비선형 함수 연산을 위한 FPGA 기반의 부동 소수점 프로세서의 설계

김 정 섭, 정 슬
충남대학교 메카트로닉스공학과

Design of a Floating Point Processor for Nonlinear Functions on an Embedded FPGA

Jeong-Seob, Kim and Seul Jung
Department of Mechatronics Engineering, Chungnam National University

Abstract - This paper presents the hardware design of a 32bit floating point based processor. The processor can perform nonlinear functions such as sinusoidal functions, exponential functions, and other nonlinear functions. Using the Taylor series and the Newton - Raphson method, nonlinear functions are approximated. The processor is actually embedded on an FPGA chip and tested. The numerical accuracy of the functions is compared with those computed by the MATLAB.

1. 서 론

최근 지능형 로봇이나 전자 제어 시스템이 많이 사용되는 자동차 그리고 그 밖의 많은 가전 기기에서 자체적인 프로세서를 가지는 임베디드 시스템의 사용이 많아지고 있다. SONY의 "PSP"나 닌텐도의 "닌텐도 DS" 등의 휴대용 게임기에도 고성능의 임베디드 프로세서가 내장되어 있으며, 핸드폰에도 통신뿐만 아니라 이미지 프로세싱과 무선 인터넷 등의 처리를 위한 높은 사양의 임베디드 시스템이 내장되어 있다. 이러한 업계의 수요에 따라 점차 고성능의 CPU의 요구가 많아지면서 임베디드 프로세서로 유명한 ARM 프로세서와 DSP 제품군의 사용이 늘어가고 있다.

데이터 처리량이 많은 이미지 프로세싱을 FPGA 상에서 구현하는 방법에 관한 많은 연구가 이루어지고 있으며, 제어 분야에서도 FPGA를 이용하여 다양한 제어기를 구현하고자 하는 연구가 있었다[1]. 하지만 대부분의 경우 구현의 복잡성으로 인해 고정소수점 기반으로 연구가 이루어지며 비선형 함수의 연산이 필요할 경우 Look-Up Table(LUT)을 많이 사용하고 있다[2]. 그로 인해 부동소수점 기반의 연산에 비해 정확도가 낮아지며 표현 가능한 수의 범위에도 한계가 있을 수밖에 없다.

FPGA를 이용한 제어기 설계의 경우 기본적인 제어 알고리즘의 계산 이외에도 기구학이나 동역학 연산이 필요하기도 하다. 따라서 로봇 제어에 필요한 제어기의 설계에 있어서 덧셈기나 곱셈기, 비교기나 쉬프터 등의 고정된 모듈로 모든 수식을 표현하는 것은 전체적인 연산 속도는 빠를 수 있을지 모르나 제어 알고리즘이 복잡하고 동역학 연산량이 많으며 제어해야할 축이 많다면 너무 많은 FPGA 용량이 필요하게 된다. 따라서 제어 주기에 영향을 미치지 않는 범위에서 제한된 연산 모듈을 통해 순차적으로 처리하는 것이 보다 더 효율적이다. 최근에는 SIMD(Single Instruction Multiple Data) 방식을 이용하여 순차처리 기법과 병렬처리 기법을 동시에 사용하는 연구도 행해지고 있다[3].

본 논문에서는 부동소수점 기반의 덧셈기, 뺄셈기, 곱셈기 그리고 나눗셈기를 포함한 수치 연산 모듈을 설계하여 로봇 제어에 필요한 비선형 함수를 순차적인 연산을 통해 구할 수 있는 시스템을 소개한다. 각각의 연산을 위한 제어 코드인 명령어(instruction)를 자체적으로 설계하였으며, 이의 해석 및 시스템의 전체적인 제어를 순차적으로 처리하기 위한 파이프라인 구조를 설계하였다.

