Design Project II Final Report
Developing a Representative Automotive Embedded Control Loop

Anthony Delage 260476213
Alexander Lunan 260372311
Gregory Williamson 260426256

April 14, 2015

Abstract

Today's automotive, aviation, and medical industries rely considerably on electronics to perform a multitude of tasks, which range in importance based on how they impact the safe operation of the overall system. Whereas a failure executing a non-critical task would lead to decreased user experience in most worst-case scenarios, errors in the execution of critical tasks can have have tragic or even fatal consequences. The reliability constraint placed on safety-critical embedded systems has traditionally forced the use of a method called redundant lockstepping to safely detect errors. While this method is reliable, its implementation comes at the cost of scheduling inflexibility, added design complexity, and high communication bandwidth. An alternative method that has been proposed to avoid making these trade-offs is execution fingerprinting. This method has been studied and implemented in software and simulated on virtual hardware by Professor Brett Meyer and Mr. Jonah Caplan of McGill University, yet has not been tested in a fully representative situation. The team's task in the context of this project is to develop a representative automotive control loop that allows the evaluation of the already-implemented fingerprinting framework in a characteristic test case. This control loop simulates the functionality of an embedded microprocessor in an automobile by capturing inputs from sensors, processing this data, then acting upon this information by sending related control signals. The loop was designed using MATLAB's Simulink library, converted to C code using Simulink's Embedded Coder feature, implemented into the existing fingerprinting framework, then tested on a virtual platform using Imperas' OVPSim environment. Though the team was not able to fully implement the control loop in the final fingerprinting environment, it was able to test individual sub-parts of the loop successfully, and is confident that these results will help further the advancement of research in this area. This report will focus on the team's tasks within the project, highlighting the specifics of the fingerprinting method, the design of the control loop, and the testing process. By facilitating the evaluation and performance of execution fingerprinting through the implementation of the embedded control loop, the team hopes to bring execution fingerprinting one step closer to adoption by industry and to playing a discrete yet valuable role in everyday life.

Acknowledgements

The group members would first like to acknowledge the help of Professor Brett Meyer, who has been of invaluable assistance to us. Our team also wishes to acknowledge the help of Honours Thesis student Jonah Caplan. Mr. Caplan has been working tirelessly on this project for over two years, and has been of great help to us throughout this project. The guidance, patience and advice of Professor Meyer and Mr. Caplan is greatly appreciated.

Contents

List of Figures

1 List of Abbreviations and Acronyms

Abbreviation or Acronym	Meaning
ECU	Electronic Control Unit
DMR	Dual Modular Redundancy
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
RTOS	Real-Time Operating System

2 Introduction

As the human race progresses well into the twenty-first century, we live in an age where electronics play an increasingly vital role in many aspects of our lives. The sweeping changes to society brought about by innovations such as personal computers, smart phones and tablets is both obvious and well-documented, yet studying these devices would only offer a small glimpse at the virtual omnipresence of electronics today.

One relatively unheralded field where electronic advancements have been crucial is in the realm of safety-critical embedded systems. A modern automobile can contain over 100 ECUs, most of which operate with the explicit purpose of assuring the safety of its passengers [2], while a similar paradigm exists in the medical device, avionic, and railway industries [1]. While these devices routinely do what was once thought to be impossible, there is a constant expectation that each new generation safety system perform better, cost less, and be more reliable than its predecessor.

The current status quo in these applications is the usage of lockstepped redundant cores to perform any safety-critical task [1]. Briefly put, this means that each computer core used is actually made up of multiple permanently connected cores working in tandem. By having multiple redundant evaluations of the same computation, the system can avoid letting any error propagate to the outside world. While assuring a system's reliability, this method has certain drawbacks, most notably that any non-critical task will use additional resources that simply aren't needed. In addition, designing a tightly lockstepped system can be difficult and cost inefficient [3].

An alternative approach that has been proposed to remedy these issues is called *execution fingerprinting* [4]. Rather than having multiple cores perpetually connected and forced to redundantly execute tasks, fingerprinting proposes a system in which each core acts independently. When redundancy is required, a comparison method that will be detailed later in this paper is used to ensure the reliable completion of any safety-critical task. Figure 1 shows, on the left, six cores connected in a traditional lockstep scheme. To the right, the same six cores are set up according to the fingerprinting method, able to operate both independently or redundantly. Fingerprinting improves upon the lockstep method by being simpler to design for, more efficient when executing non-critical tasks, and requiring less communication bandwidth between cores [4] [3].

The team's role in the continued development of fingerprinting as an alternative to lockstepped redundancy is to develop a representative automotive control loop within the context of Professor Brett Meyer and Honour's Thesis student Jonah Caplan's research on the topic. While Professor Meyer and Mr. Caplan have developed an environment implementing the fingerprinting method, their research requires a representative test case. The end goal of this project is ultimately to verify if the findings of Professor Meyer and Mr. Caplan hold true under conditions mimicking a real-life safety-critical system. The team was required to conduct sufficient research to educate ourselves in topics relating to safety-critical systems (summarized in Section 3), create the control loop, and

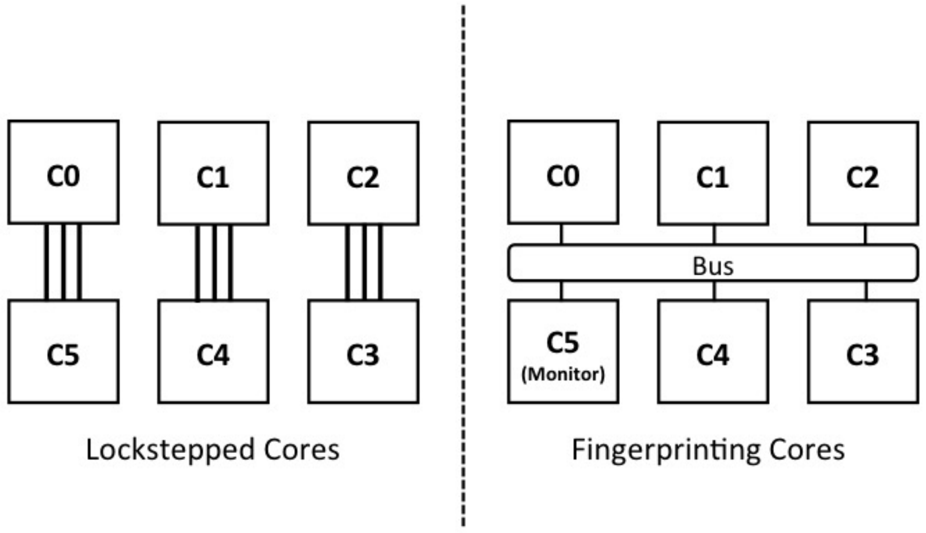

Figure 1: Comparison of Core Distribution in Lockstep Versus Fingerprinting

initiate testing (explained in Sections 4 and 5). This component of Professor Meyer's overall project will provide valuable feedback on the designed platform and hopefully bring fingerprinting one step closer to implementation in the safety-critcal devices we use every day.

