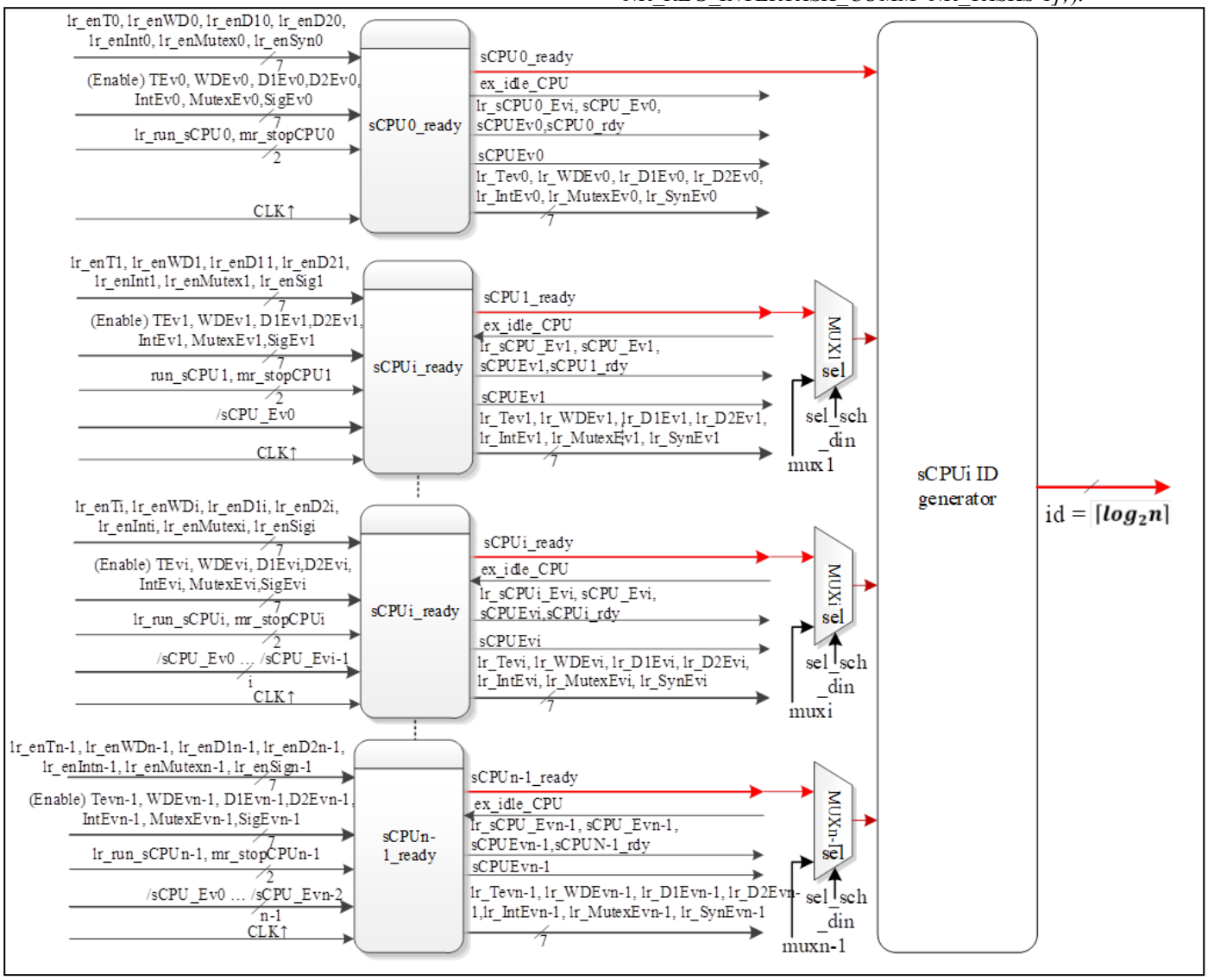

Fig. 2. Multiple prioritized events logic based on real-time event handling unit.

Data hazard occurs when is a dependence between instructions, regarding destination and source operands values. Since these values are not yet calculated and updated, they cannot be used by the running instruction, thus causing a hazard situation. This research paper reflects the importance to design a custom processor architecture in order to be successfully used in RTSs.