제안된 부동소수점 프로세서를 FPGA에 구현하였으며 이를 통해 대표적인 비선형 함수인 삼각함수, 지수함수 그리고 제곱근을 Taylor series와 Newton-Rhapson method를 이용하여 계산하였다. 실험을 통해 FPGA상에서 계산된 비선형 함수의 결과를 MatLab에서의 결과와 비교하여 그 정확성을 검증하였다.

2. 비선형 함수의 전개

2.1 Taylor-Maclaurin Series

비선형 함수를 선형화 시키는 방법 중 하나는 Taylor 급수 전개에 의한 방법이다. 특히 x(0) = 0일 경우에 해당하는 Maclaurin 급수 전개에 의해 삼각함수나 지수함수를 선형화 시킬 수가 있다.

삼각함수의 Taylor-Maclaurin 급수의 일반화된 식은 아래와 같다.

$$\sin(x)=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1} \tag{1}$$

$$\cos(x)=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}x^{2n} \tag{2}$$

식 (1), (2)를 전개시키면 아래와 같이 나타낼 수 있다.

$$\sin(x)=x-\frac{x^3}{6}+\frac{x^5}{120}-\frac{x^7}{5040}+\frac{x^9}{362880}-\cdots \tag{3}$$

$$\cos(x)=1-\frac{x^2}{2}+\frac{x^4}{24}-\frac{x^6}{720}+\frac{x^8}{40320}-\cdots \tag{4}$$

Horner Scheme을 사용하여 사칙 연산의 횟수를 줄일 수가 있는데, 식 (3), (4)를 6차까지 전개하여 Horner scheme을 사용하여 아래와 같이 정리할 수 있다.

$$\sin(x)=((\cdots(\frac{x^2}{156}-1)\frac{x^2}{110}+1)\frac{x^2}{72}-1)\frac{x^2}{42}+1)\frac{x^2}{20}-1)\frac{x^2}{6}+1)x \tag{5}$$

$$\cos(x)=((\cdots(\frac{x^2}{132}-1)\frac{x^2}{90}+1)\frac{x^2}{56}-1)\frac{x^2}{30}+1)\frac{x^2}{12}-1)\frac{x^2}{2}+1)x \tag{6}$$

x의 범위가 [-π/2, π/2]일 경우, 6차까지 전개에 의해 발생할 수 있는 수렴 오차를 10^{-9} 이내로 줄일 수가 있다.

지수함수의 Taylor-Maclaurin 급수의 일반화된 식은 아래와 같다.

$$e^x=\sum_{n=0}^{\infty}\frac{1}{n!}x^n \tag{7}$$

$$e^{-x}=\sum_{n=0}^{\infty}\frac{1}{n!}(-x)^n \tag{8}$$

식 (7), (8)을 전개시키면 아래와 같이 나타난다.

$$e^x=1+\frac{x}{2}+\frac{x^2}{6}+\frac{x^3}{24}+\frac{x^4}{120}+\cdots \tag{9}$$

$$e^{-x}=1-\frac{x}{2}+\frac{x^2}{6}-\frac{x^3}{24}-\frac{x^4}{120}+\cdots \tag{10}$$

식 (9), (10)을 8차까지 전개하여 Horner scheme을 사용하여 아래와 같이 정리할 수 있다.

$$e^x=((\cdots(\frac{x}{8}+1)\frac{x}{7}+1)\frac{x}{6}+1)\frac{x}{5}+1)\frac{x}{4}+1)\frac{x}{3}+1)\frac{x}{2}+1)x+1 \tag{11}$$

$$e^{-x}=((\cdots(\frac{x}{8}-1)\frac{x}{7}+1)\frac{x}{6}-1)\frac{x}{5}+1)\frac{x}{4}-1)\frac{x}{3}+1)\frac{x}{2}-1)x+1 \tag{12}$$

x의 범위가 [0, 0.5]일 경우, 8차까지 전개에 의해 발생할 수 있는 수렴 오차를 10^{-8} 이내로 줄일 수가 있다.

