This item is the archived peer-reviewed author-version of:

Powerwindow : a multi-component TACLeBench benchmark for timing analysis

Reference:

Li Haoxuan, De Meulenaere Paul, Hellinckx Peter.- Powerwindow : a multi-component TACLeBench benchmark for timing analysis

Advances on P2P, parallel, grid, cloud and internet computing : proceedings of the 11th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC2016), November 5-7, 2016, Soonchunhyang University, Asan, Korea - ISSN 2367-4512 - Springer International Publishing, 2017, p. 779-788

Full text (Publishers DOI): http://dx.doi.org/doi:10.1007/978-3-319-49109-7_75

Powerwindow: a Multi-component TACLeBench Benchmark for Timing Analysis

Haoxuan Li
CoSys-Lab, MOSAIC
University of Antwerp
Belgium
Haoxuan.Li@uantwerpen.be

Paul De Meulenaere
CoSys-Lab, Flanders Make
University of Antwerp
Belgium
Paul.Demeulenaere@uantwerpen.be

Peter Hellinckx
MOSAIC, iMinds
University of Antwerp
Belgium
Peter.Hellinckx@uantwerpen.be

Abstract—Timing analysis is used to extract the timing properties of a system. Different timing analysis techniques and tools have been developed over the past decades. Well-rounded benchmark suites are in a high demand for the evaluation of those techniques and tools. TACLeBench is aiming to provide an available and comprehensive benchmark suite to fulfil the requirement. It is not only limited to single-core systems but also will be replenished with complex benchmarks for multi-core systems.

In this paper, we propose a multi-component benchmark which can be easily converted to benchmarks for single-core, parallel, distributed or even multi-core systems without fundamental changes.

I. INTRODUCTION

Real-time embedded systems are widely adopted in applications such as automotive, avionics and medical care. Many of these systems are required to meet strict deadlines. Understanding the timing behaviour of such systems at the early design stage can reduce the number of design iterations, which ultimately reduces the design cost and time-to-market. Timing analysis is the process of deriving or estimating the timing properties of a system.

Multi-core platforms have taken over the single-core systems in industry [1]. This evolution is triggered by the increasing performance requirements and the intention of reducing the number of ECUs (Electronic Control Unit) used in one system [2]. However, the interactions between different tasks running in parallel increases the complexity of timing analysis. The changes in hardware platform and software architecture introduced new challenges in timing analysis techniques.

Different timing analysis techniques and tools have been developed over the past decades [3]. Well-rounded benchmark suites are in a high demand for the evaluation of those techniques and tools. The current existing benchmark suites for code level timing analysis, however, do not cover multi-core software. TACLeBench [4] is aiming to provide an available and comprehensive benchmark suite to meet the requirement. It is not only limited to single-core systems but also will be replenished with complex benchmarks for multi-core systems in the near future.

Powerwindow is a multi-component benchmark included in the TACLeBench. It contains four main components including one driver-side window and three passenger-side windows.

Fig. 1. Power window [5].

These components allow the benchmark to be easily converted to benchmarks for single-core, parallel, distributed or even multi-core systems without fundamental changes. The goal of this paper is to prove that the powerwindow benchmark is suitable to be used by different timing analysis techniques according to the requirements. In this paper, we performed measurement-based timing analysis with three case studies covering the benchmark in isolation, the benchmark with interactions and the benchmark with interactions in semi-parallel.

The rest of this paper is structured as follows. Section II gives a detailed description about the powerwindow benchmark. Section III provides an overview of the case studies. The results are elaborated and compared in Section IV followed by discussion in Section V. The conclusion and the future work are covered in Section VI and Section VII respectively.

II. POWERWINDOW BENCHMARK

A. Power Window Description

A power window is an automatic window which can be raised and lowered by pressing a button or a switch instead of a hand-turned crank handle (Fig. 1). The complete application of a power window in a car normally consists of four windows which can be operated individually. The three passenger-side windows can be operated by the driver with higher priority. In addition to the basic function that controls the position of

Fig. 2. The complete scope of the power window Simulink® model.

Fig. 3. The front-passenger-side power window Simulink® mode.

the window with push-buttons, the power window model we used in this paper contains the following functions [6]:

1) *Autoup & Autodown*: The autoup and autodown functions can be triggered by pressing the push-down button then quickly release it within 0.5 second. Once initiated, the window will raise up or lower down automatically until it reaches the end of the window frame, or until an object is detected between the rising glass and the window frame.

