Embedded Control Systems Design/Learning from failure

An engineer has to keep in mind that every device he or she creates will eventually fail. Catastrophic failure should be avoided by all means since this could result in injury or even death. Some main causes of engineering failures are insufficient knowledge, underestimation of influence and ignorance or negligence.

In this chapter some examples of catastrophic failure are given. By studying these examples the engineer can learn how to avoid failure in his or her designs.

Contents

Examples of engineering failures

Patriot

On February 25, 1991, during the Gulf War, an American Patriot Missile in Dharan, Saudi Arabia, failed to intercept an incoming Iraqi Scud missile [1] (http://en.wikipedia.org/wiki/MIM-104_Patriot#Failure_at_Dhahran). The result of this failure was the death of 28 US soldiers.

Problem

The cause of the problem is located in the computer responsible for the tracking calculations of the incoming Scud missile. This computer keeps time as an integer value in units of tenths of a second, and stores it in registers of 24 bits long. Afterwards, this value is multiplied by a fixed-point 24-bit representation of 0.1 to obtain a 48-bit floating-point value of time in seconds.

The failure is caused by the 24-bit representation of 0.1 in base 2. The base 2 representation of 0.1 is nonterminating: 0.0001100110011001100110011001100.... The 24 bit register in the Patriot stored instead 0.000110011001100110011001100 introducing an error of 0.00000000000000000000000011001100... binary, or about 0.000000095 decimal. This means the accuracy of the time is reduced by 0.0001%.

However, the tracking of a missile doesn't depend on the absolute clock-time, but rather on the time that elapsed between two different radar pulses. Since this time difference is very small, the error of 0.0001% is truly insignificant. This doesn't explain the deviation of the Patriot missile, but there is a second problem.

The software used by the computer was written in assembly language 20 years ago. It was updated several times to cope with the high speed of ballistic missiles, for which the system was not originally designed. One of these updates contained a subroutine to convert clock-time more accurately into floating-point. However, the subroutine was not inserted at every point where it was needed. Hence, with a less accurate truncated time of one radar pulse being subtracted from a more accurate time of another radar pulse, the error no longer cancelled.

At the time the Scud missile arrived, the computer had been active 100 hours, so the calculated elapsed time had an error of 0.000000095×100×60×60×10=0.34 seconds. A Scud travels at about 1676 meters per second, and so travels more than half a kilometer in this error time. This was far enough for the incoming Scud to be outside the "range gate" of the Patriot.

Solution

The solution to this problem was already suggested before the accident. Increase the accuracy of the time representation, by the use of more bits. However, this solution avoids the problem from happening, but doesn't solve it. If the computer is activated for years without rebooting, the same failure can still occur.

There is a better approach. Simply measure the relative time between the two radar pulses. The clock should always be reset at the first pulse and stopped at the second pulse. This avoids the subtraction of two almost identical big numbers, and its numerical problem.

Lesson

- An algorithm where big numbers are subtracted from each other should be rewritten to avoid this loss of numerical accuracy.
- It's important to analyze the number of bits used to represent physical parameters, i.e. time, in the design.
- The system designer needs very detailed specifications of all the components he uses.

Mars Pathfinder

On July 4, 1997, the Mars Pathfinder successfully set foot on the Martian surface. However, a few days after the start of its mission, the vehicle started losing information due to several system resets. The cause of this problem was located in the embedded systems kernel used in the Pathfinder.

Problem

The tasks of the spacecraft are executed as threads with priorities, reflecting the urgency of each task. There are 3 types of threads:

- high priority threads, i.e. the bus management task.

- medium priority threads, i.e. the communications task.
- low priority threads, i.e. the meteorological data gathering task.

During execution of the bus management task, it will move data in and out of the information bus (the bus that connects all the components of the spacecraft). Access to this bus is synchronized with mutexes. The meteorological data gathering task uses the information bus to publish its data. When publishing, it acquires a mutex, writes to the bus, and releases the mutex. Using the mutexes guarantees that each bus thread waits until the previous one is finished.

