한국 퍼지 및 지능시스템학회 논문지 1999, Vol. 9, No. 6 (1999)634~643

재구성 가능한 FPGA 시스템 상에 퍼지 제어기의 구현

An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA System

김대진 · 조인현*

Daijin Kim and In-Hyun Cho*

동아대학교 컴퓨터공학과, *퓨처시스템 정보통신연구소

요 약

본 논문은 재구성 가능한 FPGA 시스템 상에 퍼지 제어기의 구현 방안을 다룬다. 제안한 구현 방안은 퍼지 제어기를 시간적으로 독립적인 여러 모듈로 분할하여 이들을 미리 독립적으로 구현하여 둔 다음, 각 시점에서 원하는 모듈을 불러 사용하는 실행 시점(run-time) 재구성 방법으로, 하나의 FPGA가 갖는 셀 집접도의 제약으로 인해 하나의 FPGA칩 상에 전체 퍼지 제어기를 구현하기가 불가능한 경우에 효과적으로 이용될 수 있다. 이를 위해 퍼지 제어기의 각 모듈은 VHDL 언어로 기술되어 FPGA 컴파일러에 의해 합성된 후, Xilinx사의 Xact 장비에 의해 최적화 및 배치, 배선이 수행되어, FPGA상에 다운로드 가능한 하드웨어 객체 (hardware object) 상태로 만들어진다. 이후 퍼지 제어기를 구현하기 위해서는 각 시점에 원하는 모듈의 하드웨어 객체를 불러 FPGA를 재구성한다. 트럭 후진 주차 제어용 퍼지 제어기를 제안한 실행 시간 재구성법에 의해 직접 구현하여 제어 동작 상태를 테스트해 봄으로서 제안한 퍼지 제어기 구현 방법의 타당성을 확인하였다.

ABSTRACT

This paper concerns an implementation of fuzzy logic controller (FLC) on a reconfigurable FPGA system. In this proposed implementation method, the FLC is partitioned into many temporally-independent modules and each module is implemented independently in advance. Then, a run-time reconfiguration method is used to allocate a necessary module at run-time. This implementation method is effective when the FLC can not be included in the FPGA due to the limitation of constituent cells. The implementation of each module consists of several processes such as VHDL description, FPGA synthesis, optimization, placement and routing. After these processes, each module becomes a downloadable hardware object that will be called to configure the FPGA for the FLC. The proposed implementation method is tested by implementing the FLC for the truck backer-upper control on the VCC's EVC-1 reconfigurable FPGA board directly.

1. 서 론

퍼지 제어기는 가전 및 산업 분야의 공정 제어에 폭넓게 응용되고 있다. 특히 시스템의 특성이 복잡하여 기존의 정량적인 방법으로는 해석할 수 없거나, 얻어지는 정보가 정성적이며, 부정확하고 불확실한 경우에 있어서 기존의 제어기보다 우수한 제어결과를 나타낸다[1].

퍼지 제어기는 그것이 갖는 비교적 간단한 구조와 동작 알고리즘으로 인해 종종 하드웨어로 직접 구현되어진다. 퍼지 제어기를 하드웨어로 구현하는 방법은 크게 아날로그 및 디지털 방법으로 대별된다. 최초의 퍼지 제어기의 하드웨어 구현은 1998년 AT & T사의 Bell 연구소에 의해 개발된 디지털 방식에 의한 구현[2]이었다. 퍼지 제어기의 최초 아날로그 구현은 일본의 Yamagawa등[3]에 의해 개발된 것으로 표준 CMOS 공정에 기반한 비선형 아날로그 전류 모드를 사용한 것이다. 이들 이후로 많은 다른 구현 방안이 보고되고 있다.

응용 목적상 퍼지 제어기의 구현 방법은 크게 퍼지 연산을 갖는 범용 퍼지 프로세서와 특정 응용을 위한 특정(dedicated) 퍼지 하드웨어 구현 등으로 대별된다. 전자는 구현이 쉽고 다양한 응용에 적용될 수 있지만, 퍼지 연산 성능이 떨어진다. 후자는 개발 시간이 오래 걸리지만 구현된 다음 높은 연산 성능을 가져다준다. 최근 하나의 알고리즘을 구현하는 FPGA(Field Programmable Gate Array)가 갖는 재구성력을 사용한 재구성 가능한 연산 시스템은 이것이 특정 ASIC 하드웨어와 범용 퍼지 프로세서간의 적절한 타협으로 작용한다는 점에서 많은 주목을 끌고 있

*본 논문은 1999년도 포항공대 자체 연구비 지원에 의해 수행되었음.

다[4]. 특정 하드웨어와는 달리 FPGA는 최종 사용자에 의해 쉽게 재구성되고, 많은 다른 응용에 반복 사용될 수 있으므로 유연하며, 적당한 EDA 장비에 의해 원하는 시스템을 합성함으로서 신속한 프로토타이핑을 제공해준다. 또한 미리 정의된 하드웨어 자원 상에 일련의 명령어에 의해 원하는 알고리즘을 구현하는 범용 프로세서와 달리, FPGA상에 원하는 알고리즘을 구현하는 경우 요구되는 특정 모듈만을 구현하므로 경제적이고 보다 높은 성능을 나타낼 수 있는 장점이 있다.

FPGA 기반 시스템은 개발 시간을 크게 줄여주는 장점이 있다. 하드웨어를 설계는 하드웨어를 여러 수준으로 모델링하는 과정이 수반되는데 흔히 VHDL (VHSIC Hardware Description Language)[5]과 같은 하드웨어 기술 언어에 의해 기술되고, 이를 시뮬레이션을 통해 검증한다. VHDL 설계의 복잡도가 증가하면 할수록, 설계의 검증 과정은 복잡하고 시간이 많이 걸리게 된다. 이러한 검증 과정의 시간 증가 문제는 FPGA가 갖는 빠른 turnaround 시간 특성을 이용하여 변경된 회로에 대한 새로운 configuration 파일을 만들어 FPGA 칩에 다운로딩하고, 재구성된 칩을 최대 속도로 동작시켜 봄으로서 해결할 수 있다. 본 연구에서는 위와 같은 여러 장점 때문에 퍼지 제어기를 구현하는 수단으로 재구성 가능한 FPGA 시스템을 사용하고자 한다.