2) *Object (Pinch Force) Detection*: If an obstacle is present between the top of the window frame and the glass when the window is closing (Fig. 1), the motor will automatically revert. The window glass will lower down by a certain distance to prevent potential injuries to hands, fingers or even accidents to children.

3) *End of the Range Detection*: The end of the range detection mechanism guarantees that when the window is fully closed, no force higher than a certain value (e.g. 100 N) is present between the glass and the window frame.

B. Power Window Model

Fig. 2 gives the overview of the complete power window Simulink® model. The model consists of the driver-side window (*DRV*), front-passenger-side window (*PSG_Front*), back-left-passenger-side window (*PSG_BackL*) and back-right-passenger-side (*PSG_BackR*). The outputs of the driver-side debounce module are used as inputs by the control exclusion module of the passenger-side windows.

Since the 3 passenger-side power windows are identical, we will use the front-passenger-side window for demonstration in

this paper (Fig. 3). Compared with the passenger-side power window, the driver-side power window excludes the control exclusion module and contains 6 more debounce circuits in the debounce module.

1) *Debounce*: When a push-button is pressed, it rebounds a bit before settling. This mechanical reaction often generates spurious inputs, for example, it may be read as multiple quick presses in a short period by the program. Without debouncing, pressing the button may cause unpredictable results. The debounce module (*Debounce_PSG_Front*) is implemented to filter out these faulty inputs. The state chart implements the debouncing function. It is connected to a pulse signal with a period of 10 ms. The state chart is triggered on the rising edge, any signal has a period of less than 30 ms will not be forwarded to the next module. The debounce module is trivial because it guarantees the input only changes when push-button is definitely pressed or released.

2) *Control Exclusion*: The control exclusion module (*ControlEx_PSG_Front*) is only included in the passenger-side windows. By using basic logic gates, the control exclusion module assigns higher priority to the driver when control inputs come from driver and passenger simultaneously.

3) *Power Window Control*: As the last module of the power window, the power window control module (*PW_PSG_Front*) takes the control signals from the control exclusion module (*up*, *down*) and the environment signals (*theendof detection*, *currentsense*) to control the motor and the safety mechanism. Similar to the debounce module, the state chart of the power window control is connected to a pulse signal with a period of 50 ms and triggered on both the rising and falling edges. The input signals *theendof detection* and *currentsense* are the feedback from the plant, which is a part of the environment.

C. Benchmark Description

The benchmark was initially generated by Matlab® Embedded Coder® from the Simulink® model described in Subsection II-B. Matlab® Embedded Coder® is a widely available tool that “generates readable, compact, and fast C and C++ code for use on embedded processors, on-target rapid prototyping boards, and microprocessors used in mass production [7].” In our case, the benchmark was generated in C code targeting Atmel AVR (8-bit) micro processor with a discrete fixed-step solver at a fixed-step size of 5 ms.

Modifications were made to convert the generated code to TACLeBench standards [4]. TABLE I is a brief explanation about the source files included in the powerwindow benchmark. The input set included in *powerwindow_inputs* can be replaced by user-defined inputs.

The input set was generated exclusively from the model. One of the reasons is to guarantee the power window model can be deployed independently from the inputs, in order to increase the flexibility of the benchmark. For example, when a micro-processor has limited data memory, the input set can be used as external inputs from communication interface such as serial port. Additionally, it is almost effortless to connect the

Fig. 4. The front-passenger-side power window inputs.

Fig. 5. The general case study implementation.

model with new input set or real-time inputs, in such way users can create new benchmarks adjusted to their needs. By using real-time inputs, the benchmark accomplishes the functionality of a real-world power window application.

Fig. 4 illustrates the input set of the front-passenger-side window. The input set was created using Simulink[®] Design VerifierTM with MC/DC (modified condition/decision coverage) coverage objectives. MC/DC is a code coverage criterion to assess whether adequate testings have been performed for the most critical (Level A) software [8]. It is used in avionics software development guidance DO-178B [9] and DO-178C [10] and highly recommended for automotive. The generated input set achieved the highest possible MC/DC coverage rate for the power window model. Thus, it may have a higher possibility to hit the WCET in the timing analysis.

The inputs were sampled every 5 ms, which resulted in 977 groups of inputs in the input set. The cases covered by the input set include window position control from the passenger, simultaneous controls from both the driver and the passenger, pinch detection, and end of the range detection. Note that this input set is used for testing the front-passenger-side window as an individual benchmark, when use the complete power window benchmark, the *Driver_Up* and *Driver_Down* will be connected to the driver-side power window.

