IET Cyber-Physical Systems: Theory & Applications

Special Section: Safety Critical Cyber-Physical Systems

Execution time analysis and optimisation techniques in the model-based development of a flight control software

ISSN 2398-3396
Received on 30th December 2016
Revised 2nd May 2017
Accepted on 31st May 2017
E-First on 5th July 2017
doi: 10.1049/iet-cps.2016.0046
www.ietdl.org

Kajetan Nürnberger[1] ✉, Markus Hochstrasser[1], Florian Holzapfel[1]
[1]Institute of Flight System Dynamics, Technical University Munich, Munich, Germany
✉ *E-mail: kajetan.nuernberger@tum.de*

Abstract: This case study analyses the possibilities to improve the execution time of model-based developed software by applying optimisations during code generation and compilation. The present case study is performed on flight control software, for which safety aspects are accounted throughout the development. Therefore, a formally verified compiler is used for the optimisation during the compilation. The optimisation is evaluated by execution time measurements on the target and a static worst-case execution time analysis. Based on the results, recommendations for certain model patterns are given, which impact the worst-case execution time analysis.

1 Introduction

The functionalities of flight control and flight management systems grew rapidly in the last decades. With the trend to unmanned aerial vehicles (UAVs), where the complete system has to be operated automatically, the functional range of flight control and flight management expanded further. In addition to these extended functionalities, the first certification standards for remotely piloted aircraft demand similar or even higher design assurance levels (DALs) for the development processes in comparison to the manned aircraft in some cases. For example, for the development of an aircraft with a maximum take-off weight of two tonnes and multiple reciprocating engines, a catastrophic failure condition demands DAL C [1] for the manned case and DAL B [2] for the unmanned case. To manage the higher demands regarding safety and functionality, model-based approaches are often used. The benefit of these approaches is that the functions can be tested in simulations at an early stage of the development for which the detection of failures and incompatibilities is easier. The work in [3] shows that errors detected during this early stage lead to less costs and effort for the whole system development. In model-based development, process steps like code generation can be automated, which leads to fewer costs. A cost-effective development is necessary so that UAVs are competitive to manned aircraft in the civil market [4].

This paper presents a case study for the integration of a model-based developed flight control algorithms into a cyber physical system. This universal flight control system, developed at the Institute of Flight System Dynamics at TU Munich, is implemented based on the certification requirements and processes and is easily adaptable to different aircraft. At the moment, the system is used in three different aircraft:

- on a small UAV with a maximum take-off weight of 150 kg as primary flight control computer (FCC) [5],
- as digital auto pilot interfaced by an experimental flight management system on a commuter CS 23 aircraft of the German Aerospace Centre [6],
- on the research demonstrator aircraft of the institute, a twin engine CS 23 aircraft, which can be used for demonstrating various flight control algorithms by providing a UAV-like environment with the security of a backup pilot on board of the aircraft. In this paper, this software variant was used to generate the results [7].

The flight control system consists of sensors that gather information about the aircraft's state, a command and control interface for the pilot or flight operator, an FCC that computes the flight control laws, and actuators that execute the calculated commands. For the systems above, the same FCC hardware is utilised. The other components are different on each platform.

This paper focuses on the integration of the algorithms into the FCC an embedded target, which is used to execute the flight control laws in the real environment afterwards. The algorithms are developed using Simulink and Stateflow [8] and then are transferred into source code using MathWorks Embedded Coder (EC) [9]. As the system shall be developed in consideration of the certification rules, the approach presented in this paper is motivated by DO-178 [10] for the overall software development and DO-331 [11] for the model-based part. These guidelines are relevant for the development of the UAV, because both the military certification requirements like STANAG 4671 [2] as well as the proposal for civil rules like AMC RPAS.1309 [12] propose these guidelines as acceptable means of compliance. The use of model-based development techniques is not completely new for the development of flight control systems. There are approaches where the model developed by the control engineers is transferred manually into a version which is target compatible [13] and also approaches where the model is transferred in an automated process exist [14]. One of the novelties of the approach in this paper is that the model developed by the system and control engineers is used for auto code generation and target integration without further modification. The benefit is a leaner process. The challenge of such an approach is that the generated code is required to be compatible with the target and is verifiable on the target from a software point of view. The compatibility with the target is especially necessary regarding the memory and timing requirements. The timing aspect is of great importance for flight control software, because assumptions are made for the delay of the whole control loop during the controller development. These timing constraints are necessary to prove stability of the control algorithm. During the software integration process, each component must show that the assumptions for the timing are met and, therefore, the delay of the whole control loop does not exceed the expected boundary. Otherwise, the stability of the control algorithm cannot be guaranteed. The model developed by control experts does not necessarily consider aspects that a software engineer would have in mind when producing code for a real-time system. The paper especially considers the optimisation possibilities of timing aspects in model-based developed software. It compares and evaluates optimisations during auto code generation and during compilation.

This is an open access article published by the IET under the Creative Commons Attribution License
(http://creativecommons.org/licenses/by/3.0/)

The used optimisations can be applied without further effort during the development process. The applied methods also do not have an adverse influence on the verification methods. Therefore, applying optimisations during the development process might compensate that no extra model was generated that explicitly addresses an efficient implementation for an embedded target. A side effect of optimisation, which is not further discussed in this paper, is the reduction of the memory consumption of the target software. For all variants, the execution time is analysed by measurement on the real target and worst-case analysis. Based on these results, it is assessed whether the approach of using the model utilised for controller design for auto code generation is a feasible way for certification.

Therefore, this paper considers more facets than [15], where the aspects of the model-based development are not considered, the timing analysis is only performed considering a WCET analysis and only the model-based code parts are compiled using an optimising compiler.

This paper is structured as follows: In Section 2, the FCC hardware is described, followed by a section where the software structure and functionalities are explained. In Section 4, different variants of the software, which are produced in this case study, are presented. The methods applied to analyse the execution timing are shown in Section 5, whereas the result follows in Section 6. Finally, recommendations based on the results are given in Section 7.