원하는 퍼지 제어기를 구현하는데 FPGA 기반 시스템을 사용할 때, FPGA가 갖는 재사용성 때문에 많은 가능한 설계를 시도해 볼 수 있는 장점이 있다. 흔히 FPGA 기반 시스템 상에 원하는 하드웨어를 개발할 때 시스템의 모델링, 합성, 검증 및 구현은 과정에 여러 가지 EDA 장비를 사용하는데, 이들을 사용하는 경우 장점은 모델링을 하기 위해 사용된 하드웨어 기술 코드를 합성, 검증 및 구현하는데 거의 그대로 사용할 수 있다는 점이다. 또한 퍼지 제어기의 일반적 구조가 입출력 변수의 수, 퍼지항의 개수, 소속함수, 비트 해상도 및 응용에 따라 달라지는 제어 규칙을 제외하고는 거의 불변이므로, 설계 사양으로부터 원하는 퍼지 제어기의 VHDL 코드를 자동적으로 생성 가능하다. 나아가, 자동 생성된 VHDL 코드로부터 원하는 퍼지 제어기를 구현할 때 여러 가지 EDA 장비 (Synopsys사의 VHDL 시뮬레이터, Synopsys사의 FPGA 컴파일러, Xilinx사의 XactStep 설계 매니저, 및 VCC사의 하드웨어 객체 기술은)의 지원을 받아 손쉽게 구현할 수 있는 이점이 있다.

본 연구에서 사용한 FPGA 칩의 가용 cell의 수는 약 13,000 게이트 급인 Xilinx사의 XC4013PQ286[6]으로 이를 이용하여 퍼지 제어기의 전체 기능 블록을 한 chip상에 구현하는 것은 불가능하다. 따라서, 전체 퍼지 제어기를 시간적으로 독립적인 기능 블록으로 나눈 뒤 이들을 한번에 하나씩 실행 시간에 재구성하는 방법을 사용하여 FPGA 칩이 갖는 가용 cell 수의 제약 문제를 해결하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 구현하고자 하는 퍼지 제어기의 구조 및 사양에 대해 설명한다. 3장에서는 실행 시점 재구성법의 원리를 설명하고, 4장에서는 전단부, 후단부 처리 과정 및 재구성 가능한 FPGA 시스템상에 퍼지 제어기 구현을 설명한다. 5장에서는 EVC1 보드 상에 구현된 트럭 후진 주차 제어용 FLC의 동작 성능을 보여준다. 마지막으로, 결론이 뒤따른다.

2. 제안한 퍼지 제어기의 사양 및 구조적 특성

퍼지 제어기를 하드웨어적으로 구현하는 경우, 하드웨어의 복잡도를 줄이고 제어 성능을 높이기 위해 다음과 같은 제약을 가한다.

1. 퍼지 제어기의 입력은 비퍼지화(crisp) 값을 갖고 유한개의 값으로 양자화되어 있다.
2. 각 입출력 변수의 소속 함수 중첩도는 최대 2이다.
3. 각 출력 변수의 소속 함수 모양은 대칭형의 삼각형이다.
4. 정확한 비퍼지화값을 얻기 위해 각 소속 함수의 소속값과 폭을 동시에 고려한다.

그림 1은 제안한 퍼지 제어기의 하드웨어 구조를 나타낸 것으로, MIN 모듈, 추론 모듈, MAX 모듈, 비퍼지화 모듈, 및 제어 모듈로 구성되어 있다. 구현하고자 하는 퍼지 제어기의 설계 사양은 다음과 같다.

- 입력 변수 개수 = 2 : (x, ϕ)
- 출력 변수 개수 = 1 : (θ)
- 입력 변수의 소속함수 개수 = (5, 7)
- 출력 변수의 소속함수 개수 = (7)
- 입·출력 변수의 크기 해상도 = 8 비트 = 256 레벨
- 소속 함수의 크기 해상도 = 8 비트 = 256 레벨
- 소속 함수의 범위 = [0-255]
- 소속 함수간 최대 중첩도 = 2
- COG defuzzification by coarse-to-fine moment searching

본 연구에서 퍼지 제어기를 하드웨어로 구현할 때 중요 고려 사항은 FPGA가 갖는 연산 자원의 한계 때문에 연산 속도는 약간 희생하더라도 하드웨어 복

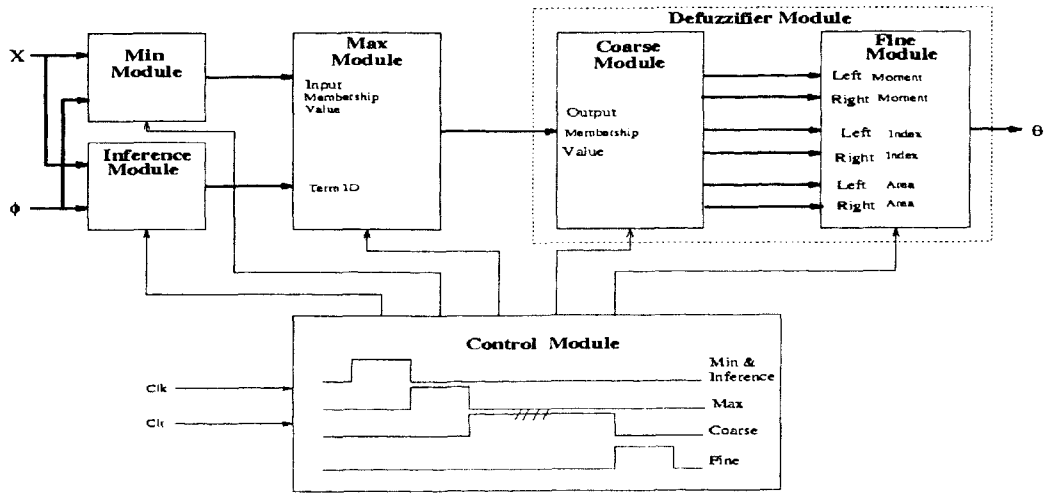

그림 1. 제안한 퍼지 제어기의 하드웨어 구조
Fig. 1. An hardware architecture of the proposed FLC

잡도를 줄이려고 다음과 같은 점을 고려하였다(이에 관한 자세한 내용은 본인이 발표한 다른 논문[7,8]에 자세히 설명되어 있음). 먼저, MIN과 MAX 모듈 구현시 MIN 연산과 MAX 연산을 비교기 트리로 구현한다면 MIN 연산의 경우, 입력 변수가 n개인 경우 $2^n \cdot (n\text{-}1)$개의 비교기가 요구되며, MAX 연산의 경우, 2^n개의 규칙 베이스가 있을 경우, $C(2^n, 2)$개의 등호 비교기와 $C(2^n, 2)$개의 크기 비교기가 요구된다. 따라서, 입력 변수의 개수 n이 커지면 하나의 FPGA칩 상에 MIN모듈 또는 MAX 모듈을 구현하는 것이 불가능하게 된다. 본 연구에서는 이를 해결하기 위해 MIN 연산의 피 연산자를 복수개의 레지스터 파일에 넣어두고 *read-modify-write* 연산[8]을 여러 번 수행하는 방안을 제안하였다. 여기서 modify는 MIN 모듈에서는 MIN 연산을 의미하는 데, 입력 변수수가 n인 경우 2^n개의 레지스터가 요구되어서 비교기에 의해 구현하는 경우보다 하드웨어 복잡도가 $O(n)$만큼 감소됨을 의미한다.

