병렬 처리 기법을 이용한 프로그래머블 로직 컨트롤러의 입출력 접점 관리를 위한 컨피규레이션 시스템 구현 알고리즘

김 광 진, 권 욱 현
서울 대학교, 전기 공학부

Configuration System Implementation Algorithm to Manage the I/O Device of the Parallel Processing Programmable Logic Controller

Kwang-Jin Kim, Wook-Hyun Kwon
Seoul National University, School of Electrical Engineering

Abstract - In this paper, an algorithm to make a configuration system for managing the I/O device of programmable logic controller(PLC) is proposed. Parallel processing architecture is used to deal with a number of I/O devices. From that architecture, a contention problem between processors can arise. To resolve this problem, the configuration system that contains informations about I/O devices is introduced. This configuration system is used to check the contention between processors in the I/O device and also used in program execution.

1. 서 론

일반적으로 제어 시스템은 센서, 제어기, 구동장치 등으로 구성되며 제어 신호의 형태에 따라 아날로그 제어, 디지털 제어, 순차 제어등으로 구분된다. 프로그래머블 로직 컨트롤러는 순차 제어를 담당하는 제어 장치로 일반적으로 '참', '거짓'으로 표현되는 입력값을 받아서 논리적 조합을 거친 뒤 이를 외부의 출력으로 내보내는 구조로 되어 있다[1],[2]. 규모나 복잡도에 따라서 그 구현 방식이 차이가 있으나 복잡하고 커다란 시스템의 제어를 위해서는 응용 프로그램을 여러개의 작은 프로그램으로 분할하고 이를 나누어서 처리하는 병렬 처리 기법을 사용할 수 있다. 그러나 병렬 처리 기법을 사용할 경우 동일한 출력 접점에 대해서 서로 다른 프로세서가 출력을 내보낼 경우에는 접점상에서 충돌하는 문제가 발생하게 된다. 이를 해결하기 위해서 입출력 접점을 미리 각 프로세서에 할당하고 각 프로세서는 담당하는 입출력 접점만을 관리하는 방식을 생각할 수도 있으나 사용자에 의해서 프로그램이 가능해야 하는 프로그래머블 로직 컨트롤러의 특성상 미리 모든 접점을 프로세서에 할당하는 방식은 시스템에 따라서 효율이 떨어지게 될 가능성이 있다. 따라서 각 입출력 접점은 사용자에 의해서 동적인 할당이 가능하면서도 프로세서간의 충돌은 방지하는 방법이 필요하게 된다. 이를 위해서, 각 입출력 접점에 대해서 읽기 권한은 모든 프로세서에게 부여하고, 쓰기에 대한 권한은 하나의 프로세서에만 할당하는 방식으로 입출력 접점을 관리하는 알고리즘을 사용하고 실제로 각 프로세서에서 사용하는 입출력 접점은 사용자의 프로그램에 의해 할당하게 한다. 이러한 알고리즘을 사용할 경우의 문제점은, 사용자가 프로그램을 잘못 작성할 경우 위에서 제기한 충돌문제가 발생할 수 있다는 점이다. 이 문제를 방지하기 위해서 사용자의 프로그램을 미리 분석하고 각 입출력 접점에 대한 다양한 정보를 추출하여 컨피규레이션 시스템을 구현하고 이를 이용하여 충돌을 미리 검사한다. 이 컨피규레이션 시스템에는 각 입출력 접점의 사용 여부, 스캔 주기, 사용 모드(읽기/쓰기) 등의 정보를 저장하게 되는데 충돌 검사만 하고 끝내지 않고 이를 컨피규레이션 시스템으로 구현하는 이유는 프로그래머블 로직 컨트롤러의 특성상, 입출력 접점의 정보가 매우 중요하기 때문이다. 컨피규레이션 테이블을 사용하여서 운영체제는 입출력 접점의 갱신, 스케쥴링등에 사용하게 되고 결국 시스템의 운영에도 도움을 주게 된다. 전체적인 흐름을 요약하면 컨피퓨레이션 시스템은 컴파일시에 구성이 되며 각 입출력 접점이 충돌 검사에 합격을 하면 실행코드와 함께 시스템에 다운로드가 되고 이를 이용하여 운영체제가 각 입출력 접점의 갱신과 스케쥴링 등에 대한 정보를 얻어서 프로그램을 실행하게 된다.