2.2 Newton-Raphson Method

제곱근을 구하는 수치 해석적 방법으로 Newton-Raphson method가 있다. 이는 숫자로 표현된 방정식을 푸는데 사용되는 강력한 기법이다. 그 방정식에 대한 적절한 근사값을 가지고 있을 경우, 반복을 이용하여 빠르게 그 해를 구할 수가 있다.

Newton-Raphson method는 $f(x)=0$의 형태의 방정식에 적용된다. 여기서 f(x)란 미분 가능한 함수로 그 미분 함수는 $f'(x)$로 표현된다.

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)} \tag{13}$$

여기에서 $f(x)=0$을 만드는 방향으로 반복연산을 수행하게 되는데,

$f(x)$를 아래와 같이 놓는다.

$$f(x) = x^2 - X \tag{14}$$

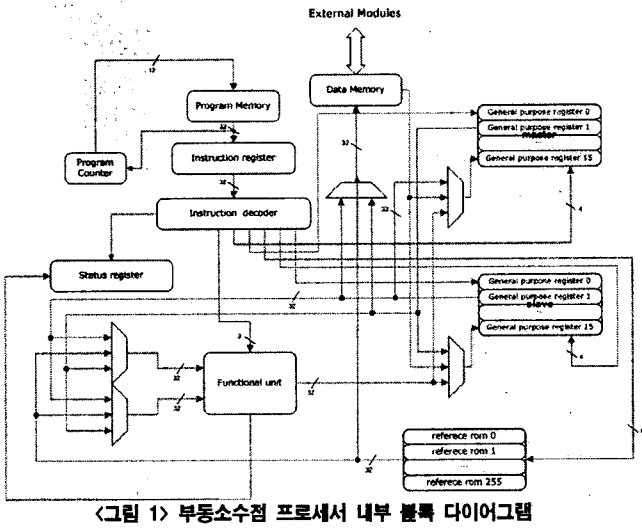
여기서 X는 구하려는 제곱근의 제곱 값, 즉 입력되는 값이다. 식 (14)를 식 (13)에 대입하여 정리하면 아래와 같이 구할 수 있다.

$$x_{n+1} = \frac{1}{2}(x_n + \frac{X}{x_n}) \tag{15}$$

초기 근사값인 x_0으로는 일반적으로 1을 사용하였다. 위의 식을 사용하여 반복 연산을 수행하여 제곱근을 근사화 시킬 수 있다.

3. 부동소수점 기반의 프로세서 설계

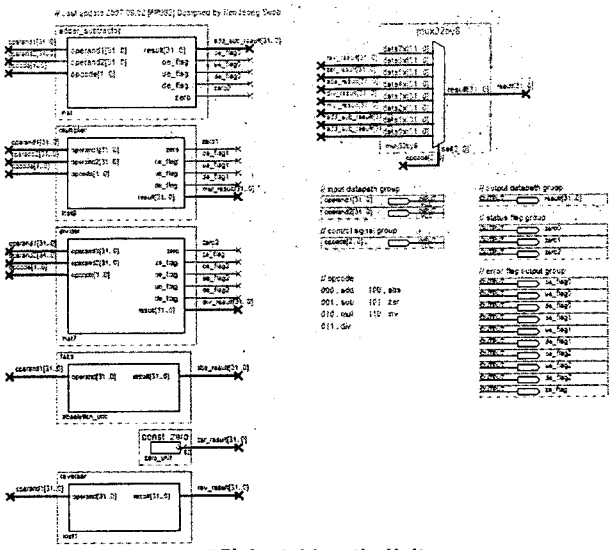
3.1 Architecture



<그림 1> 부동소수점 프로세서 내부 블록 다이어그램

그림1은 설계된 부동소수점 프로세서를 나타내고 있다. 전체적인 구성은 Program memory, Data memory, Instruction Register, Instruction Decoder, Functional Unit, Register Group, Reference ROM, Status Register, Program Counter 그리고 몇 개의 Multiplexer로 이루어져있다. 각각 모듈에 대한 설명은 아래와 같다.

