A Hybrid Framework of Worst-Case Execution Time Analysis for Real-Time Embedded System Software

Jong-In Lee
Satellite Electronics Department, Korea Aerospace Research Institute
45 Eoeun-Dong, Yuseong-Gu, Daejeon 305-333, South Korea
+82-42-860-2819, jilee@kari.re.kr
Su-Hyun Park, Ho-Jung Bang, Tai-Hyo Kim, Sung-Deok Cha
Department of Computer Science, Korea Advanced Institute of Science and Technology
373-1, Kusong-Dong, Yuseong-Gu, Daejeon, 305-701, South Korea
+82-42-869-3535
{suhyun, hjbang, taihyo, cha}@dependable.kaist.ac.kr

Abstract—Timing analysis[1, 2] is an essential process for development of real-time embedded system and knowledge about the worst-case execution time (WCET) of real-time programs is critical to validation of temporal correctness of implemented system. Recently, automated static timing analysis methods are introduced to facilitate timing analysis process for real-time software, and to provide safe and tight WCET. But static WCET analysis methods have drawback as they do not provide accurate WCET for hardware-dependent software or application software where input data rate from external environment needs to be considered. Also, the WCET obtained from static WCET analysis needs to be verified at target system before system deployment. In this paper, we propose a framework of WCET analysis for real-time embedded software which complements static WCET approach and provides tight and safe WCET by combining static timing analysis approach with dynamic measurement. The application of proposed framework to the WCET analysis of command processing and data acquisition part of KOMPSAT-2 satellite flight software is presented to show effectiveness of the proposed approach.

TABLE OF CONTENTS

1. INTRODUCTION

Flight software is real-time embedded software which resides in spacecraft on-board computer and controls in-orbit operation of satellite. As tasks of flight software are usually designed to be scheduled in synchronous and deterministic way to prevent occurrence of unexpected faults in orbit from asynchronous scheduling, timing analysis of flight software is important. Underestimation of timing can cause overrun error endangering mission success, while overestimation can cause waste of valuable resource or degraded system performance from restricted function of codes due to limited processing resource available for space environment. Traditionally additional processor throughput margin is reserved in timing analysis to compensate for inaccuracy in timing estimation and to prepare for possible code modification to alleviate hardware failure or to fix a software bug found after launch. This conservative approach imposes timing analysis as an essential process on the development of flight software. Estimation of required CPU throughput is performed from the beginning of satellite development project to select proper CPU for required mission, and throughput usage margin requirement is usually specified in software requirements specification. Periodic timing estimation is performed during flight software development phases and throughput usage is checked with requirement at major program milestone until satellite launch.

There are two approaches of timing estimation used for development of satellite flight software, measurement and estimation by analysis. Measurement is to run a program with proper input data at target environment and measure execution time using measurement tools, such as oscilloscope, logic analyzer, or in-circuit emulator. It requires codes to execute, target system or simulator environment. But measurement is infeasible for programs with complex program execution paths, because it is difficult to execute all execution paths of industry-sized program with all possible input data. The WCET obtained by measurement is actually a lower bound to WCET. Estimation is a method to predict execution time of software by analyzing program code and characteristics of target processor without running them. Recently research on static analysis is actively in progress to automatically find tight and safe bound of WCET by analyzing program codes and modeling processor hardware characteristics. But

[1] 0-7803-8870-4/05/$20.00© 2005 IEEE
[2] IEEEAC paper #1285, Version 2, Updated December 1, 2004

currently available automatic WCET analysis methods have restrictions on program size and programming language constructs supported (e.g. no indirect procedure calls, no dynamic data structures, no recursive functions, etc.) or require additional user annotations to be provided. It also has drawback that it cannot accurately estimate WCET of hardware-dependent codes because timing behavior of real-time embedded system depends on the hardware characteristics of specific target system, such as DMA, interrupts, or shared memory, nor can it take into account of feasible processing load scenario of the system (e.g., maximum amount of external input from environment). These constraints/limitations of static WCET analysis necessitate finally validating correctness of static WCET analysis result by measurement in the target system.

This paper presents a framework to perform worst-case execution time analysis for real-time embedded systems on the premise that dynamic timing analysis is required at final stage of timing analysis. Section 2 introduces traditional timing analysis methods used for the development of real-time embedded software and presents their problems taking satellite flight software as an example. Section 3 provides overview of static WCET analysis methods for real-time embedded systems. In Section 4 we propose a framework of hybrid WCET analysis for real-time embedded system software. Application of the proposed framework and experimental results are presented in Section 5. Finally Section 6 gives a conclusion and outlines future plan.

2. TIMING ANALYSIS OF FLIGHT SOFTWARE

Many spacecraft today contains on-board computers which automatically maintain spacecraft's attitude, periodically check safety of the spacecraft hardware, support the science instruments, and perform a variety of other tasks. Flight Software which runs on spacecraft on-board computer is real-time embedded software. It's mission-critical software whose correct system functionality depends both on logical correctness and temporal correctness.