Normally, the threads system with priorities and the bus management with mutexes works fine. However, a problem known as priority inversion can occur. Assume that a medium priority communications task is scheduled while the high priority bus management task is blocked waiting for the low priority meteorological data thread. The communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked bus management task from running. Since the communications task is a long-running task, a watchdog timer will eventually go off, noticing that the data bus task had not been executed for some time. The watchdog task will conclude that something had gone wrong, and initiate a total system reset. This reset is comparable to a home computer reboot; it clears the registers so that the system can restart without errors.

Solution

There are several solutions available for this problem. One of these is called priority inheritance. The meteorological thread will inherit the priority of the bus management task, causing it be scheduled with higher priority than the communications task.

Fortunately, VxWorks contains a C language interpreter intended to allow developers to type in C expressions and functions to be executed on the fly during system debugging. After uploading the right program to change the software of the spacecraft, the problem was solved.

It’s important to realize that priority inheritance is not the best solution to this problem. In complex systems, with dozens of bus tasks interacting, all the tasks will eventually obtain highest priority, and the problem remains. However, in case of the Pathfinder, it sufficed.

Lesson

- Coordinate tasks by the use of priorities is not the best approach. This indirect way gives the scheduler all responsibility. It's better to include the coordination in the system itself [2] (http://en.wikibooks.org/wiki/Embedded_Control_Systems_Design/Asynchronous_activities).
- It is important to provide support to intervene when unforeseen errors occur.
  - the watchdog: every ECS should contain a watchdog task, because a system block can never be excluded.
  - debugging facilities in the field: without the ability to modify the system in the field, the problem could not have been corrected.
- It is not possible to test every situation completely. If timing (coordination) is important, the amount of bytes is important,..., debugging can always be necessary.
- Good theory is important: the theoretical solution for this problem was presented in a paper many years ago.
- Correctness is important: the engineer's initial analysis that "the data bus task executes very frequently and is time-critical -- we shouldn't spend the extra time in it to perform priority inheritance" was wrong. It is precisely in such time critical and important situations where correctness is very important, even at some additional performance cost.

Ariane 5 Flight 501

TheAriane 5, Flight 501, was launched on June 4, 1996 and was the first unsuccessful European test flight. Due to a malfunction in the control software, the rocket veered off its flight path 37 seconds after launch and was destroyed by its automated self-destruct system when high aerodynamic forces caused the core of the vehicle to disintegrate. It is one of the most infamous computer bugs in history. What could be the reason behind the sudden change in flight path of the first flight of the Ariane 5? Certainly, reusing similar software code as in the previous Ariane 4 project without simulating was a big mistake.

Problem

An operand error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time. The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably

higher horizontal velocity values. An internal software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in the operand error, causing the on-board computer to send diagnostic data instead of real flight data. This diagnostic data commanded sudden nozzle deflections and induced a high angle of attack. (for more detailed information, see Report paper Ariane5 (htt p://www.cs.unibo.it/~laneve/papers/ariane5rep.html))

Solution

A simulation of the whole trajectory (static code analysis) should have pointed out the overflow error immediately. This step should have been performed at the stage of architectural design or implementation (prototyping) within the design process.

Lesson

This example illustrates the dramatic consequences of a data overflow software failure, whereby the onboard computer only received diagnostic information instead of realistic flight data. Although there was a back-up system of the failing part apparent in the Ariane5, failure could not be prevented because the backup system used the same software code. Some lessons came forward after the disaster with flight 501 happened:

- Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform *complete* and closed-loop testing of the *total* system: Although a (software) subsystem worked perfectly on a previous embedded control system, it can still fail as a subsystem in a new embedded system, i.e.: Always question the *reusability* of a subsystem. In this case software failed on the Ariane5 whereas it worked very well on the previous Ariane4 rocket! After the accident, a static code analysis painfully showed the overflow.

See the #Therac-25 for another failure due to reusing software.

- A sensor has to return reliable data in all circumstances: In this case, the IRS (Inertial Reference System) returned diagnostic data instead of real flight data because of a data overflow of a 16 bit signed integer value. In some cases it should be better that the sensor returns nothing at all (or just a warning sign) instead of unreliable data.

