論文 92-29A-3-13

자동 스케일링 기능이 지원되는 고정 소수점 디지탈 시그날 프로세서 개발 시스템

(A Fixed-point Digital Signal Processor Development System Employing an Automatic Scaling)

金 時 鉉* 成 元 鎔**

(See Hyun Kim and Won Yong Sung)

要 約

고정 소수점 디지탈 시그날 프로세서는 부동 소수점 디지탈 시그날 프로세서에 비해 값싸고 빠르다는 장점이 있기 때문에 통신 시스템을 비롯하여 여러분야에 널리 사용되고 있다. 그러나 산술 연산을 행하거나 데이타를 이동시킬 때에는 정확도(accuracy)를 유지하면서 오버플로우(overflow)를 방지하기 위해 데이타의 스케일링(scaling)이 필요하다. 본 연구에서는 고정 소수점 디지탈 시그날 프로세서인 TMS 320C25의 사용시 스케일링을 자동화시켜 주는 소프트웨어를 제작하였다. 이 소프트웨어는 가상적인 부동 소수점 연산을 지원하는 새로운 명령어 집합과 시뮬레이터로 구성된다. 프로그래머는 가상 부동 소수점 하드웨어와 명령어 집합, 그리고 부동 소수점 데이타 형식을 사용한다. 부동 소수점 프로그램과 데이타는 본 소프트웨어에 의해 고정 소수점용으로 변환되며, 따라서 수행 속도가 저하되지 않는다. 각 고정 소수점 변수는 자신의 범위(range)에 따라 고유한 소수점을 가지며, 산술 연산이나 데이타 이동시에 필요한 쉬프트 횟수는 각 변수의 소수점 위치로부터 결정된다. 본 소프트웨어를 이용한 TMS 320C25용 코드 생성기(code generator)도 제작되었다. 이 코드 생성기에서는 주어진 필터의 특성으로 부터 FIR, IIR, 또는 adaptive transversal filter 등의 부동 소수점 어셈블리 프로그램을 만들고, 본 개발 지원 소프트웨어를 이용하여 고정소수점 프로그램으로 바꾼다.

Abstract

The use of fixed-point digital signal processors, such as the TMS 320C25, requires scaling of data at each arithmetic step to prevent overflows while keeping the accuracy. A software which automatizes this process is developed for TMS 320C25. The programmers use a model of a hypothetical floating-point digital signal processor and a floating-point format for data representation. However, the program and data are automatically translated to a fixed-point version by this software. Thus, the execution speed is not sacrificed. A fixed-point variable has a unique binary-point location, which is dependent on the range of the variable. The range is estimated from the floating-point simulation. The number of shifts needed for arithmetic or data transfer step is determined by the binary-points of the variables associated with the operation.

A fixed-point code generator is also developed by using the proposed automatic scaling software. This code generator produces floating-point assembly programs from the specifiations of FIR, IIR, and adaptive transversal filters, then floating-point programs are transformed to fixed-point versions by the automatic scaling software.

*學生會員, **正會員, 서울大學校 半導體共同研究所 및 制御計測工學科

(ISRC and Dept. of Cont. & Instr., Seoul Nat'l Univ.)

接受日字 : 1991年 12月 26日

I. 서　론

고정 소수점(fixed-point) 디지탈 시그날 프로세서는 동급의 부동 소수점(floating-point) 디지탈 시그날 프로세서보다 상대적으로 간단한 data-path를 가지고 있기 때문에 수행 속도가 빠르고 가격면에서도 유리하며, 따라서 경제적인 시스템 구현에 적합하다. 그러나 대부분의 고정 소수점 디지탈 시그날 프로세서의 경우, 충분한 wordlength를 가지고 있는 여건에서도 정수형 (integer)이나 소수형(fraction) 데이타 형식을 사용하고 있기 때문에 모든 수를 정확히 나타내기 어렵다. 이러한 문제는 각 변수에 서로 다른 소수점(binary-point)을 할당하여 해결할 수 있는데, 이 경우 산술 연산이나 데이타 이동이 수행될 때마다 정확도(precision)를 유지하면서 오버플로우(overflow)를 방지하기 위해 데이타를 최적으로 스케일링(scaling)해야 한다. 그러나 이를 위한 적당한 쉬프트(shift) 횟수의 결정은 상당히 복잡하다. 또한 고정 소수점 디지탈 시그날 프로세서에서 정수나 소수로 표현된 데이타는 그 값을 즉시 알아내기 어렵기 때문에 프로그래머에게는 또 하나의 문제점이 되고 있다.

이와 같이 고정 소수점 디지탈 시그날 프로세서가 필연적으로 안고 있는 스케일링 문제를 해결하기 위해, 부동 소수점 프로그래밍 환경을 제공하는 고정 소수점 디지탈 시그날 프로세서용 프로그램 개발 방법이 제안되었다.[2] 이 방법의 전체적인 흐름은 그림 1이 보여주고 있다. 우선 프로그래머는 가상의 부동 소수점 하드웨어 모델과 부동 소수점 데이타 형식을 사용하여 어셈블리 프로그램을 작성한다. 어셈블리 프로그램이 작성되면, 위의 하드웨어 모델에 기초하여 시뮬레이션이 수행되고 각 변수의 범위(range)가 결정된다. 여기서 범위는 그 변수가 가질 수 있는 최대값을 추정한 수치이며 스케일링 정도를 결정할 때 사용된다. 부동 소수점 시뮬레이션이 성공적으로 수행되면 어셈블리 프로그램은 고정 소수점용으로 변환되는데, 필요한 쉬프트 횟수는 관계된 변수들의 범위에 대해 간단한 산술 연산을 함으로써 얻어진다. 한편, 한정된 wordlength에 의한 효과(finite wordlengtheffect)를 측정하고자 한다면, 기존의 어셈블러와 시뮬레이터를 사용하여 변환된 고정 소수점 프로그램을 시뮬레이션하고 그 결과를 부동 소수점 형식으로 변환하여 부동 소수점 시뮬레이션 결과와 비교함으로써 가능하다.

