Timing Analysis of Body Area Network Applications

Yun Liang Abhik Roychoudhury Tulika Mitra
Department of Computer Science, National University of Singapore
{liangyun,abhik,tulika}@comp.nus.edu.sg

Abstract

Body area network (BAN) applications have stringent timing requirements. The timing behavior of a BAN application is determined not only by the software complexity, inputs, and architecture, but also by the timing behavior of the peripherals. This paper presents systematic timing analysis of such applications, deployed for health-care monitoring of patients staying at home. This monitoring is used to achieve prompt notification of the hospital when a patient shows abnormal vital signs. Due to the safety-critical nature of these applications, worst-case execution time (WCET) analysis is extremely important.

1. Introduction

Embedded systems based on sensor networks are widely used in many contexts. Many applications running on sensor networks have real-time constraints. For example, Body Area Network (BAN) technologies are often applied in the health-care domain. Usually, the use of BAN in health care involves several low capacity sensor nodes on the human body and a powerful gateway device like a mobile phone or PDA. The nodes in this body area sensor network communicate through wireless connections and send data to the gateway. The gateway device can monitor the situation of the patient and inform the hospital if necessary in a timely fashion. Such sensor network based applications have stringent timing requirements.

Worst-Case Execution Time (WCET) is a required input to provide timing predictability in a system with timing constraints. Therefore, given an architecture and an application, estimating the WCET is very important for the system designer. In health-care domain, monitoring the patient's situation accurately and timely is so vital that the designer must perform WCET analysis for BAN applications.

In recent times, low-end sensor nodes such as the ones from Berkeley [3] have become popular and have been deployed in many applications. A typical example of a platform used by the sensor nodes is Tmote Sky [1]. Tmote Sky is a wireless sensor module for sensor network applications that require ultra low-power and high-reliability. A number of integrated peripherals including Timer, UART bus protocols, and DMA controller are provided by Tmote Sky.

The timing behavior of a BAN application on such sensor node architectures is determined by software complexity, program inputs, hardware and timing behavior of its peripherals. Hence, timing analysis for such sensor node architectures is non-trivial. This paper presents systematic timing analysis for BAN applications by integrating the timing behavior of each component on the platform and estimating the WCET of the application. First, the timing behavior of application code is analyzed through static timing analysis. This is done by extending *Chronos*, an existing timing analyzer for embedded software [4]. Next, the timing behavior of the peripheral devices are taken into account by analyzing the interrupt handler code and estimating the number of interrupts. This turns out to be extremely important in the context of WCET analysis of applications running on BAN.

The context of our work on BAN is a major programme on health-care monitoring being funded and carried out by Singapore's Agency of Science Technology and Research (A*STAR). The programme involves collaboration among many different projects — core technologies, middleware and applications. The aim is to develop and exploit embedded system technologies for health-care monitoring of patients staying at home (such that the hospital can be notified whenever any "unusual activity" in the patient's body is detected). Typical monitoring applications that we are studying include blood-pressure computation/detection (possibly for patients with cardio-vascular disease), fall detection (monitoring for elderly patients falling to the ground) and others.

The work described in this paper focuses on (a) the typical micro-architecture deployed in sensor nodes put on the patient's body, (b) modeling and timing analy-

sis of monitoring applications running on such sensor nodes, and (c) the significance of modeling the peripherals' timing behavior to accomplish a *full-system timing analysis* of such applications.

Related Work We are aware of one recent work on WCET analysis of applications running on sensor nodes [7]. The focus of this work is on the processor modeling and handling of nesC code. The applications considered are standard ones (sorting, sum, encryption, etc.) rather than from a specific domain. In terms of peripheral modeling, we know of one recent work on modeling a system controller [8] where the author develops a timing model for WCET analysis from the VHDL description of the system.

Organization of the paper The use of BAN for healthcare is outlined in Section 2. Section 3 presents a typical application based on BAN. Section 4 summarizes the key feature of Tmote sky architecture, which is a widely used sensor network platform. Static timing analysis based on Chronos is presented in Section 5. Finally, we present the results in Section 6 and provide concluding remarks in Section 7.

