# LP805X: A Customizable and Low Power 8051 Soft Core for FPGA Applications

T. Lobo, S. Pinto, V. Silva, S. Lopes, J. Cabral, A. Tavares
Centro Algoritmi – University of Minho
Portugal
{tiago.castro, sandro.pinto, vitor.silva, sergio.lopes, jorge.cabral, adriano.tavares}@algoritmi.uminho.pt

S. Yoowattana, W. Sritriratanarak, M. Ekpanyapong
Asian Institute of Technology
Thailand
{s.yoowattana, w.sritriratanarak, mongkol}@ait.ac.th

*Abstract-* — **In today's advanced technological age, embedded and real time systems have become ubiquitous, covering a wide range of applicability. As a result there is an ever growing need for low power capabilities along with reasonable performance, thus presenting a virtual demand for power-aware devices. The purpose of this work was leveraging key techniques and technologies such as design laziness, componentware design, generative design, separation of mechanism and policy, voltage-frequency island, state encoding, clock gating, and operand isolation, and then investigating their effects while designing an energy and power-conscious microcontroller based on 8051 ISA (Instruction Set Architecture) without disregard to silicon area, required application functionalities and performance. Simulations show that our purposed 2 stage pipeline system with built-in hybrid scheduler operates at 230µW@1MHz.**

**Keywords— Processor design, power-aware, design laziness, componentware, generative design, voltage-frequency island, power and clock gating.**

## I. Introduction

For most electronic systems embedding microcontroller being developed today, the emphasis on active low-power management, performance, area, reliability and robustness is increasing and so, becoming critical issues. Each of these systems is unique and highly specialized to certain hardware and application domains, and among them are the following: (1) consumer, wireless and handheld devices, (2) home electronics and (3) tethered electronics [1]. With such a pervasive pressure to reduce power and increase functionality and performance, a holistic methodology that is able to design complete systems from the ground up and deliver a total platform for system integration, applications development, and system validation, represents the future of EDA [1, 2]. Thus, moving forward on the energy conservation path requires a careful management of the interactions between the different layers of abstraction as well as performing a global tradeoff analysis [2]. As the design effort in managing power and performance metrics has ramifications for engineering productivity (i.e., it impacts schedules and risk), a methodology will be partially described for small embedded system processor design supported by a development ecosystem driven by key techniques and technologies such as [1, 3, 4, 5, 6, 7]:

1. **Voltage-Frequency Island** to dynamically change the clock frequencies and potentially voltages of the selected functional blocks driven by dynamic workload;
2. **State Encoding** to reduce the switching activity by shortening the hamming distance between subsequent executed instructions;
3. **Clock Gating** to disconnect the flip-flops clock from the clock tree and reducing power both from the flip-flops and the clock tree;
4. **Power gating** to shut down the power at non-necessary blocks to reduce the leakage power;
5. **Asynchronous core** to reduce both the clock skew problem and the clock tree which normally requires a very large silicon area with the associated increases in power dissipation and limits the microcontroller speed;
6. **Operand Isolation** to reduce power dissipation in datapath blocks controlled by an enable signal that freezes their inputs when their output are not needed;
7. **Separation of Mechanism and Policy** to improve system's flexibility in response to system designer requirements as well as to leverage reusability and stability, instead of hardwiring policy and mechanism together;
8. **Componentware, Generative and Lazy design** to promote reuse creating modular structures for the 8051 soft core making it a good prototyping vehicle and thus, providing system designers a conduit to quickly experiment potential processor configuration and optimize the soft core for a particular use (e.g., using the minimum set of needed components), without requiring detailed knowledge of the whole system. Each functional unit was developed as a separate module and modeled using a combination of Verilog code and Xpand template [8].

Outline Due to the limited space, the focus will be only on the design of a tailored 8051-based processor as part of a holistic system-level design methodology for small embedded systems design that leverages semantic interoperability between adjacent system abstraction layers to fully enable power, performance and area convergence. The semantic interoperability between abstraction layers from application down to processor layer is modeled ontologically. Section II briefly presents some similar works. Section III presents the key features of LP805X, describes the detailed design of each module, and the integration of top-level module (i.e., the

LP805X core) by individual instantiation of previous designed modules. Section IV starts with the identification of the most power-hungry components and then describes the techniques adopted for power optimization at architectural level. Section V presents and discusses some experiments carried out to assess LP805X's behavior in terms of logical response, performance, power consumption and energy efficiency. Section VI concludes.