본 연구에서는 이 방법을 이용해 고정 소수점 디지탈 시그날 프로세서인 TMS 320C25의 프로그래머

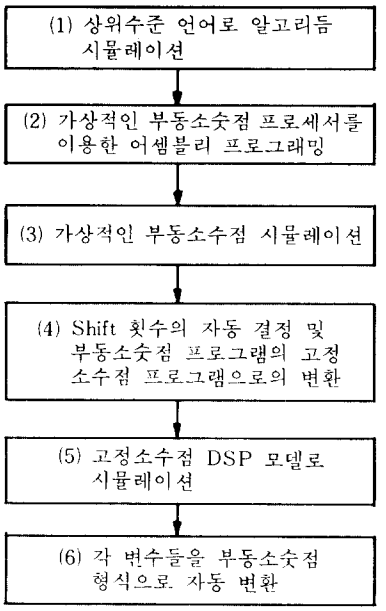

그림 1. 전체적인 개발과정
Fig. 1. The overall development procedure.

를 위해 스케일링이 필요없는 개발 소프트웨어를 제작하였다. 이 소프트웨어의 구성은 가상 하드웨어에 기초한 부동 소수점 어셈블리 코드를 분석하여 중간 코드를 생성하는 구문 분석과 생성된 중간 코드를 가상 하드웨어 상에서 수행시켜 프로그램에서 사용되는 변수의 통계적 특성, 즉 범위를 알아내는 부동 소수점 시뮬레이션, 그리고 알려진 확률적 자료에 근거하여 필요한 쉬프트 횟수를 결정하여 부동 소수점용 프로그램을 고정 소수점용 프로그램으로 변환하는 단계로 이루어져 있다. 특히 선언된 변수에 대해서는 부동 소수점 시뮬레이션 동안에 그 변수의 평균과 표준편차가 구해지고, 이에 의하여 범위가 결정된다. 프로그래머가 본 소프트웨어를 사용하면 어셈블리 프로그래밍에 부동 소수점 형식의 데이타를 사용할 수 있어 편리하며 출력으로 생성된 프로그램이 TMS 320C25에서 실행되므로 수행 속도도 떨어지지 않는다.

또한 여러가지 형태의 필터를 구현하기 위한 부동 소수점 어셈블리 코드를 생성해 주는 코드 생성기(code generator)가 제작되었으며, 자동 스케일링 소프트웨어와 연계하여 사용할 수 있다. 따라서 TMS 320C25 사용자가 코드 생성기로 부터 필요로 하는 특성의 FIR filter나 IIR, 또는 adaptive transversal filter 등의 부동 소수점 어셈블리 프로그램을 쉽게

얻을 수 있으므로 본 소프트웨어의 사용이 더욱 용이하다.

본 논문의 구성은 다음과 같다. 제2장에서는 본 개발 소프트웨어가 사용하는 데이타 표현 방식과 오버플로우를 예방할 수 있는 쉬프트 횟수의 결정 방법이 설명되어 있으며, 더불어 TMS 320C25에서의 스케일링 방법이 제시되어 있다. 제 3장에서는 TMS 320C25의 가상 부동 소수점 하드웨어 모델과 부동 소수점 명령어 및 몇가지 의사명령어가 추가된 새로운 명령어 집합이 소개된다. 개발 소프트웨어의 구성에 대한 해설과 본 소프트웨어의 사용법이 제 4 장을 이루며, 계속해서 제 5장에서 코드 생성기에 대한 내용을 비롯하여 IIR filter와 adaptive FIR filter 에 대한 수행 예를 보이고, 제 6장에서 결론을 맺는다.

Ⅱ. 데이타 표현 방식과 Shift 횟수 결정

여기서 사용하는 '데이타'라는 용어는 샘플된 신호나 계수(coefficients)를 나타내는 변수(variable) 또는 상수(constant)의 값을 말한다. 일반적으로 고정 소수점 데이타 형식을 사용할 때에는 정수나 소수의 표현 방식을 쓰고 있다. 즉, 그림2와 같은 16비트의 데이타의 경우, 정수형(integer)의 데이타이면 소수점(binary point)이 비트 15 다음에 있으며, 소수형(fraction)이면 비트 0(부호) 다음을 소수점의 위치라고 가정한다. 따라서 이 표현 방식으로는 모든 수치를 정확히 표현하기 힘들다. 그러므로 본 연구에서는 각 변수마다 고유한 소수점을 사용하는 표현 방법을 사용하였다.[2] 이때 소수점의 위치는 그 변수가 최대값을 가질 때도 오버플로우가 일어나지 않도록 정해진다. 예를 들어 어떤 변수의 최대값이 8이면, 그림2에서 비트3 다음에 소수점을 둔다.

1. 데이타 표현 방식

제안된 방법에서는 데이타를 세개의 영역(field) 즉, 범위(range) R, 지수(exponent) E, 그리고 가수(mantissa) M으로 나타낸다. 범위는 각 변수가 가질 수 있는 최대값이며, 지수와 가수는 범위로부터 구해진다. 이때 가수는 오버플로우를 일으키지 않는 소수로 표현되어야 하므로 이를 위한 지수는 다음과 같이 결정된다.

$$E(x) \geq \lceil log_2 R(x) \rceil \qquad (1)$$

단 $\lceil x \rceil$는 x보다 크거나 같은 최소의 정수이다. 예를 들어 변수 x[n]이 −4와 4사이의 값을 가진다면 범위는 4이고, 지수는 2가 될 수 있다. 한편 x[100]이 1.0이고 지수가 2이면 가수는 00100…0이 된다. 가수는 항상 소수의 형태로 나타나며 고정 소수점 디지탈 시그날 프로세서의 메모리나 레지스터에 저장된다. 그러나 지수는 프로그래머에게는 알려져 있지만 고정 소수점 프로그램 내에서는 쉬프트 횟수에 의해 암시적으로 저장될 뿐이다. 따라서, 양자화에 의한 잡음(noise)을 무시한다면 변수x의 개념적인 값은 다음과 같이 표시할 수 있다.