- The high risks associated with complex computing systems took the attention of the general public, politicians, and executives, resulting in increased support for research on ensuring the reliability of safety-critical systems. The subsequent automated analysis of the Ariane code was the first example of large-scale static code analysis by abstract interpretation.

Note: *In practice there should always be a trade off between safety, economic, (environmental) ,... aspects. For example: In the above example the sensor is a critical component to safely put the astronauts back on the planet, whereas a sensor as part of a conveyer belt system is much less life-threatening. Typically, safety is of most priority at the aviation, aerospace and nuclear sector!*

The Space Shuttle Challenger

On his 10th flight into space, the Space shuttle Challenger disintegrated 73 seconds after take off. 7 astronauts died. The cause of this failure was not only mechanical but also psychological, the so called groupthink!

Problem

Officials of the NASA team decided to launch the spacecraft while ignoring admonitions of the engineers with a sense of invulnerability, given the successful history of NASA. The team worked closely together for a long time and they were under intense pressure to launch the shuttle without further delay that they all collectively went along with the launch decision. Engineers however knew that the rubber sealing (an O-ring) of the propellant rocket wasn't functioning optimal under freezing temperatures. The cold temperature on the 28th Januar 1986, causing a leakage of heat flow along the O-ring therefore was the avoidable origin of the death of 7 human beings.

Lesson

This example shows a hardware failure mode: the effect of temperature on the deformation of material. All components of a system can work perfectly together in most of the cases, but fail in bad environmental conditions, such as the freezing conditions in this case.
An other important conclusion comes forward. A bad design isn't always the only cause of failures. Also psychological aspects matters. This is where the engineer

has to be critical in all circumstances and not collapse under peer pressure. Especially at the stage of usage of an embedded control system, operators have to check whether the environmental conditions fall within the design characteristics before starting up the system. Failures because of such psychological aspects are often hard to overcome and can be seen as an operational failure cause.

Corrosion disasters

Corrosion is often a hard to fight *hardware failure mode*. Many times disasters happen when materials collapse because of corrosion. Two of them are given below:

- Sinking of the Erika:

Erika was the name of a tanker built in 1975 and sank 70 km from the coast of France on December 13, 1999.Tons of heavy fuel oil ended up in middle of the sea, causing one of the greatest environmental disasters in the world. The *economic consequences* of the incident have been dramatically felt across the region: a drop in the income from tourism, loss of income from fishing, and a ban on the trade of sea products including oysters and crabs have added to the discomfort.

Corrosion problems had been apparent on the Erika since at least 1994. In addition, there were numerous deficiencies in her firefighting and inert gas systems, causing explosion risks. Lloyd's List reported that severe corrosion had been discovered just weeks before the incident. However, no immediate remedial action had been taken.

- The Aloha incident:

On 28 April 1988, the Aloha Boeing 737 aircraft serving the flight suffered extensive damage after an explosive decompression in flight, but was able to land safely at Kahului Airport on Maui. One hostess was the sole fatality as she was blown out of the airplane, while another 65 passengers and crew were injured. In the aircraft, evidence was found of multiple site fatigue damage leading to structural failure. The National Transportation Safety Board investigation report issued in 1989 attributed the incident to the failure of the operators maintenance program to detect corrosion damage. Because older airplanes are commonly not destroyed

and will probably end up in service with another operator, safety issues regarding aging aircraft need to be well understood by the engineer and safety programs need to be applied on a consistent basis.

Lesson

Aging effects and corrosion of materials are not visible while the product is in test phase but have to be considered unavoidably! While designing large embedded control systems, engineers should have a long term vision and therefore look at the different Points Of View in the design process (http://en.wikibooks.org/wiki/Embedded_Control_Systems_Design/Embedded_design_process) of the company. Services, maintenance and testing after sale can be solutions for large embedded systems and can prevent the company from a bad name at long term.

Case Study: The Therac-25

The Therac-25 was a computer-controlled radiation therapy machine, which severely overdosed six people between 1985 and 1987. Three of them died. A case study of the Therac-25 will reveal following design difficulties:

- Software can fail
- Testing can be difficult, but is indispensable
- Reuse of software can be dangerous
- Safety and usability are conflicting design criteria

In this case the mistake was made of reusing and trusting old software. This is typically a mistake made in high-level design.

