논문 2004-41SD-6-8

순차적 SMT Processor를 위한 Scoreboard Array와 포트 중재 모듈의 구현

(Implementation of a Scoreboard Array and a Port Arbiter for In-order SMT Processors)

허 창 용*, 홍 인 표**, 이 용 석**

(Chang-Yong Heo, In-Pyo Hong, and Yong-Surk Lee)

요 약

SMT(Simultaneous Multi Threading)구조는 여러 개의 독립적인 쓰레드들로부터의 명령어들을 이용하여, 이슈 슬롯을 채울 수 있도록 하는 쓰레드 레벨 병렬 성을 사용함으로서, 결국 프로세서의 성능을 향상시킨다. 독립적인 여러 개의 준비된 쓰레드들을 갖는다는 것은 실행 유닛들이 무용의 상태로 남아 있는 가능성을 줄일 수 있다는 의미이며, 이러한 사항은 결국 프로세서의 효율성을 증가 시키게 된다. SMT 프로세서에서 그러한 이점을 이용하기 위해서는, 이슈 유닛은 서로 다른 쓰레드들로부터의 여러 명령어들 간의 흐름을 제어해서, 그러한 명령어들 사이에서 충돌이 일어나지 않도록 해야 하지만, 이러한 사실로 인해 SMT 프로세서의 이슈 로직은 매우 복잡해지게 된다. 따라서, 본 논문에서 제안된 SMT 구조는 순차적 이슈와 완료 방식을 채택하여, 복잡한 레지스터 리네이밍이나 재순차 버퍼 등을 사용할 필요가 없이 비교적 간단한 스코어보드 어레이만을 사용하는 이슈 구조를 사용할 수 있게 하였다, 그러나, 여전히 SMT용 스코어보드 구조는 일반적인 단일 쓰레드의 범용 프로세서의 경우보다는 훨씬 더 복잡하고 많은 비용이 소요된다. 본 논문은 ARM 기본의 순차적 SMT 아키텍처 상에서의 최적의 스코어보드 메커니즘에 대한 구현을 제안한다.

Abstract

SMT(Simultaneous Multi Threading) architecture uses TLP(Thread Level Parallelism) and increases processor throughput, such that issue slots can be filled with instructions from multiple independent threads. Having multiple ready threads reduces the probability that a functional unit is left idle, which increases processor efficiency. To utilize those advantages for the SMTprocessors, the issue unit must control the flow of instructions from different threads and not create conflicts among those instructions, which make the SMT issue logic extremely complex. Therefore, our SMT architecture, which is modeled in this paper, uses an in-order-issue and completion scheme, and therefore, can use a simple issue mechanism with a scoreboard array instead of using register renaming or a reorder buffer. However, an SMT scoreboarding mechanism is still more complex and costlier than that of a single threaded conventional processor. This paper proposes an optimal implementation of a scoreboarding mechanism for an ARM-based SMT architecture.

Keywords: SMT, TLP, scoreboard, thread, issue

I. 서 론

프로세서의 성능을 나타내는 주요 지표이며, 해당 프로세서의 처리 능력을 가늠하게 해주는 대표적인 것으로, IPC(Instructions Per Cycle)수치가 사용 되는데, IPC는 단일 사이클 내에 얼마나 많은 명령어가 처리 될 수 있는가를 나타낸다. IPC를 증가 시킨다는 것은 기능

* 정회원, 삼성전자 DM 총괄
(Digital Media Business, Samsung Electronics)
** 정회원, 연세대학교 전기전자공학과 프로세서연구실
(Processor Laboratory, Dept. of Electrical and Electronic Engineering, Yonsei University)
접수일자: 2003년6월30일, 수정완료일: 2004년5월13일

유닛들로 명령어들을 이슈하는 이슈 유닛이 얼마나 많은 독립적인 명령어들을 같은 사이클에 전달 할 수 있게 하는가와 관련이 있다. 보편적인 단일 쓰레드용 파이프라인 구조의 스칼라 프로세서 같은 경우라면 IPC가 이상적 값인 1.0을 만족하지 못하는데[1], 이유는 프로세서 내에서 발생할 수 있는 여러 경우의 제한 사항들 때문이다.

이러한 제한 요소로 크게 세 가지를 얘기 하는데, 첫 번째가 자원의 충돌(resource conflict), 두 번째가 흐름 의존성(procedual dependency), 그리고 마지막으로 데이터 의존성(data dependency)문제가 존재 한다[2]. 첫 번째 문제는 하드웨어 자원을 늘림으로서 해결이 가능한 것이며, 흐름 의존성은 여러 가지 분기예측 방법 등을 수행하여 줄이게 된다. 마지막으로 이슈 유닛이 명령어들 사이의 의존성으로 인해서, 매 사이클 마다 명령어들을 이슈 할 수 없는 상황이 생기거나 매 사이클 마다 명령어를 이슈 할지라도 명령어 흐름이 일시적으로 정지되는 상황이 발생하게 되는데, 알려진 바와 같이 RAW(Read After Write), WAR(Write After Read), WAW(Write After Write) 의존성이 존재 한다[2].

따라서, 데이터에 의한 의존성 문제를 해결하기 위하여, 프로세서의 이슈 유닛이 명령어들을 이슈 하기 전에 이슈할 명령어들과 현재 파이프라인 상에서 수행중인 명령어들 사이의 의존성 문제를 검사 한 이후에 이슈를 할 수 있도록 처리해 주거나, 명령어 수행 중에 의존성이 검출되면 파이프라인을 정지 시킬 수 있도록 처리해야 하는데, 본 논문에서는 전자의 방식을 채택 한다. 그렇게 하면, 이미 이슈된 명령들에 대하여는 의존성이 존재하지 않는다는 가정 하에 파이프라인이 진행된다[3].

이러한 데이터 의존성을 해결하기 위한 방법으로 크게 두 가지 접근 방식이 있는데, 하나는 컴파일러에 의해서 미리 의존성이 없는 코드를 최대한 만들어 내도록 명령어를 실행 전에 미리 재 배치하여 주는, 정적 스케쥴링 방식과 하드웨어에 의하여 동적으로 의존성이 존재 하는 명령어를 검사하고 이슈 하기 이전에 명령어를 재 배치하거나 의존성이 해결 될 때까지 이슈를 제한하도록 하는 동적 스케쥴링 방식이 있다[2][9]. 스코어보드 어레이는 이러한 동적 스케쥴링을 지원하기 위한 방법의 한가지로 생각 할 수 있다. 기본적으로 스코어보드는 이슈 유닛이 명령어들 간의 의존성을 미리 조사해서 비순차적 방식으로 이슈를 하는 것을 가능하게 하는데, 그러한 비순차 명령 수행의 경우, 레지스터 리네이밍이나 재 순차 버퍼 등을 별도로 사용해야 함으로 인하여[2][4], 복잡한 SMT의 이슈 로직을 더욱 복잡하게 만들 수 있다.

일반적으로 이슈 유닛은 명령어의 동시 이슈 폭이 증가 할수록 비교하고 조사해야 할 명령어의 개수의 증가로 더욱 복잡해지게 되므로, 보통의 스칼라 프로세서 보다는 슈퍼스칼라 프로세서에서 더욱 복잡해지며[5], 본 논문에서 언급하는 SMT프로세서의 경우에는 그 복잡함이 더욱 증가하게 된다. 비록 순차적 수행 구조에서의 경우 보다는, 비순차적 수행 머신에서 재 순차 버퍼 등을 사용하는 경우가 프로세서의 성능 면에서는 보다 더 효과적인 구현이 될 수 있으나, SMT 구조에서는 쓰레드의 수가 증가 할수록, 비순차적 수행에 의한 프로세서의 성능 증가 폭이 순차적 수행에 의한 프로세서의 성능에 비하여 큰 차이가 나지 않으므로[11], 다중 이슈 구조의 SMT 프로세서의 경우 굳이 비순차적 이슈 구조를 채택함으로서 필요 이상으로 하드웨어의 복잡도를 증가시킬 필요는 없다. 본 논문의 SMT 프로세서에서는 순차적으로 이슈하고, 순차적으로 완료하는 구조를 채택함으로써, 불필요한 레지스터 리네이밍이나 재순차 버퍼의 사용을 없애고, 비교적 간단한 스코어보드 어레이만으로 SMT 프로세서를 설계 하도록 하였다.

