2011 년 12 월

11ZS1100-11-7031P

전체부수

초저전력 비동기 프로세서 설계 기술 개발사업

Ultra Low Power Asynchronous Processor Design Technology Development

ETRI 한국전자통신연구원
Electronics and Telecommunications Research Institute

내 표 지

인 사 말 씀

현재 세계는 새로운 기술 및 다양한 융 복합 기술 및 제품에 기반을 둔 새로운 서비스가 경쟁적으로 출시되고 있으며, 이러한 서비스들을 활용하는 우리의 삶은 좀더 풍요로워 지고 있습니다. 소비자는 이러한 제품과 서비스의 무한 경쟁 시대에 개인의 취향에 따라 다양한 선택을 할 수 있게 되었으나, 이러한 환경의 도래에 따른 정보의 공유 및 활용에 따라 개인 프라이버시 침해문제가 새로운 이슈로 대두되고 있으며, 그리고 이러한 인프라를 구축하고 운영하는데 필요한 각종 IT 기기들의 에너지 과다 소비문제 등으로 인하여 우리는 새로운 도전에 직면하고 있습니다.

이와 같이 지구 온난화 및 에너지 문제가 글로벌 이슈로 떠오름에 따라 저탄소 녹색성장이 IT 산업뿐만 아니라 전 산업에 걸쳐서 핵심 경쟁요인으로 떠오르고 있습니다. 이에 따라 부가가치가 매우 높고 주변 산업에 미치는 파급효과가 매우 큰 시장 선도형 기술인 프로세서 기술의 개발은 매우 중요하며, 의미 있는 일이라 평가됩니다. 비록 과제의 연구개발 예산 규모는 작지만 이러한 글로벌 그린 IT 기술개발 추세에 적극 대응하고, 중장기적으로 국내에서 저전력 프로세서 기술을 확보하기 위하여 이러한 연구개발을 수행하는 것은 중요한 의미가 있습니다. 본 연구개발을 통하여 국내에서도 저전력 프로세서 기술 및 관련 IPR을 확보하여 선진 기업들의 공세에 적극 대응하고, 향후 이를 기반으로 다양한 응용분야에 적극 활용될 수 있도록 다양한 노력을 기울여야 할 것입니다.

끝으로 이 사업을 수행할 수 있도록 해주신 관련 관계자 여러분들께 깊은 감사를 드리며, 직접 연구에 참여한 연구원 여러 분들의 노고를 치하합니다.

2011 년 12 월

한국전자통신연구원 원장 김흥남

제 출 문

본 연구보고서는 주요사업인 "초저전력 비동기 프로세서 설계기술 개발에 관한 연구"의 결과로서, 본 과제에 참여한 아래의 연구팀이 작성한 것입니다.

2011 년 12 월

주관연구기관명 : 한국전자통신연구원

주관연구책임자 : 김 성 남

연 구 원 : 김 성 운

오 명 훈

신 치 훈

요 약 문

I. 제 목

초저전력 비동기 프로세서 설계 기술 개발 사업

(Asynchronous Low power Processor Design Technology Development)

II. 연구목적 및 중요성

1) 연구 목적

본 연구는 유비쿼터스 서비스(ubiquitous service) 환경하에서 폭발적인 성장이 예상되는 모바일 기기 및 디바이스에서 핵심적인 역할을 수행하는 프로세서를 비동기식 설계(asynchronous design) 방식을 적용하여 개발하고 이를 통하여 비동기식 설계 기술 및 저전력 프로세서 기술 관련 IPR 을 확보하는 것을 목적으로 한다.

- **최종 목표**

  미래 모바일 기기 및 디바이스를 위한 저전력 소모 및 주변환경 변화에 강인한 프로세서 기술 개발

2) 연구 중요성

프로세서는 대부분의 정보 기기에 필수적으로 사용되고 있으며 '산업의 쌀'로 불리울 만큼 그 중요성이 크지만, 국내에서는 관련 연구개발이 활성화되지 않고 있으며 현재 주요 선진국의 몇몇 기업에서 기술을 독점하고 있다. 프로세서

기술은 개발 난이도가 매우 높고 진입장벽이 높은 기술로서 부가가치가 매우 높고 주변 산업에 미치는 파급효과가 매우 큰 시장 선도형 기술이지만, 국내의 경우 이러한 프로세서 기술을 대부분 도입에 의존하고 있어서 국내 기술 및 제품의 대외 경쟁력이 취약한 상황이어서 독자적인 기술개발 및 확보가 매우 시급하다.

본 사업에서는 비동기식 설계 기반의 저전력 프로세서에 대한 연구개발을 통하여 이를 독자적으로 확보하고 향후 이를 기반으로 다양한 응용분야에 적용하여 적극 활용될 수 있도록 본 연구개발을 추진하고자 한다.

III. 연구내용 및 범위

본 연구개발은 2007 년 1 월 전문연구사업으로서 시작되어 2012 년 12 월까지 총 6 년의 개발 일정으로 추진되고 있으며, 2010 년에는 대과제인 '융복합부품 핵심기술 연구' 대과제의 제 3 세부과제로 편입되어 연구개발이 이루어졌다. 2011 년도는 본 사업의 제 5 차년도 개발 일정으로서, 지난 4 년간의 연구개발 성과를 기반으로 하여 다음과 같은 세부 연구내용 및 범위를 목표로 설정하였다.

- 32 비트 저전력 비동기식 프로세서 코어 성능 개선
  - 프로세서 코어 기능 확장 설계 및 구현
  - 비동기식 프로세서 코어 성능 개선
  - 비동기식 프로세서 성능 검증 환경 구축
- 외부 인터페이스 통합 설계
  - 비동기식 외부 버스 인터페이스 통합 설계
- 비동기식 회로 설계를 위한 요소기술 연구
  - 동기-비동기식 접속장치 기술 연구
  - 비동기식 회로의 FPGA Prototyping 기술 연구
  - 비동기식 회로 합성 기술 분석

2011 년도 본 과제의 세부 성과목표 및 성과지표는 다음과 같이 정의하였다.

