Embedded system paranoia: a tool for testing embedded system arithmetic

Les Hatton
Computing Laboratory, University of Kent*

19 Feb 2004

Abstract

A new version of the well-known program *paranoia* has been introduced specifically for testing the arithmetic of embedded control systems. Embedded systems have become enormously complicated and widespread in most if not all consumer devices today so there is a clear need to measure the quality of the arithmetic. This paper describes the development of ESP (Embedded System Paranoia) and gives example outputs and free download sites. The example outputs indicate that even in the 21st century, the quality of arithmetic implementations cannot be taken for granted.

1 Introduction

The quality of arithmetic implementation is of fundamental importance to users in computer systems and has been addressed by a number of authors over the years,[9], [8], [2] 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. However such tools as have appeared have not in general been available for embedded control systems.

Embedded control systems are 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 micro-processors 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

*L.Hatton@kent.ac.uk, lesh@oakcomp.co.uk millions of lines of code. Consequently, 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.

Unfortunately, those tools which have been developed to diagnose arithmetic quality have not usually been ported to embedded control system environments and the quality of arithmetic implementation is therefore unknown currently.

2 Tools for measuring arithmetic quality

2.1 machar

machar is an implementation of work originally done by [2] a C implementation of which appears in [10]. Its primary function is to discover properties of a particular arithmetic implementation normally hidden from the user, for example, (using the nomenclature described by [10] with IEEE compliant values in brackets, [5]) include

- *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 $(1.0 - epsneg) \times ibeta^{maxexp}$, ($3.4 \times 10^{38}$), 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-know 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.

paranoia was first introduced as a BASIC program by W.M. Kahan in 1983. The program was subsequently converted into Pascal by Brian Wichmann in 1985 and then to C by David Gay and Thomas Sumner in 1985-6 and was later maintained by Richard Karpinski. The author used paranoia unchanged for a long time on various systems to a great effect, (most systems failed in some way, often spectacularly). However with the explosive growth of embedded systems in the 1990s and the ever more sophisticated demands on the arithmetic and particularly floating point arithmetic in those systems, there is obviously a strong need to test embedded systems in the same way.

2.2.1 What is being diagnosed

Before describing the embedded system version of paranoia (ESP), it is worthwhile reflecting on what paranoia is actually doing when it finds a defect in the arithmetic implementation. There are a number of layers involved with arithmetic, for example, the functions wired onto the chip, perhaps microcode layers and of course the compiler run-time library which sits on top. Any defect might be associated with the chip or may be in the compiler run-time library software itself. In general, paranoia has no way of knowing and simply reports when the implementation as a whole potentially misbehaves in some way. However, if a portable compiler like the GNU (http://www.gnu.org) compiler is used and the same version run on different chips, very different results can be achieved. In early experiments on a Pentium III with gcc 2.6.3, a number of serious defects and failures were observed. Running precisely the same compiler version on a Macintosh G3, a Motorola PowerPC based machine, raised one flaw only. Although not conclusive, this does rather bring into question the quality of the arithmetic implementation on the Pentium III. Later versions of gcc and different compilers like Borland C++ and Visual C++ all passed paranoia with either minor or even no flaws on the same Pentium III hardware so clearly some of the problems at chip level were being worked around by software modifications to the run-time library.

Of course such software work-arounds can affect performance considerably. Gradual movement of high level functionality from software to hardware is one of the key steps forward in improving performance. The reverse step to work-around hardware defects at least partially undoes this good work.

3 ES Paranoia

For all kinds of reasons, optimisation capabilities, small footprint, generally lightweight compiler and environment and plentiful available skill, C has been the dominant language of embedded system control in the last 10 years. However embedded control systems often do not implement the full ISO C standard, nor do they have to. Aware of the heavy use in the embedded system world where originally at least, memory and processing resources were at a premimu, the ISO C committee had deliberately distinguished between *free-standing* (perhaps no underlying OS), and *hosted environments* with a much cut-down version of the standard available in the former.

