2001년도 대한전기학회/대한전자공학회 시스템및제어분야 합동 추계학술대회 논문집 (2001.11.24)

IEEE 754-1985 단정도 부동 소수점 연산용 나눗셈기 설계

박안수*, 정태상
중앙대학교 전자전기공학부

Design of a Floating-Point Divider for IEEE 754-1985 Single-Precision Operations

ANN-SOO PARK*, Tea-Sang Chung
School of Electrical and Electronics Eng. Chung-Ang Univ

Abstract - This paper presents a design of a divide unit supporting IEEE-754 floating point standard single-precision with 32-bit word length. Its functions have been verified with ALTERA MAX PLUS II tool. For a high-speed division operation, the radix-4 non-restoring algorithm has been applied and CLA(carry-look -ahead) adders has been used in order to improve the area efficiency and the speed of performance for the fraction division part. The prevention of the speed decrement of operations due to clocking has been achieved by taking advantage of combinational logic. A quotient select block which is very complicated and significant in the high-radix part was designed by using P-D plot in order to select the fast and accurate quotient. Also, we designed all division steps with Gate-level which visualize the operations and delay time.

1. 서 론

최근 컴퓨터 환경이나 DSP, 그래픽 등의 분야에서 복잡한 수식을 빠르게 계산할 수 있는 unit의 필요성이 증가함에 따라 사칙연산 중 상대적으로 가장 느린 나눗셈의 속도 향상을 위한 나눗셈 전용의 Division unit 설계하였다. 빠른 나눗셈을 하기 위해 Clocking인한 상대적으로 속도를 저하시키는 순차회로(sequential logic)을 사용하는 방법대신 모든 과정을 완전히 조합회로(combinational logic)로 구성하였다. 많은 부동 소수점을 표현하기 위한 표준 규약 중에 IEEE 754-1985 형식의 32-bit single precision 방법을 채택했으며, 개발 툴로는 ALTELA사의 MAX PLUSII를 사용하고 있다. 본 논문의 목적은 새로운 이론을 정립하고 그것을 기술해 나가는 것이 아니라 기존의 많이 산재해 있는 이론들은 모아서 서술하고, 하드웨어로 구현되었을 때 가장 최적의 결과를 만들어 낼 수 있도록 그런 이론들을 접목하였다. 또한 IEEE 754 부동 소수점 형식에 맞는 나눗셈 전용의 하드웨어를 알고리즘과 내부구조를 달리 설계하여 속도와 성능에 대한 비교를 해 본다.

2. 본 론

2.1 부동소수점 나눗셈

IEEE 754 단정도 포맷은 다음과 같다.

31	30 ... 23	22 ... 0
S	E(exponent)	M(mantissa , significand or fraction)
1비트	8비트	23비트

위의 32-bit 데이터는

$F = (-1)^s \times (1 + \text{유효자리}) \times 2^{(\text{지수}-127)}$로 표현된다.

위와 같이 표현된 32-bit 크기의 두 개의 피연산자(제수와 피제수)를 이용하여 나눗셈을 크게 4단계로 구성되어있다.

1. $S_3 = S_1 \oplus S_2$
2. $E_3 = E_1 - E_2 + 127$
3. $M_3 = \frac{M_1}{M_2}$
4. Post-Normalization and Rounding 그 외의 예외처리 부분을 둔다(그림1)

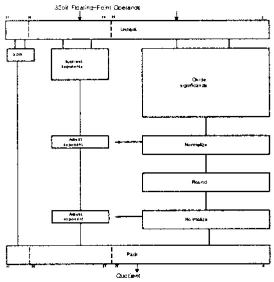


그림1 부동소수점 나눗셈기 블록 다이어그램

나눗셈 연산에서 가장 중요한 것은 얼마나 빠르게 몫과 나머지를 구하느냐에 달려있다. 몫을 구하는 방법은 크게 복원 나눗셈법과 비복원 나눗셈법이 있다. 비복원 나눗셈 방식은 부분나머지의 복원 과정을 피하고 나머지가 음수인 경우, 바로 더해서 복원시키는 대신 그냥 그 다음 단계로 넘어가서 자리 이동된 나머지에 제수를 더하는 것이다. 이 논문에서는 비복원 방법을 사용한다.