3 Background

This section draws heavily on content in the team's first semester report [5].

The most important characteristic differentiating safety-critical systems from any other embedded system is the absolute need for reliability. Whereas an erroneous computation in a smart phone or tablet will, at worst, result in a decreased user experience, it can have absolutely disastrous, even fatal consequences in the transportation or medical fields. Because experts have forecast an increase in fault rates in computer microprocessors and memory, the topic of fault tolerant computer architecture has seen a recent surge in popularity [1]. For anyone involved in the design of safety-critical embedded systems, however, this is nothing new. In these application areas, immunity to errors has, and always will be a top concern.

3.1 Fault Tolerant Computing

Faults in computer systems can occur for a variety of reasons. A *permanent* fault is one that, as its name suggests, persists for the life of the system [1]. In a safety-critical application, such a fault would ideally lead to the replacement of the core in question, as it would no longer be reliable. An *intermittent* fault is one that, while not occurring continuously, will repeat itself periodically within a system [1]. From a safety-critical perspective, this type of fault is very similar to a permanent one, and again would hopefully lead to the processor's replacement. The final type of fault seen in computers is called a *transient* fault. This type of fault happens once and does not repeat itself [1]. In most cases, these errors, called *soft errors*, are caused by high-energy cosmic neutrons hitting transistors, thus changing the value of the bits stored in those locations [6]. These faults are the most concerning to safety-critical system designers, as they are essentially unavoidable, despite their relative rarity.

The amount of research conducted in recent years concerning error detection, correction, diagnosis, and even permanent repair is staggering. The number of methods proposed to simply detect errors is ever-increasing, and ranges from simple error detection codes to complex hardware setups [1]. The unfortunate reality plaguing most of these methods, however, is that they can rarely make concrete guarantees about their correctness. In a safety-critical system, it is simply not enough to choose an error detection scheme based on a low probability of a fault; the chance of failure must also be quantifiable. This is why, as rapid progress is made in the field of fault tolerant computer architecture overall, safety-critical systems have evolved at a much slower pace.

3.2 Lockstep Execution: The Status Quo For Critical Systems

Currently, microprocessors in safety-critical applications are implemented following the rules of tight lockstep redundancy. This method forces two or more cores to perform the same operations side-by-side, compares their outputs with a checker, and proceeds with their results as long as they agree [1]. Figure 2 shows a tightly lockstepped core setup consisting of two cores, commonly called Dual Modular Redundancy (DMR).

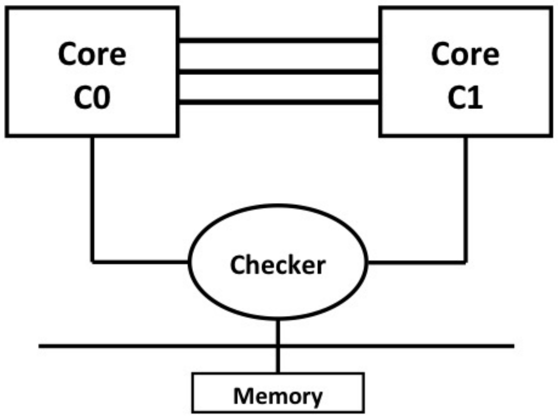

Figure 2: Tightly Lockstepped Redundancy Core Structure

The reason for this method's widespread popularity in safety-critical contexts is its virtual imperviousness to transient faults. Technically, both cores could be affected by soft errors simultaneously and transmit the same erroneous output, but the probability of this happening truly is negligible. To reduce the odds of such a coincidence happening even further, designers in fields such as avionics sometimes use systems such as triple-triple lockstep redundancy, seen in Figure 3.

The main downside to lockstep redundancy as an error detection method is that its reliability comes at the cost of many performance-related factors. As mentioned previously, the use of tightly coupled cores is very inefficient when executing non-critical tasks as the added redundancy can be described as just that: redundant. System scheduling is extremely restricted by this method, as non-critical tasks end up controlling many more resources than they should [7]. The lockstep method also requires significant communication bandwidth between lockstepped cores [4], leading to large and unwanted execution overheads.

It would be easy to look at the constraints placed on safety-critical systems and see a bleak outlook for any material advancement in their performance and efficiency. However, notable progress has and will continue to be made in this area. It has been shown that by simply removing the

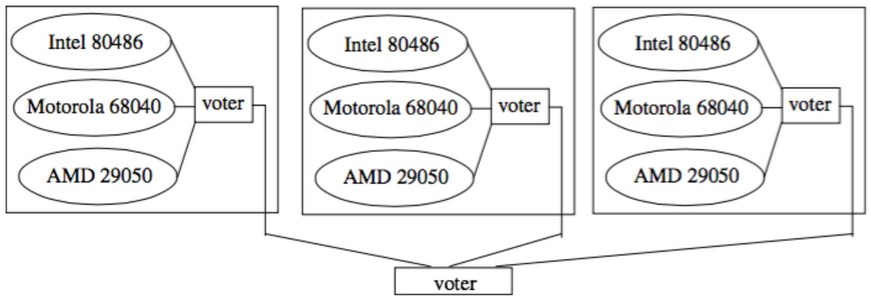

Figure 3: Triple Triple Redundancy Scheme Used in Boeing's 777 [1]

lockstep constraint on non-critical tasks, a method called *relaxed dedication*, up to 50% more non-critical tasks can execute within a DMR system [7].

3.3 Execution Fingerprinting For Safety-Critical Systems

Pushing the concept of relaxed dedication even further, one can easily imagine a system where all cores act autonomously, and a new method is used to negate soft errors for safety-critical tasks. A method proposed that follows this line of though is execution fingerprinting, and its basic structure is depicted in Figure 4.

