Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

The Patriot Missile Failure

Coding error accumulation

Jordi Bonet-Dalmau

March 16, 2022

In the course of *Embedded Systems* we have introduced fixed-point arithmetic. Now we are going to focus on the error that appears when a number is coded with a finite number of bits. If this number is multipied by another one, i.e. a counter, the result of this multiplication will have an error that is the original one multiplied by the value of the counter.

Today's exercise provides an example of the catastrofic effect that the accumulation of an originally small coding error may have in a real application.

1 Read the source of information

First of all read the report that gives information about a tragic missile failure due to arithmetic errors. A more accurate information is given in this other report. This failure took place in 1991, during the so-called Gulf War. During this war Iraq launched Scud missiles (developed by the Soviet Union) against the coalition forces led by the United States that deployed the Patriot missiles to neutralize the Scud threat. The U.S. military claimed a high effectiveness against Scuds at the time, but later analysis gives figures as low as nine percent... [for more information see, Scud and Patriot missiles].

2 Understanding fixed point coding

A binary code gives to each bit a value that is a power of 2. To code unsigned (positive) integers, the least significant bit (LSB), first from the right, is given a value of 2^0 and the nth bit to the left a value of 2^{n-1}. So, when using $b = 5$ bits, the LSB has a value of 1, the most significant bit (MSB) a value of $2^{b-1} = 16$ and the integer to be coded can take values between 0 and $2^b - 1 = 31$. To code signed (positive and negative) integers, the first bit from the left is given a value of -2^{b-1} instead of 2^{b-1}. This way, when using $b = 5$ bits, the signed integer to be coded can take values between $-2^{b-1} = -16$ and $2^{b-1} - 1 = 15$. In all cases, the total numbers of integers that can be coded is 2^b.

In the previous coding the radix point is assumed to be to the right of the LSB. We can move the radix point to the right or to the left. The bits that are to the left of the point are (positive or negatives) integers and are coded as before. The bits that are to the right of the point are fractionals with a value that is a negative power of 2: the nth bit to the right is given a value of 2^{-n}. So, when using a total of $b = 5$ bit, and $q = 2$ fractional bits to the right of the point, unsigned numbers to be coded can take values between 0 and $2^{b-q} - 2^{-q} = 7.75$, and

signed numbers to be coded can take values between $-2^{b-1-q} = -4$ and $2^{b-1-q} - 2^{-q} = 3.75$. In all cases, the total numbers of values that can be coded is 2^b, with a resolution equal to the LSB, i.e. $2^{-q} = 0.25$.

3 Exercise yourself

Task 1. Answer the following questions.

a. Which is the decimal value of '10111' (consider unsigned integer coding)?

b. Which is the 8-bit binary coding of 23 (consider unsigned integer coding)?

c. Which is the decimal value of '10111' (consider signed integer coding)?

d. Which is the 8-bit binary coding of -9 (consider signed integer coding)?

e. Repeat the previous questions considering $q = 2$ fractional bits, i.e. '101.11'.

f. Next, consider the coding of a number v that takes values $-1 \leq v < 1$. If you have $b = 4$ bits to code v, how many fractional bits q will you use? Which are the minimum and maximum values you can code? What resolution do you have?

g. Repeat the previous question considering $b = 24$ bits.

4 Approximating a real number with a finite number of bits

Consider the previous used coding with a total of $b = 5$ bits, $q = 2$ fractional bits and negative numbers to be coded: they can take values between $-2^{b-1-q} = -4$ and $2^{b-1-q} - 2^{-q} = 3.75$ with a resolution equal to the LSB, i.e. $2^{-q} = 0.25$. The complete set of numbers that we can code is: $\{-4, -3.75, -3.5, -3.25, .., -0.25, 0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, ...3.5, 3.75\}$.

What happens when we want to code the value $v = 1.2$? We can choose between code v as 1 ('001.00') or 1.25 ('001.01').

4.1 Truncation or chopping

Here we must explain how finite point arithmetic usually converts a coding with q_2 fractional bits to a coding with q_1 fractional bits when $q_2 > q_1$. It simply throws away the $q_2 - q_1$ least significant bits, with a positive contribution to the total value. So, the most precise value coded with q_2 is always truncated (chopped) to the next lower possible value coded with q_1. Obviously we have an always positive error that is less than the LSB of the q_1 coding.

We could imagine that v is coded with an infinite number of fractional bits and then we recode it taking only q fractional bits. This is what is done when choosing 1 ('001.00') to code 1.2. The following Octave code computes this **truncation error** using *floor* function.

```
% value to code
v=1.2
% total number of bits
b=5
% number of fractional bits
q=2
```

```
% in order to use dec2bin function
% that converts a decimal to a binary coded integer
% we move the radix point q places to the rigth (*2^q))
% and truncate the result to an integer (floor)
v_int=floor(v*2^q)
% the result is v_int=floor(4.8)=4

% now v_int can be coded with dec2bin using b bits
% but we have to 'imagine' the point
vq2=dec2bin(v_int,b)
% the result is '00100' (4 in decimal)
% but we should read '001.00' (1 in decimal)

% the previous operations have truncated 'v' in vq2
% to compute the decimal value of vq2
% we return the radix point q places to the left (/2^q))
vt=bin2dec(vq2)/2^q
% the result is 1

% finally we compute the truncation error in the coding
et=v-vt
% the result is 0.2
```