This text will first discuss the background of the machine. In a following section some relevant accidents are described. Next, the text will focus on some software issues related to the accidents. In a final section some engineering lessons that can be learned from the accidents will be discussed.

Background

The Therac-25, produced by Atomic Energy of Canada Limited (AECL) and CGR, was a medical linear accelerator. Such accelerators produce high-energy beams in order to destroy tumors with minimal impact on the surrounding healthy tissue. The Therac-6, capable of producing X-rays only, and the Therac-20, a dual-mode (X-rays or electron beams) accelerator, were predecessors to the Therac-25. Both machines were versions of older CGR machines, which already had a history of clinical use, augmented with a minicomputer. They both had limited software functionality: the computer merely added convenience to the existing hardware. Hardware safety features from older machines were retained.

A few years later, AECL developed a new acceleration concept (“double pass”) which required less space, was easier to use and more economical to produce. This concept was used in the Therac-25 dual-mode linear accelerator.

In each mode (electron or X-ray), the necessary tools to create the proper beam were attached to the turntable, which rotated the equipment into the beam. Correct operation of the Therac-25 thus was dependent on the position of the turntable. In the case of the X-ray mode, a beam flattener, mounted to the turntable, was used to create a uniform treatment field. In order to produce an output comparable with the electron mode output, this required a high input dose rate. A wrong position of the turntable lead to the wrong position of the flattener and resulted in a high output dose. Obviously, this is a grave hazard when using dual-mode machines.

Although the Therac-25 was controlled by the same computer as the Therac-6 and Therac-20, there were some differences. The hardware of the Therac-6 and Therac-20 was capable of standing alone (for instance, mechanical interlocks which ensured safe operation). However, the software of the Therac-25 had more responsibility for maintaining safety: the machine relied much more on the software instead of hardware safety mechanisms. For the software design of the Therac-25, design features and modules of the Therac-6 were reused. The software also contained some Therac-20 routines.

In the Therac-25, it was the software’s task to control and check the turntable’s position. The software was not flawless though as six people received a massive overdose. This is explained more in the section Accidents.

When the software detected an error the machine could shut down in two ways:

- *Treatment suspend:* which required a machine reset to restart

- *Treatment pause:* which required a single key command to restart the machine.

When a 'treatment pause' occurred, the operator merely had to push the P-button to "proceed" and resume treatment without having to reenter treatment data. When this feature was called five times the machine would automatically suspend treatment.

Accidents

July 1985, Ontario (Canada)

About five seconds after activation of the Therac-25, the machine shut down. It indicated a 'treatment pause' and that no dose was given. The operator tried a second attempt at treatment by pushing the "P-button", but the machine kept shutting down four more times until it went into 'treatment suspend'. After the treatment, the patient complained of a burning sensation to the treatment area. In reality the patient was given a large overdose, approximately 100 times the intended dose. Normal single therapeutic doses are about 200 rads. Doses of 1000 rads can even be fatal, if delivered to the whole body.

After investigation of this accident AECL found some weaknesses and mechanical problems, but could not reproduce the malfunction that occurred. AECL then redesigned some mechanisms and altered the software to tackle these problem. After these improvements AECL claimed that "analysis of the hazard rate of the new solution indicates an improvement over the old system by at least five orders of magnitude". The hazard analysis, however, did not seem to include computer failure. Thus, more accidents occurred.

March 1986, East Texas Cancer Center (ETCC)

The operator had lots of experience with the machine and could quickly enter prescription data. She wanted to type "e" (for electron mode), but she had pressed the "x"-button (for X-ray) by mistake. To correct this, she used the "up-key" and quickly changed the mode entry. The other treatment parameters remained. To start treatment she hit the "B-key" (beam on). After a moment the machine shut down and showed the error message: "MALFUNCTION 54", which was not explained nor mentioned in the machine's manual. The machine went into ‘treatment pause’ which indicated a problem of low priority. The machine showed an underdose, so the operator hit the "P-button" to proceed treatment. Again the

machine shut down with a MALFUNCTION 54 error. The patient complained he had felt something like an electric shock and was immediately examined. The physician however suspected nothing serious. In reality the patient received an immense overdose. Real doses of 16500 to 25000 rads were estimated after the facts. Five months later, the patient died from complications of the overdose.