2 Hardware overview

The requirements for the development of the hardware are that the platform provides a high computational performance as well as various digital external interfaces. The reason for a high number of interfaces is that the system is used in an environment where high availability is needed and, therefore, it might be necessary to share the information with more redundant sources or sinks. In contrast to other approaches where a whole platform including custom I/O modules is developed [16], it shall be possible to integrate the FCC in existing aircraft. Often they already provide some sensors. Hence, the embedded system is not equipped with one standardised network interface but with many different interface types common in avionics systems. To fulfil these requirements, a multi CPU approach was selected. Fig. 1 shows an overview of the FCC consisting of two processors for the external interfaces and a main CPU for computation of the flight control algorithms.

To provide the external interfaces, two ARM Cortex-M3 processors are used. In other projects, a field programmable gate array is often used for this purpose [17, 18]. The reason of utilising microcontrollers here was to analyse the usage of such devices for I/O tasks and to show the effort which is necessary for such an development in the context of safety critical airborne software that is regulated by DO-178 [10]. This is of interest, because, since the DO-254 [19] came in effect, the shifting of functionalities from software to hardware is not that attractive anymore. The two Cortex-M3 I/O processors are connected to external interfaces. The data from the external interfaces is accumulated in the random access memory of the I/O processor and is transferred to the main CPU on request. Two point-to-point dual duplex Ethernet interfaces are used to exchange the data. All messages that have to be sent on external interfaces are also transferred to the I/O CPUs using Ethernet. Either the messages are sent immediately, or they are buffered in an FIFO until the necessary resource is available.

To achieve high computational power for the computation of the flight control laws, an MPC8349 was selected. This CPU is based on an e300 core with 32 KB of instruction cache and 32 KB of data cache. Both caches consist of 128 sets and each set consists of eight blocks. The operating frequency of the core is 533 MHz. Attached to the CPU are 16 MB of Flash and 256 MB of DDR SDRAM.

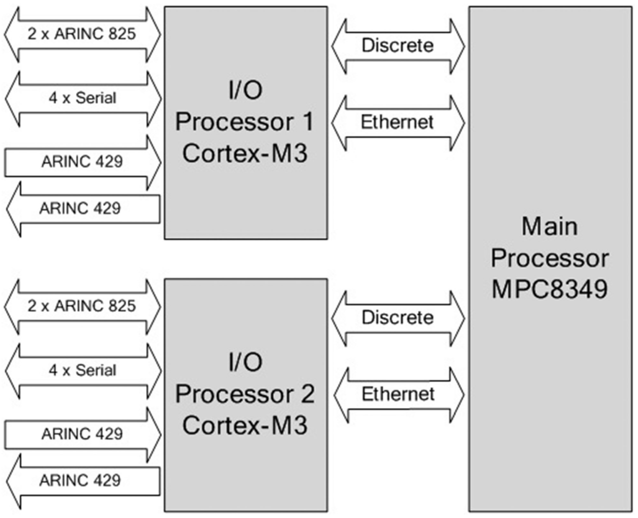

Fig. 1 *FCC hardware overview*

3 Software structure and functions

This section gives an overview of the software structure, the functionalities implemented in the Simulink model, the interface to the manually developed software parts, and the processes followed during generation of the model.

3.1 Software framework

The complete software on the main CPU is implemented as a single-rate system and, hence, no operating system is applied. To ensure the cyclic execution of the system, a static scheduler, which uses the periodic interval timer (PIT) of the CPU, is implemented. Each time cycle is split into a foreground task and a background task. An overview of the different tasks is given in Fig. 2.

At first the main application is executed in the foreground task. Afterwards, hardware check functions, which are non-time critical, are executed in the background task. The approach of a single-rate system decreases the complexity of the worst-cases analysis of the system. As the impact on the cache content due to task switches is low, the content of the caches can be analysed with a higher precession than in a multi-rate system. To trigger the beginning of a cycle, the reset of the PIT is checked during a busy wait. The reset of the PIT does not trigger an interrupt service routine, because the complete software on the main processor is implemented without interrupts. This is due to the fact that interrupts may have an impact on the execution state of the processor and, therefore, they might have a significant impact on the timing of the CPU [20]. The first task executed in a cycle is the

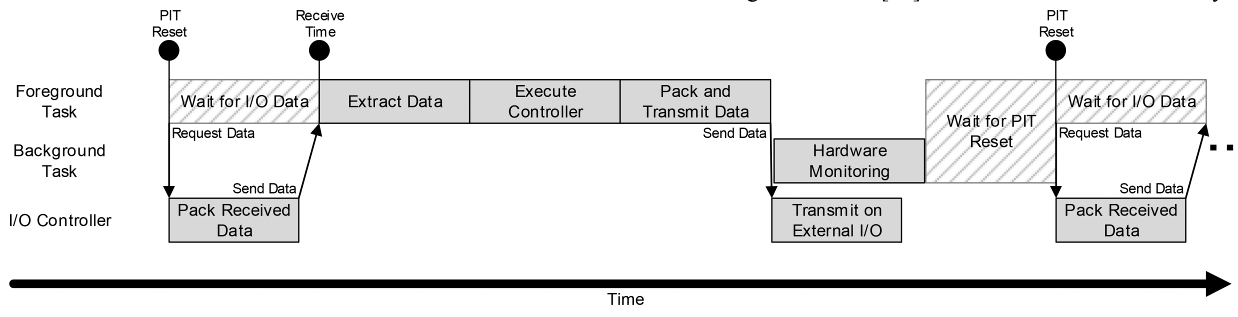

Fig. 2 *FCC task overview*

This is an open access article published by the IET under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/)