1) Program Memory: instruction code를 담고 있는 이 모듈은 ROM 형태로 이루어져있으며 32bit 길이의 word가 총 4,096개로 총 12bit의 address를 가진다. 따라서 Program Memory는 16Kbytes의 용량을 가진다.
2) Data Memory: 연산에 필요한 데이터가 저장되는 RAM 타입의 모듈이다. Program memory와 마찬가지로 1 word가 32bit로 구성되어 있으며 12bit의 address를 가지며 16Kbytes의 용량을 가진다.
3) Program Counter: 다음에 수행되어야 할 명령어가 들어있는 Program Memory의 주소를 지정한다. Status Register를 참조하여 Jump 명령이나 Condition Branch 명령을 수행하기도 한다.
4) Instruction Register: instruction을 instruction decoder에 보내기 전에 1cycle을 딜레이 시키는 역할을 한다. 딜레이가 수행되는 동안 데이터 메모리와의 동기화를 위해 일부 instruction에 대해서는 통상적인 경우보다 1cycle 먼저 해석하기도 한다.
5) Instruction Decoder: 정의되어 있는 32bit instruction code를 opcode와 operand의 주소 등으로 구분하여 해석하고 명령어 수행을 위한 코어 내부의 각 모듈을 제어하기 위한 컨트롤 시그널을 내보낸다.
6) Status Register: arithmetic unit의 연산 결과에 따른 결과나 logical unit의 명령 수행 결과 또는 코어 내부의 상태를 나타내기 위한 시스템 내부 상태 레지스터이다.
7) Register Group: Register Group은 Master Group과 Slave Group으로 나뉜다. Functional unit에서 명령어 수행을 위한 operand1과 operand2의 source로 사용된다. 이들은 각각 16개의 32bit register로 구성되어 있다.
8) Reference ROM: 연산에서 자주 참조되는 상수들, 예를 들어 π나 자연 상수 e 또는 비선형 함수의 수치해석적 연산을 위해 필요한 계수 값들이 저장되어 있다.
9) Functional Unit: Single precision 기반의 수치 연산 및 논리 연산 그리고 포맷 컨버전이 이루어지는 모듈이다. 수치 연산으로는 덧셈, 뺄셈, 곱셈 그리고 나눗셈의 사칙연산을 비롯한 절대값과 같은 부가적인 연산이 수행된다. 논리 연산은 operand1과 operad2의 크기를 비교하는 명령을 수행한다.



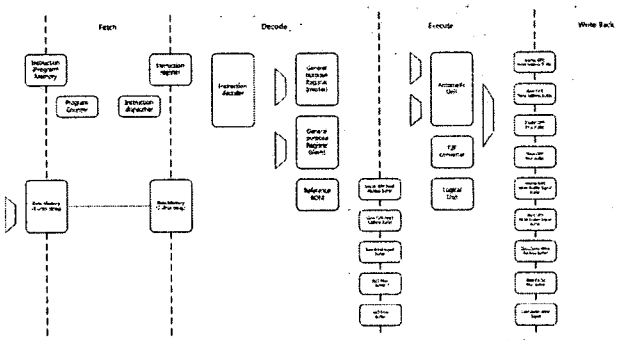
<그림 2> Arithmetic Unit

그림2는 VHDL로 구현된 arithmetic unit이다. arithmetic unit에서는 기본적으로 덧셈, 뺄셈, 곱셈, 나눗셈 연산이 수행되며 절대값 연산과 지정된 주소의 데이터를 0으로 만드는 연산 그리고 데이터의 부호를 바꾸는 연산을 수행할 수 있다.

3.2 Pipeline

파이프라인은 프로세서가 명령어 수행을 효율적으로 하기 위한 매커니즘이다. 명령어가 수행되는 동안 다음에 수행될 명령어가 해석되고 그 다음에 해석될 명령어가 레지스터에 패치되는 일련의 과정을 통해 프로세서의 동작 속도를 향상시켜준다.