세부성과 목표	목표도출 근거	성과지표	당해년도 목표	평가(검증) 방법	배점
초저전력 비동기 프로세서 기술 개발	국내 유관 연구 기관 및 사례가 적어 구현 및 기술 개발 여부가 의미를 가짐	32 비트 저전력 프로세서 코어 RTL 성능 및 동작 검증	o 비동기 프로세서 명령어셋 100% 동작검증 o 비동기 프로세서 코어 40MHz 급 동작속도 구현	o 프로세서 검증 환경을 이용한 시뮬레이션을 통한 명령어셋 동작 검증 o 프로세서 검증 환경을 이용한 시뮬레이션을 통한 프로세서 코어 동작속도 측정	3
		비동기식 회로 설계 요소기술 및 프레임워크 개발	o 비동기식 외부 버스 인터페이스 설계	o 비동기식 외부 버스 인터페이스 RTL 코드 확인	2
		특허출원 (국내/국외)	o 4 건 (2/2)	o 특허 출원 증빙자료 제시	3
		논문게재 (국내/국외)	o 3 건 (2/1)	o 논문게재/발표 실적증빙자료 제시	1
		인력 양성	o 1 건	o 연구개발 참여인력 양성	1
		소계			
결과/효과 지표	국내비동기식 설계기술분야 선도 및 저변 확산	산학연협력 또는 애로기술 지원	o 산학연연구 협력 2 건	o 산학연협력 증빙자료 제시	7.2
		소계			

IV. 연구결과

2011 년도 본 과제의 세부 성과목표 및 성과지표에 대한 당해년도 달성결과를 다음의 표에 요약하였다.

세부성과 목표	성과지표	당해년도 목표	당해년도 성과	주요성과	달성율
초저전력 비동기 프로세서 기술 개발	32 비트 저전력 프로세서 코어 RTL 성능 및 동작 검증	o 명령어셋 100% 검증 o 40MHz 급 동작속도	o 명령어셋 100% 검증 o 41.1MHz 동작속도 구현	-아키텍쳐 개선 (명령어셋 100% 구현, 동작성능 41.1MHz)	100% 달성
	비동기식 회로 설계 요소기술 및 프레임워크 개발	o 1 건	o 2 건	-버스 인터페이스 유닛 설계(보고서, RTL 코드) -비동기식 회로 합성 툴 분석 (보고서)	100% 초과
	특허출원 (국내/국외)	o 4 건 (2/2)	o 5 건 (3/2)	-국내 3 건, 국외 2 건 출원	100% 초과
	논문게제 (국내/국외)	o 3 건 (2/1)	o 4 건 (1/3)	-국내논문 1 편 -국외논문 3 편 (SCI1 건 2011 년 6 월 게제 확정)	100% 초과
	인력 양성	o 1 건	o 2 건	-참여인력 2 인 (UST 학생 1 인, KAIST 위촉 1 인)	100% 초과
결과/효과 지표		o 산학연연구 협력 2 건	o 2 건	-㈜AD 칩스 -한림대컴퓨터공학	100% 달성

V. 기대성과 및 건의

본 연구개발을 통하여 얻을 수 있는 주요 기대성과는 다음과 같다.

- 초저전력 비동기 프로세서 기술개발 사업은 현재 4 차년도 연구개발 일정을 수행중에 있으며, 글로벌 저탄소 녹색성장 추세에서 그린 IT 기술 개발의 일환으로서 중요한 의미를 가짐. 본 연구개발의 수행을 통하여 향후 시장에서 큰 수요가 예상되는 저전력 프로세서 기술 및 관련 IPR (Intellectual Property Right)을 확보하여 관련 기술 분야에서 대외 경쟁력 확보가 가능할 것으로 전망됨
- 비동기식 설계 기반 저전력 프로세서에 대한 연구성과는 기존에 대부분 외국의 기술을 도입하여 사용하던 프로세서 기술을 일정부분 대체가 가능할 것으로 기대되며, 이로 인하여 관련 연구개발 분야 활성화 뿐만 아니라 관련 분야의 산업의 경쟁력 강화에도 기여할 수 있을 것으로 기대됨
- 당해년도에는 이미 확보한 비동기식 설계환경과 검증환경, 그리고 비동기식 프로세서 코어기술을 기반으로 하여 비동기식 프로세서의 외부 환경에 대한 확장기술을 개발하였음. 이를 위하여 AMBA 버스 기반의 외부 버스 인터페이스 기술을 구현하였으며, 비동기식 프로세서 코어와 클럭 기반 동기식 기술이 상호 연동하여 동작할 수 있는 환경을 구축함으로서 향후 본 연구개발 성과의 활용도를 높이고자 하였음
- 본 연구개발을 수행하는데 있어 관련 산학연이 함께 기술 개발을 추진하고 각각의 역할에 맞는 협력을 함으로서 국내의 비동기식 설계 기반을 공동으로 확보하고자 하였음. 이를 통하여 선진국들에 비하여 매우 취약한 국내 비동기 설계기반을 구축하고 비동기식 설계 기반의 저전력 프로세서 설계에 대한 노하우 및 관련 지적재산권을 확보함으로써 대외 선진 기업과의 기술격차를 해소 하는 계기로 활용 가능함

당해년도에 본 연구개발을 수행하면서 느꼈던 애로사항은 다음과 같다.

우선 본 연구개발을 수행하면서 가장 어려웠던 점은 본 연구개발사업의 목표 및 성격에 비추어 연구비 규모가 소규모이고, 연도별 연구개발 예산이 확정되어 있지 않아 사업수행 계획을 수립하고 추진하는데 어려웠다는 점을 들 수 있다. 본 연구개발을 효과적으로 수행하기 위해서는 전체 연구개발 기간 동안 연구비의 안정적인 확보 및 지원이 매우 중요하며, 이에 따라 전체 연구개발 로드맵을 확정하고 연구개발을 수립하고 수행할 수 있지만, 본 연구개발 사업은 이러한 부분에서 어려움이 있었다고 판단되며, 이러한 점이 아쉬움으로 남는다.