Flight software timing analysis is crucial for the following purposes:

Firstly, some satellite control functions are required to be executed within a predefined time duration, which requires prior knowledge about their execution times. Generally, satellite flight software tasks are scheduled in predictable ways, i.e., they should be executed in a periodic or synchronous ways to reduce faults caused from asynchronous, unpredictable scheduling. Their scheduling is usually based on a predefined duration of time called 'minor cycle' or 'major cycle', and tasks allocated for spacecraft subsystems need to be synchronized. Also in a distributed multiprocessor system, inter-processor communication and execution of tasks in each processor need to be synchronized.

Secondly, there are timing constraints for a task to process inputs from external environment and to respond them.

Thirdly, software timer function or delay routines is required for certain application where hardware timer is not supported.

Fourthly, satellite flight software is required to have sufficient timing and sizing margin to prepare for unpredictable asynchronous events in space environment and to meet unexpected software modification during in-orbit operation. This imposes periodic check and trace of throughput and memory usage during flight software development because those margin requirements at launch time are usually specified in software requirements specification document [15].

Figure 1 shows these requirements of flight software timing analysis.

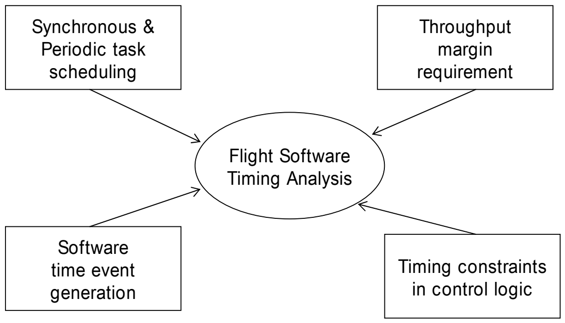

Synchronous & Periodic task scheduling
Throughput margin requirement
Flight Software Timing Analysis
Software time event generation
Timing constraints in control logic

Figure 1 - FSW Timing Analysis Requirements

Traditional Flight Software Timing Analysis Approaches

Timing analysis for satellite flight software is performed in an incremental, iterative way as shown in Figure 2. Required processor throughput is roughly estimated during system requirements analysis and conceptual design phase. This estimation is performed in system-level, and analysis result is used for selection of CPU type or for trade study of hardware/software design. During design phase, software timing is predicted or estimated using pseudo code or heritage code. Execution time is estimated roughly by counting number of floating-point operations or lines of pseudo code. A processor simulator is used to measure execution time of code during coding and unit testing phase. Usually WCET of program is estimated in bottom-up manner starting measurement of leaf node modules in call tree. To find the WCET of code, the longest execution time path is searched explicitly by running the executable paths of code using simulator. The selection of execution paths or test cases largely depends on programmer's heuristic decision if there are too many paths to execute. To measure time-critical function, oscilloscope or logic analyzer is used in this phase. During software integration and test phase, WCET of task is measured by running code in target system using in-circuit emulator, oscilloscope, or logic analyzer. The input data used for WCET measurement is chosen by programmer assuming feasible processing load scenario. The WCET of integrated software is obtained by considering functional or data dependency between modules. Usually domain-specific knowledge is utilized in selecting execution path or test cases during this process. The timing profile data obtained at this phase can be used for code optimization of critical section. During verification test phase, WCET of overall flight software executing in the on-board computer is measured under the maximum feasible processing load to verify throughput usage requirement.

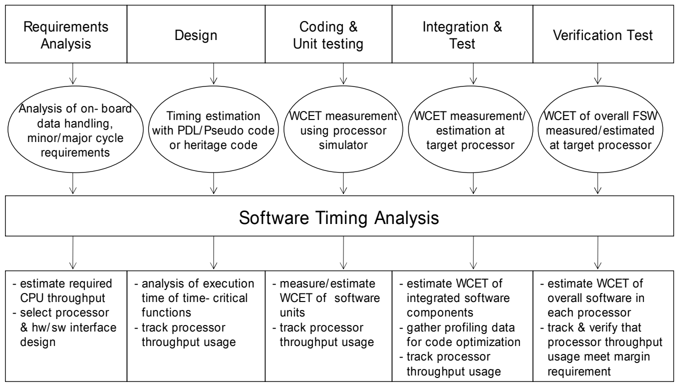

Requirements Analysis
Design
Coding & Unit testing
Integration & Test
Verification Test
Analysis of on- board data handling, minor/major cycle requirements
Timing estimation with PDL/Pseudo code or heritage code
WCET measurement using processor simulator
WCET measurement/ estimation at target processor
WCET of overall FSW measured/estimated at target processor
Software Timing Analysis
- estimate required CPU throughput
- select processor & hw/sw interface design
- analysis of execution time of time- critical functions
- track processor throughput usage
- measure/estimate WCET of software units
- track processor throughput usage
- estimate WCET of integrated software components
- gather profiling data for code optimization
- track processor throughput usage
- estimate WCET of overall software in each processor
- track & verify that processor throughput usage meet margin requirement

Figure 2 - FSW Timing Analysis Flow

Problems in Traditional Flight Software WCET Analysis

A WCET estimation method generally used for timing analysis of flight software was to search for all the executable paths of program by the programmer, and then he/she finds test cases for each path to run the program using target processor simulator or in-circuit emulator. The number of instruction cycles executed for each executable path is obtained and the largest is chosen as WCET of the program. The WCET of modules corresponding to the higher nodes of call tree is obtained similarly in bottom-up manner by finding test cases first then executing them. This manual approach requires much effort and time of programmers and error-prone because a flight software module usually contains many executable paths and generating test cases for them to measure the execution times of the paths is very hard.

