Програмчлагддаг Логик Чипэнд суурилсан 32 бит Floating-Point тооны үржүүлэгчийн дизайн

Ж.Баттогтох, Б.Зориг, М.Доржжамц
Монгол Улсын Их Сургууль, Мэдээллийн Технологийн Сургууль,
Электроникийн Тэнхим
Battogtokh8@gmail.com

***Хураангуй*—Floating Point арифметик нь бодолтын үр дүнг илүү нарийвчлалтай илэрхийлэхийн тулд тооцоолон бодох системд өргөн хэрэглэгддэг. Гэхдээ floating point үйлдлүүдийн дизайныг програмчлагддаг логик чипэнд хийх нь нилээд ярвигтай. Энэ ажлаар дан нарийвчлалт floating point тооны үржүүлэгчийн hardware дизайныг хийхийг зорьсон. Үржүүлэгчийн хэсгийг хийхэд Modified booth and Wallace-Tree үржүүлэгч ашиласан. Санал болгож буй Floating point үржүүлэгчийн дизайныг VHDL хэл дээр загварчилж Modelsim SE 5.7f програмыг ашиглан симуляц хийсэн ба ISE Xilinx 12.3i програмаар Spartan 3S500E FPGA – дээр синтез хийсэн.**

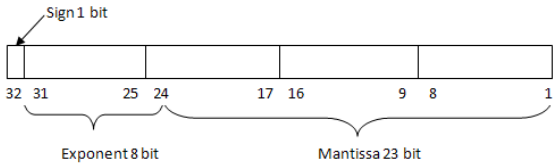
Түлхүүр үг—Floating point; FPGA; Modified boot; Wallace–tree; Carry Lookahead

I. УДИРТГАЛ

Тоон дохио боловсруулах салбарт удаан хугацаанд микропроцессорт суйралсан боловсруулалтууд зонхилж байсан. Дохио боловсруулах алгоритмыг хэрэглэсэн хардвар бүтээгдэхүүнүүд маш олон хэрэглээнд өргөн хэрэглэгдэж байна. Электроникийн дизайныг автоматжуулах түүлүүд болон прогармчлагддаг логик чип гарч ирснээр эдгээр дохио боловсруулах үйлдлүүдийг илүү хурдтай, үр ашигтай боловсруулах боломжууд нээгдсэн. Дүрс болон дуу боловсруулах маш олон салбарт том хэмжээний өгөгдлийг харьцангуй өндөр нарийвчлалтайгаар боловсруулах шаардлага гардаг. Floating point үйлдлүүдийг програмчлагддаг логик чипэнд хийх нь нилээд ярвигтай яагаад гэвэл тэдгээрийн алгоритмууд нь нилээд төвөгтэй байдаг. Floating point үйлдлүүд нь програмчлагддаг логик чипэнд хийхэд үргэлж их хэмжээний зай шаарддаг. Мөн маш олон шинжлэх ухааны салбарт floating point арифметикийг өндөр нарийвчлалтай хэрэглэхийг шаарддаг. Иймээс 32 битийн floating point операторыг програмчилагддаг логик чипэнд хийсэн. Бид хоёртын floating point арифметикийн IEEE 754 стандартыг ашиглан floating point үржүүлэгчийг хийсэн. Floating point операторуудаас үржүүлэгчийг сонгосон шалтгаан нь дизайныг хийхэд арай олон алхамыг дамжиж боловсруулагддаг. Үржүүлэгчийг Modified booth and Wallace tree алгоритмыг ашиглан хийсэн. Энэ паперийн дараахи байдлаар зохион байгуулагдсан: II хэсэгт floating point форматын тухай, III хэсэгт үржүүлэгчийн архитектур дизайн болон түүний ажиллагааны талаар дэлгэрэнгүй тайлбарласан ба IV хэсэгт booth modified ба Wallace tree үржүүлэгчийн тухай тайлбарласан. V симуляцийн үр дүнг үзүүлсэн ба VI хэсэгт дүгнэлтийг бичсэн.