2.2 가수(Mantissa)부분의 나눗셈기

부동소수점 나눗셈기에서는 실제적인 나눗셈 연산을 실행하는 가수부분의 가장 중요하다. 그러므로 어떤 나눗셈 알고리즘을 적절히 선택하느냐에 나눗셈기의 성능이 결정지어진다.

2.2.1 Radix-2 비복원 나눗셈

부동소수점 가수부분의 피제수와 제수의 범위는 [1,2)이다. 그러나 실제로 로직상으로 구현함에 있어서 MSB bit을 부호비트로 사용하는 SD(Signed Digit)수체계를 쓰기 때문에 두 오퍼랜드의 hidden bit = 1 은 두 피연산자를 음수로 인식하게 된다. 그리하여 피연산자를 양수로 만들어 주기 위해 $N_0 = D_0 = 0$, $N_1 = D_1 = 1$ 으로 고정시키는 작업이 필요하다. 그런 다음 입력으로 23-bit 제수, 피제수 데이터 값을 받아들인다.

$$\text{피제수} \quad N = N_0 . N_1 \ N_2 N_3 \cdots N_n$$

$$\text{제수} \quad D = D_0 . \underbrace{D_1 \ D_2 D_3 \cdots D_n}_{\text{입력데이타 23bit}}$$

$$\text{부분나머지} \quad R = R_0 . R_1 R_2 R_3 \cdots R_n$$

$$\text{몫} \quad Q = Q_0 . Q_1 Q_2 Q_3 \cdots Q_i$$

실제적으로 $\frac{1}{2} \le N, D < 1$ 값이 연산하게 되므로 몫은 양수이고 $\frac{1}{2} < Q < 2$ 범위에 있게 된다. 양수인 피제수와 제수, 몫 또한 양수 값을 가지게 되지만 부분 나머지는 양수 또는 음수가 될 수 있다. 그러므로 2의 보수법으로 표현하여 사용한다. 또한 Radix-2 나눗셈기를

만듦에 있어서 2의 보수법의 사용은 하드웨어 설계상의 여러 가지 장점을 제공해 준다. 첫째는 덧셈기만으로 뺄셈과 덧셈을 할 수 있다. 둘째는 각 단계별 leftmost carry-out bit을 그대로 몫의 비트로 사용할 수 있는 장점이 있다.

$$2^n = C_{out} = q_i = \begin{cases} 1 & r_i - D \ge 0 \\ 0 & r_i - D < 0 \end{cases}$$

2.2.1.1 Cellular array divider using CAS cells

Cellular array divider의 가장 기본적인 형태는 CAS (controlled add/subtraction) cell을 이용하여 연속적인 배열 구조(그림2)로 구현한다[1]. 이 CAS cell은 컨트롤 신호 T에 따라 덧셈과 뺄셈을 할 수 있도록 되어 있다

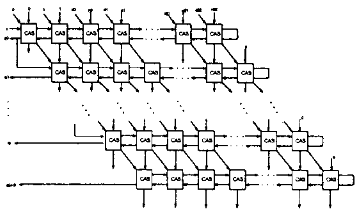


그림2 Cellular array divider using CAS cell[1][2]

그러나 이 나눗셈기는 전 부분에서의 ripple의 발생하기 때문에 빠른 목적의 나눗셈기에는 맞지 않다.

2.2.1.2 Carry-Look-Ahead cellular array divider

CAS-cell을 이용하여 만든 나눗셈기의 전 과정ripple delay를 줄이고자 CSA(carry save adder)과 CLA의 알고리즘을 사용하여 array divider(그림3)을 다시 설계하였다[1]. 이 나눗셈기는 CSA의 과정을 A-cell과 S-cell을 이용하여 구현한다.

