한국정밀공학회지 제 25 권 제 9 호 (2008 년 9 월)
Journal of the Korean Society for Precision Engineering Vol. 25, No. 9, September 2008

RTAI 기반의 웨이퍼처리 로봇 제어기 구현

장순필*, 신익상**, 문승빈#

Implementation of Wafer Handling Robot Controller Based on RTAI

Soonpill Chang*, Iksang Shin** and Seungbin Moon#

ABSTRACT

As multiple functions are required in a robot controller, RTOS(Real Time Operating System) should be adopted to manage complex situations, such as choosing most urgent task among competing ones. In this paper, we implemented RTAI(Real Time Application Interface) based robot controller for wafer handling robots including graphic simulator. We showed how multiple tasks are organized and also explained in detail about task priorities and execution periods. Finally, we presented simulation results.

Key Words : Linux (리눅스), Robot controller (로봇 제어기), RTAI (실시간 애플리케이션 인터페이스), RTOS (실시간 운영체제)

1. 서론

산업 고도화와 더불어 로봇제어기의 기능 및 성능향상 뿐만아니라 주어진 작업을 실시간으로 처리하는 요구가 증대되고 있다.[1-2] 실시간 처리란 규정된 시간 내에 주어진 작업이 처리되는 것을 의미한다. 로봇 분야 역시 GUI(Graphic User Interface), 그래픽 시뮬레이터등 많은 시스템 자원을 요구하는 작업들이 증가되었으며, 이러한 작업들이 수행되는 중에도 로봇 동작이 정확하게 이루어지기 위해서는 실시간 처리가 요구된다.[3-7]

본 논문에서는 실시간 운영체제인 RTAI를 이용하여 로봇 제어기를 구현하였다. 로봇 제어기는 주어진 시간 내에 사용자가 지시하는 명령을 수행해야하고, 로봇의 동작 역시 정확해야 한다. 이를 위해서 로봇 제어기를 구동해주는 운영체제가 실시간성을 보장해야한다. 실시간 운영체제와 일반 운영체제의 가장 큰 차이점은 수행되는 태스크(task)들이 운영체제자원을 사용하여 수행중지(block)상태가 되더라도 마감시간(deadline)을 정확히 맞추어 수행시켜 주는 것이다. 현재 실시간 운영체제 시장에서는 VxWorks, QNX 등 실시간 운영체제가 사용되고 있다.[8-10] 하지만, 이 제품들을 사용하기에는 제품들이 고가이며, 제품 마다 라이선스비를 지불해야하는 문제점이 있다. 이에 대한 대안으로 오픈소스기반인 Linux[11]의 커널과 실시간성을 보장해주

☞ 접수일: 2008년 1월 2일; 게재승인일: 2008년 5월 30일
* 세종대학교 컴퓨터공학과
** 경희대학교 기계공학과
\# 교신저자: 세종대학교 컴퓨터공학과
E-mail: sbmoon@sejong.ac.kr Tel. (02) 3408-3243

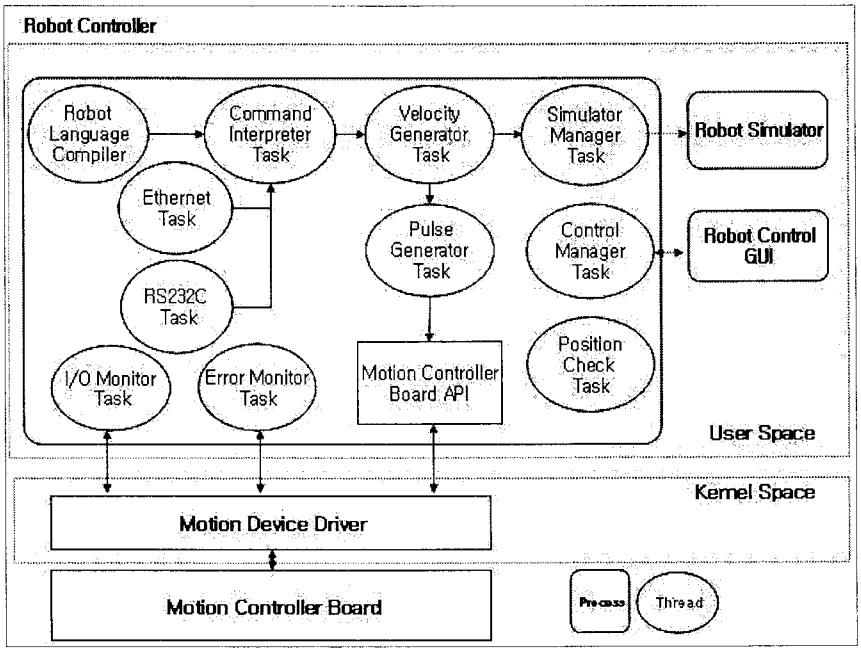

Fig. 1 Task configuration of robot controller

는 RTAI를 사용하여, 이런 점들을 해결할 수 있다.[12,13]

