www.bezem.de

Safe Coding Techniques

How to modify your Coding Techniques for ASIL modules

Johan Bezem

Target Audience

- Developers:
  - Tasked with creating modules in a safety-related environment (ISO 26262)
  - a good understanding of the C programming language
  - a basic understanding of logic design

Program

- Introduction
- ISO 26262 and ASIL
- Hamming distance
- Why 75%
- Safe multiple error values
- Safe TRUE and FALSE values
- Sentinels
- Loose ends

Introduction

www.bezem.de

- Johan Bezem, freelance developer (1990-)
- C programming experience since 1984
- Relevant efforts:
  - Worked in multiple ASIL A – D projects since 2011
  - Created an architecture and design for an ASIL A component
  - Trained more than 20 developers on safety-related development issues

ISO 26262 and ASIL

- The ISO standard defines safety levels
- The standard remains high-level and independent from the programming language being used
- The goal is to be able to detect any unsafe event, and then to transition into a safe state
- “Everything OFF” is a safe state!

Hamming Distance

- The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different
- We consider only binary strings, 0 and 1
- Our string lengths are 8, 16 or 32 bit, maybe 64 bit in some cases

100%

- We are not concerned about transmission errors, but about data storage resilience
- If we store one bit, and that value is modified, we lose our data
- If we replicate the one bit in two or more other locations, we may be able to detect modifications

75%

- A HD of 100% gives us 2 values
- If you want to store the result of an operation, you frequently need one “OK value”, and multiple different “Error values”
- Using a 75% HD provides this flexibility

Values

www.bezem.de

- If you need values for a byte-sized variable, start with 0x00 for an ‘OK’ result
- The ‘NOK’ values (HD 6) are:
  `0x3F, 0x5F, 0x6F, 0x77, 0x7B, 0x7D, 0x7E, 0x9F, 0xAF, 0xB7, 0xBB, 0xBD, 0xBE, 0xCF, 0xD7, 0xDB, 0xDD, 0xDE, 0xE7, 0xEB, 0xED, 0xEE, 0xF3, 0xF5, 0xF6, 0xF9, 0xFA, 0xFC`
- All have an HD of 2 among each other, some have HD 4

Calculation

- The number of HD 75% values in a formula:
- $\dfrac{\prod_{i=n}^{i=n-p+1} i}{p!}$
- n is the total number of bits
- p is the number of identical bits, $n/4$

Calculation II

www.bezem.de

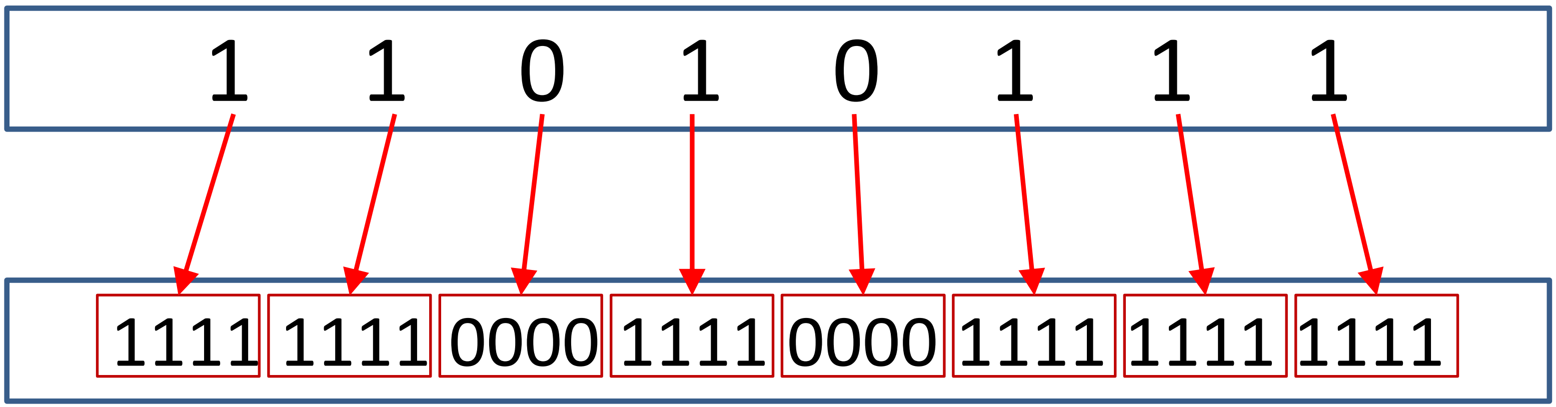
- For 8 bits (n = 8, p = 2) we get:
  8 * 7 / 2 = 28
- For 32 bits (n = 32, p = 8) we get:

$$\frac{32 * 31 * 30 * 29 * 28 * 27 * 26 * 25}{40320}$$

- Or: 10518300 different values
- It is easier to use the 28 byte values and for each bit use 4 bit instead

Spreading-out

- Take 0xD7:



- We get 0xFF0F0FFF, with the same properties for 32 bits as 0xD7 for 8 bits

TRUE and FALSE

www.bezem.de

- In C, TRUE is 0x01, FALSE is 0x00
- A HD of 1…
- For safe coding, use a HD of 6!
- Take your own values, or use mine:

```
#define SAFE_TRUE  0xF5
#define SAFE_FALSE 0x9A
```

- Be aware: Coding a test is different now!

Coding example


```
if (sbFlag == SAFE_TRUE) {
    …
} else if (sbFlag == SAFE_FALSE) {
    …
} else {
    /* System Error !! */
}
```

Sentinels

- For important error codes I use 32-bit error codes, I call ‘sentinels’
- I start with a BCD-coded date as ‘OK’ value, like 0x0712_1905 (Birthday of Gerrit Pieter Kuiper)
- I then use the 28 HD6 bytes spread-out and use XOR to obtain the ‘NOK’ values:
  `0x0712_1905 ⊻ 0xFF0F_0FFF = 0xF81D_16FA`

PST loose ends I

- The result of the processor self-test cannot be stored in RAM yet, since it is not yet tested.
- Look for an available 32-bit location, maybe in the unused registers of any peripherals of the processor
- And no matter what: Test the storage location before using it, especially if it must be stored in RAM!

PST loose ends II

- If the PST shows an error, you cannot do much: Reset and try again, or shutdown immediately
- Be aware of possible reset loops. And test your error reaction code
- If a safe-boolean is detected to have an invalid value, first make sure it is not a programming error! Then reset, and make sure you can find out the reason

Thank You!

www.bezem.de

Author:

Johan Bezem

t: +49 172 5463210

m: administration@bezem.de

LinkedIn requests welcome, use “#concept”