Ultra Low-Power FSM for Control Oriented Applications

Muhammad Adeel Pasha and Steven Derrien
IRISA, University of Rennes 1
Rennes, France
Email: adeel.pasha@irisa.fr, sderrien@irisa.fr

Olivier Sentieys
IRISA, University of Rennes 1
Lannion, France
Email: sentieys@irisa.fr

Abstract — In this paper, we propose an approach combining the use of distributed hardware tasks implemented as Finite State Machines (FSM) and power gating techniques to obtain ultra low-power implementations. We target for control dominated applications represented as control task graphs, and propose a complete flow including a C to hardware task compiler. Our approach is validated experimentally and shows impressive improvement over software implementation on leading edge low-power microcontrollers such as the MSP430.

I. INTRODUCTION

In the last few years, the increasing need for low-cost embedded devices has paved the way for new generations of ultra-low-power microcontrollers such as the MSP430 [1] from Texas Instrument or CoolRISC [2] from EM Microelectronic. These programmable processors share common characteristics: they combine reasonable processing power with very low power consumption at a very low cost.

However, there are still application domains such as Wireless Sensor Networks (WSN) where the power dissipation of these devices is still orders of magnitude too high, since these systems expect sensor node modules to operate with extremely limited energy resources for very long time periods (months if not years). Worse, because these nodes remain idle most of their lifetime, their static power consumption plays a major role in their actual energy budget.

In situations, the only way to further improve the energy efficiency of the system is to specialize the design to the application at hand. This can be done either by designing a fully dedicated circuit, or by combining a programmable core with application specific co-processors. Indeed, it has been shown that for many compute-intensive applications, the use of specialized co-processors could dramatically improve the power efficiency of a design (up to more than one order of magnitude [3]).

Unfortunately, the vast majority of the applications targeted by such small microcontrollers do not fall into this category. These applications have complex control-flows which are often driven by external events and are therefore much less predictable. In practice, they are implemented using threads-like constructs provided by the operating system, and hence rely on the system task scheduling policy. To our knowledge, there are still very few experimental evidences showing that, for this type of application, power savings could also be obtained by means of hardware specialization: this is the topic addressed in this work.

In this paper, we propose an approach to specialization which consists in implementing each task of the application graph on a specialized architecture. This architecture is in the form of a minimalistic datapath controlled by a custom finite state machine (FSM) and is being automatically generated from a task specification in C, by using an ASIP-like design environment retargeted to our purpose.

This approach first allows dynamic power savings by reducing hardware complexity (we avoid the need for instruction memory and decoder), and also permits the use of aggressive FSM power optimization techniques (FSM partitioning, glitch free/low activity state encoding, etc.). Since the static power contribution will necessarily grow due to the possibly numerous number of hardware tasks implemented on chip, we propose to combine our approach with power gating techniques. The idea consists in supplying power to the hardware controller only when its associated task is to be executed.

More specifically, the contribution of this article lies in an in-depth quantitative analysis of the benefits of this approach. This analysis is built upon a wide set of experimental data gathered from power estimation for control-oriented benchmark FSMs and SPICE transistor level timing simulations. The results show the relation between the number of elementary gates in the gated-component, the size of the power gating transistor and the on/off switching delays for component’s output.

Our experimentations show that this approach is very promising:

- First, it offers dynamic power saving by factors between 210 and 150 for small to medium sized control-flow FSMs (w.r.t a low-power MCU such as the MSP430).
- Second, the dynamic power gating scheme happens to have very short switching time delays, in the orders of a few hundred of nano seconds for the larger designs.

Of course, such an approach only makes sense if the designer is able to automate the synthesis of these hardware tasks from a high level description of the application in C. As a consequence, and for the sake of completeness, we also outline the key features of our C to FSM compilation flow prototype.

This paper is organized as follows. We start by presenting the related works in Section II and describe thoroughly our proposed approach in Section III. In Section IV we present experimental results which confirm the validity of our approach and propose a brief discussion on open issues. Finally, conclusion and future research directions are drawn in Section V.

II. RELATED WORK

In the last decade, there have been a large number of research results dealing with power optimization in VLSI circuits (see Devadas et al. [4] for a survey on the topic). A lot of research has also been done to optimize power at microarchitectural level like low-power MCUs and at operating system (OS) level. On the other hand, research results related to low power FSM implementations are also of interest for our proposal. This section briefly covers some of these power optimization techniques.

