論文97-34C-9-4

디지탈 신호처리용 고정 소수점 최적화 유틸리티

(Fixed-point Optimization Utility for Digital Signal Processing Programs)

金 時 鉉 *, 成 元 鎔 **

(Seehyun Kim and Wonyong Sung)

C 또는 C++ 언어로 기술된 디지탈 신호처리 알고리듬의 빠른 고정 소수점 성능 평가와 최적화를 위한 소프트웨어가 개발되었다. 이 유틸리티는 범위 추정기와 고정소수점 모의 실험기로 구성되어 있다. 전자는 부동 소수점 변수의 범위(range)를 추정하고, 이에 의하여 스케일링 정보를 알려준다. 후자는 부동 소수점 프로그램으로부터 고정 소수점 프로그램을 생성하여, 유한 단어 길이 효과를 모의 실험으로 측정할 수 있게 해 준다. C++ 언어의 연산자 오버로딩(operator overloading) 특성을 이용하기 때문에 범위 추정과 고정 소수점 모의 실험이 단지 원래 프로그램의 선언(declaration) 부분만 바꾸어 줌으로써 가능하다. 이 유틸리티는 모의 실험에 의해 범위 추정과 고정소수점 성능을 측정하므로 비선형이나 시변(time-varying), 또는 다중율(multi-rate)과 다차원 신호처리와 같은 거의 모든 디지탈 신호 처리 알고리듬에 적용할 수 있다. 또한, 이 유틸리티는 서로 다른 구현 구조에 따른 고정소수점 특성을 비교할 목적으로도 사용될 수 있다.

Abstract

Fixed-point optimization utility software that can aid scaling and wordlength determination of digital signal processing algorithms written in C or C++ language is developed. This utility consists of two programs: the range estimator and the fixed-point simulator. The former estimates the ranges of floating-point variables for automatic scaling purpose, and the latter translates floating-point programs into fixed-point equivalents for evaluating the fixed-point performance by simulation. By exploiting the operator overloading characteristics of C++ language, the range estimation and the fixed-point simulation can be conducted just by modifying the variable declaration of the original program. This utility is easily applicable to nearly all types of digital signal processing programs including non-linear, time-varying, multi-rate, and multi-dimensional signal processing algorithms. In addition, this software can be used for comparing the fixed-point characteristics of different implementation architectures.

I. 서 론

디지탈 신호처리 알고리듬의 초기 개발 단계에서는 대부분 부동 소수점 연산 방식을 사용하지만, VLSI나 고정 소수점 디지탈 신호처리기를 이용하여 구현하고자 할 때에는 하드웨어 비용이나 속도 측면에서 고정 소수점 연산을 사용하게 된다. 그러나, 모든 내부 신호가 적절히 스케일링되지 않거나 충분한 단어길이가 할당되지 않는다면, 고정 소수점 연산을 이용하여 구현된 시스템은 오버플로우나 양자화 잡음으로 인한 과도한 유한 단어 효과 (finite wordlength effects) 를 유발할 수 있다[1]. 선형 디지탈 필터나 몇몇 특정한 알고리듬의 경우, 스케일링이나 단어길이 최적화를 위한 유한 단어 효과 분석 방법이 보고되었지만[2,3,4],

* 正會員, LG綜合技術員 이노베이션 센터
(LG Corporate Institute of Technology)
** 正會員, 서울大學校 電氣工學部
(School of Electrical Eng., Seoul National University)
接受日字:1996年12月4日, 수정완료일:1997年9月2日

이 분석 방법을 비선형 알고리듬을 포함한 일반적인 디지탈 신호처리 시스템의 고정 소수점 구현에 활용하기는 매우 어렵다. 따라서, 구현에 앞서 고정 소수점 연산을 사용하여 디지탈 신호처리 알고리듬을 충분히 모의 실험하는 것이 필요하다. 그러나, 비선형이나 시변 블럭을 포함하는 복잡한 신호처리 알고리듬의 경우 적절한 스케일링 정보를 찾아 고정 소수점 모의 실험 모델을 만드는 일은 일반적으로 많은 노력을 필요로 한다.

최근 수년 동안에는 Silage[5] 나 SIGNAL[6], 그리고 DSP/C[7] 와 같이 디지탈 신호처리 알고리듬을 효과적으로 기술하기 위해 새로운 구조적 언어 (structural language) 를 정의하는 노력이 있었다. 또한 Ptolemy[8] 나 DSP Station[9], 그리고 SPW[10] 등은 블럭 다이어그램 표현 방식을 제공하기도 한다. 비록 이들 디지탈 신호처리 알고리듬을 위한 언어들과 블럭 다이어그램 표현 방식이 점차 많은 사람들의 인정을 받아가고 있지만 C 언어가 디지탈 신호처리 알고리듬을 기술하기 위해 아직도 널리 쓰이고 있는 실정이다. Samani 등이 C++ 언어의 연산자 오버로딩을 이용하여 가변 정밀도의 부동 소수점 연산 모의 실험기를 개발하였지만[11], C 언어를 위한 고정 소수점 데이타 형식은 아직 지원되지 않고 있다. 따라서, 부동 소수점 C 프로그램을 고정 소수점 프로그램으로 바꾸기 위해서는 많은 노력이 필요한 실정이다.

이러한 문제점들을 개선하기 위해서, 본 논문에서는 C나 C++ 언어로 작성된 디지탈 신호처리 알고리듬의 자동 스케일링과 고정 소수점 모의 실험을 위한 유틸리티를 연구하였다[12]. 이 유틸리티는 범위 추정기 (range estimator)와 고정 소수점 모의 실험기 (fixed-point simulator) 로 구성된다. 고정 소수점 알고리듬의 개발을 위해 제안된 절차는 그림 1과 같다.

