Fault-tolerant Embedded Control Systems for Unreliable Hardware

Dip Goswami[1], Daniel Müller-Gritschneder[2], Twan Basten[1], Ulf Schlichtmann[2], Samarjit Chakraborty[3]
[1]Eindhoven University of Technology, Netherlands, [2]Institute for Electronic Design Automation, TU Muenchen, Germany
[3]Institute for Real-time Systems, TU Muenchen, Germany
d.goswami@tue.nl, daniel.mueller@tum.de, a.a.basten@tue.nl, ulf.schlichtmann@tum.de, samarjit.chakraborty@tum.de

***Abstract*— Past years have seen intense research on reliability techniques for error detection recovery at various levels ranging from circuit level up to architectural level or even software level. In such scenarios, affordable techniques for error correction usually imply a timing penalty, e.g., check-pointing usually requires to repeat some part of the computation, which imposes a higher computation time. This can be problematic for real-time embedded control applications especially in the presence of intermittent hardware faults, for which delays due to re-computation are repeatedly encountered with high repetition rate. In this work, we investigate a setting where the control loops are executed on an unreliable embedded platform that may suffer from such intermittent faults. First, we characterize the impact of intermittent faults in the hardware by using an intermittent bit-flip fault model and RTL level error effect simulation. Subsequently, we look at novel fault-tolerant control algorithms that guarantee stability of the loops even in presence of repeating timing errors due to the error recovery of the unreliable hardware.**

***Index Terms*— Fault modeling, fault-tolerant control law**

I. Introduction

The number of embedded electronic components in different systems, such as automotive systems, avionics, industrial machines, etc., is constantly rising. In many domains, one major category of applications that execute in such platforms is feedback control loops. These applications usually offer safety critical functionality, which is required to be operational under all circumstances. A high-level view of such systems is shown in Figure 1. The design of such systems involves design of the control law as well as the mapping and scheduling of its implementation, e.g. as a piece of SW executed on compute units. Such design becomes particularly challenging if correct functionality must be assured in presence of *fault*s. In general, faults can happen at various layers and, the nature of a fault and fault mitigation policies vary widely. Fault-tolerant design of such systems has been an active research direction for more than a decade from various perspectives.

In work stemming from the control systems community [1][2][3], a *fault* is defined as any kind of malfunction or degradation in the *plant*, *sensors* or *actuators* that can lead to a reduction in performance or loss of important functionality (i.e., safety). Such faults are mitigated by designing an appropriate control law that runs on a platform building on a *reliable* hardware architecture. Such fault-tolerant control laws are designed for robust behavior of the feedback loops under faults, guaranteeing safe operation.

978-1-4799-4833-8/14/$31.00 ©2014 IEEE

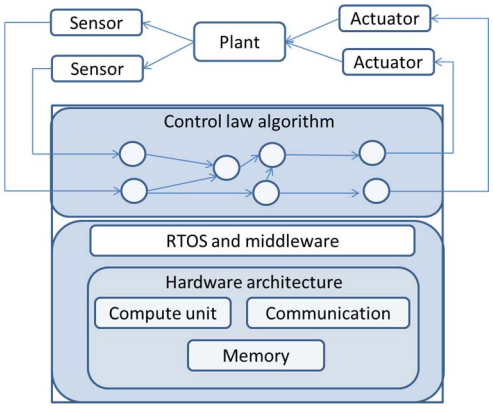

Figure 1: Embedded control systems

On the other hand, work originating from the embedded and real-time systems community is concerned with faults in the hardware architectures [4][5][6][7]. Faults include malfunctioning of resources temporarily (i.e., *transient faults*) or permanently (*permanent faults*). Such fault situations are dealt with by a fault-tolerant middleware. For example, the middleware can (re)map and (re)schedule the safety critical application tasks into an operational compute platform and maintain safe operation. Usually this requires to add error detection and recovery mechanisms to the control system. Error recovery techniques usually guarantee that no data errors may propagate to the actuator inside the control feedback loop by applying different levels of redundancy [4][5][6]. Costly variants, such as triple modular redundancy (TMR) can correct an error by voting over the outputs. Other techniques require to rollback and re-compute, e.g. a single lock-step processor only detects that the checker core computes a value different from the master core but it is not possible to identify with full certainty which is the correct core. One needs to stop the computation and roll back to a check point. This results in timing penalties and a delayed control loop possibly leading to stability and performance related issues.

