The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 2, pp.187-193, Apr. 30, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.2.187
JIIBC 2020-2-25

부동소수점 명령어를 지원하는 ARM 프로세서의 설계 및 모의실행

Design and Simulation of ARM Processor with Floating Point Instructions

이종복*

Jongbok Lee*

요 약 마이크로프로세서에서 부동소수점 연산은 결과의 정확도를 높이기 위하여 실수형 데이터를 대상으로 시행하는 덧셈, 뺄셈, 곱셈, 나눗셈 등의 계산을 의미한다. 일반적으로 프로세서를 설계할 때는 복잡도 때문에 부동소수점 연산은 제외하고 정수형 연산만을 지원하는 경우가 많다. 그러나, 공학 기술 연산, 디지털 신호처리 뿐 만이 아니라, 오늘날 각광을 받고 있는 인공지능 및 신경망에 대한 연산을 수행하기 위하여 필요에 따라서 부동소수점 연산이 포함되어야 한다. 본 논문에서는 VHDL을 이용하여 부동소수점 연산 명령어 기능을 갖는 32 비트 ARMv4 계열의 프로세서를 설계하고, ModelSim으로 검증하였다. 그 결과, ARM의 부동소수점 명령어에 대한 연산을 성공적으로 수행할 수 있었다.

Abstract Floating point arithmetic in microprocessor is the computation of addition, subtraction, multiplication, and division of floating point data to improve accuracy. In general, when designing a processor, floating point instructions are often excluded because of its complexity and only integer instructions are provided. However, in order to carry out the computations for not only engineering and technical operations but also artificial intelligence and neural networks that are in the spotlight today, floating point operations must be included. In this paper, we design a 32-bit ARMv4 family of processors with floating-point arithmetic instructions using VHDL and verify with ModelSim. As a result, ARM's floating point instructions are successfully executed.

Key Words : ARM, Floating Point, VHDL, ModelSim

Ⅰ. 서 론

부동 소수점 처리부 (FPU)는 부동 소수점 숫자에 대한 연산을 수행하도록 특별히 설계된 마이크로프로세서 시스템의 일부로서, 일반적인 연산은 부동소수점 데이터에 대한 더하기, 빼기, 곱하기, 나누기 및 제곱근이다. 일부 FPU는 지수 로그함수 또는 삼각법 계산과 같은 다양한 초월 함수를 수행 할 수도 있지만 정확도의 문제로 인하여 일부 시스템은 소프트웨어에서 이러한 함수를 계산한다. 과거에는 FPU가 보조프로세서로 이용되었으나, 오

*정회원, 한성대학교 기계전자공학부
접수일자 2019년 12월 23일, 수정완료 2020년 2월 23일
게재확정일자 2020년 4월 3일

Received: 23 December, 2019 / Revised: 23 February, 2020 / Accepted: 3 April, 2020
*Corresponding Author: jblee@hansung.ac.kr
Dept. of EI Engineering, Hansung University, Korea

늘날 FPU는 중앙 처리 유닛 내에 실행 유닛으로서 통합하여 설계할 수 있다. 일반적으로 프로세서를 설계할 때는 부동소수점 연산은 제외하고 정수형 연산만 지원하는 경우가 많다. 그러나, 다양한 범위에 대한 높은 정확도를 요구하는 공학 기술 연산, 디지털 신호처리 뿐 만이 아니라 최근 각광을 받고 있는 인공지능, 신경망 처리를 위하여 필요에 따라서 부동소수점 연산이 포함되어야 한다.

ARM 프로세서는 thumb 및 쉬프트 기능을 강화한 독특한 명령어 특징뿐만이 아니라, 낮은 전력을 사용하도록 설계하여 모바일 및 임베디드 시장에서 큰 강세를 보이고 있다. 오늘날, 데스크탑 컴퓨터나 서버 컴퓨터를 제외한 대부분의 스마트폰 및 임베디드 기기에는 전부 ARM 프로세서가 탑재되어 있다고 해도 과언이 아니다. ARM 계열의 프로세서 중에서 대부분이 정수형 명령어만을 취급하지만, Cortex-M4가 IEEE 754 표준의 단일정밀도 부동소수점 명령어를 지원한다[1].