본 논문의 구성은 다음과 같다. 우선 Ⅱ장에서 SMT 아키텍처 및 본 논문의 기본 모델이 되는 ARM기본의 SMT구조에 대하여 설명 하고 Ⅲ장에서는 제안된 SMT 구조를 바탕으로 실제 스코어보드 어레이의 설계에 대한 주제를 다루었다. 그리고 Ⅳ장 에서는 기본적인 스코어보드 어레이를 다중 쓰레드 구조에 맞게 포트 확장을 시키기 위한 포트 중재 로직에 대한 구조를 보이고, Ⅴ장에서 설계 방법 및 합성결과를 보였다. 마지막으로, Ⅵ장에서 향후의 연구 방향에 대한 고찰과 함께 결론을 맺는다.

Ⅱ. ARM-based SMT architecture

1. SMT 아키텍처

예전부터 프로세서의 성능을 높이기 위한 많은 아키텍처 모델이 발표 되었고, 구현 되었다. 슈퍼스칼라 나 VLIW같은 프로세서 아키텍처가 그러한 실례라고 볼 수 있는데, 비순차적 8-way 마이크로프로세서라고 할지라도 IPC는 보통의 파이프라인 가능한 스칼라 프로

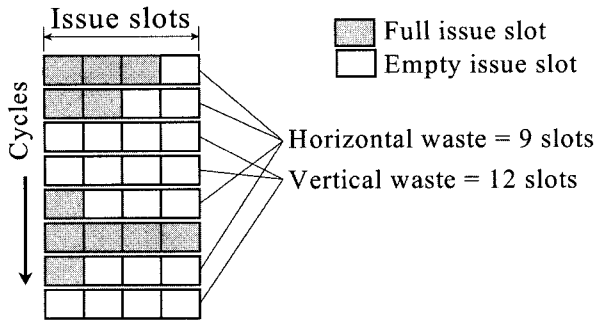

그림 1. 4 way issue 상에서 발생 가능한 낭비 시간
Fig. 1. Possible waste cycle of 4 way issue.

세서의 두 배 정도에 불과 하다[3].

프로세서의 성능을 높이려는 시도에서, 명령어들 간의 병렬성(ILP, Instruction Level Parallelism)을 더 이끌어 내고 효율적으로 이용할 수 있는 구조를 생각하게 되었는데, 명령어들 간의 병렬성을 하나의 쓰레드 내에서 뿐만 아니라, 독립적인 다른 쓰레드에서도 찾아내는, 쓰레드 간 명령어의 병렬성(TLP, Thread Level Parallelism)으로 확장하게 되었고, SMT 프로세서가 그러한 쓰레드간의 병렬성을 이용하여, 하나의 프로세서로 마치 여러 개의 프로세서가 명령을 수행하는 것과 유사한 동작을 보여 주는 구조를 지니고 있다.

그림1은 4-way 이슈 구조를 지니는 프로세서에서 발생 할 수 있는 낭비 사이클을 나타낸 그림이다[6].

기존 방식인 단일 쓰레드 구조에서는 명령어를 수행할 때 명령어간 의존성이 존재하면, 이후 명령어의 수행이 제한되어서 결국에는 기능 유닛의 사용율을 저하시키게 되는데, 그러한 ILP의 부족은 위의 그림 1에서처럼, waste slot을 발생 시키며, 이는 프로세서의 성능을 떨어뜨리는 원인이 된다. 그렇지만, SMT 프로세서는 쓰레드별로 다중 콘텍스트(context)를 갖는 구조로 되어있기 때문에, 쓰레드 간 독립성이 보장되고, 따라서 서로 다른 쓰레드의 명령어들 간에는 병렬성이 존재 할 수 있다. 이러한 쓰레드 간 병렬적인 명령어들을 이슈하게 함으로서, 위에서 발생하는 waste slot의 수를 현저하게 감소시킬 수 있게 된다. 따라서 SMT 구조에서는 여러 쓰레드들로부터의 명령어들이 실행 유닛들을 파이프라인 상에서 동적으로 공유 하도록 설계가 되어져야 하는데, 이러한 부분들은 SMT 구조의 레지스터 파일이나 스코어보드 어레이 등의 읽기 포트, 쓰기 포트 등의 증가를 가져오고[7], 이슈 유닛과 기능 유닛들이 쓰레드 구분 처리가 되어질 수 있게 설계 되어야 하는 등 SMT 구조를 하드웨어적으로 매우 복잡하게 만드는 요인이 된다.

표 1. ARM 기본의 SMT 구조
Table 1. ARM-based SMT architecture.

Thread number	8
Fetch width	8
Decode width	8
Issue width	8
Issue order	In-order
ALU/MAC/LSU	6/2/4 units.
Pipeline stage	S-F-D-I-R0-R1-E-M-W

2. ARM 기본의 SMT 구조

본 논문에서 정의된 SMT구조는 ARM ISA(Instruction Set Architecture) version 5와 호환되는 순차적 8 way 이슈 구조를 지닌다. 조건부 실행 명령, 분기 명령, 그리고 다중 사이클 명령들과 같은 특징들은 SMT 프로세서의 파이프라인 제어나 의존성 검사 등을 더욱 복잡하게 만드는 요인이지만, 이러한 사항들은 순차적 이슈와 완료 구조를 선택함으로써, 해결될 수 있다[3].

표 1은 본 논문에서 사용된 ARM 기본의 SMT아키텍처 모델을 보여 준다.

제안된 SMT 프로세서가 처리할 수 있는 쓰레드는 총 8개로서, 최대한으로 TLP를 활용하게 한다면 동시에 8개의 쓰레드들로부터 8개의 명령어를 이슈 할 수 있는 구조로 되어있다. SMT 프로세서는 슈퍼스칼라 구조를 기반으로 하고 있지만, 설계 복잡도가 슈퍼스칼라 구조에 비하여 상당히 높기 때문에 실용화에 큰 장애가 되고 있다. 따라서 그룹화에 의하여 자원과 쓰레드들을 몇 개의 그룹으로 나누고, 같은 그룹에 속하는 자원들을 공유하도록 하는 방식을 사용하게 된다[6].

SMT에서는 다중 쓰레드 구조 때문에, 동시에 여러 개의 기능 유닛에서 레지스터 파일로의 읽기, 쓰기 경로가 존재 하여야 한다. 본 구조에서처럼 8개의 쓰레드를 지원하는 구조라면, 8개의 레지스터 파일이 쓰레드마다 독립적으로 존재해야 하며, 각각의 레지스터 파일은 쓰레드 당 최대 2개로 명령어 이슈를 제한하고, 소스 레지스터의 개수를 명령어 당 3개로 생각 하여도, 읽기 포트 6개, 쓰기 포트 2개가 필요 하게 된다. 이러한 구조는 레지스터 파일의 액세스 속도를 매우 느리게 하여, 특별한 구조의 레지스터 파일을 설계하거나, 포트 인터페이스를 간략화 하도록 하는 연구를 별도로 필요

로 하게 된다[7]. 따라서 액세스 시간의 증가나 멀티 쓰레드로 인한 내부 로직 지연시간 등의 증가로 인하여, 페치 와 디코드 및 레지스터 읽기 단계는 파이프라인이 좀 더 세분화 되어져 있다. 본 구조에서 제안하는 파이프라인은 총 9단계로 이루어져 있는데, 페치 선택(S), 페치(F), 디코드(D), 이슈(I), 레지스터 읽기(R0, R1), 실행(E), 메모리(M), 레지스터 쓰기(W) 단계이다.

3. 이슈 제어 로직의 동작

제안된 아키텍처 모델에서는, 이슈 유닛은 명령어들 간의 의존성이 모두 해결되기 이전에는 이슈를 하지 않는다. 따라서 레지스터 읽기 스테이지 이후에는 파이프라인 스톨이 발생하지를 않게 된다. 이렇게 함으로써, 일단 이슈된 명령들이 파이프라인 상에 흘러 갈 때에는 의존성을 고려하지 않고 수행이 되기 때문에 레지스터 읽기 단 이후의 파이프라인 설계가 수월해 질수 있다. 이슈 제어 로직은 이슈 시에 의존성 검사를 위하여 다음과 같은 전략을 세우게 된다.

