Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A floating point conversion algorithm for mixed precision computations

Choon Lih HOO, Sallehuddin Mohamed HARIS, Nik Abdullah Nik MOHAMED

(*Department of Mechanical and Materials Engineering, Universiti Kebangsaan Malaysia, UKM Bangi 43600, Malaysia*)

E-mail: steven85@vlsi.eng.ukm.my; {salleh, enikkei}@eng.ukm.my

Received Feb. 21, 2012; Revision accepted July 12, 2012; Crosschecked Aug. 3, 2012

Abstract: The floating point number is the most commonly used real number representation for digital computations due to its high precision characteristics. It is used on computers and on single chip applications such as DSP chips. Double precision (64-bit) representations allow for a wider range of real numbers to be denoted. However, single precision (32-bit) operations are more efficient. Recently, there has been an increasing interest in mixed precision computations which take advantage of single precision efficiency on 64-bit numbers. This calls for the ability to interchange between the two formats. In this paper, an algorithm that converts floating point numbers from 64- to 32-bit representations is presented. The algorithm was implemented as a Verilog code and tested on field programmable gate array (FPGA) using the Quartus II DE2 board and Agilent 16821A portable logic analyzer. Results indicate that the algorithm can perform the conversion reliably and accurately within a constant execution time of 25 ns with a 20 MHz clock frequency regardless of the number being converted.

Key words: Double precision, Single precision, FPGA, Verilog, HooHar algorithm
doi:10.1631/jzus.C1200043 **Document code:** A **CLC number:** TN402

1 Introduction

Despite its complex configuration, the floating point number representation is widely used in computing applications because of its ability in providing higher precision. In recent years, much of the research on floating points has involved their use in practical applications, e.g., floating point analogue-to-digital converters (FP-ADCs), which are used to quantize large dynamic signals such as those found during seismic monitoring (Linnenbrink and Gaalema, 1991). There have also been studies on floating points that focus on the speed of conversion and accuracy (Groza and Dzerdz, 2004), and also on arithmetic implementation cost and power dissipation (Seidel, 2004). Many floating point applications use double precision, where the use of fast, cheap, and small digital signal processors (DSPs) is required (Boldo and Daumas, 2001). This requirement exists because a higher precision floating point requires an increased number of transistors and wider data paths within the chip. This, however, induces some difficulty in meeting time constraints (Goddeke *et al*., 2007).

Recent processor architectures exhibit better single precision performance compared to double precision arithmetic, in that they give higher theoretical peaks, use fewer data paths, and run at higher speeds. Hence, if the numerical range is within its limits, the use of single precision is preferable. A single precision operation is often at least twice as fast as a double precision operation because the amount of bytes moved through the memory is halved (Baboulin *et al*., 2009). Some researchers try to exploit single precision performance and, at the same time, gain double precision features (Langou *et al*., 2006), and this leads to the need for double to single precision conversion.

The benefits of single precision become more prominent when used in mixed precision algorithms

© Zhejiang University and Springer-Verlag Berlin Heidelberg 2012

(Baboulin *et al*., 2009) and solvers (Goddeke *et al*., 2007). In mixed precision applications, the single precision format plays an important role in arithmetic operations as it works faster than double precision. The solution is then refined with 64-bit accuracy (Goddeke *et al*., 2007; Baboulin *et al*., 2009). The approach works well, giving added speedup factors of four to five times and space savings of three to four times, while maintaining the same accuracy as reference solvers that operate solely in double precision (Goddeke *et al*., 2007).

There are also devices that operate only in the single precision format. As an example, Spence *et al*. (2009) wrote a program originally in double precision for parallel implementation, to accelerate numerical quadrature using a graphical processing unit (GPU). However, it had to be converted to single precision as the GPU does not support double precision arithmetic. In another development, a preliminary approach to accelerating double precision finite element simulations by using the GPU has also been presented (Goddeke *et al*., 2005), where the GPU acts as a co-processor that implements mixed precision defect corrections. Results showed equally good accuracy but significantly improved speed in comparison to double precision central processing unit (CPU) solvers (Goddeke *et al*., 2005). The value of double to single precision conversion is even more prominent when the error correction solver for linear systems uses single precision, which has a lower precision in its inner error correction solver, while maintaining a double precision format in the outer loop. This promising approach has shown speedup characteristics (Anzt *et al*., 2011a). This indicates the importance of mixed precision and, in particular, the conversion from double to single precision, as the single precision format requires less computational effort. Anzt *et al*. (2011b) showed a reduction in power consumption when using mixed precision in the iterative solution of sparse linear systems.