국내의 프로세서 설계 분야는, 세계 최고 수준의 메모리 반도체 설계 및 공정 기술에 비하여 상대적으로 미흡한 형편이다. 따라서 프로세서 설계 분야에 대한 연구를 활발히 하고 박차를 가하여 균형을 이룰 필요가 있다. 본 논문에서는 VHDL을 이용하여 부동소수점 연산 기능을 포함하는 5 단계 파이프라인을 갖는 ARM 프로세서를 설계하였으며, 코딩 및 검증을 위하여 ModelSim을 활용하였다. 본 연구에서 채택한 ARM의 아키텍처 계열은 ARMv4로서, 후속 버전인 ARM8 및 Strong ARM에서 채택한 구조이다.

본 논문은 다음과 같이 구성된다. 2 장에서 ARM 프로세서의 부동소수점 연산을 살펴보고, 3 장에서 ARM 프로세서의 명령어 및 파이프라인 단계별 동작과 본 논문에서 설계한 ARM 프로세서의 구조를 기술한다, 4 장에서 모의실험 환경과 모의실험 결과를 보이고, 5 장에서 결론을 맺는다.

II. ARM 프로세서의 부동소수점 연산

표 1에 부동소수점에 대한 IEEE 754 단일정밀도 표준을 나타냈으며, 1 비트를 부호, 8 비트를 지수부, 나머지 23 비트를 소수부 (mantissa)로 표현한다. 가능한 많은 비트를 나타내기 위하여, 정규화된 이진수의 최상위 비트 1은 존재하지만 실제로 표시하지 않는다. 따라서 소수부는 단일정밀도에서 사실상 24 비트이다. 지수부와 소수부가 모두 0일 때는 숫자 0을 나타내며, 지수부는 8 비트이므로 0부터 255의 값을 가진다. 지수부가 255이고 소수부가 0이면 무한대를 나타내며, 지수부가 255이고 소수부가 0이 아니면 NaN(숫자가 아님)에 해당한다.

표 1 부동소수점의 IEEE 754 단일정밀도 표현
Table 1. single precision expression of IEEE754 floating point number

31	30•••23	22••••••••••0
부호	지수부	소수부

IEEE 754는 부동소수점에 대한 소팅을 용이하도록 부호비트를 제일 앞에 놓아서 양수, 0, 음수를 테스트할 수 있게 하였다. 또한 지수부를 소수부 앞에 먼저 둠으로써 지수부의 부호가 같을 때는 지수부가 큰 수가 당연히 지수부가 작은 수보다 크므로, 단지 정수 비교에 의하여 소팅이 가능하도록 하였다.

만일에 음의 지수를 2의 보수로 표현한다면 매우 큰 수처럼 보일 것이므로 가장 작은 음의 지수를 0으로, 가장 큰 양의 지수를 255로 표현하기 위하여 지수부에 127의 바이어스를 더한다[2,3].

1. 부동소수점의 덧셈 및 뺄셈

두 개의 부동소수점의 덧셈을 시행하기 위한 절차는 다음과 같다. 먼저 덧셈 결과의 부호를 결정하는 방법에 대하여 살펴본다. 만일 두 피연산자의 부호가 같으면 덧셈 결과의 부호도 동일하다. 그러나, 만일 두 피연산자의 부호가 다르면 결과의 부호는 더 큰 지수를 갖는 피연산자의 부호와 같게된다. 만일 두 피연산자의 부호가 다르지만 지수의 크기가 같다면, 덧셈 결과의 부호는 더 큰 소수부를 갖는 피연산자의 부호와 같아야 한다.

두 수의 지수부가 다르면 덧셈을 할 수가 없으므로, 지수부를 비교하여 작은 숫자의 지수부를 큰 숫자의 지수부와 같도록 맞춘다. 이것을 위하여 작은 숫자의 소수점을 왼쪽으로 이동해야하는데, 이것은 결국 작은 숫자의 소수부를 지수부 값의 차이만큼 오른쪽으로 이동하는 것과 같다. 그 다음에 두 수의 소수부끼리 실제로 더한 후에, 만일 그 결과가 정규화 되지 않았으면 이것을 정규화 시켜야한다. 정규화 과정은 소수부를 오른쪽이나 왼쪽으로 이동시키고 지수부를 그에 맞게 각각 감소시키거나 증가시킴으로써 이루어진다.