- RAW 데이터 의존성에 의한 이슈 제한

만일 이전 명령어의 목적 레지스터가 현재 명령어에 의하여 사용되어 진다면, 이전 명령어의 목적 레지스터가 완전히 갱신 되거나 바이패싱[2][9] 가능한 상태가 될 때까지 현재 명령어는 이슈 될 수 없다.

- WAR 데이터 의존성에 의한 이슈 제한

비록 제안된 SMT 구조가 순차적 이슈를 기본으로 하더라도, 로드 명령어로 인해서 WAW 데이터 의존성이 생길 수 있는 경우가 있다. 즉, 먼저 이슈된 로드 명령어가 외부 메모리 참조를 필요로 하게 될 경우가 생겨서 아직 메모리로부터의 로드 동작이 완료되기 이전에 후속하는 명령어에 의하여 동일한 목적 레지스터가 먼저 갱신되는 비순차적[4] 수행결과를 가져 올수 있다. 따라서, 이슈 로직은 이전 명령어의 메모리 단계가 완료 되었는지 안 되었는지를 관찰해서, 메모리 단계가 완료 된 이후에 후속 하는 명령어가 이슈 되도록 해야 한다.

- 다중 사이클 명령에 의한 이슈 제한

다중 사이클 명령어[9]는 명령어가 E(Execute) 단으로 이동함에 따라서, 새로운 RR(Register Read)단계를 생성 시키게 되는데, 새롭게 생긴 RR 단계는 새로운 파이프라인을 형성 하게 되고, 후속하는 명령어에 영향을 미칠 수가 있다. 그러한 경우에는, 먼저 이슈된 다중 사이클 명령이 아직 완료되기 이전에, 후속 하는 명령이 먼저 완료 되는 결과를 가져온다. 따라서 이슈 로직은 다중 사이클 명령어가 완료된 이후에 후속하는 명령어를 이슈할 수 있도록 제한해야 한다.

- 자원 제한에 의한 이슈 제한

이외에, 이슈 로직은 레지스터 포트 개수, 기능 유닛의 개수, 최대 이슈 폭 등의 제한에 맞추어서 이슈 할 수 있도록 해야 한다.

III. Scoreboard array의 설계

1. 스코어보드 어레이의 포트 크기 결정

스코어보드 어레이를 구현하기 전에, 우선 스코어보드의 읽기, 쓰기 포트 크기 및 전체 엔트리 사이즈를 결정을 해야 한다. SMT 구조에서 스코어보드 어레이는 쓰레드마다 독립적으로 존재한다. 제안된 구조의 이슈 제어 부는 한 쓰레드 당 최대 2개의 명령어를 동시에 이슈 할 수 있는 구조로 되어있기 때문에, ARM 명령어[8]에 따라, 4개의 읽기 포트와 1개의 쓰기 포트를 명령어 마다 가지고 있어야 한다. 즉, 총 8개의 읽기 포트와 2개의 쓰기 포트가 하나의 쓰레드를 위한 스코어보드 어레이의 포트로 존재 해야 한다. 또한, 목적 레지스터를 두개 지니고 있는 두 사이클 명령어로 인하여, 각각의 쓰기 포트는 한번의 쓰기 동작에 두개의 목적 엔트리가 동시에 쓰여 질수 있도록 해야 한다.

기본적인 쓰기 포트 이외에 우리는 조건 플래그들에 대한 쓰기 갱신도 고려해야 하는데, 명령어가 컨디션 플래그 갱신 명령일 경우가 있기 때문이다. 컨디션 플래그 갱신 명령인 경우에는 해당 명령어가 이슈되고, 해당 목적 엔트리가 갱신됨과 동시에 관련된 컨디션 플래그용 엔트리도 같이 갱신되어 가야 하기 때문이다. 그렇게 되면, 이슈 로직으로부터의 기본적인 쓰기 포트는 3개가 필요하게 되고, 읽기 포트는 한 개가 별도로 추가 되어야 한다.

기본적인 읽기 포트는 이슈 로직에 의하여 이슈 되기 이전에 사용되고, 이슈와 동시에 쓰기 포트가 사용되어지지만, 이와는 별도로 기능 유닛들로부터의 스코어보드 엔트리 갱신을 위한 경로가 있어야 한다[3]. 예를 들면, 조건부 명령의 실패, 플러쉬 신호 발생, 로드 명령어

표 2. 스코어보드 어레이의 자원
Table 2. Scoreboard array resources.

Resources	Number per thread
Read ports	9 ports
Write ports	7 ports
Entry	31 entries
LDM flag	1 bit

표 3. 스코어보드 엔트리 정보
Table 3. Scoreboard entry information.

Information	Description
FU	Function unit ID
RUS	Recent update stage
BPS	Bypass stage information
CU	Conditional execution
LU	Load instruction
I_ORDER	Instruction order
S_ORDER	Stage order

의 로드 히트 정보 등이 그것이다. 따라서 이슈로부터의 쓰기 포트 3개 와 기능 유닛들로부터의 조건부 명령 실패 포트 2쌍, 로드 히트 정보를 위한 포트 2쌍을 모두 고려하면, 모두 7개가 필요하다. 두 쌍씩 필요한 이유는 명령어가 쓰레드 당 2개가 실행 중일 수 있기 때문이다. 표 2는 본 논문에서 제안된 스코어보드 어레이의 자원을 보여 준다.

스코어보드 어레이의 각각의 엔트리는 ARM 레지스터마다 한 개씩 대응하여 존재 하여야 하는데, 이중에 PC(Program Counter) 레지스터는 항상 모든 파이프라인 흐름을 따라서 같이 이동되고, PC와 관련된 의존성은 분기예측 메커니즘에 의하여 해결 될 수 있으므로, 스코어보드 엔트리에서 제외 한다. 따라서, 스코어보드 엔트리의 총 개수는 컨디션 플래그를 포함하여, 모두 31개가 된다[8]. LDM flag는 파이프라인 상에 LDM (Load Data Multiple)명령어가 존재함을 나타내는 플래그로 사용된다.

2. 스코어보드 엔트리 정보의 결정

스코어보드 어레이가 사용되는 과정을 살펴보면, 이슈 유닛이 명령어를 이슈하기 이전에 우선 스코어보드 어레이를 읽어 보아야 한다. 이때, 스코어보드 어레이의 해당 엔트리를 최근에 갱신하는 명령이 현재 어떤 상태

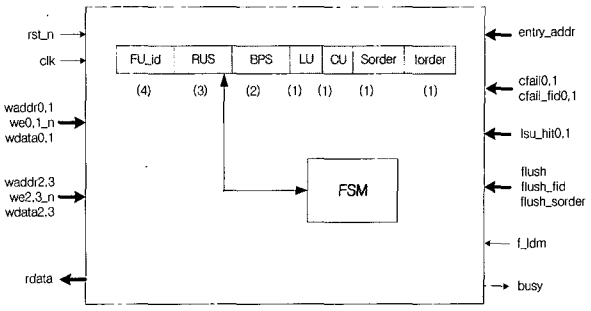

그림 2. 스코어보드 어레이의 한 엔트리
Fig. 2. One entry of scoreboard array.

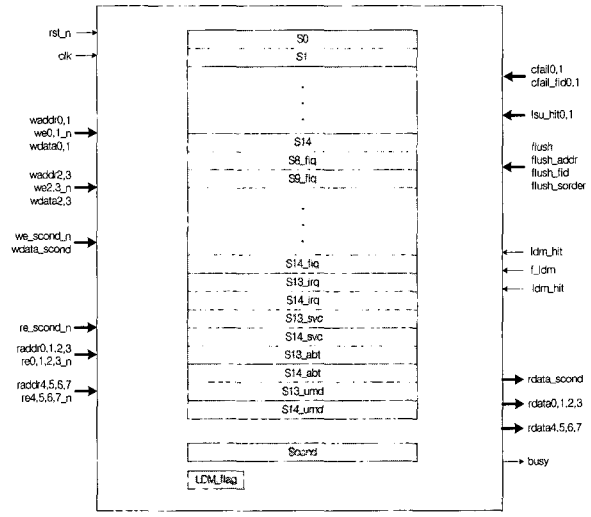

그림 3. 단일 쓰레드의 스코어보드 어레이
Fig. 3. A scoreboard array for 1 thread.