## II. RELATED WORK

The choice of the 8051 microcontroller is due to its high popularity as its CISC architecture became an industry standard used in several small embedded systems, and the accumulated experience using it in several in-house developed projects. There are huge amounts of work related to power-conscious processor design and this proposal is a fusion of ideas and techniques applied in [3, 4, 5] and extended to leverage a laziness design paradigm of microcontroller components or modules.

Chang-Jiu Chen et al. [4] proposed a novel pipelined asynchronous 8051 microcontroller as an answer to low power, reliability, and robustness issues. The 8051 microcontroller is implemented with Balsa language which is a CSP-based asynchronous HDL and then synthesized into Xilinx netlist by the Balsa synthesis tool.

Francesco Iozzi, et al. [3] described an 8051 IP core applying several RTL techniques such as state encoding, clustered clock gating and operand isolation for minimizing the switching activity while avoiding performance loss and preserving the reusability of the macrocell.

Sakina [5] described an 8051 microcontroller soft core in the Verilog HDL with each functional developed as a separate module, and tested for functionality using the open-source VHDL Dalton model as benchmark. Although this work doesn't focus on power reduction, it tried to promote some modularity with the modules integrated to operate as concurrent processes in the 8051 soft core.

## III. III. LP805X FEATURES AND COMPONENTIZING

In spite of the potentially low-power operation of asynchronous core compared to the synchronous one due to its hazard-free (i.e., no energy wasted in spurious transitions) and event-driven properties (i.e., only operating parts consume energy) [7], the focus here will be only on the latter as the former is still a work in-progress. The proposed synchronous 8051 soft core, as shown in Fig. 1, should be modular and customizable to applications requirements, and the following key features were supported:

1. Compatibility with 8051 standard ISA (Instruction Set Architecture) and enough flexibility to be customizable according to the system designer needs;
2. Up to 256 bytes of internal RAM (Random Access Memory) should be addressable, according to 8051 ISA and with SFR registers added as needed;

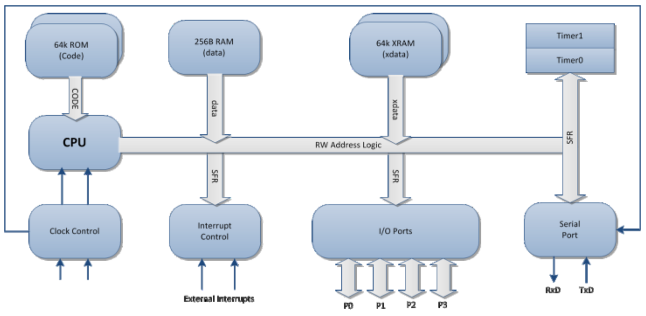


**Fig. 1 LP805X abstract block diagram**

3. Up to 64 KB of program memory and external memory should be addressable, and allowing the use of a hybrid model integrating on-chip and off-chip memory;
4. At least four 8-bit I/O ports, bit-addressable and bi-directional which can be multiplexed for alternate functions;
5. Programmable multi-priority interrupt with individual interrupt line for each device;
6. A power control unit capable of controlling statically and dynamically several parts of the microcontroller;
7. Communication between different regions of the microcontroller possibly working at different voltage and frequency levels;
8. Reduce to the minimal the number of clock cycles required for each instruction as it is directly proportional related to the dissipated power. All SFRs are part of the same bus which is shared by all three types of peripherals: (1) internal to external and external to internal signal converters, (2) stub controller for internal signals with no external interface and (3) hybrid peripherals which combine the two previous functions.

### A. *The CPU: Control Unit and 2-Stage Pipelined Datapath*

Although pipelining had been known as a popular technique to increase the processor's throughput, it is also used today to reduce power consumption as a pipelined execution unit presents a shorter stage delay than a non-pipelined execution unit [7]. It is therefore possible to work at the same operating frequency while reducing the supply voltage which helps to save a lot of dynamic power. However, the growth in latch or register count with pipeline depth in an asynchronous core or synchronous one induces additional hardware, affecting consequently the power consumption of core which shifts the optimal design point to shorter pipelines.