두번째는 본 연구개발사업의 성격에 대한 인식 차이로 기인한다. 당초 본 사업은 중장기적인 차원에서 핵심 요소기술 개발을 위한 전문연구사업으로 시작하였는데, 2009 년부터 정부에서 대과제화 정책을 시행하여 대과제에 통합됨으로서 연구개발 수행 목표 설정에 어려움을 겪은 점이다. 당초 본 연구개발사업에서 지향하는 연구개발 목표와 대과제로 통합된 이후에 대과제에서 지향하는 상용화 목표 간에는 상호 차이가 있었는데, 대과제로 통합하면서 이를 효과적으로 잘 반영하기 어려웠다. 또한, 2009 년에 이어 2010 년에는 본 사업의 소속 대과제가 또 변경되어서 이에 따른 새로운 목표설정 및 수립 등으로 인한 행정적인 어려움이 가중되어 연구개발 이외에도 부수적인 어려움이 있었다.

세번째는 정부출연금 사업의 다양한 연구개발 특성을 반영하지 않은 평가 기준이 어려움으로 작용하였다. 당초 본 사업은 소규모의 예산을 활용하여 소규모 인력이 중장기적으로 비동기식 설계 기반 핵심 요소기술을 확보하는 것을 목표로 하였는데, 과제수행 평가시에는 상용화나 기술이전 등 상용화 측면의 요구사항이 많았기 때문에 이를 반영하는데 어려웠다. 또한, 사업의 예산 및 투입인력 규모에 비추어 볼 때 비동기식 프로세서 코어의 반도체 제작을 통한 검증은 예산상 반영되지 않았음에도 불구하고 평가시에는 이러한 점들이 이슈화되어 평가시에 부정적인 요소로 작용하는 등 어려움을 겪은 점을 들 수 있다. 이러한 점에

비추어 볼 때 향후에는 과제 성격에 맞는 평가지표 및 평가 기준의 개발이 매우 중요하다고 판단한다.

상기 기술한 몇 가지 사항이 본 연구개발을 수행하면서 느꼈던 어려운 점이라 판단되었으며, 향후 남은 연구개발년도 동안에는 이러한 점이 어느 정도라도 개선이 되었으면 하는 바람이며, 이를 통하여 안정적이고 예측 가능한 환경에서 본 연구개발을 성공적으로 마무리 짓고 싶다.

ABSTRACT

Ⅰ. TITLE

Ultra Low Power Asynchronous Processor Design Technology Development

Ⅱ. THE OBJECTIVES

The objective of this project is to develop a low powered 32 bit processor based on the asynchronous design technology. In this asynchronous design approach, all signaling between logic blocks are made by handshaking protocols instead of using a global clock signal. This processor can be applicable to various applications such as mobile handheld devices and ubiquitous sensor network devices and so on.

III. THE CONTENTS AND SCOPE OF THE STUDY

The contents and scope of the study are summarized as follows.

- Implementation of an extended version of 32-bit low-power asynchronous processor core
  - Implementation of an extended version of the asynchronous processor core in terms of functionality
  - Improvement of the microarchitecture to enhance the performance of the processor core

- Establishment of a verification environment for the asynchronous processor core

- Design of an External Bus Interface Unit
  - Design of the interface structure between the asynchronous processor core and an external bus

- Research on fundamental technologies on design of asynchronous circuits
  - Research on the interface between synchronous and asynchronous circuits
  - Research on prototyping asynchronous circuits on commercial synchronous FPGA
  - Analysis on CAD tools for synthesizing asynchronous circuits

IV. RESULTS

We designed the asynchronous processor core based on 7-stage pipeline structure. Integrating these modules at the gate level and applying them to timing simulation, we have proved the correctness of functionality of each module. All instructions of EISC architecture have been also verified with the same flow. The estimated maximum performance of our processor core is about 41.1 MHz that is the converted clock frequency according to a synchronous counterpart.

Fully asynchronous cache modules as well as the processor core itself were considered to raise the performance. Comparing with ALTHEA without an instruction and a data cache module, we observed that the performance improvement of ALTHEA integrating with their cache modules is about 46.6 %.

V. EXPECTED RESULT & PROPOSITION

The technology for implementation of asynchronous circuit as a design method of SoC is prevalent in foreign countries such as EU and USA, so it already has been widely used in commercial electronic product. From this project, we acquired the know-how for asynchronous low power circuit designing technology and the related IPRs.

In this project, we are focusing on 32-bit asynchronous processor which is targeted for embedded applications. Even though the needs for the low power processor technology related applications are enormous, we rely on several foreign companies for these solutions. So we have a plan for developing low power asynchronous processor technology and we will transfer the results to domestic companies to increase the competitiveness of the product. Also we convince our results will contribute to the balancing of semiconductor industry of Korea and it will be useful for various application areas.