3.1 ES Paranoia re-engineering

Unfortunately, paranoia was originally written to use a comprehensive set of functionality in the C language and worse, it was designed to be interactive, prompting the user at various stages during the execution. To cater for this, the original source, paranoia.c was treated to the following re-engineering stages:-

- The program was re-written into a consistent style taking into account the original authors' comments ”this program does NOT exhibit good C programming style”.
- The program was updated to use ISO C facilities such as function prototypes.
- The program was modified to make it as standard conforming as possible with ISO C90, ISO C99 and ISO C++99.
- The program was then redesigned as a batch environment.
- All I/O was funnelled through a single function message() whose job is to send a text string back to the host via a serial port, TCP/IP link or whatever facilities happen to be available.
- Two support functions, float2string() and int2string() were created so that there was no dependence on I/O facilities in C which are frequently not implemented in an embedded environment.
- Various parts of the C language can be switched out during compilation by any combination of the flags NOSTDLIB, (if stdio.h is not available which requires the programmer to supply a way of getting a string off the embedded target back on to the host, NOSIGNAL (if signal.h is not available), NOSETJMP (if setjmp.h is not available), and NOZERODIVIDE in case the embedded target is unable to handle either 1/0 or 0/0 without halting.
- Additional tests for other arithmetic problems the author has experienced over the years. This is an ongoing process. Initially only$(x^2 - y^2) - (x - y)(x + y)$ has been added but optional tests for the transcendental and other functions are being designed.

The source code in revision 1.5, 2003-09-26 consists of 3277 source lines.

3.2 ES Paranoia intrinsic quality

Various tests were performed to ensure that annoying defects did not creep in. These separate naturally into static and dynamic tests.

3.2.1 Static tests

The Safer C toolset [11] was used to ensure the absence of the following:-

- Syntax violations
- Constraint violations
- Undefined behaviour
- Common static defects
- Data-flow defects such as the use of objects before initialisation
- Implementation defined behaviour unless it was intrinsic to the way the program works. There are three such occurrences in revision 1.5.
- The program adheres to **EC**− as described in [4], a subset of ISO C which excludes most known failure modes.

The reader is referred to [3] or [7] for explanation of these concepts. As an additional measure, the code was compiled at the highest level of warning with both Visual C++ 6.0 and also GNU C. No serious warnings remain.

3.2.2 Dynamic tests

The program was tested against the original version of paranoia on several different architectures to ensure that the results were consistent. These are summarised in a later section.

3.3 Diagnostic levels in ESP

The original paranoia had four kinds of anomaly which in decreasing order of importance were, FAILURE, SERIOUS DEFECT, DEFECT and FLAW. Based on comments in the original version of paranoia, ESP naturally extends this to six levels of quality defined as follows:-

1. **Excellent**. No FAILURES, SERIOUS DEFECTS, DEFECTS or FLAWS and IEEE 754/854 compatible rounding.
2. **Very good**. No FAILURES, SERIOUS DEFECTS, DEFECTS or FLAWS but non IEEE compatible rounding.
3. **Good**. No FAILURES, SERIOUS DEFECTS or DEFECTS but some inconvenient FLAWS.
4. **Acceptable**. No FAILURES or SERIOUS DEFECTS but some DEFECTS and perhaps some FLAWS.
5. **Unacceptable**. No FAILURES but some SERIOUS DEFECTS and perhaps some DEFECTS and FLAWS.
6. **Broken**. FAILURES and perhaps some SERIOUS DEFECTS, DEFECTS and FLAWS.

3.4 Porting ESP to a new environment

All that is necessary is to modify message(), float2string() and int2string() for the embedded target, compile it on the host switching out facilities as necessary, download it and then run it capturing the output on the host. The source code of message() follows to exemplify what is necessary. It generally takes a very short while to do the necessary changes, (look for the TODO string).

```
/*-----------------------------------------------------------------*/
/*
 *   Get a string to the outside world somehow.
 */
void
message( int append_nl, char * string )
{
/*
 *   Guard - passing NULL to the C run-time library can seriously
 *   damage your health.
 */
     if ( string == NULL )    return;
/*                            =======             */
/*
 *   If stdio is present, use it gratefully.
 */
#ifndef   NOSTDIO
     printf("%s", string);
#else
/*
 *   TODO
 *   Do what you can with local facilities for sending a stream
 *   of characters in the character variable 'string' to the outside world.
 */
#endif
     if ( append_nl )
     {
#    ifndef    NOSTDIO
          printf("\n");
#    else
/*
 *   TODO
 *   Do what you can with local facilities for sending a newline
 *   to the outside world.
 */
#    endif
     }
}
/*-----------------------------------------------------------------*/
```

float2string() and int2string() are similar although may need a little more thinking about. The whole of the code is available for free download and the author will undertake to merge user's contributions for this.