에 있는지를 나타내는 정보가 필요한데, 그것이 RUS (Recent Update State) 정보이고, RUS는 해당 명령어의 목적 레지스터가 최종적으로 갱신이 되고 나면, COMMIT (갱신완료) 상태로 바뀌기 된다. 따라서, 이슈 로직은 RUS가 레지스터 쓰기단계 혹은 COMMIT 상태이거나, 바이패싱 가능 스테이지가 될 때까지는 해당 레지스터를 소스 오퍼랜드로 취하는 명령어의 이슈를 막아야 한다[10].

RUS는 실행 유닛이 존재 할 수 있는 파이프라인 단계인 RR0, RR1, E(Execute), M(Memory), EM (Extend-ed Memory), W(Write), C(Commit) 상태중의 하나를 가지게 된다. 보통의 경우 각 단계는 매 클럭 당 한 단계씩 이동되게 되는데, 특별한 경우를 위한 제어가 필요하다. 예를 들면, M단계에서 캐쉬미스로 인해 외부 메모리를 액세스 하는 동안 RUS가 EM 상태로 변하게 해야 하거나, E 단계에서 후속하는 명령어에 의하여 다시 최신 갱신 되어서 RR0 단계부터 시작하도록 갱신시켜야 하는 경우가 발생한다.

기본적으로 RUS나 BPS의 정보 이외에 표 3에서 보여 지듯이 다른 정보 비트들도 이슈 유닛을 위해서 필요하게 된다. FU는 현재 엔트리를 갱신하는 명령 유닛이 어떤 것인지를 나타내며, CU는 현재 엔트리를 갱신하고 있는 명령어가 조건부 수행 명령에 의한 것임을 나타낸다. 또한, 로드 명령어임을 알려 주는 비트와 명령어간의 순서, 명령어내의 순서 등의 정보 비트들이 필요 하다.

그림 2 와 그림 3은 실제 구현된 스코어보드 엔트리와 하나의 쓰레드에 해당하는 스코어보드 어레이의 구조를 보여 준다.

3. 동작 고려 사항

모든 엔트리의 정보 비트들은 초기에 이슈 유닛에 의하여 이슈와 동시에 설정 되어지고, RUS를 제외한 필드들은 파이프라인 중간에 동일한 상태를 유지 하게 된다. RUS 필드는 스코어보드 엔트리 각각 마다 독립적으로 관리가 되어져야 하기 때문에, 별도의 엔트리 마다 존재 하는 내부 스테이트 머신의 제어를 받아 갱신 되어 진다. 따라서 엔트리가 갱신 되는 도중에 외부로부터의 예외 상황이나 캐쉬미스, 조건부 수행 실패 등의 경우를 제외하면 RUS는 매 클럭마다 다음 파이프라인 단계로 단순 변경 만 시키면 되고, 이외의 경우에 대해서는, 아키텍처 모델과 ARM 명령어 세트의 구조에 따라, 다음의 여러 가지 경우들과 상태를 고려해야 한다.

• 조건부 수행명령의 조건 검사 실패시의 RUS 갱신

ARM 명령어와 호환 되는 구조를 지녔으므로, 조건부 수행 명령어에 대해서 고려를 해야 한다. 조건부 수행 명령어라는 것은, 상태 레지스터의 특정 조건 코드가 명령어 상에서 지정된 조건을 만족하는 경우에만 수행을 하는 명령어들을 가리킨다. 이러한 조건부 수행 명령의 경우에, 기능 유닛들은 파이프라인의 E 단계에서 조건의 옳고 그름을 판단하게 된다. 만일 조건이 실패 하였다면, 해당 엔트리에 해당하는 스코어 보드상의 정보는 모두 플러쉬가 되어야 한다. 만일 두 사이클 명령어가 수행 중이었고, E 단에서 조건 검사 실패가 판단되면, 해당되는 명령의 첫 번째 목적 엔트리뿐만 아니라, 두 번째 목적 엔트리의 내용도 같이 플러쉬 되어야 한다. 즉, 첫 번째 목적 엔트리의 RUS가 E인 상태는, 동시에 두 번째 목적 엔트리의 RUS 가 RR1 상태에 있었을 것이기 때문에, 스코어보드 입장에서는 E단에서의 조건 검사뿐만 아니라, RR1 단에서의 조건 검사를 위한 경로도 존재해야 함을 의미하게 된다.

• 플러쉬 동작에 따른 RUS 갱신

기능 유닛에서 예외 상황이 발생 하거나 분기예측 오류 등에 의하여, 플러쉬 신호가 발생할 수 있다. 일단 스코어보드 어레이가 플러쉬 신호를 받게 되면, 해당 쓰레드 내의 전체 엔트리 중에서 플러쉬 신호를 발생 시킨 명령보다 이후에 이슈된 명령들에 대해서는 모두 플러쉬를 시켜주어야 하고, 이전에 이슈된 명령들에 대해서는 계속 진행 할 수 있도록 해주어야 한다.

제안된 아키텍처에서는 모든 플러쉬 신호는 M 단까지 끌고 가서, 그곳에서 발생 시키도록 함으로써, 스코어보드와의 경로를 줄일 수 있도록 하였기 때문에, 플러쉬 신호가 발생 되면, 그것이 발생된 스테이지는 M 단이라고 기준을 잡고, 스코어보드 엔트리 상에서 RUS가 M보다 이전의 것들에 대해서는 모두 플러쉬를 발생 시킨 명령 보다 이후에 이슈된 명령으로 생각 할 수 있으므로 플러쉬 시키도록 하면 된다. 만일, 엔트리의 RUS가 M단 이후의 것이면 이전에 이슈 된 명령어 이므로 그냥 계속 진행 시키도록 하면 된다.

그렇지만, 동일 사이클 내에서 한 쓰레드 당 최대 두 개의 명령어가 동시에 이슈가 가능하다는 것을 고려한다면, 조사하는 엔트리가 플러쉬를 발생시킨 명령과 동일한 RUS(M 스테이지)를 지니고 있을 수 있고, 이러한 경우에 둘 간의 명령어간의 순서 비트를 별도로 따져 보아야 한다. 두개의 명령 중에 어느 것이 순서 적으로 나중 명령어 인지를 알아야 하기 때문이다. 이때 사용되는 비트가 스코어보드 엔트리 정보 중에서 I_ORDER 비트 이다. 보통의 경우라면, 같은 M단에 존재하는 엔트리의 수는 두개가 최대이겠지만, 다중 사이클 명령어로 인해서, 한 가지 더 고려해야 할 상황이 생길 수 있다. 제안된 아키텍처에서는 다중 사이클과 후속하는 명령어를 동시에 이슈 할 수 없도록 제한하고 있는데, 이러한 경우에서 라면 동시에 RUS가 M으로 있는 엔트리의 수는 최대 3개가 될 수 있게 된다. 이렇게 되는 경우에는, 플러쉬가 발생 할 때, M단에 존재하는 엔트리 3개중에 어떤 엔트리를 플러쉬 해야 하고 어떤 엔트리를 플러쉬 하지 말아야 하는지를 결정하기 위해서, 명령어간 순서 비트 외에 명령어 내의 순서 비트도 같이 비교해야 하는 경우가 생긴다.

다음의 예로써 그러한 상황을 설명 할 수 있다. 만일,

다중 사이클과 후속하는 명령어가 동시 이슈 될 수 있다고 규정한다면, 더욱 복잡한 경우를 고려해야 된다.