April 1986, East Texas Cancer Center (ETCC)

Three weeks after the first accident at ETCC, a similar one occurred. The same operator noticed an error in the mode and used the "up-key" to correct it. Again the machine showed a MALFUNCTION 54 error. This patient died from overdose three weeks after the accident. The ETTC physicist immediately took the machine out of service after this second accident and investigated the error on his own. The operator who remembered what she had done, worked with him. With much effort they were able to reproduce the MALFUNCTION 54 error. The key factor in reproducing this error was the speed of entering data: if the data were entered quickly the error occurred.

The same computer bug was present in the Therac-20. But because the Therac-20 had independent hardware protective circuits, this problem was just a nuisance.

Problem

Just a small part of the software will be studied, yet this can demonstrate the overall design flaws.

The two basic mistakes involved in the accidents are:

- poor software engineering practices
- building a machine that relies on software for safe operation

Race conditions as a result of the implementation of multitasking are possible. Following reasons state why:

- Concurrent access to shared memory is allowed
- Aside from data stored in shared variables, there is no real synchronization
- The "test" and "set" for shared variables can’t be divided into two single operations.

These race conditions played an important role in the accidents.

Software bugs for ETCC accidents

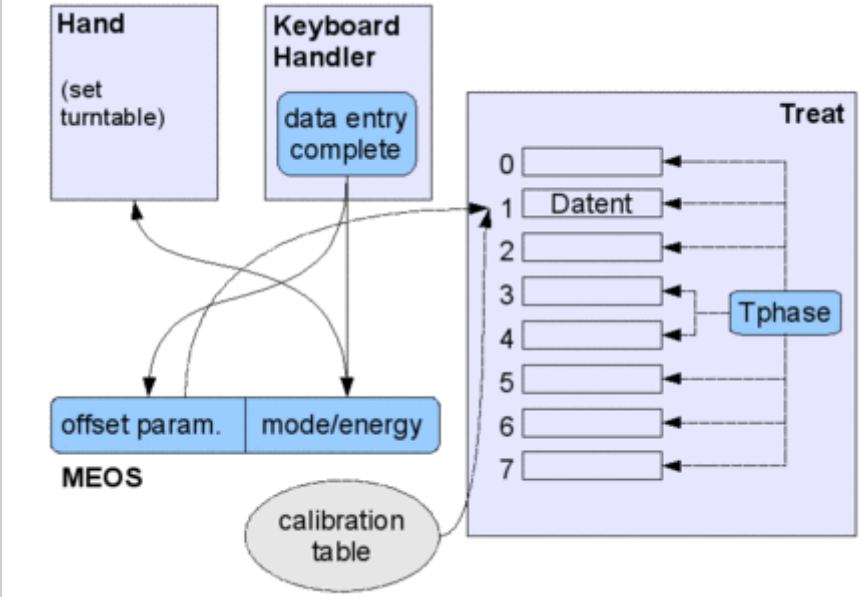

Figure 1: Tasks and subroutines responsible for the ETCC accidents (inspired by Nancy Leveson)

Figure 1 illustrates the treatment process. The treatment monitor (Treat) controls the eight treatment operation phases or subroutines according to the value of the Tphase variable. The subroutine Datent (data entry) and the Keyboard Handler task use a shared variable (data entry complete flag) to communicate with each other. Once the data entry is complete, the Keyboard Handler task changes the value of this shared variable, which denotes Datent to change the value of Tphase. The Datent subroutine exits back to Treat, which will reschedule itself in order to start the following subroutine.

Another shared variable, MEOS, contains the mode and energy level specified by the operator. MEOS exists of 2 bytes: a low-order byte to set the turntable and a high-order byte, used by Datent, to set several operating parameters.

The operator is forced to enter mode and energy level by the data entry process but he can edit the data later. A problem can occur when the operator changes the data in MEOS after the ‘data entry complete’ flag has been set by the Keyboard Handler task. Datent is then already exited and thus won’t detect the changes. However, the turntable is set in accordance to the low-order byte of MEOS and can therefore be inconsistent with the information in the high-order byte. This inconsistency apparently can’t be detected by the software.