3. Overview of Static WCET Analysis

Static Worst-Case Execution Time (WCET) analysis is to provide a priori knowledge about the worst-case execution time of a program without running it, while dynamic timing analysis is based on measurement of execution time [1, 2, 3]. The WCET provided by static WCET analysis methods should be safe and tight estimation of WCET as shown in Figure 3.

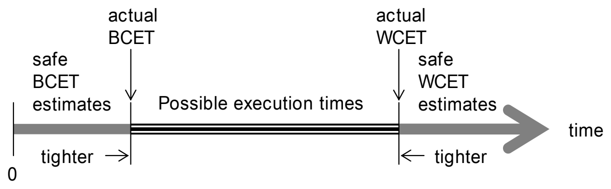


Figure 3 - Approximate WCET

Static WCET analysis proceeds through phases of program flow analysis, low-level analysis and calculation as shown in Figure 4. The program flow analysis phase analyzes the code of the program, and determines the possible program flows. It provides information about which functions get called, how many times loops iterate, if there are dependencies between if-statements, etc., by automatically analyzing program's dynamic behavior or by user annotations. The low-level analysis phase analyzes the object code and target hardware to determine the timing behavior for instructions running on the target hardware, giving the execution time for each atomic unit of flow (i.e., basic block). The calculation phase combines the results of the flow and low-level analyses to calculate a WCET estimate for the program. There are three categories of calculation methods: tree-based, path-based, and Implicit Path Enumeration Technique (IPET)-based calculation.

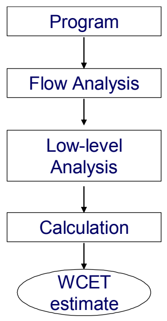


Figure 4 - Static WCET Analysis Phases

Program Flow Analysis

The purpose of program flow analysis it to get information about which functions are called, how many times loops iterate, if there are dependencies between if-statements, etc [4]. The flow information such as maximum loop iteration, branch constraints, and infeasible paths can be extracted from program code by automatic flow analysis methods or with manual annotations. It has three sub-phases: flow extraction, flow representation, flow information conversion for calculation. Flow extraction actually determines flow of code, flow representation represents the information obtained in the analysis phase, and flow information conversion for calculation is to process the information to be useful for the particular calculation method. The automatic flow analysis is still limited to well-structured programs which do not use pointers, dynamic data structures, or recursion. In this case, automatic flow analysis needs to be complemented with manual annotations providing additional flow information such as bound of loop iteration and description of flow dependencies. Further research on automatic flow analysis to get tighter WCET is in progress by detecting loop bounds or infeasible paths automatically instead of manual annotations which may be error prone [5, 6, 7].

Low-Level Analysis

The low-level analysis is to determine the execution time of basic blocks given the architectural feature of the target hardware. It determines the timing effect of machine-dependent factors that need to be modeled such as cache, branch predictor and pipeline. But as modern processors utilize various performance-optimizing features to enhance performance, modeling of processors' timing behavior becomes more complex and hard to predict. Research on modeling complex processor's timing behavior as well as validation of the hardware model is in progress [8].

Calculation

Calculation phase is to calculate the WCET estimate for the program, given the program flow and low-level analysis results. There are three categories of calculation methods: tree-based, path-based and IPET (Implicit Path Enumeration Technique).

(1) Tree-based Calculation.

In tree-based calculation, the WCET estimate is generated by a bottom-up traversal of a program syntax tree [9]. The program syntax tree is a representation of program whose non-leaf nodes corresponds to structure of the program (e.g., sequences, loops, conditionals) and whose leaf-nodes represent basic blocks. This method is simple and computationally cheap, while it has drawback that the computation is local within a single program statement and cannot consider dependencies between statements.

(2) Path-based Calculation.

Path-based calculation is to find longest execution time path using graph search algorithm after converting source codes into control flow graph [10]. This method explicitly represents possible execution paths but has problems with handling flow information of loop-nesting levels.

(3) IPET Calculation

IPET (Implicit Path Enumeration Technique) calculates WCET by solving the objective function which satisfies structural or functional constraints extracted from program CFG or provided by user [11, 12].

$$WCET = MAX\left(\sum_{i}^{N} t_i x_i\right)$$

where x_i is the execution count of basic block B_i,
t_i is the execution time of basic block B_i

The result of IPET calculation is a worst-case count for the basic block instead of explicit execution paths. IPET constraints systems can be solved using constraint-solver or ILP (Integer Linear Programming) technique. IPET-based approach can handle more complex flow information compared to other calculation methods. Comparison with path-based calculation is performed in [13].

Issues and Research Area in Static WCET analysis

Modeling the timing behavior of modern microprocessors becomes more complex as they utilize performance-optimizing features such as pipeline, caching and branch prediction. Recently research on precisely modeling these advanced features is actively in progress. However it becomes more difficult to obtain the detailed timing characteristics of processor internals because processor manufacturers are reluctant to release them for competition with other manufacturers, and this make it difficult to validate target processor's timing model. Also fully automated static WCET analysis tools which support most of modern embedded processors and can be applied to real-world programs without reprogramming or user annotation are not developed yet. Research on integrating WCET tool with existing compilers is actively in progress, too. Research on the use of evolutionary algorithms in assessment of execution time, research on parametric timing analysis, research on probabilistic approach for WCET analysis, and research on WCET analysis for component-based software are in progress.