본 논문의 구성은 다음과 같다. 2장에서는 제어기의 구성, 3장에서는 제어기의 구현 방법에 대해서 살펴보고, 4장에서는 실험 결과, 5장에서는 향후 연구방향에 대하여 다루어 보겠다.

2. 로봇 제어기의 구성

일반적으로 로봇 제어기는 매우 복잡하게 구성되어 있으므로 각각의 요구되는 작업들을 실시간 OS 환경하에서 독립적인 작업(Task)으로 프로그래밍하는 것이 S/W관리 측면에서 매우 유리하다. 이렇게 하면 각각의 작업 간에 우선순위를 지정함으로서 스케줄링이 지정되게 되고, 프로그래머는 각각의 작업을 별개의 작업처럼 작성할 수 있어서 프로그램이 용이하게 된다. 물론 작업간의 통신은 실시간 OS에서 제공하는 다양한 IPC(Inter Process Communication)를 통하여 하게 된다. 특히 RTAI는 완전한 실시간성을 보장하는 커널부분과 그렇지 못한 사용자공간으로 나누어서 제공됨으로, 실시간성이 매우 중요한 작업은 커널영역에서 처리를 하고 일반적인 작업은 사용자 영역에서 처리를 하게된다.

Fig. 1에서 보는 것처럼 로봇 제어기는 크게 두 영역으로 나뉜다. 우선 편집기, 인터프리터(interpreter), 시뮬레이터(simulator)와 같이 사용자 편의를 제공해주는 사용자 영역과 실제 로봇을 구동시켜주는 모션보드(motion board)와 사용자 프로그램과 모션보드를 연결해 주는 디바이스 드라이버(device driver)와 같은 커널 영역으로 나뉘어 진다. Fig. 2에서의 로봇은 이 논문에서 사용한 300mm 반도체 웨이퍼(wafer) 이송용 로봇으로 현재 사용하고 있는 초기 모델은 각 링크의 길이가 200mm인 평면 로봇이다.

2.1. 사용자 영역(User Space)

사용자 영역의 태스크들은 사용자의 로봇 제어 편의를 제공하는 태스크와 로봇을 제어하기 위한 태스크 그리고, 다른 태스크들의 동작을 관리하는 태스크들로 구성된다. 우선 사용자 편의를 위한 태스크는 로봇언어를 실제 로봇을 구동시키기 위한 명령어로 번역해주는 Robot Language Compiler, 네트워크상의 다른 컴퓨터에서 로봇을 제어할 수 있

도록 명령을 대기하는 Ethernet 태스크, TP(teaching pendant)와 같은 장치로부터 명령을 받기위한 RS232C 태스크가 있으며, 이 태스크들로 들어온

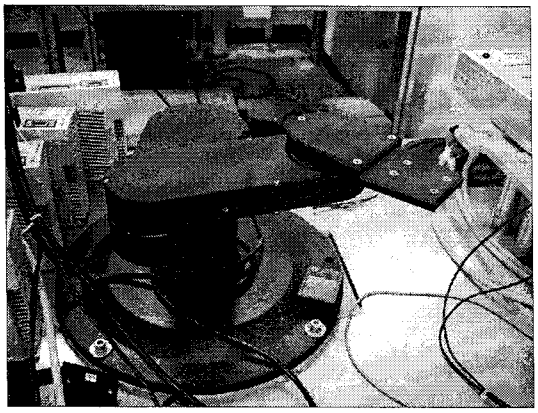

Fig. 2 Transfer robot for 300mm semiconductor wafer

명령을 모아서 일관된 명령 순서대로 로봇에게 명령을 전달해주는 Command Interpreter 태스크, 로봇의 동작을 볼 수 있는 로봇 시뮬레이터가 있으며, 로봇을 제어하기 위한 태스크는 Command Interpreter 태스크에서 전달된 명령 중 실제 로봇을 구동시키는 명령을 실행시킬 때 서보모터에서 출력될 펄스를 만들어주는 Velocity Generator 태스크가 있으며, 태스크들의 동작을 관리하는 태스크는 만들어진 펄스를 정해진 주기(10msec)마다 서보모터로 출력시켜주는 Pulse Generator 태스크, 혹은 시뮬레이터로 출력시켜주는 Simulator Manager 태스크로 구성된다.