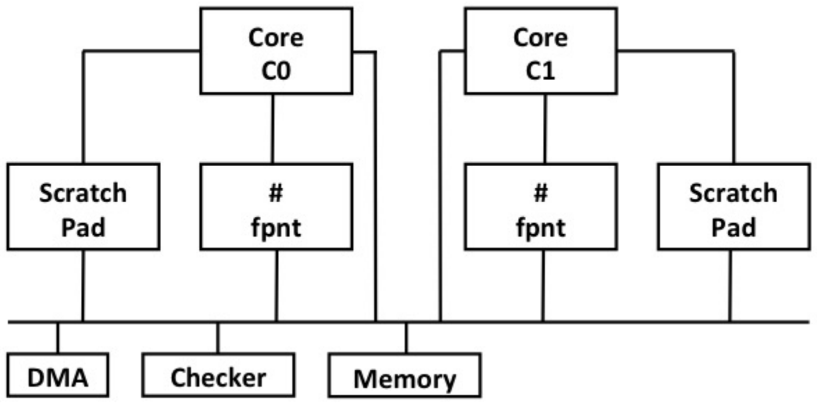

Figure 4: Fingerprinting Method Core Structure

In a microprocessor performing fingerprinting, every core acts individually by default, with one such core designated as the monitor, it itself being fault-tolerant, oftentimes through the use of lockstepping. This monitor core not only executes computations on its own, it also must act as a scheduler for the rest of the processor. When a critical task is initiated, the monitor will designate two or more cores to execute any computations necessary. Once these cores have completed their tasks, the change of state they wish to incur is buffered in a scratchpad memory location. In addition, a version this change of state is compressed using a hash function, typically cyclic redundancy check (CRC), yielding what is referred to as an execution fingerprint [8]. This fingerprint is then compared to that of all other cores redundantly executing the same task; if the fingerprints match, their related change of state is released from the scratchpad buffer and allowed to take effect [4]. A point worth noting is that all cores in a fingerprinting system access main memory via Direct Memory Access (DMA), which reduces memory access latency.

This method of error checking has many advantages. First, non-critical tasks have many more windows in which they can execute without using additional resources. This reality makes scheduling much simpler in a fingerprinting system compared to a traditional lockstep system [3]. Additionally, much less bandwidth is required for inter-core communication, since only their compressed fingerprints are being compared [3]. Finally, by tuning the number of cores redundantly executing the same critical task, it is possible to adjust the error detection probability relative to the system's application.

4 Requirements

This section draws heavily on content in the team's first semester report [5].

4.1 Scope Of The Design Project

The team's mandate in this project was to design a representative automotive control loop that would act as a testing benchmark for the existing fingerprinting system created by Professor Meyer and Mr. Caplan. While the functionality of the fingerprinting framework was not in question, the control loop was needed to simulate the actual performance of the fingerprinting process in a real-life safety-critical application. This would permit the design team to obtain quantitative performance metrics for the execution fingerprinting process which could be compared to known benchmarks for lockstep execution.

Under Professor Meyer's guidance, Mr. Caplan had created a file structure of representative C code describing the fingerprinting process prior to our team joining the project. All of the hardware components associated with the fingerprinting process were represented using C, including the scratchpad memory, comparators, and fingerprint generator (CRC hashing function) among others. This component of the overall project created by Mr. Caplan was ready and waiting for our control loop to be constructed to permit testing upon our joining the project in September. In order to develop a suitable control loop for this project, we were required to understand the behaviour of a control loop from a theoretical standpoint.

In any embedded system, a microprocessor's operation generally follows three sequential phases within its control loop. First, the system will use some variety of sensor to acquire data from its environment, the *sensing* phase. Second, it will process this data and extract information from it, the *processing* phase. The third phase, which is optional, involves using this information to operate a variety of actuators and therefore respond to its environment. A car's cruise control system which senses a car's current speed, determines if it is the driver's desired speed, and makes any necessary changes is an example of an embedded control loop, and can be seen in Figure 5.

4.2 Safety-Critical vs. Non-Critical Tasks

Cruise control in an automobile is an example of a safety-critical task. An error in the cruise control system could potentially cause a life-threatening change to the car's speed. Other examples of safety-critical tasks include airbag deployment monitoring, collision avoidance and traction control in cars, as well as autopilot functions in airplanes, electronic signaling and switching systems for railways and electronic hospital monitoring instruments in the medical industry. These tasks differ

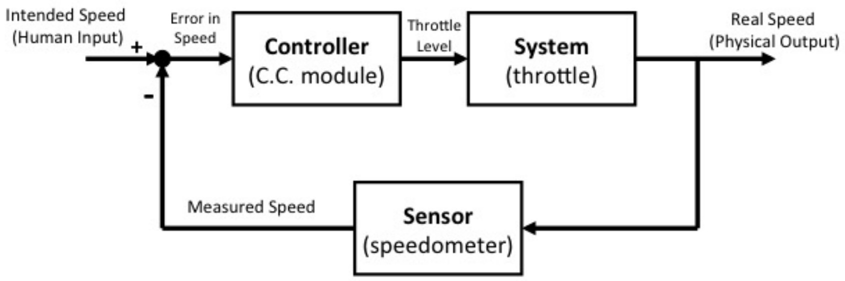

Figure 5: Automobile Cruise Control Embedded Control Loop

starkly from non-critical tasks, which do not cause a life-threatening situation if an error propagates to the outside world. Examples of non-critical tasks include fuel guage and fuel economy readouts on automobiles and passenger audiovisual entertainment systems aboard airplanes. The goal for our team was to prepare an embedded system with both safety-critical and non-critical tasks to be executed, which is what fingerprinting was optimized to execute on. By evaluating the current fingerprinting framework in a representative scenario, there will be an opportunity to re-evaluate any assumptions or conclusions drawn from previous research and testing performed by Professor Meyer and Mr. Caplan, and thus further the research already done.

Of note regarding the loop to be designed is the term *representative* in its description. It would be easy to assume that this representativeness has to do with the control loop closely mimicking a car's internal process. In reality this is of very little importance. From the perspective of the outside world, it does not matter whether or not the loop closely resembles what one would find in an automobile. Instead, the term representative is important from the fingerprinting system's point of view, in that it must act as if it was controlling an automobile. This distinction is subtle, yet of vital importance.