범위 추정기는 실제 입력을 사용한 부동 소수점 모의 실험을 통하여 내부 신호의 통계적 특성을 찾아낸다. 모의 실험 동안의 통계 자료를 기록하기 위해 새로운 부동 소수점 데이타 클래스, 즉 fSig,가 개발되었다. 고정 소수점 모의 실험기는 C나 C++로 작성된 부동 소수점 디지탈 신호처리 프로그램을 고정 소수점 데이타 클래스인 gFix를 사용하여 고정 소수점 프로그램으로 변환한다. 소수형 (fractional) 이나 정수형 (integer) 형식의 표현상의 제한을 극복하기 위해 임의의 위치에 소수점을 가질 수 있는 일반적인 고정 소수점 형식이 채용되었다[13,14]. 이 고정 소수점 클래스에는 '=' 이나 '+,' 그리고 '$-$,' '$*$,' '/,' 과 같은 연산들도 함께 정의되어 있다. 따라서 C++ 언어의 연산자 오버로딩 기능에 의해 부동 소수점 연산 대신에 고정 소수점 연산이 자동적으로 수행된다[15].

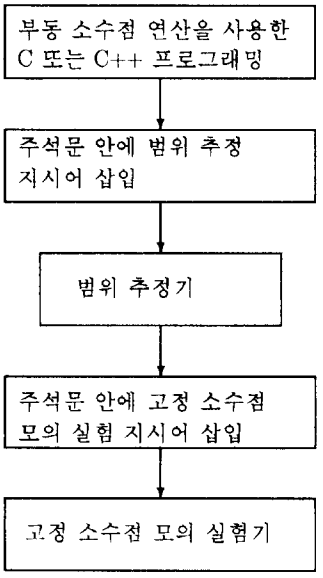


그림 1. 제안된 고정 소수점 알고리듬 개발 절차
Fig. 1. Proposed fixed-point algorithm development procedure.

본 논문의 제 2 장에서는 고정 소수점 표현 방식이 설명되어 있다. 범위를 추정하기 위한 알고리듬은 제 3 장에서 논의되며, 정수 단어길이를 결정하기 위한 범위 추정기는 제 4 장에서 다루어 진다. 고정 소수점 모의 실험기에 대한 자세한 내용은 제 5 장에 실려있다. 마지막으로 제 6 장에서 결론을 맺는다.

Ⅱ. 고정 소수점 데이타 표현 방법

고정 소수점 데이타를 표현하기 위해 다음과 같은 속성정보 (attribute) 를 갖는 고정 소수점 형식[13] 을 채택하였다.

<단어길이, 정수 단어길이, 부호 오버플로우 양자화 모드> (1)

'단어길이'는 고정 소수점 신호를 표현하기 위해 사용되는 전체 비트 수이다. 각 표현 방법의 정확도나 양자화 잡음의 양은 바로 단어길이에 의해 결정된다. '정수 단어길이'는 그림 2에서 보는 바와 같이 (가상)

소수점 왼쪽에 있는 비트 수이다. '부호'는 무부호(unsigned, 'u')이거나 2의 보수(two's complement,'t')가 될 수 있다. 2의 보수 부호 방식은 부호를 나타내기 위해 한 비트를 더 필요로 한다. 신호의 표현 가능한 양의 범위는 2^(정수 단어길이)과 같으며, 양자화 단계의 크기는 2^(-소수 단어길이)와 같다. 즉 부호가 있으며 총 10 비트이고, 정수 단어길이가 2인 신호의 경우에 소수 단어길이는 7 비트이다. 이 고정 소수점 신호의 경우 -4와 $+4$ 사이의 신호를 2^{-7}의 정밀도로 표현할 수 있다. 오버플로우 모드는 오버플로우가 발생하였을 때 그 값을 그대로 유지하거나('o') 포화(saturation)시키는 ('s') 경우를 구분하여 지정하며, 양자화 모드는 LSB 쪽을 양자화할 때 반올림 ('r') 또는 절단 ('t') 기능을 선택한다. 예를 들어, 고정 소수점 형식 '<10,2,''tsr''>' 은 10 비트의 단어길이와 2 비트의 정수 단어길이, 2의 보수 부호, 오버플로우 발생 시에 포화, 그리고 단어길이를 줄일 때 반올림을 지정한다.

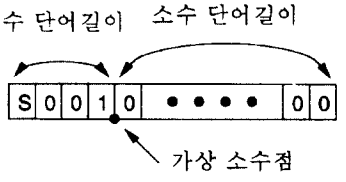


그림 2. 고정 소수점 형식에서의 데이타 표현 방법
Fig. 2. Data representation method in the fixed-point format.

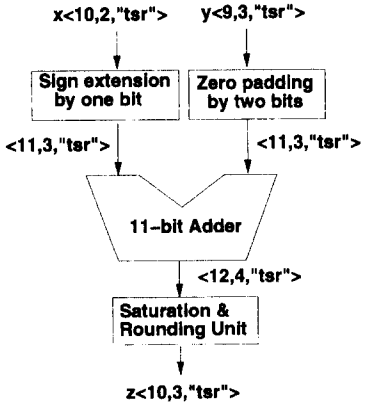


그림 3. 서로 다른 정수 단어길이를 갖는 두 고정 소수점 변수의 덧셈을 위한 간단한 하드웨어
Fig. 3. A hardware model for adding two fixed-point variables of different integer wordlengths.

고정 소수점 형식에서는 두 수를 더하거나 빼기 전에 가상 소수점을 정렬하여야 한다. 일예로 x와 y를 더하여 z를 만드는 경우를 생각하여 보자. 단, x, y, z는 각각 '<10,2,''tsr''>,' '<9,3,''tsr''>,' '<10,3,''tsr''>,'의 고정 소수점 형식을 갖는다고 가정하자. x와 y를 고정 소수점 가산기로 더하기 위해서 x는 부호 확장 (sign-extension) 으로 한 개의 정수 비트를 더 가져야 하며, 반면 y는 두 개의 소수 비트를 더 필요로 한다. 이어서 11 비트 이상의 덧셈을 수행한 후 반올림하고 포화시킨다. 위와 같은 절차는 그림 3과 같이 하드웨어로 구현할 수 있다.