정규화를 거친 후에 만일 오버플로우나 언더플로우가 발생한다면 예외처리를 하고 그렇지 않다면 비트 수에

맞춰서 자리올림을 시행한다. 만일에 자리올림을 시행 한 후에 정규화되지 않았으면 위 과정을 다시 반복한다. 한편, 부동소수점의 피연산자1에서 피연산자2의 뺄셈은, 피연산자2의 부호를 반대로 뒤집어서 덧셈을 수행함으로써 얻을 수 있다.

2. 부동소수점의 곱셈

부동소수점의 곱셈은 먼저 각 피연산자의 지수부를 더하여 새로운 지수부의 값을 얻는 것으로 시작한다. 그런데, 각 피연산자에 이미 127이라는 바이어스가 존재하므로, 두 개의 지수부를 더하는 경우 254의 바이어스가 실제 지수값에 추가로 생성된다. 따라서 지수부를 합한 결과에서 127을 빼줘야 올바른 지수부 값이 된다. 다음 단계는 소수부끼리 곱셈을 시행한다. 곱셈의 결과가 정규화되지 않았다면 정규화 처리를 한다. 소수점을 왼쪽으로 이동해야하므로 소수부를 오른쪽으로 이동시키고, 지수부를 그에 맞춰 증가시킨다. 그 결과 오버플로우나 언더플로우가 발생했으면 예외처리를 하고, 그렇지 않으면 비트 수에 맞춰서 자리올림을 한다. 만일에 자리올림을 시행 한 후에 정규화 되지 않았으면 위 과정을 반복한다.

3. 부동소수점의 나눗셈

부동소수점의 나눗셈은 뺄셈을 반복하는 것으로 생각할 수 있으며, 이것을 그대로 적용한 반복법을 이용한다. 피제수 A를 제수 B로 나눈 몫이 Q이고 나머지가 P일 때, P를 0으로 초기화하고 피제수 A의 최상위비트를 P의 최하위비트로 쉬프트시킨다. P에서 제수 B를 뺀 결과가 음수이면 몫 Q를 0으로 놓고, 그 결과가 양수이면 몫 Q를 1로 놓고 뺄셈의 결과를 P의 값으로 업데이트한다. 다시 피제수 A의 최상위비트를 P의 최하위비트로 쉬프트시키면서 위 과정을 반복한다. 그림 1에 나눗셈에 대한 VHDL 코드의 일부를 표시했다.

III. ARM 프로세서의 구조

표 2에 본 논문에서 설계한 ARM 명령어 집합을 나타냈다. ARM 명령어는 크게 산술 논리, 곱셈, 메모리, 분기, 부동소수점의 5 가지 유형으로 구성된다. 산술 논리 명령어는 AND, EOR, SUB 등의 21 개 명령어로 구성된다. 곱셈 명령어는 6 개의 명령어로 이루어지며, 결과가 32 비트인 것과 64 비트인 것으로 나뉜다. 메모리 명령어는 STR, LDR 등의 스토어 및 로드 명령어로 구성되며 워드 단위 이외에 바이트 단위 또는 하프워드 단위로 수행한다. 분기 명령어는 B와 BL로 구성되며, 마지막으로 32 비트 부동소수점 연산을 위하여 추가로 필요한 명령어는 VMOV, VADD, VSUB, VMUL 및 VDIV 5 개다.

표 2. 설계된 ARM 프로세서의 명령어 집합
Table 2. The instruction set of the Designed ARM processor

명령어 유형	명령어
산술 논리	AND EOR SUB RSB ADD ADC SBC RSC TST TEQ CMP CMN ORR MOV LSL LSR ASR RRX ROR BIC MVN
곱셈	MUL MLA UMULL UMLAL SMULL SMLAL
메모리	STR LDR STRB LDRB STRH LDRH LDRSB LDRSH
분기	B BL
부동소수점	VMOV VADD VSUB VMUL VDIV

```
a := "01" & x_mantissa;
b := "01" & y_mantissa;

for i in 25 downto 0 loop
 p := a - b;
 if ( p(24)='0' ) then
    q(i):='1';
    a := p;
 else
    q(i):='0';
end if;
a := a(23 downto 0) & '0'; -- shift left
end loop;
```

그림 1. 나눗셈에 대한 VHDL 코드의 일부
Fig. 1. The partial divider code of VHDL

그림 2는 ARM 프로세서의 5 단계 파이프라인이다. 이것은 인출(Fetch), 해독(Decode), 메모리 접근(Memory), 실행(Execute), 되쓰기(Write Back)의 5 단계로 나뉜다. 정수형 연산은 실행단계가 한 싸이클만에 완료되지만, 부동소수점 연산은 그 종류에 따라 다중 싸이클이 실행된다.