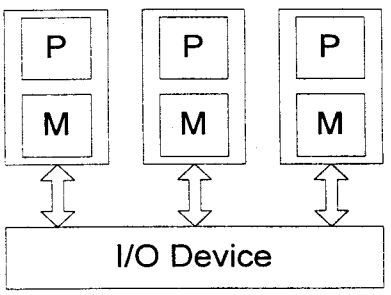
본 논문에서는 먼저 실제로 구현된 프로그래머블 로직 컨트롤러의 구조에 대해서 살펴본 뒤 이 시스템 상에서 생기는 문제점을 설명한다. 뒤에 이를 해결하기 위한 컨피규레이션 시스템과 적용 결과를 보이고 결론을 맺는다.

2. 본 론

2.1 시스템 구조 및 문제점

2.1.1 시스템 구조

프로그래머블 로직 컨트롤러는 목적이나 대상에 따라서 다양한 방법으로 구현이 가능하다. 본 논문에서 예로 드는 프로그래머블 로직 컨트롤러는 주로 크고 복잡한 플랜트를 제어하기 위해서 입출력 접점이 12,000점 정도 되고 다수의 프로세서를 사용해서 병렬 처리기법을 사용하는 대용량 시스템이다. 그림 1이 대략적인 시스템의 구조를 나타낸다.



P : 프 로 세 서
M : 지역 메모리
I/O : 입출력 접점

그림 1 전체 시스템 블록 다이어그램

위의 그림에서 보듯이 세 개의 프로세서에 의해서 프로그램이 실행이 될 수 있으며 이중 하나가 전체를 관리하는 마스터/슬레이브 구조로 되어 있다. 각 프로세서는 자신의 프로그램을 실행할 수 있는 메모리를 갖고 있으며 외부에는 입출력 접점이 존재한다. 프로그램의 실행

순서는 외부의 입력 접점에서 필요한 데이터를 읽어온 후 각 프로세서에서 필요한 논리적인 연산을 하고 이를 다시 출력 접점으로 내보내게 되는데 보통의 프로그래머블 로직 컨트롤러는 이러한 과정을 한 주기로 해서 계속 주기적으로 이러한 과정을 수행하게 된다. 따라서 시스템의 외부에서 바라볼 때는 입출력 접점이 각각 주어진 주기에 따라서 일정하게 변하는 시스템이 프로그래머블 로직 컨트롤러라고 볼 수 있고 이러한 성질때문에 입출력 접점에 대한 정보가 중요하다는 것을 알 수 있다. 각 프로세서에서 수행되는 제어 프로그램은 여러개의 태스크 형태로 나뉘어져서 각 태스크마다 독립적으로 스캔 주기와 우선 순위를 갖게 되며 이들의 스캔주기와 우선 순위가 바로 그 태스크에서 사용되는 입출력 접점들의 스캔주기와 우선 순위가 되는 것이다. 이러한 정보는 태스크를 구성하는 입력 프로그램에는 나타나지 않으며 제어 시스템을 작성하는 프로그래머에 의해서 컴파일하기 전에 할당이 된다.