Hasanien (2011) noted that field programmable gate array (FPGA) technology is playing a significant role in the development of very large scale integrated (VLSI) circuits due to its fast response, high reliability, flexible and programmable architecture, and the cost reductions that come with increasing density (Wen *et al*., 2006). These benefits have resulted in FPGAs receiving rapidly increasing attention, as can be seen from recent industry trends. FPGAs could also possibly replace the use of expensive software in advanced control technology.

Clearly, the floating point number representation and its conversion from double to single precision are of significant importance. Hence, this paper focuses on establishing a double to single precision conversion algorithm in Verilog, which allows for implementation on FPGA. The algorithm serves as a simple conversion concept but yet could lend itself to practical and useful implementation on integrated circuit hardware.

2 Floating point numbers

Today, the IEEE Standard 754 floating point has become the most common representation for real numbers on computers as it gives better precision flexibility (Hollasch, 2005). This is because the floating point format represents real numbers in scientific notation that provides very large or very small numbers, compared to the fixed point format which has limited precision ability due to its fixed window of representation.

The floating point number is made up of three components: the sign bit, the exponent, and the mantissa (Hollasch, 2005). The sign bit, which is the leftmost bit, denotes whether the number is positive or negative. The exponent, with the aid of the bias, represents both positive and negative exponents through suitably stored exponents, while the mantissa, also known as the significand, denotes the precision bits of the number.

In the floating point basic number system, there are two well-known formats: single precision format and double precision format. Both formats comprise the same components as described above, but they vary in terms of the precision and flexibility of their number range representation (IEEE, 1985).

2.1 Single precision

The single precision, also known as the 32-bit binary number, has 32 binary bits with one sign bit represented by the leftmost bit, followed by eight binary bits as the exponent field and 23 mantissa bits. The bias for this format is $2^7-1=127$.

2.2 Double precision

For double precision or 64-bit binary numbers, as the name implies, the format exhibits 64 binary bits with one sign bit and 11 exponent bits, followed by 52 mantissa bits. As suggested by the double quantity of binary bits, the double precision can represent a wider range of decimal numbers compared to the single precision format. In double precision, the bias is 1023, which results from $2^{10}-1$. Table 1 summarizes the differences between single and double precision.

Table 1 Summary of single and double precision formats (Hollasch, 2005)

Format	Number of bits			Bit range			Bias
	Sign	Exp.	Man.	Sign	Exp.	Man.	
Single	1	8	23	31	30–23	22–0	127
Double	1	11	52	63	62–52	51–0	1023

Exp.: exponent; Man.: mantissa

3 The HooHar method

The HooHar algorithm, which converts double precision format into single precision format, is proposed in this study. Performing the conversion would appear to be natural, as both numbering formats are similar in terms of the three floating point components: the sign bit, the exponent, and the mantissa. However, the double precision representation has higher numbers of bits forming the exponent and mantissa (Table 1). This accounts for the difference in precision between the two.

Despite the disparity in the number of bits, both formats fundamentally work in the same way (Table 2). Hence, notwithstanding values in the out-of-bounds range induced by the double precision, the single precision format can represent real numbers to the same decimal precision as the double precision representation, provided that the power of the exponent is made equal.

Table 2 Representation of single and double precision formats (Hollasch, 2005)

Format	Normalized	Denormalized
Single	$(-1)s\times1.f\times2^{e-127}$	$(-1)s\times0.f\times2^{-126}$
Double	$(-1)s\times1.f\times2^{e-1023}$	$(-1)s\times0.f\times2^{-1022}$

s represents the sign bit, *f* is the bit value from the mantissa, and *e* indicates the bit value from the exponent

3.1 Conversion part

The conversion method can be classified into five different categories according to the range in which the number lies. The way that the conversion is performed for each of these classifications is explained as follows.