II. FLOATING POINT ТООНЫ ФОРМАТ

Хоёртын (Binary) floating-point тооны форматыг америкийн Цахилгаан болон Электроникийн инженерийн институтээс IEEE 754 гэсэн стандарт болгон баталсан. IEEE 754 стандарт нь дотроо floating point тооны нарийвчлалаас хамаарч 32 битийн өргөнтэй дан нарийвчлалт (single precision), 64 битийн өргөнтэй давхар нарийвчалт (double precision) гэсэн 2 хэсэгт хуваагддаг.



Зураг 1. Floating point тооны дан нарийвчлалт хэлбэр

Дан нарийвчлалт хэлбэрийн хувьд 32 бит тооны сүүлийн 23 бит нь мантис (mantissa), дундах 8 бит нь зэрэг (exponent), хамгийн эхний нэг бит нь тэмдэгийг (sign) илэрхийлдэг. Зураг 1-д floating point тооны дан нарийвчлалт хэлбэрийг үзүүлэв. IEEE 754 стандартын дан нарийвчалт хэлбэрийн хувьд floating point тооны зэргийг илэрхийлэхдээ Excess-127 буюу зэрэг дээр 127 –г нэмэж хадгалдаг. Жишээ болгон -1.1111011 x 2^3 хоёртын floating point тоог IEEE 754 стандартын Excess-127 хэлбэрээр хадгалах болон аравтын тооллын систем рүү хөрвүүлэхийг үзье. IEEE 754 стандартаар хадгалахад мантисын даслалын өмнөх 1 –ийг хадгалдаггүй. Иймээс -1.1111011 x 2^3 тоо 1 10000010 11110110000000000000000 гэж хадгалагдана. IEEE 754 хэлбэрээс аравтын тооллын систем рүү хөрвүүлэн бичье. 1 10000010 11110110000000000000000 тооны тэмдэгийн бит 1 учраас энэ тоо сөрөг, зэрэг нь Excess-127 хэлбэрээр бичигдсэн учраас 127 –г хасч тооцох буюу 10000010 – 01111111 = 00000011, мантис нь $1.1111011_2 = 1x2^0 + 1x2^{-1} + 1x2^{-2} + 1x2^{-3} + 1x2^{-4} + 1x2^{-6} + 1x2^{-7} = 1 + 0.5 + 0.25 + 0.125 + 0.0625 + 0.015625 + 0.0078125 = 1.9609375_{10}$ болно. Иймээс - 1.9609375_{10} x 2^3 = - 15.6875_{10} эсвэл даслалыг баруун тийш 3 шилжүүлэн - 1.1111011_2x2^3 = - 1111.1011_2 = - 15.6875_{10} гэж илэрхийлж болно.

III. FLOATING POINT ТООНЫ ҮРЖҮҮЛЭГЧИЙН АРХИТЕКТУР ДИЗАЙН

Энэ хэсэгт Floating point үржүүлэгчийн архитектур дизайныг тайлбарлана. $a = \alpha * 2^s$ болон $b = \beta * 2^t$ гэсэн 2 floating point тоо өгөгдсөн. Энд α, β нь мантис бөгөөд s, t нь зэрэг юм. a болон b тоог үржүүлбэл $a * b = \alpha * \beta * 2^{s+t}$ хэлбэрээр бичигдэнэ. Энэ томъёоноос харвал a, b хоёр тооны мантисууд хоорондоо үржигдэх ба зэргүүд нэмэгдэнэ. IEEE 754 стандартад мантисын урт 23 бит учраас 23 битийн үржүүлэгч хэрэглэгдэх бөгөөд үржүүлэгчийн архитектур нүсэр бүтэцтэй болох юм. Энэ явдлаас зайлсхийхийн тулд floating point тооны мантисыг 8 битээр 3 хувааж 8 битийн 2 үржүүлэгч ашиглан үржүүлэгчийн дизайныг хийв. Жишээлбэл a = 1,1001100 10001000 01100110 * 2^{s-23} өгөгдсөн гэж үзвэл a тооны мантисыг 8 битийн урттай 3 хувааж дараахи байдлаар илэрхийлж болно. хэрэгживэл

