LCTES'18 Home

francisco.santanna@gmail.com Profile · Help · Sign out
Response deadline in 9 minutes

#16 WIP: Transparent Standby for Low-Power, Resource-Constrained Embedded Systems: A Programming Language-Based Approach

Main Edit <#11 Your submissions (All) Search

Email notification
Select to receive email on updates to reviews and comments.

PC conflicts
None

Submitted

Submission 23 Feb 2018 8:25:39am EST · 77f979c7

▶ Abstract

Standby efficiency for connected devices is one of the priorities of the G20's Energy Efficiency Action Plan. We propose transparent programming language mechanisms to enforce that applications remain in the deepest standby modes for

[more]

▼ Authors (blind)

Francisco Sant'Anna (Rio de Janeiro State University) <francisco.santanna@gmail.com>
Alexandre Sztajnberg (Rio de Janeiro State University) <alexszt@ime.uerj.br>
Noemi Rodrigues (PUC-Rio) <noemi@inf.puc-rio.br>

▶ Topics

	OveMer	RevExp
Review #16A	4	3
Review #16B	3	3
Review #16C	3	3
Review #16D	3	4
Review #16E	4	3

1 Comment: *Response* (F. Sant'Anna)

You are an **author** of this submission.

Edit submission **Edit response**

Reviews and comments in plain text

Review #16A

Overall merit 4. Accept

Reviewer expertise 3. Knowledgeable

Paper summary

The paper makes contribution by introducing the par/or keyword into CEU. The par/or keyword triggers creation of two lines of execution. In so doing, the user code structure is preserved in its form. Further, the await keyword notifies the runtime the specific peripheral that is awaiting an event. Knowing specifically which peripheral awaiting an event, the runtime can put the device to different levels of sleep but still allow the device to be waken up should event arrives for that peripheral. It applies the async/isr keyword from CEU to define ISR in the driver. (Interrupt service routine may executes asynchronously) The finalize keyword is also introduced into the driver to allow proper cleanup of the line of execution. The driver further calls down to the power management runtime which determines the best power saving strategy.

Comments for author

Applying the par/or keyword to the application code in order to preserve program structure is nice. It is also clever to use the await keyword to convey the peripheral information to the runtime.

Questions for authors

I would appreciate getting a size estimate of the user code, runtime, driver, power management module as the cortex-m0 is very small in ROM/RAM. I would also appreciate knowing the amount of power saved with this approach on some real life application. Perhaps what is illustrated in Fig 1 is a real life example? If so, may we get a bulk estimate of the amount of power saved?

Review #16B

Overall merit

3. Weak accept

Reviewer expertise

3. Knowledgeable

Paper summary

This paper extends the programming language CEU to enforce an IoT device remains in the deepest standby mode for the longest period of time. It relies on the synchronous semantics of the CEU language which guarantees that reactions to the environment always reach an idle state amenable to stand by.

Comments for author

This is an interesting framework for IoT devices where power is the first order consideration. It is not clear to me how the timing requirement can be met under this framework. Also, a quantitative analysis on the performance and energy for real applications is needed to show this work's value in practice.

Review #16C

Overall merit
3. Weak accept

Reviewer expertise
3. Knowledgeable

Paper summary
Work in progress paper.

The authors describe their work in progress on extending the Ceu programming language with support for managing low power and standby modes at the application level. They implement it for the ATmega328p (Arduino).

Comments for author
strengths

- interesting use of a synchronous language
- power savings is still hard to do and important
- sufficiently interesting for a work in progress paper (proper length)
- implemented for the ATmega328p

weaknesses

- very simple language extension, unclear what the novel contribution is here OK b/c of this being a work-in-progress paper
  - no empirical evaluation or comparison to existing work
  - no discussion of related work
  - no evaluation of the overhead imposed by the use of the language
- doesn't this place an extra burden on programmers and a level of expertise beyond that novices to control at the program level?
- Why was Ceu chosen here? Can we not do something similar and simpler via libraries and wrappers using existing low level languages? its unclear to me that synchronous semantics have a clear adoption advantage given the cost/overhead of its implementation

other things

- the intro over does it a bit in my opinion with the emphasis on the G20 action plan. this isn't appropriate for a technical conference.

Questions for authors
Please address the weaknesses above

Review #16D

Overall merit
3. Weak accept

Reviewer expertise
4. Expert

Paper summary

The paper proposes a transparent standby mechanism for low-power resource-constrained embedded systems.

Comments for author

Making effective use of standby mode is important. The authors propose to provide standby mechanisms at the programming language level. While it is possible to integrate this approach with CEU, which is based on synchronous concurrency model, it is not clear whether the approach can be extended to regular programming languages. Moreover, the benefit of this approach in real systems has not been evaluated.

Review #16E

Updated 18 Mar 2018 10:04:01am EDT

Overall merit

4. Accept

Reviewer expertise

3. Knowledgeable

Paper summary

The paper proposes a programming abstract and runtime framework that allows the system to take advantage of the longest periods of time and deepest sleep modes possible for power-constrained IoT devices without complicated programming efforts. The crux of the technique is to expose standby semantics at the language semantics level (i.e., await and exploiting asynchronous event handling semantics) for app developers, to expose I/O operations that may affect the standby mode at the language semantics level (e.g., emit, output, input) for driver developers, and device the runtime system (interrupt system) to take advantage of such semantics for longest standby mode. This is a very neat abstraction

Comments for author

Pros:

+ Identified an important problem for an emerging application domain (IoT & standby power)

The paper made a strong motivation on why standby power saving is important for IoT devices, which is an area where not much programming and systems research have been done. It is important to bring important new problems to the community.