4.2 Rounding to the nearest

Another option is to round to the nearest value. Now we have a positive or negative error whose absolute value is less than half the LSB of the q_1 coding. The following Octave code computes this **rounding error** using *round* function.

```
v=1.2
b=5
q=2

% ... and truncate the result to the nearest integer (round)
v_int=round(v*2^q)
% the result is v_int=round(4.8)=5

vq2=dec2bin(v_int,b)
% the result is '00101' (5 in decimal)
% but we should read '001.01' (1.25 in decimal)

vr=bin2dec(vq2)/2^q
% the result is 1.25

% finally we compute the rounding error in the coding
er=v-vr
% the result is -0.05
```

5 Approximating 0.1 with a finite number of bits

Unfortunately 0.1 needs an infinite number of fractional bits to be coded in binary: the sequence '0011' is repeated from the second fractional bit. So, an error is introduced when a finite number of fractional bits are used. Although in the report 0.1 is truncated using 24 bits, we will also

consider the possibility of rounding to the nearest integer and the use of a different number of bits. This way we will see if these other possibilities could have avoided the failure.

Task 2. You can use the previously given code in Octave to answer the following questions.

a. After reading the report, do you agree with the following sentence excerpted from the text: *the value 1/10... was chopped at 24 bits after the radix point..*? Which are the parameters b and q that are used in the report?

b. Compute the error when truncating using $b = 24$ bits. This result, the one that appears in the report, will be used as a **reference**.

c. Compare this error with the one introduced when rounding using $b = 24$ bits.

d. Repeat the two previous questions for $b = \{22, 23\}$ and compare the results with the ones obtained with $b = 24$. Do you think that it is worth using 24 bits instead of 22?

e. And with registers of $b = 32$ bits?

f. How many bits do we need to make the error lower than 500 ns?

Task 3. Propose a coding esqueme that, using just 24 bits, improves the results that appear in the report. Hint: consider a different codification.

6 Accumulating the error

The report provides information about the way the *small* coding error of 0.1 is accumulated when computing the time in seconds: *... the time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds.* As a consequence, after a reset of the system, the originally *small* error accumulates while the system is running.

Apparently, the system was intented to run for a few hours after a reset: *Army officials presumed that Patriot users were not running the systems for longer than 8 hours at a time.*

Task 4. Compute the accumulated error in seconds in the following situations:

a. Truncating using $b = 24$ bits after $h = 100$ hours. This result, the one that appears in the report, will be used as a **reference**.

b. Rounding using $b = 24$ bits after $h = 100$ hours.

c. Repeat the two previous questions for $h = 6$ hours.

d. How will the previous results, $h = 100$ and $h = 6$, change with registers of $b = 32$ bits?

7 Distance error

The time in seconds was used to compute the position of the target (Scud) since the last detection from a ground-based radar. The target is expected to be inside a range gate where the Patriot looks for it. If the target is outside this expected range gate (because, e.g., it has not been accurately computed), the Patriot will fail to track it. If the travelled distance from the last detection is computed multiplying the velocity of the target by the difference between

the actual time (taken from the internal Patriot clock with its accumulated error) and the time provided by the ground-based radar (that it is supposed to be the reference clock when a reset to the Patriot has been made), then the distance error is directly the velocity of the targe multiplied by the accumulated time error.

Task 5. Compute the accumulated error in meters, considering a target velocity of 1676 m/s, in the following situations:

a. Truncating using $b = 24$ bits after $h = 100$ hours. This result, the one that appears in the report, will be used as a **reference**.

b. Rounding using $b = 24$ bits after $h = 100$ hours.

c. Repeat the two previous questions for $h = 6$ hours.

d. How will the previous results, $h = 100$ and $h = 6$, change with registers of $b = 32$ bits?

Task 6. Consider that an error of more than 100 m when computing the range gate means that the Patriot will fail to track the target. Which is the maximum operating time of the Patriot's system to avoid this failure? Consider the following situations:

a. Truncating using $b = 24$.

b. Rounding using $b = 24$.

c. And with registers of $b = 32$ bits?

8 Computing the interception point of a missile

The following are advanced tasks.

Consider a simplified not-real two dimensional situation in which a Patriot starts following a target at coordinates that simplify the problem (in a real situation the Patriot doesn't follow, but goes against the target).

A Patriot battery is placed at coordinates $(x_0, y_0) = (0, 0)$. The Patriot's velocity is $s_0 = 1750$ m/s. The last detection of a Scud missile has taken place at coordinates $(x_1, y_1) = (0, 100)$. with a velocity in the x axis of $s_{1x} = 1676$ m/s at $t = 0$ h.

Task 7. Considering that the Patriot missile is launched at $t = 0$ h, after being rebooted (i.e. accumulated error can be neglected and considered zero):

a. Which direction is heading the missile?

b. At which coordinates and time will the Patriot expect to intercept the Scud?

Task 8. Considering now that the Patriot missile is launched at $t = 0$ h (time of the ground-based radar clock), six hours after being rebooted (i.e. accumulated error has to be considered):

a. Which is the actual internal Patriot's clock (time) when it is launched?

b. Which direction is heading the missile? Don't forget that there are two clocks: the internal Patriot's clock and the ground-based radar clock.

c. At which coordinates and time will the Patriot expect to intercept the Scud?

d. At which distance from the Patriot will the target be at the expected interception time?