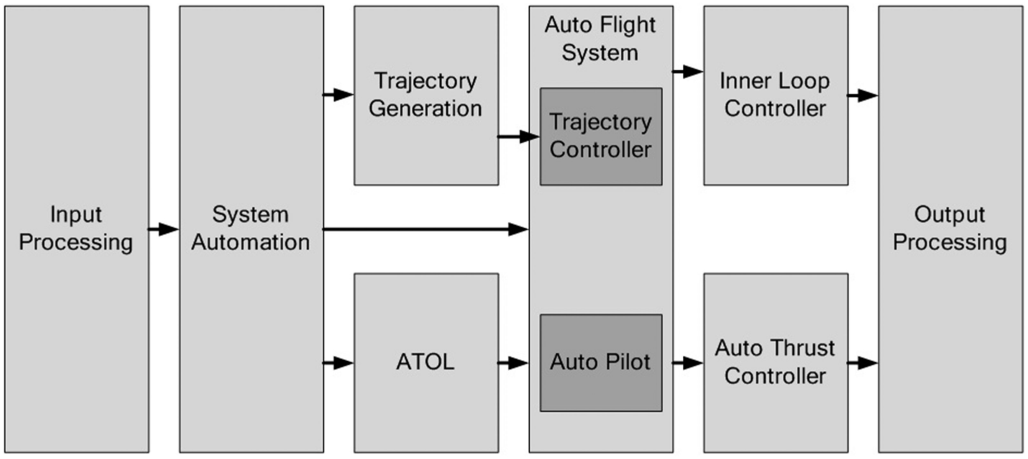

Fig. 3 *Flight control system functional overview*

request of the data received from the I/O processors during the last cycle. A discrete I/O is toggled, which triggers transmission of data via the Ethernet connections. The data is sent via a given custom Ethernet protocol, which uses only OSI layer one and two. The main controller executes a busy wait for fixed time until all Ethernet packages are received. The wait time is based on a worst-case consideration of the I/O controller. This approach will lead to a slight loss in performance, as the overall worst-case for the I/O controller might not occur in a certain system. On the other hand, the verification of the I/O controller is independent from a specific use case and can be reused for different systems. Therefore, the development and verification effort can be reduced significantly over the different projects. After the reception, all data are extracted from the Ethernet protocol. Then the data is forwarded to the flight control functions by assigning the content of all messages to the corresponding interface variable of the Simulink model. The interface is defined in interface control documents (ICDs) during the design phase. ICDs are parsed by a custom code generator, which then generates the C functions to extract the individual values and scales them before they are assigned to the interface variables. This automatic code generation enables fast adaption to different system architectures, in which the FCC is integrated. The data provided to the Simulink model is mainly the sensor data and the controller commands. To enable the functional input monitoring to check when messages are updated, a counter is created for each input message indicating, in which period the message was received. After all input data has been assigned, flight control algorithms are called. The developed controllers are all included in a single top model, which consists of several referenced sub-models. During code generation, this top model including all references is transferred to C source code. Therefore, only one-step function of the integration model needs to be called in the target code to execute the cyclic part of the model.

Subsequent to the calculation of the Simulink functionalities, the data marked to be sent are scaled and packed into the corresponding messages. This is again performed by source code, automatically generated with the ICDs as input. As last step in the foreground task, these messages are packed into the Ethernet protocol and are transmitted to the I/O processors.

3.2 Function overview

The developed system implements flight control functions necessary for a completely automatic operation of an aircraft. An overview of the system is given in Fig. 3. Modularity of the controller allows flexible adaption of the software for multiple applications. Functionality can easily be added or removed. For example, the system for the German Aerospace Centre does not include the trajectory mode because this part is implemented in the external flight management system [6].

The input processing is the first computation step performed in each software frame. In this block, the sensor and command data are consolidated and monitored. The valid signals are mapped to internal data busses of the integration model. As the monitoring and validation of signals is dependent on the performance of the sensor and the number of available sensors, this block is platform specific and is replaced for each project. Based on the command inputs, the system automation determines, which controller modes must be activated. In case the commands are not available, for example during link loss, emergency procedures are also activated by the system automation. A detailed description of the system automation can be found in [21]. Based on waypoints, the trajectory generation module calculates a flight path that can be achieved by the applicable aircraft, and calculates the deviations between the actual aircraft position and the desired trajectory [22]. As the computations of the generation module are based on geometric considerations, multiple trigonometric functions are called within this submodule. For automatic take-off and landing (ATOL), a special mode selection logic is introduced. Owing to the low height above ground level, special fault detection and diagnostics are active during these flight phases. The result of the diagnostics is then used to automatically trigger the necessary flight controllers to execute the desired manoeuvre or to execute a safe abort or go-around [23]. The auto flight module consists of two parts. One part provides classical autopilot function like heading hold, altitude hold or speed control [7], and the other one is a special controller for the trajectory mode. This controller reduces the deviations of the actual aircraft position to the calculated trajectory to zero [24]. Both controllers are based on the concept of non-linear dynamic inversion [25, 26] and comprise scheduled limits and gains to obtain consistent performance over the whole flight envelope. The auto flight controllers provide load factor commands to the inner loop and auto thrust controller. These controllers are specific for each platform. Based on the load factor in direction of the flight path, the auto thrust controller commands the actual trust setting. The inner loop controller transforms the load factors normal to the flight path into deflections of the control surfaces. Here, the controller is designed as linear multiple-input-multiple-output controller, where limits and gains are scheduled over airspeed and static pressure to ensure performance and margins for the permissible flight envelope. After the control surfaces deflections are calculated, the output processing performs the transformation of the model internal signals to the signals specific for the messages of the applicable architecture. For all scheduled gains and limits, lookup tables are used which use a binary search algorithm.

3.3 Model-based developed process

In difference to other approaches, e.g. [14], in this case study the same model is used throughout the whole development lifecycle. So the model, which is used for the controller design, is also transformed automatically to C source code and is afterwards deployed to the embedded target. If only one model that is not transformed into another model or modelling language is used, no errors can be introduced in those transformation steps. Simulink was selected, because the tool is quite familiar to most control engineers, and MathWorks toolboxes offer various possibilities for

This is an open access article published by the IET under the Creative Commons Attribution License
(http://creativecommons.org/licenses/by/3.0/)

Table 1 WCET analysis and measurement results

	Non-optimising compiler (GCC)	Optimising compiler (CompCert)
no EC optimisation	case G0	case C0
EC optimisation	case G1	case C1

Table 2 Selected optimisation settings

Option	No EC optimisation case *0	EC optimisation case *1
conditional input branch execution	Off	On
signal storage reuse	On	On
enable local block outputs	On	On
eliminate superfluous local variables (expression folding)	Off	On
reuse local block outputs	Off	On

simulation and development of control algorithms. The allowed blocks were limited, and a special configuration setting was introduced to take the benefit of automatic code generation that can be verified by tools. The main reason for this limitation is the use of the Simulink Code Inspector [27], a tool that can show compliance between the model and the generated source code. The impact of the applied limitation was acceptable for the controller design engineers. A more detailed view on the model-based development process is given in [28].