명령어 파이프라인에서 명령어 간에 쓰기 후 읽기(RAW) 종속이 발생하는 경우를 해저드라고 하며, 해저드 유닛에서 처리한다. 인접한 명령어 간에 발생하는 해저드는 메모리단계의 결과를 포워딩시키고, 한 개의 명령어만큼 떨어진 명령어 간에 발생하는 해저드는 되쓰기단계의 결과를 포워딩 시켜서 해결할 수 있다. 마지막으로, 두 개의 명령어만큼 떨어진 명령어 간에 발생하는 해저드는 레지스터 화일의 내부 포워딩에 의하여 해소된다[2,3].

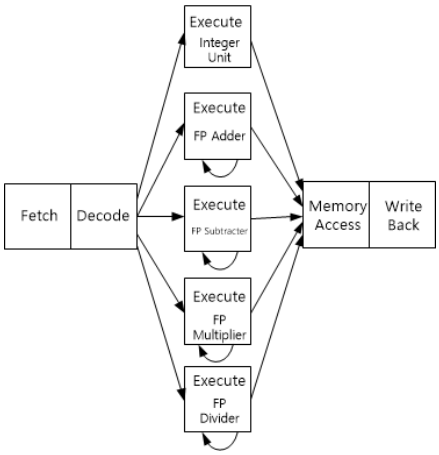

그림 2. 다중싸이클 실행의 ARM 프로세서의 5 단계 파이프라인
Fig. 2. The 5 stage pipeline of ARM processor with multicycle operations

그림 3에 본 논문에서 설계한 부동소수점 연산 기능을 지원하는 ARM 프로세서의 전체블럭도를 보였다. 인출단계, 해독단계, 실행단계, 메모리단계, 되쓰기단계의 각 사이마다 파이프라인 레지스터가 삽입되어 명령어 파이프라인 체계를 지원한다. 복잡성을 피하기 위하여, 제어신호는 선을 직접 연결하지 않고 제어신호 이름으로 연결 상태를 알 수 있게 하였다.

인출단계는 명령어 메모리와 2 개의 멀티플렉서, 덧셈기로 구성된다. 명령어 메모리에는 ARM 프로세서가 실행할 명령어들이 기억되어있으며, 프로그램 카운터 (PC) 레지스터에 수록된 명령어 주소를 공급받아 명령어를 출력하는 기능을 수행한다. 인터럽트가 걸렸을 때 벡터 어드레스가 강제로 입력될 수 있도록 멀티플렉서를 설치하였다.

해독단계에서 레지스터 화일은 동시에 3 개의 피연산자를 읽고, 2 개의 결과값을 쓸 수 있도록 설계되었다. 레지스터 화일의 제 1 피연산자 입력은 R_n, R_{15}, R_{14}, R_m 중에서 선택한다. R_n은 일반 데이터처리 명령어에서 제 1 소오스 레지스터이다. R_{15}는 프로그램 카운터로 이용되는 레지스터이고, R_{14}는 인터럽트가 걸렸을 때 복귀주소 또는 BL 분기 명령어에서 복귀주소를 저장한다. R_m은 일반 데이터처리 명령어에서 제 2 소오스 레지스터이나, 곱셈 명령어에서는 제 1 레지스터에 해당되므로 위 단자로 공급된다. 레지스터 화일의 제 2 피연산자 입력은 R_m과 R_d를 멀티플렉서로 선택한다. R_m은 일반 데이터처리 명령어의 제 2 레지스터이다. 정수형 데이터를 저장하는 레지스터화일에 부동소수점 데이터를 저장하는 부동소수점 레지스터화일이 추가되었다.