2.1.2 제안된 구조의 문제점

위에서 제안된 구조는 일반적인 멀티 프로세싱 컴퓨터에서 발생하는 캐쉬 데이터의 일관성 문제(Cache Coherence Problem)와 비슷한 문제를 갖게 된다[3],[4]. 멀티 프로세싱 컴퓨터의 경우 각 프로세서마다 캐쉬 메모리가 할당이 되고 공유 메모리가 존재해서 각 프로세서는 공유 메모리에서 필요한 데이터를 자신의 캐쉬에 복사를 하고 사용을 하게 되는데 한 프로세서에 의해서 데이터가 변경이 된 경우 동일한 데이터를 써야 하는 다른 프로세서는 그 데이터를 공유 메모리상에서 구할 수가 없게 된다. 제안된 구조도 동일한 토폴로지를 갖게 되는데 이 시스템에서의 메모리가 캐쉬 메모리에 해당하고 입출력 접점이 공유 메모리에 해당하게 된다. 즉 입출력 접점의 내용을 자신의 지역 메모리에 복사한 후에 논리적인 제어 연산을 수행하게 되는데 이 내용이 바뀔 경우 이를 출력 접점에 반영을 하게 되는데 다른 프로세서 역시 동일하게 이 출력 접점을 변경을 시키려 한다면 이 접점에 대한 일관성을 해치게 되는 것이다. 실제로 동일한 출력 접점에 대해서 서로 다른 프로세서에서 출력을 내게 되는 일은 있어서는 안되며 이는 사전에 예방이 되어야 한다. 그러나 수많은 프로그램을 다루는 프로그래머는 이를 일일이 검사하기가 어려우며 이는 입력 프로그램을 컴파일하는 단계에서 자동으로 검사하는 것이 더 효과적이다. 이를 자동으로 검사하기 위해서 컨피규레이션 시스템을 구현하며 또 이를 사용해서 시스템이 좀 더 효율적으로 운영이 되게 된다.

2.2 컨피규레이션 시스템

프로그래머블 로직 컨트롤러는 사용자가 원하는 제어 프로그램을 다양한 형태로 구현할 수 있는데 LD(Ladder Diagram), FBD(Functional Block Diagram), SFC(Sequential Flow Chart), IL(Instruction List), ST(Structured Text)등이 프로그래머블 로직 컨트롤러를 위한 국제 표준인 IEC1131-3에 정의가 되어있다[5]. 본 논문에서 예로 든 시스템은 IL을 입력 프로그램으로 사용하고 이를 컴파일한 후 실행 코드를 다운로드시켜서 실행하는 방식으로 실행이 된다. IL은 그 형태가 하위 수준 언어인 어셈블리 언어와 비슷한 구조로 되어 있어서 컴파일 단계에서 입출력 정보를 추출하기가 용이한 구조로 되어있다. 그림 2는 IL의 구조를 설명하는 그림인데 이 그림을 살펴보면 IL은 크게 두 부분으로 구성되어 있는 것을 알 수 있다. 앞 부분은 동작을 기술하는 명령어 부분이고 뒷 부분은 이 명령어가 실행되는 입출력 접점이다. 앞 부분의 명령어에서 실제로 수행하는 동작(쓰기/ 읽기)을 추출하게 되고 접점에서 사용되는 접점을 알아내게 되며 컴파일시에 사용자가 각 태스크에 지정하는 스캔 주기가 위의 정보와 더해져서 최종적인 접점에 대한 정보가 완성이 된다. 그 이름에서 알 수 있듯이 IL은 이러한 명령어 라인의 연속으로 이루어지게 된다.

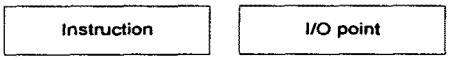


그림 2 IL(Instruction List) 의 형태

시스템에서 사용하는 입출력 접점에 대한 내역이 표 1에 제시가 되는데 각 접점은 프로그래머에 의해서 임으로 할당이 될 수 있다.

