論 文
40～8～13

병렬 구조에 의한 가변 논리제어장치의 기능적 설계

A Functional Design of Programmable Logic Controller Based on Parallel Architecture

李 政 勳*·申 玹 植**
(Jung-Hoon Lee · Hyeon-Sig Shin)

Abstract - PLC(programmable logic controller) system is widely used for the control of factory. PLC system receives ladder diagram which is drawn by the user to implement hardware logic, converts the ladder diagram into sequence program which is executable in the PLC system, and executes the sequence program indefinitely unless user breaks. The sequence program processes the data of on/off signal, and endures 1 scan delay and missing of pulse-type signal shorter than a scan time. So, data dependency doesn't exist. By applying this characteristic to multiprocessor architecture, we design parellel PLC functionally and evaluate performance upgrade. Parallel PLC consists of central processing module, N general processing unit, and a shared memory by master-slave type. Each module executes allocated sequence program by the control of central processing module. We can expect performance upgrade by parallel processing, and reliability by relocation of sequence program when error occurs in processin g module.

1. 서 론

가변 논리제어장치(PLC ; programmable logic controller)는 공장 자동제어에 사용되는 기기로서 사용자의 래더 다이아그램을 시퀀스 프로그램(sequence program)으로 변환하고, 이 프로그램에 의해 입력을 처리한 후 제어 결과를 출력해주는 기능을 갖고 있다. 시퀀스 프로그램이란 PLC가 수행하는 프로그램으로 논리 회로를 기술하기 쉬운 언어로 작성된다[1]. PLC는 이 프로그램을 반복수행하여 제어 논리를 구현한다. 프로그램의 수행시 변화하는 디지탈 신호에 대한 처리가 PLC의 주요 목적이며, 이를 위해 같은 프로그램을 반복 수행한다. 시퀀스 프로그램을 한번 수행하는데 소요되는 시간을 스캔 시간(scan time)이라고 하는데, 이 스캔 시간을 위주로 PLC의 성능을 평가한다. 스캔 시간은 처리기 및 주변 회로의 속도와 시퀀스 프로그램의 크기에 의존한다. 같은 시퀀스 프로그램을 빨리 수행하면 할수록, 또 허용

*正 會 員 : 大宇通信 電送研究室 社員
**正 會 員 : 서울大 工大 컴퓨터工學科 助教授·工博
接受日字 : 1990年 7月 18日
1次修正 : 1991年 3月 13日
2次修正 : 1991年 7月 22日

범위 안에서 처리할수 있는 프로그램이 크면 클수록, PLC의 성능은 향상된다. 그런데 점차로 제어 범위가 확대됨에 따라 제어해야할 신호의 양이 많아지고 따라서 시퀀스 프로그램이 커지게 되어 허용 범위 안에서 한 스캔을 완수하지 못하게 되므로 처리기 및 주변회로의 속도를 증가시켜 스캔 시간을 줄이려는 노력이 진행되고 있다. 본 논문에서는 그 한 방안으로서 다중 처리기를 사용하여 PLC의 성능을 증가시키고, 처리 가능한 시퀀스 프로그램의 크기를 확대시키며 임의의 처리 모듈에 오류가 발생하더라도 전체 시스템이 시퀀스 프로그램을 계속 수행할 수 있는 결함 허용(fault tolerant)능력을 갖도록 한다[2]. 단일 처리기에서 코프로세서(co-processor) 등 속도 증진방법을 쓴다면 성능 향상 차수가 $O(k)$이다(k는 상수). 다중 처리기 구조에서는 $O(\log n)$으로 성능 향상이 이루어진다(n은 처리기 수). 그러므로 단일 처리기에 비해 상당한 성능 향상을 기대할 수 있다. 비용 면에서 보더라도 주변 회로 속도나 코프로세서에 의한 방법은 custom chip으로 설계하므로 비용이 막대하다. 다중 처리기는 이미 나온 IC로 구성할 수 있으므로 위의 방법보다 적은 비용으로 속도 향상을 기대할 수 있다.

PLC가 수행하는 시퀀스 프로그램에는 제어 의존성과 데이타의 의존성이 없어서 병렬처리에 유리하다. 프로그램을 다중 처리기 환경에서 수행할 때, 데이타의 의존성과 제어 의존성이 장애가 되는데 PLC의 래더 다이아그램과 시퀀스 프로그램은 하드웨어 논리회로를 표현한 것으로 각 줄은 독립적이기 때문에, 제어 의존성은 없다. 또한 PLC는 데이타를 처리하는데 있어서 에지 트리거(edge trigger)논리보다는 현 신호의 온/오프를 처리하고, 하나 혹은 두 스캔의 지연으로 제어 신호를 처리하여도 무방한 환경에서 사용되므로 데이타의 의존성도 없다.

이 논문의 구성은 다음과 같다. 2장에서는 PLC에 대한 설명과 동작 원리, 또 다중처리에 대해 설명하고, 3장에서는 PLC가 처리하는 데이타의 특징과 그에 있어서의 다중처리의 타당성을 제시한다. 4장에서는 다중처리 PLC의 전체 구성도를 보이고, 5장에서는 성능을 평가한다.