4. HYBRID WCET ANALYSIS FRAMEWORK

Static WCET analysis is a technique to find out WCET by analyzing software codes without executing it. But this static WCET analysis has drawback compared to dynamic WCET analysis technique which measures WCET by running the software at target system environment.

Limitation in analyzing WCET of hardware-dependent software

For example, data acquisition module shown in Figure 5 which polls hardware ready status after writing request waits for data ready up to predefined timeout duration. In most of nominal cases, hardware data is available much earlier than the timeout which is set conservatively with sufficient margin. But with static WCET analysis method which calculates WCET theoretically, the path which takes timeout is always chosen causing overestimation compared to real-world.

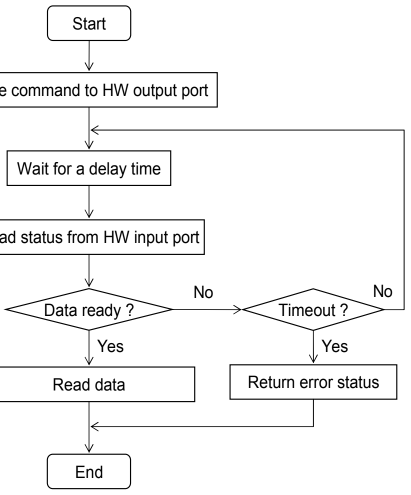


Figure 5 - Timeout Loop for Hardware Polling

Another example is target system specific features, such as access to shared memory, DMA, or interrupt. These features cannot be considered in static WCET analysis technique, even though non-preemptive scheduling is assumed. Timing effect due to contention in accessing shared memory between CPU and I/O controller cannot be covered in low-level analysis, which considers only internals of CPU such as pipeline, cache, or branch prediction.

Lack of consideration for constraints on input data from environment

Typically real-time embedded systems acquire input from environment, process them, and control system hardware to perform their mission. The program execution time in the system is dependent on input data (amount of input data, type of input data) from environment. To estimate the WCET of real-time embedded system, we need to consider the constraints on input data from environment. Without considering the constraints on input data, the WCET obtained from the static analysis technique will be unacceptably overestimated.

Requirement to validate correctness of static WCET result

Static WCET analysis techniques estimate WCET by analyzing program codes. But the WCET result from static analysis need to be validated before its application to design of real-time embedded system since the estimation process often involves human interaction such as user annotation which may be prone to error.

Therefore the timing analysis for real-time embedded system requires validation at target system by actually executing software and measuring execution time before system deployment. But time and effort to find out WCET by measurement is inhibitive to perform because it is difficult to execute all execution paths of industry-sized program with all possible input data.

This paper introduces efficient WCET analysis method which provides tighter WCET by utilizing constraints on input data from environment. It also presents hybrid WCET analysis technique which combines static and dynamic WCET analysis techniques to validate correctness of timing behavior of real-time embedded system.

(1) Find longest execution time path using path-based static WCET analysis technique.

(2) Generate input data which traverses the longest execution path.

(3) Extract constraints on input data by analyzing system specification, equipment specification, and interface control documents, etc.

(4) Measure WCET by running software at target environment with input data obtained as above.

(5) Compare the result of static and dynamic WCET analysis.

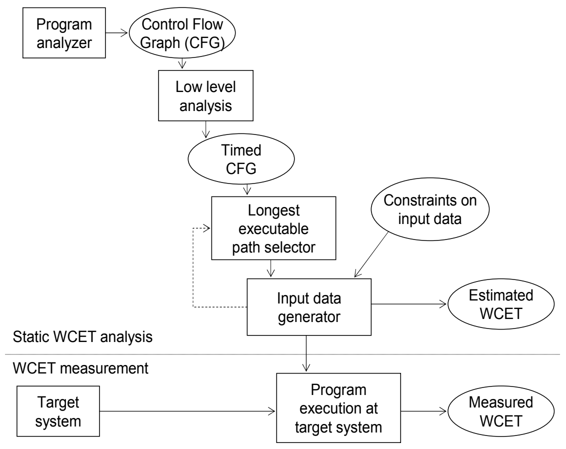


Figure 6 - Proposed WCET Analysis Method

Search of the Longest Execution Path using Static WCET Analysis Technique

Path-based WCET analysis technique is preferred to IPET because the longest execution path is provided in addition to estimated WCET. After flow analysis and low-level analysis of software, timed CFG is generated. The longest execution path in the timed CFG is searched using graph search algorithm. As the number of execution paths of a program grows exponentially with the number of control flow branches and loops, it is difficult to check executability of all execution paths. In searching the longest execution path feasible, [14] suggested a method to find k longest execution paths using heuristics, then to check their feasibility starting from longest execution path first. The first feasible execution path among the k longest execution paths is chosen as WCET path. If all the k execution paths are infeasible, then k-th path is chosen as WCET path.

Generation of Input Data Traversing WCET Path

Path-oriented automated test data generation can be used to find input data which traverses WCET path. The input data generation problem is converted to problem of finding solution of linear systems of equality/inequality of input data. Input data should include global variables in addition to input arguments of function/procedure.