실행단계는 ALU, FPU, 정수형곱셈기, 쉬프터, 멀티플

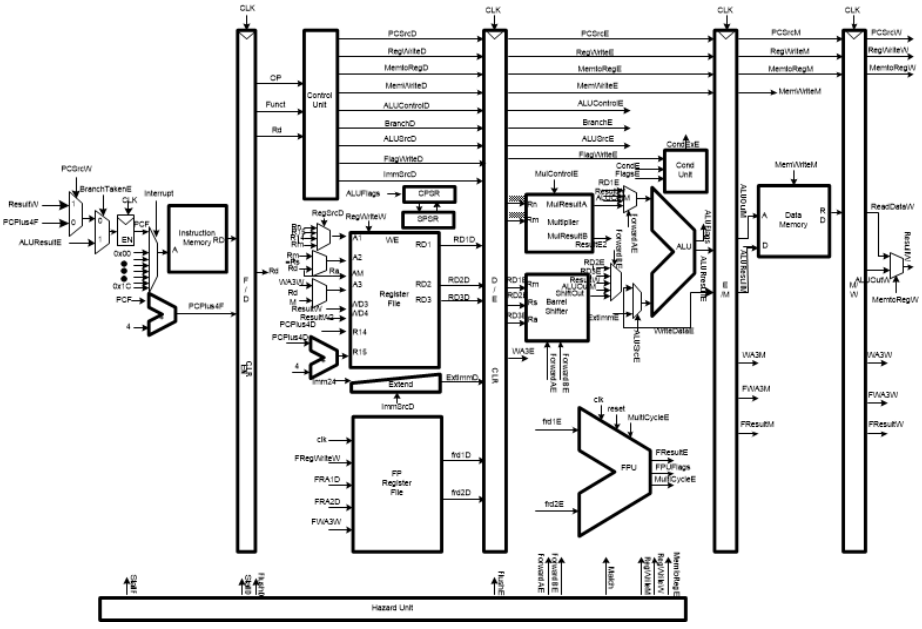

그림 3. 부동소수점 장치를 갖는 ARM 프로세서의 전체 블럭도
Fig. 3. The block diagram of ARM processor with floating point unit

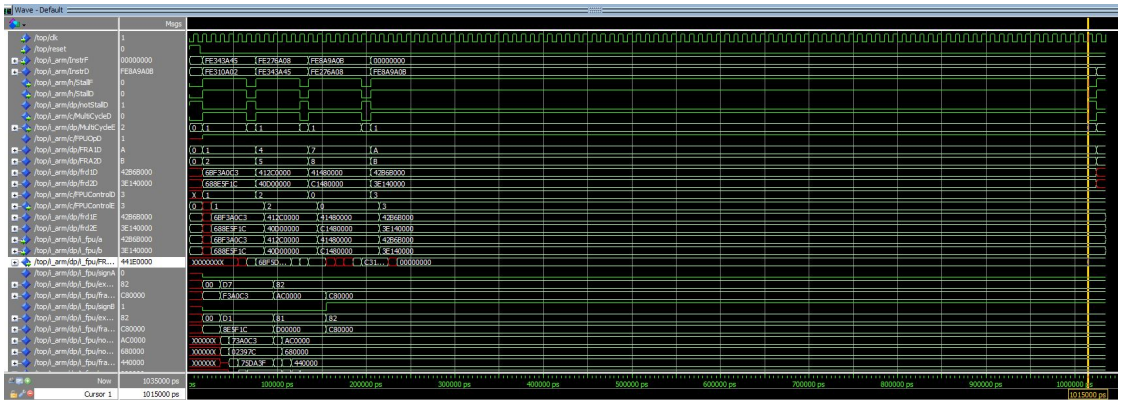

그림 4. ARM 프로세서의 부동소수점 ModelSim 실행결과
Fig. 4. ModelSim simulation waves of ARM processor with floating point operations

렉서, 조건플래그 단위 및 해독단계에서 파이프라인 래치를 통하여 넘어온 각종 제어신호 등으로 구성된다. ALU는 논리 및 정수형 연산을 수행하고, FPU는 부동소수점 연산을 수행한다. ALU의 상단에 곱셈기와 쉬프터를 설치해서 산술논리연산 외에 곱셈과 쉬프팅 동작을 지원한다. ALU의 제 1 입력은 제 1 소오스 레지스터, 메모리 단계에서 포워딩되는 값, 되쓰기 단계에서 포워딩되는 값, 곱셈기에서 출력되는 값 중에서 멀티플렉서로 선택한다. ALU의 제 2 입력은 제 2 소오스 레지스터, 메모리단계에서 포워딩되는 값, 되쓰기 단계에서 포워딩되는 값, 부호확장기로부터 공급되는 값, 그리고 쉬프터의 출력 중에서 선택한다.