$$x = M(x) \cdot 2^{E(x)} \qquad (2)$$

2. 쉬프트 횟수 결정

위의 예에서 x[n]에 대한 산술 연산의 결과로부터 생긴 새로운 변수 z[n]의 범위가 16이 되었다면, 가수는 오버플로우를 일으킨다. 따라서 가수를 하향 스케일 즉, 오른쪽으로 쉬프트해야 하며, 식(2)에 의한 개념적 값, x의 보존을 위해 지수를 증가시켜야 한다. 이 경우 가수는 2비트 오른쪽으로 쉬프트되며, 지수는 4가 된다. 이러한 절차는 다른 산술 연산에 대해

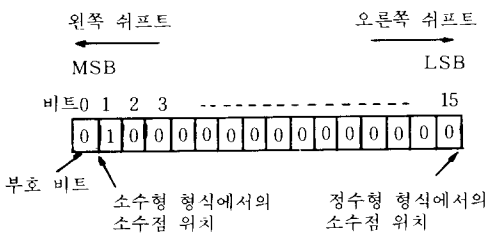

그림 2. 고정 소수점 디지탈 프로세서에서의 데이타 표현 방식

Fig. 2. Data representation method in a fixed point signal processor.

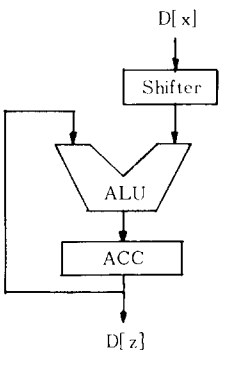

그림 3. Pre-scaling ALU 모델

Fig. 3. A model of pre-scaling type ALU.

서도 확장될 수 있다. 그림3의 pre-scaling ALU 모델을 사용하여 간단한 덧셈 연산에 대해 예를 들어 보도록 하겠다. z=x+y를 어셈블리 코드로 표현하면 다음과 같은 식(3)의 형태를 가진다.

$$ACC<-Sft(sx)D[x]$$
$$ACC<-ACC+Sft(sy)D[y] \qquad (3)$$
$$D[z]<-ACC$$

단 여기서 ACC는 accumulator를 나타내며, D [x]은 변수x에 할당된 메모리의 데이타를 가리키고, Sft (sx)은 sx비트 왼쪽 쉬프트를 의미한다. 즉 변수x의 데이타를 메모리로부터 가져와서 sx비트 만큼 왼쪽 쉬프트를 한 후 accumulator에 저장한다. 그런 다음 다시 변수y의 데이타를 메모리로부터 가져와서 sy비트 만큼 왼쪽 쉬프트한 후 그 값을 accumulator에 더한다. 마지막으로 accumulator의 데이타를 변수 z에 해당하는 메모리에 저장한다. 이때 모든 변수, x, y, z의 지수를 안다고 가정하면 쉬프트 횟수, sx, sy는 다음식(4)와 같이 결정되며, 이렇게 정해진 쉬프트 횟수에 의해 식(3)이 실행될 때에는 오버플로우가 발생하지 않는다.

$$sx=E(x)-E(z)$$
$$sy=E(y)-E(z) \qquad (4)$$

앞의 예를 적용시켜 보면, z의 범위는 16이므로 E(z)=4이고, x의 범위는 4이므로 E(x)=2이다. 한편 변수 y의 범위를 2로 가정하면 E(y)=1이므로 따라서 sx는 −2가 되고 sy는 −3이 된다.

3. TMS 320C25에서의 스케일링 방법

TMS 320C25의 data-path는 그림4와 같은 구조를 갖는다.[1][8][9] 그림에서 보는 바와 같이 세 종류의 쉬프터가 내장되어 있는데 각각 shifter a, shifter m, shifter o로 구분하겠다. Shifter a는 accumulator에 새로운 값을 로드하거나 accumulator와 연산을 하고자할 때 그 데이타의 스케일링을 위한 것이며, shifter m은 곱셈기의 출력을, shifter o는 accumulator의 출력을 스케일링할 때 사용된다.

Shifter a의 동작을 모델링할 때에는 주의를 기울여야 하는데 이는 0비트 쉬프트의 경우, 쉬프트의 16비트 입력이 32비트 출력의 하위 비트(LSB)로 채워지기 때문이다. 따라서 소수형태(fractional format)의 데이타 형식을 사용하는 경우, 0비트 쉬프트는 16비트의 오른쪽 쉬프트의 효과에 해당한다. 즉 shifter a의 전후의 데이타를 제안된 표현 방식으로 나타내면,

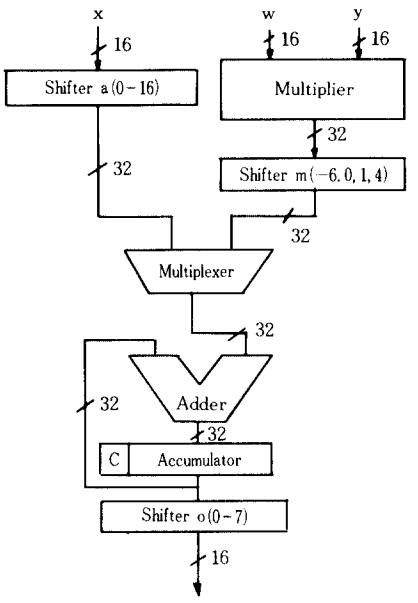

그림 4. TMS 320C25의 ALU 모델
Fig. 4. ALU model of the TMS 320C25.

$$R(post\_x)=R(pre\_x)$$
$$M(post\_x)=M(2^{-16+sa}\cdot pre\_x)$$
$$E(post\_x)=E(pre\_x)-(sa-16) \qquad (5)$$

와 같다. 단 pre_x는 쉬프터의 입력이며, post_x는 쉬프터를 거친 출력이고, sa는 쉬프트 횟수이다.