4 Software integration

In this section, the software integration incorporating the auto code generation and compilation is described. All considered optimisation techniques are applied in this process step. As shown in Table 1, four variants of the software are considered.

4.1 Auto code generation

To generate C source code from the model, the EC from MathWorks is used. As stated in Section 3.3, the code generation shall be compatible to the Simulink Code Inspector. This limits the number of allowable coder optimisation settings, but still some settings that apply execution time optimisations can be used without restrictions. The optimisation methods applied here are available in EC. The impact of the most important remaining settings shall be analysed. Therefore, two variants of the auto code with different optimisation settings are examined. These are listed in Table 2.

The non-optimising setting of Table 2 still enables the 'signal storage reuse' and 'enable local block outputs'. If local block outputs are not enabled, all local signals are written into a global structure by EC. This leads to a huge amount of unaligned accesses, which impede WCET analysis as discussed in Section 7. With both options selected, the local signals are kept in the local scope of the function as separate local variables. The signal storage reuse is selected because the enable local block outputs is dependent on this option.

The meaning of the options selected for the optimising version is described as follows:

- *Conditional input branch execution*: Simulink executes only blocks that compute the control input and data input that the control input selects. This optimisation improves execution speed [9].
- *Eliminate superfluous local variables (expression folding)*: more blocks are collapsed to one single statement. The efficiency of the code is improved [9].
- *Reuse local block outputs*: Local variables will be reused [9]. This mainly reduces the necessary memory consumption of the stack. With a lower memory consumption, the cache misses might decrease and, therefore, this feature has potentially impact on the execution timing.

4.2 Compilation

For compilation, GNU GCC or CompCert is used. GCC is used for the non-optimised cases (Cases G*) and CompCert for the optimised cases (Cases C*). The manual code and the auto-generated code are compiled and linked in one process step. Therefore, one executable with the same compiler settings for all parts is generated.

To ensure the confidence in correctness of the compilation in cases with GCC, optimisation is completely turned off (setting -O0). This configuration is also beneficial in case of a DAL A development, where the structural coverage has to be verified based on executable object code.

To accelerate the execution time while keeping confidence in the correctness of the compilation process, the optimising compiler CompCert [29] is evaluated in this paper. The benefit of CompCert is that a formal specification is available for this compiler and the optimisations are proved against this specification. Due to the formal verification, the compiler does not support all statements of the C language. It only supports the so-called Clight subset [30]. When Clight is compared with the subsets of C that are allowed for safety critical application as in [31], this restricted language set was no problem in this project. The complete auto-generated code of EC was compatible to the language set of the CompCert. To prove the correctness of the compilation, the compilation is performed using several intermediate languages. The optimisations performed by CompCert are basic methods, which are also implemented in other compilers. The following methods are applied [32]:

- instruction selection to take advantage of combined instructions
- constant propagation
- common subexpression elimination
- dead code elimination
- function inlining
- tail call elimination
- register allocation

5 WCET analysis

Execution times in this paper are assessed for the model-based developed part. To evaluate the benefit of the applied optimisation techniques, the execution time is measured during a test in a hardware in the loop (HIL) environment and a static analysis is performed to get a safe bound for the worst-case timing. DO-178 explicitly lists the worst-case execution timing as one of the objectives which need to be fulfilled for DAL A, B and C software [10]. A concrete method how the worst-case execution timing shall be determined is not proposed in DO-178, but further information is given in DO-248 in the answer of FAQ #73 [33]. It states that if measurements shall be used to gain the WCET, they should be supported by an analysis to show that the actual worst-case is measured. Further, it notes that the use of cache leads to a more complex calculation for the WCET. A more detailed view on using cache in airborne systems is given in [34]. It is explained that the execution time depends on the cache content. As the content of the cache is dependent on the cache management mechanism and the control flow of the software, these must be considered when the WCET shall be verified. Using a static analysis of the software, these effects can be considered and a safe bound for the WCET can be determined. As the system, which is presented in this paper, uses both data and instruction cache, the WCET is determined by a static analysis tool. The analysis is based on the executable object code and considers the control flow and cache management. Therefore, the WCET can be evaluated with reasonable effort by using static analysis.

This is an open access article published by the IET under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/)

5.1 Execution time measurement

For the HIL measurements, the execution time is calculated based on the internal timers of the main processor and is transmitted via a special message on the CAN bus. The test case used for the execution time measurement triggers all operational modes of the software. The system is operated under normal conditions. This means that no failures are introduced, e.g. all sensor signals are within their specified ranges, and no range violations occur. This leads to some limitations. Branches in the input monitoring of the software that handle invalid sensor data are not triggered and, therefore, the measurement might not represent the worst case. Due to conservative programming, the software has branches which cannot be triggered from outside of the software and, therefore, the execution time for these branches will be zero for the HIL measurements.

5.2 Static WCET analysis

To determine the WCET, in this case study, the tool aiT from Absint Software [35], which supports the e300 core, was used. Input to the analysis is the executable object code and potentially some user annotations containing, e.g. loop bounds, flow facts or certain register values. As the tool uses the executable object code and not the source code, effects of the compiler are considered. The first step in the analysis is the reconstruction of the control flow. This control flow is then annotated with information needed for the further steps. The next step is a value analysis, which uses abstract interpretation. Here ranges of registers are gained in order to demine infeasible path, addresses of indirect memory access or bounds of loops. After this process, the cache and pipeline analysis follows. Abstract interpretation is also used here. As last step follows the bound analysis in the context of aiT, this is called path analysis. This step can either be performed using an integer linear programming (ILP) method, a prediction file-based method or a combination of both. For further information regarding aiT and the theoretical background refer to [36, 37]. In this case study, the analysis was performed using the prediction file-based method for the path analysis. According to [37], the prediction file-based method generates the highest precision but requires an acyclic call graph. To get an acyclic call graph, the option ‘default unroll’ had to be set to a high value. The ‘default unroll’ controls how many contexts are generated for recursive calls or loops. The alternative to a prediction file-based path analysis is the ILP method. The precision for the prediction file-based method is higher because a global state graph is the basis here; in the ILP case, a local state graph is used [37]. For the case C0, the analysed WCET is about 37% lower with the prediction file-based path analysis compared with the ILP method. To determine the loop bounds, some annotations were necessary. For all variants of the software, annotations of some basic math functionalities like square root were necessary, and some so-called shared utils of the EC like the table lookup algorithms had to be annotated. With the annotation language of aiT, it is possible to do annotation based on register values at a given point. This feature was used to annotate the loop bounds of these functions based on their input values. It allows keeping the annotations universally valid. Annotations must not be adapted due to a functional change, e.g. if the number of breakpoints of a certain table is increased. For the executables generated with the GCC, no additional annotations were necessary to avoid unknown memory accesses in the analysis. However, to get the same result for the executables generated with the CompCert (Cases C*), an additional annotation was required. The range of an index of a lookup algorithm directly implemented in Simulink could not be automatically determined by the value analysis and, therefore, the value was annotated.