2.2. 커널 영역(Kernel Space)

최근의 산업용 FA제어기는 종전의 전용기 형태에서 범용제어기 형태로 바뀌어 가고 있다. 이 같은 추세에 맞춰 모션 제어기 또한 PC기반 개방형 구조로 가는 추세이다.[4] 이 논문에서 사용하는 PCI 기반의 모션보드는 PC 또는 산업용 컴퓨터에서 다축 동시 제어를 하는 모션 제어보드이다. 한 개의 모션보드는 최대 4개축을 동시 제어 할 수 있으며, 또한 4개의 모션보드를 사용하여 최대 16축을 동시 제어할 수 있다. 모션보드는 TMS302C3X DSP를 메인 Processor로 사용하며, PC와 모션보드간의 통신을 위해 DPRAM(Dual Port RAM)을 사용한다.

사용자 영역의 프로세스와 모션보드 간에 통신을 하기 위해서 DPRAM을 접근을 해야 한다. 하지만, 실제 응용프로그램이 모션보드 내에 있는 DPRAM을 직접 접근한다는 것은 불가능한 일이다. 이 역할을 해주는 것이 Motion Board 디바이스 드라이버이다. Fig. 3에서 보이는 바와 같이 디바이스 드라이버는 모션보드내의 DPRAM을 실제 메모리의 물리주소와 맵핑하여 응용 프로그램이 물리 메모리를 접근하는 방법으로 실제 모션보드내의 DPRAM에 접근할 수 있게 해준다.

3. 로봇 제어기의 구성

3.1 RTAI(Real Time Application Interface)

Fig. 4에서 보듯이 RTAI에서는 HAL(hardware abstract layer)에 의해서 Linux커널은 유휴 태스크(idle task)로 취급된다. 그래서 RTAI 스케쥴러에 의해 관리되는 실시간 태스크는 Linux커널보다 높은 우선순위로 수행되어 실시간성을 보장받는다. 하지만, 실시간성을 제공하는 라이브러리들은 커널 영역에서만 사용할 수 있으며 사용자 영역에서는 사용하지 못하는 것이 일반적인 실시간 운영체제들의 특징이다.

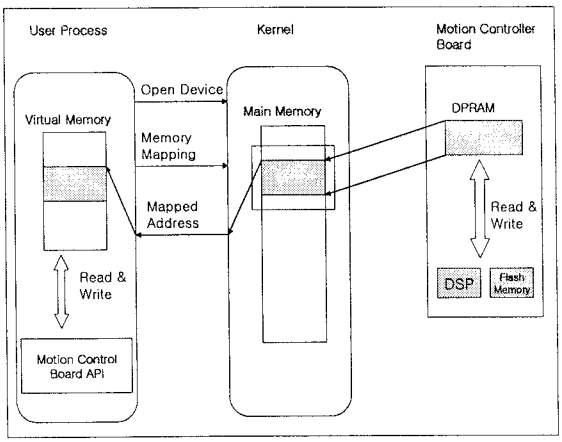

Fig. 3 Memory mapping of device driver

RTAI는 커널 영역에서 수행하기 힘든 디버깅 작업을 사용자 영역에서 수행 가능하도록 하는 LXRT 모듈을 제공하여 실시간 태스크를 작성할 수 있는 방법을 제시하였다.[2]