CONTENTS

CHAPTER 1 Introduction ····· 19

SECTION 1 Research Goal ····· 19

SECTION 2 Necessity of Research ····· 20

SECTION 3 Scope & Contents of Research ····· 22

SECTION 4 Configuration of Document ····· 23

CHAPTER 2 Body ····· 24

SECTION 1 Asynchronous Circuit Design Methodology ····· 24

SECTION 2 Asynchronous Circuit Verification Environment ··· 29

SECTION 3 Research Scope of the Year 2010 ····· 31

SECTION 4 Research Details of the Year 2010 ····· 34

SECTION 5 Problem Analysis & Future Plan ····· 94

CHAPTER 3 Conclusions ····· 96

References ····· 100

Abbreviations ····· 102

목 차

제 1 장 서론 ····· 19

제 1 절 연구개발의 목표 ····· 19

제 2 절 연구개발의 필요성 ····· 20

제 3 절 연구개발의 내용 및 범위 ····· 22

제 4 절 문서의 구성 ····· 23

제 2 장 본론 ····· 24

제 1절 비동기식 설계 환경 ····· 24

제 2절 비동기식 회로 검증 환경 ····· 29

제 3절 당해년도 연구범위 ····· 31

1. 목표 및 연구내용 ····· 31

2. 연구수행 계획 및 일정 ····· 31

제 4절 당해년도 연구내용 ····· 34

1. ALTHEA 프로세서 구조 ····· 34

2. ALTHEA 프로세서 코어 설계 ····· 36

3. 버스 인터페이스 유닛 설계 ····· 77

4. Co-Processor (CP) 유닛 설계 ····· 82

5. ALTHEA 기능 및 성능 분석 ····· 85

제 5절 문제점 분석 및 차년도 연구 수행 방향 ····· 94

제 3 장 결론 ····· 96

참고문헌 ····· 100

약 어 표 ····· 102

표 목 차

<표 1> TiDE의 대표 컴포넌트 26
<표 2> ALTHEA IF 블록의 주요 기능 및 규격 37
<표 3> 각 기능 별 설명과 필요 입출력 39
<표 4> ALTHEA ID 블록의 주요 기능 및 규격 46
<표 5> ALTHEA DE1 블록의 주요 기능 및 규격 51
<표 6> 레지스터 접근 방법 54
<표 7> DD_bp_cmd 채널의 구성 61
<표 8> DD_Op_Sel 채널의 구성 62
<표 9> De_ex_op 채널의 구성 65
<표 10> ALU 모듈의 명령어별 flag 반영 66
<표 11> UNA 모듈의 명령어별 flag 반영 67
<표 12> 포트에 할당되는 입력 72
<표 13> Result1 포트의 입력 인덱스 73
<표 14> RF 블록의 주요 기능 및 규격 73
<표 15> ALTHEA 명령어 분류 85
<표 16> ALTHEA 명령어 구현 및 동작 체크리스트 88
<표 17> 비동기식 프로세서 코어의 면적 정보 91
<표 18> SHA 실행시의 성능 지표 92
<표 19> Qsort 실행시의 전력 소모 지표 92
<표 20> 캐시 연동 모델의 성능 분석 93

그 림 목 차

<그림 1> TiDE를 이용한 설계 흐름 25

<그림 2> HASTE 예제 코드와 핸드쉐이크 회로 27

<그림 3> HASTE 예제 코드의 구현 일부 28

<그림 4> 비동기식 회로의 검증환경 29

<그림 5> 비동기 프로세서의 기능 시뮬레이션 환경 30

<그림 6> ALTHEA 마이크로아키텍쳐 35

<그림 7> IF 블록과 기타 블록과의 관계 37

<그림 8> IF 블록 내부 구성도 38

<그림 9> IF 블록 외부 인터페이스 39

<그림 10> LERI 디코더 42

<그림 11> 선 해독기 43

<그림 12> 순환큐 자료구조 44

<그림 13> ID 블록과 기타 블록과의 관계 46

<그림 14> ID 블록 내부 구성도 47

<그림 15> ID 블록 외부 인터페이스 47

<그림 16> ID 블록 내부 처리 동작 흐름 50

<그림 17> DE1 블록과 기타 블록과의 관계 51

<그림 18> DE1 블록 외부 인터페이스 52

<그림 19> DE1 블록 내부 구성도 53
<그림 20> DE2 블록과 다른 블록과의 관계 57
<그림 21> DE2 블록의 입출력 채널 58
<그림 22> DE2 내부 블록도 60
<그림 23> EX 블록과 다른 블록과의 관계 62
<그림 24> EX 블록의 입출력 채널 63
<그림 25> EX 블록 구조도 64
<그림 26> ME 블록과 다른 블록과의 관계 68
<그림 27> ME 블록의 입출력 채널 69
<그림 28> ME 블록 구조도 71
<그림 29> RF 블록과 기타 블록과의 관계 74
<그림 30> RF 블록 내부 구성도 75
<그림 31> RF 블록 외부 인터페이스 75
<그림 32> 버스 인터페이스 유닛 입출력 80
<그림 33> 버스 인터페이스 유닛 내부 구조 81
<그림 34> CP 블록과 다른 블록과의 관계 82
<그림 35> CP 블록의 입출력 채널 83
<그림 36> CP 블록 구조도 84
<그림 37> CP 블록 관련 명령어별 데이터 흐름도 84
<그림 38> ALTHEA 기능 검증환경 87
<그림 39> ALTHEA FPGA 기능 검증 구성 90

제 1 장 서론

제 1 절 연구개발의 목표

유비쿼터스 서비스 패러다임으로 급격히 전환되고 있는 현 상황에서 각종 정보화 기기들의 수요는 크게 증가하고 있으며 이들은 다음과 같은 방향으로 진화되고 있다.

- 지능형 / 맞춤화
- 정보기기의 융합화
- 개인형 / 이동형
- 에너지 효율성 추구

본 연구개발사업에서는 향후 폭발적으로 수요가 증가할 것으로 전망되는 미래 모바일 기기 및 디바이스를 위한 저전력 프로세서 기술의 개발을 목표로 한다. 이를 위하여 다음과 같은 연구개발 목표를 설정하였다.

- 최종 목표 : 미래 모바일 기기 및 디바이스를 위한 저전력 소모 및 주변환경 변화에 강인한 프로세서 기술 개발

- 세부 목표
  - 저전력 비동기식 프로세서 코어 기술 개발
    - 32bit RISC 프로세서 코어 기술 (RTL code)
    - 75MHz 급 동작속도

- 프로세서 응용 프로그램 개발을 위한 환경 구축

▪ 비동기식 설계 핵심 요소 기술 개발

- 캐시 및 비동기식 버스 인터페이스 기술 개발
- 동기-비동기식 접속장치 기술 개발
- 비동기식 회로설계 지원을 위한 설계 프레임워크 개발

본 연구개발 사업에서는 이러한 연구개발을 통하여 그린 IT 기반 저전력 프로세서 기술에 대한 수요에 부응하고 국내 반도체 산업의 경쟁력 확보에 기여함으로서 향후 큰 성장이 기대되는 미래 휴대/이동 기기등에서 활용 가능한 비동기식 설계기반 프로세서 기술 기반을 확보하고자 한다.

제 2 절 연구개발의 필요성

본 사업은 향후 산업분야에서 큰 수요가 예상되고 있는 저전력 프로세서 기술의 확보를 위하여 중장기적인 관점에서 추진되고 있는 연구개발 사업으로서, 본 연구개발의 필요성 및 중요성은 다음과 같이 요약할 수 있다.