2. Body Area Network for Health Care

For some patients, their health conditions need to be monitored not only when they are at hospital, but also when they stay at home. Such daily medical report is very crucial for the doctors to diagnose some chronological diseases. Wearable systems can monitor the patients's condition by sensors placed on the body. Body Area Network technologies are used in such situations.

A Body Area Network comprises of some intelligent low-power devices including biomedical sensors and storage devices. The BAN works around the human body. The sensors are able to monitor and store important biomedical information and data. The BAN can also send information to the external world through a gateway, e.g, a PDA or mobile phone.

The overall architecture in which a BAN is deployed/used is shown in Figure 1, sensors on the human body communicate with both the gateway and other sensors.

3. BAN Applications

BAN applications are low-power applications. Hence, these applications keep the processor in low-power mode as much as possible and use *interrupts* to wake up the processor when data processing is required. Also, the peripherals are switched on only when needed in order to save energy. The

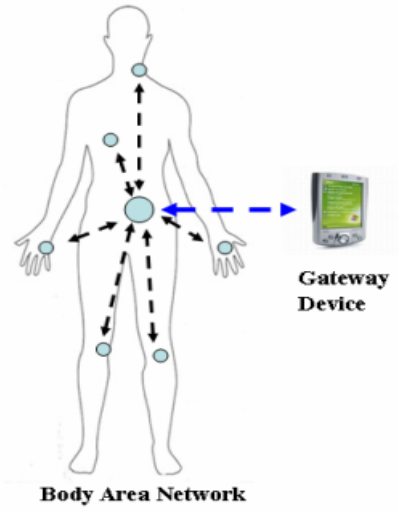

Figure 1: Body Area Network technologies for healthcare applications

high-level work-flow of a BAN application is as follows: *initialization, followed by a loop consisting of sampling and processing*, that is

initialization,
sampling, processing,
sampling, processing,
...

During sampling, the CPU could be turned off to save energy. When the sampled data is ready, it is processed by the application, and appropriate data is transmitted to remote devices. For analyzing such applications, we need to analyze the timing behavior of the main application and the interrupt handlers through static timing analysis.

The BAN application we analyze here continuously monitors the blood oxygen saturation level (SpO2) of a patient with *non-invasive* optical plethysmography also known as pulsoximeter. This measurement of the oxygen level and heart rate can be used to sound an alarm if they drop below a pre-determined level. This type of monitoring is specially useful in neonatal care and post-operative recovery. In a pulsoximeter, the estimation of blood oxygen saturation level (SpO2) is based on measuring the intensity of light that has been attenuated by body tissue. The amount of light absorbed by the body tissue depends on the oxygenation level of blood that is passing through it. Two different wavelengths of light are used — visible red wavelength and infrared wavelength.

The light intensity is sampled at a regular interval of 16ms. Every 16ms, first the red LED is turned on and the light passes through the finger of the patient to a photodiode. The intensity of light at the photodiode is

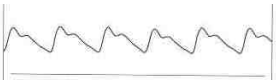

Figure 2: The waveform of sensor data.

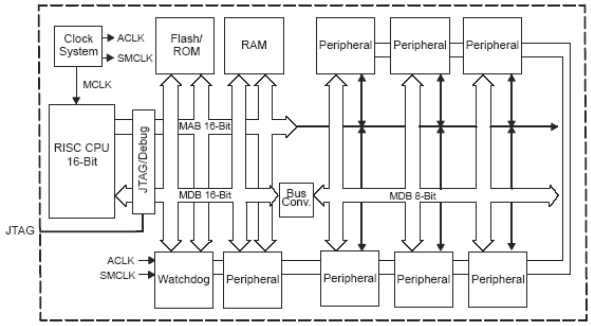

Figure 3: MSP430 Architecture ©Texas Instruments

sampled 16 times and the average value is taken. Next, the infrared LED is turned on and again the intensity of light at the photodiode is sampled 16 times to obtain an average. This process is repeated every 16ms. Figure 2 shows this average value of red light intensity attenuated by body tissue over time. The heart rate and SpO2 are estimated from these two waveforms corresponding to red and infrared light intensity. A window of a particular size is defined and the window slides on the waveforms. The application detects peaks in the waveforms inside the current window. The time interval between two peaks is used to measure both heart rate and SpO2 of the patient. These values are subsequently sent to some gateway device such as PDA that can detect abnormalities and send information to the hospital or health care provider.