Ⅲ. 통계적 특성을 이용한 범위 추정

어떤 신호 x를 고정 소수점 변수로 만들기 위해서는 최대나 최소값에서 오버플로우가 일어나지 않도록 정수 단어길이가 결정되어야 한다. 따라서 변수 x의 최소의 정수 단어길이, $IWL_{\min}(x)$는 다음과 같이 범위, $R(x)$로부터 결정된다.

$$IWL_{\min}(x) = \lceil \log_2 R(x) \rceil \qquad (2)$$

여기서 $\lceil x \rceil$는 x와 같거나 작지 않은 최소의 정수를 나타낸다. 범위를 찾기위한 기존의 시도로는 전달함수의 L1 노옴(norm)을 이용하는 방법을 들 수 있다[11]. L1 노옴에 의해 추정된 범위는 모든 경우에 대해 오버플로우가 발생하지 않음을 보장하지만, 많은 경우에 상당히 보수적인 추정치가 된다. 또한 적응(adaptive)이나 비선형 시스템의 경우 L1 노옴을 구하기는 매우 어렵다. 이러한 문제점을 피하기 위해서, 본 유틸리티에서는 부동 소수점 모의 실험에 의해 범위를 추정한다. 균일 (uniform) 분포나 가우시안(Gaussian), 또는 라플라시안(Laplacian)과 같은 단순한 분포는 몇몇 통계적 정보로써 쉽게 표현할 수 있다. 음성 신호의 경우 라플라시안 분포로 모델링할수 있음은 널리 알려져 있는 사실이다. 그러나, 실제 시스템에 존재하는 모든 신호을 위와 같은 단순한 분포로 모델링할 수는 없다. 일반적으로 신호의 분포는 비대칭 (nonsymmetric) 이거나 또는 다중 모드(multimodal)일 수 있다. 만약 단일 모드(unimodal) 분포에 적용했던 범위 추정 방법을 다중 모드 분포에 적용한다면 추정된 범위는 너무 크거나 작을 수 있다는 사실에 유의해야 한다. 즉, 범위를 추정하기위한 방법은 분포의 성격에 따라 달라야 한다. 이 장에서는 신호의

분포를 이용한 범위 추정 방법을 설명한다.

1. 신호의 통계적 특성

주어진 데이타, $x_i, 1 \le i \le N$, 에 대한 n차 중앙 모먼트 (central moment), μ_n,은 다음과 같이 정의된다.

$$\mu_n = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^n \tag{3}$$

단, $\bar{x}$ 는 평균이다 ($\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$). 우선 분포가 정규 분포가 아닌 경우, 그 신호가 평균을 중심으로 왼쪽이나 오른쪽으로 치우쳐 분포되어 있는지의 여부를 가릴 필요가 있다. 이를 위하여서 다음과 같이 정의되는 의도(Skewness)를 이용한다.

$$s = \frac{\mu_3}{\sigma^3} \tag{4}$$

여기서, σ^2(분산)는 μ_2에 해당한다. 신호의 분포가 평균을 중심으로 대칭이 될수록 의도는 0에 가까운 값을 갖는다. 0이 아닌 의도는 신호가 평균을 중심으로 왼쪽이나 오른쪽으로 치우쳐 분포함을 의미한다. 다음으로 평균을 주위로 얼마나 많은 샘플이 모여 있는가를 알기 위해 다음과 같이 정의되는 첨도(Kurtosis)를 이용한다.

$$k = \frac{\mu_4}{\sigma^4} - 3. \tag{5}$$

첨도의 값은 가우시안 분포일때 0이다. 모드(Mode))는 분포의 극대점(local maximum)이다. 단일 모드 분포는 하나의 모드를 가지며, 다중 모드 분포는 여러 개의 모드를 가진다. 단일 모드 분포의 표준편차, σ, 는 그림 4(a)에서 보는 바와 같이 모드의 넓이로 생각할 수 있다.

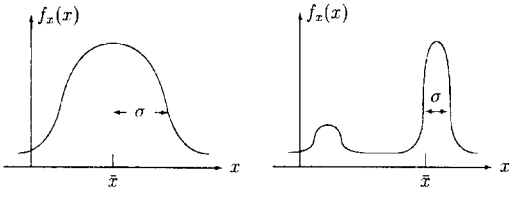


그림 4. (a) 단일 모드와 (b) 이중 모드 분포
Fig. 4. (a) Unimodal and (b) bimodal distributions.

예를 들어, 가우시안 분포는 4배의 σ로 전체의 99.99%를 포함할 수 있다. 따라서, 단일 모드를 가지고 대칭인 신호는 μ와 n배의 σ로 범위를 추정할 수 있다. 여기서 n은 분포에의해 결정된다. 그러나 다중 모드의 경우에는 σ로 분포의 넓이를 알 수 없다. 그 한 예로서 그림4(b)와 같은 이중 모드 분포를 생각해 보자. 대부분의 적응 시스템의 내부 신호는 이와 같은 분포를 가지며, 두개의 모드는 각각 초기 상태와 수렴 후의 상태에 해당한다. 따라서, 평균과 표준편차 뿐만 아니라 분포의 특성 또한 중요하다.

2. 범위의 추정

효과적인 범위 추정을 위해 분포는 다음과 같이 분류될 수 있다.

- 단일/다중 모드
- 대칭/비대칭
- 평균이 0인 경우/평균이 0이 아닌 경우

대칭성과 평균이 0인지는 각각 의도와 평균값으로부터 알 수 있으나, 모드의 갯수는 간단히 추정할 수 없다. 표 1과 같이 실험 결과로부터 단일 모드와 다중 모드 분포를 구분하기 위한 실험법칙을 유도할 수 있다.

표 1. 테스트 신호의 통계적 특성
Table 1. Statistical information of a few signals.