다음, 퍼지 제어기 구현시 가장 연산 시간이 많이 걸리고 구현하기 복잡한 구성 모듈은 승산과 제산을 수행해야하는 비퍼지화기이다. 따라서, 비퍼지화기를 어떻게 효과적으로 설계하느냐가 전체 퍼지 제어기의 연산 성능과 비용 면을 크게 좌우한다. 본 연구에서는 효과적인 비퍼지화기의 구현을 위해 특히 다음과 같은 사항을 고려하였다. 먼저 본 연구에서 사용한 COG 비퍼지화기의 비퍼지화값 연산시 요구되는 곱셈기를 확률론적 연산[9]으로 대치하여 곱셈기를 사용하지 않도록 하였다. 확률론적 연산에 따르면 한 수치값이 확률적으로 비트열내의 "1"의 개수와 비례하도록 표현되며, 이 경우 약간의 연산 오차가 발생되지만 두 수의 곱이 두 비트열간의 AND 연산에 의해 가능하다. 따라서 곱셈기를 하나의 AND 게이트에 의해 구현함으로서 구현 비용을 크게 줄일 수 있다.

다음, COG 비퍼지화기의 비퍼지화값 연산시 구현 비용이 높고 연산시간이 많이 걸리는 나눗셈을 수행하는데 이를 출력 변수상의 좌·우측 모멘트의 균형점을 찾는 것으로 대신하였다. 그러나 Ruitz 등[10]이 제안한 모멘트 균형점 탐색은 매 탐색마다 출력 변수상에 매우 작은 크기의 동일 간격만큼 좌측 또는 우측으로 탐색점을 이동시킴으로서 탐색 시간이 많이 걸리는 단점이 있다. 이러한 단점을 극복하기 위해 coarse-to-fine 탐색 방법[7]을 제안하였는데 이 방법에서는 coarse 탐색시 출력 변수상 탐색점의 이동을 퍼지항 단위로 수행하며, 인접하는 두 퍼지항에 도달한 다음 Ruitz 등이 사용한 Fine 탐색을 수행함으로서 탐색 시간을 크게 줄일 수 있다.

3. 재구성 가능한 FPGA 시스템 상에 구현

재구성 가능한(reconfigurable) 컴퓨팅 시스템은 FPGA가 가지는 재구성 능력을 이용하여 구현하고자 하는 응용 알고리즘내 연산시간이 많이 걸리는 부분을 hardwiring에 의해 구현함으로서 알고리즘 수행시간을 크게 줄여줄 수 있는 능력을 나타낸다[11]. 이 시스템의 가장 중요한 구성 요소는 SRAM 기반 FPGA로서 기능이 미지정된(uncommitted) 많은 프로

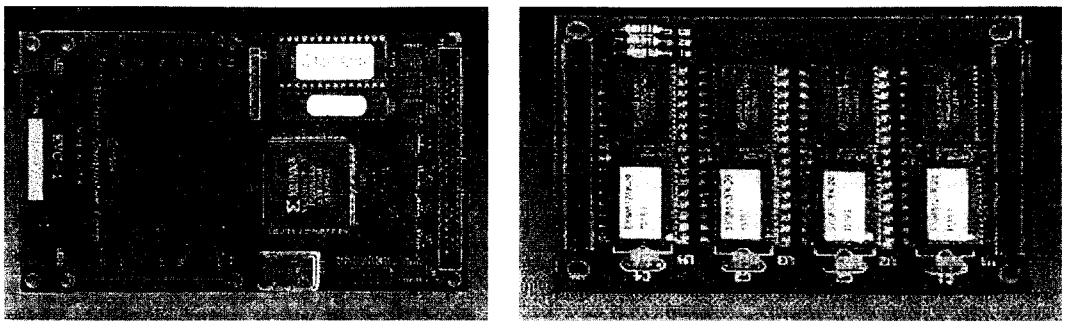

그림 2. a. EVC1 보드 외양, b. 2M SRAM 보드 외양
Fig. 2. a. EVC1 board's appearance. b. 2M SRAM board's appearance