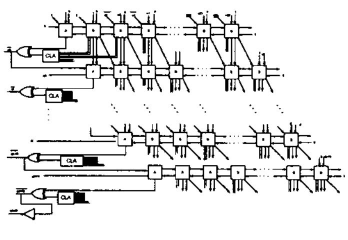


그림3 A carry lookahead array divider using CSA adder[1][2]

2.2.1.3 Divider using CLA adder

Cellular array 방식의 divider는 비트의 크기가 커질수록 MAX PLUS Ⅱtool을 이용하여 구현하고 검증하기에 많은 단점을 가지고 있다. 그것은 flex칩 내부적 구조 때문이다. Alter flex계열의 칩들은 내부가 LAB 구조로 되어 있는데 이것은 여러 개의 램으로 구성되어 있다. cell단위로 조각조각 나누어진 덧셈기를 이용한다면 피연산자의 크기가 커질수록 개당 LAB을 차지하는 비율이 비효율적이며 컴파일 시간과 차지하는 면적도 지수적으로 증가하게 된다. 그러므로 cellular array 방식을 이용하되 속도와 칩 사용면적에 적합한 덧셈기 설계가 필요하다. 이 논문에서는 CLA(Carry Look-Ahead) 덧셈기를 사용하여 divider를 다시 설계하였다. CLA는 직접회로를 만드는데 적용 할 수 있는 간단화와 모듈 방식에 기인해서 가장 인기가 많은 덧셈기이다. 먼저 25-bit 크기의 CLA는 선택한 칩의 구조에 효율적인 크키-bit slice CLA를 여러개 연결하여 구현한다.(그림4)

2.2.2 Radix-4 비복원 나눗셈

Radix-2에서보다 속도를 향상시키기 위해서 증가한 진수($\beta=\gamma=4$이고 $a=2$)을 선택하였다. Radix-4 나눗셈기에서는 각 단계별로 2bit 몫을 구할 수 있는데 radix-2 일때처럼 각 단계별 leftmost carry-out bit

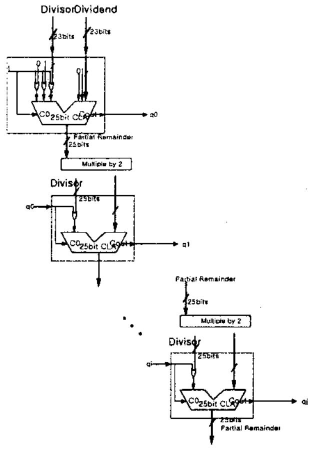


그림4 Radix-2 floating-point divider using CLA adder for he fraction

이나 부분나머지의 부호만으로 2bit의 몫을 구할 수 없다. 그러므로 부분나머지의 범위에 따라 몫의 값을 정해주는 Quotient Selection Logic이 따로 필요하다. Raidx-4에서는 몫을 선택함에 있어서 redundancy가 존재한다. 이처럼 부분나머지의 범위에 따라 올바른 몫을 결정하기 위한 Quotient Selection Logic block을 어떻게 설계할 것인가가 Radix-4 나눗셈기에서 가장 중요한 부분을 차지한다. radix-2와는 다르게 hidden bit부분 $N_0 = D_0 = 0$ $N_1 = D_1 = 0$ $N_2 = D_2 = 0$ $N_3 = D_3 = 1$으로 고정하였는데 이는 각 단계별로 몫을 구할 때 쓰이는 비교 연산을 용이하게 사용하기 위해서 이다

피제수 $N = N_0 .\ N_1 N_2 N_3\ N_4 \cdots\ N_n$

제수 $D = D_0 .\ \underbrace{D_1 D_2 D_3\ D_4 \cdots\ D_n}_{}$

입력데이타 23bit

2.2.2.1 Divider using CLA

Radix-4 나눗셈기의 전체블록도는 그림5와 같다.

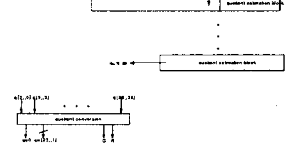


그림 5 Radix-4 floating-point divider using CLA for the fraction

2.2.2.2 Quotient encoding

G.S Tayler의 P-D plot[3]을 이용하여 Quotient select block에서 얻은 quotient digit는 3bit으로 인코딩한다. 이는 radix-4에서 생기는 몫의 digit값을 음수와 양수, 크기별로 구별하여 적절한 제수값을 선택하여 피연산자로 이용하기 위해서이다. 또한 마지막에 여러 단계에서 구한 몫의 digit값들을 binary 결과값으로

변환시켜주기 쉽게 하기 위함이다. qs는 몫의 부호비트이고 qj와 qk는 몫의 절대적 크기를 나타낸다. 이것은 기존의 sequential 나눗셈기에서 음수와 양수의 몫의 각 레지스터에 두어 저장하는 작업대신 부호비트를 두어 구별하게 한 것이다

Quotient digit	qs	qj	qk
+2	0	1	0
+1	0	0	1
0	0	0	0
-1	1	0	1
-2	1	1	0

표1 Quotient digit encoding

2.2.2.4 Divisor select block

몫이 정해지면 다음 단계에 행해질 operation도 결정된다. 표2는 몫의 값에 따라 선택할 제수와의 관계를 나타낸다.