3.1.1 Signaling range

A number in this range is classified as ‘not a number’ (NaN), indicating invalid operation when the exponent reaches its maximum value. The remaining mantissa bits are application-specific, such that they could, for example, encode diagnostic information.

3.1.2 Overflow range

This refers to the range in which the conversion can result only in the maximum exponent value with all mantissa bits being zero. The exponent achievable by the single precision would have reached its upper limit and any further increment in power would not be representable in single precision format.

3.1.3 Normal range

In this category, the exponent in the single precision format ranges from 0000 0001 to 1111 1110, corresponding to the range of 0111 0000 001 to 1000 1111 110 in the double precision representation. The normal range occurs when both formats have the same exponent power, after taking into consideration each one’s respective bias. Equivalence exists once the double precision’s exponent reaches 0111 0000 001 giving a power of −126. The mantissa is adopted from the first 23 bits of the mantissa in double precision format, as it carries the same meaning in single precision.

3.1.4 Denormalized range

In the denormalized range, the single precision format is required to establish the same exponent power as in double precision through a combination of its exponent and mantissa components. For single precision, the denormalized range induces all zeros in the exponent bits, giving a power of −127. In double precision, this is equivalent to 0111 0000 000. The double precision’s exponent could go much lower with a reduction of 1 bit value at a time. This can be compensated in single precision with the introduction of a 1 at the bit location of the deficient power value.

As an example, for a double precision exponent component of 0110 1111 011 which gives a power of −132, the exponent for single precision would provide −127, and the remaining power of −5 will be made up by introducing a 1 at the bit position of −5 (Table 3). The preceding bits would be zero and the succeeding bits would be adopted from the mantissa in the double precision format. This can be explained by the following calculation:

$$2^{-132} \text{ (exponent of double)}$$
$$=2^{-127} \text{ (exponent of single)}\times 2^{-5} \text{ (mantissa of single)}.$$

Following this rule, Eq. (1) is established for any double to single precision conversion:

$$X=Y-1023+127, \quad (1)$$

where Y is the base-10 value given by the double precision's exponent bits, 1023 and 127 are the biases of the double and single precision formats respectively, and X is the range indicator.

In the denormalized range, the absolute value of X determines the number of leading zeros in the mantissa, which is followed by a bit 1. The subsequent bits adopt the first (22+X) mantissa bits from the double precision representation. X also marks the location of the first bit 1 in the mantissa.

3.1.5 Underflow range

In this case, as the name implies, the exponent power would be too small to be represented in the single precision format. The single precision representation is set to zero for every bit, giving its smallest possible value.

3.2 Rounding to the nearest value

Conversion from higher to lower precision will naturally cause some loss in precision. Rounding is essential to ensure that the conversion retains the decimal value as closely as possible. As stated in IEEE Std 754-1985 (IEEE, 1985), rounding takes or even modifies the number in order to fit it into the destination's format. Several rounding modes are available, but only 'rounding to the nearest value' has been adopted in this conversion algorithm, as it is the most common rounding type. This mode connotes that the nearest representable value to the infinitely precise result shall be taken, and the one with its least significant bit zero will be chosen when two equally near, nearest representable values meet.

In the HooHar algorithm, rounding works by choosing the nearest possible representable value of the destination's format. The immediately lower and immediately higher representable values are compared to the desired input value, and the one giving the smaller difference would be picked.

In digital systems, a comparison between decimal values would be difficult, especially when one needs to compare two decimal numbers represented by two different floating point formats (double and single precision). The decimal point would be the main concern when comparing the numbers as it determines the precision. Hence, the rounding mode is achieved by using the mantissa component of the floating point number. Note that the rounding mode is applicable only to normal, denormalized, or underflow range values, as rounding does not bring any meaning to the overflow and signaling categories. Table 4 shows the necessary variables used in the rounding mode.