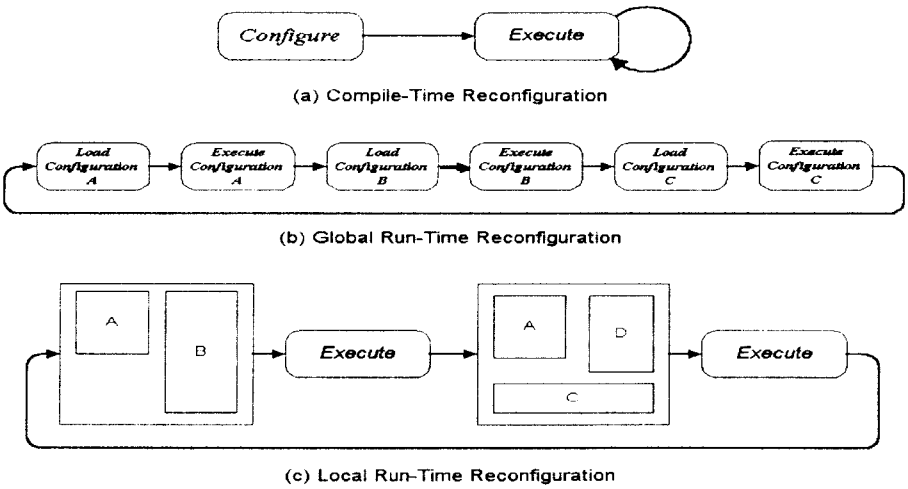

그림 3. 여러 가지 재구성 방법
Fig. 3. Various Reconfiguration methods

그래머블 논리 게이트와 프로그래머블 연결선으로 구성된 집적 회로의 일종으로 최종 사용자에 의해 원하는 기능을 나타내도록 이들 게이트와 연결선이 구성(또는 재구성)된다. 현재 나오고 있는 재구성 가능한 컴퓨팅 시스템은 대부분 호스트 컴퓨터에 대한 co-processing 디바이스 역할을 하며 호스트 컴퓨터의 슬롯에 직접 꽂을 수 있는 형태로 개발되고 있다. 본 연구에서 사용한 재구성 가능한 FPGA 시스템은 VCC사가 개발한 EVC1 보드로 SUN 워크스테이션의 S버스 슬롯에 직접 꽂아서 사용할 수 있다. 그림 2(a)는 사용된 EVC1보드의 외형을 나타낸 것으로 아래쪽의 조그만 IC가 프로그래머블 클럭 발생기이고 가운데 IC가XC4013PQ208이며, 위쪽에 있는 PLA는 EVC1보드와 workstation 간의 S버스를 통한 상호 통신을 제어하는 목적으로 사용된다.

알고리즘을 재구성 가능한 FPGA 시스템 상에서 hardwiring에 의해 구현하는 방법에는 크게 두 가지가 있다[11]. 하나는 컴파일 시점(compile-time) 재구성법이고 다른 하나는 실행 시점(run-time) 재구성법이다. 전자는 원하는 알고리즘을 수행하는 동안 하나의 FPGA가 하나의 동일한 configuration을 계속 유지하는 것을 말하므로 기존의 EDA 설계 장비로서도 원하는 알고리즘을 충분히 구현 가능하다. 후자는 알고리즘이 시간적으로 서로 무관한 여러 개의 부분으로 분할될 수 있는 경우, 하나의 FPGA가 각 분할에 대응하는 configuration을 여러 단계에 걸쳐 동적으로 가질 수 있는 것을 말하므로 기존의 EDA 설계 장비로서는 실행 시점 재구성에 의해 알고리즘을 구현하기는 불편한 점이 있다.

후자는 다시 각 단계 수행마다 FPGA의 configuration이 변하는 형태에 따라 전체적 run-time 재구성과 부분적 run-time 재구성으로 나뉜다. 전자는 각 단계

수행시 FPGA의 configuration이 전체적으로 바뀌어지는데 반해, 후자는 각 단계 수행시 FPGA의 configuration내 일정 부분만 원하는 기능을 갖도록 재구성되고 나머지는 변하지 않고 그대로 유지되는 것을 말한다. 그림 3은 앞에서 설명한 여러 가지 재구성 방법을 예시한 것이다. EVC1보드 상에 장착된 XC4013PQ208은 국부적 변경이 불가능하므로 본 연구에서는 전체적 run-time 재구성법에 의해 알고리즘을 구현하였다. 현재 출시되고 있는 XC6020은 run-time시 국부적 재구성이 가능하므로 재구성 시간이 크게 단축되고 응용분야가 더욱 커질 것으로 전망된다. 나아가, 실행 시점 재구성에 의한 알고리즘 구현 시에는 앞선 configuration의 중간 결과를 뒤따르는 configuration이 입력 데이터로 사용하는 경우가 흔하므로 이를 위해 configuration간의 데이터를 주고받을 수 있도록 메모리 보드가 요구된다. 이를 위해 본 연구에서는 그림 2(b)에 보인 2M SRAM 보드를 사용하였다.

4. 퍼지 제어기의 구현 과정

제안한 퍼지 제어기를 구현하는 과정은 다음과 같다. 각 모듈은 VHDL 언어에 의해서 기술된 뒤, Synopsys사의 FPGA 컴파일러[12]에 의해 합성된다.

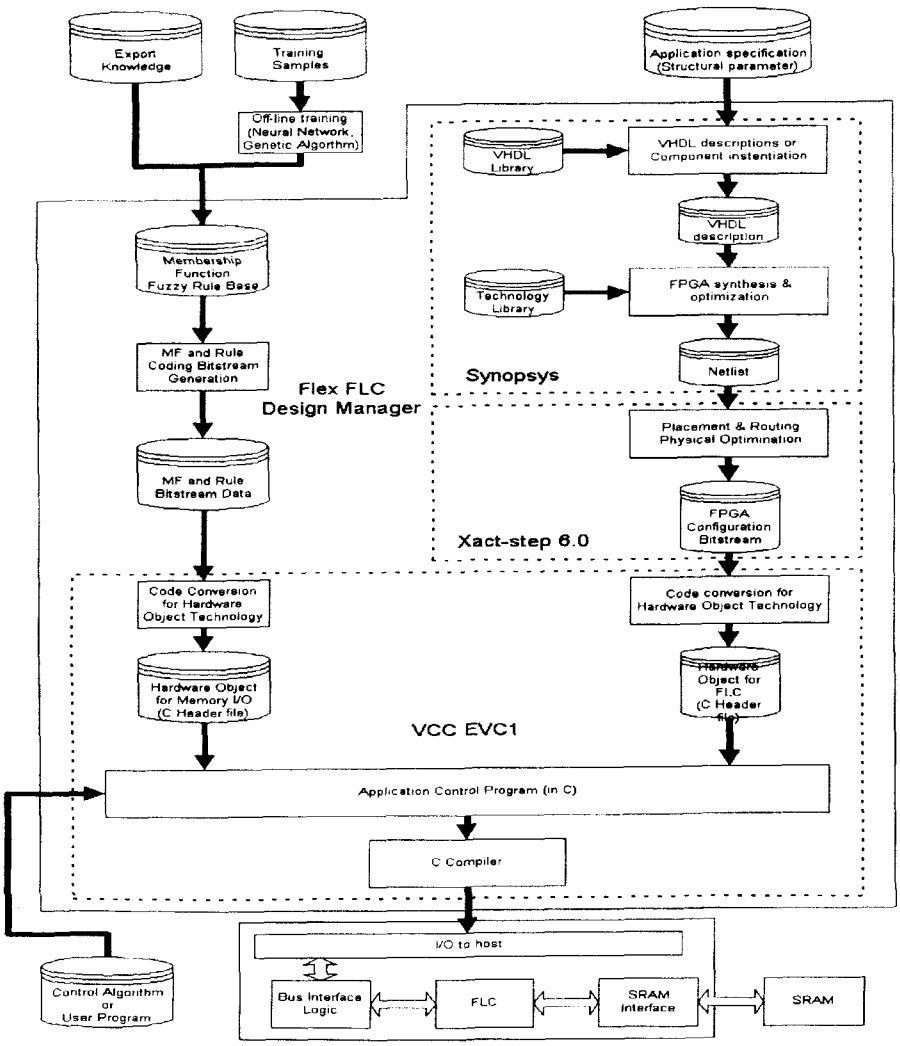

그림 4. 제안한 퍼지 제어기의 하드웨어 구현 과정
Fig. 4. An implementation procedure of the proposed FLC