Thus, the first microarchitectural change was translating the standard 8051 CISC-ISA into RISC style micro-operations, allowing the use of a RISC-style execution core supported by a low-power 2-stages pipelined datapath (Fig. 2) and control structures to ensure the required power/performance tradeoff. Instructions are fetched from memory in the first pipeline stage, while the instruction's behavior will be performed in the second stage. A hardwired control unit which directly and

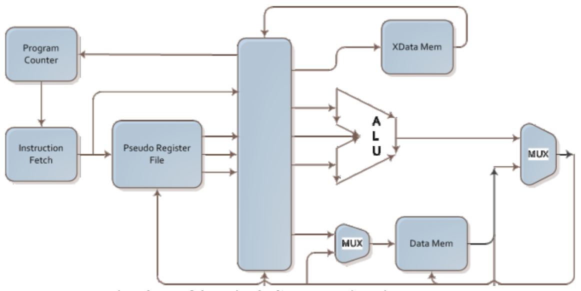


**Fig. 2 LP805X's 2-Stages Pipelined Datapath**

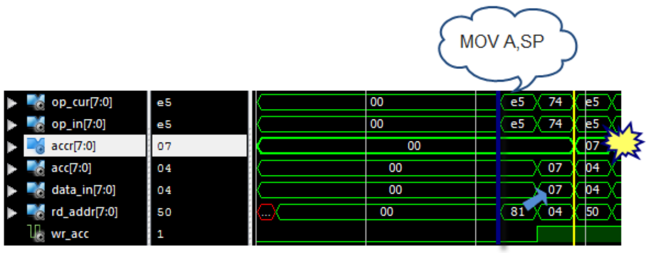


**Fig. 4 Partial simulation of MOV A, SP**

indirectly (i.e., some signal control are delegated to smaller and local control units, e.g., the ALU operand enabling) controls all control signals of the core. In doing so, it will be easier to pinpoint and fix possible errors as well as adding new functionalities without changing the main unit control.

The control unit was implemented as a finite state machine (FSM) named ST_STATE (Fig. 3) and is strongly coordinated with instruction decoding by using Verilog constructs, enabling cross-module connections optimization at RTL level. LP805X's CPU will transit among three control states ST_RESET, ST_WORK and ST_FIEI, which resets all internal registers to their default values, manages all instructions consuming more than one clock cycle, and fetches instructions and execute the micro-operations of the previous instruction, respectively. Each ST_WORK state is self decremental, e.g., being in ST_WORK_3 at clock cycle t, means that at t+1 the FSM will transit to ST_WORK_2. The ST_STATE FSM is split in two parts, consisting of combinational control signal outputs and sequential control signal outputs to enable better and easier management of operations involving memory or file register reading in just one clock cycle, without using asynchronous memory that are much more expensive and consuming lookup tables instead of FPGA dedicated memory blocks. Fig. 4, partially shows the activities during the simulation of "MOV A, SP": (1) the blue line marks the decoding of MOV A, (direct) instruction and (2) in the next clock cycle starts the next instruction and the content of the SP SFR is read, as shown by the blue arrow. To avoid any occurrence of RAW (Read After Write) data hazard in the same clock cycle, as only one cycle later register A will have available a copy of SP content, register forwarding was implemented. The above splitting of ST_STATE in combinational and sequential parts with the combinational part decoded by a local control unit leverages reduction of temperature (and consequently power reduction) and also dynamic power due to shorter travelled distance of data in the datapath.

The remaining signals which present less variability and shorter control word width are managed by the main control unit. This strategy was applied at RTL level to promote (1) laziness design and greater flexibility in HDL code management and (2) defeating synthesis tools cross-module optimization weakness. Thus, manual placement was applied since few are the optimizations that synthesis tools can forward to the placement and route back-end. Additionally, binary encoding was applied to the few states of the control unit, defeating power consumption.

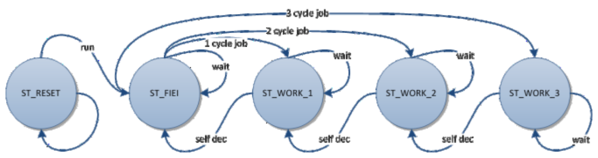


**Fig. 3 ST_STATE State Diagram**

## *B. The Decoder*

Given that some 8051 instructions are split and differentiated at register level, e.g., the ISA differentiates by opcode the following two instructions "MOV R7, #immediate" from "MOV R6, #immediate", a strategy was devised using the Verilog casez construct to group instructions based on similar operations, like the two presented above. Such register grouping of instructions is driven by expression (1) and it enables greater saving of power compared to a strategy based on a full opcode width, which should be dictated by a lookup-table overhead. Fig. 5 shows both opcodes 8'b1110_1000 and 8'b1110_1010 stacked into the same case statement, allowing a more compact and intuitive implementation. All instruction operands will be easily decoded except for register grouping instructions as they represent variability at operands. Based on expression 1, variable operands are decoded using those instruction bits masked with logical value z, i.e., the bits at don't care indexes (e.g., OPCODE[2:0]) were used to obtain the register reference number.

