Reducing Energy Consumption in Microcontroller-based Systems with Multipipeline Architecture

Cristian Andy Tanase
Faculty of Electrical Engineering and Computer Science
Stefan cel Mare University of Suceava
Str. Universitatii 13, 720229 Suceava, Romania

***Abstract*—Current mobile battery powered systems require low power consumption as possible without affecting the overall performance of the system. The purpose of this article is to present a multi-pipeline architecture implemented on a RISC V processor with 4 levels pipeline. Each thread has an assigned CLKSCALE registry that allows to use a clock with a lower or higher frequency, depending on the value written in the CLKSCALE registry. Depending on the importance and the need to be executed at a lower or higher speed each thread will enter into execution with its frequency given by CLKSCALE. It is known that each system has its own "real time". The notion of real time is very relative depending on the environment in which the system operates. Thus, if the system responds to external stimulus for a time that does not affect the operation of the whole, then we say the system is in real time. The system response can be quick or slow. It is important that this response does not lead to malfunction in operation. Therefore, certain threads can work at lower frequencies (those responding to slower external stimulus) and others must operate at high frequencies to allow quick response to fast external stimulus. It is known that the consumed power is directly proportional to the frequency of computing. Thus, the threads that do not require to run at maximum frequency, will consume less energy when they run. The entire system will consume less energy without affecting its performance. This architecture was implemented on a Xilinx FPGA ARTY A7 kit using the Vivado 2018.3 development tools.**

***Keywords*—Multi-pipeline register; RISC V (Reduced Instruction Set Computer); power consumption; multi-threading; FPGA (Field Programmable Gate Array); variable frequency**

I. Introduction

With the development of integrated systems, energy consumption has become a more important constraint in the RTL design. As these integrated systems become more sophisticated, they also need a higher level of performance. The task of satisfying the energy consumption and performance requirements of these embedded systems is a rather difficult task to ensure. One of the most used techniques of enhancing CPU performance is the use of ILP (Instruction Level parallelism) through pipeline technology. The instruction pipeline allows an increased clock frequency by reducing the amount of work to be performed for an instruction in each clock cycle [1].

A reduced energy consumption can increase the standby-time of the terminal which reduces the user annoyance related to recharging the battery too often. A reduced energy consumption could also mean that one can get the same standby-time as earlier but with a smaller sized battery, which reduces the overall size and weight of the terminal. A smaller battery is beneficial from an environmental aspect as well.

Energy consumed in CMOS devices is a product of time and power consumption and is measured in Joules (eq. 1). Power consumption in a CMOS device is consumed both statically and dynamically. Currently, the majority of the power is consumed dynamically, but devices implemented using future process technologies will most likely consume as much static as dynamic power consumption due to increased leakage currents (eq. 2). As can be seen from eq. 3, dynamic power is a function of the voltage level (V_{dd}), frequency (f), capacitive load (C), and the activity factor (α). The activity factor represents the number of transitions between a logic zero and a logic one, which corresponds to charging of capacitances. One observation is that a near-cubic reduction of dynamic power consumption can be achieved by reducing the voltage and frequency. Dynamic power consumption can also be lowered by reducing or eliminating the transistor switching activity. Another source of dynamic power consumption is the current dissipated from short circuits in transistors during switching. Short circuit currents are dissipated when a logical value of a transistor is in the process of doing a transition of its output. During this transition there is a small period of time, where there is a direct path from the supply voltage and the ground, which results in dissipated currents (see eq. 4) The static power consumption comes from non-ideal switch behaviour of transistors, thus the transistors leak currents (see eq. 5) [2] -[12].

$$Energy = Power \cdot Time \tag{1}$$

$$P_{total} = P_{switch} + P_{shortcircuit} + P_{leakage} \tag{2}$$

$$P_{switch} = V_{dd}^2 \cdot f \cdot C_L \cdot \alpha \tag{3}$$

$$P_{shortcircuit(sc)} = V_{dd} \cdot I_{sc} \tag{4}$$

$$P_{leakage} = V_{dd} \cdot I_{leakage} \tag{5}$$

II. Related Work

In [13] the authors propose an runtime environment for next-generation dual-core MCU platforms. These platforms complement a single-core with a low area overhead, reduced design margin shadow-processor. The runtime decreases the