$a_1 = 01001100$, $a_2 = 10001000$, $a_3 = 01100110$

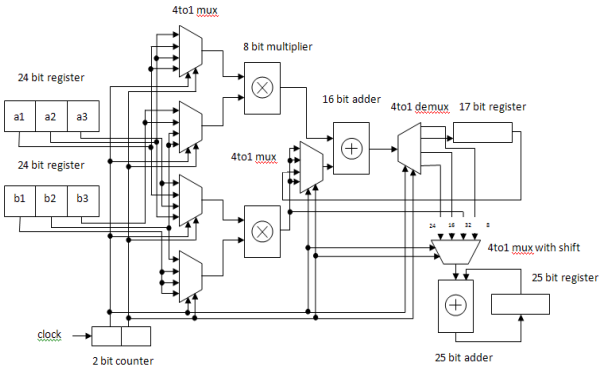
Иймээс a болон b тооны мантисыг 8 битээр 3 хувааж илэрхийлсэн хэлбэрээр бичвэл :

$a = (a_1 * 2^{16} + a_2 * 2^8 + a_3) * 2^{s-23}$

$b = (b_1 * 2^{16} + b_2 * 2^8 + b_3) * 2^{t-23}$

болох ба a,b –ийн үржвэр нь

$a * b = [a_1b_1*2^{32} + (a_1b_2 + a_2b_1)*2^{24} + (a_1b_3 + a_2b_2 + a_3b_1)*2^{16} + (a_2b_3 + a_3b_2)*2^8 + a_3b_3]*2^{s+t-46}$ болох юм. Дээрх томъёоны дагуу ажиллах floating point үржүүлэгчийн дизайныг зохиосон бөгөөд зураг 2 –т үзүүлэв.



Зураг -2. Floating point үржүүлэгчийн архитектур дизайн

a болон b тооны мантисыг 8 битийн урттай 3 хэсэг болгон үржүүлэхэд a_1b_1, a_1b_2, a_2b_1, a_1b_3, a_2b_2, a_3b_1, a_2b_3, a_3b_3 гэсэн нийт 8 үржих үйлдэл хийгдэх ёстой. Мөн бага хаалтан доторхи 4 нэмэх үйлдэл, мантисыг нормчилсний дараа нэмэх 4 нэмэх үйлдэл, зэргүүдийг хооронд нь нэмэх 1 нэмэх үйлдэл, зэргээс 46 –г хасах 1 хасах үйлдэл хэрэглэгдэнэ. Ингэж олон тооны үржүүлэгч болон нэмэгч хэрэглэх нь үржүүлэгчийн дизайнийг нүсэр том болгох юм. Иймээс дизайны хэмжээг багасгахын тулд цөөн тооны логик элементээр хийгддэг тоолуур, демультиплексор, мультиплексоруудыг floating point үржүүлэгчид нэмсэн. Ингэснээр floating point үржүүлэгчийг 8 битийн 2 үржүүлэгч, 16 битийн 1 нэмэгч, 40 битийн 1 нэмэгч, зэргүүдийг хооронд нь нэмэх 1 нэмэгч, 46 –г хасах 1 хасагчтайгаар илүү цомхон бүтэцтэй хийх боломжтой болсон. Тоолуурын гаралтаар a болон b регистрт холбогдсон мультиплексоруудыг удирдах ба тоолуурын тоолох утгаас хамаарч мультиплексоруудын гаралтанд a_1, a_2, a_3, b_1, b_2, b_3 –ийн хослолууд сонгогдох юм. Ингэснээр тоолуурын нэг утганд 2 үржих үйлдэл болон үржүүлэгчээс гарсан үр дүнг хооронд нь нэмэх 1 нэмэх үйлдэл хийгдэнэ. Үүнийг хүснэгт 1 -д үзүүлэв. Тоолуурын 00 утганд floating point үржүүлэгчийн 2^8 зэргийн өмнөх үйлдэл $a_2b_3 + a_3b_2$ хийгдэх бөгөөд демультиплексорын 1-р гаралтаар дамжих нормчилох мултьплексортын 1 –р оролтонд холбогдоно. Нормчилох мултьплексор 2^8 зэргийн сүүлийн 8 утгыг 0 –ээр дүүргэнэ.

Тоолуурын гаралтын утга	Хоёртын зэрэг	Харгалзах үйлдэл
00	8	$a_2b_3 + a_3b_2$
01	16	$a_2b_2 + a_3b_1$
10	32, 16	a_1b_1 , a_1b_3
11	24	$a_1b_2 + a_2b_1$

Хүснэгт 1.