$$\begin{gathered} R\_i \leftarrow \{R\_0, R\_1\} \\ R\_N \leftarrow \{R\_0, R\_1, R\_2, R\_3, R\_4, R\_5, R\_6, R\_7\} \end{gathered} \qquad (1)$$

## *C. Other LP805X Components*

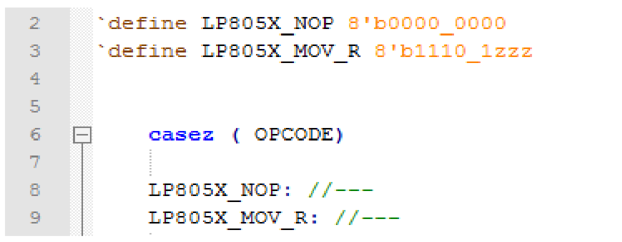
Additionally to the power optimization modules which will be described next, other LP805X components (e.g., Memory Interface, ROM Subsystem, Internal Data Memory Subsystem and ALU) and peripherals (e.g., pseudo random number generator, watchdog timer and AES cryptography) were also implemented and integrated, but they will not be elaborated on due to limited space.

The memory interface module supports several memory types which are statically decided according to application specifics when LP805X is synthetized, avoiding extra power consumption overhead compared to a dynamic configurability. It consists of interfaces to the internal data memory, to the external data memory and to the program memory (Fig. 1), with the first two interfaces based on a Wishbone bus optimized for low power consumption. A speculative handshake was implemented which predicts when read and write accesses are successfully concluded in the absence of a bus grant mechanism.

The program memory module was implemented by a 32-bit dual-port memory, avoiding multiple clock cycles reading for 8051-ISA 3-bytes length instructions. It can be customizable to use memory implementations offered by several FPGA providers as well as RTL codified memory. Additionally, it can also be configured to the application required size by the development ecosystem that integrates the RTOS.

The internal data memory module is split into separate components due to different datapaths related to SFR and to data memory, and also two special strategies is presented to guarantee hazard-free data read. The first one adds extra logic to the bus in order to bypass the read value instead of writing it back to memory, in case of a write back activity over the same address. The second one is based on a RTL codified memory with a built-in hazard-free mechanism, enabling synthesis tools to use dedicated FPGA memory resources that internally support hazard resolution. The choice between the two strategies is available through the development ecosystem. A local control unit was implemented to tackle any indirect addressing mode overhead in terms of power and performance by controlling additional register banks to accommodate copies of registers R0 and R1. Each SFR is individually created using 1-byte wide flip-flops and a module was implemented to manage several classes of SFR. Targeting the modularity, the SFRs can also be connected to the SFR bus using 3-state buffers and even using a combination of both.

Due to the scarce power gating capability at FPGA level, the ALU was implemented following a hybrid architecture combining tree with chain structures to execute 16 different instructions, and LUT combination was applied to synthetize 16-input multiplexers into one slice. To accommodate the other instructions, the carry flag was used to group ADD and ADDC, and INC and DEC, each in just one instruction. Also, a local control unit for selecting ALU operands was implemented fully separated from the ALU itself, avoiding the placement of operand decoding far from the data source

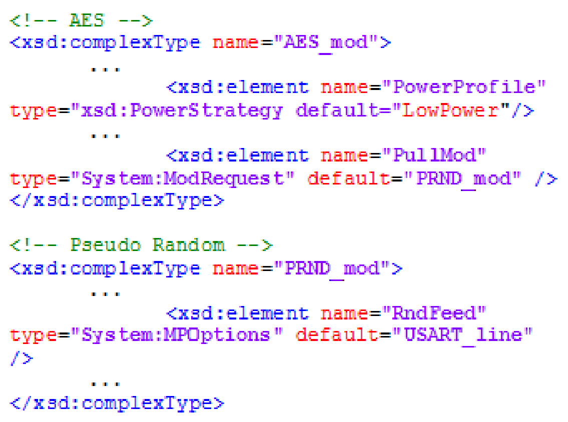


```
`define LP805X_NOP 8'b0000_0000
`define LP805X_MOV_R 8'b1110_1zzz


    casez ( OPCODE)

    LP805X_NOP: //---
    LP805X_MOV_R: //---
```

Fig. 5 Use of casez construct to implement register grouping

and consequently reducing dynamic power consumption.