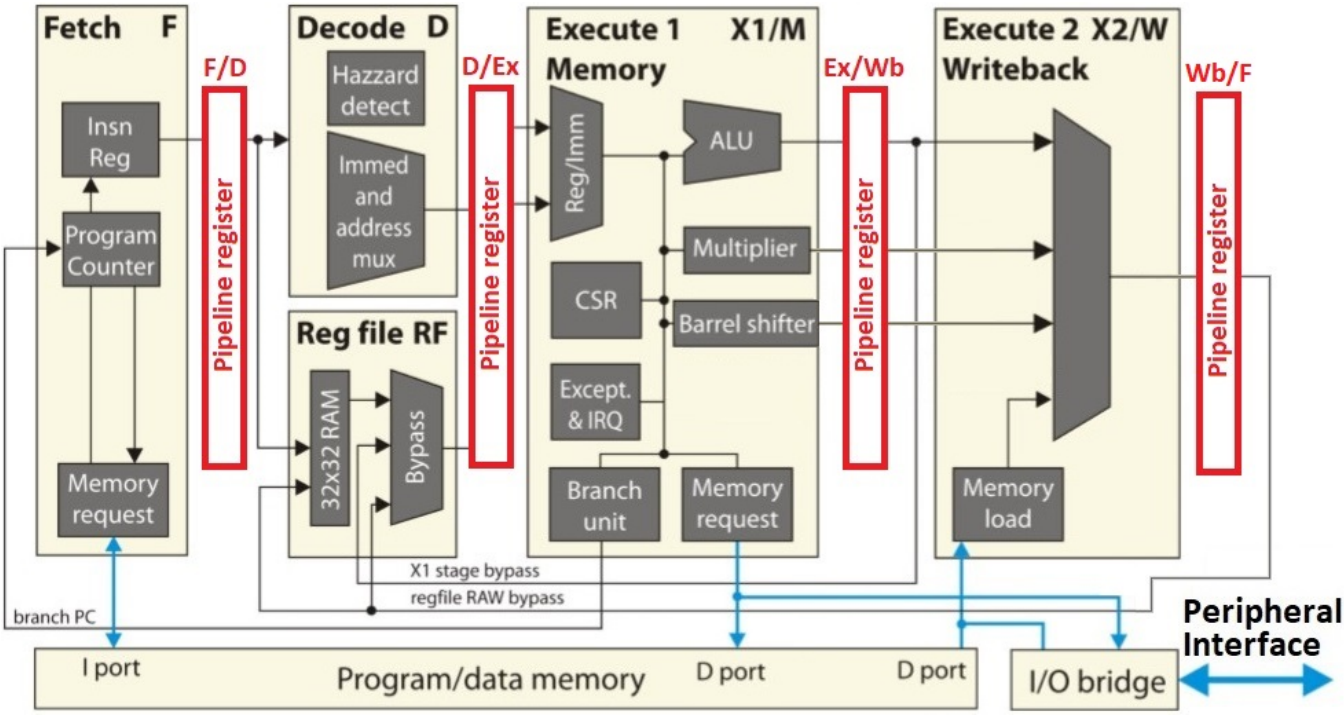


Fig. 1. RISCV Processor (Block Diagram) [17]

overall energy consumption by exploiting design corner heterogeneity between the two cores, rather than increasing the throughput. This allows the platform's power envelope to be dynamically adjusted to application-specific requirements. Depending on the ratio of core to platform energy, total energy savings can be up to 20%.

In [14] the authors realized a single-ISA (Instruction Set Architecture) heterogeneous multi-core architecture, including four Alpha cores and a MIPS R4700 (Microprocessor without Interlocked Pipelined Stages). The allocation of tasks among cores is integrated as part of the operating system.

In [15] the authors show that at NTC (Near-Threshold Voltage Computing - the supply voltage is only slightly higher than the transistor's threshold voltage), is a promising approach to reduce the energy per operation and a simple chip with a single V_{dd} domain can deliver a higher performance per watt than one with multiple V_{dd} domains.

Compared to high-end systems, there has been very little attention paid to task allocation/scheduling on low-cost, limited performance systems. The closest work in this domain would be [16], which focuses on optimal resource management for control tasks in MCUs using a minimal real-time kernel.

III. Background

The proposed architecture was implemented on a RISC V core with four levels pipeline, presented in Fig. 1 [17].

RISC V employs a modified Harvard architecture: code and data reside in a shared 32-bit memory space, but are accessed through separate memory interfaces. Instructions are executed by a four-stage, single-issue pipeline, shown in Fig. 1 and consisting of the following stages:

1) Fetch(F), calculating the address of the next instruction and requesting it from the code memory;
2) Decode(D), which also computes the immediate values (sign-extensions and bit reordering), pre-computes the operand values for the Execute1/Memory(X1/M) stage and manages the hazards by inserting empty instructions into the Execute stage;
3) Execute1/Memory(X1/M), which executes most of the instructions, generates memory addresses and issues memory read/write requests;
4) Execute2/Writeback(X2/W), which completes execution of Load, Multiply and Shift instructions and writes the result back to the CPU registers.

In parallel to the D stage, there is a Register File (RF), hosting the 32 architectural registers(x0 - x31). The RF is built on top of two FPGA RAM blocks, providing two read ports and one write port, with a single clock cycle latency. The core includes a platform-optimized barrel shifter and multiplier (both with 2 cycle latency). Most of the instruction results are bypassed to achieve interlock-free execution of dependent instructions, either by the Read-After-Writer (RAW) bypass path in the RF or after the X2/W stage. The RISCV Control and Status Registers (CSRs), interrupts and a limited set of exceptions, although the interrupt CSR layout has been simplified to minimize the occupied FPGA area. There is another block to manage the exceptions and interrupts.

IV. Experimental Resources

For this project the author used ARTY A7 board, Vivado 2018.3 and Vivado HLS tools. The ARTY A7, is a ready-to-use development platform designed around the Artix-7