2. PLC와 다중처리

2.1 PLC(programmable logic controller)

공장자동제어에서 제어 논리를 구성하기 위해 제어 논리를 래더 다이아그램(ladder diagram)으로 작성하고, 이를 릴레이 배선으로 실제회로를 구성하여 제어회로를 완성한다. 그러나 제어내용의 간단한 변경이나 수정을 하는 경우 하드웨어 개조 작업이 커지는 경우가 많았다. PLC는 종래에 사용하던 제어반 내의 계전기(relay), 시간 계수(timer), 계수기(counter)의 기능을 반도체 소자와 IC등의 소자로써 대체시킨 기기이며 릴레이 배선 작업 없이 래더 다이아그램을 입력으로 받아 그의 제어 논리를 구현해준다. PLC는 1968년 미국의 General Motors사에서 각종 제어 장치가 구비해야 할 조건을 구매 사양서에 명시한 것이 그 효시라고 볼 수 있으며, 거의 모든 산업 분야의 자동화 시스템에 사용되고 있다[3]. PLC는 디지탈 회로와 마이크로 프로세서를 이용하여 제어 논리를 구현할 수 있는 능력을 갖고 있으며, 래더 다이아그램을 프로그램화하여 수정이나 변경을 쉽게 할 수 있다.

PLC는 다음과 같은 기능을 갖추고 있다.

1) 래더 다이아그램을 프로그램화하여 기억 장치에 격납하고 래더 다이아그램의 변경시에는 배선을 바꾸는 릴레이 시퀀스와는 달리, 메모리에 격납된 프로그램을 바꾼다.

2) 유접점 릴레이에 대신하여, 반도체 소자를 사용한 무접점 회로로 되어 있다.

3) 유니트화 한 것이 많아졌고, 보수나 검사가 쉽다. 기본적인 입출력 유니트는 디지탈 입출력 제어를 담당하며 그외 DA(digital-analog)제어, AD(analog-digital)제어 유니트, 컴퓨터와의 교신 유니트, PLC네트워크 유니트 등이 있다[4].

그림1은 래더 다이아그램과 시퀀스 프로그램의 예이다.

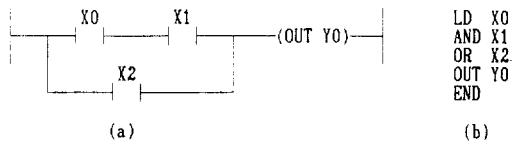

그림 1 (X0∧X1) ∨X2를 구현하는 회로(래더 다이아그램)와 그의 시퀀스 프로그램

Fig. 1 Ladder diagram and sequence program for (X0∧X1) ∨X2

그림 2 다중처리기 시스템

Fig. 2 Multiprocessor system

위의 그림에서 X0, X1X2, Y0은 PLC에 내장된 입력점과 출력점이고, LD, AND, OR, OUT 등은 PLC용 명령어이며 그림1(a)의 래더 다이아그램과 (b)의 시퀀스 프로그램은 같은 의미를 지닌다. PLC는 1(b)의 프로그램을 한 명령어씩 수행하며 무한 루프를 돌게되고, 1회 전체 프로그램을 수행하는 것을 1스캔이라고 한다. 이 스캔이 반복됨으로써 제어를 수행한다. 그리고 스캔과 스캔 사이에는 PLC의 시스템 데이타를 처리하는데 주로 고장 처리 데이타, 입출력 데이타, 그리고 경우에 따라 PLC네트워크 데이타를 처리한다. 하드웨어 논리를 프로그램화함으로써 접점이 변화는 그 접점이 포함된 명령을 수행할 때에만 검사되므로 다음과 같은 문제점이 발생할 수 있다. 하드웨어 회로에서는 X0=X2=X4=0인 상태에서(따라서 Y1=0)X4=1로 되면 Y1은 즉시 1이 되지만, 그림 1(b)의 경우 OUT Y1을 수행할 때 X4가 1로 변화한다 하여도, 즉시 1이 되지 않고 다음 스캔에서 OR X4를 수행해야 비로소 Y1이 변환한다. 즉 한 스캔의 지연을 감수하며 이를 한 스캔 지연(1 scan delay)이라고 한다. 또한 연속된 스캔중 처음 스캔에서 OR X4를 수행한 뒤 다음 스캔에서 OR X4가 수행하기 전에 X4가 잠시 온했다가 오프한 경우는 두 스캔에서 각각 OR X4를 수행할 때는 모두 X4가 오프인 상태이므로 X4의 변화는 Y0에 아무런 변화도 주지 못한다. 이를 펄스 신호의 손실이라고 한다. PLC에는 이런 문제점들이 있지만 실제 현장에서 제어용으로 사용되는데는 지장이 없다.

접점은 외부 상태의 변화이므로 접점을 접근하기 위해서는 빈번한 입출력을 하여야 한다. 그러므로 시퀀스 프로그램을 수행하는데는 많은 시간을 입출력에 소요하게 된다. 이 접점을 처리하는 방식에 의해 PLC시스템을 직접 처리(direct)방식과 리프레쉬(refresh)방식으로 나눈다. 직접 처리 방식은 접점의 내용을 입력하거나 제어 내용을 출력할 때마다 실제 입출력 장치에 대해서 입출력을 행한다. 그러므로 스캔 시간이 오래 걸리게 되나, 접점의 내용은 가장 최근의 것을 반영하게 된다. 반면 리프레쉬방식은 접점 기억장치를 따로 두어 스캔과 스캔 사이에 접점 기억장치를 그 당시의접점 내용으로 갱신하며 스캔 중에는 입출력을 하지 않고 접점 기억장치에 접근하여 제어를 행한다.

2.2 다중처리(multiprocessor)시스템의 특징

다중처리 구조는 하나 이상의 처리기들과 공유 기억장소로 이루어진 구조를 의미한다.