Within the proposed processor, data forwarding is performed via the Hazard_Detection module, the non-availability of the necessary data determining the introduction of NOPs (No OPeration), thus reducing the performances of the assembly line. The ALU unit carries out the corresponding operation, providing at the output the result to be saved in the *EX_ALUResult[31:0]* pipeline register. It is important to mention the correlation between the control signals, the provided operands, and the signals necessary for their selection in the case of hazard situations, as well as the content of the control signals, in the case of exceptions [12], [13]. When a program sequence generating a hazard situation occurs on the assembly line of the processor, the Hazard_Detection module must generate a set of signals to successfully resolve this problem [14]. As the data redirecting and hazard detection operations are performed concurrently, the multiplexers from the ID and those from the EX stages retrieve data and transfer them directly to the arithmetic and logic unit; this way, a further delay of the assembly line is avoided. In order to ensure the consistency of data when hazard situations which cannot be resolved by redirecting data occur, *IF_Stall* and *ID_Stall* signals avoid the retrieving and execution of the following instructions. Thus, control signals and data already existing in the pipeline registers are overwritten with the same fields. The data redirecting unit provides the necessary signals so that all data dependencies are resolved when this operation is possible. To do this, and depending on the occurred situation, data forwarding is performed in the ID, EX and MEM pipeline stages.

IV. Validation of the Proposed Hardware Scheduler

The *DataMem_Address[29]* signal indicates an operation with inputs/outputs mapped in the data memory address space, and the *DataMem_Address[28:26]* = 3'b100 signals indicate LEDs as destination outputs for the *M_ReadData2* data. A bit will signal a possible error if there is no active interrupt. Taking into account the applicability fields and capabilities range of the new concept, the economic impact is significant; moreover, it must be taken into account the fact that it involves a real-time processor that eliminates the need for pricey testing required to determine the WCET for testing and certification. In the validation section, various tests have been introduced to test that hardware accelerated RTOS based on real-time scheduler is more efficient than other conventional schedulers in terms of HW-RTOS support or RTSs performance. The implementation of timers was necessary for increasing the speed of task scheduling (the response to time-type events is significantly faster). Implementing this mechanism in software using a single timer would introduce additional and different time delays to the time-type events. Fig. 3 illustrates the monitoring registers for verifying processor performance implemented at custom real-time scheduler level. At the time moment marked by marker C1, *the mrCntRuni[2]* register is incremented and then remains at the value 0x000000a8, because sCPU2 stops from execution. It can be noticed how during the time the *nHSE_EN_sCPUi* signal is inactive, the *mr0CntSleep* register counts the processor's non-running cycles. The C2 marker indicates the moment when sCPU1 is scheduled for execution, the *mrCntSleepi[1]* register contain the number of cycles during which sCPU1 has not been in the RUN state. The *mrCntRuni* and *mrCntSleepi* registers are multiplied for each sCPUi, and they store the real-time processor status with regard to machine cycles.