Analysis of Constraints on Input Data

The WCET of real-time embedded system often depends on input data from environment. Figure 7 shows a typical structure of the input data processing task in a real-time embedded system.

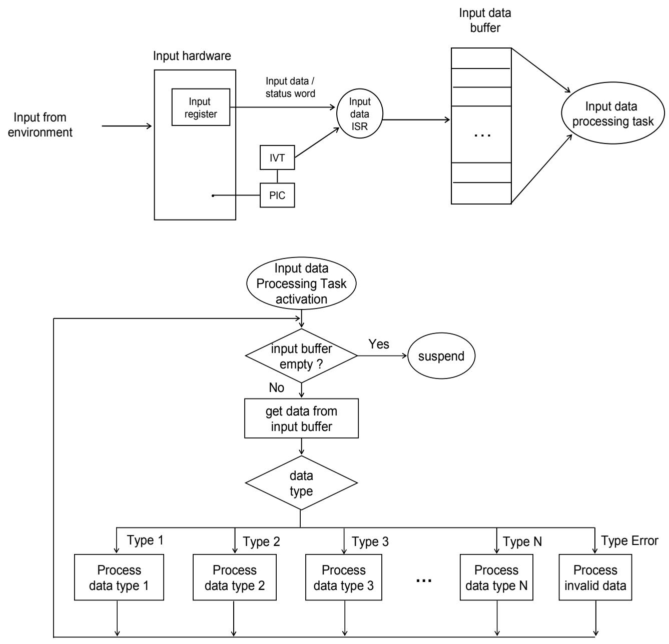


Figure 7 - Input Data Processing Task

Input data from environment is acquired and stored in the input data buffer by interrupt or polling scheme for processing by input data processing task. Usually input data buffer size is set with sufficient margin to prevent buffer overflow. When activated, the input data processing task processes input data in the buffer according to type of input data. The execution time of input data processing task depends on amount and type of input data in the buffer. If there are predefined patterns on input data or maximum input data rate is defined, then disregarding these constraints in WCET estimation will result in overestimation. Therefore constraints on input data such as amount of input data, type of input data, predefined pattern of input data, need to be considered in WCET estimation. As these constraints are not included in program codes, domain knowledge is required to extract them from documents, such as system specification, equipment specification, or interface control document, etc. Constraints on input data can be categorized into following criteria:

- Maximum input data rate from environment/user
- Predefined input data pattern
- Exception handling

These constraints can be specified in flow facts language as proposed in [12].

(1) Representation of Maximum Input Data Rate

Maximum input data rate can be represented as loop bound. Input data processing task of real-time embedded software usually processes input data in the input buffer until it is empty. The input buffer size is set large enough to hold maximum input data from environment with sufficient margin, but actual amount of input data from environment is usually much smaller than capacity of input data buffer. Without considering constraints on input data, static WCET analysis technique overestimates WCET of input data processing task. Constraint of the maximum input data rate can be given as specifying loop bound of input processing task in static WCET analysis tool.

For example,

$$scope : [\ ] : x_{header\ (scope)} \leq bound$$

(2) Representation of Predefined Input Data Pattern.

In a real-time embedded system, input data from environment can have predefined data pattern. Communication protocol processing software is an example, where input data is always in a predefined format as shown in Figure 8.

Frame Header	Packet Header	Data 1	Data 2	Data 3	...	Data N

Figure 8 – Example of Input Data of Predefined Pattern

The execution time of input data may be different according to input data type. Most of static WCET techniques choose the path which processes the input data type with the longest execution time regardless of predefined input data pattern, resulting in overestimated WCET.

For example, if we assume in Figure 7 that:

– Max loop count of input data processing task: *Max_Loop*

– WCET of codes "process data type *i*" : w_i, $1 \le i \le N$

– input data has predefined data pattern: *Type-1 data* → *Type-2 data* → *Type-m_1 data* → *Type m_2 data* → ••• → *Type-m_n data*, where $m_i \in [3..N]$ and $1 \le n \le Max\_Loop - 2$

Then actual WCET is $w_1 + w_2 + n * max(w_j), 3 \le j \le N$, instead of $Max\_Loop * max(w_i), 1 \le i \le N$.

These constraints can be specified in flow facts language as follows:

$$scope : <1..1> : x_{TYPE-1} = 1$$

$$scope : <2..2> : x_{TYPE-2} = 1$$

$$scope : [3..Max\_Loop] : x_{TYPE-1} + x_{TYPE-2} = 0$$

If we further assume that input Type-3 and Type-4 cannot appear at the same time:

$$scope : [] : x_{TYPE-3} + x_{TYPE-4} \le 1$$

(3) Representation of Constraint on Exception Handling

Error checking and logging is important to detect, isolate, and recover from fault in real-time embedded system. If error or exception occurs, error information such as error code, time of occurrence, and error description, is logged in the error table in addition to required exception handling. In static WCET analysis, the path to invoke error handling is chosen as WCET path if execution time to handle error is larger than that of normal operation. However occurrence of faults all the time is very unlikely and exception handling usually involves aborting input data stream in the buffer until new valid input data sequence appears. If static WCET analyzer considers subsequent input data as error without discarding them, the WCET will be overestimated. Figure 9 shows an example of overestimated WCET in exception handling at data acquisition task which gathers hardware sensor data.