Shifter a가 내부의 상태 플래그(status flag), SXM에 의해 산술 쉬프트(arithmetic shift)와 논리 쉬프트(logical shift)가 모두 가능한 반면, shifter o는 논리 쉬프트만 가능하다. 따라서 accumulator의 데이타가 shifter o를 통과할 때 부호 비트가 잘려나갈 수 있으며 이 경우 대상 시스템의 성능에 심각한 장애가 될 수 있다. 특히 적응 필터에서 오차 신호를 구할 때와 같이 입력의 범위보다 출력의 범위가 작은 경우, 즉 shifter a의 오른쪽 쉬프트의 횟수보다 shifter o의 왼쪽 쉬프트 횟수가 많을 때에는 부호 비트가 잘려나갈 가능성이 높다. 이와 같은 문제점의 해결책으로는 모든 변수에 가드 비트를 할당하는 방법과 출력 변수의 지수만을 증가시키는 방법이 있다. 그러나 첫번째 해결 방안으로는 프로그램의 다른 부분에 대한 파급효과가 고려되지 않으므로 이 두가지 대책간의 trade-off가 필요하다.

곱셈기와 ALU를 사용하여 곱셈 결과를 더해가는 연산(multiplication and accumulation)은 일반적으로 다음과 같이 표현될 수 있다.

$$ACC=\sum Sft(sa[i])x[i]+\sum Sft(sm)(w[j]\cdot y[i])$$
$$z=Sft(so)ACC \qquad (6)$$

단 모든 w[j]와 모든 y[j]의 지수는 같다고 가정한다. 이때 입력의 지수와 accumulator의 지수, 그리고 곱셈 결과의 지수 중에서 최대값을 E(t)라고 하여 그림5와 같은 절차에 따라 각 쉬프트의 횟수를 결정한다. 우선 shifter m의 횟수를 결정해야 하는데 그 이유는 쉬프트 가능 횟수가 연속적이지 않고 단지 −6, 0, 1, 그리고 4만이 제공되기 때문이다. 쉬프트 횟수는 다음 식(7)을 만족하는 sm 중에서 가장 큰 값으로 결정되지만 곱셈기의 출력이 항상 정규화(normalize)되는 것은 아니다.

$$sm=-6, 0, 1, \text{또는 } 4$$
$$E(ACC)=E(w[j])+E(y[j])+1-sm$$
$$E(ACC)\geq E(t) \qquad (7)$$

sm이 결정되면 E(ACC)도 같이 결정되므로 shifter a와 shifter o의 쉬프트 횟수인 sa[i]와 so는 다음 식(8)에 의해 간단히 계산된다.

$$sa[i]=E(x[i])+16-E(ACC)$$
$$so=E(ACC)-E(z) \qquad (8)$$

결정된 shifter o의 왼쪽 쉬프트 횟수가 shifter a의 오른쪽 쉬프트 횟수보다 크면 출력변수, z의 지수를 하나 증가시켜 전체적인 쉬프트 횟수의 결정 과정을 다시 밟는다.

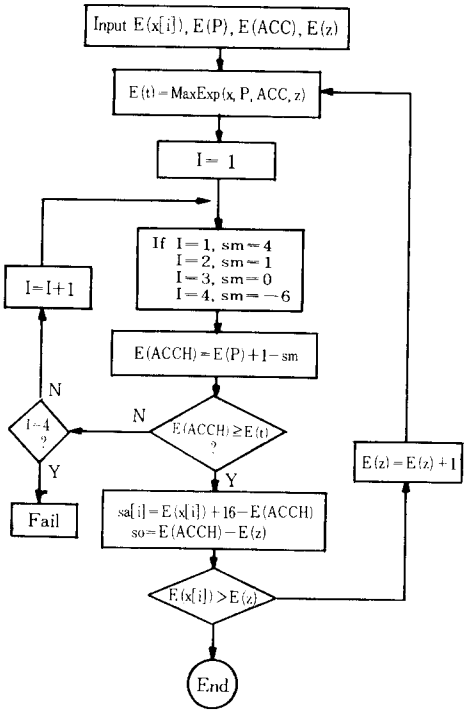

그림 5. TMS 320C25에서의 쉬프트 횟수 결정 방법
Fig. 5. A method of determining the number of shift for the TMS 320C25.

Ⅲ. TMS 320C25의 가상 하드웨어 모델 및 명령어 집합

1. TMS 320C25의 가상 하드웨어 모델

프로그래머는 TMS 320C25의 기본적인 구조를 가지면서 부동 소수점 데이타의 연산이 가능한, 그림 6과 같은 가상 하드웨어 모델을 가정한다. 메모리의 wordlength는 32비트로서, 부동 소수점 데이타를 저장할 때에는 32비트 모두가 사용되지만 프로그램 코드나 정수형 데이타가 저장될 때에는 하위 16비트만이 사용된다. 부동 소수점 데이타는 IEEE 표준 형식을 가지며, 상위 8비트의 지수부와 1비트의 부호비트 그리고 하위 23비트의 가수부로 나누어 진다. 본 가상 부동 소수점 디지탈 시그날 프로세서는 부동 소수점과 고정 소수점 연산 기능이 있으나, 인자의 형식이 부동 소수점형과 고정 소수점형 간의 혼합일 때에는 연산이 허용되지 않는다. 한편 shifter a, shifter m, shifter o는 고정 소수점 연산이 수행될 때만 동작하고 부동 소수점 연산이 수행될 때에는 바이패스(bypass) 된다.

2. 명령어 집합

TMS 320C25가 고유하게 갖는 고정 소수점 명령어 외에 부동 소수점 데이타를 처리하는 명령어들이

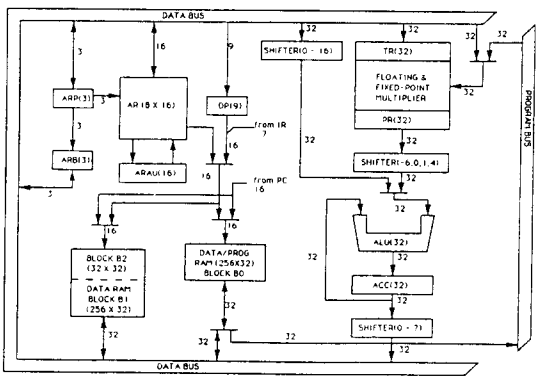

그림 6. 가상 하드웨어 모델
Fig. 6. A model of hypothetical hardware.

추가되는 새로운 명령어 집합을 이룬다[1][5]. 추가된 부동 소수점 명령어는 ALU에 관계되는 명령들이며, 프로그램 콘트롤, 비트 테스트, 데이타 메모리간의 명령어 등은 바뀌지 않는다. 부동 소수점 명령어에 대한 설명, 그리고 대응하는 고정 소수점 명령어가 표 1에 기술되어 있다.