Quotient digit	Next operation
+2	Sub 2D
+1	Sub D
0	Add 0
-1	Add D
-2	Add 2D

표2 Multiple of divisor select

몫에서 정한 제수 값은 $-2D, -D, 0, D, 2D$ 이다. 여기서 $-2D$, $2D$는 $-D$, D를 한 비트 왼쪽으로 이동한 값이므로 MUX와 shifter를 동시에 사용할 수 있는 블록을 만들어 구성하여 입력을 $-D$, D 만으로 Radix-4에서 사용하는 제수값을 선택한다.

2.2.2.5 Quotient conversion

몫을 다 구한 다음 Radix-4에서는 각 단계별 몫이 값이 digit 값으로 얻어지기 때문에 binary로 바꿔주는 별로의 conversion logic이 필요하다. 일반적으로 구해진 몫들은 각각 음수와 양수로 구분하여 레지스터에 저장하고 binary 형태로 바꾸기 위해 마지막에 아래의 식과 같이 구한다.

$$PR < 0 \quad Q_m = Q_{pos} - Q_{neg} - 1$$

$$PR > 0 \quad Q_m = Q_{pos} - Q_{neg}$$

Q_{pos}: 양수값만 저장하는 register

Q_{neg}: 음수값만 저장하는 regiter

PR :final partial remainder

그러나 이 방법은 몫의 부호에 따라 따로 저장하는 별도의 레지스터가 필요하게 되므로 빠른 나눗셈을 구하기 위해 모든 나눗셈 과정을 조합회로로 구현하는 방법에서는 구현하기 어렵다.
위의 식을 2의 보수법을 사용하여 계산하면 다음과 같은 식으로 다시 쓸 수 있다.

$$PR < 0 \quad Q_m = Q_{pos} - Q_{neg} - 1 = Q_{pos} + (\overline{Q_{neg}} + ulp) - 1$$
$$= Q_{pos} + \overline{Q_{neg}}$$

$$PR > 0 \quad Q_m = Q_{pos} - Q_{neg} = Q_{pos} + (\overline{Q_{neg}} + ulp)$$

이는 표1에서 구한 인코딩한 값을 이용하여 별도의 레지스터를 사용하지 않고서도 조합회로로 구현할 수 있게 한다. 그림6.c는 표1에서 인코딩한 3bit의 몫을 binary로 바꿔주는 역할을 하는 디코더이다.

2.2.3 Quotient correction block

비복원 나눗셈기를 만들 때 한 가지 고려해 주어야 할 부분이 있다. 그것은 마지막 부분나머지가 음수일 때 몫을 수정해 주는 과정이다. 부동 소수점 나눗셈에서는 부호를 처리해주는 부분이 따로 존재한다.

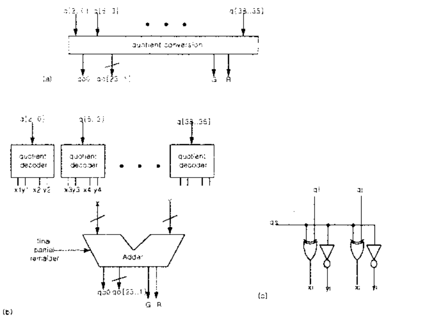


그림6 Quotient conversion block

실제로 나눗셈 연산이 실행되는 유효숫자부분은 제수와 피제수를 양수값을 가진다. 그러므로 나머지와 몫의 값도 양수이다. 마지막 나머지의 부호가 음수라는 것은 그 단계에서 구한 몫이 잘못된 값이라는 것을 의미한다.

quotient #	0	1	2	………	22	23	24	25
	MSB							LSB
	q_0	q_1	q_2	………	q_{22}	q_{23}	q_{24}	q_{25}
	×	×	×	………	×	×	×	0
	×	×	×	………	×	×	×	1

즉 $q_{25} = 0$ 일 때 $Q_{correct} = Q_m - ulp = Q_m - 1$ 으로 몫을 수정해 준다.