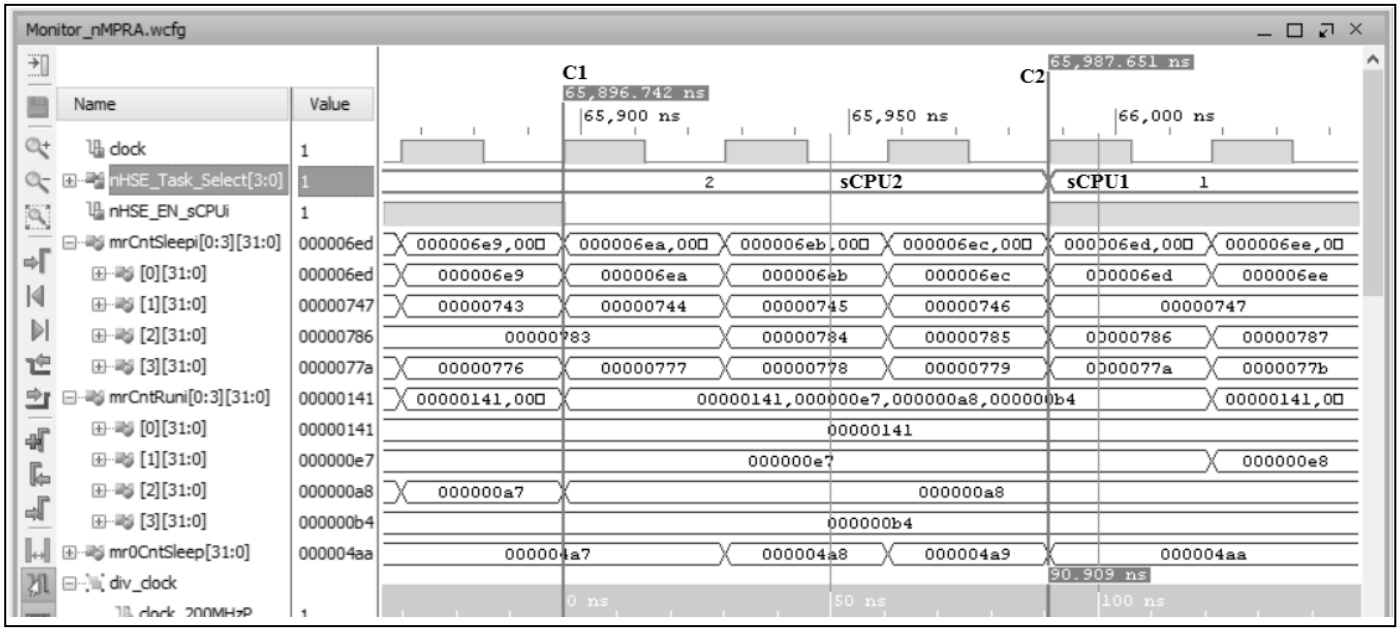

Fig. 3. Monitoring registers corresponding to free clock cycles and those allocated by hardware scheduler to each sCPUi.