이 논문에서 구현하는 로봇 제어기는 운영체제로부터 실시간성을 보장받기위해 RTAI의 LXRT모

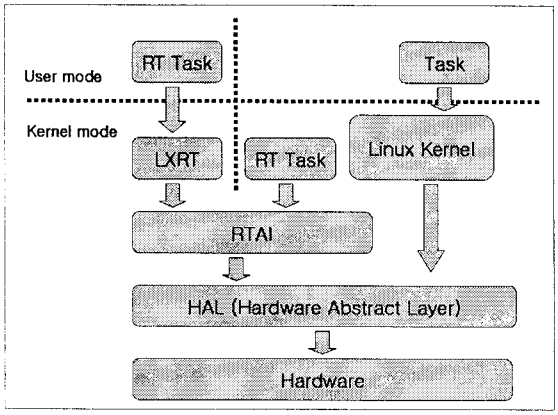

Fig. 4 Structure of RTAI kernel programming

듈 사용한다. LXRT 모듈은 연성실시간 (soft realtime) 성격을 가지며, 정확한 실시간성에 대해서는 제약을 받는다. 하지만, LXRT는 RTAI의 스케쥴러에 의해 관리를 받기 때문에 일반 Linux 프로세스가 작업을 진행 중이라 할지라도 RTAI의 스케쥴러의 관리를 받는 제어기 프로세스가 일반 Linux 프로세스를 선점하고 작업을 할 수 있다. 그렇기 때문에 일반 Linux 프로세스들보다 우선순위가 높게 적용되어 실시간성을 보장받는다.

Pulse Generator 태스크에서 단위 시간(10msec)마다 서보모터가 움직여야할 실제 펄스 값을 만들어주게 된다. 하지만 Pulse Generator Task에서 다음 단계에 출력해야 할 펄스 값을 단위 시간(10msec) 내에 출력하지 못하면 실제 구동중인 로봇의 동작은 중지하거나 잘못된 동작을 수행하게 된다. Velocity Generator 태스크와 실제 펄스를 모션보드로 내려주는 Pulse Generator 태스크간의 통신(IPC)을 실시간성이 보장되는 함수를 사용하지 않으면 위에서 언급한 문제가 발생할 가능성이 높아진다. 그래서 통신방법을 일반 POSIX 기반의 함수를 사용하지 않고 RTAI에서 제공하는 API를 사용하여 이 문제를 해결하였다.

제어기에서 사용되는 태스크들은 실시간성을 가지며, 다른 Linux상의 프로세스들보다 높은 우선순위를 가지고 있다. 그렇기 때문에 각 태스크들이 각자의 작업을 위해 바쁜 대기(busy waiting)를 하게 되면, 우선순위가 낮은 다른 작업들은 선점(preemption)을 당하여 수행되지 못하는 기아상황(starvation)이 발생한다. 그렇기 때문에 제어기의 실시간 태스크들은 자신에게 주어진 작업시간동안 작업을 수행하고, 수행 후 남은 시간을 다른 태스크들에게 양보해야한다. 그리고 다음 수행시간에 정확히 수행을 해야 한다. 이런 기능을 충분히 지원하는 RTAI 스케쥴러를 사용하여 이 문제점을 해결하였다.

3.2 Priority 설정

로봇 제어기에서 가장 중요한 점은 로봇의 정확한 구동이다. 그렇기 때문에 로봇의 모션을 제어하는 모션보드에 정확한 시간에 출력 값을 보내줘야 한다. 로봇 제어기에서 모션보드가 실제 로봇을 정확하게 구동하기 위해서는 작업의 마감 시간을 지켜야한다. 반면에 시뮬레이터의 경우는 로봇의 제어와는 관계가 없으나 다른 태스크에 비해 CPU를 많이 사용하기 때문에 자칫 시뮬레이터로 인해 로봇 제어에 문제가 발생할 수 있다. 그렇기 때문에 우선순위를 Table 1와 같이 설정하였다. RTAI의 우선순위는 낮은 숫자가 높은 우선순위를 가진다.

다음 절부터는 로봇 제어기를 구성하는 주요 태스크인 Command interpreter 태스크, Velocity Generator 태스크, 로봇 시뮬레이터 및 로봇 제어 GUI에 대하여 설명하겠다.