A. Clock and power gating

Clock Gating and Power Gating are well-known VLSI circuit level techniques to reduce power. Clock Gating consists in gating the clock signals that drive inactive portions of the circuit. The purpose is to minimize the switching activity on flip-flops and clock distribution lines, since they represent a large part of the circuit dynamic power consumption. This technique is very efficient for reducing dynamic power, however it has no impact on static power (which, in our context, is a significant part of the power budget).

To the contrary, Power gating consists in turning off the power supply of inactive circuit components. Power gating helps in reducing...
both dynamic and static power, and is thus very efficient for devices in which components remain idle for long time periods.

![Figure 1. Power gating used by adding sleep transistors to the logical blocks being power-gated.](image)

The technique consists in adding a sleep transistor between the actual $V_{dd}$ rail and the component’s $V_{dd}$, thus creating a virtual supply voltage called $V_{vdd}$ as illustrated in figure 1. Sleep transistor allows the supply voltage of the block to be cut off to dramatically reduce leakage currents. This technique however has a few drawbacks:

- Power gating is tedious to use, since it must be implemented at a very low-level (i.e. transistor level) of the circuit and requires manual intervention of the designer.
- Power switching is not instantaneous, and this switching delay increases as the number of transistor in the power-gated block grows. As a consequence, the power switching decision must be taken with care.

The approach has already been used in the context of high-performance CPUs [5], and FSM implementations [6] where part of the design are switched on/off according to their activity. The approach helps in reducing the static power dissipation for FPUs of a high-end CPU by up to 28% at the price of a performance loss of 2%, for FSMs the average reported power reduction was also 28%.

### B. Low-power MCUs and operating systems

As far as the ultra low-power microcontrollers (such as the MSP430 and the CoolRisc) are concerned, they share many characteristics: a simple datapath (8/16-bit wide), a reduced number of instructions (only 27 instructions for the MSP430), and several power saving modes which allow the system to select at runtime the best compromise between power saving and reactivity (i.e. wake up time).

Most CPU packages comprise a limited amount of RAM memory (only a few hundred to a few kilo bytes) and non volatile flash memory. This limited amount of storage resources poses great challenges to the software designers since both the user application and operating system must work with this very small amount of memory.

As a consequence, there have been several attempts to reduce the complexity of OS on these devices. In particular, many approaches have been proposed to reduce the overhead caused by dynamic scheduling of threads by using alternative concurrency computational models. For example, the TinyOS is built upon an event-driven approach, without explicit thread management [7], and Contiki [8] proposes a simplified thread execution model (named protothread), in which preemption can only only occur at specific points in the Task Control Flow.

### C. Low-power FSM implementations

As we propose a specialized hardware architecture, having a minimalistic datapath controlled by an FSM, for each task of an application, low-power FSM implementation techniques were also explored. There have been many proposals dealing with this issue, ranging from power friendly state encoding schemes [9], [6], to clock and/or power gating through FSM decomposition [10], [11].

However, we will not address the problem of low-power FSM realization in this work and rely on existing off-the-shelf logic synthesizers.

### III. Proposed Approach

In this work, we target control-oriented/event-based applications that can be represented as a Control Task Graph. In this type of graph, a task execution is generally triggered either by another task or by a combination of external events. We restrict ourselves to such multitasking system in which preemption can only occur at certain specific cache steps of the program, as in the case of the prototool implementation available in the Contiki’s ultra-light-weight OS.

This section describes the principles behind the mapping of an application task graph representation to our hardware execution model, and presents the internal structure of our hardware task model.

#### A. Application mapping

As mentioned previously, each task of the application is mapped onto a specialized hardware structure called a hardware task. This hardware task is controlled by a power gating mechanism and can therefore be powered on/off when needed. Each of these hardware tasks can access a global shared memory, and can be directly connected to some of the peripheral I/O ports.

These hardware tasks are monitored by a global controller which is responsible for starting a hardware task when its activation condition becomes valid. Each hardware task can be activated by setting its enable flag by the monitor and after finishing, it sets its done flag to signal that it is free for the next event. A global view of such a system is given in Figure 2.a which shows an example of a mapping with three tasks.

In our approach, the monitor also ensures that no two tasks accessing a same shared resource can be active at a given time instant. This mutual exclusiveness allows a drastic simplification of the shared resources access control logic. Indeed, there is no need for an arbiter, and no need for tristate/multiplexer logic since we are sure that only one of the hardware tasks is able to drive the shared lines at a given instant (the other tasks being powered off).