Signal	μ	σ	s	k	$\hat{n}_{99.9\%}$	$\hat{n}_{100\%}$
Unform	0.00	0.29	-0.01	-1.20	1.72	1.72
Gaussian	0.00	1.00	-0.01	-0.02	3.84	4.35
Laplacian	0.00	1.00	0.05	2.95	6.27	7.26
Speech	0.00	0.05	0.16	3.47	5.02	6.30
FIR_uniform	0.01	0.52	-0.04	-0.39	2.72	2.95
FIR_gaussian	-0.01	3.80	0.00	0.00	3.35	3.70
FIR_speech	0.00	0.17	0.10	0.13	3.23	3.98
IIR_uniform	0.00	1.02	0.00	-0.11	3.24	3.52
IIR_gaussian	-0.01	3.80	0.00	0.00	3.35	3.70
IIR_speech	0.00	0.06	0.29	1.04	3.88	4.22
LMS_error	0.00	0.13	-6.49	196.18	5.71	26.24
LMS_coeff	1.00	0.04	-17.37	404.34	1.30	1.38

즉, $-1.2 < k < 5$ 이면 그 신호의 분포는 근사적으로 단일 모드이다. 일예로 표 1의 "LMS_error"와 "LMS_coeff" 신호는 의도 값이 0이 아니고 첨도도 매우 큰 값을 갖는다. 따라서 이 신호들은 비대칭이며 다중 모드 분포를 갖는다고 추정된다. 실제로, 이 두 신호는 초기 상태와 최종 상태를 갖는 이중 모드 분포를 갖는다.

단일 모드이고 대칭인 분포의 경우에는, 다음과 같

이 범위를 효과적으로 추정할 수 있다.

$$R = |\mu| + n \times \sigma, \quad n \propto k \tag{6}$$

동일한 분산을 갖는 두개의 대칭 분포의 경우 상대적으로 큰 첨도 값을 갖는 분포가 작은 쪽보다 더 넓게 퍼져있다. 따라서 범위 추정을 위해 더 큰 n 값이 필요하다. 실험 결과를 분석하여 n 값으로 $k+4$를 사용하였다. 실제로 표 1의 ''IIR_speech''의 분포는 첨도 값에따라 다섯 배의 σ로 포함할 수 있다. 그러나, 위의 법칙이 다중 모드이거나 비대칭 경우와 같은 다른 분포에도 모두 만족스럽지는 않다. 이러한 경우에는 다른 법칙을 생각할 수 있다.

$$R = \hat{R}_{99.0\%} + g \tag{7}$$

여기서 g는 범위를 위한 보호값 (guard) 이며, $((\hat{R}_{100\%} - \hat{R}_{99.9\%}) \times r_R)$과 같이 정의된다. 단 여기서 $\hat{R}_{P\%}$는 전체 샘플의 $P\%$를 포함하는 부분 최대값이며 여러 부분 최대값이 모의 실험 동안에 수집된다. $\hat{R}_{100\%}$와 $\hat{R}_{99.9\%}$가 많이 다를수록 더 큰 보호값이 필요하다. 계수 r_R은 보호값을 조절하며, 현재로는 2를 사용하고 있다.

한편, 통계적 정보는 입력 데이타에 따라 달라지므로, 보다 신뢰성 있는 범위를 추정하기 위해서는 여러 가지의 다른 데이타 집합이 필요하다. 입력 신호에 따른 분포 변화의 정도를 측정하기 위해서, 각각 평균, 분산, 의도, 그리고 첨도에 대해 민감도를 다음과 같이 계산한다.

$$S_\mu = \mu_{\max} - \mu_{\min} \tag{8}$$

$$S_\sigma = \sigma_{\max} - \sigma_{\min} \tag{9}$$

$$S_s = s_{\max} - s_{\min} \tag{10}$$

$$S_k = k_{\max} - k_{\min} \tag{11}$$

단, 여기서 $(\cdot)_{\max}$와 $(\cdot)_{\min}$은 각각 여러개의 입력을 이용한 모의 실험 결과 중에서 최대값과 최소값을 나타낸다. 이제 통계 특성은 다음과 같이 수정된다.

$$\mu' = \mu_{\max} + S_\mu codtr_S \tag{12}$$

$$\sigma' = \sigma_{\max} + S_\sigma codtr_S \tag{13}$$

$$s' = s_{\max} + S_s codtr_S \tag{14}$$

$$k' = k_{\max} + S_k codtr_S \tag{15}$$

계수 r_S로는 0.1이 사용되고 있다. 만약 s'과 k'이 대칭이고 단일 모드이라면, 식 (6)에 의해 범위를 추정할 수 있으며, 그렇지 않는 경우에는 식 (7)이 사용된다.