Table 1 Priorities of multiple tasks

Priority	Task	Period
10	Error Check Task	100 msec
20	Pulse Generation Task	10 msec
25	Simulator Manager Task	10 msec
30	Velocity Generator Task	50 msec
40	Command Interpreter Task	50 msec
45	I/O Monitor Task	100 msec
45	Ethernet Task	100 msec
45	RS232C Task	100 msec
-	Editor, Compiler	-
-	Robot Simulator	-

3.3 Command Interpreter Task

사용자가 로봇 제어기로 명령을 보내는 경로는 우선, 사용자가 작성한 로봇언어를 컴파일하여 보내는 Robot Language Task, 네트워크상 다른 컴퓨터로부터 보내는 Ethernet Task, TP(teaching pendant)와 같은 장치로 보내는 RS232C Task 와 같이 다양한 명령 경로가 존재한다. Command Interpreter 태스크

는 이들 경로들에서 들어오는 명령 중 긴급한 상황에 대처하는 명령을 우선적으로 처리하거나, 혹은 명령 경로들에 대해 우선순위를 제어한다.

3.4 Velocity Generator Task

Velocity Generator 태스크에서 수행하는 일은 서보모터 구동 속도를 가속 구간과 감속 구간을 추가하여 물리적으로 가능한 속도를 만들어 주는 속도 제어와 로봇의 이동 경로를 제어하여 사용자가 원하는 동작을 수행하도록 하는 경로 계획(path planning)이다. Velocity Generator 태스크에서 출력되는 펄스 값을 서보모터에게 직접 펄스를 출력하는 Pulse Generator 태스크에 전달하여 로봇을 구동하거나 시뮬레이터의 상태를 관리하는 Simulator Manager 태스크로 전달하여 로봇 시뮬레이터를 관리한다.

로봇의 동작을 제어하는 방법은 PTP(Point To Point)동작과 직선(straight)동작이 있다. 로봇 제어 방법 중 직선 동작의 경우는 PTP동작과 달리 로봇의 종단점(end effector)의 시작위치와 결과위치의 경로가 직선을 이루어야한다. 그러기 위해서는 로봇 이동에 대한 경로 계획이 필요하다. 경로 계획은 Table 2의 방법으로 수행하며 도출된 각 경로에 도달할 수 있는 로봇 관절의 이동 값들을 구하여 Pulse Generator 태스크와 Simulator Manager 태스크로 값을 보내준다.

Table 2 Straight motion procedure

Procedure
단위시간에 이동해야할 월드 좌표들의 집합을 구한다.
$P = \{p_1, p_2, p_3, \cdots, p_n\}$
$for\ i = 1\ to\ n$
좌표 p_i 에 도달하기 위해 각축의 이동 각을 구한다.
구해진 축의 각을 모터의 펄스 값으로 변환한다.
Pulse Generator Task 로 펄스 값을 전달한다.
$next$

로봇을 구동할 경우 로봇 관절의 이동 속도가 서보모터의 구동한계를 넘어서서 적용되면 모터에 과전류가 흐르게 되어 서보모터가 정지되는 현상이 발생한다. 이를 위해 다음과 같은 식을 이용하여 서보모터로 출력되는 출력 펄스를 구한다.

입력펄스 값으로부터 가감속이 반영된 출력 펄스를 구하는 방식은 마스크 함수와 입력펄스와의 컨벌루션을 계산하는 것이다. 마스크 함수는 아래의 수식 (1)에 의하여 구하였다.

$$\begin{cases} K(t) = A \cdot t & (t \leq n/2) \\ K(t) = A \cdot (n - t) & (t > n/2) \end{cases} \quad (1)$$

여기서, A: 가속도 상수, n: 가감속 구간의 수, t: 가감속 구간의 번호

수식(1)은 입력 펄스에 적용될 가감속 구간을 정의하기 위한 수식이며, 상수 A값을 변화시켜 서보모터에 적용되는 가속도를 제어할 수 있다.

수식(2)는 구해진 컨볼루션 마스크를 입력 펄스와 곱하여 실제 출력되는 펄스 값을 구하는 수식이다. 여기서 $O(i)$는 i번째 출력되는 출력 펄스, $K(t)$는 마스크 함수, n은 주어진 가감속 구간의 개수, $P(t)$는 t번째 입력 펄스를 의미한다.

$$\begin{cases} O(i) = \sum_{t=0}^{i} (K(t) \times P(t)) & (i \leq n) \\ O(i) = \sum_{t=i-n}^{i} (K(t) \times P(t)) & (i > n) \end{cases} \quad (2)$$

여기서, O(t): 출력 펄스, P(t): 입력 펄스

3.5 로봇 시뮬레이터

Fig. 5는 OpenGL을 이용하여 구현한 시뮬레이터이다. 이 시뮬레이터를 이용하여 실제 로봇의 구동 상태와 로봇의 수행과정을 확인 할 수 있다.