4 Conversion algorithm

The HooHar algorithm was developed in Verilog with the aid of ModelSim-Altera 6.5b and Quartus II 9.1sp2 (Altera, USA). The algorithm can briefly be described by the following steps:

Table 3 Example of denormalized conversion

	Sign	Exponent								Mantissa																						
Bit No.	[31]	[30]	[29]	[28]	[27]	[26]	[25]	[24]	[23]	[22]	[21]	[20]	[19]	[18]	[17]	[16]	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Bit power		7	6	5	4	3	2	1	0	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13	−14	−15	−16	−17	−18	−19	−20	−21	−22
Bit		0	0	0	0	0	0	0	0	0	0	0	0	0	1	First 17 bits from double mantissa																
Power value		−127								−5																						

Table 4 Variables for rounding to the nearest value part

Format	Bit [53]	Bits [52:0]
Double	0	Mantissa from double precision
Single1	0	(22+X) converted bits 53−(22+X) zero bits
Single2	0	(22+X) converted bits 53−(22+X) zero bits with added bit 1*
	1	(22+X) converted bits 53−(22+X) zero bits with added bit 1**

* If no carrier 1 brought forward to the 53rd bit; ** If obtaining carrier 1 brought forward to the 53rd bit

1. Conversion.

Input a double precision binary number.

Adopt the sign bit directly from the double precision format while the exponent and mantissa fields are converted according to Table 5.

2. Rounding to the nearest value.

Define the three variables (double, single1, and single2) as in Table 4.

Compute [single2−double] and [double−single1].

If [single2−double]<[double−single1], choose single2 as the rounded value.

If [single2−double]>[double−single1], choose single1 as the rounded value.

If [single2−double]=[double−single1], choose the single* with an even least significant bit, where single* denotes [either single1 or single2].

3. Choosing from temp_B1 (Table 5) the 32-bit result that corresponds to the chosen single*.

4. Outputting the chosen 32-bit binary number.

The algorithm starts with the input of a double precision binary number, whose exponent is subsequently checked against Table 5. The sign bit follows that of the input double precision number, and the exponent and mantissa follow the corresponding values from Table 5. The result is a 32-bit number that needs to go through the rounding mode. For every result from the conversion part, the converted number is compared to the next representable higher number. This number is obtained by adding a bit 1 to the least significant bit. Since the exponent power already matches the desired exponent power, only the mantissa is compared. In the rounding process, three 54-bit variables need to be defined. The first variable is double precision with a zero first bit, followed by all 53 mantissa bits from the double precision number. The single1 variable is made up of a zero in the first bit, followed by 23 bits from the mantissa, and the last 30 bits are set to zero. For the third variable which is single2, the first bit is initially zero, followed by 23 bits of the mantissa. Then 1 is added to this number, and if a carrier of 1 emerges, the first bit is changed to 1; otherwise, it remains 0. The remaining 30 bits are set to zero. The same concept applies to the denormalized and underflow classes, but with only 22+X bits used, and the remaining bits set to zero.

Two comparisons are made: [single2−double] and [double−single1]. The single* giving the smaller comparison result is taken as the final rounded binary number. If single1 is better, the converted 32-bit from the conversion part will be output as the final converted value. However, if single2 is nearer to the desired value, the converted 32-bit in the conversion part, with the addition of 1 to the least significant bit, will be the final result. If both comparisons result in equal difference values, the single* with an even least significant bit will be the final rounded binary number. A flowchart of the overall process is shown in Fig. 1.