Ⅲ. 범위 추정기

본 연구에서는 범위 추정을 위해 모의 실험 방법을 사용하기 때문에 모의 실험 동안 통계 정보를 저장하는 프로그램이 필요하다. 범위 추정을 위해 원래의 C 또는 C++ 부동 소수점 프로그램을 수정하지 않기 위해서 C++ 언어의 오버로딩 (overloading) 기능을 사용하였다. 신호의 범위를 추적하기위해 새로 개발된 데이타 클래스의 이름은 fSig이다. C++ 언어에서 클래스는 일종의 형식(type)이므로, C 또는 C++ 디지탈 신호처리 프로그램의 범위 추정 모델을 만들기 위해서 단지 변수의 type을 float에서 fSig로 바꾸어 주면 된다. fSig 클래스는 현재 값을 저장할 뿐 만 아니라, 사적 멤버(private member)를 사용하여 변수의 각종 특성을 기록한다. 따라서, 모의 실험이 완료되면 fSig로 선언된 변수의 범위는 클래스에 저장된 기록들로부터 쉽게 구할 수 있다. 단, 이와 같은 범위 추정 방법에서는 실제 상황에서 발생하는 데이터를 모의 실험 입력으로 사용해야 한다. 그림 5에서 보는 바와 같이 fSig 클래스는 몇개의 사적 멤버(private member)들을 가지고 있다. 변수 Data는 현재 값을 저장하며, Sum과 Sum2는 각각 과거 값들의 합과 제곱합을 기록한다. Sum3와 Sum4는 각각 3차와 4차 모멘트(moment)를 위해 3승합과 4승합을 가지고 있다. 이들은 이후에 평균과 분산, 의도와 첨도를 계산하는데 사용된다. AMax는 모의 실험 동안에 절대값의 최대치를 저장한다. 또한, 각 변수의 수정 횟수는 SumC에 기록된다. fSig 클래스는 산술 및 관계 연산자를 오버로딩한다. 따라서, 덧셈이나 뺄셈, 곱셈과 나눗셈과 같은 기본적인 산술 연산의 경우 fSig를 위한 연산자가 자동적으로 수행된다. 이 특성은 '==,' '!=,' '>,' '<,' '≥,' '<='와 같은 관계 연산자들에도 적용된다. 이 클래스에는 float와 같은 다른 형식(type)과의 연산도 정의되어 있으므로 fSig 변수는 float 변수와도 비교될 수 있다. fSig 클래스의 member들은 그 변수

가 '=,'와 ' + =,' ' − =,' ' * =,' ' / ='와 같은 저장 연산자에 의해 새로운 값으로 할당(assign)될 때만 갱신된다. 예를 들어, AMax는 새로이 저장된 현재 값의 절대치가 과거의 값보다 클 때 수정된다. 변수의 선언을 수정하여 범위 추정을 위한 모의 실험 모델을 만들고 나면, 범위 추정기가 실행된다. 범위 추정기는 모의 실험 모델을 컴파일하고, 그 오브젝트 파일을 모의 실험 구동기와 fSig 클래스의 오버로딩된 연산자들과 링크시켜 실행시킨다.

```
class fSig
{
private:
  ......
  double Data;
  double Sum;
  double Sum2;
  double Sum3;
  double Sum4;
  double AMax;
  long SumC;
  ......

public:

  ......
  fSig& operator = (const fSig&);
  fSig& operator = (double);

  friend double operator + (const fSig&, const fSig&);
  friend double operator + (const fSig&, double);
  friend double operator - (const fSig&, const fSig&);
  friend double operator - (const fSig&, double);
  friend double operator * (const fSig&, const fSig&);
  friend double operator * (const fSig&, double);
  friend double operator / (const fSig&, const fSig&);
  friend double operator / (const fSig&, double);

  friend short operator == (const fSig&, const fSig&);
  friend short operator == (const fSig&, double);
  friend short operator != (const fSig&, const fSig&);
  friend short operator != (const fSig&, double);
  friend short operator > (const fSig&, const fSig&);
  friend short operator > (const fSig&, double);
  friend short operator < (const fSig&, const fSig&);
  friend short operator < (const fSig&, double);
  ......

};
```

그림 5. fSig 클래스의 선언
Fig. 5. Declaration of the fSig class.

본 연구에서는 프로그램과 응용 프로그램 개발에 GNU C++ 컴파일러인 g++ 버젼 2.6.3을 사용하였다[16]. 모의 실험 구동기는 모의 실험 모델을 실행시키며, 모의 실험 동안 범위 정보가 수집된다. 모의 실험이 끝나면 Sum, Sum2, Sum3, Sum4 and SumC를 사용하여 평균 (μ), 표준 편차 (σ), 의도 (s), 그리고 첨도 (k)를 구할 수 있다. 이제 fSig 변수의 통계적 범위는 앞절에서 설명한 절차에의해 구할 수 있다. 마지막으로, 모든 신호의 정수 단어길이는 식 (2)에 의해 각각의 범위로부터 계산할 수 있다. 하나의 예로, 1차 IIR 필터를 생각해 보자. 필터의 원래 C++ 프로그램과 범위추정을 위해 변환된 프로그램이 그림 6에 나타나 있다.

```
void
iir1(short argc, char *argv [ ] )
{
  float Xin;
  float Yout; // fSig()
  float Ydly; // fSig()
  float Coeff;

  Coeff = 0.9;
  Ydly = 0.;
  for( i = 0; i < 10000; i++ ) {
    infile >> Xin ;
    Yout = Coeff * Ydly - Xin
;
    Ydly = Yout ;
    outfile << Yout << '\n';
  }
}
```

(a) 사용자가 작성한 C++ 프로그램

```
void
iir1(short argc, char *argv [ ] )
{
  float Xin;
  static fSig Yout("iir1/Yout");
  static fSig Ydly("iir1/Ydly");
  float Coeff;

  Coeff = 0.9;
  Ydly = 0.;
  for( i = 0; i < 10000; i++ ) {
    infile >> Xin ;
    Yout = Coeff * Ydly + Xin
;
    Ydly = Yout ;
    outfile << Yout << '\n';
  }
}
```

(b) 범위 추정을 위해 변환된 프로그램

그림 6. 1차 IIR 필터를 위한 C++ 프로그램
Fig. 6. C++ programs for a first order IIR filter.

```
Statistics:
  VarName  Mean  StdDev  Skewness  Kurtosis  R99.9%
  R100%  Update
iir1/Ydly  -0.1133  +1.3076  +0.0220  -0.0258  +4.2638  +4.4214
3001
iir1/Yout  -0.1134  +1.3078  +0.0220  -0.0268  +4.2638  +4.4214
3000

Integer wordlengths:
    VarName        Range        IWL
   iir1/Ydly     +5.309891      +3
   iir1/Yout     +5.309515      +3

Save the statistical results? [ Y/n ]
Filename? iir1.sta
```

그림 7. IIR 필터의 범위 추정 결과
Fig. 7. The result of the range estimator for the IIR filter.