시뮬레이터상의 로봇을 구동하기 위해서 Simulator Manager 태스크에서 출력된 값을 이용하며 실제 로봇 구동과 시뮬레이션을 독립적으로 혹은 동시에 확인할 수 있으며, 실제 로봇과 시뮬레이터의 설정값(링크의 길이, 로봇 축의 수 등)을 동일하게 설정하여 로봇의 실제 동작 결과와 같은지 여부를 그래픽으로 확인 가능하다. 시뮬레이터는 10msec의 주기로 구동되지만, 낮은 우선순위 때문에 다른 Task에 의해 실제 수행 주기는 더 길어진

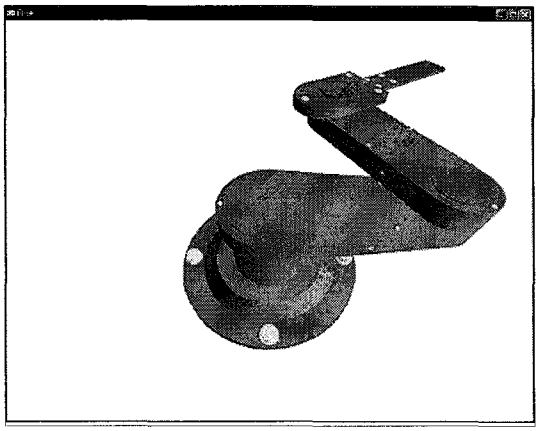

Fig. 5 Robot simulator

다. 따라서 실제 로봇과 비동기로 수행되며, 실시간성은 가지지 못한다. 로봇 시뮬레이터와 시뮬레이터 매니저 태스크 간에는 UDP 통신방식을 채택하였다.

3.6 로봇 제어 GUI

Fig. 6은 로봇의 상태를 확인 혹은 로봇에게 동작 명령을 내릴 수 있는 제어 GUI 태스크의 화면이다. Fig. 6에서 보듯이 Current 항목에서는 현재 로봇의 각축의 값이 출력되며 하단 Destination 항목은 사용자가 원하는 로봇의 동작을 수행할 수 있다. 이 경우 수행하는 동작 방식은 PTP, 직선동작 그리고 각 로봇축에 직접 펄스를 출력하는 Joint 모드로 나눠진다. 그리고 마지막 오른쪽의 Signal 항목은 주변의 센서들로 들어오는 정보를 표시한다.

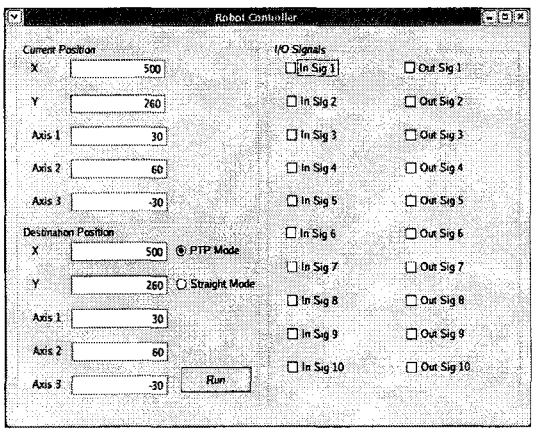

Fig. 6 GUI for robot controller

로봇 제어 GUI 태스크는 Linux 상에서 구동 가능한 QT라이브러리를 사용하여 구현하였으며, 로봇 제어기와의 통신은 로컬 IPC를 사용하였다.

4. 실험 결과

4.1 서보모터 출력 결과

Fig. 7과 8은 Velocity Generation 태스크에서 출력되는 펄스를 PTP동작과 직선동작에 대하여 각각 나타내었다. Fig. 7, 8에서 출력 펄스는 서보모터에 무리를 주지 않기 위해 가감속 영역을 만들어서 출력하는 값으로, 실제 적용되는 가속도를 보게 되면 무리 없이 서보모터를 구동 가능하다는 것을 알 수 있다. Fig. 7은 PTP(point to point)동작으로 모든 구간을 등속으로 이동하는 것을 부드러운 곡선을 생성함을 보여준다. Fig. 8은 직선 동작의 결과로서 PTP 동작과 마찬가지로 모든 구간에서 부드러운 곡선을 생성함을 보여준다.