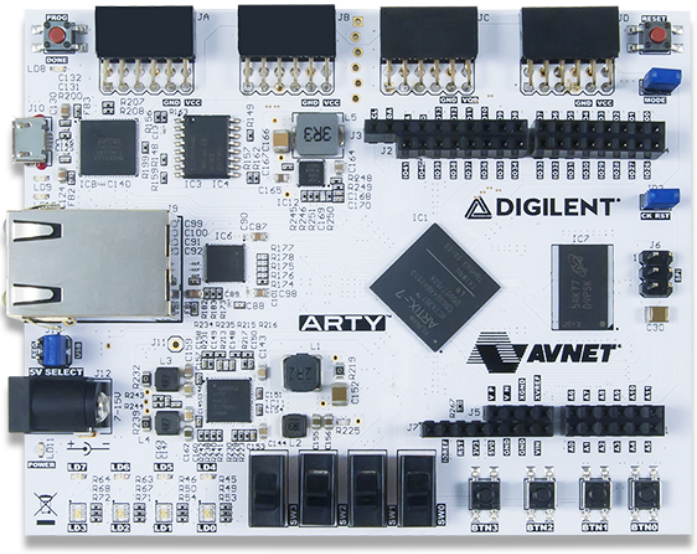


Fig. 2. ARTY A7 Board

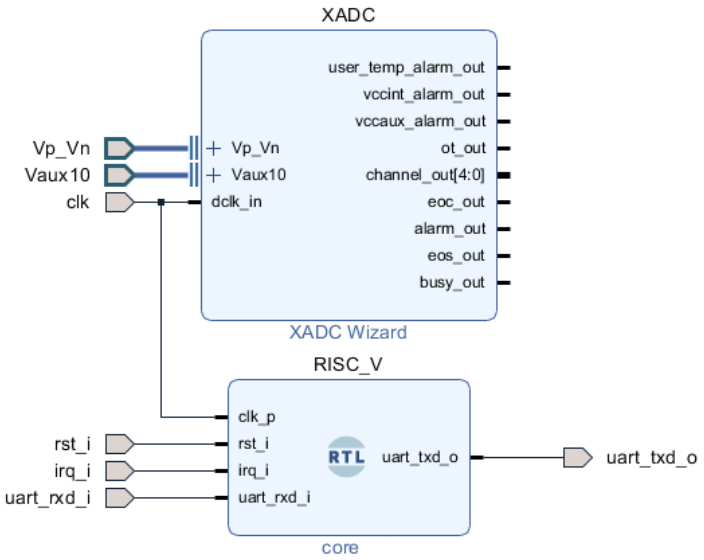


Fig. 3. System Block Diagram

Field Programmable Gate Array (FPGA) from Xilinx (Fig. 2) with features: Artix-7 XC7A35T-L1CSG324I FPGA, On-chip analog-to-digital converter (XADC), Programmable over JTAG and Quad-SPI Flash, 256 MB DDR3L with a 16-bit bus @ 667 MHz, 16 MB Quad-SPI Flash, 10/100 Mb/s Ethernet, USB-UART Bridge, Switches, Buttons, RGB LEDs, Four Pmod interfaces (32 I/O), Arduino/ChipKit "shield" connector (49 I/O).

V. Experimental Work

A. Variable Clock Generator

The hardware system implementation was done using the Block Design facility in Vivado 2018.3. The system block diagram is shown in Fig. 3. It contains an analog-digital converter that allows to determine the current consumption of the entire system. Channel 10 of the converter is connected to an external circuit (INA199A1) of the current measurement consumed by the FPGA. The circuit generates 500mV/A.

RISC V processor presents two external communication pins (uart_rxd/uart_txd) and a pin for interruption (irq). The instruction and data memory is implemented inside of the CPU. All registry blocks have been multiplied: pipeline (F/D, D/Ex, Ex/Wb, Wb/F) and the registers file RF. This allows to retain, at some point, a maximum of four independent threads. The context of each thread is retained in a set of registers. The change of context is made through a switching of registers in a single clock period. Also, each set of registers assigned to a thread is piloted by a programmable clock through scaling registers (CLKSCALE). These registers allow the generation from the system clock of four independent CLK signals and proportionate to the values written in the scaling registers. Each thread will work with a frequency given by its own clock register (Fig. 4).

The multi-pipeline architecture is shown in Fig. 5. It can be seen from the Fig. 1 multiplication of pipeline registers and registers file RF. There is also the clock generation block with the four CLKSCALE registers. CLKSCALE registers

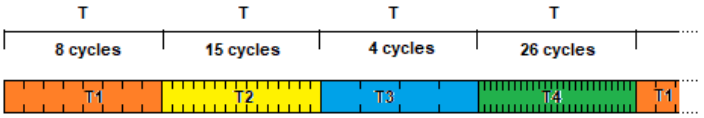


Fig. 4. Schematic Presentation of 4 Threads Running

were mapped in free space of the CSR registers (Control and Status Registers) to addresses 0x0E00-0x0E03. In Fig. 5, the clock generator block diagram is also observed. Its main components are the registry for dividing the clock and the thread selection unit. When a specific thread is selected, the clock signal assigned to it is also activated. That clock is proportional to the value written in the CLKSCALE register dedicated to the thread.

Listing 1: CLKSCALE registers selection.

```
/*************************************/
    wire wr_genclk;
    assign wr_genclk = (dm_addr_o == 32'h0E00)
    || (dm_addr_o == 32'h0E01)
    || (dm_addr_o == 32'h0E02)
    || (dm_addr_o == 32'h0E03);

    clk_gen clkgen (
          .clock(clkin),
          .reset(1'b0),
          .wr(wr_genclk && dm_store_o),
          .inter(irq_i),
          .data(dm_data_s_o),
          .addr(dm_addr_o[1:0]),
          .selchn(slct),
          .selchnout(select),
          .clkout(clk_i)
);
/*************************************/
```

The Listing 1 also shows the Verilog interface with the clock generation block. This block shows three signals (*wr*,

data, and *addr*) used to write the four scaling registers. It also presents an interruption signal (*inter*) that announces this block that an interruption occurred. The signals (*selchn* and *selchnout*) are used in the selection of the registers set assigned to each thread. The *clkout* signal is the clock signal that drives the threads.

The way to access CLKSCALE registers in C is done using pointers:

```
  volatile int *CLKSCALE0 = (int*)0x0E00;
*CLKSCALE0 = 0x03ff;
```

The default value of the CLKSCALE registers is 0. That is, the thread assigned to that register runs at full frequency. The clock scaling factor is the value of CLKSCALE + 1. In Listing 2 is presented the C code written for the first thread. The program sends a message through USART and sets the frequency with which this thread will run as CLK_MAX/0x0400. The other source programs for threads 1, 2, and 3 are similar. It differs only the message sent by USART and the value written in CLKSCALE. The program runs in an infinite routine and copies the message from the **hello* address to the **TX_REG* address from where it is sent to USART.

Each thread is called periodically. It will send your own message to USART and run it with its own frequency (Fig. 4).

Listing 2: C code for thread 0.

```
/************************************/
  const char *hello="Hello sCPU0";

  volatile int *TX_REG = (int*)0x0F00;
  volatile int *CLKSCALE0 = (int*)0x0E00;

void main()
{
  char *s = (char*)hello;
  *CLKSCALE0 = 0x03ff;

  while(1){
    s = (char*)hello;
    TX_REG = (int*)0x0F00;
    while(*s) {
        *TX_REG++ = *s++;
    }
  }
}
/************************************/
```

The result of the four thread execution is shown in Fig. 6 (the most important signals). You can see the thread selection signals, the values written in the CLKSCALE registry, the clock signals and the data sent by USART (Fig. 6a). The writing of the CLKSCALE registers is made when the SELECT_TH signal takes the corresponding value. That means that the code is running for each thread in part: e.g. when the thread 1 is active it writes CLKSCALE1 with the value 0x10, etc.