– It invokes *read_hardware_data* routine to read sensor data from hardware

– In *read_hardware_data* routine, sensor data is acquired from A/D converter hardware

- Output request command (channel number, gain, offset)
- Wait for data ready (until *timeout*)
- Read data
- If data read timeout occurs, log timeout error in the error table
- Usually *timeout* is set conservatively with sufficient margin (10 times of nominal case)

The estimated WCET by static WCET analyzer will be overestimation, because the probability that all data acquisitions result in timeout is nearly impossible in practice considering the margin of timeout.

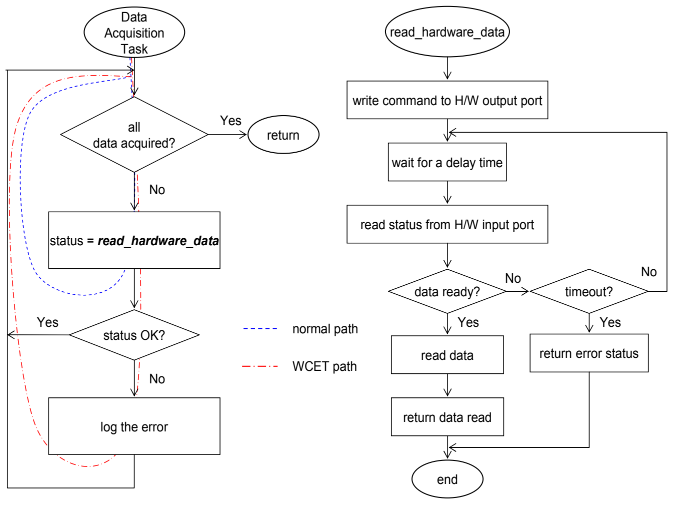


Figure 9 - Example of Exception Handling

WCET Measurement at Target System

After the WCET has been estimated along with the longest execution path and the input data which traverse it, verification of the result at target system before system deployment is important. As input data for WCET path is available, the execution time can be measured easily at target system using in-circuit emulator, digital oscilloscope, or logic analyzer. The measurement should be non-intrusive, so as not to affect execution time of program running. This measurement at target system will uncover system-dependent hardware timing effect, such as decrease of CPU throughput due to contention in accessing shared memory with I/O controller, DMA, or interrupt.

5. Case Study

We performed a case study according to proposed WCET framework by applying it to estimate WCET of KOMPSAT satellite flight software components: CCI and DAQ. Command and communication Interface (CCI) receives and processes telecommands from ground. Telecommands from ground are in CLTU (Command Link Transmission Unit) format of CCSDS (Consultative Committee for Space Data Systems) recommendation. It consists of 17 software units and total source line of codes is 979. Data Acquisition (DAQ) acquires telemetry data to be used by application software from serial, analog, or parallel ports, and formats them according to predefined telemetry format for downlink to groundstation. It consists of 5 units and total source line of codes is 583. Both CCI and DAQ are programmed in C-language.

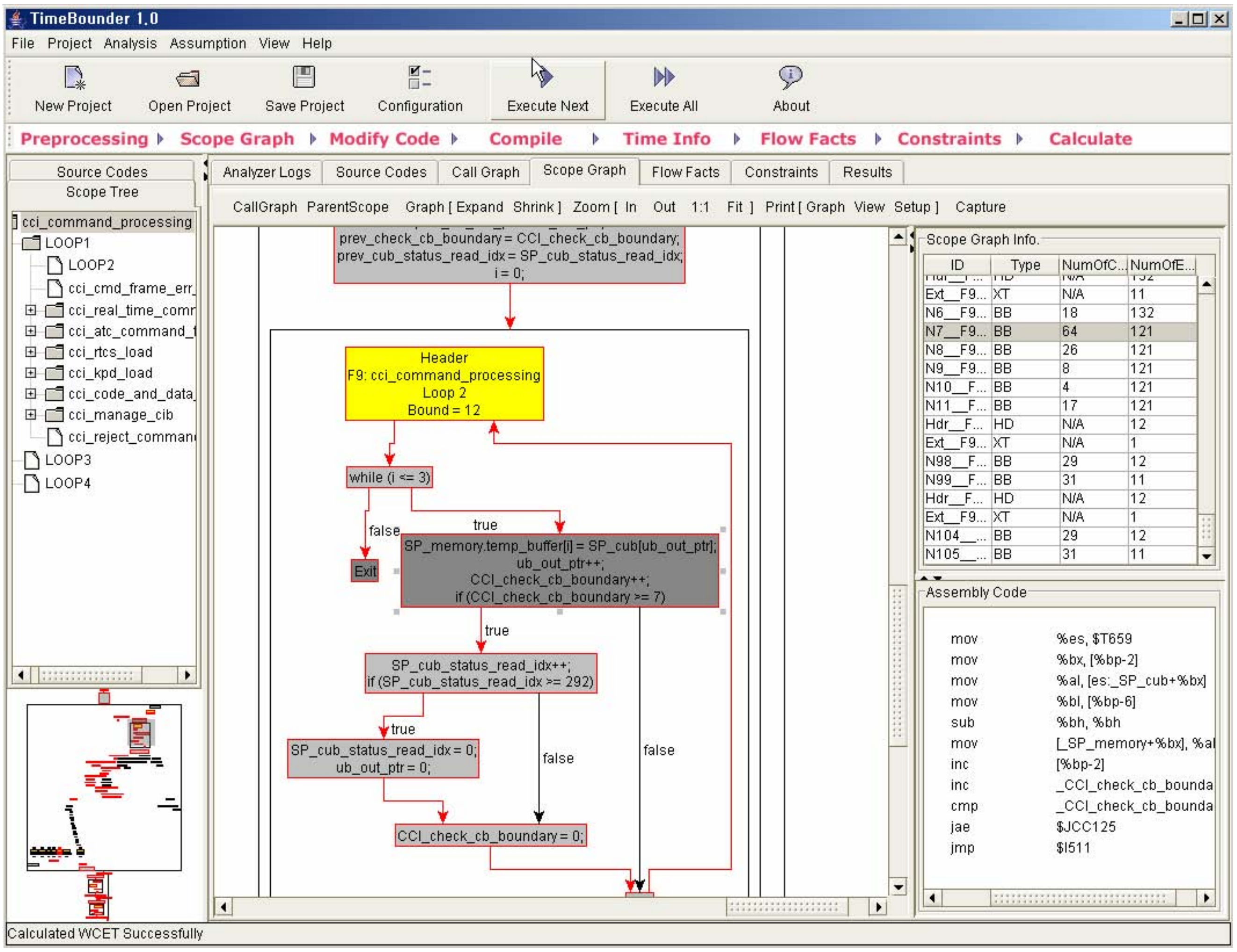
The WCET acquired by traditional measurement was compared to the estimated WCET obtained using a static WCET analysis tool called TimeBounder which has been developed at KAIST. TimeBounder takes C source code and user defined flow constraints as its input, and returns WCET with the number of executions of each basic block. Figure 10 shows a screenshot of the tool. Current prototype version supports only Intel 80386 target processor.