그림 6에서 보는 바와 같이 범위추정을 위해 프로그램을 만들기 위해서는 원래 프로그램의 선언 부분만 바꾸어주면 된다. 이 예에서는 −1 에서 +1 사이에서 균일 분포를 갖는 백색 잡음을 입력 데이타로 사용하

였으며, 4배의 표준편차를 범위 추정에 이용하였다. 위의 가정으로부터 X_{in}의 정수 단어길이가 0임을 알 수 있다. 범위 추정 결과는 그림 7과 같다.

V. 고정 소수점 모의실험기

대부분의 상위 수준 언어의 컴파일러들은 식 (1)의 일반적인 고정 소수점 형식과 연산을 지원하지 않는다. 따라서 부동 소수점 프로그램으로부터 비트 단위의 정밀도를 갖는 고정 소수점 프로그램을 자동으로 얻고, 또한 이를 이용한 모의 실험에 의하여 유한 단어 효과를 알아보기 위하여 새로운 고정 소수점 데이타 클래스 gFix와 그에 따른 연산자들을 개발하였다. gFix 클래스의 선언 부분은 그림 8와 같다. 이 그림에서 보는 바와 같이, 고정 소수점 데이타 클래스 gFix는 몇 가지 사적 멤버(private member)를 가지고 있는데, 각각은 가수 (m), 단어길이 (wl), 정수 단어길이 (iwl), 그리고 속성정보 (represent, saturation, round) 등이다. 32 비트보다 더 큰 정밀도를 필요로 하는 가수를 표현하기 위해서 복수 정밀도의 정수 연산을 지원하는 GNU C++ 라이브러리의 Integer 클래스를 사용하였다[17].

gFix 클래스는 C 나 C++ 언어에서 지원되는 할당 및 산술 연산자들 모두를 지원한다. 지원되는 연산자의 목록은 그림 8과 같다. 연산자들에 대해 간단히 설명하면 다음과 같다.

1. 할당 연산자, '=,'는 입력 데이타를 좌측 변수의 고정 소수점 형식에 따라 변환한 다음 좌측 변수에 할당한다. '할당 연산자의 우측에 존재하는 입력 데이타는 부동 소수점 값과 고정 소수점 값 모두 가능하다. 만일 좌측 변수의 형식이 입력 데이타를 표현하기에 충분치 못한 정밀도를 갖는다면, 주어진 데이타는 좌측 변수의 속성 정보 즉, 포화, 오버플로우, 반올림, 그리고 절단 정보에 따라 변환된 후 할당된다.
2. 고정 소수점 덧셈 연산자, '+,'의 동작은 그림 9에서 프로그램으로 설명되고 있다. 연산 과정 상에서 발생할 수 있는 정밀도의 손실을 막기위해서 먼저 두 입력 데이타의 최대 정수 및 소수 단어길이를 계산한다. 예를 들어, 첫번째 인자의 정수 및 소수 단어길이가 각각 2와 8이고 두번째가 4와 6인 경우를 생각해 보자. 이 경우 내부 데이타의 정수 단어길이는 오버플로우를 막기 위하여 5가 되고, 소수 단어길이는 정밀도를 잃지 않기 위하여 8이 된다. 이어서 입력 데이타들은 쉬프트 연산으로 정렬된 후 고정 소수점 덧셈이 실행된다.

```
class gFix
{
  Integer m; // mantissa
  short iwl; // integer word-length
  short wl; // total word-length
  char represent; // 't' or, 'u'
  char saturation; // 's' or, 'o'
  char round; // 'r' or, 't'

public:

  // constructors
  gFix();
  gFix(short wlen, short iwlen, char *fmt);
  gFix(double d, short wlen, short iwlen, char *fmt);
  .....

  // assignment operators
  gFix& operator = (gFix& x);
  gFix& operator = (double d);

  // basic operators
  friend gFix operator + (gFix& x, gFix& y);
  friend gFix operator - (gFix& x, gFix& y);
  friend gFix operator * (gFix& x, gFix& y);
  friend gFix operator / (gFix& x, gFix& y);
  friend gFix operator << (gFix& x, short b);
  friend gFix operator >> (gFix& x, short b);
  .....

  // assignment based operators
  gFix& operator += (gFix& x);
  gFix& operator -= (gFix& x);
  gFix& operator *= (gFix&);
  .....

  // relational operators
  friend short operator == (gFix& x, gFix& y);
  friend short operator != (gFix& x, gFix& y);
  friend short operator >= (gFix& x, gFix& y);
  .....

  // miscellaneous operators
  friend istream& operator >> (istream& s, gFix& x);
  friend ostream& operator << (ostream& s, gFix& x);
  .....

};
```

그림 8. gFix 클래스의 선언
Fig. 8. Declaration of the gFix class.

3. 고정 소수점 곱셈 연산자, '*,'는 그림 9에 나타나 있다. 2의 보수 표현법의 경우, 곱의 단어길이는 두 입력의 단어길이의 합에서 1을 뺀 값이 된다. 이는 추가의 부호 비트 하나를 없애기 위함

이다. 정수 단어길이는 두 입력의 정수 단어길이의 합이 된다.