표 1. 부동 소수점 명령어와 해당 고정 소수점 명령어

Table 1. Floating-point instructions and corresponding fixed-point instructions.

	부동 소수점 명령어	해당 고정 소수점 명령어
명령어	동 작 설 명	
ADDF	Accmulator에 더함	ADD
ADLKF	Accmulator에 상수를 더함	ADLK
APACF	P 레지스터의 값을 Accmulator에 더함	APAC
FIX	부동 소수점 데이타를 정수형으로 바꿈	-
FLOAT	정수형 데이타를 부동 소수점형으로 바꿈	-
INF	부동 소수점 데이타를 입력	IN
LACF	Accmulator에 로드	LAC
LACKF	Accumulator에 상수를 로드	LACK, LALK
LTF	T 레지스터에 로드	LT
LTAF	T 레지스터에 로드, P 레지스터의 값을 accumulator에 더함	LTA
LTDF	T 레지스터에 로드, P 레지스터의 값을 accumulator에 더함, 데이타 이동	LTD
LTSF	T 레지스터에 로드, P 레지스터의 값을 accumulator에서 뺌	LTS
MACF	곱하고 더함	MAC
MACDF	곱하고 더함, 데이타 이동	MACD
MPYF	곱함(T*Operand->P 레지스터)	MPY
MPYAF	곱하고 P 레지스터 값을 acc.에 더함	MPYA
MPYSF	곱하고 P 레지스터 값을 acc.에서 뺌	MPYS
OUTF	부동 소수점 데이타를 출력	OUT
SACF	Accumulator의 값을 메모리에 저장	SACH
SPACF	P 레지스터의 값을 accumulator에서 뺌	SPAC
SQRAF	제곱해서 accumulator에 더함	SQRA
SQRSF	제곱해서 accumulator에서 뺌	SQRS
SUBF	Accumulator에서 뺌	SUB
ZACF	Accumulator의 값을 0으로 만듬	ZAC

Accumulator나 P 레지스터 또는 T 레지스터 등과 같은 내부 레지스터나 메모리는 경우에 따라 부동 소수점 또는 고정 소수점 데이타를 가질 수 있으므로 인자(operand)로 사용할 때에는 같은 데이타 형식의 명령어만을 써야 한다.

3. 의사 명령어

TMS 320C25가 제공하는 명령어 집합 외에 몇가지 의사 명령어가 추가로 지원된다. 시뮬레이션에 의해 범위를 알고자 하는 부동 소수점 변수를 선언할 때에는 의사 명령어 'SEQU'를 사용한다. 계수와 같은 부동 소수점 상수나 한 변수의 지연된 값과 같이 확률 분포를 알 필요가 없는 부동 소수점 변수는 의사 명령어 'FEQU'로써 선언하며, 고정 소수점 변수를 선언하고자할 때에는 의사 명령어 'IEQU'를 사용한다. 또한 프로그램 내에서 부동 소수점 상수를 정의하고자할 때에는 'DATAF'가 쓰인다. 문법은 다음과 같다.

변수이름　SEQU　주소
변수이름　FEQU　주소[, 관계된 변수이름]
　　[, 변수갯수]
변수이름　IEQU　주소[, 변수갯수]
　　DATAF　부동 소수점 상수

FEQU를 사용하여 부동 소수점 변수를 선언할 경우, 선언하고자 하는 변수가 어떤 변수의 지연된 값이라면 그 변수의 이름을 '관계된 변수이름' 영역에 표시하여 통계적 특성이 같음을 나타낼 수 있다. 또한 같은 형식의 변수를 명시한 주소와 인접하여 중복 선언하고자 한다면, 그 갯수를 '변수갯수' 영역에 사용한다.

$$y[n]=0.5*y[n-1]+x[n] \qquad (9)$$

그림7에서는 위 식(9)와 같은 1차 IIR filter를 사용하여 부동 소수점 명령어와 고정 소수점 명령어의 프로그래밍 예를 보이고 있다.

```
XN   SEQU   >20          XN   EQU   >20
YN   SEQU   >21          YN   EQU   >21
YD1  FEQU   >22, YN      YD1  EQU   >22
A1   FEQU   >23          A1   EQU   >23
  . .                        . . . . .
     DATAF  0.5               DATA  >4000
  . . . . .                  . . . . .
     INF    XN, PA2           IN    XN, PA2
     LACF   XN                LAC   XN, 14
     LTF    YD1               LT    YD1
     ***                      SPM   1
     MPYF   A1                MPY   A1
     APACF                    APAC
     SACF   YN                SACH  YN, 0
     DMOV   YN                DMOV  YN
```

그림 7. 부동 소수점 명령어와 고정 소수점 명령어 사용한 프로그램 예

Fig. 7. Examples using floating and fixed-point instruction sets each.

Ⅳ. 소프트 웨어

본 소프트웨어의 구성은 다음과 같은 3단계로 이루어져 있다.

1. 구문분석

이 구문분석은 2패스(pass) 알고리듬으로 작성되었다. 첫번째 패스 과정에서는 가상 하드웨어에 기초한 부동 소수점 어셈블리 프로그램을 분석하여 사용되는 변수, 상수, 레이블들을 구분해 낸다. 선언된 변수에 대해서는 적절한 데이타 영역을 각각 할당하고, 상수나 레이블 등에 대해서는 이들이 사용된 메모리 번지나 초기값 등을 표로 작성한다.[10] 따라서 레이블의 전향 참조(forward reference)가 가능하며 변수나 상수에 대해서는 그들이 위치하는 메모리 번지를 쉽게 알 수 있을 뿐만 아니라 그 데이타의 참조도 용이하다. 특히 의사 명령어 'SEQU'로 선언된 변수의 표에는 합과 제곱합의 영역이 있어 그 변수의 통계적 특성이 기록된다. 그러나, 문법적인 오류가 발견될 때에는 오류의 종류를 표시하고 소프트웨어의 실행을 중단한다.