4. Features of Tmote Sky Architecture

In this section, we outline the underlying micro-architecture of the sensor node platforms used by the BAN applications. Specifically, we describe the Tmote Sky platform [1], which employs the MSP430 processor from Texas Instruments. The MSP430 has a simple micro-architecture and a reduced instruction set with only 27 instructions. So, for WCET analysis, *the focus is not so much on the processor modeling.* Instead, modeling the timing effects of the peripherals turns out to be extremely important. We now describe the platform architecture in detail.

Tmote Sky is a mote platform designed for extremely low power, high data-rate, sensor network applications. The micro-controller present in Tmote Sky note is Texas Instruments's MSP430 F1611. The MSP430 incorporates a 16-bit RISC CPU, peripherals (12-bit ADC and DAC, Timer, USART, and a performance boosting DMA controller) and a flexible clock system. The architecture of MSP430 is shown in Figure 3. The current consumption of the micro-controller in low active and sleep mode is so small that an application can run for a very long time with only a single pair of AA batteries.

The features of Tmote sky platform with significant impact on the timing behavior of a BAN application are the following.

- **Flexible Clock System:** The platform includes a low-frequency auxiliary clock (ACLK) and a high-frequency master clock (MCLK). The peripherals can use different clocks.
- **Operating Modes:** MSP430 is designed for extremely low-power applications and features different operating modes distinguished by power, speed and current consumption. Operating mode can be selected by setting mode-control bits.
- **16-bit RISC CPU:** The number of CPU clock cycles required to execute an instruction depends on the instruction format and the addressing mode. All jumps instructions take two cycles to execute, regardless of whether the jump is taken or not. When computing execution cycles for interrupt handlers, the additional cycles due to interrupt overhead and reset should be taken into account.
- **Timer_A:** Timer_A is an asynchronous 16-bit timer counter with four operating modes. Clock source is configurable.
- **Timer_B:** Timer_B is identical to Timer_A with the exception that it can be programmed as 8, 10, 12, or 16 bit timer.
- **USART:** USART is used for asynchronous serial transmission and reception of characters to/from another device. The time to send/receive one character is based on the selected baud rate of the USART. Baud rate frequency is the same for both transmit and receive functions.
- **Hardware Multiplier:** The hardware multiplier is a peripheral and is not part of the 16-bit RISC CPU. The registers used by hardware multiplier are special peripheral registers.

The SpO2 application uses Timer_B, Timer_A and universal synchronous/asynchronous receive/transmit (USART). Timer_B is used to wake up the CPU and trigger Timer_A every 16ms. Timer_A is used to take 32 samples from the photodiode (16 for the red light and 16 for the infrared light) as discussed in Section 3. Timer_B and Timer_A use different clocks. USART is

used to send the heart rate and SpO2 measurements of the patient to some gateway device such as PDA.

5. Static Timing Analysis

The execution time of a program is determined by the program path taken during execution. If worst case input is known, then simulating the system with the worst case input will result in worst case execution time. However, determining the worst case input is very difficult when the application is non-trivial. Hence, static timing analysis is widely used technique to estimate worst case execution time. While many approaches have been proposed, we use Chronos [4, 5, 6] — an open-source timing analysis tool with detailed micro-architectural modeling developed by our research group. Given an architecture and an application, Chronos returns an upper bound on the execution time across all the inputs. WCET analysis in Chronos proceeds in two phases: *path-analysis* and *micro-architecture modeling*. The interested reader can get more details about (or even download) the Chronos toolkit from its website

`http://www.comp.nus.edu.sg/~rpembed/chronos`

An overview of the framework of Chronos is illustrated in Figure 4. During path analysis, Chronos constructs the control flow graph (CFG) of the application and generates functional constraints like loop bounds and flow constraints. In micro-architecture modeling, Chronos models complex micro-architectural features such as cache, pipeline, branch prediction and generate micro-architectural constraints. Following that, Chronos represents the execution time of the whole program through an Integer Linear Programming (ILP) formulation, and uses ILP/LP solver to find the maximum execution time.