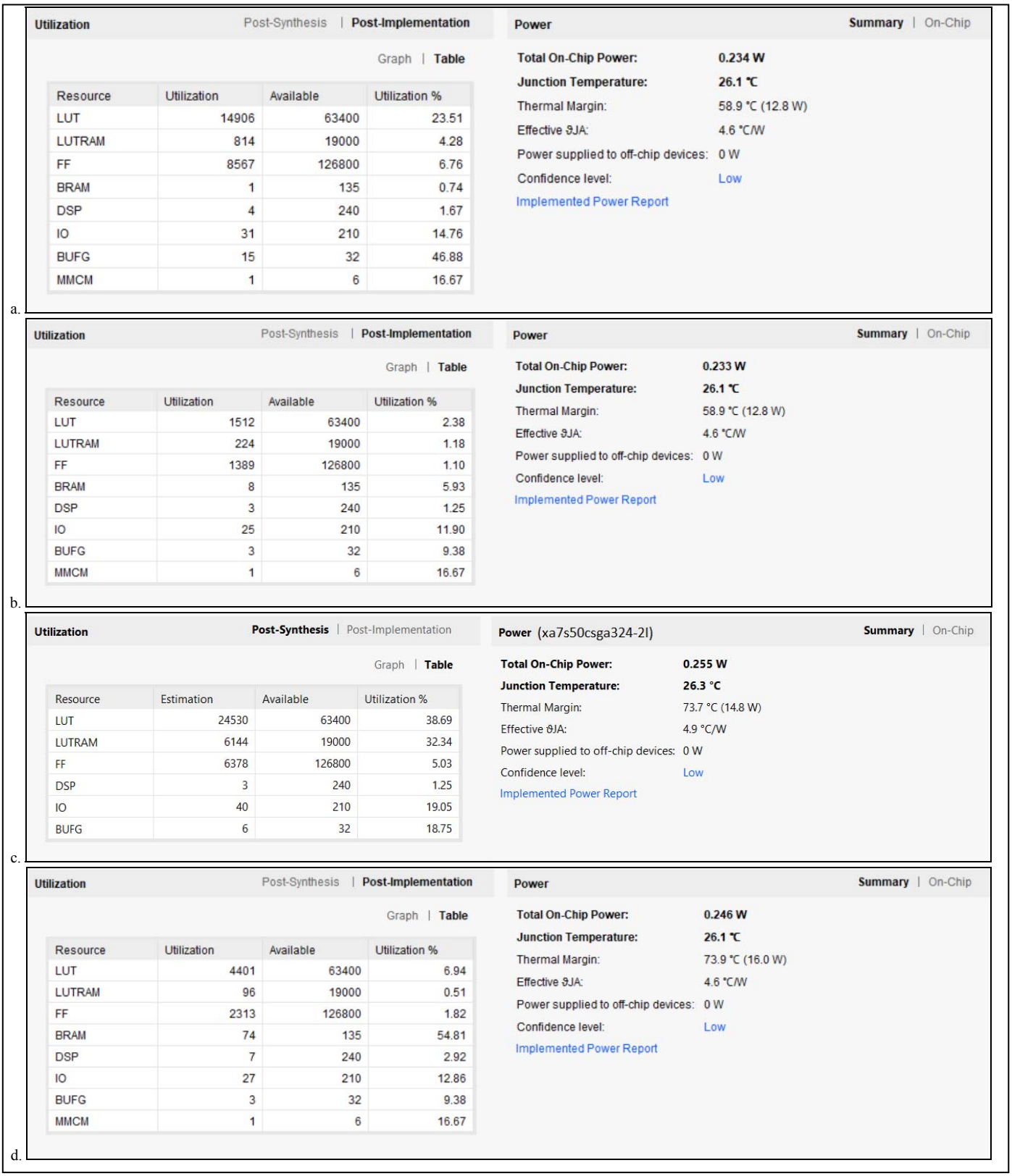

Fig. 4. Nexys4 DDR resource utilization and power report based on different design architectures: a) nMPRA project based on hardware scheduler and 4 sCPUi); b) MicroBlaze microcontroller (3 stage pipeline stages); c) ARM Cortex-M3 microcontroller [15]; d) MIPS32 (5 stage pipeline stages).

The *mrCntRuni[0:3]* registers enable the reading of the life cycles of the sCPUi, the stored value representing the number of processor cycles during which the sCPUi was in execution. The *mrCntSleepi[0:3]* registers enable the reading of the non-operating cycles of each sCPUi. The stored value is the number of processor cycles during which the sCPUi has not been in the running state. The *mr0CntSleep* registry is the registry enabling the reading of the non-operating cycles of the entire CPU, the global value representing the number of processor cycles during which the CPU was not running, ie none of the sCPUi was active. Fig. 4 presents the Nexys4 DDR FPGA resource utilization for Cortex-M3 microcontroller, MicroBlaze IP, MIPS32 and nMPRA (4 sCPUi) based on hardware scheduler. Using Verilog 2001 HDL and Nexys4 DDR (Nexys A7) to design these timers (*reg [31: 0] Timer [0: NR_TASKS-1];*), the resource requirements (LUTs (Look-Up Tables) and FFs (Flip-Flops)) are insignificant compared to other implementations, such as MicroBlaze or ARPA-MT. The register-transfer level (RTL) equations implemented in the real-time event handling block are the following ("∘" AND logic, "+" OR logic):

ID_Static0 = (/sCPU0_ready ∘ sCPU1_ready ∘ /sCPU2_ready ∘ /sCPU3_ready ∘ /sCPU4_ready ∘ /sCPU5_ready ∘ /sCPU6_ready ∘ /sCPU7_ready) + (/sCPU0_ready ∘ /sCPU1_ready ∘ /sCPU2_ready ∘ sCPU3_ready ∘ /sCPU4_ready ∘ /sCPU5_ready ∘ /sCPU6_ready ∘ /sCPU7_ready) + (/sCPU0_ready ∘ /sCPU1_ready ∘ /sCPU2_ready ∘ /sCPU3_ready ∘ /sCPU4_ready ∘ sCPU5_ready ∘ /sCPU6_ready ∘ /sCPU7_ready) + (/sCPU0_ready ∘ /sCPU1_ready ∘ /sCPU2_ready ∘ /sCPU3_ready ∘ /sCPU4_ready ∘ /sCPU5_ready ∘ /sCPU6_ready ∘ sCPU7_ready)

ID_Static3 = (/sCPU0_ready ∘ /sCPU1_ready ∘ /sCPU2_ready ∘ /sCPU3_ready ∘ /sCPU4_ready ∘ /sCPU5_ready ∘ /sCPU6_ready ∘ /sCPU7_ready)

The logic for the case of 2, 4, 16 and 32 sCPUi can also be designed. For *muxi* signals the implementation of FPGA is based on the dynamic scheduler support.

ID_Dynamic0 = (/sCPU_ev0 ∘ mux1 ∘ /mux2 ∘ /mux3 ∘ /mux4 ∘ /mux5 ∘ /mux6 ∘ /mux7) + (/sCPU_ev0 ∘ /mux1 ∘ /mux2 ∘ mux3 ∘ /mux4 ∘ /mux5 ∘ /mux6 ∘ /mux7) + (/sCPU_ev0 ∘ /mux1 ∘ /mux2 ∘ /mux3 ∘ /mux4 ∘ mux5 ∘ /mux6 ∘ /mux7) + (/sCPU_ev0 ∘ /mux1 ∘ /mux2 ∘ /mux3 ∘ /mux4 ∘ /mux5 ∘ /mux6 ∘ mux7)

ID_Dynamic3 = (/sCPU_ev0 ∘ /mux1 ∘ /mux2 ∘ /mux3 ∘ /mux4 ∘ /mux5 ∘ /mux6 ∘ /mux7)

Fig. 5 shows the resource requirements both for the implementation of the real-time event handling module and the entire SoC (System on Chip) project. The implementation in the hardware of these timers does not generate significant additional consumption of resources.