두번째 패스 과정에서는 프로그램에서 사용되는 레이블(label), 명령어, 인자(operand) 그리고 설명어(comment) 등을 각각의 영역에 따라 구분하여 중간 형태의 코드를 생성한다. 이때 'EQU'로 선언된 상수는 그 값으로 대체되며 레이블이나 변수는 그들이 선언된 번지로 변환되어 저장된다. 또한 변수나 상수 또는 레이블 등의 심볼(symbol)을 포함한 산술식으로 표현된 인자는 그 식의 값이 평가되어 사용된다. 한편 프로세서의 내부 레지스터인 T 레지스터와 accumulator에 대한 표도 준비되어 부동 소수점 시뮬레이션을 대비한다.

2. 부동 소수점 시뮬레이션

구문 분석 과정에서 생성된 중간 코드가 TMS 320C25에 근거한 가상적인 부동 소수점 하드웨어 모델에 의하여 실행된다. 프로그램은 사용자가 지정한 횟수 만큼 실행되며, 입력은 데이타 화일이나 난수를 발생시켜 사용할 수 있다. 또한 선언된 변수는 시뮬레이션 데이타를 화일로 받을 수도 있다.

Accumulator에 어떤 값을 정해주거나 0으로 만드는 명령어가 수행될 경우에는, 독자적인 범위를 할당하기 위해서 새로운 프로그램 단락이 시작된다. 변수의 값을 변화시키는 명령어가 수행되면 그 변수의 값이 변경될 뿐만 아니라 그 변수에 대한 합과 제곱합의 수치도 갱신된다. 또한 각 프로그램 단락에 대하여 곱셈기의 두 입력(T 레지스터 포함)과 accumulator에 대한 변화 과정이 기록되어 쉬프트 횟수를 정하는 자료가 된다.

3. TMS 320C25용 프로그램으로의 변환

부동 소수점 시뮬레이션이 완료되면 실행 횟수 및 각 변수와 내부 레지스터의 합과 제곱합을 알 수 있으므로 이로부터 평균(m)과 표준편차(σ)를 계산한다. 계산된 통계적 수치, m, σ로부터 다음 식(10)과 같이 변수의 범위를 추정할 수 있다.[3]

$$R = |m| + n\sigma \tag{10}$$

위 식에서 사용한 n의 값은 제 7장의 시뮬레이션 결과에 비추어 볼 때 4인 경우가 가장 적당함을 알 수 있었다.

추정된 범위로부터 식(1)에 의하여 내부 레지스터와 각 변수들의 지수를 계산할 수 있으며, 이때 가드 비트에 의해 오버플로우에 대한 안정성을 높일 수 있다. 지수가 결정되면 그림 5의 절차에 따라 쉬프트 횟수가 결정된다. 쉬프트 횟수 sa는 acumulator에 어떤 값을 로드(load) 할 경우에 필요한 값이며, sm은 곱셈기의 출력에, so는 accumulator의 값을 메모리로 저장할 때 쓰인다. 결정된 쉬프트 횟수를 바탕으로 하여 부동 소수점 명령어는 대응되는 고정 소수점 명령어로 변환되고 최종적으로 TMS 320C25 어셈블리 프로그램이 생성된다.

4. 프로그램 사용법

본 소프트웨어는 C언어로 작성되었고 약 4,000 줄의 분량이며 IBM PC 호환기종에서 수행된다.[7] 프로그램의 실행 방법은 다음과 같다.

C : /> AUTOSCAL Finame. Ext ↵

여기서 Filename. Ext은 부동 소수점 프로그램이다. 본 소프트웨어가 실행되면 어셈블리 프로그램의 시뮬레이션 횟수가 가드 비트의 크기, 그리고 몇배의 표준 편차로 범위를 추정할 것인가에 대한 자료를 사용자로부터 대화적으로 얻는다. 부동 소수점 시뮬레이션 중에는 임의의 데이타 화일이나 난수를 입력으로 사용할 수 있으며, 시뮬레이션 결과를 화일로 저장할 할 수도 있다. 프로그램이 종료되면 선언된 변수들의 통계적 특성이 저장된 Filename. out과 스케일링된 고정 소수점 어셈블리 프로그램인 Filename. as가 생성된다. 또한 사용자가 데이타 화일을 지정한 경우, 그 화일에 시뮬레이션 데이타가 저장되어 출력된다.

Ⅴ. 코드 생성기 및 실행 예

1. 코드 생성기

어셈블리 언어를 사용하여 시스템을 개발하는 작업은 상위 레벨 언어를 사용하는 경우보다 상대적으로 많은 시간과 노력을 필요로 한다. 이는 어셈블리 언어 자체가 사용하기 힘들다는 측면 뿐만 아니라 대상 하드웨어가 갖는 특수성도 고려해야 한다는 이유 때문이다. 즉 대상 어셈블리 문법에 따른 디버깅(debugging) 작업과 목적하는 동작을 제대로 수행하는가에 대한 검증을 위해 개발 시간을 할애해야 하기 때문에 전체적인 개발 기간이 연장된다. 따라서 일반적으로 많이 쓰이는 필터등에 대해 그 동작이 검증된 어셈블리 프로그램이 제공된다면 시스템의 개발 시간의 단축에 큰 도움이 될 것이다.

본 논문에서는 부동 소수점 명령어를 사용하여 여러가지 형태의 필터 프로그램을 생성해 주는 코드 생성기(code generator)가 제작되었다.[4] 생성 가능한 필터는 FIR filter, IIR filter, 그리고 adaptive transversal filter 등이며, 특히 IIR filter는 direct form, cascade form, 그리고 parallel form 등 서로 다른 구조로의 구현이 가능하다. 코드 생성기의 사용자는 대화적으로 필터의 종류와 구조, 그리고 필터의 차수(order) 및 계수 값 등을 입력함으로써 원하는 특성을 갖는 부동 소수점 어셈블리 프로그램을 얻을 수 있다.

2. 실행 예

제작된 코드 생성기를 사용하여 2차의 타원형(elliptic) IIR 필터와 4탭의 적응 트랜스버설 필터를 얻었다. 이들 두가지의 부동 소수점 어셈블리 프로그램에 대하여 본 소프트웨어를 수행시켰고, 각각의 수행 조건하에서 얻어진 자료를 분석하였다.