3.5 Portability

Most embedded system compilers now implement ISO C quite well at least in its 9899:1990 incarnation. The author took great pains to make the code compliant with ISO C and ISO C++ so there should be little problem running ESP on a reasonable embedded system.

4 Running ESP on a general purpose machine

ESP is now a batch program with no user intervention. As an example, it was built to assume the presence of signal.h, setjmp.h, and stdio.h and allowed to try to divide by zero. It then produced the following output (abbreviated here) on a SuSE 8.2 Linux machine running gcc version 3.3 20030226.

```
COMMENT: =========================================
COMMENT: Welcome to ESP - Embedded System Paranoia
COMMENT: Please let me know your experiences
COMMENT: and suggestions at lesh@oakcomp.co.uk or
COMMENT: L.Hatton@kent.ac.uk
COMMENT:
COMMENT: $Revision: 1.1 $ $Date: 2004/02/18 05:20:19 $
COMMENT: This version will attempt divide by zero.
COMMENT: This version uses <stdio.h>
COMMENT: This version uses <signal.h>
COMMENT: This version uses <setjmp.h>
COMMENT: This version uses double precision.
COMMENT: =========================================

-------> Diagnosis resuming after Milestone 0, Page 1
COMMENT: -1, 0, 1/2, 1, 2, 3, 4, 5, 9, 27, 32 & 240
PASSED : small integer tests are all OK.
COMMENT: Searching for Radix and Precision.
COMMENT: Radix = 2.00000000000000000e+00
COMMENT: Closest relative separation found is U1 = 1.11022302462515654e-16
COMMENT: Recalculating radix and precision
COMMENT: confirms closest relative separation U1.
COMMENT: Radix confirmed.

-------> Diagnosis resuming after Milestone 10, Page 2

-------> Diagnosis resuming after Milestone 20, Page 3
COMMENT: The number of significant digits of the
COMMENT: Radix is 5.30000000000000000e+01

-------> Diagnosis resuming after Milestone 25, Page 4
COMMENT: Some subexpressions appear to be calculated extra precisely with
COMMENT: about 3.31132995230379290e+00 extra significant decimals.
COMMENT: This is not tested further by this program.

-------> Diagnosis resuming after Milestone 30, Page 5
```

```
COMMENT: Subtraction appears to be normalized, as it should be.
COMMENT: Checking for guard digit in *, / and -.
PASSED : *, /, and - appear to have guard digits, as they should.

-------> Diagnosis resuming after Milestone 35, Page 6
COMMENT: Checking rounding on multiply, divide and add/subtract.
FLAW   : Multiplication neither chopped nor correctly rounded.

         ***********************************
         LOTS MORE OUTPUT NOT SHOWN HERE ...
         ***********************************

COMMENT: Trying to compute 1/0 gives inf
COMMENT: Trying to compute 0/0 gives nan

-------> Diagnosis resuming after Milestone 210, Page 30
COMMENT: =========================================
COMMENT:     Embedded System Paranoia SUMMARY
COMMENT:
COMMENT: Number of DEFECTs  discovered = 1
COMMENT: Number of FLAWs  discovered = 1
COMMENT:
COMMENT: The arithmetic diagnosed may be Acceptable
COMMENT: despite inconvenient DEFECT.
COMMENT: Rating ...
                Excellent
                Very good
                Good
         =====> Acceptable
                Unacceptable
                Broken
COMMENT: END OF TEST.
COMMENT: =========================================
```

The size of the generated object module with everything compiled in as above is around 80K with the GNU compiler.