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)
Top	14906	9052	1535	627	5762	14092	814	1535	1	4	31	15	1
Clock_Generator (PLL_200MHz_to_33MHz_66MHz)	0	0	0	0	0	0	0	0	0	0	0	3	1
LCD_Screen (LCD)	142	288	21	7	90	142	0	12	0	0	0	0	0
LEDs (LED)	0	16	0	0	6	0	0	0	0	0	0	0	0
Memory (BRAM_592KB_Wrapper)	14	70	0	0	34	14	0	3	1	0	0	0	0
MIPS32 (Processor)	14359	8358	1514	620	5578	13641	718	1376	0	4	0	0	0
ALU (ALU)	668	852	64	0	470	667	1	33	0	4	0	0	0
BranchAddress_Add (Add)	30	0	0	0	8	30	0	0	0	0	0	0	0
Compare (Compare)	0	0	0	0	3	0	0	0	0	0	0	0	0
Controller (Control)	51	140	0	0	59	51	0	1	0	0	0	0	0
CP0 (CPZero)	311	569	0	0	294	278	33	35	0	0	0	0	0
DataMem_Controller (MemControl)	2	32	0	0	7	2	0	1	0	0	0	0	0
EXALUImm_Mux (Mux2)	36	0	0	0	26	36	0	0	0	0	0	0	0
EXMEM (EXMEM_Stage)	784	0	0	0	339	600	184	0	0	0	0	0	0
EXRsFwd_Mux (Mux4)	9	0	0	0	7	9	0	0	0	0	0	0	0
EXRtFwdLnk_Mux (Mux4_2)	32	0	0	0	20	32	0	0	0	0	0	0	0
EXRtRdLnk_Mux (Mux4__parameterized0)	5	0	0	0	3	5	0	0	0	0	0	0	0
HazardControl (Hazard_Detection)	105	0	0	0	62	105	0	0	0	0	0	0	0
IDEX (IDEX_Stage)	1446	0	24	12	546	1225	221	0	0	0	0	0	0
IFID (IFID_Stage)	1574	0	1	0	788	1477	97	0	0	0	0	0	0
MEMWB (MEMWB_Stage)	878	0	0	0	435	740	138	0	0	0	0	0	0
MWriteData_Mux (Mux2_3)	32	0	0	0	16	32	0	0	0	0	0	0	0
nHSE (nHSE)	1149	42	0	0	714	1149	0	0	0	0	0	0	0
PC (Register)	93	128	0	0	73	93	0	0	0	0	0	0	0
PC_Add4 (Add_6)	19	0	0	0	23	19	0	0	0	0	0	0	0
PCSrcHSE_Mux (Mux2_4)	32	0	0	0	23	32	0	0	0	0	0	0	0
PCSrcStd_Mux (Mux4_5)	34	0	0	0	20	34	0	0	0	0	0	0	0
RegisterFile (RegisterFile)	3371	3968	1420	608	2336	3371	0	8	0	0	0	0	0
RegisterFilenHSE (RegisterFilenHSE)	3687	2625	5	0	1879	3643	44	734	0	0	0	0	0
TrapDetect (TrapDetect)	8	0	0	0	5	8	0	0	0	0	0	0	0
WBMemtoReg_Mux (Mux4_7)	32	0	0	0	22	32	0	0	0	0	0	0	0
WBMemtoReg_nHSE_Mux (Mux2_8)	32	0	0	0	21	32	0	0	0	0	0	0	0
Switches (Switches)	73	56	0	0	25	73	0	43	0	0	0	0	0
UART (uart_bootloader)	322	263	0	0	130	226	96	95	0	0	0	0	0

Fig. 5. Resources used to implement the SoC project that includes the processor with 4 sCPUi.

A FPGA circuit consists of three main elements: LUTs, FFs and routing channels. These constitutive elements, characteristic to the technology of programmable logic, are interconnected in order to form a flexible and state-of-the-art device.

Among the results which contain relevance and novelty, both nationally and internationally, we can mention: rapid switching of the context when moving from the execution of a task to another (one machine cycle); definition of a standard set of events; the implementation of a distributed and versatile interrupt system which allows the attachment of an interrupt to a single task, the implementation of timers and counters for performance measurement, etc.