합성된 각 모듈은 Xilinx사의 XactStep 6.0[13]에 의해 최적화 및 배치 배선이 이루어진다. 얻어진 Xilinx rawbit 파일은 VCC사의 r2h[14]에 의해 C 언어 프로그램의 header 파일 형태의 하드웨어 object로 변환된다. 원하는 목적(본 논문의 경우 트럭 후진 주차 제어)을 수행하기 위해 이들 하드웨어 object를 포함하는 응용 프로그램을 C 언어로 작성한 후 이를 C 컴파일러에 의해 컴파일 한다. 이 실행 파일은 재구성 가능한 FPGA 시스템인 EVC1 보드[11]로 다운로드 되어 원하는 목적을 수행하는지를 점검하게 된다. 그림 4는 본 연구에서 수행된 퍼지 제어기의 하드웨어 구현의 전체 과정을 나타낸 것이다.

4.1 Front-End 처리

제안한 퍼지 제어기를 재구성 가능한 FPGA 시스템 상에 구현하기 위한 첫 단계는 VHDL 언어로 기술된 각 모듈을 게이트 수준으로 netlist로 합성하는 것이다. 이를 위해 본 연구에서는 Synopsys사의 FPGA 컴파일러를 사용하였다. FPGA 컴파일러를 사용하기 위해서는 먼저 사용되는 FPGA의 모델에 따라 요구되는 link 라이브러리와 target 라이브러리를 설정해야 하며 FPGA 컴파일러가 동작하는 동작 조건과 만족시켜야하는 시간 및 면적에 대한 제약 조건들이 명시되어 있다[12].

제안한 퍼지 제어기의 각 모듈을 게이트 수준에서 합성하는 과정을 설명하면 다음과 같다. 먼저 FPGA 컴파일러가 VHDL 파일을 받아들여(read 명령) 원하는 제약 조건하에서 합성한 후(compile 명령) 합성 결과를 게이트 레벨의 netlist 파일(이 경우 Synopsys xnf 파일)로 저장한다(write 명령). 여기서 읽혀지는 VHDL 파일은 각각 합성시 사용되는 연산자 package, 입출력 포트 어드레스를 정의한 package, workstation의 S버스 인터페이스 프로그램, 메모리 인터페이스 프로그램, coarse 모듈 프로그램, 그리고 앞의 5개 VHDL 프로그램을 결합한 최상위 프로그램을 의미한다. 그리고, replace_fpga 명령은 컴파일 결과 얻어진 파일내 존재하는 CLB셀이나 IOB셀 등을 Xilinx FPGA 칩내에서 실제 사용 가능한 기본 셀(and 또는 or 등 게이트 수준)로 바꾸어 주는 역할을 한다. 위 과정을 제안한 퍼지 제어기의 각 모듈에 적용하여 게이트 수준으로 합성된 각 모듈의 netlist 파일(min.sxnf, max.sxnf, inference.sxnf, coarse.sxnf, fine.sxnf, control.sxnf)을 얻음으로서 퍼지 제어기 구현의 전단계가 끝난다. 그림 5는 게이트 수준으로 합성된 제안한 퍼지 제어기의 최상위층의 스키메틱 표시를 나타낸 것이다.

4.2 Back-End 처리

제안한 퍼지 제어기를 재구성 가능한 FPGA 시스템 상에 구현하기 위한 두 번째 단계는 앞 단계에서 얻어진 netlist 파일(*.sxnf 파일)을 Xilinx FPGA상에 적합하도록 배치 및 배선을 하여 논리 셀 어레이 파일(*.lca 파일)을 얻는 과정으로 구체적 내용은 다음과 같다. 먼저 syn2xnf 명령어는 Synopsys사의 FPGA 컴파일러가 만든 netlist를(*.sxnf 파일) Xilinx사의 XactStep 6.0에서 사용 가능한 netlist로(*.xnf 파일) 변환시키는 역할을 한다. 다음 xnfmerge 명령어는 여러 개의 xnf 파일을 다음에 오는 배치 및 배선을 쉽게 하도록 하기 위하여 하나의 펼쳐진(flat) netlist 파일 (*.xff)로 변환시키는 역할을 한다. 다음 xnfprep

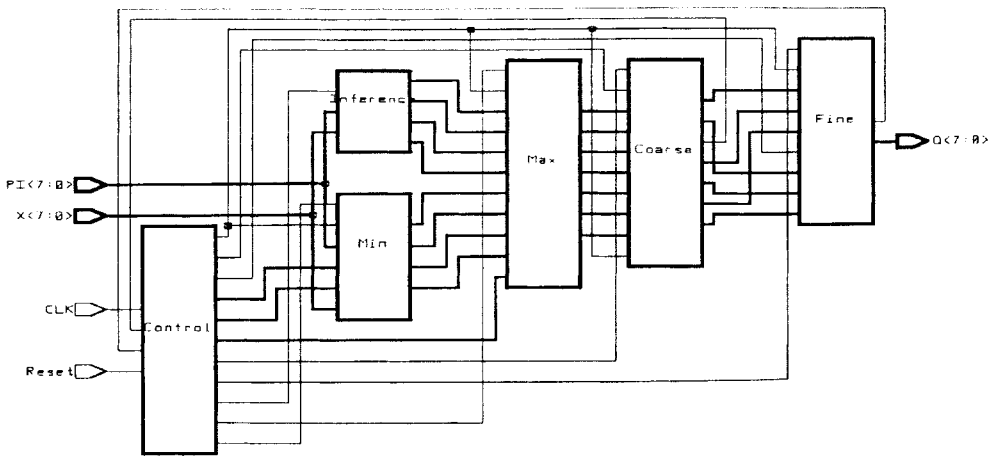

그림 5. 합성된 퍼지 제어기의 최상위층 스키메틱 표시
Fig. 5. A top-level schematic view of the synthesized FLC

명령어는 설계 규칙에 맞는가를 검증하고(DRC) 디바이스에 독립적으로 얻어진 netlist 형태를 특정 target인 FPGA의 형태(*.xtf)에 적합하도록 변환시키는 역할을 한다. 다음 ppr 명령어는 변환된 netlist를 Xilinx의 기본 구성 요소인 CLB와 IOB로 매핑하고, 매핑된 CLB와 IOB를 연결하는 선의 길이가 최소가 되도록 배치시키며, 마지막으로 위치가 정해진 CLB와 IOB간을 최소 지연 시간을 갖도록 연결시킨다. ppr 명령어 수행시 입력으로 사용되는 *.cst 파일은 FPGA가 갖는 실제 핀에 FPGA가 원하는 동작을 하도록 하는데 요구되는 기능적 핀을 매핑시킨 정보를 갖고 있어 원하는 대로 회로를 쉽게 변경 가능하게 한다. 다음 makebits 명령어는 Xilinx FPGA에 원하는 기능을 하도록 하는 configuration 파일에 대응하는 비트열을 자동적으로 발생시킨다.