6 Results

In this section, the results of the measurements and analyses are presented and discussed. Due to the limitations of the measurement described in Section 5.1, the overestimation in Table 3 has to be handled with care.

Table 3 WCET analysis and measurement results

Case	Max measured execution time, ms	Analysed WCET, ms	Overestimation, %
G0	9.442	19.249	103.9
C0	3.569	6.522	82.7
G1	6.826	15.891	132.8
C1	2.897	5.965	105.9

Table 4 Assembler code

Description	C0/C1	G0	G1
execute multiplication	fmul 5,0,4	fmul 0,13,0	fmul 13,13,0
store result of multiplication		stfd 0,16(31)	
load base address of function parameter		lwz 9,28(31)	lwz 9,12(31)
load second operand for subtraction	lfd 2,16(4)	lfd 13,16(9)	lfd 0,16(9)
load first operand for subtraction		lfd 0,16(31)	
execute subtraction	fsub1,5,2	fsub 0,0,13	fsub 0,13,0

The overestimation is higher than the overestimation of the functional algorithms in [38]. In contrast to [38], a CPU with cache is analysed here. The cache introduces uncertainties in the analyses and, therefore, an increased uncertainty was expected. Especially, the access to large lookup tables has a big impact on the cache in the analysis. Due to the abstract interpretation of the value analysis, potentially all data points of the lookup table are accessed in the analysis. However, during the execution of the program in one cycle, for a two-dimensional look up table not more than four data points are actually accessed. For example, the algorithm has several two-dimensional lookup tables each containing 396 breakpoints. If a value shall be determined based on one of these lookup tables, maximal four values at breakpoints read in order to perform the interpolation. Therefore, in the actual execution, a read operation of these four values will have an impact to not more than four out of 128 sets of the cache. In the analysis potentially all data points are accessed. Therefore, a potential impact to 100 out of 128 cache sets has to be considered in the analysis.

In Section 5.1, it is explained that some branches in the input monitoring and branches due to conservative programming might not be executed. However, as aiT does not consider floating point arithmetic in the value analysis, these branches are also considered in the analysis. Considering these circumstances, the analysis result of the CompCert variant with 82.7% overestimation can be seen as a reasonable result. The analyses of [39] with a magnitude of 50% overestimation are not much lower for a software specifically built for embedded real-time systems.

In cases applying EC optimisations (G1 and C1), the overestimation in those cases is higher. The reason is related to the conditional input branch optimisation. Instructions that are always executed in the case of the non-optimised auto code are encapsulated in conditional statements in case of the optimised version. As shown in Section 7, the exclusiveness of different branches might not be analysable with aiT dependent on the way these conditions are modelled. Therefore, precision is lost in those cases. The WCET of variant G1 is improved by ∼17% in comparison to G0. This is almost twice as much as for the CompCert case, since some optimisations performed by the EC might also be applied in a similar form in the CompCert and, therefore, they will take no effect in case of a compilation with CompCert.

In Fig. 4, an example is presented where the EC optimisation has no impact on the executable generated with CompCert but the performance of the GCC executable G1 case is enhanced. The code generated without EC optimisation is listed in Fig. 5 and the code generated with EC optimisation is listed in Fig. 6. Table 4 shows the generated assembler code for all cases. Only the code from the multiplication instruction to the subtraction instruction is listed.

This is an open access article published by the IET under the Creative Commons Attribution License
(http://creativecommons.org/licenses/by/3.0/)

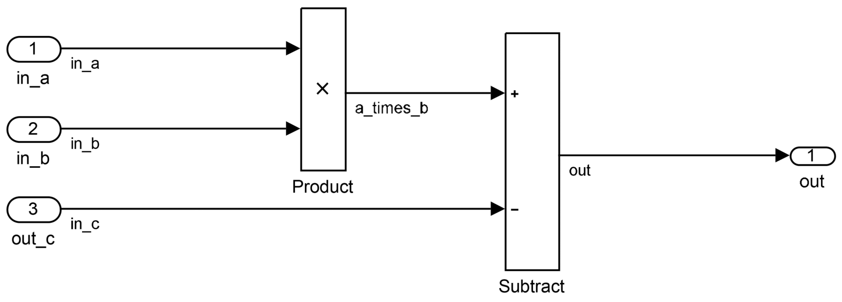

Fig. 4 *Example model to demonstrate the expression folding effect*

```
b_a_times_b = expression_folding_U->in_a * expression_folding_U->in_b;

b_Subtract = b_a_times_b - expression_folding_U->in_c;

expression_folding_Y->out = b_Subtract;
```

Fig. 5 *Code generated from the model shown in Fig. 4 without EC optimisation (comments removed)*

```
expression_folding_Y->out = (expression_folding_U->in_a *
    expression_folding_U->in_b) - expression_folding_U->in_c;
```

Fig. 6 *Code generated from the model shown in Fig. 4 with EC optimisation (comments removed)*

In case the source code is compiled with optimisation using CompCert, the generated assembler is identical for both cases. During the register allocation, it is recognised that the local variable b_a_times_b (correlates to the signal a_times_b in the Simulink model) is not further used and, therefore, the value is not written to a local variable at a stack location. In the case the source code is compiled without optimisation using GCC, the two generated assembler files differ. For the case G0, the result of the multiplication is stored at a stack location and reloaded from this location. For the G1 case where the local variable b_a_times_b is eliminated by the EC, the result of the multiplication is directly used for the subtraction, and the store and load operations are omitted in the assembler.