- 저전력 프로세서 기술 및 비동기 설계 기술의 확보를 통한 Green IT 기반 기술 확보 및 유비쿼터스 패러다임에서 큰 수요가 예상되는 u-서비스 기반 모바일 기기를 위한 에너지 효율적인 프로세서 기술의 자체 확보
- 저전력 프로세서 기술의 높은 부가가치와 기술적, 경제적 파급효과로 인한 유관 산업 분야의 동반 성장에 기여하고, 연구개발 성과의 국내 기업 기술이전을 통하여 제품의 차별화 및 대외 경쟁력 향상
- 시장에서 수요가 매우 큰 저전력 프로세서 기술 분야에 대한 전략적인

연구개발을 통하여 날로 치열해지는 글로벌 경쟁환경에서 경쟁력 있는 기술로 시장과 트렌드를 주도할 필요성 증대

- 큰 성장세를 보이고 있는 휴대형/이동형 정보기기에 필요한 핵심 기술의 개발을 통한 경쟁력 확보를 통하여 국내 비메모리 분야의 산업 육성에 일조하고, 이를 통한 관련기술에 대한 수입대체 및 수출증대 효과 기대
- 비동기 설계 기반 프로세서 IP 기술을 활용한 응용기술 개발 및 이를 통한 신규 수요처 발굴 및 신시장 개척에 활용하고, 비동기식 설계 분야의 대외 기술 경쟁력 확보
- 전력 클록에 기반을 두고 설계하는 동기식 설계 방식이 직면하는 문제점인 시간 종결(Timing closure)을 위한 클록 분배로 인한 전력소모 문제 및 전자기파적인 특성 저하, 환경 변화에 민감하여 오동작 가능성 등의 문제점을 해결할 수 있는 방안으로서 핸드세이킹에 기반하여 동작하는 비동기식 설계 기술에 대한 연구개발이 관심을 받고 있어 이에 대한 기술 확보가 필요함

제 3 절 연구개발의 내용 및 범위

본 연구개발 사업에서 추구하고 있는 저전력 프로세서 기술 개발에 내용에 대한 연차별 주요 연구개발 목표 및 내용은 다음과 같다

구 분	목 표	내 용
1 차년도 (2007)	비동기식 설계기술 및 동향 분석 및 설계환경 구축	o 비동기식 설계 기술 동향/특허 동향 분석 o 비동기식 설계 기반 제품 동향 및 요소기술 분석 o 32 비트 저전력 비동기식 프로세서 요구사항 정의 o 비동기식 설계 환경 구축(CAD)
2 차년도 (2008)	저전력 비동기식 프로세서 규격 정의	o 32 비트 저전력 비동기식 프로세서 규격 정의 o 32 비트 저전력 비동기식 프로세서 코어 기능 및 구조 설계
3 차년도 (2009)	저전력 비동기식 프로세서 코어기술 개발	o 32 비트 저전력 비동기식 프로세서 코어 기능 모듈 구현 및 통합검증 o 비동기식 캐시 및 외부 인터페이스 구조 설계 o 프로세서 마이크로아키텍쳐 기능 개선
4 차년도 (2010)	저전력 비동기식 프로세서 확장 기술 개발	o 32 비트 저전력 비동기식 프로세서 코어 성능 개선 o 외부 인터페이스 통합 설계 o 비동기식 회로 요소 기술 분석 및 개발
5 차년도 (2011)	저전력 비동기식 프로세서 통합 및 검증	o 저전력 비동기 프로세서 코어와 주변장치 통합 및 최적화 개발 o 저전력 비동기 프로세서 설계를 위한 비동기식 설계 프레임워크 개발
6 차년도 (2012)	저전력 비동기식 프로세서 안정화	o 저전력 비동기 프로세서 성능 최적화 및 기능 안정화 o 저전력 비동기 프로세서 설계를 위한 비동기식 설계 프레임워크 확장 개발

제 4 절 문서의 구성

본 문서는 초저전력 비동기 프로세서 개발 사업의 제 4 차년도 연구개발에 대한 최종연구개발 보고서로서, 서론, 본론, 그리고 결론의 총 3 장의 내용으로 구성된다.

서론에서는 본 연구사업의 목표와 필요성, 그리고 연차별 연구개발 내용 및 범위에 대하여 기술하였다.

본론에서는 본 연구개발을 수행하는데 있어서 적용한 비동기식 설계 방법과 검증환경을 소개하고, 당해년도 연구개발 범위 및 내용에 대하여 기술하였다. 또한, 당해년도 연구개발 성과에 대하여 분석해 보고 이를 통하여 향후 연구개발 추진 방향을 설정하는데 반영하였다.

마지막으로 3 장 결론 부분에서는 본 연구개발사업의 당해년도 성과에 대한 결론을 맺고, 본 사업을 수행하는데 있어서 애로사항에 대하여 기술하였다. 본 문서의 마지막 부분에는 참고문헌과 본 문서에서 사용된 축약어를 포함하였다.

제 2 장 본론

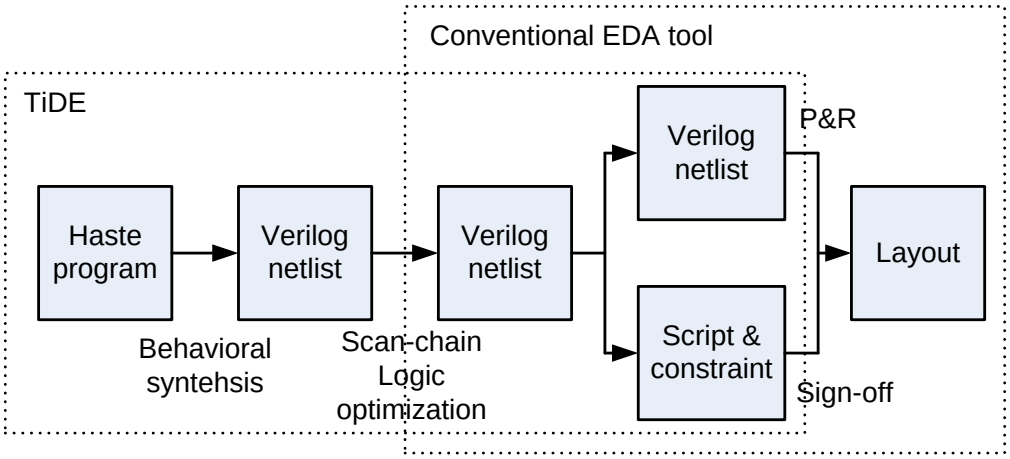
제 1절 비동기식 설계 환경

상위 수준 언어를 기반으로 회로 규격을 기술하는 방법으로는 초기에 CSP (Communicating Sequential Processes) 언어가 개발되었다. 병행 수행을 하는 여러 모듈들을 정의된 문법으로 기술하고 각 모듈들의 통신은 역시 정의된 채널을 통해 수행된다. 기술된 회로 규격은 production rule 이라고 불리는 연산자들로 직접 변환되며, 이들 연산자들은 모두 비동기 회로 셀 라이브러리 형태로 미리 구현되어 있으므로, 합성 시에는 연산자들과 회로들을 직접 mapping 하는 작업을 수행한다. 지금까지 연구된 비동기 회로 관련 상위 수준 언어로는 trace theory, Occam, Tangram, Balsa 등이 있다.