The development ecosystem plays an important role connecting and optimizing key parts of the design, such as



```
<!-- AES -->
<xsd:complexType name="AES_mod">
      ...
            <xsd:element name="PowerProfile"
type="xsd:PowerStrategy default="LowPower"/>
      ...
            <xsd:element name="PullMod"
type="System:ModRequest" default="PRND_mod" />
</xsd:complexType>

<!-- Pseudo Random -->
<xsd:complexType name="PRND_mod">
      ...
            <xsd:element name="RndFeed"
type="System:MPOptions" default="USART_line"
/>
      ...
</xsd:complexType>
```

Fig. 6 Module configuration

automatic module insertion and parameterization. Fig. 6 illustrates how a peripheral module (i.e. AES) can request the insertion of another module into the design. This is achieved by the use of the System:ModRequest option that allows a module to request another. Similarly, but more specific to the module, the option System:MPOptions was utilized to specify the default source seed to the random number generator. By making use of Verilog parameters, a seamless integration with the development ecosystem backend was attained, allowing customization of the RTL according to the ecosystem specifications. Fig. 7 shows the integration of the parameterization alongside with the RTL source code in respect to the option System:MPOptions which dictates the default seed for the pseudo random number generator.

## IV. LP805X Architectural-Level Power Optimization

To guide the LP805X power optimization at architectural level, the results of the studies presented in [3, 9] were took into account and later extended to support system-level power optimization driven by the application needs under a RTOS (Real-Time Operating System) control. They started by identifying the most power consuming blocks, on which power optimization efforts should be concentrated using Keil Software Dhrystone test code, and then the power consumption of the identified blocks (i.e., Control Unit, UART, Timer, ALU) were estimated with Synopsys™ Power Compiler.

### *A. The Power Control Unit*

This module basically offers multiple clock frequencies and sleep mode services. The former was implemented to divide the clock frequency into multiple of two and is driven by a binary counter, while the latter was implemented to reduce to zero the clock frequency of a given module and its design is FPGA's supplier dependent.

As a generic comparison between clock resources provided by Xilinx and Altera architectures, the Altera clock-tree resources are able to combine a greater number of inputs, while with Xilinx the same effect will be achieved using

```
«FOREACH MPOptions AS MPOption»
parameter «MPOption.name» =
«MPOption.value» ;
«ENDFOREACH»

generate
        if ( seedSource == "USART_line")
                assign
                        seed = usart_line;
                        .....
endgenerate
```

**Fig. 7 Ecosystem managed parameterization**

several bufgmux blocks (notwithstanding considering other internal restrictions). However, to guarantee a safe clock-frequency transition, bufgmux resources were used to select the clock-frequency and drive it to dedicated clock bus. A new output was added to the clock frequency module by taking advantage of the two bufgmux entries to simulate two different clock sources. As shown in Fig. 8, the two entries will be the clock source selected by the SFR configuration and the clock source currently active.

Knowing both clock sources and the flag register that demand clock source change, SFR CLKSR, the order of the bufgmux resource input will be accordingly changed (Fig. 9). For Altera FPGA family, the clock-gating strategy presented in Fig. 10 was adopted to ensure a glitch-free signal and later changed using OR-gate instead of AND-gate as LP805X core implementation follows posedge clock (i.e., positive edge clock transition). Additionally, this model can automatically be converted to a flip-flop clock enable model by enabling EDA tools options in corner cases (e.g., the maximum clock frequency is degraded above an acceptable value). Note that the flip-flop enable strategy was used throughout all LP805X core but to switches off the clock tree. For Xilinx FPGA architecture, bufgmux resource was used to drive the clock line to zero.