This is especially critical if we consider an unreliable hardware architecture, which suffers from intermittent faults caused by wear our effects, e.g, due to aging while we assume that the physical system (i.e., plant, sensors, actuators) is reliable. Unlike the results in [4][5][6][7] as described above, we show the potential of another possible design solution -- a fault-tolerant control algorithm for such unreliable hardware.

In particular, we first characterize intermittent hardware faults and their effect on the execution of the control feedback loop. We show that intermittent faults can lead to a high number

of control faults of the control loop within a small time frame. In real-world scenarios, this could lead to instability before the system can be moved to a safe state and shut down. We propose a linear matrix inequality (LMI) based control law taking into account the derived fault model to mitigate the impact of these faults. This technique guarantees safe operation as long as the faults are followed by a minimal recovery period.

The remainder of the paper is structured as follows. Section II shows the basics of control systems, Section III the fault model and fault effect simulation with injection, Section IV addresses the fault-tolerant control design, Section V shows experimental results for a case study of a cruise control system while Section VI concludes the paper.

II. Control Applications

In this work, we consider a linear time invariant (LTI) system modeled by the difference equation of the following form.

$$x(k+1) = Ax(k) + Bu(k) \quad (1)$$

where A,B are the system matrices and x(k) is the plant state. We assume that the *sampling period* of the control loop is *h*. The plant state x(k) is periodically read by the sensors and based on the plant state x(k), the control input u(k) is computed. Further, we consider a state-feedback controller of the following form.

$$u(k) = Kx(k) \quad (2)$$

where K is the state feedback gain. The computed input u(k) is applied to the plant by the actuator. Sensing, computing and actuating takes *negligible* time compared to the sampling period h. In an ideal scenario without faults, this gives us a closed-loop system.

$$x(k+1) = (A+BK)x(k). \quad (3)$$

It is possible to design controller gain K using any traditional design method [8] such that the high-level system requirements are met.

III. Fault Modeling and Simulation

Faults in the hardware can usually be classified into three categories: Transient, intermittent and permanent faults. Transient faults are random single events, which lead to an error, such as a particle strike on a memory element. Permanent faults are deterministic, e.g., a stuck-at error at a certain circuit location. In contrast, intermittent errors appear non-deterministically but repeatedly at the same location under certain operating conditions. Such errors occur usually due to wear-out caused by semiconductor degradation effects such as NBTI, HCI or TDDB [9][10].

This work focuses on tolerance towards intermittent faults. Intermittent faults usually occur before a fault manifests permanently. The presented control design can be used to keep the system stable in the sudden presence of intermittent faults until it reaches a safe state. When a safe state is reached the system can be shut down and repaired before the fault turns permanent. In the following, we derive a statistical model for intermittent faults to test the control design.

A. Intermittent Hardware Fault Model

In contrast to transient and permanent faults, which have well established fault models, modeling intermittent faults is not yet fully agreed upon. In this work *intermittent bit-flip* is used as fault model, which was proposed with other fault models in [9].

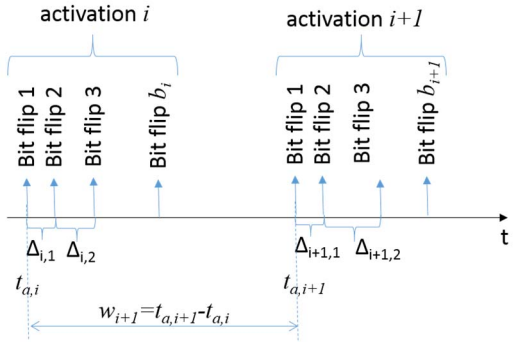

Figure 2: Intermittent bit-flip fault model

In contrast to bit flips injected in transient fault models, *intermittent bit flips* are activated in bursts as shown in Fig. 2. Statistical parameters are the burst lengths b_i (number of bit flips) and the duration between injection of each bit flip $\Delta_{i,j}$. Another important parameter is the time between activations w_i. Studies on wear out failures found that aging failures follow a log-normal or Weibull distribution [9] with CDF:

$$CDF_{WB,w}(w,\alpha,\beta) = 1 - e^{(\frac{w}{\beta})^{\alpha}}$$

The parameter α depends on the wear out rate of the system and determines together with β the mean time between two activations. In this work, we look at intermittent faults only in a small time window of a few control periods. We expect no wear out during this time such that α=1. The distribution of *w,* from which we draw random samples w_i, becomes exponential:

$$CDF_{EXP,w}(w,\beta) = 1 - e^{\frac{w}{\beta}} \quad (4)$$

The parameter β=mean(w_i) is the mean time between activation (MTBA) of the intermittent fault.