4.3 재구성 가능한 FPGA 시스템상에 FLC 구현

앞에서 설명한 실행 시점 재구성에 의해 원하는 알고리즘을 구현하는데 기존의 EDA 설계 장비를 이용하는데는 불편한 점이 따른다. 이를 해소하기 위해 본 연구에서는 VCC사가 개발한 하드웨어 object 기술(Hardware Object Technology, H.O.T)을 사용하였다. 이 기술에 의하면 EVC1에 원하는 알고리즘을 구현하는 과정은 다음과 같다[15].

1) 앞에서 얻어진 퍼지 제어기의 각 모듈의 raw 비트 파일 (*.rbt)을 r2h 명령어에 의해 동적으로 다운로딩 가능하도록 하드웨어 object인 header 파일로 (*.h) 변환시킨다. 얻어진 하드웨어 object는 virtual processing 라이브러리에 저장되어 재사용이 가능하다.

2) 하드웨어 object는 원하는 제어 목적에 맞는 응용 프로그램을 C언어로 작성할 때 필요하면 header 파일로 불려와 EVCdownload 명령어에 의해 EVC1 보드에 다운로드되어 FPGA를 configuration한다.

3) 하나의 configuration이 수행을 종료하면 다음 configuration을 위해 EVC1보드를 EVC reset 명령어에 의해 초기화시킨다.

EVC1 보드 상에 하드웨어 object를 다운로드시켜 원하는 기능을 수행하기 위해서는 Host(SUN 워크스테이션)와 EVC1 보드내의 FPGA 상에 다운로드된 하드웨어 object(사용자 정의 모듈) 사이에 데이터를 주고받기 위한 S버스 인터페이스와 EVC1 보드내의 FPGA 상에 다운로드된 하드웨어 object와 2M SRAM 보드 상에 데이터를 읽고 쓰기 위한 메모리 인터페이스를 FPGA상에 구현하는 것이 요구된다. 이들 두 인터페이스 로직은 FPGA내 CLB의 약 1.5%에서 11%정도를 사용하므로 실제 하드웨어 object 구현하는데 사용되는 CLB수는 그만큼 줄어든다. 그림 6은 EVC1 보드상의 FPGA상에 구현된 두 개의 인터페이스와 주변 Host 컴퓨터와 메모리 보드사이의 연결을 나타낸 것이다.

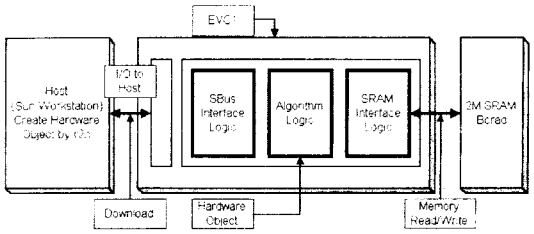

그림 6. 두 인터페이스와 주변 시스템사이의 관계
Fig. 6. Relationship between two interfaces and external systems

VCC사에 의해 제공된 위의 두 인터페이스는 Viewlogic 시스템에서 만든 스키메틱 마크로 상태여서 Viewlogic 설계 장비가 없는 관계로 이를 그대로 사용할 수가 없어서 본 연구에서는 앞서의 퍼지 제어기 내 여러 모듈을 얻는 과정과 마찬가지로 이들 인터페이스 로직을 VHDL을 사용하여 기술한 후 이를 Synopsys에 의해 합성하여 사용하였다. 메모리 인터페이스는 입출력 버퍼들과 읽기/쓰기 제어 신호들로 구성되어 있다. 메모리 인터페이스와 메모리 모듈의 연결은 입출력이 동일한 양방향 포트에 의해 이루어지고, 사용자 정의 모듈과의 연결은 단방향 포트 2개로 이루어진다. 메모리 모듈은 read, write, CS 신호에 의해 제어되고, 메모리 모듈과 연결된 양방향 포트는 read, write 신호에 의해 제어된다. 모든 신호는 low 상태에서 동작하므로, read 신호가 low 상태이고, write 신호가 high 상태이면 OE가 high 상태가 되고, 메모리 모듈로부터 데이터를 읽어 들이는 상태가 되므로, 양방향 포트는 메모리 모듈로부터 데이터를 읽어 들이는 기능만을 수행한다. 반대로, 경우도 이와 동일하게 메모리 모듈로 쓰고자 하는 데이터를 출력하는 기능을 수행한다.

FPGA가 갖는 CLB 및 IOB 개수가 유한하므로 제안한 퍼지 제어기를 하나의 FPGA에 구현하는 것을 불가능하므로 퍼지 제어기를 어떻게 분할하는 것이 효과적인가 하는 분할 문제에 부딪히게 된다. 본 연구에서는 제안한 퍼지 제어기의 각 모듈을 하나의 FPGA에 대응시킴으로서 이 문제를 간편하게 해결하였다. 표 1은 각 모듈의 배치 및 배선이 끝난 결과 사용한 CLB, flipflop, 및 I/O 핀의 백분율을 나타낸 것이다. 이 표으로부터 하나의 FPGA에 하나의 모듈을 매핑시키는 분할법이 아주 효과적이지는 않지만 각 모듈

표 1. 각 모듈의 FPGA 요소 사용율
Table 1. Utilization ratio of FPGA elements in each module

	Min	Inference	Max	Coarse	Fine
CLBs	308(53%)	196(34%)	353(61%)	510(88%)	362(62%)
F/Fs	249(21%)	149(12%)	253(21%)	294(25%)	185(16%)
I/O pins	102(63%)	102(63%)	102(63%)	102(63%)	102(63%)