각 처리기간 통신은 공유 기억장치를 통해 이루어지며, 처리기는 자신의 기억장치와 공유 기억장치에 접근할 수 있다. 각 처리기들은 동등한 계산 능력을 가졌으며, 자율권을 갖고 있다. 이 시스템에서는 입출력등 대부분의 자원들이 공유되어 있고, 처리기간 동기화가 필수적이며, 시스템 전체의 단일 운영 체제에 의해 위의 기능들이 관리된다[5]. 처리기간 통신이 오로지 공유 기억장치를 통해 이루어지므로 공유 기억장치에 대한 충돌때문에 성능에 제약이 생기며 한 시스템을 구성할 수 있는 처리기 수의 한계가 된다. 이 기억장치 충돌에 의한 성능 감소를 줄이기 위해 공유 버스, 크로스바 스위치(crossbar switch), 멀티포트 기억장치(multiport memory)등의 교환 구조가 시도되었다. 이 시스템에서는 처리기들이 서로 협력하여 동시에 한 작업을 수행하므로 처리량(throughput)이 증가하고, 하나의 처리기라도 동작하는 한 전체 시스템은 계속 동작할 수 있기 때문에 시스템의 고장 발생률이 감소한다[6]. 그러므로 다중 처리 시스템은 단일 처리기의 능력을 벗어난 실시간 제약조건을 요구하는 응용과 고도의 신뢰성을 요구하는 응용에 적합하다. 특히 마이크로 프로세서들로 구성된 다중처리기 시스템은 각 마이크로 프로세서들이 기능이 비슷하고 독립적인 수행을 하며, 상호 데이타 교환이 적은 경우에 효과적이다.

3. PLC의 병렬화 분석

3.1 병렬화 가능성

순차적인 프로그램을 다중처리기로 수행하는 경우, 데이타의 의존성과 제어 의존성이 문제가 된다[7, 8]. 래더 다이아그램은 원래 순서가 없는 하드웨어 논리회로이며, PLC가 등장하면서 래더 다이아그램을 시퀀스 프로그램으로 변환하여 순차적으로 수행하게 되었다. 제어 의존성은 프로그램내에서 제어의 순차적인 흐름이 존재하는 경우에 발생한다. 래더 다이아그램은 하드웨어 논리회로 이므로 회로에서는 제어의 흐름이 있을 수 없다. PLC는 래더 다이아그램을 시퀀스 프로그램으로 바꾸어 순차적으로 한 문장씩 수행한다. 그러나

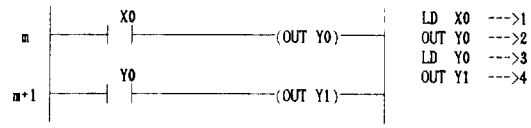

그림 3 래더 다이아그램에서의 데이타 의존성
Fig. 3 Data dependency in ladder diagram

같은 프로그램을 반복하여 수행하므로 줄과 줄 사이에 순서를 둘 수 없다. 그림3의 래더 다이아그램에서 줄 m은 LD X0, OUT Y0로 줄 $m+1$은LD Y0, OUT Y1으로 변환되면 PLC는 4문장을 반복하여 수행한다. 그러므로 한 스캔 내에서는 1), 2) 문장이 3), 4)문장보다 먼저 수행되지만 2)문장이 수행 완료된 시점에서는 3), 4)문장이 1), 2)문장보다 먼저 수행된다. 그러므로 래더 다이아그램의 m, $m+1$은 순서를 둘 수 없으며 줄들 간에는 제어 의존성이 없다.

그림3은 입력점 X0가 온했을 때 출력점 Y0와 Y1이 모두 온하는 래더 다이아그램이며, 출력점 Y0가 줄 m의 출력, 줄 $m+1$의 입력으로 사용되었다. 즉 출력점 Y0에 대해 데이타의 의존성이 존재한다. 그러나 반드시 Y0와 Y1이 동시에 온할 필요가 없고, PLC에서는 1스캔의 시간을 감수하므로 Y1은 다음 스캔에서 온해도 무방하다. 라프레쉬 방식에서 줄 m과 줄 $m+1$이 각각 다른 처리기에 할당되어 수행될 경우, X0가 온한것을 탐지한 첫번째 스캔에서 Y0가 온하고, 스캔 사이에서 그 처리가 이루어진 후 최소한 다음 스캔에서는 Y0에 의해 Y1이 온한다. 그러므로 줄 m, $m+1$에서 Y0의 데이타 의존성은 무시하여도 된다. 결국 줄 m과 줄 $m+1$은 서로 다른 처리기에 의해 동시 수행될 수 있다.

3.2 PLC의 명령어

PLC명령어는 크게 접점 명령과 응용 명령으로 나눌 수 있다. 접점 명령은 접점이나 접접의 내용을 갖고 있는 기억 장치로부터 그 온/오프 내용을 읽어 내부 레지스터에 저장하거나 현재 레지스터의 값과 비트 연산을 수행하거나 계산된 내용을 출력시키는 명령들이다. 예를 들어 LD는 특정 접점의 내용을 읽어와 저장시키는 명령이며, AND는 접점의 내용을 읽어 논리곱을 수행하며, OUT는 현재까지 계산된 내용에 의해 접점을 갱신한다. 그외, 현재의 중간 결과를 push하거나 pop하는 명령도 있다. LD, OUT, OR, AND등의 PLC 명령들은 프로세서의 명령어 집합과는 다르다. 그러므로 시퀀스 명령에 대한 십육진 값을 미리 저장한 뒤 프로세서는 이를 읽어와서 해석을 하고 그 명령에 대한 연산자로써 접접을 사용하게 된다. 그러므로 PLC의 명령어를 수행하기 위해서는 명령어 훳치를 위한 자신의 스캔 기억장치 참조와 이를 이어 연산자의 접점의 내용을 읽기 위한 공유 접점기억장치 참조가 필요하다.