Тоолуурын 01 утганд floating point үржүүлэгчийн 2^{16} зэргийн өмнөх үйлдэл $a_2b_2 + a_3b_1$ хийгдэх бөгөөд демультиплексорын 2-р гаралтаар дамжин 17 битийн регистрт хадгалагдах ба тоолуурын дараагийн утганд a_1b_3 дээр нэмэгдэнэ.

Тоолуурын 10 утганд 2^{32} болон 2^{16} зэргийн өмнөх үйлдэл a_1b_1 , a_1b_3 хийгдэх бөгөөд 2^{32} нь нормчилох мултьплексортын 2 –р оролтонд шууд холбогдон нормчилогдоно. Нормчилох мултьплексор 2^{32} –ийн сүүлийн 32 битийг 0 –ээр дүүргэнэ.

Тоолуурын 11 утганд floating point үржүүлэгчийн 2^{24} зэргийн өмнөх үйлдэл $a_1b_2 + a_2b_1$ хийгдэх бөгөөд демультиплексорын 4-р гаралтаар дамжих нормчилох мултьплексортын 4 –р оролтонд холбогдоно. Нормчилох мултьплексор 2^{24} –ийн сүүлчийн 24 утгыг 0 –ээр дүүргэнэ.

a, b гэсэн 23 бит мантис бүхий 2 тоог хооронд нь үржүүлэхэд үржвэрийн мантис 46 бит болно. Үржвэрийн мантисаас ахлах 23 битийг авч үлдсэн хэсгийг тооцооллоос хасна. a, b тоог 8 битээр 3 хэсэгт хуваан үржүүлэн гарсан үр дүнг хооронд нь нэмэх тул битүүдийг нормчлох хэрэгтэй. 8 битийн өргөнтэй $a_{(i)}$ болон $b_{(i)}$ тооны хамгийн их утга 11111111 бөгөөд үржвэрүүдийн хамгийн их утга 11111111 x 11111111 = 1111111000000001 болно. Иймээс 2 үржвэр хооронд нь нэмж байгаа 2^8 болон 2^{24} зэргийн хувьд хамгийн их утга 1111111000000001 + 1111111000000001 = 1111111000000010 болно. Харин 3 үржвэр хооронд нь нэмж байгаа 2^{16} зэргийн хамгийн их утга 1111111000000010 + 1111111000000001 = 1111111000000011 болох юм. 2 –ийн зэргээс хамааруулан гарсан үр дүнгүүдийг 48 битийн регистрт зүүн болон баруун тийш шилжүүлэн үлдсэн хэсгүүдийг 0 –ээр дүүргэхийг дараахи хүснэгтэд үзүүлэв. Энэ үйлдлийг нормчлох мультиплексор гүйцэтгэнэ. Хүснэгт 2 -д тод хараар бичсэн хэсэг 2 –ийн зэргийн хувьд хамгийн их утга ба бүдэг хэсэг нь 0 –ээр дүүрсэн хэсэг юм. Харин саарал

дөрвөлжин дотор тод хараар бичсэн хэсэг нь урагш орон хэтрэх боломжтой гэдгийг үзүүлэв.

Хоёрын зэрэг	Нормчилсон утга
32	**11111110 00000001** 00000000 00000000 00000000 00000000
24	**1 11111110 00000010** 00000000 00000000 00000000
16	**1 11111110 00000011** 00000000 00000000
8	**1 11111110 00000010** 00000000
0	**1 11111110 00000001**

Хүснэгт 2. Нормчилох мультиплексор 2 –ийн зэргээс хамаарч 0 –ээр дүүргэх байгаа байдал

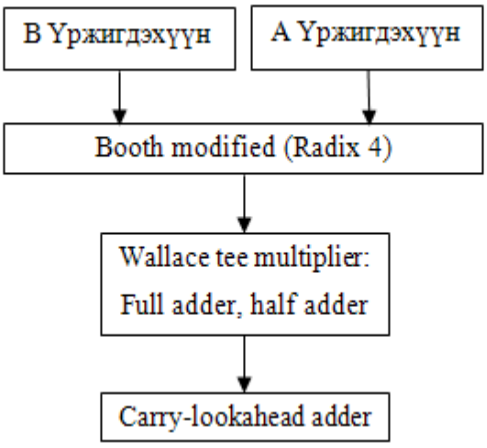
IV. Booth Modified and Wallace Tree үржүүлэгч

A. Эхлэл

Найман битийн үржүүлэгч схемийг гейтүүдийн төвшинд програмчлан угсрах зорилготой байгаа. Шаардлага нь хамгийн хурдан ажиллаж байх хэрэгтэй. Иймээс нийт үржүүлэгч схемүүдийн дундаас Booth modified and Wallace tree multiplier нь хамгийн хурдан ажиллаж буй схем тул сонгож авч 8 битийн болгон загварчлав.