Fig. 6b shows how to switch the thread 0 with thread 1. You can also see the change of the clock. When the thread changes, it is expected to finish the last clock period in thread 0 and start the first clock period in thread 1. The clock's frequency for thread 1 is 17 times less than the frequency of thread 0 (0x10 + 1).

As in Fig. 6c, switching clocks happens after changing the threads at the end of the last clock period.

Listing 3: ASM code for thread 0. (fragment)

```
/************************************/
38:     fec42783    lw   a5,-20(s0)
3c:     00178713    addi a4,a5,1
44:     0007c603    lbu  a2,0(a5)
48:     08802783    lw   a5,136(zero)
4c:     00478693    addi a3,a5,4
50:     08d02423    sw   a3,136(zero)
54:     00060713    mv   a4,a2
58:     00e7a023    sw   a4,0(a5)
5c:     fe842783    lw   a5,-24(s0)
60:     00178793    addi a5,a5,1
64:     fef42423    sw   a5,-24(s0)
68:     fec42783    lw   a5,-20(s0)
6c:     0007c783    lbu  a5,0(a5)
70:     fc0794e3    bnez a5,38 <main+0x38>
74:     fadff06f    j    20 <main+0x20>
/************************************/
```

In Listing 3 a fragment of the ASM code generated in the compilation is presented. Fig. 6d shows the last instruction in the thread 0 performed when switching to thread 1. When the thread returns to execution 0, it resumes its execution from the point where it was suspended. (red circles)

In Fig. 7, the result of running the four threads on the multi-pipeline RISC V system is presented. Each thread runs at different frequency but sends its message to TX USART.

Considering that interruptions must be taken into consideration immediately after their appearance, the problem that arises when an interruption occurs and the current thread in execution operates at a small frequency, it must accelerate the execution of the last instruction and activate immediately the thread dedicated to that interrupt.

In Fig. 8, the response time of the system to the occurrence of an interruption is presented. It is noted that the interruption signal *irq_i* is activated. Immediately the execution of the last instruction is accelerated in thread 1 (0x6c6...) and proceed to the execution of the interruption handler (in our case the thread 0). The worst case (Fig. 8a), is when the first instruction in the interruption handler is executed after three cycles of the system clock (*clkin*). The most favorable case is shown in Fig. 8b and it appears when the first instruction in the interrupt routine is executed after two cycles.

The Fig. 9 shows the flow chart of the clock generation block, implemented in the verilog and used as IP block. The program checks if the interrupt is set. If so, make the output *clkout=0* and *clkout=1* with *contor=CLKSCALE0* (the default frequency for interrupt routine). Otherwise, depending on the values from CLKSCALE registers, is generate clkout signal.

The code handles the situation when an interruption occurs or when it is inactive. When an interruption occurs, it should be addressed immediately. If, at the time of interruption, the clock signal is on high level then it must be switched to low

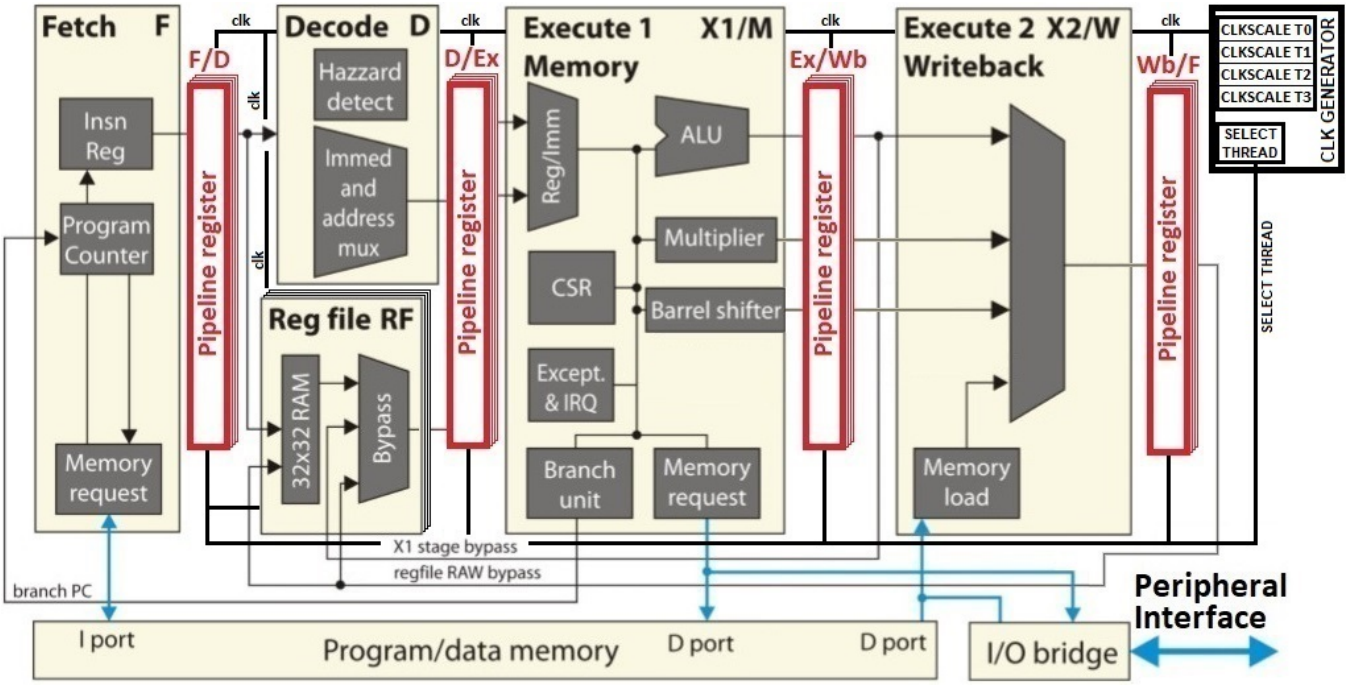


Fig. 5. Multi-Pipeline Architecture with CLKSCALE Registers for each Pipeline Registers Set

level, after which the first cycle of fetching instructions from the interruption handler and the change of frequency are started (Fig. 8a). If, when the interruption occurs, the clock signal is on low level, then at the next cycle the first instruction from the interruption handler is fetched and the frequency is changed (this is the most favorable situation Fig. 8b).