최근에는 Philips 에서 개발된 Tangram 을 기반으로 한 HASTE 를 사용하여 게이트 수준 netlist 를 생성하는 상위 수준 및 다른 EDA 툴에서 ASIC 화 할 수 있는 하위 수준까지 지원하는 TiDE (Timeless Design Environment)가 상용화되었다. 다시 말하면, TiDE 설계 흐름에서는 시스템의 동작을 언어를 통해 기술한 것만으로써 실제 구현까지의 모든 단계를 자동화된 툴에 의해서 진행시킬 수 있음을 의미한다. 이는 설계 시간을 단축시킬 수 있을 뿐만 아니라, 그래프 기반 방식에 비해서 설계자의 비동기 회로의 기반 지식에 덜 의존하므로 비동기식 시스템 구현 시에 초기 접근이 용이하다. 본 과제의 비동기식 프로세서 설계는 이러한 이점이 있는 TiDE 를 기반으로 한다. 다음에서 TiDE 기반 설계 흐름을 설명한다.

<그림 1>은 TiDE 를 이용한 설계 flow 를 나타내고 있다. 행위수준 합성을 통해 HASTE 로 작성된 코드를 합성한 후 최적화 및 scan-chain 삽입 단계부터는 상용 EDA 툴을 사용한다. 그리고 최종 layout 을 위한 스크립트 및 constraint 를 자동 생성하고 이것으로 layout 작업이 수행된다. Logic 최적화 단계 이후는 상용 EDA 툴의 것을 사용하므로, 본 보고서에서는 행위 수준 합성에 대해서 살펴본다.

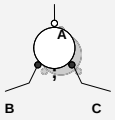
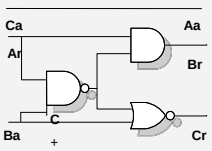
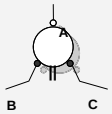
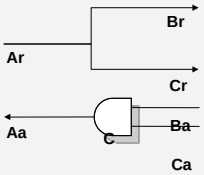
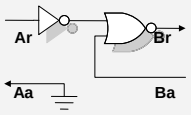
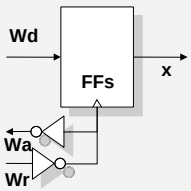
즉, HASTE 로 작성된 코드가 TiDE 를 통해 어떻게 표준 네트리스트로 합성되는지를 설명한다.



<그림 1> TiDE를 이용한 설계 흐름

행위 수준의 합성 과정은 컴파일러가 HASTE 코드를 분석해 핸드쉐이크 회로를 생성하는 단계와 이 핸드쉐이크 회로를 분석하여 표준 셀로 매핑하는 단계로 구성되어 있다. 이 두 단계에서 이미 설계된 컴포넌트(Component)들을 기반으로 한 syntax-directed 컴파일 방식을 지향한다. 즉, 핸드쉐이크 회로를 생성할 때 HASTE 구문 상의 여러 오퍼레이터(Operator)들은 라이브러리에 정의된 컴포넌트들로 대체되고, 이 컴포넌트들은 이미 표준셀로 구현되어 매핑된다. <표 1>은 대표적인 컴포넌트들로써 핸드쉐이크 회로 심볼과 구현 회로, 그리고 컴포넌트 입출력 신호의 발생 순서를 보여준다.

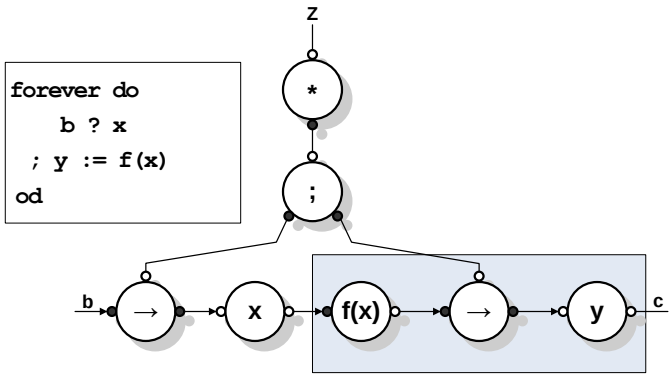
<표 1> TiDE의 대표 컴포넌트

컴포넌트	심볼	구현 회로	동작방식
Sequencer			*[Ar↑;Br↑;Ba↑;Br↓;Ba↓;Cr↑;Ca↑; Aa↑;Ar↓;Cr↓;Ca↓;Aa↓]
Parallel			*[Ar↑;(Br↑;Ba↑||Cr↑;Ca↑);Aa↑;Ar↓; (Br↓;Ba↓||Cr↓;Ca↓);Aa↓]
Repeater			Ar↑;*[Br↑;Ba↑;Br↓;Ba↓]
Flip-Flop variable			*[Wr↑(…);Wa↑;Wr↓(v);x:=v;Wa↓]