응용 명령은 PLC의 기능이 확대됨에 의하여 생겨난 명령으로 데이타의 여러 바이트 이동, 비교, 또 다른 PLC와의 통신 명령 등 그 종류가 다양하며, 이들은 어떤 처리기로 구현되든지, 그 처리기의 어셈블리 명령어로 구현되어야 하므로 응용 명령을 수행하는데는 접점 명령에 비해 무척 많은 시간이 소요된다. 응용 명령 중에도 접점의 정보를 필요로 하는 경우가 있지만 그 시간은 극히 적다. 접점기억장치를 공유 기억장치로 할 경우 응용 명령이 많을수록 공유 기억장치에 대한 충돌이 적어지므로 다중처리에 유리한다.

접점 명령 OUT와 응용 명령은 출력 명령에 속하며, 나머지는 모두 입력 명령에 속한다. 래더 다이아그램에서 한 줄(line)은 시퀀스 프로그램의 입력 명령군과 출력 명령으로 이루어진다. 입력 명령문군에서는 LD, AND, OR 등의 비트 연산을 계속 수행하며, 그 최종 결과가 출력 명령을 수행할지를 결정한다. 즉 최종 결과가 온상태이면 출력 명령을 수행하고 오프 상태이면 출력 명령을 수행하지 않는다. 그러므로 수행할 당시의 신호에 크게 의존하여 스캔 시간을 예측하는 것이 어렵고, 효율적인 프로그램 분할이 어렵다. 한 줄 안에서 입력문들은 직전 명령의 결과를 토대로 자신의 동작을 처리하며, 이의 결과가 출력문의 수행 여부를 결정하므로 한 줄에서는 그 데이타들이 밀접한 의존성을 갖는다. 다중 처리를 위해 시퀀스 프로그램을 분할할 경우 래더 다이아그램의 한 줄에 해당하는 시퀀스 프로그램의 명령들은 반드시 한 처리기에 할당되어야 한다.

3.3 PLC의 데이타

기본적으로 PLC는 온/오프의 신호를 다룬다. PLC는 최소한 X(입력점), Y(출력점), M(메모리 접점)을 지원해야 한다. X, Y접점은 제어용 배선에 사용되는 접점으로 사용자는 X접점으로 신호의 입력을 할 수 있고, Y접점으로 제어선을 연결할 수 있다. 또한 시퀀스 프로그램 작성시의 도움을 위하여 임시 기억장치로 M접점을 사용할 수 있으며, 그 외 시간 계수용 접점 T, 일반 계수용 접점 C를 사용할 수 있다. 이중 X접점은 입력 명령의 연산자로만 사용될 수 있으며, 나머지는 입력 명령과 OUT명령의 연산자로 사용될 수 있다. 그리고 PLC기능의 확장에 따라 다양한 의미를 갖는 접점들이 나타나게 되었다.

4. PPLC(parallel PLC)의 구조

중앙 처리기와 N개의 시퀀스 프로그램 처리 모

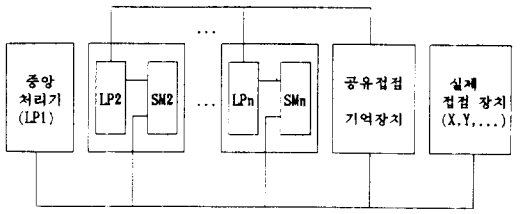

그림 4 전체 시스템의 연결도
Fig. 4 Architecture of the system

듈로 구성된 마스터 슬레이브(master-slave)구조이며, 공유 접점기억장치를 두어 리프레쉬 방식으로 접점의 내용을 스캔사이에 갱신하고 N개의 처리 모듈로 하여금 접점 처리시 입출력 대신 이 기억장치에 접근하여 시퀀스 명령을 수행하도록 한다. 중앙 처리기가 필요한 이유는 로더와의 통신, 시퀀스 프로그램의 분배, 공유 접점기억장치의 갱신 등 총괄적인 기능이 필수적이며, 이 기능들이 여러 처리 모듈에서 수행되기에는 오버헤드가 심하기 때문이다. 중앙 처리기, N개의 처리 모듈, 공유 접점기억장치의 전체적인 연결은 다음과 같다.

중앙 처리기가 공유 접점기억장치, 각 처리 모듈의 스캔 기억장치에 대한 통제권을 갖고 있으며, 각 처리 모듈은 중앙 처리기가 스캔 시작 명령을 내린 경우만 자신의 스캔 기억장치와 공유 접점기억장치에 접근할 수 있다. 스캔 중에는 중앙 처리기도 LP1으로 동작하여 자신에게 할당된 시퀀스 프로그램을 수행한다. 각각의 처리 모듈은 자신에게 할당된 시퀀스 프로그램을 수행하며 처리 모듈간 통신은 오로지 접점의 상태를 변화시킴으로써 이루어진다.

4.1 중앙 처리기

클락 발생기, 처리 모듈, 인터럽트 제어기(interrupt controller), 시간 계수기(timer controller), 기억장치, 고장 검출 회로, 스위치 검사 회로 등으로 구성되어 있으며, 전체 시스템의 통제를 담당한다.

(1) 처리 모듈

처리 모듈은 처리기와 명령어 내장 ROM으로 구성된다. 처리기의 명령어 집합과 PLC의 명령어 집합이 다르므로, PLC의 명령어를 구현하기 위한 처리기 자체의 명령어로 구성된 PLC 명령어 구현 프로그램이 ROM에 들어 있다. 또한 부팅시의 초기화와 검사프로그램, 스위치 검사와 로더로부터 시퀀스 프로그램을 받아 각 기억 장치로 분배하는 프로그램, 스캔 사이에 시스템 데이타를 갱신하거나 스캔중의 각종 고장에 대한 검사 프로그램이 내장된다.

(2) 기억 장치

시스템 변수 영역, 로더와 PLC, 중앙 처리기와 각 처리 모듈과의 통신 버퍼영역이다.