B. Хоёр тоог үржүүлэх Booth алгоритм

Цифр тус бүрээр үржүүлэхэд гарах найман нэмэгдэхүүнийг багасгахын тулд Modified Booth алгоритмыг ашиглаж байгаа бөгөөд В үржигдэхүүний эхний тэмдэгийн утга тэгийг өгч ард нь нэмэлт тэгийг залгаж гурав гурван битээр таслан А үржигдэхүүнийг үржүүлэж нийт таван нэмэгдэхүүнийг гаргана.



Зураг 3. Modified booth алгоритм

Тэдгээр нь Radix 4 гэсэн гурав суурьтай 0, ±A, ±2A, ±3A гэсэн сонголтуудаас үүсэнэ. Үржигдэхүүнүүд нь Modified Wallace tree аргаар нэгтгэгдэнэ.

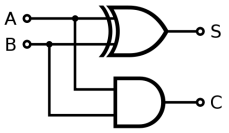
Гурав гурваар нь В үржигдэхүүнийг хувааж 5 хэсэг болгоно. Хамгийн бага 3 бит нь нэгдүгээр хэсэг, хоёрдугаар хэсэг нь нэгдүгээр хэсгийн ахлах бит ба В үржүүлэгчийн 4, 5 дугаар битийг агуулсан байна. Гэх мэт цааш үргэлжилнэ. Дээрх хэсэг тус бүрийг А үржигдэхүүнтэй үржүүлэж таван нэмэгдэхүүнүүдийг гаргана. Үүнийг хүснэгт 3-д үзүүлэв.

RADIX-4 BOOTH ENCODER MULTIPLIER BITS	Гарах нэмэгдэхүүнүүд
000	0
001	+1A
010	+2A
011	+3A
100	-3A
101	-2A
110	-1A
111	0

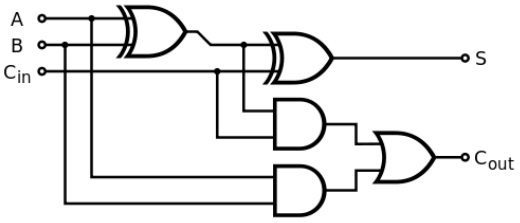
Хүснэгт 3.

C. Wallace Tree үржүүлэгч

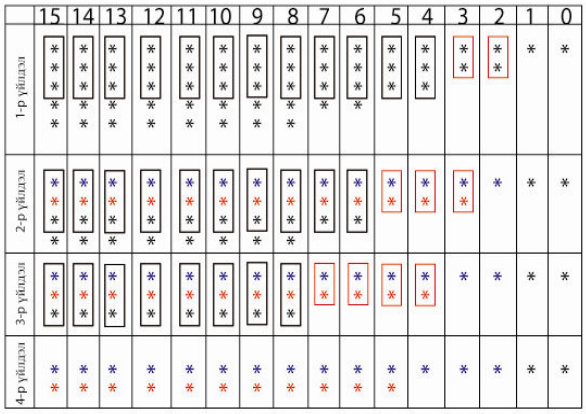
Энэхүү нэмэгч нь хоёроос дээш нэмэгдхүүнийг нэгэн үйлдэлээр хийх боломжийг олгоно. Үүнийг хийхийн тулд бүтэн нэмэгч ба хагас нэмэгчүүдийг ашиглана.Хагас нэмэгч болон бүтэн нэмэгчийн схемийг зураг 4 болон зураг 5 –д үзүүлэв



Зураг 4. Хагас нэмэгчийн схем



Зураг 5. Бүтэн нэмэгчийн схем

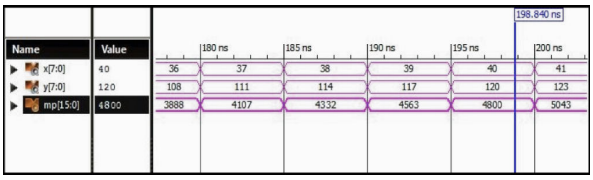


Хүснэгт 4.