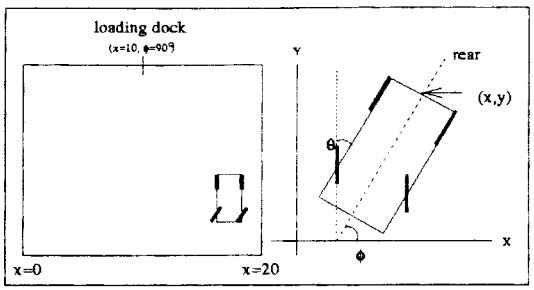

그림 7. 모형 트럭과 주차대 위치
Fig. 7. A model truck and loading dock

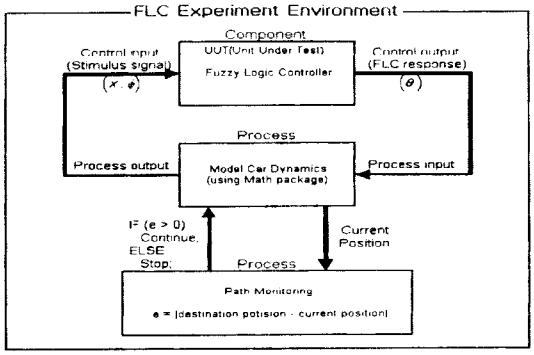

그림 8. 트럭 후진 주차 제어 실험 환경
Fig. 8. Experimental environment for the truck backer-upper control problem

내의 인터페이스 부분이 서로 동일하므로 실행 시간에 부분적으로 재구성이 가능한 FPGA 시스템을 사용하는 경우 재구성 시간을 줄일 수 있는 장점이 있다.

5. 실험 및 결과 분석

트럭 주차 문제의 목표는 가능한 빨리, 그리고 정확하게 트럭을 주차시키는 것이며, 이 문제는 기존 제어 기술로는 풀기 힘든 전형적인 비선형 제어 문제로 FLC나 신경망 제어기를 사용하여 제어하기에 적당하다[16]. 그림 7은 트럭 주차 제어 문제에서 사용된 트럭과 주차대의 위치를 보여준다. 트럭의 위치는 (x, y, ϕ)에 의해 결정된다. 단, 여기에서 ϕ는 트럭 진행 방향과 x축간의 각도이며, 트럭의 후진 주행 제어는 ϕ와 핸들의 축 간의 각도인 θ에 의해 결정된다. 트럭이 움직이는 운동 방정식은 아래와 같이 나타내어진다.[16]

$$x(t+1) = x(t) + \cos[\phi(t) + \theta(t)] + \sin[\theta(t) \cdot \sin[\phi(t)]$$
$$y(t+1) = y(t) + \sin[\phi(t) + \theta(t)] - \sin[\theta(t) \cdot \sin[\phi(t)]$$
$$\phi(t+1) = \phi(t) - \sin^{-1}[2\sin(\theta(t)/b)] \quad (1)$$

여기서 b는 트럭의 길이이며, 본 논문에서는 b = 4로 하였다.

그림 8은 구현한 퍼지 제어기를 트럭 후진 주차 문제에 직접 적용할 때 고려한 시뮬레이션 환경을 나타낸 것이다. 여기서 퍼지 제어기는 앞에서 설명한 구현 과정을 통해 EVC1 보드상에 구현된 것을 나타내고, 재구성 가능한 FPGA 시스템의 일종인 아래 부분의 모델 트럭은 식 (1)에 의해 동작하는 하나의 프로세스로 취급하였다. 실험 과정은 주차대내 임의의 한 제어 입력(x, y, ϕ)가 퍼지 제어기에 가해지면 EVC1 보드상의 퍼지 제어기는 제어 출력 θ를 연산하여 모델 트럭으로 내보낸다. 모델 트럭은 제어 출력 θ에 대해 모델 트럭의 운동 방정식인 식 (1)을 통해 다음 제어 입력 (x, y, ϕ)를 계산한다. 아래 경로 모니터링부에서는 위와 같은 과정을 반복적으로 수행하여 모델 트럭의 움직이는 궤적을 모니터링한다.

제안한 퍼지 제어기를 트럭 후진 주차 문제에 적용하여 재구성 가능한 FPGA 시스템의 일종인 EVC1 보드 상에 구현한 경우와 Synopsys사의 VHDL 시뮬레이터 상에 구현한 경우, 워크스테이션상에 C언어에 의해 구현한 경우 각각의 제어 수행시 연산 속도를 서로 비교하기 위해 목적지 도달에 관계없이 임의로 선택한 1,000개의(x, ϕ)점에 대한 퍼지 제어기의 출력 θ가 얻어질 때까지 걸리는 시간을 평균하여 얻은 평균 퍼지 연산시간을 비교하였다(실제 Synopsys사의 VHDL 시뮬레이터 상에서 퍼지 연산을 수행하는데는 너무 많은 연산시간이 걸려 10개의 10스텝에 대해서만 수행하여 평균 시간을 얻었다). 이 실험에서 세 가지 경우에 대한 수행시간 비교시 실험 조건을 될 수 있는 데로 공평하게 비교하기 위해 Synopsys VHDL

표 2. 세 가지 경우 평균 퍼지 연산시간
Table 2. Average fuzzy operation time for three different cases

	Synopsys VHDL simulation	C language simulation	FPGA implementation
퍼지연산 수행 수	10	1000	1000
전체 걸린 시간	320.4	88.36	5.14
평균 퍼지 연산 시간	32.04	0.08830	0.00514
FOPS	0.03	11.32	194.55

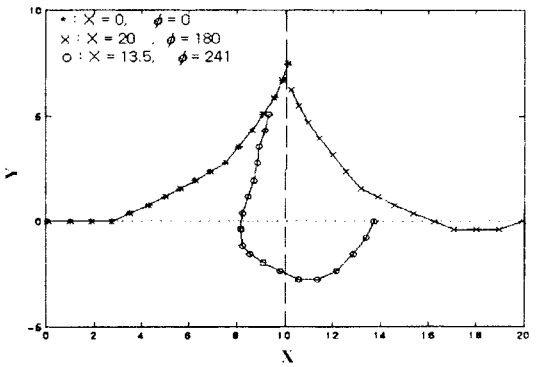

그림 9. 임의의 세 점으로부터의 모형 트럭의 주행곡선
Fig. 9. Trajectories of a model car from three random positions

시뮬레이션의 경우 그래픽 환경에서 수행하지 않고 Shell 환경에서 Vhdlsim을 이용하였다. 표 2는 위의 세 가지 경우의 초당 수행한 퍼지 연산의 개수(FOPS: Fuzzy operations per second)를 나타낸 것이다.