B. Simulation of Intermittent Faults

Not all intermittent faults in a computing element such as a RISC processor have impact on the running software but the effect depends heavily on the location of the fault and the executed application [10]. For evaluation, the proposed fault-tolerant controller is implemented in software and emulated on a register transfer level (RTL) model of a micro-processor. The RTL model is wrapped into a test bench that models the environment of the system and supplies sensor values based on the computed actuator values (so-called model-in-the-loop) simulation. Without protection and recovery mechanism, the hardware faults could propagate as data errors to the actuator. To protect the system against actuator value corruption, dual-redundancy is used. The actuator values computed by a master core are compared against the values computed by a checker core. An activation of the fault in hardware manifests itself as a fault for the control software, if the master's and checker's actuator commands have a different value or are sent at different points in time (*mismatch*); otherwise the fault is masked because

it has no effect on the control output. The important aspect of intermittent faults is that they usually appear at the same location repeatedly. The probability that a hardware fault is masked is usually quite high for a typical RISC processor. In contrast, the probability that an intermittent hardware fault is masked under the condition that at the same location it was not masked before, is significantly lower. We estimate this conditional probability by Monte-Carlo simulation. We inject faults (burst of bit flips) R times at the same location l (register and bit number), with random b_i, $\Delta_{i,j}$ and activation time $t_{a,i}$. These R simulations are repeated S times for random locations l. From this simulation, we can extract the probability P_m that any intermittent error is masked as:

$$P_m = 1 - (\#Mismatch / RS)$$

Additionally, we extract the conditional probability P_{mc} that a hardware fault is masked under the condition that for l^* in L^* at least one fault was not masked for the R simulations:

$$P_{mc} = 1 - \frac{1}{|L^*|} \sum_{l^* \in L^*} \frac{\#Mismatch(l^*) - 1}{R - 1}$$
$$\text{with } L^* = \{l | \#Mismatch(l^*) > 0\} \quad (5)$$

C. Intermittent Control Fault Model

From the intermittent hardware fault model and dual redundancy setup, we derive a single intermittent control fault model. In particular, we characterize the single intermittent control faults by the following: *at most one control fault occurs at a time and the faulty execution is followed by a minimum number N of non-faulty executions of the control loop* (see Figure 2). We refer to the N samples following after the faulty execution of the control loop as *fault recovery period*.

Such characterization is possible using the statistics introduced above. The probability for a control fault to be followed by a fault-free N sample recovery period given a control period h can be roughly approximated by:

$$P_r(N) = 1 - CDF_{EXP,w}(Nh, \beta) + CDF_{EXP,w}(Nh, \beta) * P_{mc} \quad (6)$$

The first term describes the probability, that the time to the next activation is higher than the recovery time, while the second term describes the probability that the time to the activation of the next burst is lower than the recovery time but the error is masked. For controller design, we need to consider P_{mc} because the recovery period happens after a control fault, for which, at the same location, an intermittent hardware fault activation was not masked before.

The length N of the possible recovery period must be chosen such that this probability $P_r(N)$ is sufficiently large for the given intermittent hardware fault model.

IV. Fault-Tolerant Controller Design

In non-faulty/ideal executions, the feedback gain K is given as shown in Equation (2). As illustrated in Figure 3 at the occurrence of control faults, the compute platform does not execute the control task and therefore, the old control input based on the older feedback signal is applied to the plant. Therefore, in a faulty execution with a single intermittent control fault, the control law (2) becomes

$$u(k) = Kx(k-1). \quad (7)$$

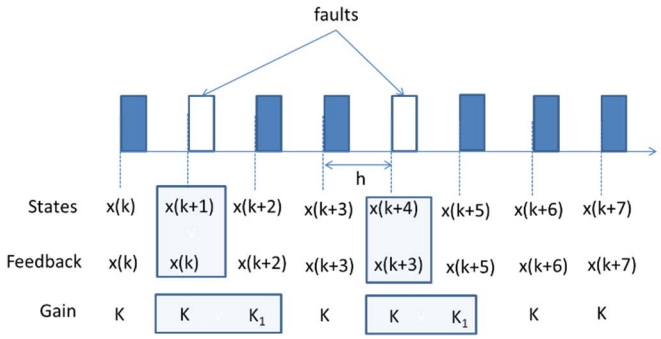

Figure 3: Single intermittent fault with N=2.