7 Correlation between model and WCET analysis

This section exemplarily presents modelling patterns that can be introduced on model level to improve WCET analysis execution and get more accurate WCET results. It shall give the reader an impression on how model patterns may affect abstract interpretation and which constructs have to be regarded more closely.

The analysed Simulink model intensively uses Simulink bus objects, and as the model shall be compliant to the DO-178/DO-331 workflow of MathWorks, it is not possible to make these busses virtual in most cases. After the auto code generation, non-virtual busses are represented as structures in the C code. In some cases, especially if busses are restructured, this leads to an extensive amount of non-functional data copies. In a real-time embedded system, such behaviour should be avoided. However, as shown before, this can be compensated by the high computational power of the target hardware.

For the copy of one bus into another, which is in the C source code an assignment of two structures, both compilers generated assembler code leading to unaligned memory access. The unaligned access occurs if more elements of a structure are copied at once using a bigger data type than the one of the elements. For example, when a structure that contains two elements each one byte in size is copied using a read and write of a word. Elements that are one byte in size do not have a certain alignment restriction, so the structure containing the elements might start at an odd address. If the whole structure is accessed using a word specific instruction, this leads to an unaligned access, because words should be aligned on even addresses. As the timing penalties for unaligned access are not documented for the used processor, the calculated WCET might not be safe. This can be avoided by specifying predefined alignment of the busses in Simulink, which is propagated by the code generator and the compiler. This non-functional change must be considered by the modeller. Identifying forced alignment depends on the structure of the busses and the structure layout algorithm of the compiler. Therefore, the modeller will have to consider specific information how the compiler works in order to introduce the alignment information for the necessary structures. Alternatively, the alignment of busses must be set to a default value of four, which will lead to an increased memory consumption.

Furthermore, a model pattern was detected that leads to an overestimation in the WCET analysis. In case certain functionalities need only to be calculated at a certain state, these functionalities are often encapsulated in enabled subsystems. The same construct is also used in case exclusive functionalities exist. The problem of this structure is that it is translated into source code with two independent ‘if’ statements. As the aiT WCET tool works with abstract interpretation [36], the value analysis determines ranges of variables. This leads to the circumstance that both ‘if’ statements are considered as potentially true and both encapsulated functionalities are on the WCET path. Depending on the extent of the subsystems, this might lead to a huge over estimation.

In Fig. 7, an example is presented. The corresponding code is shown in Fig. 8.

Dependent on the caller of the model, the value analysis will determine the value of the input flight_phase. If both subsystems are coverable, the result for the flight_phase in the value analysis will at least be [0 … 1] (as landing_phase corresponds to 0 and takeoff_phase to 1). This leads to the fact, that both if statements at the beginning of the enabled subsystems will evaluate to true in the WCET analysis.

To get a more precise result in the WCET analysis, multiple approaches exist. The first possible approach would be to work with annotation for the WCET tool [25]. The value of the variable flight_phase can be annotated and then for each flight phase an analysis can be executed. Afterwards the highest value of all analysis must be selected which is the WCET of the complete software. An alternative to this is a trace annotation. With such an annotation for each called subroutine, an extra context can be created within the calling function. Then the worst case would be selected during the path analysis. A flow constraint can also be introduced as annotation to the WCET analysis. This would restrict the possible path variant analysis methods. A pure prediction file-based analysis would not be possible.

For all those solutions, additional annotations would be necessary for the WCET analysis. Such annotations can either be done manually, which would lead to a huge effort, or automatically by analysing the Simulink model and writing a custom code

This is an open access article published by the IET under the Creative Commons Attribution License
(http://creativecommons.org/licenses/by/3.0/)

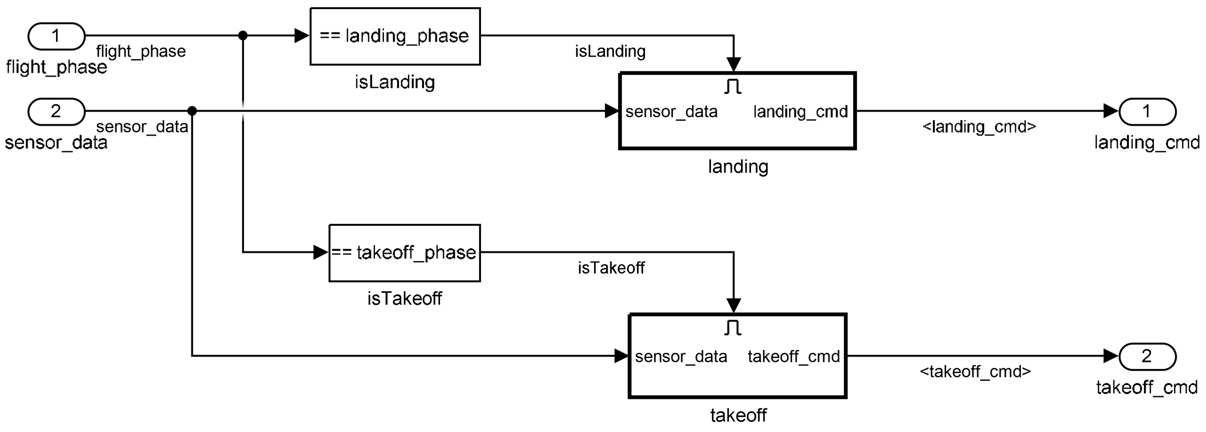

Fig. 7 *Exclusive enabled subsystems*

```
if (enabled_subsystem_U->flight_phase == landing_phase) {
    landing(&enabled_subsystem_U->sensor_data, &enabled_subsystem_B->landing_cmd);
  }

if (enabled_subsystem_U->flight_phase == takeoff_phase) {
    takeoff(&enabled_subsystem_U->sensor_data, &enabled_subsystem_B->takeoff_cmd);
  }
```