For the designed loop to be compliant with this requirement, little attention will have to be paid to its inputs or outputs. Whether or not the sensors act like those in a cruise control loop, or that the system outputs signals that would properly control a vehicle, is unimportant. What the control loop must therefore do to be deemed representative is supply the system with a mixed set of critical and non-critical tasks over a fixed time period. The rate of arrival and number of these tasks, regardless of their actual purpose, is what must be representative of an automotive control loop. It is worth nothing that the choice of an automotive control loop is purely cosmetic. Had the application area been different, the overall goal would remain the same.

4.3 Required Steps

To design the embedded control loop and implement it within the current fingerprinting structure, our team was instructed by Professor Meyer and Mr. Caplan at the outset of the project to follow the multi-step process which is described briefly below. A considerably more thorough description of the design process is presented in Section 5.

First, the actual loop was to be designed using Simulink, a MATLAB extension that allows a user to graphically model and simulate dynamic systems. Our team had not worked with Simulink prior to this project, and woud therefore have to learn how to use the software through tutorials and experimentation. The team was provided with sufficient training material to obtain a functional knowledge of Simulink, thus allowing it to generate a control loop. The control loop our team developed is highlighted in detail in Section 5.

As a second step in the design process, a MATLAB extension called Embedded Coder was required to convert the Simulink control loop into the C programming language. Since the fingerprinting structures have been generated in C code by Mr. Caplan, the control loop needed to be coded in C as well so that they could mesh together. This may seem like a small step, but this generated code would have to be modified several times to mesh appropriately with the rest of the project. In effect, studying and becoming knowledgeable in the C programming language is another critical component of this project that does not pertain directly to control loop design.

Third, a software called Quartus would be used to interface with a Nios II embedded microprocessor. Within Quartus, a version of the popular Eclipse IDE was used to write C code that properly interfaced with the processor using preexisting libraries. This step involved the combination of C code associated with our control loop and the C structure associated with Mr. Caplan's fingerprinting structures.

Fourth and finally, the control loop and fingerprinting C code would be tested on a virtual platform using Imperas' OVPSim technology. OVPSim is a multiprocessor emulation platform that allows users to simulate multi-core applications in real-time, while also providing a debugging interface [9]. By using a virtual platform to simulate the fingerprinting method rather than an actual Nios II microprocessor, the team was able to gain much more insight into the inner workings of the processor, in addition to having an easier time eliminating bugs from the system. The team expected its most valuable insights to be gained in this step, as this is where any representative simulation would occur. The duration and extensiveness of this step would be dependent, however, on the amount of time remaining for testing once all of the previous four steps were completed. While a goal was set to begin testing on a cruise control loop, it was understood that debugging and repeating work from earlier steps could lead to delays in performing real quantitative testing.

A flowchart depicting the full process can be seen in Figure 6.

The main constraint affecting this project was that all work had to mesh coherently with the work already done by Professor Meyer and Mr. Caplan. Failure to meet this requirement would prevent either the group or future groups from performing any concrete tests on the fingerprinting method.

One final requirement on the team's part is to properly document any and all steps taken to implement the control loop within the existing fingerprinting framework. The process undertaken to reach a final result is as important if not more so than the product itself. While the team may be the first to attempt this task, there will certainly come a point where someone else needs to either modify the existing control loop or generate a new one from scratch. Properly documenting the team's design process will enable those who come after to learn from its mistakes and improve upon its work.

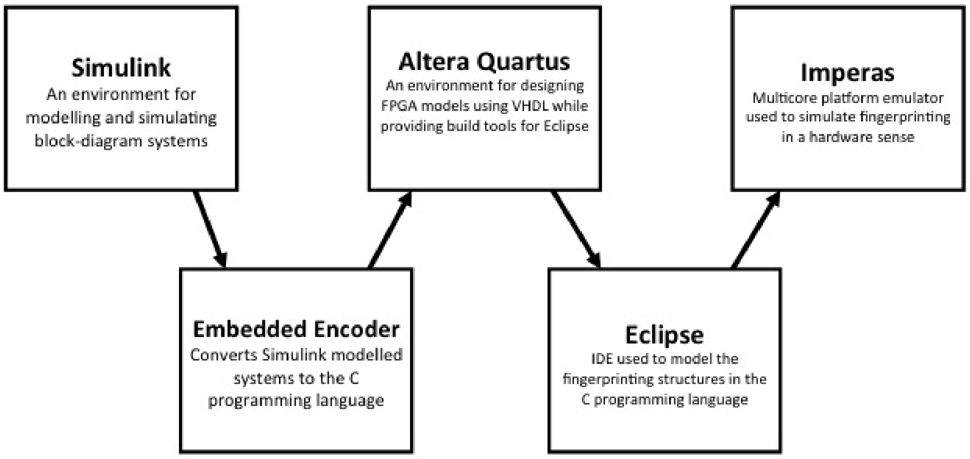
Simulink
An environment for modelling and simulating block-diagram systems
Embedded Encoder
Converts Simulink modelled systems to the C programming language
Altera Quartus
An environment for designing FPGA models using VHDL while providing build tools for Eclipse
Eclipse
IDE used to model the fingerprinting structures in the C programming language
Imperas
Multicore platform emulator used to simulate fingerprinting in a hardware sense

Figure 6: Control Loop Generation and Implementation Process

5 Design and Results

5.1 Theory

Once the team had fully understood and gained the prerequisite knowledge outlined above as requirements in Section 4, it was ready to embark on the design process. The first step was to design the control loop in Simulink. It was decided that the group would try to replicate (with some simplifications) an automotive embedded system. This system would monitor different inputs that were measured by instruments on the automobile, perform calculations and process data, and generate outputs which could perhaps adjust the physical activity of some of the car‘s moving parts or transmit indications to the driver. An automobile embedded system was chosen since it executes a variety of non-critical and safety-critical tasks which are somewhat familiar to the team members (having all driven automobiles and interacted with some of these features), compared to airplane embedded systems and the tasks they perform, which are relatively foreign to people outside this industry. The system was designed assuming a four-wheeled automobile with two-wheel drive (either the front or rear axle is powered).