명령 1: RR0 → RR1 → E → M (1)

명령 2: RR0 → RR1 → E → M (2-1)

RR0 → RR1 → E → M (2-2)

명령 3: RR0 → RR1 → E → M (3)

명령 4: RR0 → RR1 → E → M (4)

만일, 두 사이클 명령 2 가 (2-2)의 경우처럼, 자신의 두 번째 사이클의 M단에서 플러쉬 신호를 발생 시켰다면, 스코어보드 엔트리 중에서 명령 3과 명령 4에 해당하는 엔트리 역시 RUS를 M 으로 지니고 있을 것이다. 이러한 경우라면, 명령 3과 명령 4가 플러쉬 되어져야 하는지를 판단하는 기준이 있어야 한다. 당연히 명령 3과 명령 4는 순서적으로 나중의 명령 이므로 플러쉬 되어져야 한다. 다시 말하면, 만일, 두 사이클 명령의 두 번째 스테이지의 M 단에서 플러쉬가 발생 하였다면, 해당 엔트리 중에서 M 단을 RUS로 지니고 있는 엔트리는 순서 비트를 따질 필요 없이 항상 플러쉬 시켜 주어야 한다는 것을 의미 한다.

이번에는 명령 3이나 4의 M단에서 플러쉬가 발생 하였다면, 명령 2는 플러쉬 되면 안 되고, 명령 4나 3은 명령어 간의 순서 비트에 따라서 플러쉬 되어야 할 것이다. 즉, 만일 플러쉬를 발생시킨 명령이 단일 사이클 명령 혹은, 두 사이클 명령의 첫 번째 스테이지 의 M단이면, RUS가 M 단에 있는 엔트리 중에서 명령어내의 순서 비트가 0인 것은 명령어간의 순서 비트를 따져서 플러쉬 해주어야 하고, 명령어내의 순서 비트가 1인 것은 다중 사이클의 두 번째 스테이지에 해당하므로 현재 플러쉬를 발생시킨 명령어 보다 이전에 이슈된 다중 사이클 명령이 되므로, 플러쉬 시키면 안 된다는 의미 이다.

따라서 이러한 관계는 표 4의 관계처럼 정리 될 수 있다. f_sorder 비트는 플러쉬를 발생 시킨 명령의 명령내의 순서비트를 의미하고, r_sorder 비트는 현재 검사되는 엔트리의 명령어 내 순서비트를, r_iorder 비트는 명령어 간 순서 비트를 의미 한다.

해당 비트가 0인 것이 순서적으로 앞선 명령임을 의미 한다. 즉, f_sorder가 1이면, 명령 내 순서 비트가 1인 것에 의하여 플러쉬가 발생했음을 나타낸다. 명령 내 순서비트가 1이라는 것은 두 사이클 명령어의 두 번째 사이클임을 의미 하고, 명령 간 순서 비트는 동시 이슈된 두개의 명령간의 순서 비트를 의미 한다.

표 4. 순서비트에 따른 플러쉬
Table 4. Flush according to order bits.

f_sorder	r_sorder	r_iorder	Operation
0	1	x	No flush
0	0	0	No flush
0	0	1	Flush
1	x	x	Flush

- 로드 스토어 유닛의 동작에 따른 RUS 갱신

만일, 로드 명령이 외부메모리 로의 접근을 필요로 한다면, 스코어보드 어레이의 해당 엔트리는 외부 메모리에 대한 히트가 발생하기 전까지는 M단에서 W상태로 바뀔 수 가 없을 것이다. 그러한 경우에 스코어보드는 외부 메모리로부터의 로드가 완료되어 내부 로드버퍼에 로드 할 데이터가 준비가 되었는지 안 되었는지를 알아야 할 필요가 있다. 따라서 앞에서 언급 한 대로, 로드 스토어 유닛은 로드 명령을 위한 메모리 히트가 발생했음을 알려 주는 신호가 필요 한데, 이것이 LSU_hit 신호 이다. 스코어보드는 LSU_hit 신호를 보고, 엔트리의 RUS를 W로 바꿀 수 있게 된다.

그렇지만, LDM 같은 다중 사이클 명령어로 인하여, 동시에 갱신 되어져야 하는 다수의 엔트리가 존재하게 되면, 그러한 엔트리들은 갱신되어질 때에 서로 다른 순서에 따라 갱신되어져야 한다. 즉, PC 레지스터를 갱신하느냐 혹은 베이스 레지스터를 갱신 하느냐 에 따라 갱신되어지는 목적 레지스터들의 순서가 변하게 된다. 또한, LDM 명령 같은 다중 사이클 명령어 바로 뒤에 후속 하는 명령어가 온다면, 후속 하는 명령어가 선행하는 LDM 명령보다 먼저 완료 될 수 있다. 그러한 사항은 비순차적 수행 결과를 야기 시켜서 본 논문의 아키텍처 규정상 어긋나게 되고, 여러 개의 갱신 가능한 엔트리를 관리 한다는 것은 엔트리의 모든 순서 정보뿐만 아니라 바이 패스 정보 등을 따로 관리 해 주어야 하기 때문에, 복잡도에 따른 부담이 너무 커지게 된다.

그래서, 본 규정에서는 LDM 명령이 완료 될 때까지는 후속하는 명령어의 이슈를 제한하는 규정을 두기로 한다. 따라서 LDM 명령이 현재 파이프라인 상에 존재

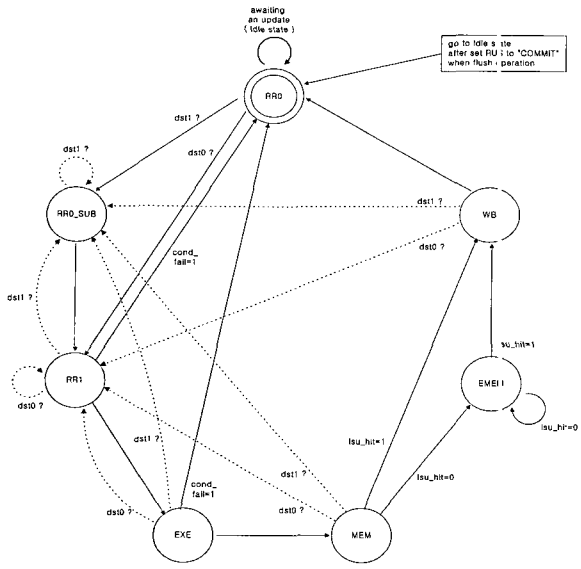

그림 4. 스코어보드 엔트리의 상태도
Fig. 4. State flow of the scoreboard entry.

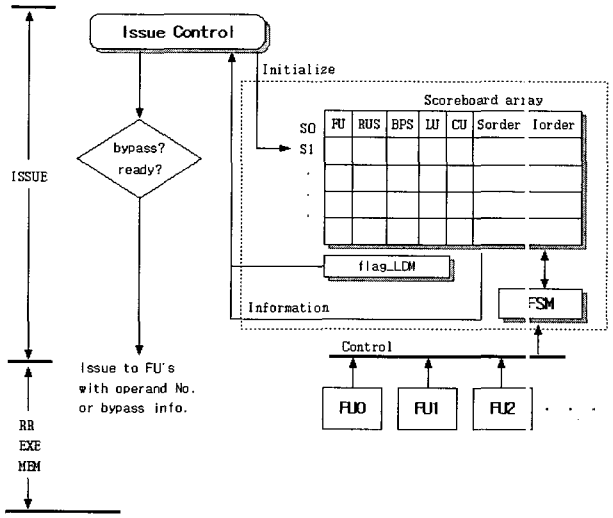

그림 5. 이슈 로직과 스코어보드 어레이
Fig. 5. Issue logic and scoreboard array.

하는지를 나타내는 비트로서 표 2에서 언급 하였던 LDM flag를 두기로 한다. LDM flag는 이슈 유닛으로부터 오는 정보를 이용하여 LDM 명령의 이슈 시에 설정 하였다가, LDM 명령의 마지막 목적 레지스터가 로드 히트 되면, 스코어보드 어레이가 자체적으로 지워주도록 한다. LDM의 마지막 목적 레지스터가 로드 히트 되는 정보는 로드 스토어 유닛으로부터 받게 되고, LDM 플래그가 1인 동안에는 이슈 유닛은 다음 명령어의 이슈를 제한하게 하도록 설계 한다.