We modify the Chronos toolkit in order to model the micro-controller MSP430. Chronos is targeted towards SimpleScalar PISA instruction-set architecture (ISA). As MSP430 has a different ISA, the control flow graph construction is modified. The other components in path analysis such as flow constraints generation and loop bound detection remain the same. While Chronos models cache, pipeline, branch prediction, these modeling are omitted here as MSP430 processor does not support these features. The execution cycles corresponding to an instruction is obtained by a simple table look-up with the corresponding instruction format and addressing mode. Then, the execution time of a basic block is simply the sum of the execution time of the instructions within the basic block.

The main contribution of this work is that we extend Chronos to model and analyze the timing behavior of the peripherals. For SpO2 application, the peripherals include Timer_B, Timer_A and USART. The next section describes the modeling of the peripherals in detail.

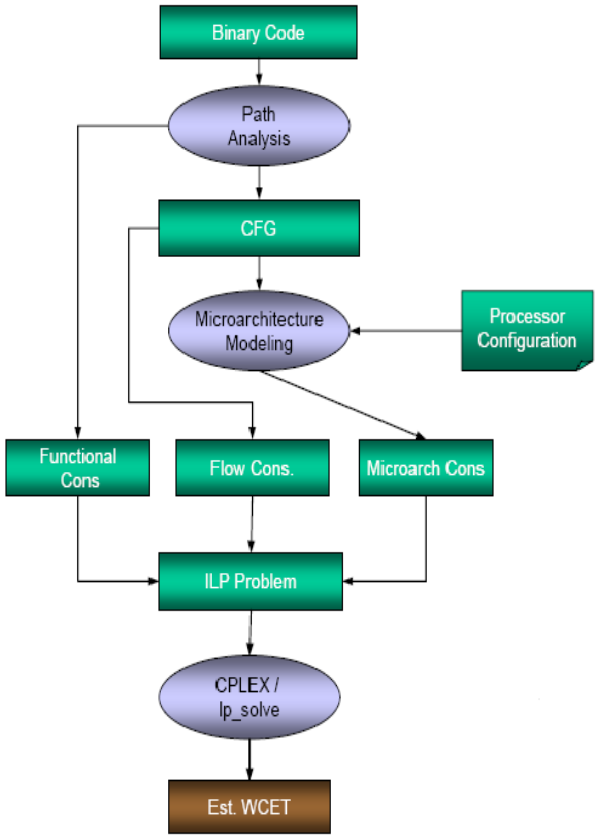

Figure 4: Workflow of Chronos Timing Analysis Tool

6. System-Level Modeling for BAN Application

The interesting aspect of any BAN application is that the sensor nodes are sampled at regular intervals through the use of timers. In other words, the number of interrupts can be estimated accurately. Therefore, the timing behavior of the main application and the interrupt handlers for Timer_A and Timer_B can be analyzed separately. We use Chronos to perform static timing analysis of the interrupt handlers and the main application individually. However, we should ensure that the timing overheads due to interrupt acceptance and return are taken into account. Finally, summing up the WCET of all the sub components yields the WCET of the whole system. Here, we define WCET as the total time required to acquire and process the samples every 16ms. The timing analysis of the interrupt handlers and the main application for SpO2 are illustrated

Component	Clock System	WCET cycles	WCET	WCET Percentage	ACET
Processing	High Speed (8 MHz)	64352	8.044 ms	61.73%	0.586 ms
Timer_B	Low Frequency (32 KHz)	38	4.75 μs	0%	4.75 μs
Timer_A	High Speed (8 MHz)	35724	4.4655 ms	34.27%	3.766 ms
USART	Asynchronous Transmission		0.52ms	4%	0.52 ms
Total			13.03ms	100%	4.872 ms

Table 1: Results of Timing Analysis of SpO2 application with interrupt

in the following.