전체 Device	입출력 특성	접점수
X0~X1FFF	입력(Bit)	8192
WX0~WX1FFF	입력(Word)	8192
Y0~Y1FFF	출력(Bit)	8192
WY0~WY1FFF	출력(Word)	8192
R0~R7FFF	입출력(Bit)	32768
L0~L1FFF	입출력(Bit)	8192
WR0~WR7FFF	입출력(Word)	32768
WL0~WLFFF	입출력(Word)	4096
C0~CFFF	입출력(Bit)	4096
T0~TFFF	입출력(Bit)	4096
DR0~DRFFF	입출력(Double)	4096
합계		122880

표 1 전체 입출력 접점

사용자가 일련의 동작을 담당하는 하나의 입력 프로그램을 작성하면 이는 하나의 태스크에 해당하게 되고 각 태스크에는 우선 순위와 스캔 주기, 그리고 수행될 프로세서를 선택한 후에 컴파일의 과정을 거치게 된다. 컴파일러의 경우에는, 프로세서를 TMS320c40 DSP를 사용하였기 때문에 상업적인 이유로 컴파일러와 어셈블러 툴까지 모두 자체 제작되었는데 그중에서 컴파일러 부분에 프로세서간의 충돌을 검사하는 코드가 삽입이 되었고 오류가 없을 경우에만 실행 코드가 생성이 된다. 그림 3은 이러한 일련의 과정을 보여주는 순서도인데 로더라고 불리는 사용자 인터페이스 프로그램을 통해서 프로그래머가 각 태스크의 스캔주기, 우선순위, 수행될 프로세서 드의 프로젝트 정보를 입력한 후에 컴파일러를 호출하면 컴파일러는 각 태스크의 입출력 접점에 대한 컨피규레이션을 작성하고 그 정보를 모아서 각 프로세서에 대한 컨피규레이션을 작성한 후에 다시 이를 모아서 전체 시스템의 입출력 접점에 대한 컨피규레이션을 작성하여 실행 코드와 함께 다운로드 시키게 된다. 여기서 컨피규레이션이란, 앞에서 설명한 입출력 접점에 대한 모든 정보를 일련의 규칙에 따라서 제작한 테이블을 말한다.

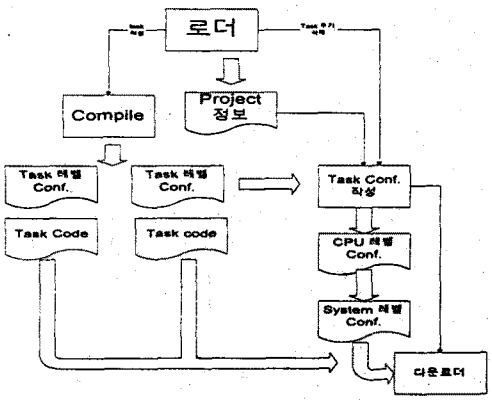


그림 3 컨피규레이션 시스템 생성 구조

컨피규레이션을 구성하는 내용을 살펴보면 우선 각 태스크의 컨피규레이션은 그 태스크에서 사용하는 입출력 접점의 블록 번호, 각 접점의 모드(읽기/쓰기), 스캔주기 등의 내용을 담게 된다. 각 태스크에 대한 컨피규레이션은 수행되는 프로세서의 컨피규레이션을 구성하게 되는데 각 프로세서의 컨피규레이션 역시 동일한 정보를 담게 된다. 이렇게 프로세서의 입출력 정보를 담은 각 프로세서의 컨피규레이션은 프로세서간의 충돌 여부를 검사하기 위해서 사용이 되고 이 검사를 통과하게 되면 최종적으로 시스템에서 사용되는 입출력 접점의 컨피규레이션으로 정리가 된다. 그림 4는 각 레벨에서의 컨피규레이션의 형태를 보인다.