Then the tighter WCET obtained by applying constraints on input data provided by domain engineer was compared to them. The representative result of the experiment is shown in Table 1.

Table 1. Summary of Case Study

Module name	File Name	Measurement (cycles)	Timebounder 1.0 Estimation (cycles)			
			Method-1	Ratio-1	Method-2	Ratio-2
cci_command_processing	CCI_C003.C	320,052	1,787,030	558%	373,474	117%
cci_ccsds_frm_processing	CCI_C016.C	323,928	3,649,220	1127%	406,515	125%
daq_read_hardware_data	DAQ_RDHW.C	39,135	2,395,010	6120%	46,210	118%

Column Method 1 represents result of static WCET without applying the proposed method and column Method 2 is a result obtained after applying constraints on input data. Measurement was performed at target system using in-circuit emulator with input data obtained from static analysis. As the tool does not provide explicit WCET path, we have to identify the WCET path and input data for measurements manually with help of GUI information



```
prev_check_cb_boundary = CCI_check_cb_boundary;
prev_cub_status_read_idx = SP_cub_status_read_idx;
i = 0;

while (i <= 3)

SP_memory.temp_buffer[i] = SP_cub[ub_out_ptr];
ub_out_ptr++;
CCI_check_cb_boundary++;
if (CCI_check_cb_boundary >= 7)

SP_cub_status_read_idx++;
if (SP_cub_status_read_idx >= 292)

SP_cub_status_read_idx = 0;
ub_out_ptr = 0;

CCI_check_cb_boundary = 0;
```

Scope Graph Info.

ID	Type	NumOfC...	NumOfE...
Ext__F9...	XT	N/A	11
N6__F9...	BB	18	132
N7__F9...	BB	64	121
N8__F9...	BB	26	121
N9__F9...	BB	8	121
N10__F...	BB	4	121
N11__F...	BB	17	121
Hdr__F...	HD	N/A	12
Ext__F9...	XT	N/A	1
N98__F...	BB	29	12
N99__F...	BB	31	11
Hdr__F...	HD	N/A	12
Ext__F9...	XT	N/A	1
N104__...	BB	29	12
N105__...	BB	31	11

Assembly Code

```
mov     %es, $T659
mov     %bx, [%bp-2]
mov     %al, [es:_SP_cub+%bx]
mov     %bl, [%bp-6]
sub     %bh, %bh
mov     [_SP_memory+%bx], %al
inc     [%bp-2]
inc     _CCI_check_cb_bounda
cmp     _CCI_check_cb_bounda
jae     $JCC125
jmp     $I511
```

Figure 10 - TimeBounder 1.0

provided by the tool. In the experiment, we could get almost an order-of-magnitude tighter WCET estimation.

6. Conclusion

WCET analysis is critical to design and validate real-time system software. Traditional timing analysis techniques depend on measurement which requires much time and effort to find proper test cases, to execute and measure their execution time. The WCET obtained by measurement is usually not safe either due to large input data space infeasible to try. The static WCET analysis techniques are introduced to resolve the problems of traditional dynamic timing analysis techniques, and to facilitate timing analysis process for real-time software. But static WCET analysis methods also have drawbacks. They overestimate WCET of real-time embedded system where there are constraints on input data or hardware dependency which may not be analyzable from the code itself. Furthermore, the estimated WCET from static analysis needs to be ultimately verified at target system before system deployment.