In the situation when no interruption occurs, then when the context changes it waits until the last instruction from the previous thread is executed, and the following execution instructions in the active thread are started when the frequency changes. The frequency is changed based on the counters read from the CLKSCALE registry.

B. Measuring the Energy Consumed

The energy used by the system was measured with the XADC IP block (Fig. 3). Depending on the current consumed by the entire system implemented on the FPGA, the voltage on the XDAC Channel 10 (Vaux10) changes. At each 500 mV measured on Vaux10, the FPGA consumes 1A.

According to equations 1-5 the use of lower frequencies should result in the decrease of the current consumption.

Several measurements have been made with various CLKSCALE values. Initially, measurements of the current consumed with identical values of the four CLKSCALE registers were made, with values ranging from 0x0fff to 0x0000. The first lines in Table I show the current consumed at these CLKSCALE values. The last lines in this table show the current consumed with random values of the CLKSCALE registry.

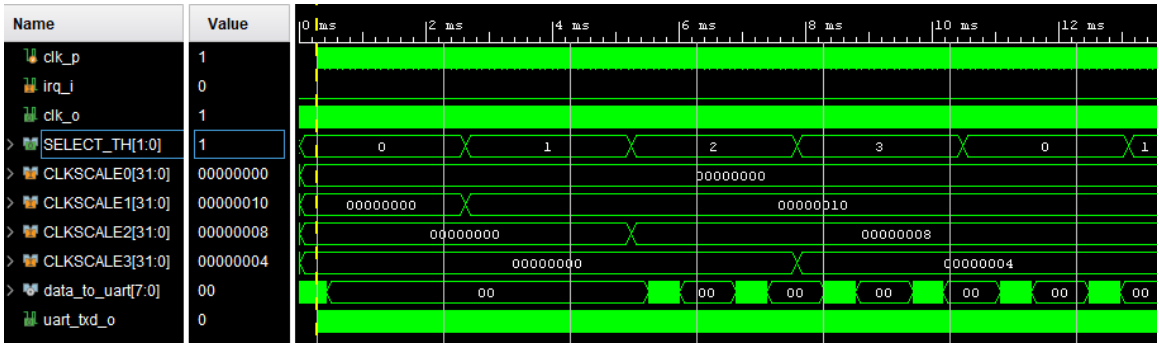
You can see a decrease in the current consumed when the semi-processors operated at lower frequencies. If this does not affect the functioning of the whole system as a whole then we can say that we have achieved a reduction of the energy consumed without low performance. From the Table I you can see that between the maximum current and the minimum current consumed it is a ratio of about 30%. If it is also taken into account the current consumed by the XADC circuit that has been implemented only to perform measurements, then we can talk about a yield greater than 30% of saved energy.

TABLE I. CURRENT CONSUMED ACCORDING TO THE VALUES WRITTEN IN CLKSCALE REGISTERS

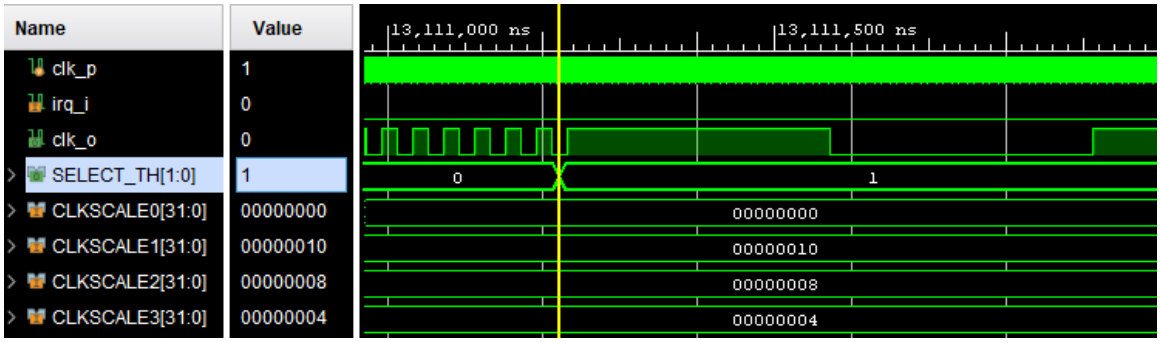
Vaux10	A	CLKSC0	CLKSC1	CLKSC2	CLKSC3
0.012V	24mA	0x0fff	0x0fff	0x0fff	0x0fff
0.013V	26mA	0x3ff	0x3ff	0x3ff	0x3ff
0.014V	28mA	0x0ff	0x0ff	0x0ff	0x0ff
0.014V	28mA	0x00f	0x00f	0x00f	0x00f
0.015V	30mA	0x003	0x003	0x003	0x003
0.016V	32mA	0x002	0x002	0x002	0x002
0.017V	34mA	0x001	0x001	0x001	0x001
0.018V	36mA	0x000	0x000	0x000	0x000
0.014V	28mA	0x010	0x0f0	0x010	0x001
0.014V	28mA	0x000	0x010	0x100	0x003
0.016V	32mA	0x000	0x004	0x003	0x000
0.013V	23mA	0x000	0xfff	0x0ff	0xfff