As can be noted in Figure 3 due to a faulty execution, the closed-loop system becomes,

$$x(k+2) = Ax(k+1)+BKx(k).$$

Note that there is no update available at sampling instant k+1. Further, using Equation (3) in the above, we have

$$x(k+2) = [A^2 + (AB + B)K]x(k). \quad (8)$$

It is important to note that the occurrence of a fault is *unpredictable* and it leads to a system dynamics given by Equation (8). Therefore, with a faulty hardware architecture, the closed loop system essentially switches between subsystems given by Equations (8) and (3). Essentially, the design must make sure that such switching does not lead to an unstable system, i.e., to a safety failure. If we design the feedback gain K in such a way that *arbitrary* switching between (8) and (3) is stable, the closed loop system will be stable under *any* single intermittent fault. Thus, the gain K should be chosen such that a common quadratic Lyapunov function (CQLF) exists between systems (8) and (3). In general, such designs that allow a significant flexibility are overly conservative and, often, unable to meet the performance requirements in safety critical applications. Since the system runs without any fault most of the time, it is important to ensure good performance under normal operation. At the occurrence of a fault, it is important to avoid instability while it is acceptable to provide a degraded performance temporarily during the fault recovery period.

In this work, we emphasize exploiting the additional information available to the designer from the fault modeling described in Section III to meet the above requirements. That is, we know that a faulty execution will be followed by a minimum of N non-faulty executions. The idea is to design several *fault recovery gains* for N post-fault control inputs (or executions) such that the overall closed loop system recovers ensuring stability. With such design, we ensure that (i) the closed-loop system provides the usual performance in the no-fault scenario and that (ii) in the faulty scenario, the stability is guaranteed while performance can potentially degrade in this short period. For a given fault recovery period N, the gain K and fault recovery gains are designed to stablitize the *switching* between the following two systems: (S1) *the system starting from a faulty execution followed by N non-faulty executions* and (S2) the system given by Equation (3). The controller gains are designed such that a CQLF exists between systems (S1) and (S2). We compute such gains by an LMI formulation (details skipped because of space constraints). Figure 3 shows an example with

CPU frequency	10 MHz
Control period h	10 ms
Intermittent fault model parameters	
b_i	Drawn from uniform dist. in [1 9]
$\Delta_{i,j}$	Drawn from uniform dist. in [100ns 1100ns]
w_i	Drawn from exp. dist.(4) with MTBA β=10ms
RT level fault simulation	
Nr. simulations	350 (R=5, S=70)
Simulation time	~ 15min per fault scenario (4s of simulated time)
Estimated P_m	0.949
Estimated P_{mc}	0.722
Estimated P_r(N=2)	0.825

Table 1: Experimental setup and system parameters

fault recovery period N=2 where K_1 is the fault recovery gain. In this example, K and K_1 should be designed to stabilize a system starting from *a fault followed by two non-faulty executions*. For higher N, e.g., N=4, the fault recovery gains have to be applied to (N-1)=3 executions after a faulty execution.

V. Experimental Results

We consider a simple cruise control application with three states $x_1(k)$, $x_2(k)$ and $x_3(k)$. With sampling period h=10ms, we have the following discrete-time system matrices in Eq. (1):

$$A = \begin{bmatrix} 1.00 & 0.01 & 0.00 \\ -0.0003 & 0.9997 & 0.01 \\ -0.0604 & -0.0531 & 0.9974 \end{bmatrix}, B = \begin{bmatrix} 0.0001 \\ 0.0001 \\ 0.0247 \end{bmatrix}.$$

We consider the braking scenario where the controller has to bring back the speed (i.e., $x_1(k)$) to *zero* as soon as possible. As a performance index we consider,

$$F = \sum_k x_1^2(k) \ , \qquad (6)$$

which should be minimized for better performance. We implement the control system with two OpenRISC processors (opencores.org). The core is simulated with a SystemC RTL model. The control application is implemented with the FreeRTOS scheduler (freertos.org). The remaining system is modeled at transaction level using SystemC/TLM for fast simulation including the plant to obtain sensor values. The simulation setup and results are given in Table 1.