4. 스코어보드 엔트리의 상태 머신

각각의 엔트리는 ARM 기본의 SMT 아키텍처의 규

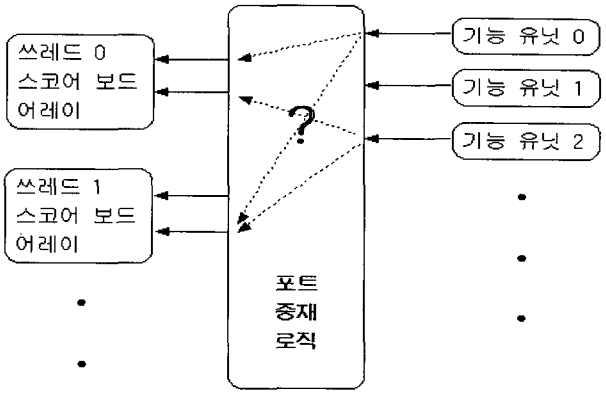

그림 6. 포트 중재 로직
Fig. 6. Port arbitration logic.

정에 따라, 자신의 독립적인 스테이트 머신을 지니고 있다. 보통의 조건에서는 RUS는 RR0부터 M단 까지 순차적으로 클럭의 상승 에지에서 변하도록 동작되지만, 앞 절에서 설명 한대로 여러 가지 경우를 고려해서 RUS의 상태를 갱신 할 수 있도록 해야 한다.

그림 4는 주어진 제한 사항에서 동작 되도록 설계 되어진 스코어보드 엔트리의 스테이트 머신의 상태 천이도 이고, 그림 5는 설계된 스코어보드 어레이와 이슈 로직과의 연결 상태를 보여 주는 블록도이다.

Ⅳ. Port arbitration로직의 설계

앞장에서 이미 소개 하였듯이, 이슈 로직과 직접적으로 연결되는 두개의 읽기 포트와 3개의 쓰기 포트는 쓰레드별로 관리가 되고, 이슈 큐는 쓰레드별로 개별적으로 존재하기 때문에, 이슈 제어 로직을 통하여 직접적으로 각 쓰레드의 읽기, 쓰기 포트로 별다른 중재 없이 연결이 가능 하다고 볼 수 있다. 하지만, 기능 유닛들로부터의 쓰기 참조 경로인 4개의 쓰기 포트들에 대해서는 추가적인 포트 중재가 필요 하게 된다.

기본적으로 중재가 필요한 이유는 다음과 같다. 기능 유닛의 개수는 여러 개인데, 기능 유닛으로부터 쓰레드별 스코어보드 쪽으로 입력되어야 할 포트의 수는 최대 2개 뿐 이기 때문이다. 따라서 어떠한 식으로든 여러 개의 기능 유닛들로부터 2개의 쓰레드 포트 쪽으로의 포트 중재가 필요 하다.

예를 들어, 기능 유닛 0번이 조건 실패 신호를 발생하고, 기능 유닛 4번이 또 조건 실패 신호를 발생 하면, 각각의 신호들이 어떠한 쓰레드의 명령 때문에 발생 한 것인지를 쓰레드 식별자를 이용하여 분리하여 내고, 다

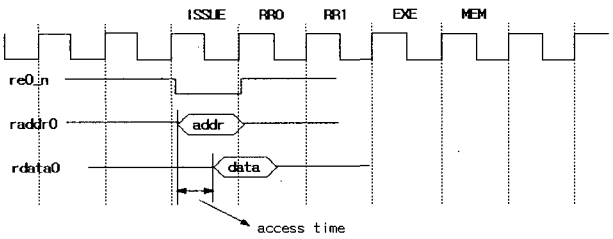

그림 7. 이슈 유닛의 스코어보드 어레이 읽기 동작
Fig. 7. Read operation of issue unit

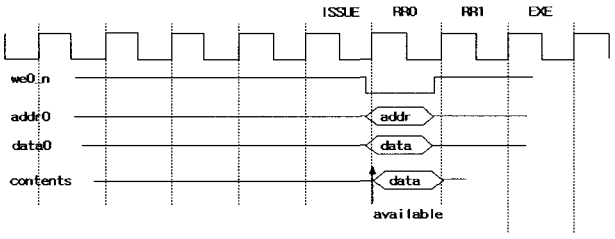

그림 8. 이슈유닛의 스코어보드 어레이 쓰기 동작
Fig. 8. Write operation of issue unit.

시 해당 쓰레드에서도 두개의 포트 중에 어느 쪽 포트를 통하여 신호를 넣어 줄지를 결정해주는 과정이 필요한 것이다. 따라서 본 논문의 아키텍처에서 소유하고 있는 기능 유닛의 개수는 그룹 당 모두 7개 이므로, 7개의 기능 유닛들로부터, 4개의 쓰레드의 2개씩의 포트로 적절하게 포트 할당이 되어져야 한다. 스코어보드의 쓰기 포트에 대한 이슈 로직, 기능 유닛들과의 포트 연결은 다음과 같이 세 가지 부류로 나누어서 고려 할 수 있다.

1. 이슈 로직으로부터의 쓰기 경로

일반적으로 이슈 로직이 명령어를 이슈 하는 시점에서, 두개의 쓰기 포트를 통하여, 스코어보드 어레이에 초기값을 주게 된다. 이슈유닛으로 부터의 쓰기 포트는 이슈 큐로부터 오게 되는데, 이슈 큐가 그룹 당 4개의 총 쓰레드 개수만큼 존재 하므로, 별도의 포트 중재는 필요 없다. 즉, 그룹 당 4개의 이슈 큐로부터의 경로는 4개의 쓰레드별 스코어보드 어레이 쪽으로 직접 연결 되어진다. 이때, 같은 쓰레드의 명령이 두개라면 해당 쓰레드의 스코어보드 어레이의 포트 두개 쪽으로 연결이 되도록 이슈 쪽에서 구성이 되면 될 것이다. 이럴 때는 별도의 포트 중재 없이 포트 0에 이슈 큐 의 첫 번째 명령, 포트 1에 이슈 큐의 두 번째 명령이 들어가도록 고정 적으로 할당해 놓도록 하였다.

그림 7과 그림 8은 각각 이슈 유닛이 스코어보드 어레이에 별다른 중재 로직을 거치지 않고 직접 읽고 쓰는 동작에 대한 타이밍을 보여 준다.

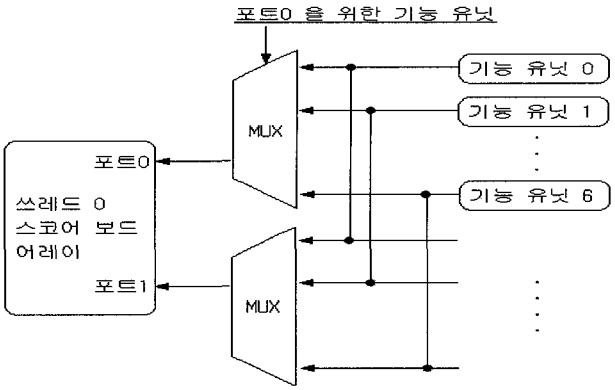

그림 9. 포트 중재를 위한 mux 단
Fig. 9. Mux for port arbitration.

2. 로드 스토어 유닛으로부터의 로드 히트 경로

LSU(Load Store Unit)는 그룹 당 두개가 존재 하고, 따라서, 두개의 LSU로부터의 로드 히트 신호는 각각 쓰레드의 LSU를 위한 포트 0, 포트 1로 고정적으로 할당시키면 된다. 즉, LSU0은 포트 0으로 연결하고, LSU1은 포트 1로 연결하면 포트에 대한 중재는 별 달리 필요가 없다. 하지만, 쓰레드 구분은 쓰레드 식별 자를 이용해서 중재해야 한다.

3. 기능 유닛으로부터의 조건부 실행 실패 경로

포트 중재 로직이 가장 복잡한 부분이 이 부분인데, LSU의 로드히트 정보와는 달리, 조건부 실행의 실패 신호는 모든 기능 유닛들로부터 발생 할 수 있기 때문에, 그룹 당 총 7개의 기능 유닛들로부터 각각의 쓰레드용 스코어보드 어레이의 두개의 조건부 실행 실패 신호를 위한 쓰기 포트 쪽으로 중재 로직이 필요 하다. 만일 서로 다른 기능 유닛에서 같은 쓰레드의 명령이 동시에 실행되고 있을 수 있다면, 해당 쓰레드의 포트를 할당할 때 낮은 기능 유닛의 번호에 포트 0번을 할당 하고, 높은 기능 유닛의 번호에 포트 1번을 할당하는 식으로 고정 시켜 놓도록 한다.