4. 산술적 우측 쉬프트 연산자, '≫,'는 gFix 신호를 비트 수준에서 오른쪽으로 쉬프트 시킨다. 그러나 단어길이와 정수 단어길이는 이 연산자에 의해 변하지는 않으므로 한 비트 오른쪽 쉬프트는 gFix 신호의 실제 값을 반으로 만든다.
5. 산술적 좌측 쉬프트 연산자, '<<,'는 gFix 신호를 비트 수준에서 왼쪽으로 쉬프트 시킨다. 위와 비슷한 방식으로, 한 비트 왼쪽 쉬프트는 gFix 변수의 실제 값을 두배로 만든다.

```
gFix operator + (const gFix& x, const gFix& y)
// assume that
// result.iwl = max(x.iwl, y.iwl) + 1
// result.wl = max( result.iwl + max(x.fwl, y.fwl) + 1,
MAXWL )
// ,where fwl means fraction word-length.
{
  short xflen, yflen, maxflen; // fraction length
  short iwlen, wlen;
  short shift;
  Integer I;
  xflen = x.wl - x.iwl - 1;
  yflen = y.wl - y.iwl - 1;
  maxflen = ( xflen > yflen) ? xflen : yflen;
  iwlen = (x.iwl > y.iwl ) ? x.iwl+1 : y.iwl+1;
  wlen = iwlen + maxflen + 1;
  if( wlen > MAXWL ) wlen = MAXWL;
  if( xflen == yflen ) I = x.M + y.M;
  else {
    shift = xflen - yflen;
    if( shift > 0 ) I = x.M + (y.M << shift);
    else I = (x.M << (-shift)) + y.M;
  }
  return gFix(I,wlen,iwlen);
}
```

(a) 덧셈

```
gFix operator * (const gFix& x, const gFix& y)
// assume that
// result.wl = x.wl + y.wl - 1
// result.iwl = x.iwl + y.iwl
{
  short      iwlen, wlen;
  Integer    I;
  wlen = x.wl + y.wl - 1;
  if( wlen > MAXWL ) wlen = MAXWL;
  iwlen = x.iwl + y.iwl;
  I = x.M * y.M;
  return gFix(I,wlen,iwlen);
}
```

(b) 곱셈

그림 9. gFix 클래스의 연산자들
Fig. 9. Operators of the gFix class.

위에서 기술한 바와 같이, 고정 소수점 덧셈이나 곱셈 연산 과정에서는 정밀도의 손실이 없다. 그러나, 우측이나 좌측의 산술적 쉬프트 연산의 경우 정밀도의 손실이나 오버플로우를 일으킬 수 있다. 형식 변환은 오직 할당 연산자에서만 일어난다. 따라서 그림 10-(a)와 그림 10-(b)의 두 프로그램은 서로 다른 고정 소수점 결과를 가질 수 있다. 그림 10-(b)에서는 $a+b$ 결과가 10 비트 데이타 *tmp*로 변환되고 c로 더해지며, 마지막으로 d로 형식 변환된다.

```
gFix a(12,0,"tsr");
gFix b(12,0,"tsr");
gFix c(12,0,"tsr");
gFix d(10,1,"tsr");
d = a + b + c;
```

(a)

```
gFix a(12,0,"tsr");
gFix b(12,0,"tsr");
gFix tmp(10,1,"tsr");
gFix c(12,0,"tsr");
gFix d(10,1,"tsr");
tmp = a + b;
d = tmp + c;
```

(b)

그림 10. 서로 다른 구조를 갖는 세 입력 덧셈
Fig. 10. Three operand addition employing different architectures.

```
void
iir1(short argc, char *argv [ ] )
{
  gFix Xin(12,0);
  gFix Yout(16,3);
  gFix Ydly(16,3);
  gFix Coeff(10,0);
  Coeff = 0.9;
  Ydly = 0.;
  for( i = 0; i < 10000; i++ ) {
    infile >> Xin ;
    Yout = Coeff * Ydly + Xin ;
    Ydly = Yout ;
    outfile << Yout << '\n';
  }
}
```

그림 11. 1차 IIR 필터의 고정 소수점 C++ 프로그램
Fig. 11. A fixed-point C++ program for a first order IIR filter.

같은 알고리듬에 기초한 서로 다른 구현 구조의 고정 소수점 성능을 위와 같은 특성을 활용하여 비교할 수 있다. 관계 연산자 및 할당 연산을 포함한 연산자들도 지원된다. 관계 연산자로는 '==,' '!=,' '>=,' '<=,' '>,' '<' 등이 포함된다. gFix 변수는 실수로 해석된 후에 비교되기 때문에, 관계 연산자는 다른 gFix 변수 뿐 만 아니라 float나 double 변수와도 비교 가

능하다. 할당 연산을 포함한 연산자들로는, '+=,'와 '-=,' '*=,' '/=,' '≫=,' '≪=' 등이 지원된다. 제 3 절에서 논의된 IIR 필터를 다시 한번 생각해 보자. gFix 클래스를 사용하면 그림 6-(a)의 부동 소수점 프로그램으로부터 IIR 필터의 고정 소수점 모의 실험 모델을 쉽게 얻을 수 있다. 그림 11은 제 3 절에서 얻었던 단어길이와 정수 단어길이 정보를 사용한 고정 소수점 프로그램을 보여주고 있다. 여기서 주의하여 보아야할 점은 단지 변수의 선언 만이 gFix로 바뀌었고 나머지 부분은 변하지 않았다는 것이다. 이 고정 소수점 프로그램을 모의 실험하여 얻은 고정 소수점 연산 결과와 부동 소수점 결과를 비교하여 52.15dB의 SQNR(Signal to Quantization Noise Ratio)을 얻었다.

VI. 결 론

C나 C++로 기술된 알고리듬의 스케일링과 단어길이 최적화를 도와주는 고정 소수점 최적화 유틸리티가 개발되었다. 범위 추정기를 사용하여 각 신호의 정수 단어길이를 결정하며, 고정 소수점 모의 실험기로 유한 단어길이 효과를 평가할 수 있다. C++ 언어의 연산자 오버로딩 기능을 이용하여 단지 변수 선언 부분만을 수정하여 부동 소수점 프로그램으로부터 자동적으로 범위 추정 모델과 고정 소수점 모의 실험 모델을 얻을 수 있다. 이 유틸리티는 비선형, 시변, 다중률 및 다차원 신호처리 알고리듬을 비롯하여 거의 모든 디지탈 신호처리 프로그램의 고정 소수점 최적화에 이용할 수 있다. 비록 생성된 C++ 고정 소수점 프로그램은 실시간 구현을 위한 것은 아니지만, 비트 수준의 정밀도를 가지므로 VLSI나 고정 소수점 디지탈 신호 처리기를 이용한 구현의 고정 소수점 성능 평가에 사용될 수 있다. 제안된 고정 소수점 데이타 형식을 사용한 디지탈 신호처리 프로그램은 VHDL 코드나 정수형 프로그램으로 변환할 수 있다. 일예로 본 유틸리티를 이용하여 IEEE std. 1180-1190을 만족하는 8×8 IDCT의 단어길이 최적화를 수행하였다[18,19]. 또한 MPEG 오디오나 CELP 보코더의 고정 소수점 성능 평가에 본 유틸리티가 유용하게 쓰였다[20,21]. 이 유틸리티는 웹 사이트 'http://www.VSPL.snu.ac.kr' 에서 학문적 목적으로 필요할 경우 자유로이 가져 갈 수 있다.