Fig. 8 *Code generated from the model shown in Fig. 7 (comments removed)*

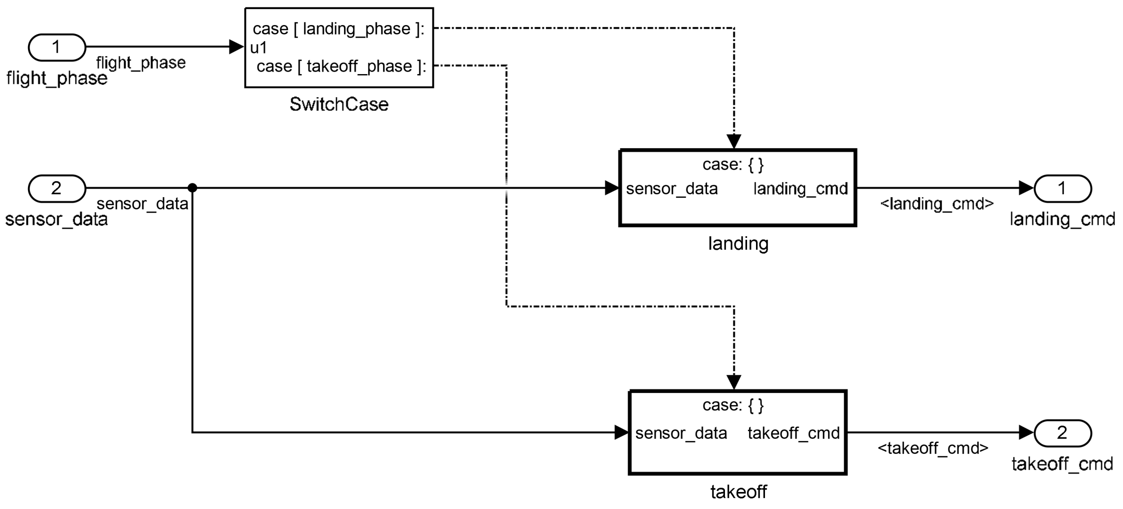

Fig. 9 *Switch-case subsystems*

```
switch (IfElse_subsystem_U->flight_phase) {

   case landing_phase:
    landing(&IfElse_subsystem_U->sensor_data, &IfElse_subsystem_B->landing_cmd);
    break;

   case takeoff_phase:
    takeoff(&IfElse_subsystem_U->sensor_data, &IfElse_subsystem_B->takeoff_cmd);
    break;

  }
```

Fig. 10 *Code generated from the model shown in Fig. 9 (comments removed)*

generator. An attempt to automate this process is presented in [40]. If the automatically generated annotations will not be reviewed manually, the generator needs to be qualified. Both variants lead to a huge effort. Thus, the model and code should be restructured in order to omit such effort. The goal is to get structures where the exclusive functionalities are represented in code constructs such as switch-case or if-else constructs. Then aiT can determine the correct control flow without any further annotations. Restructuring the model can also be beneficial for other process steps that also rely on abstract interpretation. For example, a static code analysis in order to detect runtime errors.

Fig. 9 shows a restructured model. With this model, an exclusive switch-case statement is generated in the code as shown in Fig. 10.

Restructuring the trajectory sub-model, where enabled subsystems were frequently used, reduced the calculated WCET by 44%. This shows that a good software architecture must be chosen if tight WCET bounds shall be achieved without further analysis of the design model for control flow constraints. It is important that the modeller knows what code constructs are generated out of certain model constructs by the used code generator.

This is an open access article published by the IET under the Creative Commons Attribution License
(http://creativecommons.org/licenses/by/3.0/)

8 Conclusion

This paper shows that it is possible to get a safe execution time bound for software, which is generated out of a Simulink model developed during system development. To get tight bounds for the analysed WCET without a huge effort in the analysis, exclusive parts of the model must be implemented in exclusive code constructs. To achieve this goal, the modeller must be aware of model structures that lead to such constructs. The usage of the CompCert leads to a huge improvement in the execution time in comparison to the non-optimised compilation. The benefit here is bigger than applying optimisations during the auto code generation. The most significant improvement can be achieved if both optimisations are applied. In future work, it shall be investigated if it is possible to improve the precision of the WCET in case of applied optimisation during the auto code generation. Further topics of future work are to investigate how the formal proofs of the compiler can be beneficial in a certification process, and how to close the gaps in the non-formally proven process steps like the pre-processor. Further, it shall be analysed if it is possible to perform all necessary verification activities with the model developed during system development process. The necessary activities depend on the benefit, which can be taken from the formally proven compiler and the possibility to automatically inspect the source code for compliance with the Simulink model.

9 Acknowledgments

The authors thank Absint Software for providing an evaluation licence of aiT and an executable binary of the CompCert. They also thank all our colleagues who implemented the Simulink model that was used for this case study. Moreover, special thanks to Volker Schneider for changing his model to the proposed structure.

10 References

[1] AC: 23.1309-1E: 'Advisory circular: system safety analysis and assessment for part 23 airplanes', November 2011

[2] STANAG 4671 Edition 1: 'Unmanned aerial vehicles systems airworthiness requirements', May 2007

[3] Stecklein, J.M., Dabney, J., Dick, B., *et al.*: 'Error cost escalation through the project life cycle'. 14th INCOSE Int. Symp. Annual, Toulouse, France, June 2004

[4] DeGarmo, M.: '*Issues concerning integration of unmanned aerial vehicles in civil airspace*' (2004)

[5] Braun, S., Geiser, M., Heller, M., *et al.*: 'Configuration assessment and preliminary control law design for a novel diamond-shaped UAV'. Int. Conf. on Unmanned Aircraft, Orlando, FL, May 2014, pp. 1009–1022

[6] Kreienfeld, M., Giese, K., Heider, J., *et al.*: 'Development of a RPV-demonstrator for ATM research'. SCI-269 Symp. on 'Flight Testing of Unmanned Aerial Systems (UAS)', Ottawa, Canada, May 2015, pp. 18:1–18:10

[7] Karlsson, E., Schatz, S.P., Baier, T., *et al.*: 'Automatic flight path control of an experimental DA42 general aviation aircraft'. 14th Int. Conf. on Control, Automation, Robotics and Vision, Phuket, Thailand, November 2016