Timer_B: Timer_B uses the low frequency auxiliary clock (ACLK) running at 32 KHz. The counter for this timer is set such that it generates an interrupt every 16ms. The functionality of Timer_B interrupt handler is to simply enable Timer_A to sample data. The WCET of Timer_B interrupt handler routine is shown in Table 1.

Timer_A: Timer_A is enabled by Timer_B. As mentioned earlier, each time Timer_A takes 32 samples from the photodiode (16 at red light wavelength and 16 at infrared wavelength). Finally, two average values are computed from these samples. The interrupt handler is invoked for each sample. However, only 2 out of these 32 invocations lead to average value computation, which is more time consuming. So, the call context is taken into account when analyzing the worst case execution time of the Timer_A interrupt handler. The WCET of Timer_A handler routine for one round (i.e., 32 samples) is shown in Table 1. When the average values have been computed, the CPU exits the low power mode and returns to active mode for data processing.

Processing: In the main application, the average values obtained from Timer_A are filtered and stored into a window. Then, the application tries to detect a peak in the middle of this window. If a new peak is detected, heart rate and SpO2 are calculated using the peak value. Finally, USART is called to send some feedback to the gateway device. From the description above, it is obvious that the worst case behavior happens when a peak is detected. The WCET of the main application (Processing) is shown in Table 1.

USART: The USART takes 1/115200 second to send 1 bit and SpO2 needs to transmit 6 bytes per round. In the USART, for every byte of data, two more bits (start bit and stop bit) are added. Therefore, a total of 60 bits are transmitted per round. The time spent in the USART is shown in Table 1. In SpO2 application, the USART transmission takes place via polling; so the time spent in the USART is the blocking time due to transmission.

From the Ratio column in Table 1, we find that the time consumed in peripherals is 38.27% for one round of acquisition and processing. *Clearly, if the timing effects of the peripherals are not modeled, the WCET of the whole system will be an under-estimation.*

The average case execution time (ACET) is shown in Table 1 too. In terms of ACET, the time for one round of data acquisition and processing is close to 4.872 ms. Clearly, the WCET value is essential here to claim that timing constraint is satisfied.

7. Conclusion

In this work, we present a systematic timing analysis framework for BAN applications. BAN applications are safety-critical and have stringent timing requirements. We analyze the timing behavior of application code and interrupt handlers for peripherals separately. Finally, the WCET of the interrupt handlers are multiplied by the number of interrupts during one round of processing and summed up with the WCET of the main data processing part to estimate the WCET of the entire system.

TinyOS [3] is an operating system designed for wireless embedded sensor networks and has been used widely for sensor nodes. NesC [2] is the corresponding language for programming sensor network applications in TinyOS. In the future, we plan to adapt our framework to provide WCET analysis for NesC code along with modeling of TinyOS.

Acknowledgments

This work was supported by NUS project R252-000-171-112 and A*Star SERC EHS-II project R-252-000-258-305.

References

[1] Tmote sky platform. http://www.moteiv.com/products/.

[2] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach

to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM Press.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104, New York, NY, USA, 2000. ACM Press.

[4] X. Li, Y. Liang, T. Mitra, and A. Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 2007.

[5] X. Li, T. Mitra, and A. Roychoudhury. Modeling control speculation for timing analysis. *Real-Time System.*, 29(1):27–58, 2005.

[6] X. Li, A. Roychoudhury, and T. Mitra. Modeling out-of-order processors for wcet analysis. *Real-Time System.*, 34(3):195–227, 2006.

[7] S. Mohan, F. Mueller, D. Whalley, and C. Healy. Timing analysis for sensor network nodes of the atmega processor family. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 405–414, Washington, DC, USA, 2005. IEEE Computer Society.

[8] S. Thesing. Modeling a system controller for Timing Analysis. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 292–300, New York, NY, USA, 2006. ACM Press.