VI. CONCLUSION

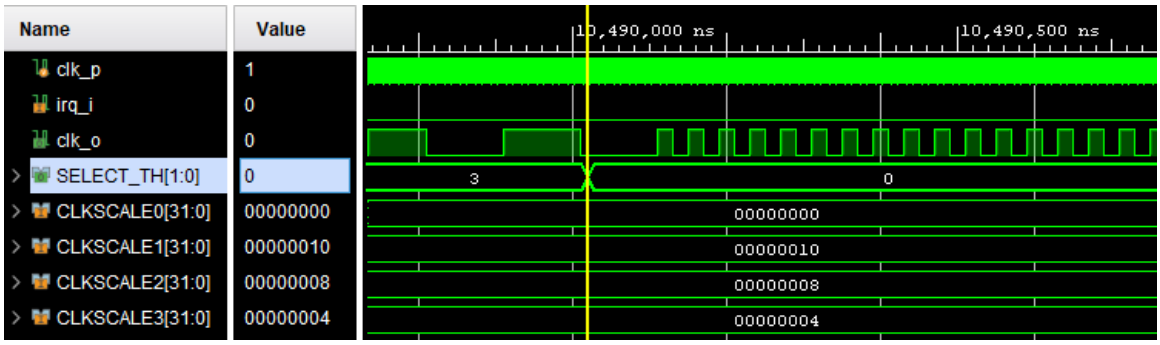
The conclusions drawn from this study are as follows: by multiplying the pipeline registers and the registry file of a RISC V architecture, four semi-processors using the same hardware resources have been obtained. Each semi-processor runs a thread at different frequencies. Switching between threads is done in a clock cycle due to the multiplication of pipeline



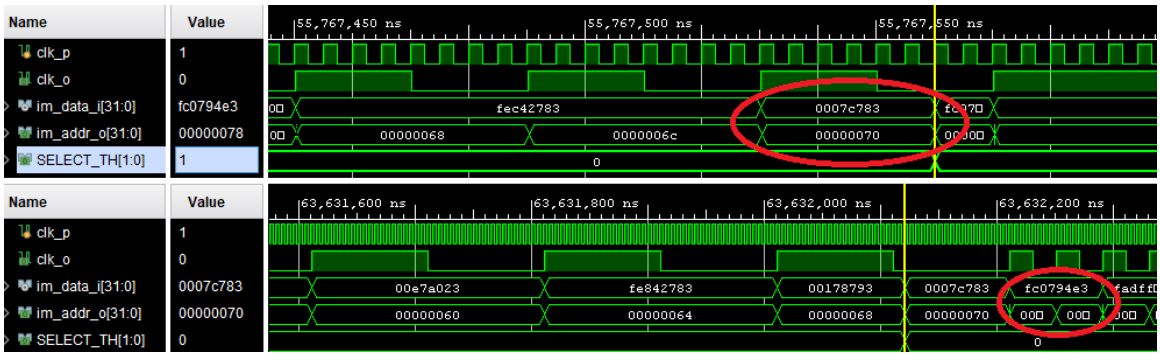
(a) Select Thread Signal.



(b) Clock Signal Changing (High to Low)



(c) Clock Signal Changing (Low to High)



(d) Last and Next Instruction in Thread 0.

Fig. 6. The Most Important Signals.