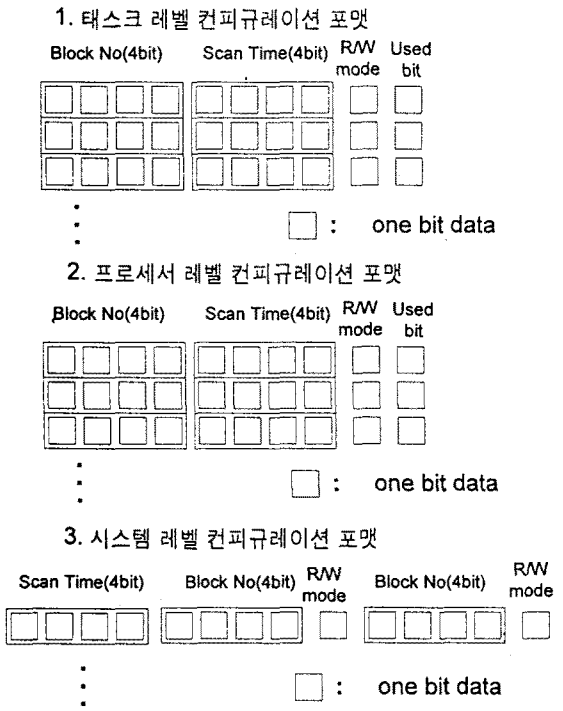


그림 4 생성된 컨피규레이션 시스템의 구조

실제로 입출력 접점은 일정한 단위의 블록 단위로 취급이 되는데 그 이유는 모든 접점에 대한 정보를 일일이 기록하고 검사하기에는 오버헤드가 너무나 크기 때문이다. 따라서 모든 접점은 일정한 크기의 블록 단위로 관리가 되며 프로그래머는 이 점을 유의하여야 한다. 즉 동일한 접점이 사용되지 않더라도 한 블록내의 접점을 다른 프로세서에서 출력 모드로 사용할 경우에는 에러를 발생하게 된다. 이 블록의 크기는 16개를 기본으로하여 사용자가 16의 배수를 마음대로 선택할 수 있도록 되어 있다.

문제점으로 지적된 프로세서간의 입출력 접점에서의 충돌 검사는 위의 그림 4의 두 번째 과정인 프로세서 레벨의 컨피규레이션을 통해 이루어 진다. 각 프로세서마다 생긴 위의 컨피규레이션에서 쓰기 모드로 설정된 입출력 블록들만 검사하여 중복되는 블록이 존재하면 충돌이 발생하는 것이다. 또한 최종적으로 생성되는 시스템 레벨 컨피규레이션을 이용하여 각 입출력 접점 블록의 실행에 사용을 한다.

3. 결 론

본 논문에서는 대용량 입출력 접점을 다루기 위해서 병렬 처리 기법을 사용하여 프로그래머블 로직 컨트롤러를 제작하고 이의 구현과정에서 생기는 입출력 접점의 프로세서간의 충돌을 해결하기 위한 컨피규레이션 시스템을 생성하는 방법에 대해서 설명을 하였다. 생성된 컨피규레이션은 충돌을 검사하는 본래의 목적으로 사용된 이후에는 입출력 접점에 대한 정보가 매우 중요한 프로그래머블 로직 컨트롤러의 특성상 시스템의 실행에 도움을 주기 위해 다시 사용이 된다. 본 시스템은 실제로 제작되어 (주)포스콘.에 납품이 되었으며 본 논문에서 설명한 부분에 대해서는 충분한 검증이 이루어 졌다.

[참 고 문 헌]

[1] Gilles Michel, " Programmable Logic Controllers", John Wiley & Sons.

[2] Ian G. Warnock, "Programmable Controllers", Prentice Hall.

[3] David A. Patterson, John L. Hennessy, "Computer Architecture. A Quantitative Approach", Morgan Kaufmann Publishers, 2nd Edition

[4] Nikitas Alexandridis, "Design of Microprocessor Based Systems", Prentice Hall.

[5] R.W.Lewis, "Programming Industrial Control Systems using IEC1131-3", The Institution of Electrical Engineers.