After Datent finished setting the operational parameters, via the high-order byte information, it calls the subroutine Magnet, which sets the bending magnets of the turntable. Following pseudocode shows relevant parts of the software (taken from the paper of Nancy Leveson, p. 27):

Datent

```
if mode/energy specified then
  begin calculate table index
    repeat
      fetch parameter
      output parameter
      point to next parameter
    until all parameters set
    call Magnet
    if mode/energy changed then return
  end
if data entry is complete then set Tphase to 3
if data is not complete then
  if reset commando entered then set Tphase to 0
return
```

Magnet

```
Set bending magnet flag
repeat
  Set next magnet
  call Ptime
  if mode/energy has changed then exit
until hysteresis delay has expired
Clear bending magnet flag
return
```

Ptime

```
repeat
  if bending magnet flag is set then
    if editing taking place then
      if mode/energy has changed then exit
until hysteresis delay has expired
Clear bending magnet flag
return
```

It takes about 8 seconds to set the magnets. The subroutine Ptime is used to introduce this delay. Ptime is entered several times, because several magnets have to be set. To indicate the bending magnets are being set, a flag is initialized when Magnet is entered. This flag is cleared at the end of Ptime. When an editing request is submitted, the keyboard handler sets a shared variable which is checked by Ptime. If edits are present, Ptime will exit to Magnet, which then exits to Datent. However, this shared variable is only checked when the bending magnet flag is set. But this flag is cleared after the first execution of Ptime. So any edits performed

after the first pass through Ptime will not be noticed! In the ETTC accidents the error occurred, since the edits were made within 8 seconds. Hence Datent never detected the changes. This problem was partially solved by clearing the bending magnet flag when all magnets are set (at the end of Magnet).

Lesson

Some lessons can be learned from these accidents.

- **Do not ignore or overrely on software**
  Engineers may not assume a problem is caused by hardware. Extensive testing and safety analyses at software level is necessary, not on system level alone. Every safety-critical system should have incident analysis procedures that are applied whenever a problem is about to occur that might lead to an accident.

- **Be careful reusing software**
  The reuse of software does not imply safety because it already has been exercised extensively. Revising and even rewriting the entire software may be safer in many cases, because every new development will introduce its own (new) errors.

- **Design for the worst case**
  It is crucial that an error is detected and indicated properly (towards the operator) in order to take the right steps. A way to obtain correct information about the errors is by designing error detecting code into the software from the beginning. Self-checks, error-detection and error-handling features must be given high priority in making tradeoff decisions.

References

- Medical Devices: The Therac-25 (http://sunnyday.mit.edu/papers/therac.pdf) Nancy Leveson, University of Washington
- Engineering Disasters and Learning from Failure (http://www.matscieng.sunysb.edu/disaster/) Vasudevan Srinivasan, Gary Halada, and JQ, Department of Materials Science and Engineering, State University of New York at Stony Brook

- Corrosion disasters (http://corrosion-doctors.org/Forms/Accidents.htm)
- L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An

Approach to Real-Time Synchronization. In IEEE Transactions on Computers, vol. 39, pp. 1175–1185, Sep. 1990.

- What really happened on Mars Rover Pathfinder (http://catless.ncl.ac.uk/Risks/19.49.html#subj1)
- Round off errors and the Patriot missile (http://autarkaw.wordpress.com/2008/06/02/round-off-errors-and-the-patriot-missile/)
- Roundoff Error and the Patriot Missile (http://www.mc.edu/campus/users/travis/syllabi/381/patriot.htm)
- The Patriot Missile Failure (http://www.ima.umn.edu/~arnold/disasters/patriot.html)
- Patriot Missile Software Problem (http://www.cs.usyd.edu.au/~alum/patriot_bug.html)
- Software Engineering: Ariane 5 (http://www.vuw.ac.nz/staff/stephen_marshall/SE/Failures/SE_Ariane.html)
- Ariane 501 - Presentation of Inquiry Board report (http://www.esa.int/esaCP/Pr_33_1996_p_EN.html)