Of course, taking advantage of this simplification requires the system monitor to take the switching on/off latency for each FSM into account, in order to guarantee that their lifetimes do not overlap.

#### B. Hardware task model and generation

Our hardware tasks follow a relatively simple architectural template, illustrated in figure 2.b, and consists of the following components:

- A simplified (and customizable) 8-bit RISC-like datapath that contains a limited set of execution unit (arithmetic and logical operations). This datapath includes small configurable $4 \times 8$ register file, that is used to store intermediate results.
- A local RAM memory block which contains the program data set, and a ROM used to store all immediate values used needed by the program. Both RAM and ROM can serve as instruction operands which avoids the need for a large register file.
- Accesses to shared resources such as external I/O ports and shared memory.
- A hardware Finite State Machine (FSM) which implements the microcode driving the control signals to the datapath so as to execute the operations corresponding to the task at hand.

The generation of a hardware task is done automatically from a software specification of the task behavior in C. Detailing our compilation flow is out of the scope of this paper, however for the sake of completeness we outline its main characteristics.

Our compilation flow (figure 2.c) is based on the GECOS compiler infrastructure [12], a retargetable C compiler framework, from which
we use the BURG instruction selector that is retargeted to our simplified datapath model. This low-level program representation is then used to generate (i) a VHDL representation of an FSM that will control the execution units of our datapath, acting as a microcoded program, and (ii) a custom datapath which implements the minimum required set of operations for the task at hand.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

This section describes the experimental setup used for the generation of control-oriented benchmark and extraction of output switching timings for a power-gated block. We then show the power reduction gained by the hardware task over the MSP430 and discuss some open issues regarding our proposal.

A. Benchmark generation

A key issue for validating our approach is the choice of the benchmarks used for experiments. Clearly, mapping a real WSN application (including its OS layer) to our execution model would require a full rewriting of the code, which is currently out of our reach. Besides, our goal in this work was not to quantify precisely the power benefits of this approach, but rather to provide a early stage evaluation of the expectable benefits. As a consequence, we made the choice of basing our experiments on two other benchmarks:

- We first used a logic synthesis VHDL FSM benchmark (LGSynth’93 [13]) for which we generated equivalent VHDL and C (targeting the MSP430) representations. In this representation, the I/O pins of the FSM were mapped to the I/O ports of the MCU, these input ports being then used to determine the output ports state and the sequel of the execution flow.
- Because this first benchmark was somewhat biased in favor of an hardware implementation, we also generated random MSP430 assembly level control-flow graphs, with characteristics chosen so to be as close as possible to a typical control dominated program (short basic blocks containing 2 to 16 states, at most two output transitions per state to match branch-like machine instructions). These control-flows were then used to generate both a C (MSP430-based) and VHDL description of the equivalent hardware task FSM.

We are aware that these benchmarks may seem artificial, since they do not correspond to real software code, but we believe that they are sufficient for our purpose, since they share many characteristics with our target applications: a moderate code size (a few hundreds equivalent lines of codes), and an execution flow largely dominated by control structures.

<table>
<thead>
<tr>
<th>Name</th>
<th>ROM Size (Bytes)</th>
<th>Power (µW)</th>
<th>Hardware Power (µW)</th>
<th>Task Gain (x)</th>
<th>IGLOO Power (µW)</th>
<th>Gain (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s832b</td>
<td>1770</td>
<td>480</td>
<td>227</td>
<td>194</td>
<td>3710</td>
<td>11</td>
</tr>
<tr>
<td>tbc</td>
<td>7266</td>
<td>600</td>
<td>234</td>
<td>188</td>
<td>3655</td>
<td>12</td>
</tr>
<tr>
<td>s820b</td>
<td>1628</td>
<td>480</td>
<td>221</td>
<td>199</td>
<td>3575</td>
<td>11</td>
</tr>
<tr>
<td>s1494b</td>
<td>2492</td>
<td>480</td>
<td>247</td>
<td>178</td>
<td>3590</td>
<td>7</td>
</tr>
<tr>
<td>r30</td>
<td>608</td>
<td>480</td>
<td>211</td>
<td>208</td>
<td>3244</td>
<td>13</td>
</tr>
<tr>
<td>r42</td>
<td>932</td>
<td>480</td>
<td>221</td>
<td>199</td>
<td>3323</td>
<td>13</td>
</tr>
<tr>
<td>r66</td>
<td>1428</td>
<td>480</td>
<td>242</td>
<td>181</td>
<td>4024</td>
<td>11</td>
</tr>
<tr>
<td>r96</td>
<td>1998</td>
<td>480</td>
<td>254</td>
<td>173</td>
<td>4076</td>
<td>10</td>
</tr>
<tr>
<td>r147b</td>
<td>2818</td>
<td>480</td>
<td>286</td>
<td>153</td>
<td>6897</td>
<td>6</td>
</tr>
</tbody>
</table>