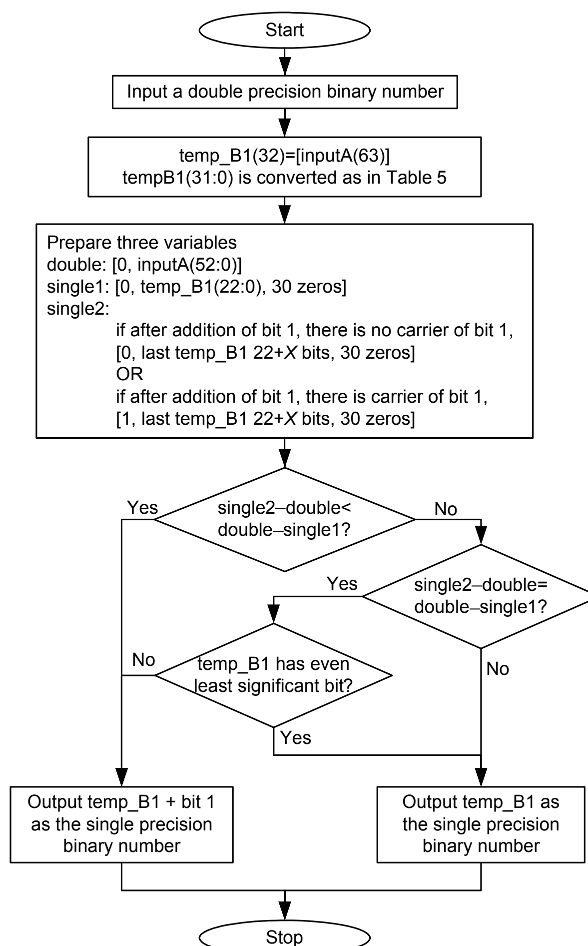


Fig. 1 Flowchart of the HooHar algorithm

Table 5 Algorithm for the HooHar conversion part

inputA (64-bit)			temp_B1 (32-bit)			
Exponent		Mantissa	X	Exponent	Mantissa	Status
1111 1111 111		All one	1151	1111 1111	inputA [51:29]	Signaling
	to					
1111 1111 111		0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001				
1111 1111 111		All zero	255–1151	1111 1111	All zero	Overflow
	to					
1000 1111 111		All zero				
1000 1111 110		Any value	1–254	inputA[62], inputA[58:52]	inputA [51:29]	Normal
	to					
0111 0000 001		Any value				
0111 0000 000		Any value	0	0000 0000	First absolute value of X zeros, bit 1, the first (22+X) double precision mantissas	Denormalized
0110 1111 111		Any value	−1	0000 0000		
0110 1111 110		Any value	−2	0000 0000		
0110 1111 101		Any value	−3	0000 0000		
0110 1111 100		Any value	−4	0000 0000		
0110 1111 011		Any value	−5	0000 0000		
0110 1111 010		Any value	−6	0000 0000		
0110 1111 001		Any value	−7	0000 0000		
0110 1111 000		Any value	−8	0000 0000		
0110 1110 111		Any value	−9	0000 0000		
0110 1110 110		Any value	−10	0000 0000		
0110 1110 101		Any value	−11	0000 0000		
0110 1110 100		Any value	−12	0000 0000		
0110 1110 011		Any value	−13	0000 0000		
0110 1110 010		Any value	−14	0000 0000		
0110 1110 001		Any value	−15	0000 0000		
0110 1110 000		Any value	−16	0000 0000		
0110 1101 111		Any value	−17	0000 0000		
0110 1101 110		Any value	−18	0000 0000		
0110 1101 101		Any value	−19	0000 0000		
0110 1101 100		Any value	−20	0000 0000		
0110 1101 011		Any value	−21	0000 0000		
0110 1101 010		Any value	−22	0000 0000		
0110 1101 001		Any value	−23	0000 0000	All zero	Underflow
	to					
0000 0000 000		Any value				

5 Validation and conclusions

The HooHar algorithm was tested using a hardware testbench, where inputs of 64 bits were sent from the Quartus II 9.1sp2 Web Edition software to the Quartus II DE2 board for processing, and the result was output into the Agilent 16821A portable logic analyzer. The result was later compared with calculators A and B. Calculator A is composed of a 64-bit double precision applet and a 32-bit single precision applet developed by Werner & Walter Randelshofer (32 Bit Single Precision Applet, http://

www.randelshofer.ch/fhw/gri/floatapplet.html; 64 Bit Double Precision Applet, http://www.randelshofer.ch/fhw/gri/doubleapplet.html). The calculator B webpage was created by Kevin J. Brewer of Delco Electronics, and later modified by Christopher Vickery from the Computer Science Department at Queens College of CUNY (Christopher, 2012). A decimal number was inserted into the 32-bit double precision applet to generate its corresponding 32-bit binary format, and into a 64-bit double precision applet to obtain the 64 bits. These later served as the comparators. In the meantime, the output from the 64-bit double precision applet was input into calculator B and the Quartus II testbench for conversion.

