

Using Sequencing to Trigger a Better Analysis

Nathan Coopriider and John Regehr
University of Utah School of Computing
{coop, regehr}@cs.utah.edu

Abstract

We improve the static analysis of control flow in programs running on embedded microcontrollers. Our new techniques, called sequencing and triggering analysis, can sometimes determine that the scheduler always runs a collection of tasks in a certain order or that some interrupt handlers are dead or at least cannot fire at certain times. Triggering identifies root causes of indirect control flow through system idioms and programmer annotations. Sequencing propagates the results of triggers through system execution in order to refine our estimate of the program control flow graph. We show that triggering and sequencing improve the effectiveness of static analyses of sensor network applications. Improved static analyses result in better optimizations, increased verification power, and more accurate worst-case-execution time estimations.

1. Introduction

Wireless sensor networks (WSNs) have been deployed in locations varying from glaciers to mine fields [5]. Nodes typically consist of one or more sensors, a radio, a microcontroller (MCU), and a battery. They face severe constraints in terms of power, size, features, and price. A WSN developer must cope with these constraints in order to produce effective applications for deeply embedded systems.

Advanced development tools can make it easier to create WSN applications. For example, optimizers may reduce code size, data size, execution time, power usage, or any combination of those. Optimizers may even trade one resource for another. Other tools support checking for stack overflow, timing overruns, and language-level bugs. All of these tools are based on *static analysis*, which must be precise in order to deliver good results.

Hidden dependencies in embedded software make it difficult for an analyzer to precisely (but conservatively) estimate control flow. Determining the control-flow successors of a given statement often requires only local knowledge to obtain adequate precision. However, local information is in-

sufficient to accurately model the effects of indirect control flow through the scheduler or interrupt controller. These cases require implicit non-local information because they depend on global structures, underlying hardware, or environmental events. Without using this implicit information, an analyzer must be conservative, for example by estimating that any interrupt may fire at any time, and that any task may be dispatched any time the scheduler is called.

We propose that capturing these implicit dependencies in a static analysis will significantly add precision, leading to better optimization and verification. These dependencies are common when analyzing interrupt driven systems code for WSNs. Specifically, we will:

- Identify **trigger** relationships in MCU code. A trigger indicates an implicit control-flow dependency.
- Discover the chains of triggers existing in a program, called **sequencing**.
- Combine sequencing graph with the control-flow graph (CFG) to improve analysis precision.

2. Background and Related Work

This work builds upon our previous efforts with cXprop and TinyOS. cXprop [1] is our static analysis and transformation tool for C code and includes options to specifically target WSN applications. It primarily functions as a whole program optimizer. TinyOS 2 [3] is an operating system for WSN nodes. It is written in nesC, a C dialect that is translated into C. TinyOS uses a specific interrupt-driven concurrency model. Most code runs in scheduled and dispatched functions called tasks. Each task completes before the next task is dispatched, so tasks do not preempt one another. Interrupts may preempt tasks and interrupts.

We mention two projects that have used ideas similar but not equivalent to our work. Ramaprasad and Mueller [4] examine preemption points to increase the precision of WCET analysis. The Ptolemy project [2] studies the analyses of concurrent embedded systems. Their tagged signal model, concept of causality, and description of ordering all relate to triggers and sequencing. Our work specifically targets improving static analyses for interrupt-driven MCUs.

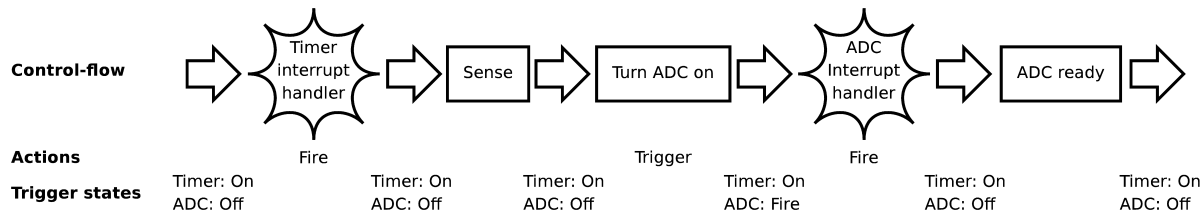


Figure 1. Example showing behavior of timer and ADC data ready triggers

3. Proposed solution

We introduce trigger recognition and use triggering information to create a sequencing graph. The created sequencing graph supplements the CFG, increasing the overall precision of the analysis by removing false control-flow edges. We measure improvement by comparing optimizations on TinyOS 2 code using the sequencing graph with those using traditional CFG generation.

Trigger recognition A trigger represents an implicit control-flow dependency. The triggers of interest to our analysis are pieces of code which exclusively cause a task or interrupt to run. Posting a task means the task is scheduled to run and the task will not run unless it is posted. Turning on the timer eventually leads to timer interrupts and timer interrupts to not occur until the timer is activated. Activating the analogue to digital converter (ADC) eventually results in a data ready interrupts, but the interrupt will not be sent by an inactive ADC. Many triggers such as these exist, but may not be obvious at the source level.

Our analysis makes triggers explicit using system specific information or programmer annotations. The scheduling of a task follows a certain language idiom in TinyOS: a call to the scheduler with the task identifier as a parameter. Our analysis identifies task triggers by searching for this idiom in the code. Additional modeling of the hardware can identify some interrupt triggers. Certain bits of memory-mapped hardware registers turn on certain features, identifying a trigger. Annotations identify additional triggers.

Scheduler information and interrupt priorities may be used to remove infeasible triggers. Since a particular task may appear only once in the scheduling queue for TinyOS, when that task dispatches it does not remain triggered. Similarly, if a low priority task or interrupt executes, none of the higher-priority tasks or interrupt are triggered.

Figure 1 illustrates two interrupts and how our analysis will handle their triggers. The hardware timer has been turned on prior to the start of the example, so the timer interrupt trigger is *on*. This means that it can fire more than once per trigger, so even after the interrupt the trigger remains on. The ADC interrupt trigger indicates the interrupt will *fire*, meaning after one execution it returns to an untriggered state, or *off*.

Sequencing graph One behavior of triggers shown in Figure 1 is that they form a sequence of execution. The timer interrupt is triggered, which then triggers the ADC interrupt. All of the possible trigger sequences for a program form a sequencing graph. Information from the sequencing graph augments the computed CFG. For example, after an atomic section an analysis must assume that all available interrupts can fire. The sequencing graph eliminates interrupt firings which have not been triggered and are therefore unavailable. The sequencing graph introduces non-local information about implicit control-flow dependencies. This information will then reduce the set of potential control-flow successors, which increases the precision of the analysis.

Evaluation We evaluate the effect of sequencing by comparing the performance of cXprop with and without sequencing information. We use benchmarks from the TinyOS 2 [3] release applications. cXprop should eliminate more dead code and propagate more constants when sequencing information is used. We also compare cXprop with sequencing information to gcc.

References

- [1] N. Cooper and J. Regehr. Pluggable abstract domains for analyzing embedded software. In *Proc. of the 2006 Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 44–53, Ottawa, Canada, June 2006.
- [2] E. A. Lee. Overview of the Ptolemy project. Technical Memorandum UCB/ERL M01/11, University of California at Berkeley, Mar. 2001.
- [3] P. Levis, D. Gay, V. Handziski, J.-H. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szweczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz. T2: A second generation OS for embedded sensor networks. Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universität Berlin, Nov. 2005.
- [4] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *Proc. of the 27th IEEE International Real-Time Systems Symposium (RTSS)*, pages 212–224, Washington, DC, USA, 2006. IEEE Computer Society.
- [5] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, Dec. 2004.