Case I: In this case, we use a feedback gain K such that switching (due to any arbitrary sequence of single intermittent faults) between systems (7) and (3) is stable. Therefore, we apply the same gain in all sampling intervals. With such design, we obtain the feedback gain K = [-6.6201 -16.3914 -24.1122]. Clearly, we do not need a specific fault characterization for such designs.

Case II: In this case, we design the gains for the specific fault characterization N=2. In this case, we use the feedback gain K= [-2827.4 -575.9 -33. 3] which is computed using *pole placement*. Of course, we can use other techniques such as LQR. Just after the faulty execution, we use a fault recovery gain K_1=[-4090.8 -730.3 -46.5] which is designed to ensure stablity after the faulty execution.

Figure 4 shows a comparison between the two possible fault-tolerant designs for 10s of simulated time applying the intermittent control fault model with the parameters of Table 1. As already discussed, Case I is a conservative design leading to a worse control performance with F= 955.94. Case II is offering better control performance with F=33.43. Clearly, Case II is superior to Case I – which is expected. We repeated the simulation for 1000 different scenarios of the intermittent fault model and both controllers were stable in all simulations.

VI. Conclusions and Acknowledgments

In this paper, we have shown a novel approach for the design of fault-tolerant embedded control systems. This research is partly supported by the ARTEMIS joint undertaking under Almarvi (641439) and the German BMBF project EffektiV (01IS13022).

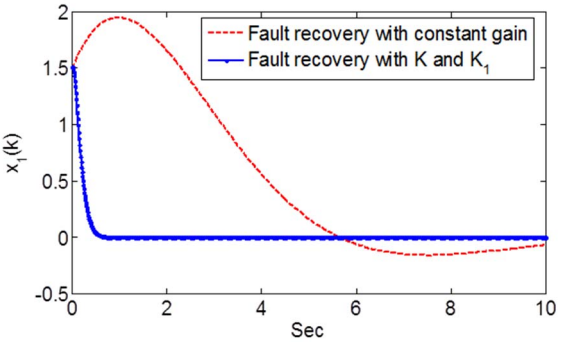

Figure 4: Control performance with the fault tolerant designs

References

[1] H. Noura; D. Theilliol; D. Sauter, "Actuator fault-tolerant control design: demonstration on a three-tank-system," *International Journal of Systems Science*, vol. 31, no. 9, pp. 1143-1155, 2000

[2] M. Staroswiecki; D. Berdjag; "A general fault tolerant linear quadratic control strategy under actuator outages," *International Journal of Systems Science*, vol. 41, no. 8, pp. 971-985, 2010

[3] P. M. Marusak; P. Tatjewski; "Actuator Fault Tolerance in Control Systems with Predictive Constrained Set-Point Optimizers," *Applied Mathematics and Computer Science*, vol. 18, no. 8, pp. 539-552, 2008

[4] M. Baleani; A. Ferrari; L. Mangeruca; A. Sangiovanni-Vincentelli; M. Peri; S. Pezzini. "Fault-tolerant platforms for automotive safety-critical application," CASES, 2003.

[5] J. Kim; G. Bhatia; R. Rajkumar; M. Jochim. "SAFER: System-level Architecture for Failure Evasion in Real-time Applications," *IEEE RTSS*, 2012

[6] P. Sinha; "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives," *Reliability Engineering & System Safety*, vol. 96, no. 10, 2011

[7] C. Pinello; L. P. Carloni; A. L. Sangiovanni-Vincentelli; "Fault-tolerant Distributed Deployment of Embedded Control Software," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 906-919, 2008

[8] B. C. Kuo; "Automatic control systems," Holt, Rinehart and Winston, 1980

[9] L. Rashid; K. Pattabiraman; S. Gopalakrishnan; "Intermittent Hardware Errors Recovery: Modeling and Evaluation," *QEST, 2012*

[10] J. Gracia-Moran; J.C. Baraza-Calvo; D. Gil-Tomas; L.J Saiz-Adalid; P.J. Gil-Vicente; "Effects of Intermittent Faults on the Reliability of a Reduced Instruction Set Computing (RISC) Microprocessor," *IEEE Transactions on Reliability*, vol. 63, no.1, pp.144-153, 2014