그림 9에 여러 개의 기능 유닛과 각 스코어보드 어레이상의 포트로의 멀티플렉서의 연결 상태를 나타내었다. 예를 들어, 기능 유닛 3번의 조건부 실행 실패 신호가 해당 쓰레드의 특정 포트 쪽으로 할당이 되어지는 경로를 고려 해 보기로 하자. 우선, 기능 유닛 3이 쓰레드의 어떤 포트를 가질지 포트 선택을 위해서, 기능 유닛 3의 쓰레드 식별자와 동일한 쓰레드 식별자를 가진 기능 유닛이 기능 유닛 0,1,2 중의 하나에 존재 한다면

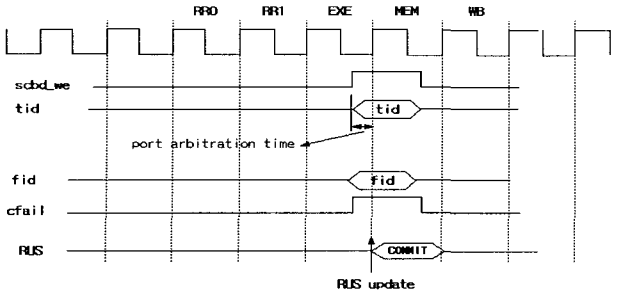

그림 10. 기능유닛의 스코어보드 어레이 쓰기 동작
Fig. 10. Write operation of functional unit.

기능 유닛 3을 위한 포트는 1번으로 할당 하고, 동일 쓰레드 식별자를 지닌 낮은 번호의 기능 유닛들 중의 하나에 포트 0 번을 할당 하도록 한다.

이렇게 하면, 일단, 각 기능 유닛별로 포트를 몇 번을 사용 하게 될지를 결정 할 수 있게 된다. 이것을 각 기능 유닛들이 지니는 쓰레드 식별자와 연결하면, 기능 유닛 0 은 어떤 쓰레드의 몇 번 포트를 사용 하고, 기능 유닛 1은 어떤 쓰레드의 몇 번 포트를 사용 하는지 등등을 모든 기능 유닛들에 대하여 알 수 있게 된다. 최종적으로 "기능 유닛들에 할당 되어진 쓰레드와 포트 번호" 정보로부터 "각 쓰레드의 포트들에 할당 되어져야 할 기능 유닛 번호" 정보를 만들어 냄으로써, 각각의 쓰레드의 포트들에 연결 될 7-입력 멀티플렉서의 제어 신호로서 사용 할 수 있게 된다. 그림 10 에서는 기능 유닛들로부터의 스코어보드 어레이 쪽으로의 중재 모듈에 대한 경로가 포함된 쓰기 동작에 대한 타이밍 도를 보여 준다. 기능 유닛의 E단에서 나온 출력이 포트 중재 모듈을 거쳐 스코어보드 어레이로 입력되면, 스코어보드 어레이 내부 로직에서 클럭의 상승에지에서 RUS를 갱신 하게 된다.

V. 설계 방법 및 합성 결과

1. 설계 및 검증 방법

제안된 스코어보드 어레이의 구조를 검증하고, 성능을 평가하기 위하여, 우리는 C 언어로 작성된 사이클 기본의 SMT 코어 시뮬레이터를 사용 하였으며, SPEC CPU2000 벤치마크 프로그램들과 ADS(ARM Developer Suits) 의 예제 코드들을 사용하여 성능 측정을 하였다.

우리가 디자인 한 모든 스코어보드 알고리즘은 시뮬레이터 상에서 동일하게 구현 되었으며, 만족할 만한 결과를 얻었다. 결론 적으로 구현된 스코어보드 어레이를 사용하여, 8 이슈 SMT 프로세서 상에서 4.5417의 IPC

표 5. 합성 결과
Table 5. Synthesized results.

module	Gate count	Delay
1 스코어보드 엔트리	747	2.47
1 스코어보드 어레이	25,441	3.18
포트 중재 로직	2,107	3.76
전체 스코어보드 / 그룹	91,838	5.38

수치를 얻었다.

이러한 결과와 시뮬레이터의 알고리즘 검증을 바탕으로 본 논문에서 구현된 모든 구조는 Verilog HDL 언어를 사용하여 기술 되었고, Verilog-XL을 사용하여 검증하였다.

2. 합성 결과

모든 디자인된 하드웨어 로직들은 삼성 반도체의 0.35 um CMOS 표준 셀 라이브러리를 사용하여, Synopsys사의 Design Analyzer를 이용하여 합성 되었다. 최종적으로 합성된 스코어보드 어레이 및 포트 중재 로직에 대한 합성결과는 표 5에서 보여 진다. 모듈별 사용 면적과 임계경로 지연에 대한 타이밍을 나타내었다.

지연 시간은 3.0V, 85°C 의 최악 조건 하에서 보고된 것이며, 사용 면적은 등가 게이트 사이즈로 나타내었다.

하나의 스코어보드 엔트리에 내에서의 최대 지연 경로는 RUS가 M 단계 일 때 거쳐 가는 경로를 포함하며, 스코어보드 엔트리가 메모리 단에서 플러쉬 동작을 수행할 때, 가장 많은 경우의 수를 지니기 때문이다. 총 31개의 엔트리가 포함된 하나의 스코어보드 어레이는 하나의 엔트리의 임계경로와 같은 경로를 지니지만, 팬 아웃의 증가로 로직 게이트의 딜레이가 조금씩 더 커짐으로 인하여, 증가 된 것을 알 수 있다. 조합 논리 회로의 가장 많은 부분을 차지하는 포트 중재 로직에서는 비교적 큰 수치의 지연 시간을 보이고 있으며, 총 4개의 스코어보드 어레이와 하나의 포트 중재 로직이 포함된 하나의 그룹의 스코어보드 전체 로직에 대한 최대 임계 경로 지연시간은 5.38 ns 로 나타났다. 이 시간은 스코어보드 어레이의 포트 중재 로직을 거쳐 최종적으로 해당 쓰레드에서의 스코어보드 어레이의 특정 엔트리의 RUS가 갱신 될 때까지의 최대 지연 시간을 나타낸다.

풀 커스텀 디자인을 한 부분이 아니고,, 합성 툴을 이용하여 로직 합성을 한 것에 대한 사항이기 때문에, 전

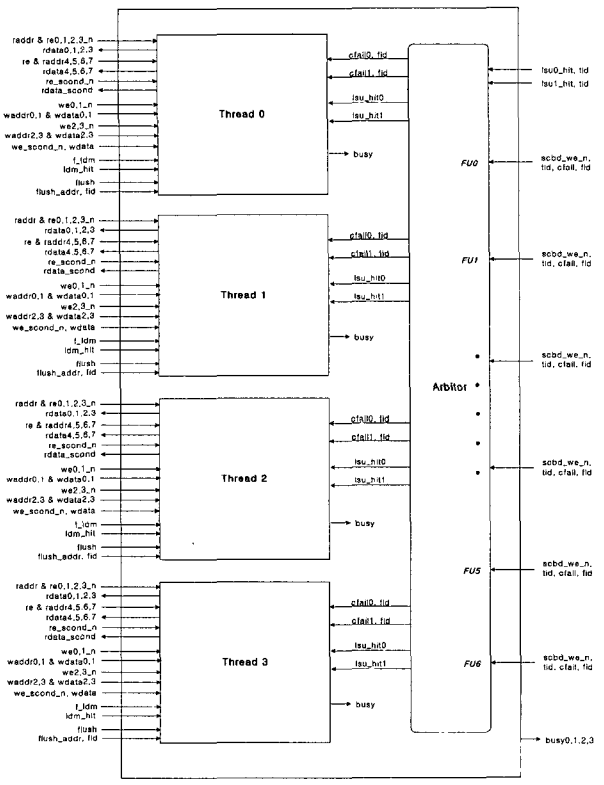

그림 11. 스코어보드 어레이와 인터페이스
Fig. 11. Scoreboard array and interface.