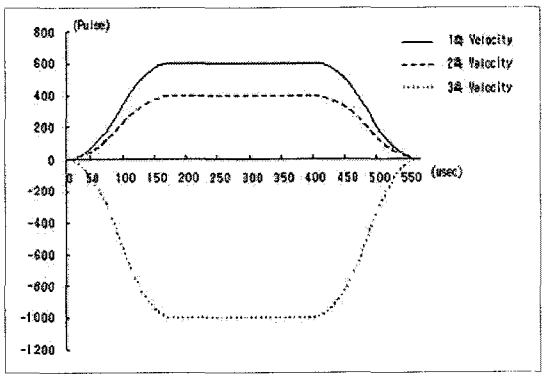

Fig. 7 Output pulses for a PTP motion

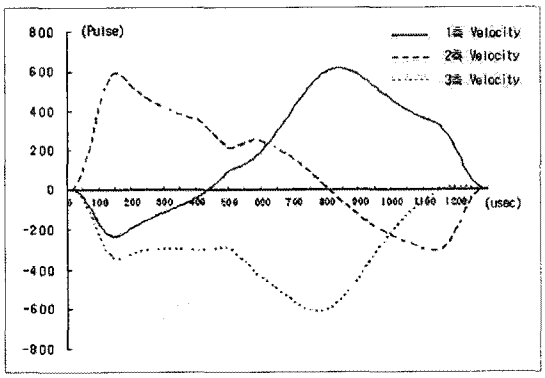

Fig. 8 Output pulses for a straight motion

4.2 실시간 태스크의 수행결과

Table 3은 제어기내 실시간 태스크들에 대한 태스크의 500회 수행 결과이다. 태스크내에서 하드웨어와 통신과 다른 태스크와의 동기화로 인한 수행중단(block)이 발생하지만, RTAI 스케쥴러에 의해 태스크의 정해진 수행시간을 정확히 지키며 수행되는 것을 알 수 있다. 최대 수행주기는 Pulse Generator 태스크의 수행주기며, 최소 수행 주기는 타이머 주기와 CPU 타임 틱(time tick)과의 차이에 의해서 발생하는 시간이다. 제어기에서 사용하는 실시간 타이머(realtime timer)의 타임 틱은 10usec이다.

Table 3 Execution periods for realtime tasks

태스크명	최소 수행주기 (usec)	최대 수행주기 (usec)	평균 수행주기 (usec)
Error Check	99968	100010	99986
Pulse Generator	9998	10998	10000
Simulator Manager	9965	10015	9992
Position Check	49682	50286	49997
I/O Monitor	49925	50000	49989
Velocity Generator	49982	49997	49992

Table 4는 실시간 태스크들의 실제 수행 시간으로서 IPC 함수 수행시간을 포함한 시간을 나타내며, Table 5는 로봇 제어기에서 사용한 IPC들의 수행시간을 나타낸다. Table 5에서 보는 것처럼 IPC 함수들의 수행시간은 극히 짧다는 것을 알 수 있다.

Table 4 Execution times for realtime tasks

태스크명	최소 수행시간 (usec)	최대 수행시간 (usec)
Error Check	134	386
Pulse Generator	0	2999
Simulator Manager	50	427
Position Check	8	17
I/O Monitor	0	59
Velocity Generator	5185	9181

Table 5 Latencies for IPCs

IPC	최소수행주기 (usec)	최대 수행주기 (usec)
Semaphore	8	17
Message Queue	8	17
UDP Packet	34	109

즉, IPC 함수 수행은 태스크 수행 시간에 영향을 거의 주지 않는 다는 결론을 내릴 수 있다. Table 4에서 최소 수행시간이 0이라는 것의 의미는 IPC 함수의 논블럭킹(none blcoking) 설정에 의하여 수행할 작업이 없는 것을 의미한다. 로봇 제어기에서 사용한 IPC중 세마포, 메시지 큐는 RTAI에서 제공하는 실시간 IPC이며 UDP는 Linux에서 제공하는 함수를 사용하였다.