(3) 시간 계수기

PLC에는 일정 시간을 계수하여 동작하는 명령들이 있다. OUT $Ti(k=3)$는 계수 코일이 온하고 3*(정해진 시간 단위)후에 계수 접점을 온시키라는 명령인데, 이를 위해 시간을 계수할 필요가 있다. 또한 시스템이 무한 루프에 걸리는 일이 없도록, 검사 계수기(watchdog timer)의 역할을 한다.

(4) 인터럽트 제어기

계수기, 로더와의 통신, N개의 처리 모듈의 동기 등 시스템 전체의 비동기적인 일들을 처리한다.

4.2 공유 접점기억장치

입력(X), 출력(Y), 메모리(M)접점, 계수 및 시간계수 설정치, 현재값, 코일, 그리고 시퀀스 프로그램 작성용 각종 레지스터 및 접점의 내용을 갖고 있는 공유 기억장치이다. 중앙 처리기가 스캔 사이에 갱신하며, 입력점의 경우 실제 입력점으로부터 접점의 상태를 읽어와 그 내용에 의해 공유 접점기억장치를 갱신하고, 출력점의 경우 이 공유 접점기억장치의 내용이 실제 출력점에 반영된다. 스캔 사이에는 중앙 처리기만 접근할 수 있으며 스캔 중에는 각 처리 모듈이 실제 입출력점 대신 이 공유 접점기억장치에 접근한다. 또는 처리 모듈간의 통신용으로도 사용될 수 있다.

4.3 N개의 처리 모듈

LPi는 처리기이며, ROM은 PLC명령어를 구현

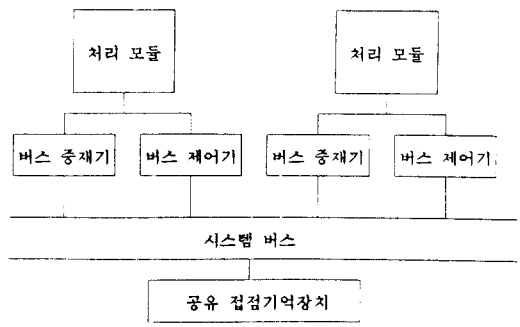

그림 5 공유 접점기억장치
Fig. 5 Shared contact memory

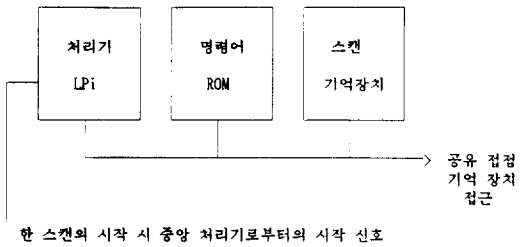

그림 6 처리 모듈 i의 연결도
Fig. 6 Organization of processing element

하기 위해 LPi의 명령어로 구성되어 있으며, 스캔 기억장치는 이 i번째 모듈에 할당된 시퀀스 프로그램을 지니고 있다. LPi는 중앙 처리기로부터 스캔 시작 신호를 받으면 자신의 스캔 기억장치에 있는 시퀀스 프로그램의 수행을 시작한다. 중앙처리기는 사용자가 시퀀스 프로그램을 작성 혹은 변경할 때마다 로더의 분할에 의해 SMi에 프로그램을 적재한다. 그리고 한 스캔을 시작할 때마다 LPi에게 스캔 시작 신호를 보낸다. LPi는 SMi에서 시퀀스 명령을 읽어와 수행하며, 접점 명령인 경우 공유 접점기억장치에 접근한다. LPi가 자신에게 할당된 시퀀스 프로그램 수행을 끝내고 중앙처리기에 보고하면, 중앙 처리기는 각 스캔 사이마다 공유 접점 기억장치를 실제 접점 내용으로 갱신해준다. SMi는 중앙 처리기와 LPi만이 접근할 수 있으며, LPi는 스캔 중에 중앙 처리기는 스캔 사이에만 접근할 수 있다. 또 SMi에는 한 스캔의 종료시 중앙 처리기에게 스캔에 대한 보고를 하기위한 영역이 있다. 중앙 처리기는 스캔 종료시 모든 SMi를 읽어 직전 스캔에 관한 정보를 얻을 수 있다.

4.4 스캔 기억 장치에 대한 충돌 해결(arbiter)

SMi에 접근할 수 있는 처리기는 LPi와 중앙 처리기이다. 처리 모듈 간의 통신은 공유 기억장치에 의한다. LPi와 중앙 처리기는 서로 SMi에 접근하는 시간이 다르다. LPi는 시퀀스 프로그램 수행 시에만 SMi에 접근할 수 있다. 그러므로 중앙처리기가 SMi에 대한 전체 통제권을 갖고, 시퀀스 프로그램 수행 시에만 LPi로 하여금 SMi를 접근하도록 한다.

4.5 중앙 처리기와 N개의 처리 모듈과의 상호 접속

중앙 처리기와 각 처리모듈간의 동기화 과정이다. 이들간의 동기는 스캔의 시작과 종료에서 필요하다. 스캔의 종료가 명확해야만 중앙 처리기가 공유 접점기억장치의 내용을 실제 입출력점에 반영할 수 있다. 중앙 처리기와 각 처리모듈간에 특정한 어드레스 라인에 의한 인터럽트로 스캔의 시작/종료를 각각 알릴 수 있다. 중앙 처리기는 LPi에게 스캔 시작 신호로 시퀀스 프로그램의 수행 시작을 알리며, LPi는 시퀀스 프로그램이 끝나거나, 혹은 고장 발생시에 중앙 처리기에게 알린다. 중앙 처리기는 모든 LPi로부터 이 신호를 받았을 때 한 스캔의 종료를 감지하고 각 SMi의 특정 영역을 읽어 LPi에서의 한 스캔에 대한 보고를 받는다.