2.3 정밀도 고려

2.3.1 Guard bit

연산 후 결과값의 좀 더 정확한 값을 구하기 위해 3개의 추가 비트을 사용하는데 이것을 guard bit들(G,R,S bit)이라고 한다. 첫 번째 G-bit은 post-normalization으로 사용이 되고, 두 번째 R-bit은 round-to-nearest 방법을 채택했다면, 반올림에 필요하게 된다. 나눗셈 연산에선 나누어 떨어지는 경우를 제외하고 생성될 수 있는 모든 여분의 digit들은 구한다는 것은 의미가 없다. 그러므로 마지막 부분나머지 값의 모든 bit들을 OR하여 sticky bit으로 대신 사용한다.

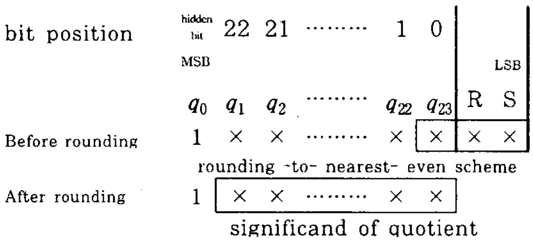
2.3.2 정규화

각기 다른 알고리즘으로 구한 몫들이지만 가수부분 나눗셈 연산이 마친 후 binary 몫은 두 가지 일정한 패턴을 가진다. 또한 G, R, S 값들도 정해져 나온다

quotient#	0	1	2	………	22	23	24	25	26
	MSB								LSB
	q_0	q_1	q_2	………	q_{22}	q_{23}	G	R	S
$N < D$	0	1	×	………	×	×	×	×	×
$N \geq D$	1	×	×	………	×	×	×	×	×

정규화(post-normalize)과정은 이 두 가지 패턴으로 나온 몫들을 IEEE-754 규격으로 맞추어 주는 작업을 하는 것이다. 만약 MSB가 1이면, 그때 가수는 $q_0 \sim q_{23}$bit을 취하고, 지수의 값은 증가하지 않는다. 그렇지 않고, MSB가 0면, 그때는 가수로 $q_1 \sim q_{24}$bit들의 취하고, 지수의 값은 감소 시켜준다.(underflow경우). 주의할 점은 반올림 자체가 MSB를 1로 만들 수 있다는 것이다. 이것은 반올림 후의 overflow의 경우이다. 이 부분은 뒷부분에서 처리하였다.

2.3.3 부동소수점 반올림



2.4 지수부분의 하드웨어 수행

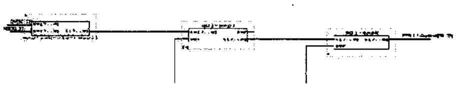


그림7 Exponent부분의 전체 블록도

나눗셈의 연산에서 지수들간의 연산은 뺄셈의 과정이다. 지수부분의 전체적인 하드웨어 블록도는그림7과 같다. 지수 부분의 첫 블록(exp-sub-cla-result)은 $e = e_1 - e_2 + 127$ 을 구현하기 위해 있는 부분이며, 두 번째 블록(adj-exp1)은 가수부분에서under-flow 신호에 따라서 지수부분의 감소 여부를 , 세 번째 블록(adj-exp2)은 반올림 한후 추가의 지수부분 증가 여부을 결정하여야 하므로, 서로 연결되어 지수 값을 결정한다.

3. 기능 검증 및 비교분석

이상에서 알아본 부동 소수점 연산기의 나눗셈기의 기능 검증을 ALTERA사의 MAX-PLUS Ⅱ의 simulator로 테스트 해 보았다(그림8)