In order to accurately but efficiently portray an automobile's embedded system, the group selected four safety-critical tasks and two non-critical tasks to model in Simulink. The four safety-critical tasks were cruise control, traction control, airbag control and collision avoidance, which are described in the paragraphs below. These discussions include some of the physics behind our systems as well. These control loops (to be referred to as modules) were created by our team with simplicity in mind, with strategies drawn from examples of cruise control and airbag control systems presented in publicly available Simulink tutorials. The fully designed system can be seen in Figure 7.

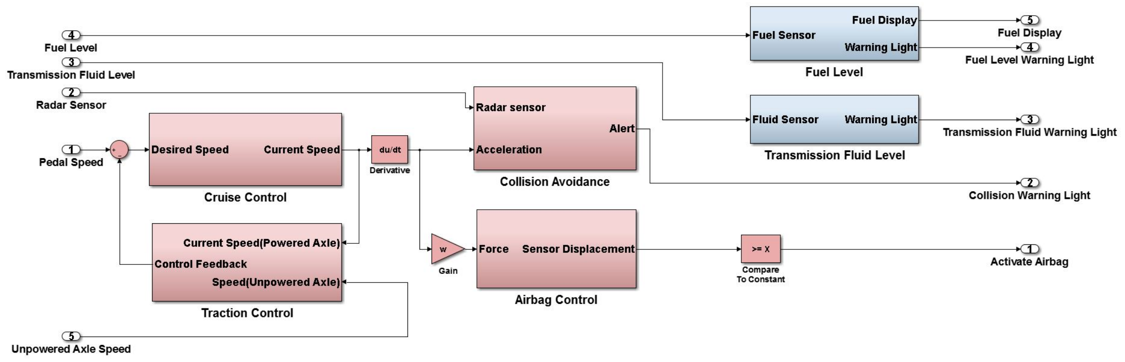

Figure 7: Final Design of the Embedded Automotive Control Loop

5.2 Cruise Control

The cruise control module was the first loop created in Simulink. Its implementation is shown in Figure 8. Cruise control is a function available in many mid- to well-equipped automobiles which

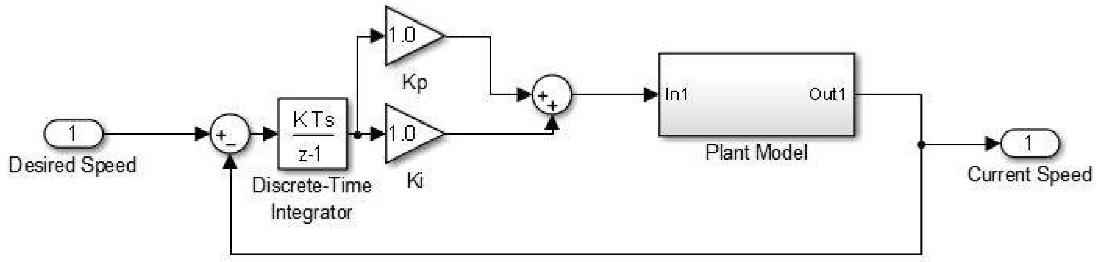

Figure 8: Simulink Design of the Cruise Control Module

allows the driver to set a desired speed for the automobile to travel at. It is then up to the car to maintain that set speed until the driver adjusts the desired speed by altering the cruise control setting or by applying one of the braking and acceleration pedals. The first input to the system is the desired speed set by the driver by adjusting the cruise control knob or button. This electronic signal carrying information about how fast the driver wants the car to travel at is then compared to a signal representative of the actual speed of the car, typically related to the revolutions of the drivetrain or powered axle. If a significant difference is noticed between the desired speed and actual speed, the throttle is increased or decreased until the desired speed is attained. This is a safety-cririical task as an error in speed selection can potentially have fatal consequences.

5.3 Traction Control

The traction control module was designed to address safety issues when the car's wheels are slipping due to slick road conditions, drastic acceleration, and skidding. The module is designed for a vehicle with two-wheel drive, where either the front or rear axle is powered (connected to the drivetrain and transmission) and the other axle is unpowered. The traction control module takes both the rotational speeds of the powered and unpowered axle as input. The goal of the system is to correct the speed of the powered axle when its wheels are slipping, as indicated by a large discrepancy between the rotational velocity of the powered and unpowered axle. If the discrepancy between the two speeds is greater than a certain threshold (a *compare to constant* block in Simulink is used to decide if so), then the feedback to the throttle controller will raise or lower the power to the driven axle to make its speed closer to that of the unpowered axle. The loop must consistently verify that the axle speeds are similar to avoid an unsafe situation such as a *spinout*. While traction control is a relatively modern feature in North American cars (gradually becoming more commonplace since the 1990s), the team considers this to be a safety-critical task if the feature is installed on a vehicle, since an error in its functionality could result in dangerous shifts in drivetrain velocity. Accordingly, traction control is often referred to as a "safety feature" in vehicle descriptions. Its Simulink schematic is shown in Figure 9.

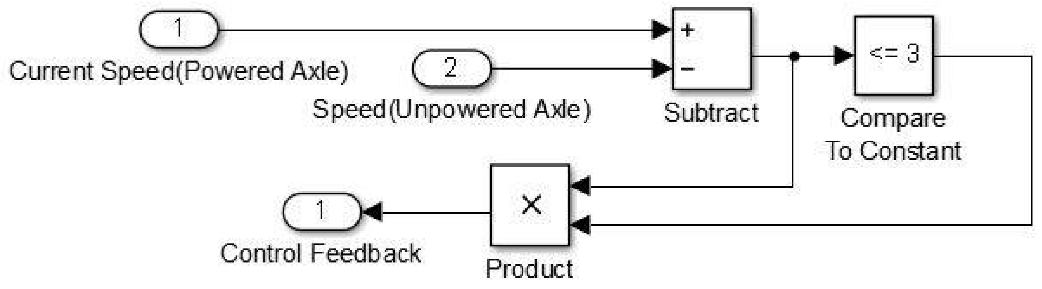

Figure 9: Simulink Design of the Traction Control Module

5.4 Airbag Control

The airbag control module was designed after completing a Simulink tutorial which touched on the subject. Airbags have become prominent in all North American cars since the 1990s and are intended to absorb the impact of a passenger's forward momentum in the case of an accident. In the team's system setup, a small weight is suspended, free to move if subjected to sudden changes in the car's speed (rapid deceleration or acceleration). The acceleration A of the weight is obtained from its displacement force F thanks to equation 1, in which m is the mass of the weight.

$$a = \frac{F}{m} \tag{1}$$

The acceleration is integrated twice over the time period of observation to obtain a value of displacement. If that displacement value exceeds that of an acceptable threshold amount, it is deemed that the car is decelerating at a dangerous rate and the airbags are deployed. Small changes in displacement are constantly fed back into the input to allow steady monitoring of the car's acceleration during a trip. The airbag control module is shown in Figure 10.