메모리단계는 데이터메모리와 실행단계에서 파이프라인 래치를 통하여 전달된 각종 제어신호들로 구성된다. 명령어 메모리의 주소 입력은 실행단계의 ALU 출력과 연결되며, 데이터 입력은 레지스터화일의 제 2 출력으로부터 공급된 데이터와 연결된다. 명령어 메모리의 출력은 되쓰기 단계의 파이프라인 레지스터로 전달된다.

되쓰기단계는 명령어 파이프라인 제 2단계인 해독단계에서 데이터를 읽어들이는데 이용하는 레지스터 화일에 반대로 데이터를 기록하는 과정이다. 파이프라인에서 레지스터 화일은 유일하게 읽기와 쓰기를 위하여 두 번 접근되는 장치이다. 로드인 경우에 명령어메모리로부터 읽은 값과, ALU 연산인 경우에는 그 결과를 레지스터 화일에 역시 써야 한다[4-11].

표 3. 모의실험에 이용된 부동소수점 연산
Table 3. The data used for floating point operation simulation

피연산자 1	피연산자 2	연산	결과
6BF3A0C3	688E5F1C	덧셈	6BF5DA3F
412C0000	40D00000	뺄셈	40880000
41480000	C1480000	곱셈	C31C4000
42B6B000	3E140000	나눗셈	441E0000

IV. 모의실험

본 논문의 모의실험은 운영체제 Windows 10에서 3.1GHz로 동작하는 Intel Core i5-2400에서 시행하였다. ModelSim은 Intel 10.6d 버전을, VHDL 컴파일러는 1076-2002 버전을 이용하였으며, 2,390라인의 총 34 개의 VHDL 프로그램이 부동소수점 연산을 지원하는 ARM 프로세서를 설계하는데 이용되었다. 표 3은 부동소수점 연산의 모의실험에 사용된 피연산자 값들과 그 사칙연산 결과이다. 본 연산을 위하여, VMOV 명령어를 이용해서 필요한 부동소수점 데이터가 부동소수점 레지스터에 적재되었다.

먼저 부동소수점 덧셈의 피연산자 1은 IEEE754 단일 정밀도로 나타냈을 때 32비트의 16 진수 6BF3A0C3인데, 이것은 10 진수로 5.890563×10^{26}이다. 피연산자 2는 16 진수688E5F1C로서 십진수 값은 5.378644×10^{24}이다. 모의실행 후 덧셈은 7 싸이클이 소요되었으며, 연산결과는 6BF5DA3F인데 이것은 10 진수로 더한 결과인 $5.94434944 \times 10^{26}$과 그 값이 일치한다.

부동소수점 뺄셈의 경우 피연산자 1과 피연산자 2는 각각 412C0000와 40D00000로서 10 진수로는 각각

10.75와 6.50에 해당한다. 두 수의 뺄셈 역시 7 싸이클이 소요되었으며 연산결과는 $10.75-6.5=4.25$인데, 이것을 IEEE 754 형식으로 나타내면 40880000이므로 올바른 결과이다.

부동소수점 곱셈의 경우 피연산자 1은 41480000로서 12.5에 해당하고, 피연산자 2는 C1480000로서 -12.5이다. 곱셈 역시 7 싸이클이 소요되었고, 연산의 결과 $12.5\times(-12.5)=-156.25$이며, 이 값은 C31C4000와 일치한다.

마지막으로 부동소수점의 나눗셈은 피연산자 1과 피연산자 2가 각각 42B6B000와 3E140000인데, 10 진수로 각각 91.34375와 0.14453125에 해당한다. 이 때 $91.34375\div0.14453125=632.0$이며, 81 싸이클만에 모의실행으로 얻은 결과인 441E0000과 일치한다.