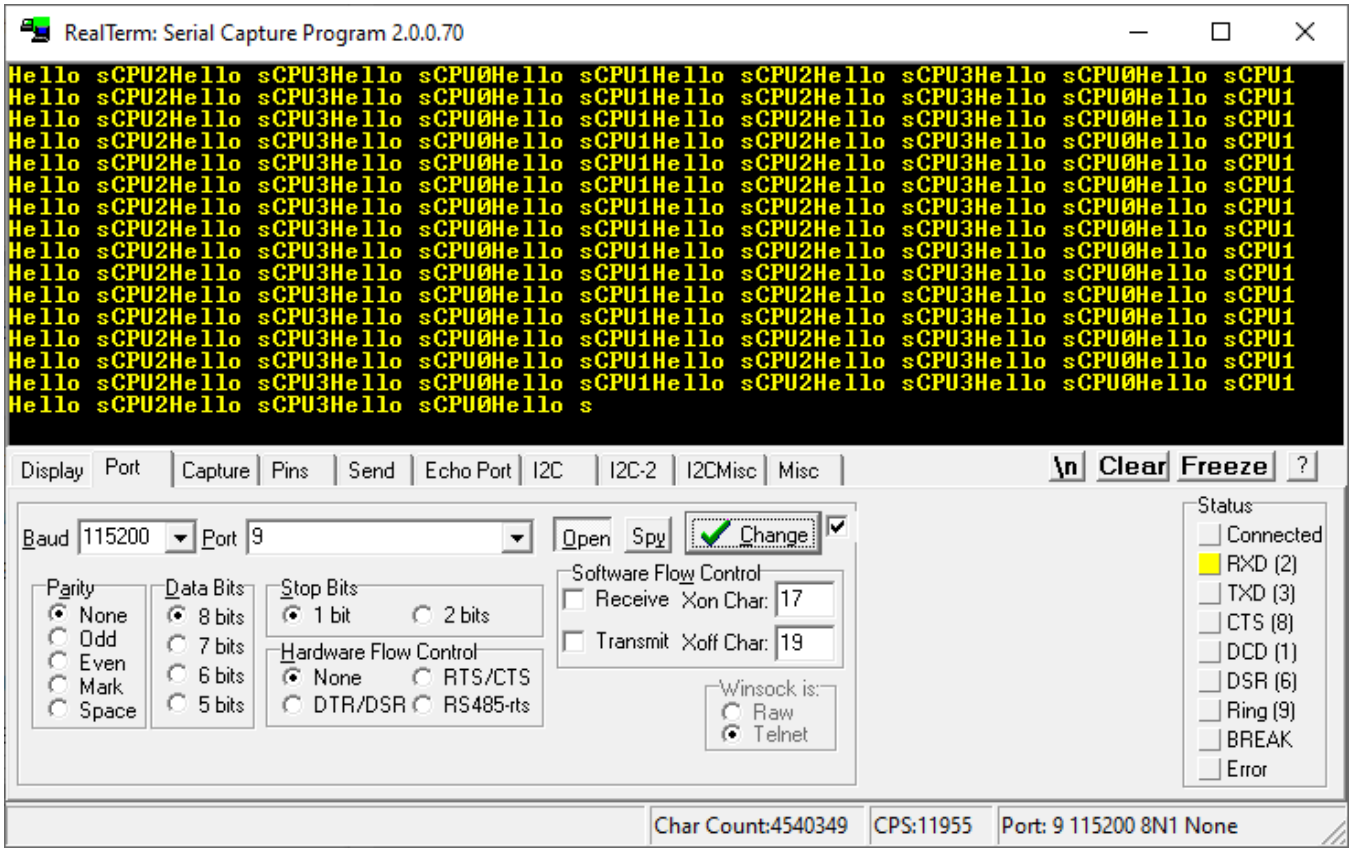
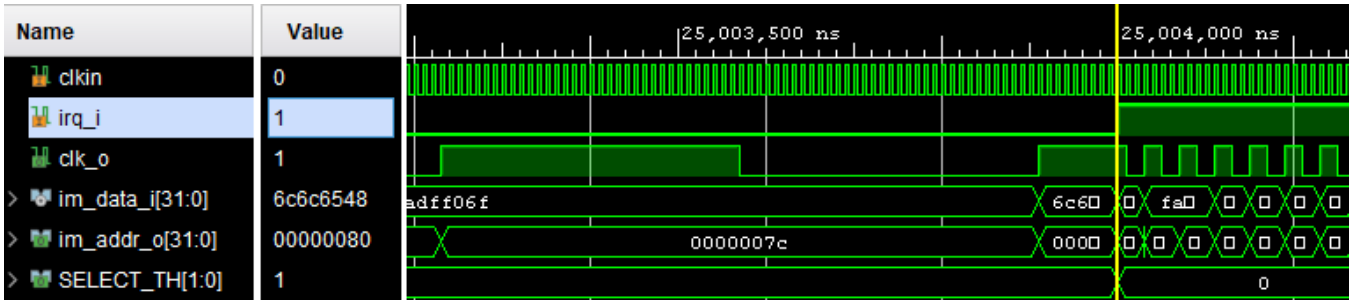
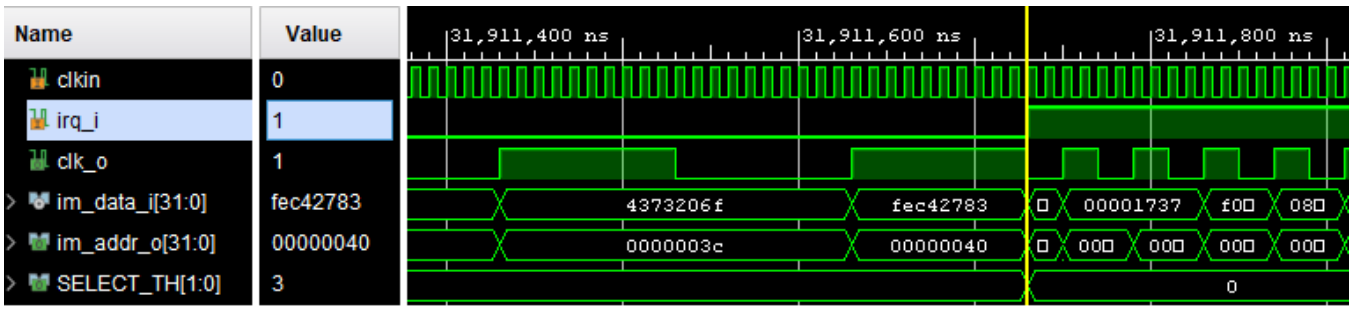


Fig. 7. RealTerm Terminal with Messages from all Threads



(a) Worst Case.



(b) Best Case.

Fig. 8. Interrupt Response Time.