5 Results of running ESP on real systems

The following results show what happens with real systems. They are shown in tabular form along with an explanation of the environment under which they were run. In one case, ESP could only be run on the simulator for space reasons.

5.1 Embedded systems

Compiler / chip	Environment	Date	Failures	Serious Defects	Defects	Flaws
IAR / MSP430	simulator + EC++ library	01-Nov-2003	0	0	0	1
IAR / MSP430	simulator + IEEE library	01-Nov-2003	0	0	6	1
TI-OMAP (ARM9)	target	26-Sep-2003	0	0	6	4
IAR / MSP430	simulator + Fast math library	01-Nov-2003	2	1	9	4
KEIL	Compiler v. 3.12k	15-Jan-2004	4	2	4	3
STAR-12	target	19-Sep-2003	6	7	12	1

5.2 General purpose systems

As a control, the results of running ESP on general purpose systems is shown below. Note that for these systems, the original paranoia was also run and gave consistent results in each case as a further control.

Compiler / chip	Environment	Date	Failures	Serious Defects	Defects	Flaws
Visual C++ (no optimisation) / Athlon 1400	Windows 98SE	18-Feb-2004	0	0	0	0
GNU C 3.3 20030226 (no optimisation) / Athlon XP2700+	SuSE Linux 8.2	26-Sep-2003	0	0	1	1
GNU C 3.3.1 (no optimisation) / Athlon XP2700+	SuSE Linux 9.0	18-Feb-2004	0	0	1	4
GNU C 3.3.1 (-O2 optimisation) / Athlon XP2700+	SuSE Linux 9.0	18-Feb-2004	2	3	5	5

Two things can immediately be noted. First of all, the average level of arithmetic quality in a general purpose environment is generally rather higher than in those embedded control systems tried so far. Second, note the perils of using optimisation levels even on general purpose machines when the compiler is quite good without optimisation.

6 Downloading ESP

ESP is available for free download from the author's personal site, http://www.leshatton.org/ as a zipped file containg a README document, the source esparanoia.c and a sample output. The author welcomes results from different systems and will endeavour to collate them for easy access on the above site.

7 Conclusions and Acknowledgements

A long overdue and enhanced version of paranoia has been made available for embedded control systems. So far results suggest that it is very valuable in flushing out anomalies in arithmetic implementations on embedded control systems. The author would like to acknowledge the help of Sivasankaran Krishnan, Sukumar Ranjeethkumar and Vibin Viswanbharan (Visteon India), Jurg Sturli (WORX) and Chris Tapp (Keylevel Consultants) for kindly taking the time to adapt, compile and run ESP on the systems shown.

Finally, the author would like to acknowledge the pioneering work of the original authors. The continuing existence of compiler / chip combinations which fail this test bears mute witness to the importance of their work.

References

[1] Cody W.J., Waite W. (1980) *Software Manual for the Elementary Functions* Prentice-Hall, Englewood Cliffs, NJ

[2] Cody W.J. (1988) *ACM Transactions on Mathematical Software* 14, p. 303-311

[3] Hatton L. (1995) *Safer C: developing software for high integrity and safety critical systems* McGraw-Hill, ISBM 0-07-707640-0

[4] Hatton L. (2004) *EC– A measurement based safer subset of ISO C suitable for embedded system development* Submitted to IST.

[5] IEEE (1985) *IEEE standard for binary floating-point numbers* ANSI/IEEE 754, IEEE, New York

[6] IEEE (1987) *IEEE standard for radix-independent floating-point arithmetic* ANSI/IEEE 854, IEEE, New York

[7] ISO (1999) *ISO 9899:1999 C programming language* International Standards Organisation.

[8] Kahan W.M. (1983) *Documentation header of the original paranoia* http://research.att.com/

[9] Malcolm, M.A. (1972) *Communications of the ACM* 15, p.949-951

[10] Press, W.H., Teukolsky S.A., Vettering W.T., Flannery B.P. (1992) *Numerical Recipes in C, second edition* Cambridge University Press, ISBN 0-521-43108-5

[11] The Safer C toolset (1999) *A toolset for enforcing statically safer subsets in the C language* http://www.oakcomp.co.uk/