현재 버전의 프로세서에는 4단계 파이프라인 구조를 가지고 있다. Fetch는 프로그램 메모리에서 명령어를 로드한다. Decode는 Instruction Register의 명령어를 해석한다. Execute는 명령어를 처리한다. Write-Back은 처리된 결과를 레지스터 또는 데이터 메모리에 저장한다.

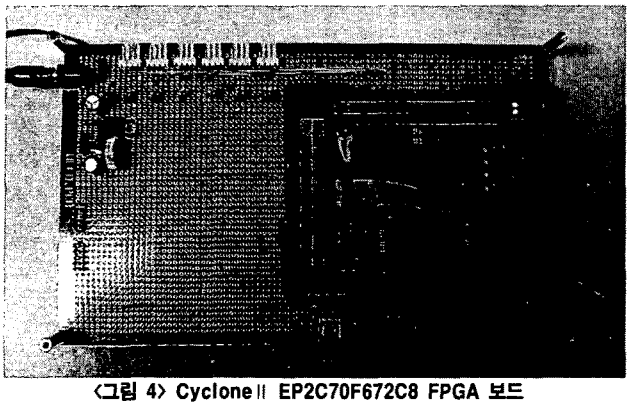


<그림 3> 4단계 파이프라인 구조를 가지는 프로세서

그림 3은 프로세서 내부 모듈들을 파이프라인 순서에 맞추어 배치한 것이다. 각 파이프라인 사이에 배치되어 있는 모듈들은 기본적으로 concurrent circuit 구조를 가지고 있으며 clock에 영향을 받지 않는다. 점선에 오버랩되어 있는 모듈들은 sequential circuit으로 clock의 움직임에 따라 동작된다.

정의된 assembly 명령어가 바이너리 형태의 기계어로 변환되고 이를 Intel-Hex 포맷 형태로 저장하여 Program memory에 저장한다. Clock cycle에 따라서 program memory에 있던 명령어 코드가 순차적으로 instruction register에 로드되고 instruction decoder에 의해 해석된다. 명령어가 해석됨에 따라 decoder에서는 명령어를 실행시키기 위해 필요한 시그널들을 내보내게 되고 동기화를 위해 시그널의 버퍼링이 이루어진다. 명령어 실행 단계에서는 버퍼링된 시그널에 따라 수치 연산 또는 논리 연산이 이루어지거나 데이터 메모리로부터의 데이터를 레지스터에 저장시키는 LD 명령 또는 레지스터의 데이터를 데이터 메모리로 저장시키는 ST 명령 등이 수행된다.

4. 실험



<그림 4> Cyclone II EP2C70F672C8 FPGA 보드

Altera사의 "Cyclone II EP2C70F672C8" 디바이스를 사용한 FPGA 보드에 하드웨어 구현을 하였다. 비선형 함수 연산에 필요한 x값은 Boot ROM에서 시스템을 초기화 시킬 때 data memory에 로드된다. 필요한 데이터의 로드가 끝나면 프로세서 코어에 run 시그널을 보내 연산을 시작하게 한다. 연산이 수행됨에 따라 연산의 결과가 시리얼 통신을 거쳐 PC로 전송되게 된다.

아래 표1은 VHDL로 구현한 부동소수점 데이터 프로세서에서 요구하는 FPGA 리소스를 나타내고 있다.

<표 1> 프로세서 코어를 구현하는데 필요한 리소스

components		Logic Cells	Memory Bits
arithmetic unit	adder/subtractor	919	0
	multiplier	104	0
	divider	1,655	0
	etc	143	0
	subtotal	2,821	0
instruction decoder		29	1,152
instruction register		31	0
general purpose register(m)		628	0
general purpose register(s)		624	0
logical unit		43	0
format converter		2,540	0
program counter		109	0
reference rom		6	7,168
program memory		0	126,976
data memory		0	131,072
etc		593	0
total		7,424	266,368

표2~6은 임의로 주어진 x입력에 대해 FPGA에서의 연산 결과를 MatLab에서의 결과와 비교한 것이다. 편의상 소수점 아래 9자리에서 버림하여 8자리까지의 데이터를 얻었다.