1) IIR Filter

새로운 명령어 집합에 따라 2차의 타원형(elliptic) IIR filter를 구현하는 어셈블리 프로그램을 코드 생성기를 사용하여 작성하였다(그림 8). 부동 소수점 데이타를 인자로 사용하고자 할 때에도 16비트의 2 진수나 네자리의 16진수로 바꿀 필요없이 소수 형태 그대로 사용하고 있다.

소프트웨어가 실행되면 어셈블리 프로그램이 구문분석을 거치면서 중간 코드가 생성되고, 이를 바탕으로 한 부동 소수점 시뮬레이션에 의해 각 변수의 범위가 결정된다(그림 9). 계산된 지수로부터 쉬프트 횟수가 결정되고 TMS 320C25의 어셈블리 프로그램이 생성된다. 범위 추정에 4배의 표준편차를 사용한 경우의 출력 프로그램은 그림 10과 같으며 7배의 표준편차를 사용한 경우에도 같은 출력 프로그램을 얻었다.

```
*    Second-order elliptic IIR lowpass filter
*
*      pass band edge 0.2*PI
*      stop band edge 0.3*PI
*
XN    SEQU  >60       * input
YN    SEQU  >61       * output
UN    SEQU  >62       * temporary variable
UD1   FEQU  >63,UN    * delayed samples
UD2   FEQU  >64,UN
A1    FEQU  >65,,2    * coefficients
B0    FEQU  >67,,2    * coefficients
*
      AORG  0
      B     START
* coefficients
COEFF AORG  32
      DATAF 1.59567   * a11
      DATAF -0.86490  * a21
      DATAF 0.99999   * b01 = b21
      DATAF -0.74048  * b11
* start processing
START EQU   $
      LDPK  0         * DP = 0
      LARK  AR0,A1    * AR0 = &A1
      LARP  0         * ARP = 0
      RPTK  3
      BLKP  COEFF,*+  * coefficients load
* initialize filter
      ZACF
      SACF  UN        * initialize temp. var's
      SACF  UD1
      SACF  UD2
* main body of filter
WAIT  BIOZ  CONT
      B     WAIT
CONT  INF   XN,PA2    * input data sample
      LACF  XN        * load XN
      LTF   UD1       * load T with de.ayed sample
      MPYF  A1        * multiply T with coeff A1
      LTAF  UD2       * load T with de.ayed sample
      MPYF  A1+1      * multiply T with coeff A2
      APACF
      SACF  UN        * store temp. variable
*
      ZACF
      MPYF  B0        * multiply T with coeff B2
      LTDF  UD1       * load T with delayed sample
      MPYF  B0+1      * multiply T with coeff B1
      LTDF  UN        * load T with delayed sample
      MPYF  B0        * multiply T with coeff B0
      APACF
      SACF  YN        * store output sample
* D/A converter
      OUTF  YN,PA2
      B     WAIT      * retry
*
      END
```

그림 8. 부동 소수점 명령어를 사용한 타원형 2차 IIR 필터

Fig. 8. A 2nd-order elliptic IIR filter using floating-point instruction set.

었다.

$$SNR=20\log_{10}\frac{\sum y_f^2[n]}{\sum (y_f[n]-y[n])^2} \qquad (11)$$

얻어진 고정 소수점 프로그램을 같은 데이타에 대해 100,000번씩 수행시킨 결과, 다음 표 2와 같은 결과를 얻었다. 신호 대 양자화 잡음비(SQNR, signal

```
Estimating the ranges...
        Mean     StdD      Range     AMax     Exp
XN    -0.083   +0.561    +1.000    +1.000     0
YN    -0.461   +2.339    +9.818    +5.663     8
UN    -0.323   +2.349    +9.721    +6.145     8

  P[2]  E(UN)+E(Coeff.)=8+1=9
Acc[2]-0.363+2.583+10.694+9.243  4
  P[3]  E(UN)+E(Coeff.)=8+0=8
Acc[3]-0.337+1.992+1.992+8.308+6.144  4

Determining the numbers of shifts...
   sa[2]=11  sm[2]=1  so[2]=1
             sm[3]=1  so[3]=0
```

그림 9. 변수의 통계적 특성을 보여주는 시뮬레이션 결과

Fig. 9. Result of simulation showing statistical characteristics of variables used.

표 2. 표준편차 수에 따른 SQNR 비교

Table 2. Comparison of SQNR according to the number of standard deviation.

표준편차 수(σ)	2 - 3	4 - 7	8 - 14	15 - 28
오버플로우 횟수	505	0	0	0
SQNR(dB)	1.02	49.66	45.10	39.50

to quantization noise ratio)의 단위는 데시벨(decibel)이며, 위 식(11)을 사용하였다. 단 y_f는 고정 소수점 연산에 의한 출력이다.

2) 적응 트랜스버설(transversal) 필터

그림11과 같은 적응 채널 식별(channel identification) 시스템에 사용되는 4-탭 적응 트랜스버설 필터를 코드 생성기를 사용하여 LMS(least mean square) 알고리즘으로 구현하였다.[6] 시뮬레이션 입력, u_n, v_n으로는 서로 다른 난수 발생기의 데이타 1000개를 사용하였다. 범위 추정에는 4배의 표준편차와 다섯 종류의 가드 비트를 사용하였다. 각각의 가드 비트에 대한 성능 분석자료는 표 3과 같다.

표 3. 구현 방법에 따른 적응 채널 식별 시스템의 성능 비교

Table 3. Comparison of performance in an adaptive channel identification system according to the implementation methods.