4.6 로더의 기능

로더는 기존의 래더 다이아그램 입력과 중앙 처리기로의 시퀀스 프로그램 전달의 기본 기능외에 분배의 기능을 추가한다. 각 줄 간의 구별은 ㅏ기호에 의한다. 각 ㄱㅏ를 하나의 노드로 삼아 노드간의 연결을 탐지하고 연결의 AND, OR등의 관계를 판단한다. 노드 간의 관계를 따라가면서 래더 다이아그램 한 줄씩 그림1(b)와 같은 형태로 변환한다. 한 줄 단위로 변환된 이진 코드는 하나의 처리 모듈에 할당된다. 그러므로 각 줄의 이진 코드 갯수에 따라 편차가 심하게 처리 모듈에 분배될 수 있다. 이는 가장 늦게 스캔 완료한 처리 모듈에 의해 결정되는 스캔 시간에 영향을 주어 시스템의 성능을 감소시킨다. 그러므로 각 줄의 코드 수를 미리 계산하여 놓고 각 처리 모듈에 분배할 때 knapsack 문제를 적용시키면 보다 균등한 분배를 할 수 있다[9]. 로더와 중앙 처리기와의 통신은 자주 발생하는 것이 아니고 또 경우에 따라서는 통신 중에도 제어가 계속 수행되어야 하기 때문에 높은 전송 속도를 요하지는 않으므로 직렬 포트를 이용한 통신을 한다.

4.7 PPLC의 동작

PPLC의 상태에는 일반 PLC처럼 프로그램 모드와 수행의 모드가 있다. 사용자가 그 모드를 결정하며, 프로그램 모드에서는 시퀀스 프로그램을 각 스캔 기억장치에 적재하고, 프로그램 수행모드에서는 연속된 스캔으로 사용자가 프로그램 모드로 바꿀 때까지, 시퀀스 프로그램을 반복 수행한다.

사용자가 시퀀스 프로그램을 변경하면 중앙 처리기는 로더로부터 시퀀스 프로그램을 받아 각 스캔 기억장소로 적재한다. 로더는 사용자로부터 시퀀스 프로그램을 입력받는데, 이 로더에서 시퀀스

프로그램을 분배한다. 중앙 처리기는 로더와 각 처리 모듈의 통로가 된다. 시퀀스 프로그램 모드는 스캔 시간과 스캔 사이 시간의 연속인데 각 처리 모듈에서 자신에게 할당된 시퀀스 프로그램 수행을 완료하면, 중앙 처리기에 보고하고, 장벽(barrier)에 도착하여 중앙 처리기로부터 다음 스캔 시작 신호를 기다리게 된다. 장벽은 새로운 동기를 받을 때까지 수행을 중지하고 있는 상태이다 [10]. PPLC는 전체적인 제어를 행하는 중앙 처리기, 각 처리 모듈, 이들 간의 상호 접속, 공유 접점기억장치 등으로 구성되며 접점의 내용을 읽거나 갱신할 경우 공유 접점기억장치를 접근하며, 현재 다른 처리 모듈이 접근호 있는 경우는 제지된(blocked)상태로 대기한다. PPLC에서의 한 스캔은 다음과 같다.

(시퀀스 프로그램이 이미 적재되어 있다고 가정)

1. 중앙 처리가 각 처리 모듈로 스캔 시작 신호 발생
   (이후 중앙 처리기는 LP1으로 동작)
2. N개의 처리 모듈은 자신에게 할당된 시퀀스 프로그램 수행
   while(스캔)
   case1) 입출력점을 연산자로 하는 경우
   · 공유 접점기억장치 접근
   · 제지된 상태(충돌 발생시)
   case2) 응용 명령을 수행하는 경우
   자신의 ROM기억장치를 접근
   /*공유 기억장치에 대한 요구 없음*/
   LP1은 고장 검사, 시간 계수, 로더와의 통신 등도 수행
   case3) 할당된 프로그램 수행 완료
   장벽에 도달하여 중앙 처리기에 완료 보고
3. 한 스캔의 종료시(스캔 사이)
   /*N개의 처리 모듈이 모두 수행 완료*/
   /*중앙 처리기의 동작*/
   시스템 변수들의 갱신
   공유 접점 기억 장치의 내용을 실제 출력 접점에 반영
   실제 입력 장치의 온/오프 상태를 공유 접점 기억장치로 적재

스캔 시간 감소의 반면, 래더 다이아그램을 시퀀스 프로그램으로 변환하는 과정에서 분할을 하는 문제, 처리기 수의 증가에 따라 공유 접점 기억 장소에 대한 충돌 해결에의 복잡도와 비용, 중앙 처리기와 각 처리 모듈과의 통신 과부하 등이 병렬 처리에 의한 단점이다.

5. 성능 평가

하나 이상의 처리기들이 공유 접점 기억 장치를 접근하여 접점의 정보를 얻고자 할때 충돌이 생겨 성능 향상의 감소를 초래한다. 그러나 데이타의 의존성과 프로그램의 순서에 의한 제어 의존성은 고려하지 않아도 된다. 여러 처리기들이 대부분 공유 접점기억장치를 접근해야 하는 접점 명령을 수행하고자 할 경우는 충돌에 의한 성능의 감소를 초래하며, 응용 명령을 자주 수행하게 되는 경우는 충돌이 적어 다중 처리기의 효율을 증대시킬 수 있다. 본 논문에서는 접점 기억장치에 대한 메모리 사이클을 단위 시간으로 하여 시뮬레이션에 의해 한 스캔 시간을 측정한다. PPLC에서의 한 스캔 시간은 각 처리 모듈 중 가장 오래 걸린 처리 모듈에 의해 결정된다. 본 시뮬레션에서 시퀀스 명령과 응용 명령의 발생 비율에 대한 성능 향상 비율과, 기존의 공정제어용으로 사용되는 시퀀스 프로그램의 명령 통계를 기초로 하여 처리기 증가에 대한 성능 향상 비율을 측정한다.