그림 4에 ARM 프로세서가 부동소수점 명령어를 실행하는 ModelSim의 결과 파형을 나타냈다. 그 결과, ARM 프로세서가 위에서 설명한 바와 같이 부동소수점 연산을 올바르게 실행하는 것을 확인하였다.

V. 결 론

본 논문에서는 ModelSim 환경에서 VHDL을 이용하여 37 개의 정수형 명령어와 5 개의 부동소수점 명령어를 지원하는 ARM 프로세서를 설계 및 개발하였다. 주어진 ARM 어셈블리 프로그램을 모의실행 시킨 결과, 부동소수점에 대한 사칙연산이 모두 올바르게 처리되는 것을 확인할 수 있었다.

추후로, 실제의 ARM 프로세서에서 제공하고 있지만 본 설계에서 누락된 제곱근 명령어와 지수 로그함수 및 삼각함수 관련 명령어를 포함시키는 것이 목표이다. 설계된 ARM 프로세서를 Xilinx사의 Vivado 또는 Altera사의 Quartus II를 이용하여 합성한 후에, 정적 시간 분석(STA)과 합성 후 모의실험(Post synthesis simulation)을 거쳐 최종적인 동작을 검증하고 FPGA로 프로그래밍하여 동작을 검증할 예정이다. FPGA로 검증을 완료한 후에는, Synopsis의 DC(Design Compiler)로 합성하여 국내 기관인 IDEC을 통하여 ASIC 칩으로 구현할 예정이다.

References

[1] ARM Architecure Reference Manual, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.architecture.reference/index.html

[2] J. L. Hennessy, and D. A. Patterson, “Computer Architecture A Quantitative Approach”, 6th Edition; 2018.

[3] S. L. Harris, and D. M. Harris, "Digital Design and Computer Architecture ARM Edition", Elsevier Korea LLC, 2016.

[4] F.J. Jurado Carmona, J. Tombs, M.A. Aguirre Echanove and A. Torralba, "Implementation of a fully pipelined ARM compatible microprocessor core," XVII Design on Circuits and Integrated Systems Conference, pp. 559-563, 2002.

[5] J. S. Pastor, I. Gonzalez, J. Lopez, F.G. Arribas, J. Martinez, "A Remote Laboratory for Debugging FPGA-Based Microprocessor Prototypes," Proceedings of the IEEE International Conference on Advanced Learning Technologies, 2004.

[6] A.A. Morgan, M.E. Allam, M.A. Salama, and H.A.K Mansourm "Implementation of an ARM Compatible Processor Core for SOC Designs," 2005 International Conference on Information and Communication Technology, Dec. 2005.

[7] M. M. Kinage and D.G. Khairnar, "Design and Implementation of FPGA Soft Core Processor for Low Power Multicore Embedded System using VHDL," Sep. 2016.

[8] J. Davidson, "FPGA Implementation of a Reconfigurable Microprocessor," IEEE Custom Integrated Circuits Conference, pp.3.2.1-3.2.4, 1993.

[9] E. Alaer, A. Tangel, M. Yakut, "MIB-16 FPGA based Design and Implementation of a 16 Bit Microprocessor for Educational Use," 6th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing, pp.284-288, 2007.

[10] H.S. Herman, C. Srihari, M. Matthew, "Pipeline Reconfigurable FPGAs," Journal of VLSI Signal Processing Systems,", 2000, Vol. 24, pp. 129-146.

[11] J. Lee, "Design and Simulation of ARM Processor with Intrerrupts," Journal of the Institute of Internet, Broadcasting and Communication, Vol. 19, No. 6, pp. 183-189, Dec. 31 2019.

저 자 소 개

이 종 복(정회원)

- 1964년 8월 20일생.
- 1988년 서울대 컴퓨터공학과 졸업.
- 1998년 동 대학 전기공학부 졸업 (공박).
- 1998~2000 LG반도체선임연구원.
- 2000년~현재 한성대 기계전자공학부 교수

- Tel : 02-760-4497
- Fax : 02-760-4435
- E-mail : jblee@hansung.ac.kr
- 관심분야 : 마이크로 프로세서,멀티코어 프로세서, 텐서 프로세서 유닛.

※ 본 연구는 한성대학교 교내학술연구비 지원과제임.