참 고 문 헌

[1] L. B. Jackson, "On the interaction of roundoff noise and dynamic range in digital filters," *Bell System Technical Journal*, vol. 49, pp. 159-184, Feb. 1970

[2] B. Liu, "Effect of finite word-length on the accuracy of digital filters: A review," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 6, pp. 670-677, Nov. 1971

[3] H. K. Kwan, "Amplitude scaling of arbitrary linear digital networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 32, no. 6, pp. 1240-1242, Dec. 1984

[4] C. Caraiscos and B. Liu, "A roundoff error analysis of the LMS adaptive algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 32, no. 1, pp. 34-41, Feb. 1984

[5] P. Hilfinger, "A high-level language and silicon compiler for digital signal processing," in *Proc. Custom Integrated Circuits Conference*, Los Alamitos, Calif., 1985, pp. 213-216

[6] P. L. Geurnic, A. Benveniste, P. Bournai, and T. Gautier, "SIGNAL-A data flow-oriented language for signal processing," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, no. 2, Apr. 1986

[7] K. W. Leary and W. Waddington, "DSP/C: A standard high level language for dsp and numeric processing," in *Proc. International Conference on Acoustics, Speech and Signal Processing*, 1990, pp. 1065-1068

[8] J. Buck S. Ha E. A. Lee and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. J. Comput. Simulation*, June 1992

[9] Mentor Graphics, *DSP Station Design Architect User's Manual*, 1993

[10] Comdisco Systems, Inc., *SPW-The DSP Framework Designer/BDE User's Guide*, Sep. 1992

[11] D. M. Samani, J. Ellinger, E. J. Pwers, and E. E. Swartzlander, Jr., "Simulation of variable precision IEEE floating point using C++ and its application in digital signal processor design," in *Proc. Twenty-seventh Annual Asilomar Conference on Signals, Systems, and Computer, 1993,* pp. 1574-1578

[12] S. Kim, K.-I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," in *Proc. 1995 IEEE Workshop on VLSI Signal Processing,* Oct. 1995, pp. 197-206

[13] S. Kim and W. Sung, "A floating-point to fixed-point assembly program translator for the TMS320C25," *IEEE Trans. Circuits and Systems,* Part II, vol. 41, no. 11, pp. 730-739, Nov. 1994

[14] S. Kim and W. Sung, "Fixed-point simulation utility for C and C++ based digital signal processing programs," in *Proc. Twenty-eighth Annual Asilomar Conference on Signals, Systems, and Computer,* 1994, pp. 162-166

[15] B. Stroustrup, *The C++ Programming Language, 2nd Ed,* Addison Wesley, 1993

[16] R. M. Stallman, *Using and Porting GNU CC, 1990,* Free Software Foundation, Inc

[17] D. Lea, *User's guide to the GNU C++ Library, version 2.0,* Apr. 1992

[18] S. Kim and W. Sung, "Optimum word-length determination of 8x8 IDCT architectures conforming to the IEEE standard specifications," *in Proc. Twenty-seventh Annual Asilomar Conference on Signals, Systems, and Computers,* 1995, pp. 821-825

[19] 금 기일, 성 원용, "VHDL을 이용한 고정 소수점 디지탈 신호처리 알고리듬 개발 소프트웨어," 대한전자공학회지 31권 A편 11호, 1994년 11월, pp. 138-148.

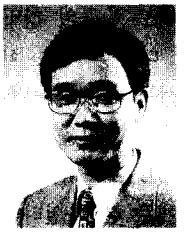
[20] M. S. Jeong, S. Kim, J. S. Sohn, J. Y. Kang, and W. Sung, "Finite wordlength effects evaluation of the MPEG audio decoder," in *Proc. International Conferational Conference on signal Processing Applications and Technology,* Oct. 1996, to be published

[21] W. Sung, J. Sohn, J. Kang, and S. Kim, "Fixed-point implementation of the FS-CELP vocoder using the Autoscaler for the TMS320C50," in *Proc. International Conference on Signal Processing Applications and Technology,* 1995, pp. 1883-1891

저 자 소 개

金 時 鉉(正會員)

1990년 서울대학교 제어계측공학과 졸업(공학사). 1992년 동대학원 졸업(공학석사). 1996년 동대학원 졸업(공학박사). 1996년 ~ 1997년 University of California, Berkeley 전기 및 컴퓨터 공학과 Post-doctorate. 1996년 ~ 현재 LG 종합기술원. 관심분야는 embedded signal processing system design 등임



成 元 鎔(正會員)

1978년 2월 서울대학교 전자공학과 졸업(공학사). 1980년 1983년 금성사 중앙연구소. 1987년 7월 미국 University of California, Santa Barbara 전기 및 컴퓨터공학과 졸업(공학박사). 1989년 2월 ~ 현재 서울대학교 전기공학부 및 반도체공동연구소 부교수. 1993년 ~ 1994년 Comdisco Systems (현 Alta Group) 기술고문. 반도체 공동연구소 설계 연구부장을 맡고 있음. 관심분야는 병렬처리 컴퓨터와 VLSI를 이용한 고속 신호 처리등임