시뮬레이션에서의 가정은 다음과 같다.

1. 래더 다이아그램 한 줄별로 각 처리기에 할당된다. 한 줄의 길이는 고려하지 않고 전체 프로그램의 줄 수를 똑같이 나누어 각 처리기에 할당한다.

2. 공유 접점기억장소에 대한 접근 시간과 내부 비트 연산 시간까지의 합을 한 단위 사이클로 가정한다. 각 처리 모듈의 상태는 크게 공유 접점기억장치 접근과 자신의 스캔 기억장치 접근으로 나누어 볼 수 있지만, 이 시뮬레이션에서는 둘다 단위 시간에 수행되는 것으로 가정한다.

3. 각 처리기는 입력 명령 훳치, 입력점을 위한 공유 접점기억장치 접근, 출력 명령 훳치, 출력점을 위한 공유 접점기억장치 접근, 응용 명령 수행, 공유 접점기억장소에 대한 접근이 제지된 상태(blocked), 수행 완료의 상태에 있다.

4. 응용 명령은 공유 접점 기억 장소에 대한 요구가 없다. 응용 명령은 수행시간이 무척 길며, 간혹 접점 내용을 요구해야하는 경우도 있지만 그 수행 시간에 비하면 무척 짧다. 그러므로 본 시뮬레이션에서는 응용 명령 수행의 경우, 20단위 시간 만큼 자신의 스캔 기억장소 접근으로 완수된다고 가정한다.

성능 향상 비율은 단일 처리기와 다중 처리기

표 1 처리기 수와 성능 향상 비율

Table 1 Number of processors and performance upgrade ratio

2	1.91	1.93
3	2.79	2.81
4	3.50	3.64
5	4.21	4.43
6	4.75	5.17
7	4.99	5.87
8	5.15	6.53
9	5.26	7.16
10	5.34	7.76

시스템에서의 한 스캔 주기(한 스캔+한 스캔 사이 시간)의 비로 한다. 제철 공장 제어에서 사용되는 시퀀스 프로그램을 기준으로 하여 명령어의 발생 비율을 산출하여, 이 명령어의 발생 비율에 근거한 20000줄의 시퀀스 프로그램을 N개의 처리 모듈에 나누어 수행한 결과 성능 향상 비와 그래프는 표1과 그림8과 같다. 이 그림에서 처리기의 수가 2일 때 한 처리기당의 효율(성능향상 비와 처리기 수의비)이 가장 크며, 처리기의 수가 6이상일 때는 처리기 수의 증가가 성능 향상에 도움을 주지 못함을 나타내고 있다. 이 결과를 Minsky conjecture에 의해 보강하면 다음 식과 같다[11].

$$A=4.65\log n+0.97$$

A : 성능 향상 계수

n : 처리기 수

성능 향상의 한계를 추정해보기 위해 단일 처리

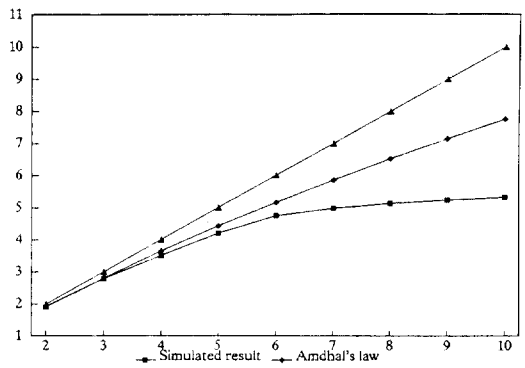

그림 7 처리기 수의 증가에 대한 성능 향상 계수

Fig. 7 Performance upgrade according to the number of processors

표 2 응용 명령비와 성능 향상 비율

Table 2 Ratio of application instruction and performance upgrade

응용 명령 비율	처 리 기 수			
	2	3	4	5
5	1.90	2.36	2.48	2.47
10	1.91	2.56	2.93	3.06
15	1.91	2.66	3.27	3.63
20	1.91	2.72	3.46	4.16
25	1.91	2.75	3.51	4.26
30	1.91	2.79	3.64	4.41
35	1.91	2.79	3.65	4.50
40	1.91	2.83	3.72	4.53
45	1.91	2.83	3.73	4.61
50	1.92	2.84	3.74	4.61
55	1.91	2.84	3.76	4.63
60	1.93	2.85	3.80	4.63

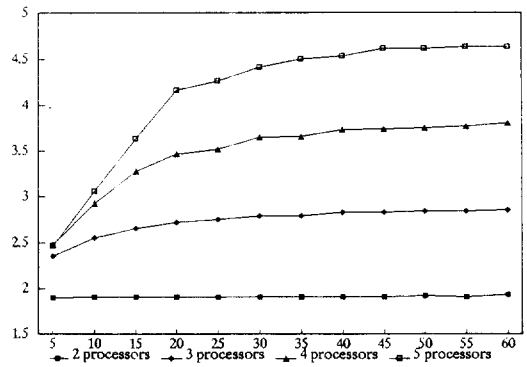

그림 8 응용 명령 발생 비율에 의한 성능 향상 비율

Fig. 8 Performance upgrade according to the application instruction ratio

기 환경에서 한 스캔의 병렬 수행 부분과 직렬 수행 부분으로 나누어 그 비율을 계산하면 Amdahl의 법칙과 비교할 수 있다[3]. Amdhal의 법칙은 프로그램 내에서 병렬 가능 부분과 불가능 부분이 존재한다는 가정하에 프로그램에서 두 부분의 비율을 고려하여 어떤 응용에서 병렬처리에 의해 향상될 수 있는 최대한도를 지정한다. 직렬 수행 부분은 병렬 수행이 불가능한 부분으로, PLC에서는 스캔 사이 시간과 스캔 시간 중 시간 계수를 처리하는 부분에 해당되고, 병렬 수행 부분은 시퀀스 프로그램을 수행하는 스캔 시간에 해당된다. 이미