We propose an efficient WCET analysis method which provides tighter WCET by utilizing constraints on input data from environment. The proposed hybrid WCET analysis framework provides tight estimation of WCET using static WCET analysis at early phase of system development, and facilitates verification and validation of timing constraints of real-time embedded system at target system during implementation and verification phase of system development. We could show effectiveness of the approach through a case study of satellite flight software.

As future work, we plan to investigate further constraints on input data for real-time embedded system and categorize them for tighter estimation of WCET. We also plan to study on automatic generation of input data which corresponds to obtained WCET path for measurement at target system.

Acknowledgments

This work was partly supported by the KOMPSAT-2 project Contract (Spacecraft Bus Design and Development) of the Korean Ministry of Science and Technology. It was also supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc), and the Information Technology Research Center (ITRC).

References

[1] A. C. Shaw, "Reasoning About Time in Higher-Level Language Software," IEEE Transactions on Software Engineering, Vol. 15, Issue 7, Pages 875-889, July 1989.

[2] Chang Yun Park, Alan C. Shaw, "Experiments with a Program Timing Tool Based on Source-Level Timing Schema," IEEE Computer, Vol. 24, Issue 5, Pages 48-57, May 1991.

[3] P. Puschner, Ch. Koza, "Calculating the maximum execution time of real-time programs," Real-Time Systems, Vol. 1, Issue 2, Pages 159-176, September 1989.

[4] Andreas Ermedahl, A Modular Tool Architecture for Worst-Case Execution Time Analysis, Dissertation for the Degree of Doctor of Philosophy in Computer Systems, Uppsala University, June 3, 2003.

[5] C. Healy, M. Sjödin, V. Rustagi, D. Whalley, "Bounding loop iterations for timing analysis," IEEE Real-Time Applications Symposium (RTAS'98), June 1998.

[6] Christopher Healy, Mikael Sjödin, Viresh Rustagi, David Whalley, Robert Van Engelen, "Supporting Timing Analysis by Automatic Bounding of Loop Iterations," Real-Time Systems, Vol. 18, Issue 2-3, Pages 129-156, May 2000.

[7] Christopher A. Healy, David B. Whalley, "Automatic Detection and Exploitation of Branch Constraints for Timing Analysis," IEEE Transactions on Software Engineering, Vol. 28 , Issue 8, Pages 763-781, August 2002.

[8] Greger Ottosson, Mikael Sjödin, "Worst Case Execution Time Analysis for Modern Hardware Architectures," Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS'97), June 1997.

[9] Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung-Do Rhee, Sang Lyul Min, Chang Yun Park, Heonshik Shin, Kunsoo Park, Soo-Mook Moon, Chong Sang Kim, "An Accurate Worst Case Timing Analysis for RISC Processors," IEEE Transactions On Software Engineering, Vol. 21, No. 7, July 1995

[10] Peter Puschner, Anton Schedl, "Computing Maximum Task Execution Times -- A Graph-Based Approach," Journal of Real-Time Systems, Vol. 13, No. 1, Pages 67-91, July 1997.

[11] Yau-Tsun Steven Li, Sharad Malik, "Performance analysis of embedded software using implicit path enumeration," Proceedings of the 32nd ACM/IEEE conference on Design automation conference, Pages 456-461, 1995.

[12] Jakob Engblom, Andreas Ermedahl, "Modeling Complex Flow for Worst-Case Execution Time Analysis," Proc.21th IEEE Real-Time Systems Symposium (RTSS '00), November 2000.

[13] Jakob Engblom, Andreas Ermedahl, Friedhelm Stappert, "Comparing Different Worst-Case Execution Time Analysis Methods," The 21st IEEE Real-Time Systems Symposium (RTSSWIP00), November 27, 2000.

[14] Friedhelm Stappert, Peter Altenbernd, "Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs," Journal of System Architecture, 46, No. 4, Pages 339-355, 2000.

[15] Jong-Wook Choi, Soo-Yeon Kang, Jong-In Lee, KOMPSAT-2 Flight Software Sizing and Timing Report, KOMPSAT Program Office, Korea Aerospace Research Institute, June 20, 2003.

BIOGRAPHY

Jong-In Lee *is a head of Satellite Electronics Department at Korea Aerospace Research Institute. He has participated in KOMPSAT-1&2 projects. He received MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST). He is a part-time PhD student in electrical engineering and computer science department at KAIST.*

Su-Hyun Park *received BS degree in computer science from Kyungpook National University in 1999. She is a MS student at KAIST. Research interests are Software Engineering and Model Checking.*

Ho-Jung Bang *received BS degree in economics from Seoul National University in 1996. He received MS degree in computer science from KAIST in 2003. He is a PhD student at KAIST. Research interests are Software Engineering and Formal Method.*

Tae-Hyo Kim *received BS and MS degree in computer science from KAIST in 1998 and 2000, respectively. He is a PhD student at KAIST. Research interests are Formal Method and Model Checking.*

Sung-Deok Cha *is an associate professor of computer science division of electrical engineering and computer science department at KAIST. He received PhD degree in information and computer science at University of California, Irvine in 1991. Research interests are Software Safety, Computer Security, and Formal Method.*