TABLE I

SUMMARY OF DYNAMIC POWER CONSUMPTION FOR VARIOUS TARGETS.

* LGSynth’93 Benchmark FSM
  * Randomly generated FSM

B. Output switching timings for a power-gated block

We used the similar model of a power-gated block as was used in [5]. However, as the author did not provide any quantitative data for the switching delays specific to a CMOS technology, we had to re-run the experiments. For this purpose, we used Mentor Graphics Eldo to perform the transistor level Spice simulations using a 130 nm CMOS technology. We used parallel Nand gates to model the timing behavior of a gated block (Figure 3.a). We observed a linear relation between the number of gates to be power-gated, n and the sleep transistor width, w. Similarly, a linear relation between n and output switching delays of the circuit was observed. Figures 3.b and 3.c show that for a block consisting of 3000 gates, we have a turn-on delay of 38 ns and turn-off delay of 451 ns (between cut-off and active mode). The sleep transistor width was set to W = 2.04 µm, which is equivalent to 17 x 0.13 µm. This must be compared with MSP430’s typical wake-up delay of 1 µs from the standby mode.

C. Power gains of hardware task approach

The hardware task VHDL designs have been synthesized for two targets : STMicro 130 nm CMOS library and Actel IGLOO [14] FPGA. We used these synthesis results to extract gate-level static and dynamic power estimations (we assumed 100 MHz clock frequency). The corresponding MSP430 source codes were compiled with gcc to determine the binary code size and the subsequent power dissipation for the ROM block containing this binary code. We used as baseline the MSP430F21x1 dissipation of 44 mW normalized at 100 MHz (the data sheet indicates a dissipation of 440 µW at 1 MHz).

The results are given in Table I in which the two first columns
MSP430 micro-controller, and under very conservative assumptions, handled by an FSM. implemented as specialized hardware blocks in which microcode is to power-gate our hardware tasks, since the output turn-on and turn-off the early results of SPICE level simulations confirm the feasibility power savings by two orders of magnitude may be possible. Similarly, IGLOO FPGA, using Actel’s Libero IDE software. Here again, our specialized implementations are more efficient (at least by a factor of 6) than the MSP430F21x1.

D. Discussion

This approach also assumes that FSMs are hard-wired into the silicon as ASIC blocks. This means that the behavior of each hardware task is fixed, making post-production upgrade or bug fixing very costly, given current NRE production costs. This is an important issue, since flexibility is often a great concern for embedded system designers.

However, as far as WSN are concerned, the programmability issue is more geared towards the application layer, which represents only a small fraction of the processing requirements. Hence, providing programmability in the form of small sized power-gated CPU would solve the problem while preserving most of the power saving.

Then, there also exist alternative target technologies, such as Structured ASICs, which offer very low NRE costs, at the price of a 2 to 5 times decrease in power efficiency. Given that our power savings are more than two orders of magnitude, this approach would still remain competitive.

V. Conclusion

In this paper, we have proposed an original approach for the ultra low power implementation of small scale control dominated applications. Our approach is based on hardware tasks which are implemented as specialized hardware blocks in which microcode is handled by an FSM.

Our preliminary experiments show that, compared with MSP430 micro-controller, and under very conservative assumptions, power savings by two orders of magnitude may be possible. Similarly, the early results of SPICE level simulations confirm the feasibility to power-gate our hardware tasks, since the output turn-on and turn-off delays (between cut-off and active mode) are 38 ns and 451 ns respectively when compared with a wake-up delay of 1 µs for the MSP430 (from standby mode).

We envision two directions for our future work. We first aim at studying the scalability and validity of the approach on more realistic benchmarks extracted from existing WSN OS implementations. We would also like to evaluate the feasibility (and power efficiency) of our approach on control oriented reconfigurable structures, which would provide support for small grain power gating techniques [15].

REFERENCES