Table 6 shows representative results of conversion using the HooHar algorithm for comparison with results from the two online floating point calculators, calculators A and B. Clearly, the HooHar algorithm and the online floating point conversion calculators produced identical single precision results, indicating that the algorithm reliably performs double precision to single precision conversions over the allowable range of numbers.

Table 6 Results obtained using the HooHar algorithm and online calculators*

Decimal	64-bit hexadecimal	32-bit hexadecimal	Execution time (ns)
2.32117E38	47E5 D403 7579 7064	7F2E A01C	25
−1.20565E34	C702 9373 6479 F6F2	F814 9B9B	25
−3.33754E38	C7EF 62D5 D51F 4CE6	FF7B 16AF	25
1.4E-45	369F F868 BF4D 956A	0000 0001	25
9.99883E-40	37D5 C688 B468 EADE	000A E344	25
3.69423	400D 8DC8 754F 3776	406C 6E44	25
−908.6149	C08C 64EB 50B0 F27C	C463 275B	25
−2.23472E-29	B9FC 5414 BDE0 E3B9	8FE2 A0A6	25

The first two columns are the input data and the last two columns are results obtained using the HooHar algorithm. * For both calculators A and B, the 32-bit results are identical to those of HooHar; the execution time is not available for calculator A, and not consistent for calculator B

Significantly, the HooHar algorithm was executed on FPGA, where the conversion took a constant execution time of 25 ns with a 20 MHz frequency clock regardless of the numbers being converted. The 25 ns includes the time taken to get the input from the test bench, convert it in the DE2 board, and show the output in the logic analyzer. For the online converter calculators, execution time is dependent on software and hardware speeds, and also on the input number being converted, resulting in inconsistent operation times. To the best of our knowledge, there is no other double precision to single precision converter that reliably performs the conversions consistently within specific execution times. The fact that it was coded in Verilog and executed on FPGA opens up the possibility of implementing the algorithm in the form of integrated circuit hardware, allowing for consistently fast executions and use in single chip applications.

References

Anzt, H., Heuveline, V., Rocker, B., 2011a. An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations. High Performance Computing for Computational Science-VECPAR 2010, p.58-70. [doi:10.1007/978-3-642-19328-6_8]

Anzt, H., Heuveline, V., Rocker, B., Castillo, M., Fernandez, J.C., Mayo, R., Quintana-Orti, E.S., 2011b. Power Consumption of Mixed Precision in the Iterative Solution of Sparse Linear Systems. IEEE Int. Symp. on Parallel & Distributed Processing Workshops and PhD Forum, p.829-836. [doi:10.1109/IPDPS.2011.226]

Baboulin, M., Buttaro, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., Tomov, S., 2009. Accelerating scientific computations with mixed precision algorithms. *Comput. Phys. Commun.*, **180**(12):2526-2533. [doi:10.1016/j.cpc.2008.11.005]

Boldo, S., Daumas, M., 2001. A Mechanically-Validated Technique for Extending the Available Precision. Conf. Record of the 35th Asilomar Conf. on Signals, Systems and Computer, p.1299-1303. [doi:10.1109/ACSSC.2001.987700]

Christopher, V., 2012. IEEE-754: Floating-Point Conversion. Available from http://babbage.cs.qc.edu/IEEE-754/64bit.html [Accessed on Feb. 20, 2012].

Goddeke, D., Strzodka, R., Turek, S., 2005. Accelerating Double Precision FEM Simulations with GPUs. Proc. ASIM 18th Symp. on Simulation Technique, p.139-144.

Goddeke, D., Strzodka, R., Turek, S., 2007. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *Int. J. Parall. Emerg. Distr. Syst.*, **22**(4):221-256. [doi:10.1080/17445760601122076]

Groza, V., Dzerdz, B., 2004. Differential predictive floating-point analog-to-digital converter. *Measurement*, **35**(2): 139-151. [doi:10.1016/j.measurement.2003.08.004]

Hasanien, H.M., 2011. FPGA implementation of adaptive ANN controller for speed regulation of permanent magnet stepper motor drives. *Energy Conv. Manag.*, **52**(2):1252-

1257. [doi:10.1016/j.enconman.2010.09.021]

Hollasch, S., 2005. IEEE Standard 754: Floating Point Numbers. Available from http://steve.hollasch.net/cgindex/coding/ieeefloat.html [Accessed on Feb. 20, 2012].