사용되고 있는 단일 처리기 PLC 시스템은 대부분 시간 계수 처리를 위해, 10ms마다 시간 계수 처리가 이루어지며, 이를 위한 해당 처리기의 프로그램은 120줄 정도이며 약 360번 정도의 기억 장소 접근을 행한다. 이 시간은 약 0.3ms이며, 결국 10 ms마다 0.3ms의 시간이 시간계수를 위해 소요된다. 또 스캔 사이 시간에 처리하는 명령어의 수는 720줄 정도로 위 가정에서 각 줄당 평균 2개의 위 사이클에 해당된다. 20000줄의 시퀀스 프로그램이 PLC에서 수행될 경우 0.6% 정도이며, 결국 직렬 처리 부분은 3.6%이다.

$$A=\frac{1}{S+P/N}$$

S : 직렬 처리 부분 (serial processing)(0.964)

P : 병렬 처리 부분 (parallel processing)(0.036)

N : 처리기 수

A : 성능 향상 계수

그림8에 이 한계와의 비교도 나타나 있다. 그림8과 표2는 처리기 수가 2, 3, 4, 5일때 응용 명령 발생 비율에 대한 성능 향상 비율을 보여준다. 응용 명령의 발생 확률이 60%이상되면 N처리기에서 수행시 성능 향상 계수가 거의 N에 가까와 짐을 보여 준다. 실제 사용되는 시퀀스 프로그램에서 응용 명령의 비율은 25%정도이다.

6. 결 론

PLC를 마스트-슬레이브 다중 처리기로 설계하여 스캔 시간의 감소를 기하였다. PLC는 데이타의 의존성, 제어 의존성이 없으므로 다중 처리기 시스템에서 좋은 성능을 보인다. 각 처리기에서 최대 수행 시간에 의해 한 스캔 시간이 결정되므로 시퀀스 프로그램을 균등하게 분할하는 방식이 필요하다. 시퀀스 프로그램의 수행 시간을 스캔 전에 예측하는 것은 어렵지만, 기대값에 의한 예측을 하면 좀더 균등한 분할이 될 것이다. PPLC에서 응용 명령의 비율이 클수록 공유 기억장치에 대한 충돌이 적어 좋은 효율을 보인다. 시퀀스 명령을 대부분 사용하는 소량 제어 프로그램에서는 성능 향상을 기대할 수 없지만 복잡한 응용 명령을 사용해야하는 큰 제어 프로그램에서는 성능향상을 기대할 수 있다. 현재 사용되는 제어용 시퀀스 프로그램의 통계에 의하면 응용 명령의 발생비율은 출력 명령의 25% 정도이며, 처리기의 수가 2~6개 사이이면 처리기의 수 증가에 따른 성능의 향상을 기대할 수 있다. 임의의 처리 모듈에 고장 발생시 시퀀스 프로그램의 재분배에 의해 정상 동작을 할 수 있지만 중앙처리기에 고장이 발생한 경우는 전체 시스템이 동작할 수 없다. 로더를 시스템 버스에 연결하여 중앙 처리기에 고장 발생시 다른 처리 모듈이 중앙 처리기로 동작하게 한다면 중앙 처리기에 대한 취약성은 감소할 것이다.

참 고 문 헌

[1] Den Otter, *Programmable Logic Controllers : Operation, Interfacing, Programming,* Prentice Hall, pp. 14~58, 1988.

[2] Daniel P. Siewiorek, "Architecture of fault tolerant computers," *IEEE Computer,* pp. 9~18, Aug. 1984.

[3] 이성원, 고대식, 제어용 컴퓨터, 도서 산업사, pp. 268~269, 1986.

[4] Ian G. Warnock, *Programmable Controllers : Operation and Application,* Prentice Hall, pp. 201~271, 1988.

[5] Eli T. Fathi, Moshe Krieger, "Multiple microprocessor systems; what, why, and when," *IEEE Computer,* pp. 23~32, Mar. 1983.

[6] Veljko M. Multinovic, *Computer Architecture,* North-Holland, 1986.

[7] G.S. Almasi, A. Gottlieb, *Highly Parallel Computing,* The Benjamin Publishing Company, pp. 218~219, 1989.

[8] D.A. Pauda, "High speed multiprocessor and compliation techniques," *IEEE Trans. on Computers,* Vol. C-29, Sep. 1980.

[9] Ellis Horowitz, Satraj Sahni, *Fundamentals of Computer Algorithms,* Computer Science, pp. 157~159, 1978.

[10] Rajiv Gupta, Michael Epstein, "Achieving low cost synchronization in a multiprocessor system," *Parallel Architectures and Language Europe,* pp. 70~84, Jun. 1989.

[11] Kai Hwang, Faye A. Briggs, *Computer Architecture and Parallel Processing,* McGraw Hill, pp. 27~29, 1984.

[12] G.M. Amdahl, "Validity of the single processor approach to achieving large scale computing capablities," *AFIPS Conference Proceedings* Vol. 30, 1967.