Undoubtedly airbag control is a safety-critical task. In the case of an accident there is a very short window of time in which the airbag must be correctly deployed. That being said, this model was designed by the team with simplicity in mind and may lack some of the complexity of real airbag systems used in cars today.

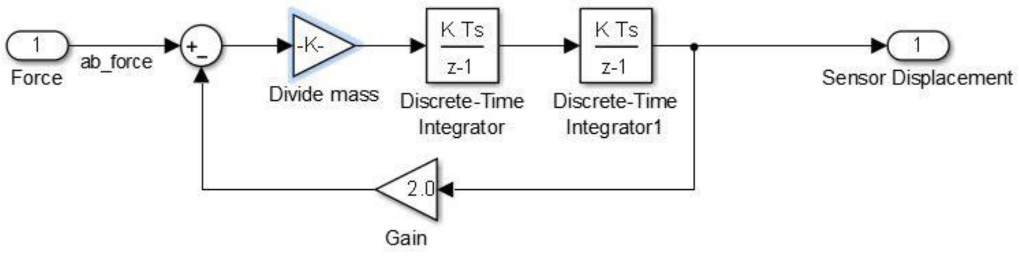

Figure 10: Simulink Design of the Airbag Control Module

5.5 Collision Avoidance

The collision avoidance module is the last of the system's safety-critical tasks. Its schematic is shown in Figure 11. This safety feature has recently gained popularity as an interactive way of reducing the risk of rear-ending another vehicle.

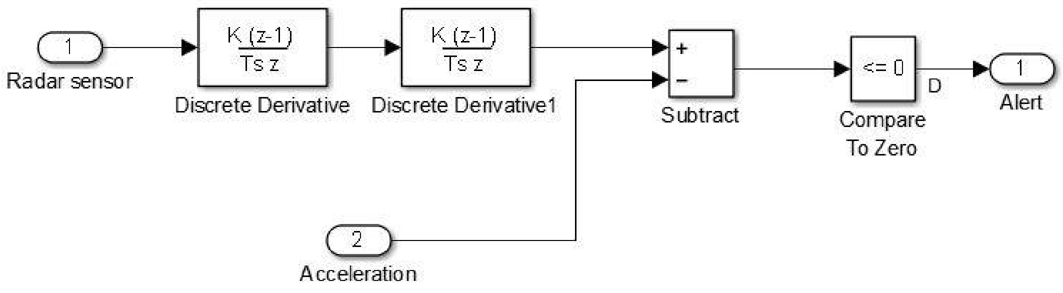

Figure 11: Simulink Design of the Collision Avoidance Module

Consider Car A is equipped with collision avoidance and is following Car B. By having Car A flash an indicator when it deems it is approaching Car B at a dangerous speed, the driver of Car A can be alerted to slow down. While the feature does not automatically slow the automobile down, it can certainly give the driver a chance to prevent a collision. The system was designed solely by the team and for the project's purposes only, without consulting current industry standards. These are outside the scope of this design project. The physics behind the collision avoidance module is depicted in Figure 12.

The collision avoidance module operates by engaging a radio frequency (RF) sensor on the front of Car A, which can project a radio wave off the back of Car B, providing Car A with the distance of separation. By taking the second derivative of this value with respect to time, the system obtain the relative acceleration of Car A with respect to Car B in front of it. This is then compared to the absolute acceleration of Car A itself, which can be obtained from the same source as the airbag module discussed above. If both cars are travelling at the same constant speeds, both the absolute acceleration of Car A and the relative acceleration of Car A with respect to Car B will be zero and no warning indication will occur. However, if, for example, Car A is travelling at a constant speed (zero acceleration) and Car B is slowing down, then the relative acceleration between Car A and Car B will be negative (Car B is accelerating toward Car A). If this exceeds a certain threshold, a warning light is activated to warn the driver of a possible impending collision.

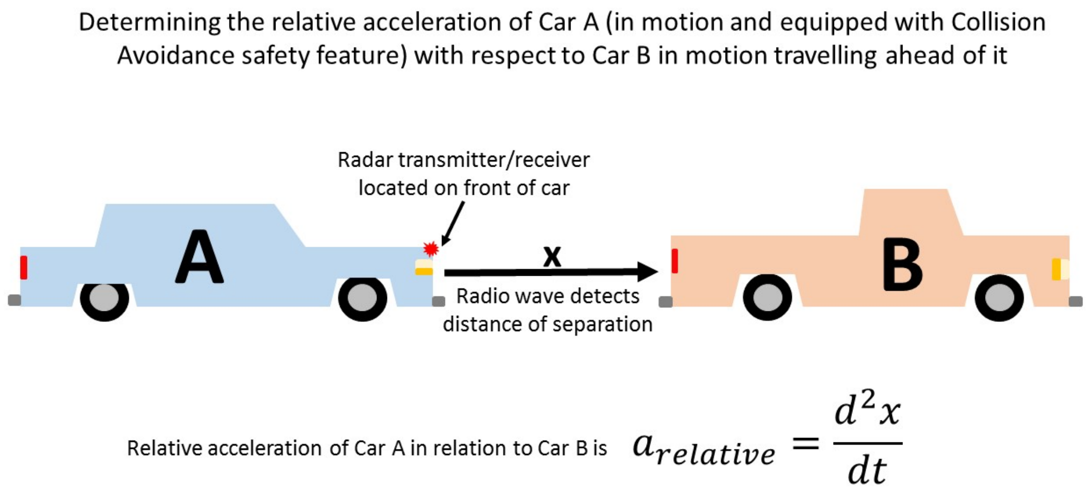
Determining the relative acceleration of Car A (in motion and equipped with Collision Avoidance safety feature) with respect to Car B in motion travelling ahead of it
Radar transmitter/receiver located on front of car
A
x
B
Radio wave detects distance of separation
Relative acceleration of Car A in relation to Car B is $a_{relative} = \frac{d^2x}{dt}$

Figure 12: Obtaining Relative Acceleration for the Collision Avoidance Module

If a vehicle is equipped with this safety feature, it must be considered a safety-critical task to its embedded processor, as a driver would likely depend on it to alert them before a potentially fatal collision.