[8] 'Simulink', https://de.mathworks.com/products/simulink.html, accessed 18 April 2017

[9] MathWorks: 'Embedded coder users's guide' (2016)

[10] DO-178C: 'Software considerations in airborne systems and equipment certification', December 2011

[11] DO-331: 'Model-based development and verification supplement to DO-178C and DO-278A', December 2011

[12] AMC RPAS.1309: 'Safety assessment of remotely piloted aircraft systems', November 2015

[13] Weber, G., Lammering, T., Thierer, S., *et al.*: 'The Liebherr fully integrated FCS design – a case study'. Aviation Technology, Integration, and Operations Conf., Los Angeles, CA, August 2013

[14] Walde, G., Luckner, R.: 'Bridging the tool gap for model-based design from flight control function design in Simulink to software design in SCADE'. 2016 IEEE/AIAA 35th Digital Avionics Systems Conf. (DASC), Sacramento, CA, September 2016, pp. 1–10

[15] França, R.B., Favre-Felix, D., Leroy, X., *et al.*: 'Towards formally verified optimizing compilation in flight control software'. PPES 2011: Predictability and Performance in Embedded Systems, Grenoble, France, March 2011, pp. 59–68

[16] Görke, S., Riedeling, R., Kraus, F., *et al.*: 'Flexible platform approach for fly-by-wire systems'. Digital Avionics Systems Conf. (DASC), East Syracuse, NY, October 2013, pp. 2C5-1–2C5-16

[17] Alvis, W., Murthy, S., Valavanis, K., *et al.*: 'FPGA based flexible autopilot platform for unmanned systems'. Mediterranean Conf. on Control Automation, Athens, Greece, June 2007, pp. 1–9

[18] Klenke, R.H., Sleemann IV, W.C., Motter, M.A.: 'A high-throughput processor for flight control research using small UAVs'. 25th AIAA Aerodynamic Measurement Technology and Ground Testing Conf., San Francisco, CA, June 2006

[19] DO-254: 'Design assurance guidance for airborne electronic hardware', April 2000

[20] Ermedahl, A., Jakob, E.: 'Execution time analysis for embedded real-time systems', in Lee, I., Leung, J.Y.-T., Son, S.H. (Eds.): '*Handbook of real-time and embedded systems*' (Chapman & Hall/CRC, Boca Raton, London, 2008), pp. 437–455

[21] Krause, C., Holzapfel, F.: 'Designing a system automation for a novel UAV demonstrator'. 14th Int. Conf. on Control, Automation, Robotics and Vision, Phuket, Thailand, November 2016, pp. 1–6

[22] Schneider, V., Holzapfel, F.: 'Modular trajectory generation test platform for real flight systems'. CEAS EuroGNC, Warsaw, Poland, April 2017

[23] Kügler, M.E., Holzapfel, F.: 'Designing a safe and robust automatic take-off maneuver for a fixed-wing UAV'. 14th Int. Conf. on Control, Automation, Robotics and Vision, Phuket, Thailand, November 2016, pp. 1–6

[24] Schatz, S.P., Holzapfel, F.: 'Modular trajectory / path following controller using nonlinear error dynamics'. Aerospace Electronics and Remote Sensing, Yogyakarta, Indonesia, November 2014, pp. 157–163

[25] Karlsson, E., Schatz, S.P., Holzapfel, F., *et al.*: 'Development of an automatic flight path controller for a DA42 general aviation aircraft'. CEAS EuroGNC, Warsaw, Poland, April 2017

[26] Schatz, S.P., Holzapfel, F.: 'Nonlinear modular 3D trajectory control of a general aviation aircraft'. CEAS EuroGNC, Warsaw, Poland, April 2017

[27] 'Simulink Code Inspector', https://de.mathworks.com/products/simulink-code-inspector.html, accessed 18 April 2017

[28] Hochstrasser, M., Schatz, S.P., Nürnberger, K., *et al.*: 'Aspects of a consistent modeling environment for DO-331 design model development of flight control algorithms'. CEAS EuroGNC, Warsaw, Poland, April 2017

[29] Leroy, X.: 'Formal verification of a realistic compiler', *Commun. ACM*, 2009, **52**, (7), pp. 107–115

[30] Blazy, S., Leroy, X.: 'Mechanized semantics for the Clight subset of the C language', *J. Autom. Reasoning*, 2009, **43**, (3), pp. 263–288

[31] MISRA-C: 2004: 'Guidelines for the use of the C language in critical systems', October 2004

[32] Leroy, X.: '*The CompCert C verified compiler: documentation and user's manual*' (2016, 2nd edn.)

[33] DO-248C: 'Supporting information for DO-178C and DO-278A', December 2011

[34] CAST-20: 'Addressing cache in airborne systems and equipment', June 2003

[35] 'aiT', https://www.absint.com/ait/, accessed 18 April 2017

[36] Wilhelm, R., Engblom, J., Ermedahl, A., *et al.*: 'The worst-case execution time problem overview of methods and survey of tools', *ACM Trans. Embed. Comput. Syst.*, 2008, **7**, (3), pp. 36 : 1–36 : 53

[37] AbsInt Angewandte Informatik GmbH: '*AbsInt advanced analyzer for PowerPC e300 user documentation*', October 2016

[38] Baufreton, P., Heckmann, R.: 'Reliable and precise WCET and stack size determination for a real-life embedded application'. ISoLA Workshop On Leveraging Applications of Formal Methods, Verification and Validation, France, December 2007, pp. 41–48

[39] Souyris, J., Le Pavec, E., Himbert, G., *et al.*: 'Computing the worst case execution time of an avionics program by abstract interpretation'. 5th Int. Workshop on Worst-Case Execution Time (WCET) Analysis, Palma de Mallorca, Spain, July 2005, pp. 21–24

[40] Wilhelm, R., Lucas, P., Parshin, O., *et al.*: 'Improving the precision of WCET analysis by input constraints and model-derived flow constraints', in Chakraborty, S., Eberspächer, J. (Eds.): '*Advances in real-time systems*' (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012), pp. 123–143

This is an open access article published by the IET under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/)