<표 2> sin(x) 연산의 실험 결과

x	in MatLab	in FPGA	Error
$\pi/5$	0.58778525	0.58778518	0.00000007
$2\pi/5$	0.95105651	0.95105659	-0.00000008
$3\pi/5$	0.95105651	0.95105671	-0.00000020
$4\pi/5$	0.58778525	0.58778631	-0.00000106
π	0.00000000	0.00002134	-0.00002134

<표 3> cos(x) 연산의 실험 결과

x	in MatLab	in FPGA	Error
$\pi/5$	0.80901699	0.80901706	-0.00000007
$2\pi/5$	0.30901699	0.30901706	-0.00000007
$3\pi/5$	-0.30901699	-0.30901682	-0.00000017
$4\pi/5$	-0.80901699	-0.80901253	-0.00000446
π	-1.00000000	-0.99989986	-0.00010014

<표 4> e^x 연산의 실험 결과

x	in MatLab	in FPGA	Error
0	1.00000000	1.00000000	0.00000000
0.1	1.10517091	1.10517084	0.00000007
0.2	1.22140275	1.22140264	0.00000011
0.3	1.34985880	1.34985876	0.00000004
0.4	1.49182469	1.49182462	0.00000007

<표 5> e^{-x} 연산의 실험 결과

x	in MatLab	in FPGA	Error
0	1.00000000	1.00000000	0.00000000
0.1	0.90483741	0.90483748	-0.00000007
0.2	0.81873075	0.81873083	-0.00000008
0.3	0.74081822	0.74081826	-0.00000004
0.4	0.67032004	0.67032015	-0.00000011

<표 6> $\sqrt{x}$ 연산의 실험 결과

x	in MatLab	in FPGA	Error
1	1.00000000	1.00000000	0.00000000
2	1.41421356	1.41421329	0.00000027
3	1.73205080	1.73205089	-0.00000009
4	2.00000000	2.00000000	0.00000000
5	2.23606797	2.23606777	0.00000020

위의 실험결과에 나타나듯이 FPGA에서의 연산 결과와 MatLab에서의 연산 결과에 오차가 발생하였으나 이는 무시할 수 있는 수준의 것이다. $\sin(x)$와 $\cos(x)$의 경우 x값의 크기가 증가함에 따라 오차가 점차 커지는 것을 알 수 있는데, 이는 Taylor-Maclaurin 급수를 전개함에 있어서 입력 값의 크기가 커질수록 오차가 커지는 것에 대한 영향 때문이다.

5. 결론

본 논문에서는 32bit 부동소수점 기반의 비선형 함수 연산을 위한 프로세서를 VHDL로 설계하였으며 이를 FPGA를 통해 구현하였고 실험으로 동작을 검증하였다. 수치 해석적인 방법을 통해 look-up table 없이도 비선형 함수 연산의 결과를 얻었다. 대표적인 비선형 함수인 삼각 함수, 지수 함수 그리고 제곱근 연산을 FPGA에서 수행하여 이를 MatLab에서의 연산과 비교를 통해 성공적으로 연산이 수행됨을 확인하였다.

[참 고 문 헌]

[1] Seul Jung and Sung su Kim, "Hardware Implementation of a Real-Time Neural Network Controller With a DSP and and FPGA for Nonlinear Systems", IEEE Transaction on Industrial Electronics, vol.54, No.1, pp.265-271, 2007

[2] S. Himavathi, D. Antitha, and A. Muthuramalingam, "Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization", IEEE Transactions on Neural Network, vol.18, No.3, pp.880-888, 2007

[3] Marttin J. Pearson, A. G. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing Spiking Neural Networks for Real-Time Signal-Processing and Control Applications: A Model-Validated FPGA Approach", IEEE Transactions on Neural Network, vol.18, No.5, pp.1472-1487, 2007