### *B. The Power-Aware Scheduler*

From a holistic system-level design, a priority-based or power-aware scheduler can be chosen, but experiments with

```
if ( clk_write) begin
        clock_select <= #1 data_in;
        do_switch <= #1 ~do_switch;
end
...
always @(posedge clk)
        toggle_select <= #1 do_switch;

assign
        clk_0 = toggle_select ? clk_new :
clk_last,
        clk_1 = toggle_select ? clk_last :
clk_new:

BUFGMUX #(.CLK_SEL_TYPE("SYNC"))
        BUFGMUX_cpu ( .O(clk), .I0(clk_0),
                .I1(clk_1), .S(do_switch));
```

**Fig. 8 On-the-fly clock source transition**

```
assign
        clock_list = { bin, clk_main },
        clk_divided = clock_list[
sfr_select],
        clk_old = clock_list[ last_selected];

assign @(posedge clki)
begin
        if ( rst)
                last_selected <= #1 1'b0;
        else
                last_selected <= #1
sfr_select;
end
```

**Fig. 9 Selecting between the old and the new clock sources**

the power-aware scheduler reveal an unaffordable clock cycles consumption and so, the power-aware scheduler was partitioned between silicon scheduler (see Fig. 11) and software scheduler (see Fig. 12). A peripheral module was designed and implemented with two special SFRs dedicated to the clock-frequency (see Table I) control and it is based on a weighting value derived from task's deadlines and WCETs (Worst Case Execution Time), as dictated by a given application. Using several and different frequency domains throughout the LP805X core dictated special support to synchronize them. The strategy based on dual-position FIFOs and Dual-port RAM as described in [11] was adopted and changed to accommodate LP805X micro-architecture. In doing so, the memory interface module was adapted using lookup tables to indicate which addresses need such special control. The control unit was adapted as well enabling it to generate control signals which ensure safe data travelling through the asynchronous SFR interface. To provide access transparency to these SFR registers, the development ecosystem has built-in C++ TMP (Template Metaprogramming) artifacts and C macros which can be used by the application or system programmer.

## V. TESTS AND RESULTS

From all of the above strategies, manual placement is perhaps the one that leaps further away from the laziness and

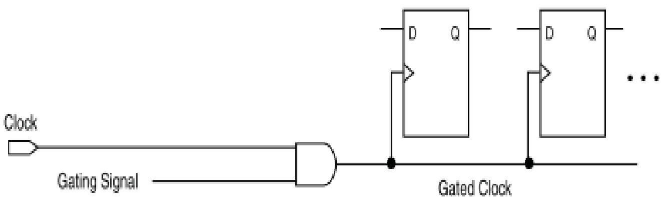
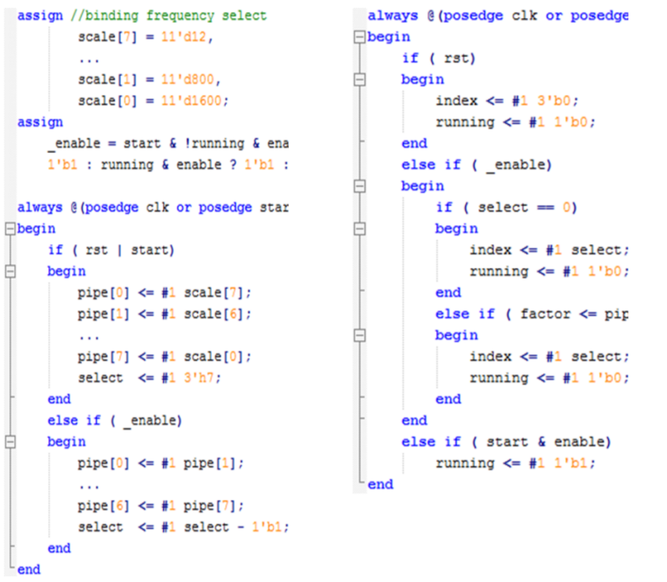


**Fig. 10 Altera recommended clock-gating strategy [10]**

TABLE I. SCHEDULER'S SFR REGISTERS TO BE CONFIGURED WITH THE WEIGHING VALUE

| | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **SCHEDH** | *start* | *enable* | *clock* | *clock* | *clock* | *factor* | *factor* | *factor* |
| **SCHEDL** | *factor* | *factor* | *factor* | *factor* | *factor* | *factor* | *factor* | *factor* |



```
assign //binding frequency select
        scale[7] = 11'd12,
        ...
        scale[1] = 11'd800,
        scale[0] = 11'd1600;
assign
    _enable = start & !running & ena
    1'b1 : running & enable ? 1'b1 :

always @(posedge clk or posedge star
begin
    if ( rst | start)
    begin
        pipe[0] <= #1 scale[7];
        pipe[1] <= #1 scale[6];
        ...
        pipe[7] <= #1 scale[0];
        select  <= #1 3'h7;
    end
    else if ( _enable)
    begin
        pipe[0] <= #1 pipe[1];
        ...
        pipe[6] <= #1 pipe[7];
        select  <= #1 select - 1'b1;
    end
end
```