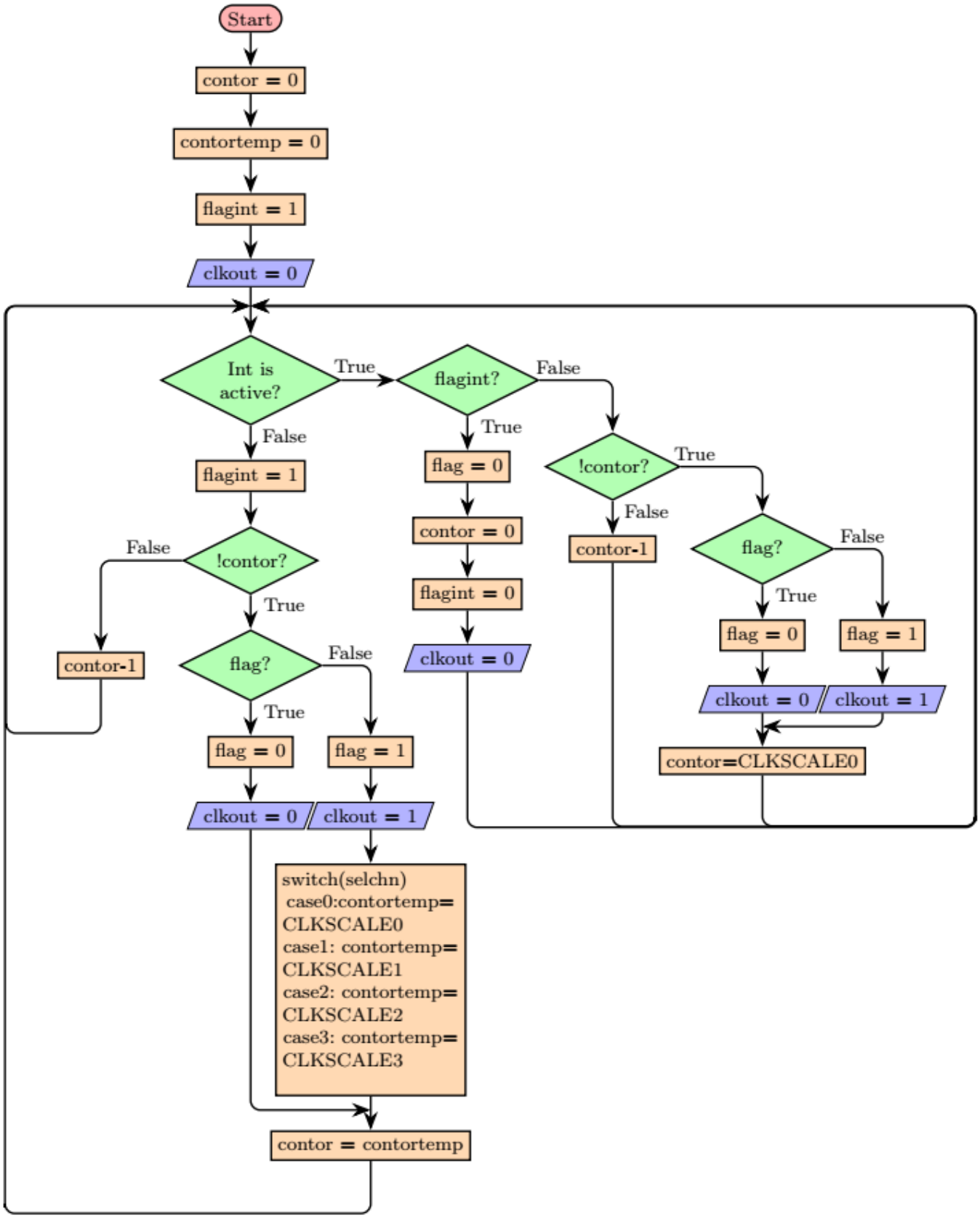


Fig. 9. Flow Chart of Clock Generator Code

registers. Depending on the needs of real-time responses of each thread, the Hard Real Time threads will work at high frequencies and the Soft Real Time threads will work at lower frequencies. In this way, a lower energy consumption will be achieved due to the fact that the energy consumed is proportional to the system's working frequency.

As future work the author wants to create an auto-tuning architecture adapted to the priorities and response time required for each task. If a task has a much shorter response time than a response time that does not generate errors in the operation of the system, it will decrease its execution frequency until it reaches close to this time without exceeding it. It will execute the task in real time with a minimum energy consumption. Thus the author aims to implement a block that detects these malfunctions due to an inadequate response time.

ACKNOWLEDGMENT

The infrastructure used for this work was partially supported by the project "Integrated Center for research, development and innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for fabrication and control", Contract No. 671/09.04.2015, Sectoral Operational Program for Increase of the Economic Competitiveness co-funded from the European Regional Development Fund.

REFERENCES

[1] Ian Finlayson, Gang-Ryung Uh, David Whalley and Gary Tyson, *Improving Low Power Processor Eficiency with Static Pipelining*, 15th Workshop on Interaction between Compilers and Computer Architectures, 2011.

[2] S. Borkar, *Design challenges of technology scaling*, In IEEE Micro, Vol. 19, Issue 4, pp. 23-29, 1999.

[3] M. Broersma, *Intel chip not ready for the cool crowd*, CNET Tech News, http://news.com.com/ 2100-1001_3-271443.html

[4] D. Brooks et. al., *Wattch: A Framework for Architectural-level Power Analysis and Optimizations*, In Proceedings of the 27th International Symposium on Computer Architecture, pp. 83-94, 2000.

[5] M. Edahiro et.al., *A single-chip multiprocessor for smart terminals*, In IEEE Micro, Volume 20, Issue 4, pp. 12-20, July 2000.

[6] M. Fleischmann, *Longrun Power Management*, Transmeta Corporation, January 2001.

[7] R. Fromm et al., *The Energy Efficiency Of Iram Architectures*, In Proceedings of IEEE International Symposium on Computer Architecture, pp. 327-337, 1997.

[8] L. Geppert, T. S. Perry, *Transmeta's magic show*, In IEEE Spectrum, Vol. 37, Issue 5, pp. 26-33, 2000.

[9] R. Gonzalez, M. Horowitz, *Energy dissipation in general purpose processors*, In IEEE Journal of Solid-State Circuits, Volume 31, Issue 9, pp. 1277-1284, September 1996.

[10] K. Flautner, N. S. Kim, S. Martin, D. Blaauwm, T. Mudge, *Drowsy Caches: Simple Techniques for Reducing Leakage Power*, In Proceedings of the International Symposium on Computer Architecture (ISCA-29), Anchorage, Alaska, 2002.

[11] T. R. Halfhill, *Transmeta breaks x86 low power barrier*, Microprocessor Report, pp.9-18, February 2000.

[12] N. S. Kim et. al., *Leakage Current: Moore's Law Meets Static Power*, In IEEE Computer, Volume 36, Issue 12, pp. 68-75, 2003.

[13] A. Gomez1, C. Pinto, A. Bartolini et. al., *Reducing Energy Consumption in Microcontroller-based Platforms with Low Design Margin Co-Processors*, in Proc. DATE, 2015.

[14] R. Kumar, K. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, *Processor power reduction via single-ISA heterogeneous multi-core architectures*, In IEEE Computer Architecture Letters, vol. 2, no. 1, 2003.

[15] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, *EnergySmart: Toward Energy-efficient Manycores for Near-Threshold Computing*, In ACM HPCA, 2013.

[16] R. Marau, P. Leite, and M. Velasco, *Performing flexible control on lowcost microcontrollers using a minimal real-time kernel*, In IEEE Trans. Ind. Informat, vol. 4, no. 2, 2008.

[17] T.Włostowski,J.Serrano, *Developing Distributed Hard-Real Time Software Systems Using FPGAs and Soft Cores*, Proceedings of ICALEPCS2015, Melbourne, Australia-Pre-Press Release23-Oct-2015.