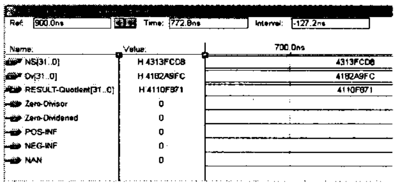


그림8 부동 소수점나눗셈기 구현 결과

모든 나눗셈기의 결과 값은 같았으나 성능 비교면에서는 아주 다르게 나타났다. 표3을 보면 radix-2 CSA divider나와 있지 않다. 이것은 cellular방식이 실제 사용 gate들의 수가 많지 않으나 MAX-칩에서 비효율적으로 큰 부피를 차지하게 컴파일 되기 때문에 10만 gate level에서의 설계가 불가능하였다. 각각 다른 divide significands의 설계에 따라 장단점이 있다. Radix-2로 구현하면 단계별 left-carry-out bit으로 쉽게 몫을 구할 수 있지만 단계별로 한 bit의 몫만 구할 수밖에 없어 동작 속도가 느린 반면, Radix-4로 구현하면 단계별 몫을 2bit으로 증가시켜 구할 수 있으므로 Radix-2일 때 보다 절반단계로 줄일 수 있다. 그러나 몫을 구하는 추가 로직이 필요하고 복잡하다. Radix-4 비복원 나눗셈 방식에 CLA-덧셈기를 사용하고, 몫을 결정하는 부분을 P-D plot을 이용하는 설계방식을 채택했을 때 속도와 면적 두 부분에서 성능이 우수하였다.

Divider 종류		Longest Path Delay	칩 이용 면적율	이용 LCs
Radix-2	CAS cell	1771.8 ns	31%	1585
	CSA	-	-	-
	CLA	1199.9 ns	61%	3054
Radix-4	CLA	827.8 ns	63%	3146

표3 EPF10K100ARC 240-1에서의 비교

[참 고 문 헌]

[1] Maurus Cappa and V.Carl hamacher "An Augmented Iterative Array for high-speed binary Division" IEEE Transactions on Computers, pp172~175, Feb 1973
[2] Kai Hwang, "Computer Arithmetic", John Wiley Sons. Inc, 1979
[3] G .S. Tayler, "Compatible hardware for division and square root" Proc. of 5th Symp. on Computer Arithmetic, pp.127~134, 1981
[4] Daniel E. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders" IEEE Transactions on Computers, Vol.C17 No10 pp 925~934, Oct 1968
[5] James E. Robertson, " A New Class of Digital Division Methods" IRE Transactions on Electronic Computers Sep. 1958 pp218~222
[6] Hosahalli R.Srinivas, "Radix 2 Division with Over-Redundant Quotient Selection " IEEE Transactions on Computers, Vol.46, No1, Jan 1997, pp 85~92
[7] Jan Fandrianto, " Algorithm for High Speed shared Radix 4 Division and Radix 4 square-root" IEEE Transactions on Computers , 1987, pp73~79
[8] Israel Koren, "Computer Arithmetic Algorithms",John Wiley &Sons. Inc, 1993
[9] 박명순, 김병순 공역, "컴퓨터 구조 및 설계" 사이텍미디어
[10] Dharma P. Agrawal, "High-Speed Arithmetic Arrays" IEEE Transactions on Computers, Vol. C28, No3, Mar 1979 , pp 215~234
[11] "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Std 754-1985, 1985
[12] 이용석, "마이크로 프로세서 부동 소수점 연산기의 구조와 설계", IDEC 교육 자료, 1998,2
[13] 정태상, "32-bit Floating Point Number Representation", 강의 노트
[14] Behrooz Parhami, "Computer Arithmetic - Algorithms and Hardware Designs " Oxford University Press ,2000
[15] Tomas Lang, "Division and square root - Digit-recurrence algorithms and implementations"
[16] Neil Burgess and Ted Williams, "Choices of Operand Truncation in the SRT Division Algorithm" IEEE Transactions on Computers , Vol.44, No7 ,Jul 1995, pp 933~938
[17] Harris, stuart F. Oberman, and Mark A. Horowitz "SRT division architectures and implementations"
[18] Chris Rowen, Mark Johnson, Paul Ries "The MIPS R3010 floating-point coprocessor" IEEE MICRO, 1988, JUNE pp53~62
[19] Hosahalli R. Srinivas, Keshab K. Parhi, "Radix 2 division with over-redundant quotient selection" IEEE Transactions on Computers, vol.46 No1, Jan 1997, pp85~92
[20] Eric M. Schwarz , Michael J. Flynn "High-radix nonrestoring division " IEEE Transactions on Computers, Vol.42, No10, Oct 1993, pp 1124~1246
[21] 박안수 "IEEE 754-1985 단정도 부동 소수점 연산용 나눗셈기 설계" ,June 2001

***논문 관련 문의 pas93@hanmail.net