+ Proposed solution is novel and the use of language technologies to abstract the problem is appropriate

I like the clean design of the CEU language semantics for both App and Driver. It seems a nice way to formulate the standby problem.

Cons:

– Expressiveness of the DSL in representing real IoT app/driver semantics not demonstrated

The paper lacks real wokload. A major hurdle of a practical DSL is its expressiveness, i.e.,

whether real-world considerations can be expressed in a well-defined abstraction. I understand this is an WIP, but would strongly encourage the authors to look for some real IoT scenarios to validate this design.

Response

Edit

[Francisco Sant'Anna <francisco.santanna@gmail.com>] 19 Mar 2018 10:42:09am EDT **868 words**

We would like to thank the reviewers for the thoughtful comments, questions, and criticism. We will accommodate all clarifications and answers in a final version of the paper. Each comment is addressed individually as follows.

References:

[1] : *Francisco Sant'Anna and others. 2013. Safe System-level Concurrency on Resource-Constrained Nodes. In Proceedings of ACM SenSys'13.*

[2] : *Adriano Branco and others. 2015. Terra: Flexibility and Safety in Wireless Sensor Networks. In ACM Transactions on Sensor Networks.*

Review #16A

> I would appreciate getting a size estimate of the user code, runtime, driver, power management module as the cortex-m0 is very small in ROM/RAM.

All code is compiled together into a single binary image, but the runtime is in the order of a few kilobytes (<5Kb). Previous work [1] shows an overhead of around 10% for ROM and 5% for RAM for complete applications in comparison to equivalent implementations in C (network protocols, radio driver, etc.).

> I would also appreciate knowing the amount of power saved with this approach on some real life application. If so, may we get a bulk estimate of the amount of power saved?

Even though the examples in the paper use standby effectively, they are too simple to draw any conclusions about power savings in real-world applications. That said, our approach is optimal in the sense that the language ensures idle states and the runtime can always figure out the most efficient sleeping mode possible. Nonetheless, this is a work in progress and we acknowledge the importance of a quantitative evaluation in the conclusion of the paper as future work. The evaluation will have to be quite extensive to cover typical passive/active patterns in IoT applications.

Review #16B

> It is not clear to me how the timing requirement can be met under this framework.

Timing requirements can be addressed in two levels of guarantees. For soft real-time requirements, synchronous reactions (i.e., code outside ISRs) are safer and more

straightforward to use because they support rich control abstractions (e.g., `par/or`), prevent race conditions, and are guaranteed to terminate in bounded time. However, since reactions run atomically and cannot interrupt each other, a non-negligible delay in starting times is possible. For more strict requirements, ISRs (i.e., code inside `async/isr`) are asynchronous and can perform sensitive operations much faster (such as filling a serial buffer) while synchronous code is active.

> Also, a quantitative analysis on the performance and energy for real applications is needed to show this work's value in practice.

A quantitative analysis for real applications will be done in future work. Please, see last comment in `Review #16A`.

Review #16C

> no empirical evaluation or comparison to existing work
>
> no discussion of related work

Empirical evaluation and comparison to existing work will be done in future work. Please, see last comment in `Review #16A`.

> no evaluation of the overhead imposed by the use of the language
>
> its unclear to me that synchronous semantics have a clear adoption advantage given the cost/overhead of its implementation

The overhead is very little. Please, see first comment in `Review #16A`.

> doesn't this place an extra burden on programmers and a level of expertise beyond that novices to control at the program level?
>
> Can we not do something similar and simpler via libraries and wrappers using existing low level languages?

The burden to deal with standby is mostly on the device drivers, which *"are write-once code that is typically packaged and distributed in a software development kit (SDK)."* `[Introduction]` By transferring the work from the applications to the language level, the idea is that novice programmers never have to deal with standby explicitly. In general-purpose languages, by exposing libraries and wrappers, the programmers would still have to call them explicitly, which is exactly what our proposal intends to avoid.

Furthermore, we show in Figure 2 that, when introducing concurrency, the structure of the program remains sequential and amenable to inference of the appropriate sleep mode. In comparison with Arduino, whose main goal is to lower the entry barrier for embedded development, Céu also preserves the sequential structure for concurrent applications.

> the intro over does it a bit in my opinion with the emphasis on the G20 action plan.

OK. We will also discuss the motivations from an academic perspective.

Review #16D

> While it is possible to integrate this approach with CEU, <...>, it is not clear whether the approach can be extended to regular programming languages.

Céu preserves a sequential programming structure for complex concurrent applications. Please, see third comment in `Review #16C`.

> The benefit of this approach in real systems has not been evaluated.

The evaluations of the benefits in real systems will be done in future work. Please, see last comment in `Review #16A`.

Review #16E

> Cons:
>
> > Expressiveness of the DSL in representing real IoT app/driver semantics not demonstrated
>
> A major hurdle of a practical DSL is its expressiveness, <...>. I <...> would strongly encourage the authors to look for some real IoT scenarios to validate this design.

Although we have not evaluated the energy efficiency aspects (see last comment in `Review #16A`), previous work `[1,2]` evaluates the expressiveness of Céu in the context of Wireless Sensor Networks and discusses the development of drivers, network protocols, and full applications in Céu.