Хүснэгт 4 –ийн одууд нь нэг битийн утгууд, цэнхэр улаан од нь карри бит ба үлдсэн нь тухайн оронгийн бит юм. Хоёр одыг хүрээлсэн тэгш өнцөгтөөр хагас нэмэгчийг, гурван одыг хүрээлсэн тэгш өнцөгтөөр бүтэн нэмэгчийг дүрсэлсэн болно. Нийт таван нэмэгдхүүн гарч ирэх бөгөөд тэдгээрийн урт нь 16 бит байна. Хүснэгт 4 –д харуулсанаар I, II, III мөрүүдэд Wallace Tree үржүүлэгчийн бүтэн ба хагас нэмэгч ашиглан нэмэж байгаа алхамууд болно. IV нь мөр нь Carry Lookahead нэмэгчээр эцсийн утгыг гаргаж байна.

V. СИМУЛЯЦЫН ҮР ДҮН

Үржүүлэгчийг симуляц хийсэн Modelsim симуляторын үр дүнг зураг 6 -д үзүүлэв.



Зураг 6. Modified booth and Wallace tree үржүүлэгчийн симуляцийн үр дүн

Floating point үржүүлэгчийг Spartan 3S500E программчлагддаг логик чипэнд хир зай эзэлсэнийг хүснэгт 5 –д үзүүлэв.

mbooth1 Project Status (04/12/2013 - 23:59:44)			
Project File:	M_booth2.xise	**Parser Errors:**	No Errors
Module Name:	mbooth1	**Implementation State:**	Synthesized
Target Device:	xa3s500e-4cpg132	**• Errors:**	No Errors
Product Version:	ISE 14.2	**• Warnings:**	13 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	79	4656	1%
Number of 4 input LUTs	144	9312	1%
Number of bonded IOBs	32	92	34%

Хүснэгт 5. Modified booth and Wallace tree үржүүлэгчийн синтезийн үр дүн

VI. ДҮГНЭЛТ

Дан нарийвчлалт 32 битийн floating point үржүүлэгчийн архитектур дизайныг гарган, түүний VHDL загварыг туршиж үзлээ. Floating point үржүүлэгчийн 8 битийн 2 үржигчийг Booth Modified and Wallace Tree алгоритмыг ашиглаж хийсэн. Энэхүү хосолмол үржигч нь энгийн үржигчтэй харьцуулхад хангалттай хурдан ажилладаг. Учир нь энгийн үржигч нь 8 битийн хувьд 7 Carry look ahead нэмэгч ашигладаг, харин энэхүү алгоритм нь 1 Carry Look ahead ашиглаж байгаагаараа давуу талтай юм. Floating point үржүүлэгчийг Spartan 3S500E программчлагддаг логик чип ашиглан синтезлэсэн. Синтезийн үр дүнд Modified booth and Wallace tree үржүүлэгч FPGA –ийн 4656 slice –ийн 79 буюу 1%, 9312 4 оролттой LUT –ийн 144 буюу 1%, 92 оролт гаралтын блокын 32 буюу 34% -ийг тус тус эзлэж байна.

НОМ ЗҮЙ

[1] J.Detrey, F. de Dinechin, “Floating point trigonometric funcitons for FPGA,” Field-Programmable logic and Application, IEEE, 2007, p.p 29-34.

[2] Loucas louca, Tod A. Cook, William H.Johnson, “Implementation of IEEE Single precision Floating point Addition and Multiplication on FPGA,” IEEE, September 1996.

[3] Pardeep Sharma, Ajay Pal Singh, “Implementation of Floating Point Multiplier on Reconfigurable Hardware and Study its Effect on 4 input LUT’s,” International Journal of Advanced Research in Computer Science and Software engineering, July 2012, vol 2, p.p 244-248.

[4] IEEE Standard for Floating-Point Arithmetic, IEEE Computer Society Std., 2008.