Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 47 (2005) 555–563
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Embedded System Paranoia: a tool for testing embedded system arithmetic

Les Hatton*

Centre for Forensic Software Engineering, University of Kingston, Kingston KT1 1LQ, UK

Received 18 February 2004; revised 2 October 2004; accepted 21 October 2004
Available online 25 December 2004

Abstract

The quality of arithmetic implementation is of concern to all who work with or depend on the results of numerical computations. Embedded systems have become enormously complicated and widespread in most if not all consumer devices in recent years so there is a clear need to measure the quality of the arithmetic in the same way that conventional systems have been measured for some time using programs such as the well-known paranoia. A new version of *paranoia* has been introduced specifically to extend the domain of testable systems to embedded control systems. This paper describes the development of ESP (Embedded System Paranoia) and gives example outputs and free download sites. The example outputs indicate that even today, the quality of arithmetic implementations cannot be taken for granted with numerous implementation problems being reported in those embedded environments tried so far.

Keywords: Embedded system Paranoia; Arithmetic implementation; Arithmetic quality

1. Introduction

The reliable implementation of a numerical algorithm depends fundamentally on the underlying quality of the arithmetic implementation. Unfortunately, there are many examples of significant failures in implementation over the years. For example, a version of the CDC Fortran compiler reported 1.0—1.0 as being less than, equal to and greater than zero simultaneously [12]. Naturally such deviations make life hard for the algorithmic programmer and the problem has been addressed successfully by a number of authors over the years [2,8,10] as a result of which effective standardised approaches have appeared [5,6]. These together with tools for diagnosing arithmetic problems have led to a gradual improvement in the quality of implementation of arithmetic such that today in general purpose systems, arithmetic quality is usually quite good, although there still remain significant concerns [9].

Perhaps the greatest concerns today are, however, associated with embedded control systems. Such tools as have appeared with the goal of diagnosing arithmetic implementation problems, have not in general been available for such systems and the average quality of arithmetic implementations for this environment therefore remains unknown.

Embedded control systems are of course at the heart of modern electronic system development. Twenty years ago, an embedded control system might have contained 2K of ROM, a simple 4 bit CPU such as the 74181 and be entirely coded in machine code. In general they controlled very simple devices and few demands were placed on them to implement high quality arithmetic. Today, things are completely different. Embedded control systems are in just about every consumer product from an electric toaster to an automobile. Not only that but the systems are as sophisticated as general purpose systems with in some cases, many megabytes of RAM, IDE discs, high end 32 bit microprocessors and are required to solve complex algorithms in real time such as coupled differential equations. Such systems are commonly programmed in C and can constitute millions of lines of code. Consequently,

* Tel./fax: +44 208 336 1151.
E-mail addresses: lesh@leshatton.org, lesh@oakcomp.co.uk.

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.10.005

the demands on the arithmetic system are as high as in general purpose systems and the distinction between the two types of system becomes increasingly more blurred each year.

After a review of previous tools designed to measure arithmetic quality, the steps necessary to re-structure paranoia will be described, a sample output shown and the results of running the re-structured program on a number of different systems will be tabulated. A discussion of the role of extended precision computation will follow and some conclusions noted.

2. Tools for measuring arithmetic quality

A number of tools of greater or lesser sophistication have emerged over the years with the object of measuring arithmetic quality in some way. These vary from simply diagnosing important properties of the implementation such as the radix to tools capable of diagnosing a much wider class of problems.

2.1. machar

machar is an implementation of work originally done by [2] a C implementation of which appears in [11]. Its primary function is to discover properties of a particular arithmetic implementation normally hidden from the user, for example (using the nomenclature described by [11] with IEEE compliant values in brackets [5]),

- *ibeta*, the radix in which numbers are represented (2,10). (This is *Radix* in the ESP source code.)
- *it*, the number of digits in the base of the radix used to represent the floating point mantissa (24 in single precision). (This is *Precision* in the ESP source code.)
- *machep*, which is the exponent of the smallest power of ibeta such that $1.0+ibeta^{machep}\neq 1.0$, (-23). (This is *U2* in the ESP source code.)
- *eps*, commonly referred to as the 'floating point precision', $ibeta^{machep}$ (1.19×10^{-7}).
- *negep*, which is the exponent of the smallest power of ibeta such that $1.0-ibeta^{negep}\neq 1.0$, (-24). (This is *U1* in the ESP source code.)
- *epsneg* $ibeta^{negep}$, (5.96×10^{-8}) another way of defining floating point precision and usually 0.5 times eps.
- *iexp* is the number of bits in the exponent including the sign, (8).
- *minexp* the smallest power of ibeta consistent with no leading zeroes in the mantissa, (-126).
- *xmin* is $ibeta^{minexp}$, (1.18×10^{-38}) the smallest useable floating value.
- *maxexp* the smallest +ve power of ibeta that causes overflow, (128).
- *xmax* is $(1.0-epsneg)\times ibeta^{maxexp}$, (3.4×10^{38}), the largest useable floating value.
- irnd, the round-off code. In the IEEE standard, bit patterns correspond to 'representable' values. The idea is that in any arithmetic operation with two operands, addition say, the bit patterns are added 'exactly' and then rounded to the nearest representable number. If this is exactly half-way, the low order bit zero value is used. If irnd returns as 2 or 5, rounding complies with IEEE. If it is 2 or 4, non-standard rounding is taking place and if irnd is 0 or 3, truncation is taking place which is not desirable. irnd also describes underflow. In IEEE, underflow is handled by freezing the exponent at the smallest allowed value whilst the mantissa gradually acquires leading zeroes, 'gracefully' losing precision. Other implementations might simply truncate to zero.
- *ngrd* is the number of guard digits used when truncating the product of two mantissae to fit the representation.

2.2. paranoia

paranoia is somewhat different and goes beyond machar [1,2]. Both paranoia and machar try to establish the radix, precision and range (over/underflow thresholds) of the arithmetic but paranoia goes beyond machar in looking for a wider class of pathologies. In its original form, it is implemented as a series of 29 milestones (reporting points) and tests which included the following amongst a large selection:

- Basic tests on arithmetic operations on small numbers. Examples include the following:
  - $1*2=2$, $2*1=2$, $2/1=2$, $2+1=3$ and a number of similar operations.
  - $(-1)+(-1)*(-1)=0$ and a number of tests of commutativity.
- Consistency of comparison generally. This can have important ramifications for many algorithms as the comparison of floating point numbers is responsible for a number of well-known failures [3]. The following are all amongst a wide range of such tests:
  - $0+0=0$
  - $(2+2)/2=2$
  - $X=1$ but $X-1/2-1/2!=0$.
  - Non-normalised subtraction such as $X=Y$, $X+Z!=Y+Z$
  - $(X-Y)+(Y-X)$ is non-zero.
- Underflow behaviour.
- Overflow behaviour.
- Presence of guard digits.
- Tests of square root and powers with particular emphasis on suitability for financial calculations.
- Behaviour with Inf(1/0) and NaN(0/0). The IEEE standards provides for bit patterns which indicate when an operation is pathological in some sense. NaNs (Not a Number) come in two forms, signalling (exceptions are indicated) and quiet.
- Compatibility with the IEEE 754/854 standards.