5.6 Non-Critical Tasks

In order to make the most of execution fingerprinting, the team sought to simulate a mixed-criticality system where non-critical tasks had to be performed in addition to safety-critical tasks. While not as exciting as their safety-critical counterparts, non-critical tasks had to be created and were imperative to the overall evaluation of the fingerprinting framework. The group decided to model a fuel level gauge as a non-critical task. In a similar way, it was possible to duplicate the fuel level gauge and make small modifications to create similar models for transmission fluid level, coolant level, oil level and others. The functionality of a fuel gauge is not deemed a safety-critical task for the team's purposes in this project as the occurence of an error would not result in a potentially life-threatening situation. The model is very simple: A sensing instrument in the fuel tank sends a signal to the fuel gauge readout on the dashboard, and if that signal indicates a level lower than an accepted threshold, a "low fuel" warning light on the dashboard is activated. This process is repeated continuously while the vehicle is running.

5.7 Obtaining Representative C Code

With all of the above described modules representing tasks in the mixed-criticality embedded system, it was time to connect them all to represent how a real automobile ECU may function. Many of the inputs and outputs were common to two or more modules. Overall, the system, comprised of four safety-critical and two non-critical tasks, had five inputs and outputs. Differentiation and integration blocks were used to convert signals such as velocity from one module into acceleration in another. This allowed many signals to be reused and had a large simplifying effect on the system. Once the team had a representative automotive embedded control loop in Simulink, the next step was to convert it into C code to properly implement it within the preexisting fingerprinting framework.

To convert Simulink models to C code, the team used a Simulink extension called Simulink Embedded Coder. It is a program used to generate C code from Simulink, optimized specifically for embedded applications. In Simulink, it is very easy to generate code of an existing model, a process that takes less than ten minutes. The challenge here lies in generating the code under the right conditions and settings. One of the team's greater challenges was to chose the correct settings for Embedded Coder to operate properly, and to fine tune the Simulink models to allow them to be properly represented in C. One decision the team made to simplify this process was to replace all differentiation and integration blocks with discrete-time equivalents. This drastically reduced the amount of information the system had to compute, since continuous time no longer had to be modeled. In addition, all differential equations within the models were set to be solved using fixed-step methods, again to simplify the output.

The team's first attempt to represent the embedded system in C saw it generate code for the entire system at once, as one lengthy C program. While it was generated without error, upon further review and after consultation with Mr. Caplan, it was determined that a C file representing the entire system would improperly represent the real operation of an ECU, and would therefore consist of a poor test case. The team subsequently generated C files for each module separately, such that each task could be executed independently on the embedded processor. Once these files were generated, the team had the right mix of elements at the right level of complexity to properly test the existing fingerprinting architecture. Initially, the group encountered difficulties with compatibility with the existing program structure. Through an iterative and lengthy debugging process, errors were eliminated, resulting in code that could eventually compile using the Quartus and Eclipse build tools. One of the main modifications made to the code was to change how arguments were passed to the MATLAB-generated functions. Whereas these functions initially relied on global variables to operate, the team altered their construction to allow them to process pointer values. This would allow team to set fixed locations in memory for the data that was to be shared between modules rather than passing the data itself to each function. This is a standard practice when coding in C that reduces the required stack size for functions, and is therefore especially important in embedded applications.

Once the functions were properly altered to be called within the fingerprinting framework, the team had to create wrapper functions that would call them within the context of a real-time operating system (RTOS). The RTOS used in this project is called μCOSII, which stands for Micro Computer Operating System [10]. This part of the project proved to be much more difficult than expected, since the team hadn't adequately prepared when it came to understanding operating systems. The team eventually learned how to program within this environment, since this topic was covered in another class that two team members were taking called Microprocessor Systems (ECSE 426). In retrospect, the team should have identified the need to understand this layer of abstraction sooner and planned in consequence.

Designing the aforementioned wrapper functions also involved the creation of structures in C to hold the arguments that were to be passed to them. Once these were complete, the team had to properly integrate the designed functions via operating system tasks. To simplify the process, each core was given one operating system task to execute, within which the control loop functions would be called in a static order, as seen in figure 13. Every iteration of the task, therefore would call a repeated execution of the loop, therefore simulating an ECU sampling its inputs.

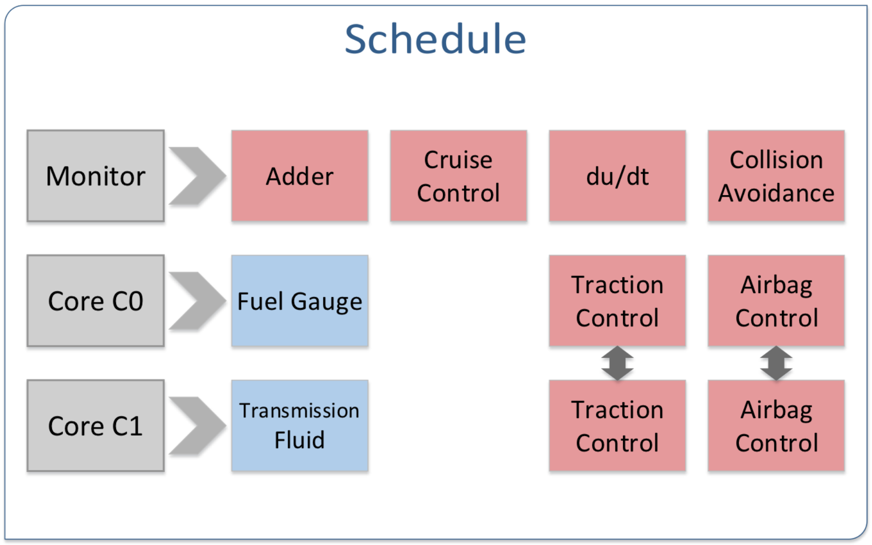

Figure 13: Static Schedule Used in Final Implementation

5.8 Imperas OVP Simulation

The final stage of the design process was to use Imperas OVP to simulate the execution of tasks in the control loop using the fingerprinting approach. The team initially simulated each block individually on the monitor, and was successful in doing so, thus proving that the way the team modified the Embedded Coder output C code was correct. Difficulties arose however, when the entire system was simulated at once. The team made the mistake of declaring the shared data within one processor core's stack, rather than in a global location. This created permission right conflicts when passing pointers to other cores during fingerprinting, thus preventing the system from properly operating. By the time this mistake was found, the team had run out of time to make any changes to the system, and therefore couldn't run a complete simulation.