III. CASE STUDY

In the case studies, the powerwindow benchmark was deployed on a DVK90CAN1 development board which holds an 8-bit Atmel AT90CAN128 single-core microprocessor. Technically speaking, every component of the benchmark can be considered as a periodic task. When a task is triggered, it reads

TABLE I
THE NAME AND FUNCTION OF THE SOURCE FILES INCLUDED IN POWERWINDOW BENCHMARK.

Name	Function
debounce	the Debounce module
controlexclusion	the Control Exclusion module
powerwindow_control	the Power Window Control module
PW_Control_DRV	the driver-side power window
PW_Control_PSG_Front	the front-passenger-side power window
PW_Control_PSG_BackL	the back-left-passenger-side power window
PW_Control_PSG_BackR	the back-right-passenger-side power window
const_params	the lookup table for stateflow chart in Power Window Control module
powerwindow	the final implementation of the complete power window
inputs	the input set of the power window, can be replaced by user-defined inputs

Note: apart from powerwindow, the complete name of all the other files include prefix *powerwindow_*

Fig. 6. The implementation of case study 1.

one group of inputs from the input set. Ideally, the benchmark should be connected to a 16-bit timer which triggers one interrupt every 5 ms. Once the interrupt is triggered, the function of the power window will be initiated. However, in our case, due to the limited data memory, the input set was sent through serial port from Matlab[®] (Fig. 5). The read and write communication was not included in the main function, so that the measured execution time would not be affected. The complete input set was used as the test case. One complete execution of the test case is considered as one test. The n^{th} execution is represented by *test-n*. In total, the test case was executed 50 times consecutively in every case study.

During the case study we performed timing analysis on the benchmark in three cases:

- 1) case study 1: the driver-side and passenger-side windows execute in isolation as individual benchmarks.
- 2) case study 2: four windows execute on one single-core processor in sequential order with interactions.
- 3) case study 3: four windows execute in semi-parallel with interactions under an operating system.

IV. RESULTS

Fig. 7. The results (partially) of execution time of case study 1 of PSG_Front.

1) *Case Study 1*: In case study 1 the driver-side window and passenger-side windows were used as individual benchmarks (Fig. 6). The test case was executed consecutively without resetting the board or reinitializing the function. The timing analysis results are demonstrated in TABLE II. The BCET (Best-Case Execution Time) of the driver-side window was reached at 1189 *cyc* (148.625 μ s), the WCET (Worst-Case Execution Time) was at 1986 *cyc* (248.25 μ s). The timing behaviours of the three passenger-side windows were identical. The BCET and the WCET were reached at 632 *cyc* (79 μ s) and 977 *cyc* (122.125 μ s) respectively.

Fig. 7:(a) is the results of the second execution (*test-2*) of the complete test case. Fig. 7:(b) is the results of the third execution (*test-3*) of the complete test case. The BCETs for both tests were at 632 *cyc*, whilst the WCETs were at 949 in *test-2* and 960 in *test-3*. Another observation was that the BCET occurred much more frequently than the WCET. Although the case study was carried on a very simple single-core processor without any non-deterministic components such as pipeline and cache, the executions of *test-2* and *test-3* did not follow the same pattern. By subtracting the results of *test-3* from *test-2*, clear differences can be seen in Fig. 7:(c).

Based on the results of case study 1 we performed one additional experiment, in which we reinitialized the power window before the execution of the complete test case. After the reinitialization, all the variables of the functions are guaranteed to be identical in the beginning of the test. As can be seen in Fig. 7:(d), the benchmark showed exactly same timing behaviours across the different tests. Note that the reinitialization was achieved without resetting the processor.

2) *Case Study 2*: To test whether the powerwindow benchmark is applicable when there are interactions between different components, case study 2 was conducted. In case study 2, four power windows were chained in sequential order before being deployed on the 8-bit Atmel AT90CAN128 single-core microprocessor (Fig. 8). The outputs of driver-side window were used as inputs for passenger-side windows. The results are presented in TABLE III. The BCET and the WCET of the driver-side window had same values as in case study 1.

TABLE II
THE RESULTS OF CASE 1.

Task	BCET (<i>cyc</i>)	BCET (μ s)	WCET (<i>cyc</i>)	WCET (μ s)
DRV	1189	148.625	1986	248.25
PSG_Front	632	79	977	122.125
PSG_BackL	632	79	977	122.125
PSG_BackR	632	79	977	122.125

DRV: driver-side window; PSG: passenger-side window; BackL: back-left; BackR: back-right; BCET: best-case execution time; WCET: worst-case execution time; *cyc*: system clock cycle

Fig. 8. The implementation of case study 2.