체적인 로직 사이즈나 지연 시간은 합성시의 제한 파라미터 등에 따라 차이를 보일 수 있으나, 본 논문의 구현에 따르면, 디자인된 스코어보드 어레이는 포트 중재 로직을 포함하여, 0.35um 공정상에서 약 185 MHz의 클럭 속도에서 동작 할 수 있도록 설계 되었다. 그림 11은 최종 설계된 스코어보드 어레이와 포트 중재 로직간의 블록도를 보여 준다.

VI. 결 론

일반적으로 슈퍼스칼라 구조에서 보여 지는 스코어보드 어레이는 단순히 하나의 비트만을 이용하여, 이슈 시에 해당 목적 레지스터가 사용 중인지 아닌지를 판별하는 비교적 단순한 기능만을 제공 한다. 또한, CDC6600에서 보여 지는 스코어보드 어레이의 구조는 해당 소스 오퍼랜드가 어떤 기능 유닛에 의하여 목적 레지스터로 사용되고 있는지에 대한 정보를 결과 레지스터 지시자 등의 엔트리를 통하여 별도로 지니고 있지만, 최종적으로 그러한 정보를 이용하여 역시 오퍼랜드의 준비(ready) 여부를 판단 할 수 있도록 하는 비교적 간단한 기능 위주로 설계 되어져 있다고 볼 수 있다[12].

그에 비하여, 본 논문에서 설계한 스코어보드 어레이는 목적 레지스터에 대한 기능 유닛정보 뿐만 아니라, 해당 레지스터가 현재 어떤 기능 유닛에 의해 어떤 스테이지 상에서 돌고 있는지에 대한 정보와 해당 오퍼랜드가 어느 스테이지에서 바이패싱이 가능 한지에 대한 정보를 같이 제공함으로 인해서, 스코어보드 상의 정보를 이용하여 데이터 바이패싱을 가능 하도록 설계 하였고, 또한 자체에 조건부 명령에 의한 조건 실패 시나 예외상황 등의 경우에 대하여 엔트리를 선택적으로 이전 상태로 복귀 시킬 수 있도록 하는 기능을 내장 시켰다.

또한, 다중 쓰레드 구조에 맞도록 쓰레드 식별자를 통하여 동적으로 공유 되는 명령 유닛들 간의 포트 중재를 가능 하게 함으로서, 간단한 구조를 유지하면서 다기능을 수용 할 수 있도록 설계를 하였다.

본 논문은 SMT 프로세서를 위한 스코어보드 어레이의 메커니즘을 순차적 이슈와 완료 구조를 지니는 ARM 기본의 아키텍처 상에서 제안하고, HDL언어를 이용하여 구현 하였다. 비록, SMT 프로세서가 다중 쓰레드 구조를 통하여, 고 성능을 실현 하지만, 높은 디자인 복잡도와 비용으로 하나의 실리콘으로 만들어 내기에는 어려움이 따른다. 따라서 제안된 SMT 프로세서의 스코어보드 어레이 메커니즘은 그 복잡도를 감소시키면서도, SMT 프로세서가 지니는 성능을 기본적으로 유지시킬 수 있도록 구현 하였다.

기본적으로 본 논문에서 구현한 스코어보드 어레이와 그것의 인터페이스 로직은 우리가 제안한 스코어보드 알고리즘을 그대로 구현 하는데 중점을 두었지만, 다중 쓰레드 구조를 가짐으로 인해서 생기는 복잡한 인터페이스 로직들, 즉 동적 자원 공유나 포트 중재 로직들에 대한 좀 더 최적화 된 연구를 필요로 한다.

예를 들면, 포트 사이즈를 쓰레드 당 2개씩 구성 하여, 전체 적으로 한 그룹(4개의 쓰레드) 당 8개의 포트를 지니는 고정 할당 방식 대신에, 전체적으로 하나의 그룹 당 최대 이슈 폭인 4개의 명령어를 위하여 4개의 포트를 두고 이슈 로직에서 사용 할 수 있도록 하는 방안을 연구 할 필요도 있다. 이러한 경우는 포트 중재 문제가 고정 방식에서 쓰레드별로 두개씩 고정 적으로 사용하여 중재 하는 것보다 로직이 더 복잡해 질것으로 판단이 되지만, 포트 수를 줄일 수 있다는 장점이 존재하는 부분이기도 하다. 그러한 부분들에 대한 연구를 통하여 성능 비교를 하고, 좀 더 나은 SMT 프로세서에 적합한 동적 자원 공유 방식을 제안 할 계획 이다.

참 고 문 헌

[1] Gurindar S. Sohi, Scott E. Breach, T.N. Vijayku mar, "Multiscalar processors", in Proc. of 22nd annual international symposium on Compu:er arc hite cture, pp. 414-425, S. Margherita Ligure, Italy, 1995.

[2] Dezso Sima, Terence Fountain, Peter Kacsuk, *"Advanced computer architectures"*, Addison-Wesley, 1998.

[3] 문병인, *"순차적 SMT 구조 및 그룹화 방안에 관한 연구"*, 연세대학교 대학원 전기전자공학과, 공학박사 학위논문, 2002년 12월.

[4] Roger Espasa, Mateo Valero, James E. Smith, "Out-of-order vector architectures", in Proc. of 30th annual ACM/IEEE international symposium on Microarchitecture, pp. 160-170, 1997.

[5] Sorin Cotofana, Stamatis Vassiliadis, "On the Design Complexity of the Issue Logic of Super scalar Machines", in Proc. of 24th Euro- micro Conference, pp. 277-284, 25-27, Aug. 1998.

[6] Dean M.Tullsen, Susan J. Eggers, Henry M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", in Proc. of 22nd Annual International Symposium on Computer Architec ture, pp. 392-403, 22-24 Jun 1995.

[7] R. Balasubramonian, S. Dwarkadas, D.H. Albo nesi, "Reducing the Complexity of the Register File in Dynamic Superscalar Processors", in Pro c. of 34th annual ACM/ IEEE international sym posium on Micro- architecture, pp. 237-248, Texas, 2001.

[8] *ARM Architecture Reference Manual*, Part A. CPU Architecture, 1996.

[9] John L. Hennessy, David A. Patterson, *"Com puter Organization and Desgin"*, Morgan Kauf mann, Publishers, Inc. 1998.

[10] Toyohiko Yoshida, Masahito Matsuo, Tatsuya Ueda and Yuichi Saito, "A Strategy for Avoi ding Pipeline Interlock Delays in a Micropro cessor", in Proc. of Computer Design:VLSI in Computers and Processors, ICCD '90., pp.14_19, 17-19 Sep 1990.

[11] Hily. S., Seznec. A., "Out-of-order execution may not be cost-effective on processors featu ring simultaneous multi-threading", in Proc. of High-performance computer architecture, fifth international symposium, pp64-67, 9-13 jan 1999.

[12] Robert J. Baron, Lee Higbie, *"Compupter Archit ecture"*, Addison-wesley publishing company, 1992.

저 자 소 개

허 창 용(정회원)

1996년 홍익대학교 전자공학과 공학사.

1996년 1월~2002년 2월 삼성전자 DM 총괄 연구원.

2004년 2월 연세대학교 전기전자 공학과 공학석사.

2004년 3월~현재 삼성전자 DM 총괄 연구소 DTV chipset 개발.

<주관심분야: 마이크로프로세서 아키텍처, 오디오 코덱 및 인터페이스, DTV용 VLSI 설계>

이 용 석(정회원)

1973년 연세대학교 공학사.

1977년 University of Michigan Electrical Engineering 공학석사

1981년 University of Michigan Electrical Engineering 공학박사.

1993년~현재 연세대학교 전기전자공학과 교수.

<주관심분야 : 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계>

홍 인 표(정회원)

1999년 연세대학교 전자공학과 공학사.

2001년 연세대학교 전기컴퓨터 공학과 공학석사.

2001년 3월~현재 연세대학교 전기전자공학과 박사과정.

<주관심분야: 마이크로프로세서 설계, VLSI 설계, 고성능 연산기 설계>