At the conclusion of the project, the team has almost finished preparing the code for testing in the Imperas environment. Unfortunately, debugging and repeating steps in our design process prevented the group from completing any concrete tests with execution fingerprinting on the control loop. The work accomplished will be of value, however, to Professor Meyer and Mr. Caplan, who may wish to perform testing themselves or allow future design project groups the opportunity to do so. Documentation is currently being prepared which will help familiarize future groups with the work the team did. Through the team's successes in some areas, and failures in others, the team has made valuable inroads in the area of applying the fingerprinting method to realistic test cases.

6 Plans for Future Work

While much of the work in preparing the control loop for testing is complete, it will be up to future design project groups to simulate the loop in the Imperas environment and record quantitative results during testing. Ultimately the goal of the overall project is to validate the effectiveness and improved efficiency of fingerprinting over known lockstep benchmarks. This can only be done by performing rigorous tests to evaluate how the fingerprinting method operates on the team's sample embedded control loop. The results of this testing will provide Professor Meyer and Mr. Caplan with critical data and evidence for their research on the potential of execution fingerprinting for mixed-criticality systems. If the initial testing shows that the fingerprinting framework is indeed functioning as expected, additional control loops and test cases should be produced to extend the research and further validate the initial expectation that execution fingerprinting has a role to play in the automotive, aviation and medical fields, among others.

7 Impact on Society and Environment

This section draws heavily on content in the team's first semester report [5].

The impact that execution fingerprinting can have on both society and the environment is both exciting and promising. A few main possibilities include growing the reliability of safety-critical systems where cost is a large factor, decreasing the need for raw materials in safety-critical electronics, reducing the cost and therefore increasing the the availability of electric vehicles, and introducing better performing safety-critical embedded systems to the average consumer.

The rail industry is a good example of one where lowering the cost of safety-oriented electronics could have a major impact. Accidents such as the one that happened just over a year ago in Lac Megantic, Quebec, are tragic not only because of their human cost, but also because they could have been avoided. There exist many electronic systems to monitor and control railway transportation in a safe and efficient manner, yet the reality is that many smaller railway companies simply don't have the financial resources to implement them. By enabling a system to do more with less, execution fingerprinting can hopefully bring even the smallest railway companies to the cutting edge of safety, and prevent the tragic from occurring.

By using execution fingerprinting over traditional lockstep redundancy, safety-critical electronics will need less processors to achieve the same results. In many cases, this can lead to a reduced usage of raw materials (including toxic heavy metals that are hazardous to persons and communities mining or disposing of them), a key environmental consideration.

With their ability to schedule more tasks in a fixed time period than lockstepped systems, fingerprinting-based systems can hopefully help reduce the cost of electric automobiles. These vehicles are extremely dependent on electronics to operate, and therefore any technological advance that would permit them to use less electronics would invariably reduce their cost. One would expect lower cost electric vehicles to gain an increasing share of the automotive market, and therefore decrease our society's dependence on fossil fuels.

Lastly, simply by replacing the lockstep systems used today in automobiles with fingerprinting-based schemes would allow these vehicles to offer more services to their users at a similar cost, thus improving the average consumer's automotive experience.

8 Report on Teamwork

The team's approach to the problem has evolved throughout the project, and the final breakdown of work was both equitable and suited to each member's strengths. Initially, all team members read the same materials to gain background knowledge on the topic. As the project wore on, however, the need arose for more specialization within the group, as the repetition of many tasks for the sake of knowledge became redundant. Greg took the lead when it came to using Simulink, meaning that his role was of the loop's chief designer. His main focus was therefore learning about the Simulink environment and interfacing with Mr. Caplan to ensure that our loop meets with his requirements. Anthony's role was to first understand the full functionality of the existing fingerprinting framework, then to integrate into it the MATLAB-generated C code. Alex, meanwhile has been the team's documentation specialist. He has worked on the progress reports, poster and final report as well as assisted Anthony and Greg with other miscellaneous tasks.

9 Conclusion

There is a constant expectation in our society that electronics must be better, faster, cheaper, and more reliable than their predecessors. The promise of Moore's Law has lead to systems that do today what yesterday we thought impossible. In safety-critical systems, however, the need for reliability has forced designers to sacrifice this performance, and rightfully so. Execution fingerprinting, at its core, is a tool that if successfully implemented, will drastically reduce this current cap on safety-critical system performance. By developing a representative automotive control loop, our team intends to bring the fingerprinting method one step closer to adoption by industry.

References

[1] D. J. Sorin, “Fault tolerant computer architecture,” *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–104, Jan. 2009. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00192ED1V01Y200904CAC005

[2] W. Ribbens, *Understanding Automotive Electronics*, 6th ed. Amsterdam ; Boston: Newnes, Dec. 2002.

[3] Brett H. Meyer, Jonah Caplan, Georgi Z. Kostadinov, and Mojing Liu, “Rapid, tunable error detection with execution fingerprinting,” *SAE International*, Sep. 2013.

[4] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, “Fingerprinting: Bounding soft-error detection latency and bandwidth,” in *In Proc. of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS*, 2004, pp. 224–234.

[5] Anthony Delage, Alexander Lunan and Gregory Williamson, “Design Project I Final Report; Developing a Representative Automotive Embedded Control Loop,” 2014.

[6] R. Baumann, “Soft errors in advanced computer systems,” *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, May 2005.

[7] Brett H. Meyer, Nishant George, Benton Calhoun, John Lach, and Kevin Skadron, “Reducing the cost of redundant execution in safety-critical systems using relaxed dedication,” Mar. 2011.

[8] J. Caplan, M. Mera, P. Milder, and B. Meyer, “Trade-offs in execution signature compression for reliable processor systems,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, Mar. 2014, pp. 1–6.

[9] *Imperas Tools Overview*, 2013.

[10] Jean J. Labrosse, *MicroC/OS-II*, 2002.