TABLE III
THE RESULTS OF CASE 2.

Task	BCET (<i>cyc</i>)	BCET (μ s)	WCET (<i>cyc</i>)	WCET (μ s)
DRV	1189	148.625	1986	248.25
PSG_Front	632	79	991	123.875
PSG_BackL	632	79	991	123.875
PSG_BackR	632	79	991	123.875

DRV: driver-side window; PSG: passenger-side window; BackL: back-left; BackR: back-right; BCET: best-case execution time; WCET: worst-case execution time; *cyc*: system clock cycle

Fig. 9. The results (partially) of case study 2.

The WCETs of the passenger-side windows were at 991 *cyc* (123.875 μ s), which were slightly higher compared with the 977 *cyc* observed in case study 1.

In Fig. 9:(a), we compared the results of driver-side window in case study 2 with the results of driver-side window in case study 1. As can be seen, for the driver-side window, *test-2* in case study 2 and *test-3* in case study 1 had identical timing behaviours. However, this observation did not hold for the passenger-side windows. Differences could be observed between the *test-2* in case study 2 and the *test-3* in case study 1.

Fig. 10. The front-passenger-side power window inputs with debounced driver-side inputs.

TABLE IV
THE RESULTS OF ADDITIONAL EXPERIMENT IN CASE STUDY 2.

Task	BCET (<i>cyc</i>)	BCET (μs)	WCET (<i>cyc</i>)	WCET (μs)
PSG_Front	632	79	991	123.875
PSG_BackL	632	79	991	123.875
PSG_BackR	632	79	991	123.875

PSG: passenger-side window; BackL: back-left; BackR: back-right; BCET: best-case execution time; WCET: worst-case execution time; *cyc*: system clock cycle

Fig. 11. The implementation of case study 3.

study 1 of the passenger-side window (Fig. 9:(b)).

In order to inspect the reasons behind the observed changes in passenger-side windows in case study 2. We performed another experiment with the isolated passenger-side window benchmark (described in case study 1). In this experiment, we created a new input set, in which the input signals from the driver were debounced before being used by the passenger-side power window (Fig. 10). It can be seen in TABLE IV and Fig. 9:(c), the results of this experiment were identical compared with the results of passenger-side windows in case study 2.

3) *Case Study 3*: Case study 3 targets the possibility of using the powerwindow benchmark to perform timing analysis on parallel tasks under an operating system. The OSEK [11] operating system was used to schedule the four tasks of the

TABLE V
THE RESULTS OF CASE 3.

Task	BCET (<i>cyc</i>)	BCET (μs)	WCET (<i>cyc</i>)	WCET (μs)
DRV	1246	155.75	3523	440.375
PSG_Front	743	92.875	2762	345.25
PSG_BackL	743	92.875	2754	344.25
PSG_BackR	743	92.875	2754	344.25

DRV: driver-side window; PSG: passenger-side window; BackL: back-left; BackR: back-right; BCET: best-case execution time; WCET: worst-case execution time; *cyc*: system clock cycle

Fig. 12. The histogram of execution times (from *test-1* to *test-50*) of PSG_Front in 3 case studies.

four power windows (Fig. 11). The tasks were defined as non-preemptive with a period of 5 ms. However, due to the delay introduced by serial communication, the tasks will pile up in the task queue. If one task was assigned a higher priority, the other tasks will be constantly pre-empted. Therefore, because of the limitations of the experiment setup, all tasks needed to have the same priority to guarantee the correctness of the inputs received from the serial port. This is the reason why case study 3 is “semi-parallel”.

As can be seen in TABLE V, the BCETs increased by 111 *cyc* in case study 3. The WCET of driver-side window almost doubled compared with the other case studies. The increases of the WCETs of the passenger-side windows were even more notable. The results of the 3 passenger-side windows were no longer identical. The WCET of the front-passenger-side window was slightly higher than the rest. Fig. 12 illustrates the execution times of the front-passenger-side window in histograms. It can be observed that in case study 1 and 2, most of the execution times were around 650 *cyc* and 750 *cyc*. The WCETs occurred rarely but still close to the average execution time. However, in case study 3, the distribution changed drastically. Most of the execution times were lower than 1000 *cyc*. Only several execution times were located around 2500 *cyc*, which was much higher than the average execution time.