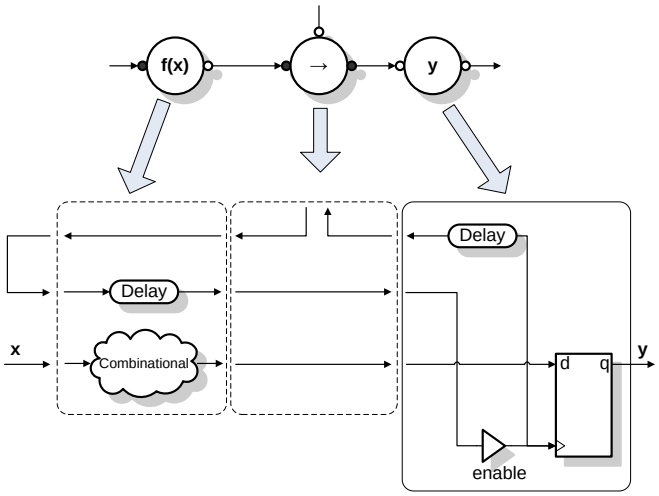
각 컴포넌트의 심볼에는 신호 발생 경로에 따라 회로에 요청을 발생시키는 능동형과 요청을 받는 수동형 포트가 있고, 각각 선 끝에 달려있는 흰색과 검은색으로 마킹된 작은 원으로 표현된다. 즉, 능동형 포트의 신호의 요청에 따라 수동형 포트의 신호가 active 되어 컴포넌트의 고유한 동작을 수행한다. 이때에는 모든 포트는 4-phase signaling 에 기반하여 요구신호 (Ar, Br, Cr)와 응답 신호 (Aa, Ba, Ca)로 핸드쉐이크 프로토콜을 발생시킨다.

Sequencer 컴포넌트는 A 포트의 요청으로 B 포트를 active 시키고 동작이 끝난 후 C 포트를 active 시킨다. 반면 parallel 컴포넌트는 B, C 포트를 동시에

active 시킨다. Repeater 는 A 포트의 요청이후 B 포트가 무한 반복 동작을 수행하고, Filp-Flop variable component 는 실제 표준셀의 플립플롭의 클록 신호를 이용하여 핸드쉐이크 프로토콜을 수행하고 플립플롭의 저장 시간 동안의 지연소자가 필요하다. 이 밖에서도 fork, join, merge, mux, demux 와 같은 컴포넌트들이 있다. <그림 2>는 간단한 HASTE 코드와 컴파일 후의 핸드쉐이크 회로를 나타내고 있다. 이 회로는 forever do 에서 od 까지의 내부 내용을 무한 반복 수행하게 되는데, 오퍼레이터 '?' 는 좌측으로부터 우측으로의 채널 입력을 의미한다. 즉 채널 b 에서 데이터를 읽어 변수 x 에 저장하게 된다. 오퍼레이터 ':=' 는 우측에서 좌측으로 변수 할당을 의미한다. 즉 변수 y 에 기능회로 f(x)의 결과 데이터를 저장하게 된다. 이 회로는 Repeater, Sequencer, variable 컴포넌트 및 기능회로 컴포넌트로 구성된다.



<그림 2> HASTE 예제 코드와 핸드쉐이크 회로



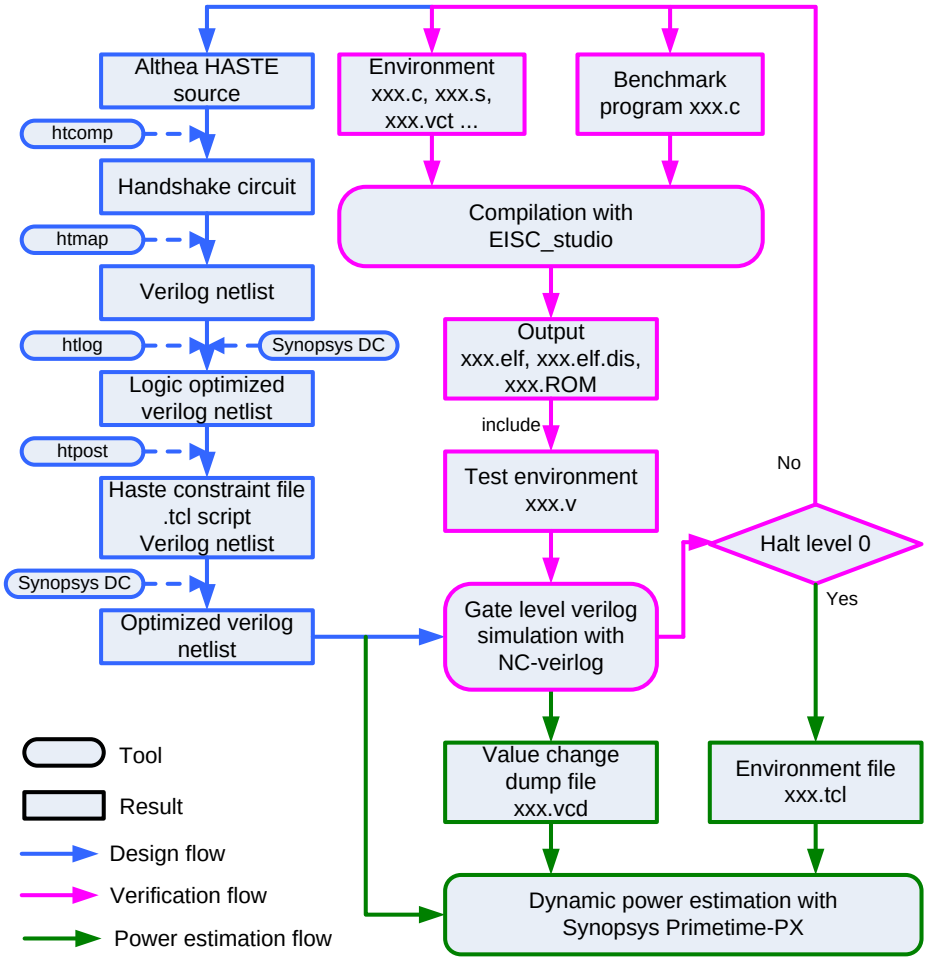
<그림 3> HASTE 예제 코드의 구현 일부

<그림 2>의 박스 안의 핸드쉐이크 회로는 <그림 3>과 같이 매핑되어 구현된다. 기능 회로 컴포넌트는 응답 신호가 가는 시점과 출력단의 데이터가 유효해지는 시점을 동기화시키기 위한 지연소자를 포함한다. 또한 variable 컴포넌트에도 홀드 타임(hold time)을 보장하기 위한 지연소자가 포함된다.