IEEE, 1985. IEEE Standard for Binary Floating-Point Arithmetic. IEEE Std 754-1985.

Langou, J., Luszczek, P., Kurzak, J., Buttari, A., Dongarra, J., 2006. Exploiting the Performance of 32 Bit Floating Point Arithmetic in Obtaining 64 Bit Accuracy. Proc. ACM/IEEE Conf. on Supercomputing, Article No. 113, p.50. [doi:10.1145/1188455.1188573]

Linnenbrink, T.E., Gaalema, S.D., 1991. Floating-Point Analog-to-Digital Converter. US Patent 5 061 927.

Seidel, P.M., 2004. On-line IEEE Floating-Point Multiplication and Division for Reduced Power Dissipation. Conf. Record of the 38th Asilomar Conf. on Signals, Systems and Computers, p.498-502. [doi:10.1109/ACSSC.2004.1399182]

Spence, I., Scott, N.S., Gillan, C.J., 2009. Enabling science through emerging HPC technologies: accelerating numerical quadrature using a GPU. *Numer. Methods Progr.*, **10**:385-388.

Wen, Z., Chen, W., Xu, Z., Wang, J., 2006. Analysis of Two-Phase Stepper Motor Driver Based on FPGA. Proc. IEEE Int. Conf. on Industrial Informatics, p.821-826. [doi:10.1109/INDIN.2006.275668]

JZUS-A wins "The Chinese Government Award for Publishing" for Journals

浙江大学学报(英文版)A辑荣获"第二届中国出版政府奖":首届期刊奖

J. Zhejiang Univ.-SCI. A
(Applied Physics & Engineering)
IF=0.408(2011)

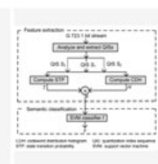
J. Zhejiang Univ.-SCI. B
(Biomedicine & Biotechnology)
IF=1.099(2011)

J. Zhejiang Univ.-SCI. C
(Computers & Electronics)
IF=0.308(2011)

Highlights

Detection of quantization index modulation steganography in G.723.1 bit stream based on quantization index sequence analysis

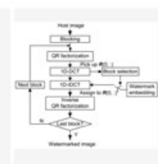
This paper presents a method to detect the quantization index modulation (QIM) steganography in G.723.1 bit stream. We show that the distribution of each quantization index (codeword) in the quantization index sequence has unbalan...



DOI:10.1631/jzus.C1100374 **Downloaded:** 257 **Clicked:**361 **Cited:**0 **Comments:**0 Full Text

A robust watermarking algorithm based on QR factorization and DCT using quantization index modulation technique

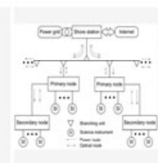
We propose a robust digital watermarking algorithm for copyright protection. A stable feature is obtained by utilizing QR factorization and discrete cosine transform (DCT) techniques, and a meaningful watermark image is embedded i...



DOI:10.1631/jzus.C1100338 **Downloaded:** 254 **Clicked:**294 **Cited:**0 **Comments:**0 Full Text

Development of a direct current power system for a multi-node cabled ocean observatory system

In this paper, we develop a direct current (DC) power system which is applied to a multi-node cabled ocean observatory system named ZERO (Zhejiang University Experimental and Research Observatory). The system addresses significant...



DOI:10.1631/jzus.C1100381 **Downloaded:** 175 **Clicked:**311 **Cited:**0 **Comments:**0 Full Text

Journal of Zhejiang University-SCIENCE C (Computers & Electronics) **homepage (part), http://www.zju.edu.cn/jzus**

New section "Highlights" has been available since August 2012. Editor from Thomson Reuters informed us: *Journal of Zhejiang University-SCIENCE C* will be included in the September reload of the 2011 JCR. The first IF is 0.308