가드 비트	0	1	2	3	4	부동 소수점
오버플로우	17	0	0	0	0	-
SNR(dB)	-1	42	40	37	32	40

```
*    Second-order elliptic IIR lowpass filter
*
*      pass band edge 0.2*PI
*      stop band edge 0.3*PI
*
XN    EQU   >60     * input
YN    EQU   >61     * output
UN    EQU   >62     * temporary variable
UD1   EQU   >63     * delayed samples
UD2   EQU   >64
A1    EQU   >65     * coefficients
B0    EQU   >67     * coefficients
*
      AORG  0
      B     START
* coefficients
COEFF AORG  32
      DATA  >661F  0.797832*2(1)  * a11
      DATA  >C8A5  -0.43245*2(1)  * a21
      DATA  >7FFF  0.999999*2(0)  * b01 = b21
      DATA  >A138  -0.74048*2(0)  * b11
* start processing
START EQU   $
      LDPK  0         * DP = 0
      LARK  AR0,A1    * AR0 = &A1
      LARP  0         * ARP = 0
      RPTK  3
      BLKP  COEFF,*+  * coefficients load
* initialize filter
      ZAC
      SACH  UN        * initialize temp. var's
      SACH  UD1
      SACH  UD2,0
* main body of filter
WAIT  BIOZ  CONT
      B     WAIT
CONT  IN    XN,PA2    * input data sample
      LAC   XN,11     * load XN
      SPM   1
      LT    UD1   * load T with delayed sample
      MPY   A1    * multiply T with coeff A1
      LTA   UD2   * load T with delayed sample
      MPY   A1+1  * multiply T with coeff A2
      APAC
      SACH  UN,1      * store temp. variable
*
      ZAC
      MPY   B0    * multiply T with coeff B2
      LTD   UD1   * load T with delayed sample
      MPY   B0+1  * multiply T with coeff B1
      LTD   UN    * load T with delayed sample
      MPY   B0    * multiply T with coeff B0
      APAC
      SACH  YN,0      * store output sample
* D/A converter
      OUT   YN,PA2
      B     WAIT      * retry
*
      END
```

그림10. TMS 320C25용 타원형 2차 IIR 필터

Fig. 10. A 2nd-order elliptic IIR filter for the TMS 320C25.

여기서 SNR은 적응 필터가 수렴한 이후의 신호와 오차의 파워(power)비를 데시벨로 표시한 값이다. 가드 비트가 0인 경우에는 accumulator의 데이타가 shifter 0를 통과할 때 부호 비트가 잘려나가 적응 필터가 동작하지 않았다. 가드 비트가 1보다 클 때에는 정상적으로 동작하며, 가드 비트가 증가함에 따라 양

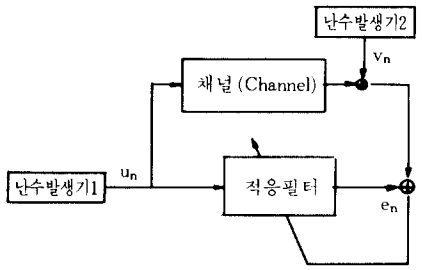

그림11. 적응 채널 식별 시스템의 블럭도
Fig. 11. Block diagram of an adaptive channel identification system.

자화 잡음이 커짐을 SNR이 감소함을 통해 알 수 있다.

위의 두가지 예에 대한 시뮬레이션 결과를 분석해 보면 4배의 표준편차에 의한 범위 추정이 가장 적당하며, 구현 대상에 따라 적절한 가드 비트가 필요함을 알 수 있다.

Ⅳ. 결 론

TMS 320C25의 프로그래머에게 부동 소수점 디지탈 시그날 프로세서와 같은 프로그래밍 환경을 제공하는 소프트웨어가 제작되었다. 이 소프트웨어에서는 가상 부동 소수점 시그날 프로세서 하드웨어 모델을 설정하였으며, 이에 해당하는 부동 소수점 명령어 집합을 새로이 구성하였다. 각 고정 소수점 변수는 부동 소수점 시뮬레이션으로부터 얻어진 자신의 범위(range)에 따라 고유한 소수점을 가지며, 산술 연산이나 데이타 이동 시에 필요한 쉬프트 횟수는 각 변수의 소수점 위치로부터 결정된다. 프로그래머는 부동 소수점 데이타와 부동 소수점 명령어로 비교적 쉽게 어셈블리 프로그램을 작성하며, 이 부동 소수점 프로그램은 고정 소수점 형식으로 변환된 후 TMS 320C25에 의해 수행되기 때문에 실행 속도가 줄지 않는다. 따라서 고정 소숫점 디지탈 시그날 프로세서의 경제성과 부동 소숫점 프로그래밍의 편리함을 결합시킬 수 있다. 또한 이를 이용하여 여러가지 형태의 필터를 고정 소수점 디지탈 시그날 프로세서로 구현하는 코드 생성기도 개발되었다.

參 考 文 獻

[1] *TMS 320C25 User's Guide,* Houston, Texas Instruments Inc., 1986.

[2] W. Sung, "An Automatic Scaling Method for the Programming of Fixed-point Digital Signal Processors," *IEEE International Symposium on Circuits and Systems '91,* pp. 37-40, Singapore, June 1991.

[3] L. B. Jackson, "On the Interaction of Roundoff Noise and Dynamic Range in Digital Filters," *Bell Syst. Tech. J.* vol. 49, pp. 159-184, Feb. 1970.

[4] A. V. Oppenheim, R.W. Schafer, *Discrete-time Signal Processing,* Prentice-Hall, Englewood Cliffs, NJ , 1989.

[5] *TMS 320C4x User's Guide,* Houston, Texas Instruments Inc., 1991.

[6] S. Haykin, *Adaptive Filter Theory,* Prentice-Hall, Englewood cliffs, NJ, 1986.

[7] B.W. Kernighan, D.M. Ritchie, *The C Programming Language,* 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1988.

[8] E.A. Lee, "Porgrammable DSP Architectures Part I," *IEEE ASSP Magazine,* October 1988.

[9] E.A. Lee, "Programmable DSP Architectures: Part II," *IEEE ASSP Magazine,* January, 1989.

[10] 조유근, 시스템 프로그래밍, 희중당, 서울, 1985.

著 者 紹 介

金 時 鉉(學生會員)
1968年 2月 27日生. 1990年 2月 서울대 제어계측공학과 졸업. 1992年 2月 서울대 제어계측공학과 석사학위 취득. 1992年 3月~ 서울대 제어계측공학과 박사과정 재학중. 주관심분야는 병렬처리 시스템과 VLSI 신호처리 등임.

成 元 鎔 (正會員) 第28卷 B編 第5號 參照
현재 서울대학교 반도체공동연구소 및 제어계측공학과 조교수