5. 결론 및 향후 연구방향

Linux는 현재 연구용 뿐 만 아니라 상업용에서도 많은 적용이 이루어지고 있다. 그리고 이 논문에서 사용한 RTAI 커널 또한 진보되는 추세이다. 일반적인 실시간 태스크들은 커널 모드로 구현하여 실시간성을 부여한다. 하지만 이 논문에서 구현한 로봇 제어기에서는 RT Task 커널 모듈을 사용하지 않고 RTAI의 사용자모드인 LXRT 모듈을 사용하여 경성 실시간 성격보다는 연성 실시간 성격이 짙어졌다. 실제로 비실시간 태스크에서 사용한 printf() 함수가 실시간 태스크의 영향을 끼치는 것이 확인되었다. 앞으로 개선해야 할 일들은 현재의 연성 실시간으로 설정된 태스크들 중 실시간 성격이 강한 태스크들을 커널 모듈로 변환 하는 것과, 모든 태스크들 간의 통신 역시 효율성을 높일 수 있는 방안으로 바꾸는 것이다. 또한, 이 논문에서 구현한 시뮬레이터의 구조는 현재 로봇에 최적화시켜 구현했지만, 앞으로는 어떤 형태의 로봇이라도 적용시킬 수 있는 구조로 발전시켜야한다.

로봇 제어기의 구성 태스크들을 분석해보면 로봇의 제어 즉 서보모터 구동에 직접적으로 관련된 태스크와 사용자 편의만을 위한 태스크들로 나뉘어진다. 이중 실제 로봇을 제어하는데 필요 없는 요소들을 별도의 PC에서 구동하게 하여 로봇 제어기의 효율성을 극대화 시킬 필요가 있다. 특히 편집

기와 시뮬레이터는 계산량이 많기 때문에 다른 PC에서 구동할 필요가 있으며, GUI 환경이 강화되어 있는 Windows 환경에서 구축한다면, 사용자 편의가 극대화 될 것이다.

참고문헌

1. Han, S. H. and Lee, M. H., "A Study on the Performance Improvement of Industrial Robot Manipulator Controller," J. of KSPE, Vol. 7, No. 4, pp. 85-102, 1990.
2. Seo, I. H., Kim J. H., Jeong J. G. and Noh, B. O., "Development of Robot Controller with Vision function based on PC," Proc. of KSPE Autumn Conference, pp. 537-542, 1994.
3. Choe, J. T. and Kim, J. S., "Real Time Operating System and Application to a Robot Controller," J. of the KSME, Vol. 36, No. 3, pp. 231-242, 1996.
4. Lee, D. M., Oh, J. H. and Lee, J. S., "The Development of General Purpose Robot Language Based on Real Time Operating System," Proc. of IEEK, Vol. 1, pp. 18-23, 1991.
5. Chung, C. Y., Lee, G. D. and Lee, B. H., "Robot Controller Software Design and Implementation Using Real-Time Operating System," Proc. of IEEK, Vol. 1, pp. 1246-1251, 1994.
6. Jang, H., Lee, K. D. and Ahn, B. C., "A Design and Implementation of PC Based Real-Time Multitasking Kernel for Robot Controller," Proc. of IEEK, Vol. 1, pp. 459-462, 1994
7. Shin, J. H. and Moon, S. B., "The Real-Time Linux Based Robot Controller," Proc. of the KIPS, Vol. I, pp. 505-508, 2002.
8. YaGuang, K. and WenHai, W., "Design of Real Time Control Software Based On QNX," Proc. of IEEE, Digital Object Id:10.1109, pp. 579-584, 2006.
9. Miyabe, T., Konno, A., Uchiyama, M. and Yamano, M., "An Approach Toward an Automated Object Retrieval Operation with a Two-Arm Flexible Manipulator," The International J. of Robotics Research, Vol. 23, No. 3, pp. 275-291, 2004.
10. Schmidt, D. C., Deshpande, M. and O'Ryan, C., "Operating System Performance in Support of Real-Time Middleware," Proc. of the 7th International Workshop on WORDS 2002, IEEE, pp. 199-206, 2002.
11. Garrels, M., "Introduction to Linux : A Hands on Guide," Lightning Source Inc., pp. 7-16, 2008.
12. Bianchi, E., Dozio, L., Ghiringhelli, G. L. and Mantegazza, P., "Complex Control Systems, Applications of DIAPM RTAI at DIAPM," Realtime Linux Workshop, 1999.
13. Cloutier, P., Mantegazza, P., Papacharalambous, S., Soanes, I., Hughes, S. and Yaghmour, K., "DIAPM-RTAI Position paper," Real Time Peration Systems Workshop, pp. 1-28, 2000.