V. CONCLUSIONS AND FUTURE WORK

HW-RTOS provides sCPUi management, mutexes, messages, time base events and interrupts. Regarding support for RTSs, practical tests have been added and described for the validation of the theoretical aspects. A real-time embedded system, like any other computing system, must be reliable and secure; moreover, they have been used in fields that can involve human security, and therefore imposing strict requirements.

The monitoring of the processor cycles is an important tool because it enables the monitoring and the improvement of the proposed processor performance; the performance achieved through this implementation is similar or even better than other implementations used for RTSs. The performance of the hardware accelerated RTOS implementation can be enhanced by designing a debug and trace module to enable better control of the system and a high diagnostic capability, providing system designers with flexible responses to the imposed requirements.

ACKNOWLEDGMENT

This work is supported by the project ANTREPRENORDOC, in the framework of Human Resources Development Operational Programme 2014-2020, financed from the European Social Fund under the contract number 36355/23.05.2019 HRD OP /380/6/13 – SMIS Code: 123847.

REFERENCES

[1] C. Koulamas,; M.T Lazarescu,. "Real-time embedded systems: Present and future," MDPI Electron. 2018, 7, 205, doi: 10.3390/electronics7090205.

[2] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin and R. Fong, "MetaWire: Using FPGA configuration circuitry to emulate a network-on-chip," in IET Computers & Digital Techniques, vol. 4, no. 3, pp. 159-169, May 2010. doi: 10.1049/iet-cdt.2009.0009.

[3] 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.

[4] V. G. Gaitan, N. C. Gaitan, and I. Ungurean, "CPU Architecture Based on a Hardware Scheduler and Independent Pipeline Registers," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 9, pp. 1661-1674, 2015. doi: 10.1109/TVLSI.2014.2346542

[5] I. Zagan, V.G. Găitan, "Hardware RTOS: Custom Scheduler Implementation Based on Multiple Pipeline Registers and MIPS32 Architecture," Electronics, 2019, 8, 211. doi: 10.3390/electronics8020211.

[6] Xilinx. VC707 Evaluation Board for the Virtex-7 FPGA User Guide. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf (accessed on28 August 2017).

[7] D. Andrews et al., "hthreads: A hardware/software co-designed multithreaded RTOS kernel", Proc. 10th IEEE Conf. Emerg. Technol. Factory Autom., pp. 331-338, Catania, Italy, Sep. 2005.

[8] M. Delvai, W. Huber, B. Rahbaran, and A. Steininger, "SPEAR – Design-Entscheidungen für den Scalable Processor for Embedded Applications in Real-time Environments", Tagungsband of Austrochip 2001, pp. 25–32, Vienna, Austria, Oct. 2001

[9] V. Yodaiken, C. Dougan, and M. Barabanov, "RTLinux/RTCore Dual-Kernel Real-Time Operating System," FSMLabs Inc. Technical Paper. [Online]. Available: http://www.yodaiken.com/papers/rtlpro.pdf

[10] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm, "The influence of processor architecture on the design and the results of WCET tools", Proceedings of the IEEE, vol. 91, no. 7, pp. 1038-1054, July 2003, doi: 10.1109/JPROC.2003.814618.

[11] C. Berg, J. Engblom, and R. Wilhelm, "Requirements for and design of a processor with predictable timing", seminar 03471 - Design of Systems with Predictable Behaviour, 2004, ISSN: 1862-4405.

[12] S. Roman, H. Mecha, D. Mozos and J. Septien, "Constant complexity scheduling for hardware multitasking in two dimensional reconfigurable field-programmable gate arrays," in IET Computers & Digital Techniques, vol. 2, no. 6, pp. 401-412, November 2008. doi: 10.1049/iet-cdt:20070060

[13] G. Ayers, "eXtensible Utah Multicore (XUM) project at the University of Utah," 2011-2012, [Online], Available: http://opencores.org/project,mips32r1, (Accessed: Sept. 2017).

[14] E.-E. Ciobanu, "The Events Priority in the nMPRA and Consumption of Resources Analysis on the FPGA," Advances in Electrical and Computer Engineering, vol.18, no.1, pp.137-144, 2018, doi:10.4316/AECE.2018.01017.

[15] J. Yiu, "System-on-Chip Design with Arm® Cortex®-M Processors," Arm Education Media, 2019, ISBN:978-1-911531-19-7.