V. DISCUSSION

Case study 1 has shown that the variations between different executions without reset were due to the fact that the

benchmark is sensitive to the initial state of the power window. The execution time is not only dependent on the inputs but also on the state (the variable values) of the power window function. The changes in the execution pattern is because the finishing state of one test will become the initial state of the following test. This can be further proved by the additional experiment with reinitialisation, in which the power window showed exactly the same behaviours in different tests.

In case study 2, after connecting the passenger-side windows with the driver-side window, differences in the timing behaviours occurred in the passenger-side windows. The additional experiment showed that the differences were due to the debounce function in the driver-side window, from which we concluded that the test case generated by Simulink® Design VerifierTM was not sufficient to cover the worse case scenario. Special cautions should be given when using Simulink® to generate test case for critical systems. After further investigation we noticed the test case generated by Simulink® is based on the step-size of the solver (5 ms in our case). However, when there is asynchronization between the solver and functional components (state charts in our case), the test case may not be taken as inputs by the functional components. Due to this, even though the test case covers all possible inputs and correlated outputs under MC/DC coverage criterion, it is still not sufficient.

The increase of the execution time in case study 3 may be due to the OSEK operating system. The increases in the WCETs were much higher compared with the increases in BCETs. To fully understand the reasons behind this dramatic increases, further investigations toward the OSEK operating system is needed.

Although the test case was not complete for detecting the absolute WCET for the passenger-side window in case study 1, we still proved that the powerwindow benchmark is a sophisticated to be used in different timing analysis techniques according to the requirements. Because the flexibility provided by the benchmark, more complicated and new benchmarks can be created with limited efforts.

VI. CONCLUSION

In this paper we presented a multi-component TacleBench benchmark. By adjusting the components (four windows), the powerwindow benchmark can be used for evaluating timing analysis techniques and tools for single-core, parallel, distributed or even multi-core systems. This flexibility provided by the components is the main advantage of this benchmark. In addition, the powerwindow benchmark showed the possibility to create benchmarks by Matlab® Embedded Coder®, with which only minimal efforts are needed to generate new benchmarks especially for specific hardware platforms.

VII. FUTURE WORK

In the next step, we will overcome the limitation of the data memory by reading the input signals from input pins instead of using serial communication. In such way we will be able to perform case study 3 using preemptible tasks with different

priorities. Timing analysis on multi-core processors with the powerwindow benchmark will thereafter be studied. For the future timing analysis, the initial state will be considered. Another topic which needs attention is how to improve the test case generated by Simulink® Design VerifierTM to cover more cases.

ACKNOWLEDGEMENT

This research was partially supported by Flanders Make vzw, the strategic research centre for the manufacturing industry.

REFERENCES

- [1] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric os for multi-core embedded systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2016, pp. 1–11.
- [2] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, "Multi-source and multicore automotive ecus-os protection mechanisms and scheduling," in *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 3734–3741.
- [3] R. Wilhelm, S. Altmeyer, C. Burguière, D. Grund, J. Herter, J. Reineke, B. Wachter, and S. Wilhelm, "Static timing analysis for hard real-time systems," in *Verification, Model Checking, and Abstract Interpretation*. Springer, 2010, pp. 3–22.
- [4] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. Bo Srensen, P. Wagemann, and S. Wegener, "Taclebench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis*, 2016.
- [5] S. M. Prabhu and P. J. Mosterman, "Model-based design of a power window system: Modeling, simulation and validation," in *Proceedings of IMAC-XXII: A Conference on Structural Dynamics, Society for Experimental Mechanics, Inc., Dearborn, MI*, 2004.
- [6] K. Vanherpen, J. Denil, H. Vangheluwe, and P. De Meulenaere, "Model transformations for round-trip engineering in control deployment co-design," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International, 2015, pp. 55–62.
- [7] I. The MathWorks. (2016) Embedded coder® getting started guide.
- [8] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierison, "A practical tutorial on modified condition/decision coverage," 2001.
- [9] L. A. Johnson, "Do-178b, software considerations in airborne systems and equipment certification," *Crosstalk, October*, 1998.
- [10] L. Rierison, *Developing safety-critical software: a practical guide for aviation software and DO-178C compliance*. CRC Press, 2013.
- [11] A. Zahir and P. Palmieri, "Osek/vdx-operating systems for automotive applications," in *OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), IEE Seminar*. IET, 1998, pp. 4–1.