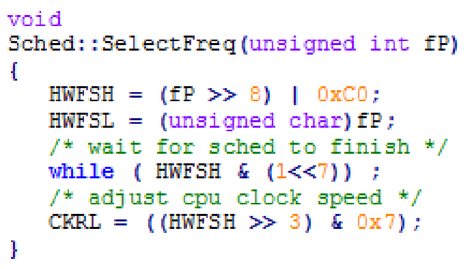
```
always @(posedge clk or posedge
begin
    if ( rst)
    begin
        index <= #1 3'b0;
        running <= #1 1'b0;
    end
    else if ( _enable)
    begin
        if ( select == 0)
        begin
            index <= #1 select;
            running <= #1 1'b0;
        end
        else if ( factor <= pip
        begin
            index <= #1 select;
            running <= #1 1'b0;
        end
    end
    else if ( start & enable)
        running <= #1 1'b1;
end
```

**Fig. 11 Software power-conscious scheduler**

generative design strategy. Thus, the end user designer is presented with the choice to: either manual place it himself or accept the one suggested by the mapper tool.

Up to and during this phase, the design was subject to several tests in relation to its (1) logic response, (2) raw performance, (3) power dissipation and (4) Et2. All ISA verification testbenches passed successfully and will not be herein referred given limited space. Focusing on a specific implementation over a Spartan 6, the resource utilization depicted on Fig. 13 shows that the power control unit has a negligible area overhead disregarding specific clock resource utilization (i.e. clock buffers) that is dependent of the needed flexibility of the final design.

While acknowledging that Et2 should be the main verification metric, this parameter is variable with the set of instructions that are executed by a given end user application.



```
void
Sched::SelectFreq(unsigned int fP)
{
    HWFSH = (fP >> 8) | 0xC0;
    HWFSL = (unsigned char)fP;
    /* wait for sched to finish */
    while ( HWFSH & (1<<7)) ;
    /* adjust cpu clock speed */
    CKRL = ((HWFSH >> 3) & 0x7);
}
```

**Fig. 12 Software power-conscious scheduler**

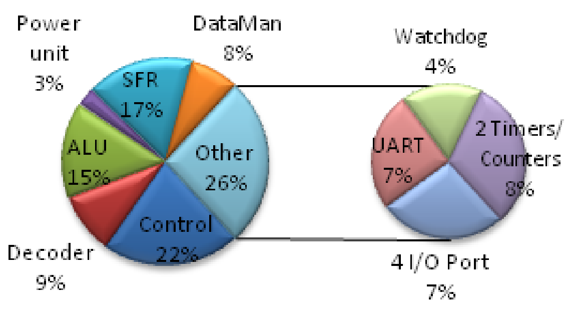


**Fig. 12 Resource Utilization**

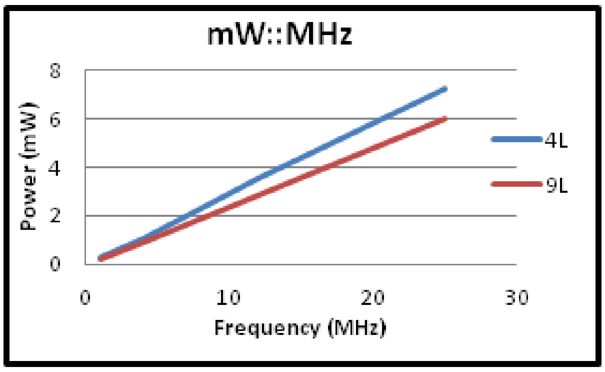


**Fig. 13 Power Consumption in mW per MHz**

In this sense, power consumption measurements were taken across a spread range of frequencies. As seen in Fig. 14, the pack factor has a stronger negative effect over power consumption as frequency rises. The post-route simulation predicts a power consumption of 230μW (in respect to the core as well as some peripherals) at approximately 1MHz while targeting a Spartan 6 device. The power aware scheduler calculates the minimum possible frequency while meeting the deadline at each scheduling point which considerably degrades the overall system performance. As the system performance is in line with the needed operating clock frequency, this led to non-optimal power consumption results as the lowest frequencies would not be engaged. A hybrid or hardware aided scheduler allowed for a very significant reduction of 2440 clock cycles in respect to a purely software approach.