제안한 퍼지 제어기를 재구성 가능한 FPGA 시스템 상에 구현한 경우 이들이 시뮬레이션에서와 똑같이 제대로 동작하는지를 알아보기 위해 임의의 세 점(왼쪽 그림 시작점 : (0.0, 0.0, 0°), 가운데 시작점 (13.5., 0.0, 241°), 오른쪽 그림 시작점 (20.0, 0.0, 180°))에서의 목적지까지 어떻게 주행하는가를 조사하였다. 그림 9로부터 세 경우 각각 16 스텝, 18 스텝, 15스텝 걸려서 목적지에 도달하는 것을 알 수 있다. 이로 미루어보아 재구성 가능한 FPGA 시스템 상에 구현한 퍼지 제어기는 정확히 원하는 데로 동작함을 확인할 수 있다.

6. 결 론

본 논문은 하드웨어 복잡도를 크게 줄인 승산기 및 제산기 없는 퍼지 제어기를 재구성 가능한 FPGA 시스템 상에 직접 hardwiring에 의해 구현하였다. 본 연구에서 구현한 퍼지 제어기는 기존의 퍼지 제어기 구조와 크게 달라, 1. 기존의 MIN-MAX 추론을 레지스터 파일 상에 read-modify-write 연산에 의해 대치했으며, 2. 비퍼지화값을 계산하는데 있어 소속 함수의 중심 출력 값과 소속함수의 폭을 동시에 고려하였고, 나눗셈기를 사용하지 않고 coarse-to-fine 탐색법에 의해 효과적으로 모멘트 균형점을 찾는 새로운 형태의 COG 비퍼지화기를 사용하였다.

제안한 퍼지 제어기의 재구성 가능한 FPGA 시스템상의 구현 과정은 다음과 같다. 각 모듈은 VHDL 언어에 의해서 기술한 후, Synopsys사의 FPGA 컴파일러에 의해 합성하였다. 합성된 각 모듈은 Xilinx사의 XactStep 6.0에 의해 최적화 및 배치 배선의 back-end 처리를 하였다. 얻어진 Xilinx rawbit 파일은 VCC사의 r2h에 의해 C 언어 프로그램의 header 파일 형태의 하드웨어 object로 변환시킨 후, 이들 하드웨어 object를 포함하는 퍼지 제어기 제어용 C 응용 프로그램을 작성한 후 이를 C 컴파일러에 의해 컴파일하였다. 이 실행 파일을 재구성 가능한 FPGA 시스템인 EVC1 보드에 다운로드하여 원하는 제어 목적을 수행하는지 확인하였다.

재구성 가능한 FPGA 시스템 상에 구현한 퍼지 제어기를 트럭 후진 주차 문제에 적용하여 그 연산 속도를 Synopsys사의 VHDL 시뮬레이터 및 워크스테이션상에 C 언어로 구현한 경우의 연산 속도와 비교하였는데, FPGA상에 구현한 경우가 VHDL 시뮬레이터보다는 $O(10^3)$배, C언어에 의한 시뮬레이션보다는 O(10)배 정도 빠름을 확인하였다. 아울러, 재구성 가능한 FPGA 시스템 상에 구현한 퍼지 제어기가 제대로 동작하는지를 확인하고자 임의의 세 점에 대한 모형 트럭의 주행 경로를 추적하여 보았는데 세 경우 모두 14-18 스텝 내에 목적지에 제대로 도달하는 것을 확인할 수 있었다.

참고문헌

[1] E. H. Mandani, "Application of fuzzy algorithms for control of simple dynamic plant," *IEEE Proc. Control & Science*, Vol. 121, No. 12, pp. 1585-1588, Dec. 1974.

[2] M. Togai and H. Watanabe, "Expert System on a Chip: An Engine for Real-time Approximated Reasoning," *IEEE Expert System Magazine,* Vol. 1, pp. 55-62, 1986.
[3] T. Yamakawa and I. Miki, "The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process," *IEEE Transactions on Computers*, Vol. 35, No. 2, Feb. 1986.
[4] S. Casselman, "Virtual Computing and Virtual Computer," *Procrrdings of IEEE Workshop on FPGAs for Custom Computing Machines,* 1993.
[5] S. Mazor and P. Langstraat, A Guide to VHDL, Kluwer Academic Publishers, 1993.
[6] Xilinx, "The Programmable Logic Data Book," Xilinx., 1996.
[7] 김대진, 조인현, "고정밀 저비용 퍼지 제어기-VHDL 설계 및 시뮬레이션" 대한 전자 공학회 논문지, 제 34권 7호, pp. 38-50, 1997년 7월.
[8] Daijin Kim and In-Hyun Cho, "An Accurate and Cost-Effective Fuzzy Logic Controller with a Fast Searching of Moment Equilibrium Point," *IEEE Trans. on Industrial Electronics,* Vol. 46, No. 2, pp. 1-14, April 1999.
[9] Y. Kondo and Y. Sawada, "Functional Abilities of a Stochastic Logic Neural Network," *IEEE Transactions on Nerual Networks,* Vol. 3, No. 3, pp. 434-443, May 1992.
[10] A. Ruitz, J. Gutirrez, and J. Fernndez, "A Fuzzy Controller with an Optimized Defuzzification Algorithm," *IEEE Micro,* pp. 1-10, Dec. 1995.
[11] VCC, "EVC1-Virtual Computer Programming Tutorial," VCC Corp., 1994.
[12] SYNOPSYS, "Design Compiler Reference Manual v3.4," Synopsys Corp., 1996.
[13] Xilinx, "XactStep Development System User Guide," Xilinx., 1996.
[14] VCC, "EVC1 Technical Reference," VCC Corp., 1995.
[15] S. Casselman, M. Thornburg, J. Schewel, "H.O.T (Hardware Object Technology) Programming Tutorial," VCC Corp., 1995.
[16] Li-Xin Wang and Jerry M. Mendel, "Generating Fuzzy Rules by Learning from Examples," *IEEE Transactions on System,* Man, and Cybernetics, Vol. 22, No. 6, pp. 1414-1427, Nov. 1992.

김 대 진 (Dai-Jin Kim)

제 8권 제 5호 참조

조 인 현 (In-Hyun Cho)

제 8권 제 5호 참조