TABLE II. COMPARISON BETWEEN SOFTWARE AND HYBRID SCHEDULER

| | **Worst Case Frequency** | **Best Case Frequency** |
|---|---|---|
| **Software Scheduler** | 12.2 MHz | 6.1 MHz |
| **Hybrid Scheduler** | 3.5 MHz | 762 KHz |

***not considering the startup condition at which the CPU starts at 12.2MHz**

The power readings of different schedulers varied according to the tasks and their respective timing information as well as real world interaction (i.e. I/O operation). Table II reports the best and worst case scenarios relative to the CPU operating frequency in respect to a test case. As power consumption increases almost linearly with operating frequency, the hybrid scheduler approach is expected to outperform the purely software based approach.

## VI. Conclusions and Future Work

In this paper a low power customizable 8051 microcontroller designed to be synthesizable manufacturer-independent while maintaining a set of features managed by an integrated graphical environment was presented. The component-based approach allows for a lazy design, as the IDE is able to optimize most settings according to a set of requirements. The proposed 2-stage pipeline combined with a RISC-like instruction set allows both significant reduction of the CPI and inherent flexibility while adding and maintaining new sets of features. A multiple frequency approach allows both CPU and peripherals to operate at different clock frequencies in a need by need basis. A hybrid power aware scheduler comprised of both software and hardware superseded a purely software based approach, helping the power unit module to cope with the system workload, thus further reducing power consumption. The overall results show that it is possible to achieve low power implementations regardless of the target FPGA device and that it can be finely tuned by a development ecosystem capable of optimizing the design at the RTL. Proposed as future work is the implementation of an asynchronous core to further reduce power consumption as well as completely enforcing the holistic development ecosystem by completely integrating all abstraction layers.

## VII. Acknowledgments

This work is supported by FEDER through COMPETE and national funds through FCT Foundation for Science and Technology in the framework of the project FCOMP-01-0124-FEDER-022674.

## References

[1] "A Practical Guide to Low-Power Design - User Experience with CPF". Internet: http://www10.edacafe.com/link/Power-Forward-Initiative-Practical-Guide-Low-Power-Design-User-Experience-with-CPF/25456/view.html, [Dec.. 18, 2012]

[2] "The EDA 360 Vision: The Way Forward for Electronic Design". Internet: http://www.cadence.com/eda360/pages/default.aspx, [Dec.. 18, 2012]

[3] Iozzi, F.; Saponara, S.; Morello, A.J.; Fanucci, L.; , "8051 CPU core optimization for low power at register transfer level," Research in Microelectronics and Electronics, 2005 PhD , vol.2, no., pp. 178- 181, 25-28 July 2005

[4] Chang-Jiu Chen; Wei-Min Cheng; Ruei-Fu Tsai; Hung-Yue Tsai; Tuan-Chieh Wang; , "A pipelined asynchronous 8051 soft-core implemented with Balsa," Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on , vol., no., pp.976-979, Nov. 30 2008-Dec. 3 2008

[5] Sakina Rangoonwala , "A VERILOG 8051 SOFT CORE FOR FPGA APPLICATIONS", Master Thesis, University of North Texas, August 2009.

[6] Meier, M.; Engel, M.; Steinkamp, M.; Spinczyk, O.; , "LavA: An Open Platform for Rapid Prototyping of MPSoCs," Field Programmable Logic and Applications (FPL), 2010 International Conference on , vol., no., pp.452-457, Aug. 31 2010-Sept. 2 2010

[7] Christian Piguet. Low-Power Electronics Design. CRC Press 2004

[8] "Eclipse openArchitectureware Modeling". Internet: http://www.eclipse.org/modeling/m2t/?project=xpand [Dec 18,2012]

[9] Sergio Saponara; Luca Fanucci; Pierangelo Terreni; , "Architectural-Level Power Optimization of Microcontroller Cores in Embedded Systems," Industrial Electronics, IEEE Transactions on , vol.54, no.1, pp.680-683, Feb. 2007

[10] "Quartus II Handbook Version 12.1". Internet: http://www.altera.com/literature/lit-qts.jsp, [Dec.. 18, 2012]

[11] Cliff Cummings. "Clock Domain Crossing (CDC) Design & Verification". Internet: .http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf [Dec.. 18, 2012]