앞서 설명한 설계방법은 모두 4-phase bundled data 방식의 핸드쉐이크 프로토콜에 기반한다. 그러므로, 제어 회로, 특히 STG 에 기반한 설계 방식의 제어 회로가 SI 특성을 갖더라도, 데이터 패스는 BD 특성을 고려하여 설계해야 한다. 그러므로, 데이터 패스의 종료 시점을 위해서는 데이터 패스의 critical path 만큼의 match 지연 소자를 사용해야 한다. 데이터 패스에 DI 혹은 QDI 특성을 적용할 수 있는 DIMS (Delay Insensitive Minterm Synthesis)나 NCL (Null Conventional Logic)를 이용하여 데이터 패스를 구현하는 방법도 있다. 이 방법을 이용하면 앞서 설명한 제어 패스 설계가 훨씬 단순해지는 반면, dual-rail 인코딩에 기반하므로 면적, 전력 소모 측면에서 단점이 있다.

제 2절 비동기식 회로 검증 환경

<그림 4>는 본 과제에 적용된 비동기식 회로 검증 흐름을 나타낸다. 최종 verilog 형식의 netlist 를 출력으로 하는 파란색의 흐름은 설계 흐름을 나타내고 있다.



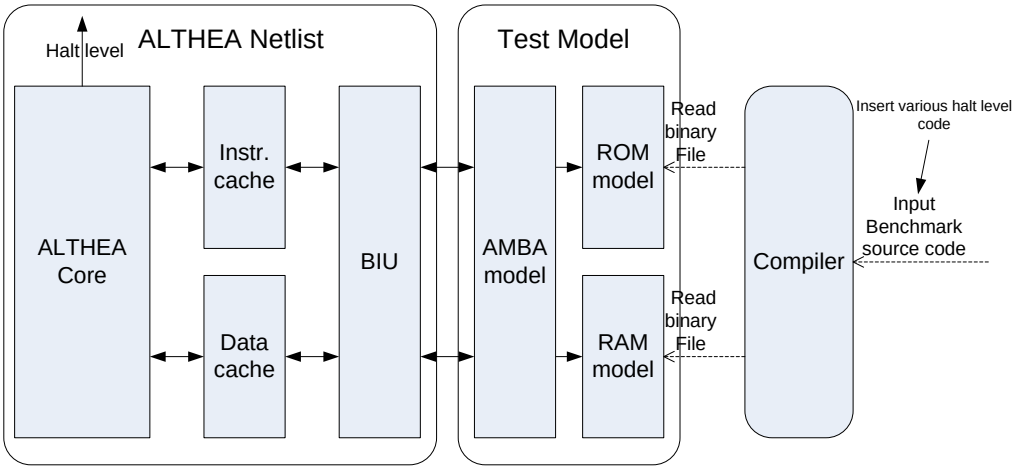
<그림 4> 비동기식 회로의 검증환경

<그림 1>의 TiDE 설계 흐름에서 필요한 각각의 세부 툴(htcomp, htmap,

htlog, htpost)과 기존의 상용툴인 Synopsys DC 가 사용되었다.

분홍색 흐름은 검증 흐름으로 EISC 용 컴파일러와 검증용 벤치마크 프로그램으로 실행 명령어들의 바이너리 ROM 파일을 생성한다. 이후에 Cadence NC-verilog 시뮬레이터로 실행시킨 뒤, 벤치마크 프로그램의 실행 종료 부분에 이식한 HALT0 명령어 코드가 발생되는지를 확인한다.

초록색의 흐름은 전력 측정 흐름으로 검증 흐름에서 확인된 netlist 와 시뮬레이션 시에 생성된 dump 파일을 입력으로 Synopsys Primetime-PX 툴에서 게이트 수준 dynamic 전력을 측정한다.



<그림 5> 비동기 프로세서의 기능 시뮬레이션 환경

<그림 5>는 시뮬레이션 환경을 나타낸다. TiDE 툴을 기본으로 설계된 ALTEHA 코어와 명령어 및 데이터 캐시가 연동되고, 인터페이스 유닛(BIU)은 외부 AMBA 버스와의 동기화를 담당한다. 이 4 모듈의 동작 검증을 위해서 BIU 와 연동하기 위한 AMBA, ROM, RAM 을 모델링하였다. ROM 과 RAM 에는 EISC 컴파일러에서 생성된 벤치마크 프로그램의 바이너리 형식의 파일을 포함한다.

제 3절 당해년도 연구범위

1. 목표 및 연구내용

당해년도인 2010 년은 전체 연구 과제수행 기간 중 4 차년도 일정으로서, 지난 3 년 동안의 연구개발 성과를 바탕으로 하여 비동기식 프로세서의 설계 및 검증을 수행 하였다. 당해년도 본 연구개발을 통하여 이루고자 하는 목표는 저전력 비동기식 프로세서 확장기술 개발이며, 이를 달성하기 위한 주요 세부 연구내용은 다음의 표에 나타낸 바와 같다.

구 분	내 용
당해년도목표	저전력 비동기식 프로세서 확장 기술 개발
연구내용	o 32 비트 저전력 비동기식 프로세서 코어 기능 및 성능개선 o 비동기식 외부 인터페이스 통합 설계 o 비동기식 회로 설계를 위한 요소기술 분석 및 개발

2. 연구수행 계획 및 일정

앞에서 언급한 바와 같이 당해년도 본 사업의 연구개발의 주요 목표는 기존에 확보한 비동기식 프로세서 코어(ALTHEA)를 효과적으로 활용할 수 있도록 하기 위하여 필요한 외부 버스 인터페이스 기술을 개발하는 것이다. 이러한 목표를 당해년도 연구개발 기간 동안에 효과적으로 달성하기 위하여 세부적인 개발 계획을 수립하여 이를 수행하였다. 2010 년의 주요 연구수행 세부 계획 및 일정은 다음에 기술한 바와 같다.

가. 연구수행 계획

당해년도 연구개발 목표 달성을 위하여 다음과 같은 세부 연구개발 계획을 수립하였다. 본 연구개발을 통하여 개발하고자 하는 3 가지 주요 연구에 대한 세부 기술개발 내용은 다음에 기술한 바와 같다.

- 32 비트 저전력 비동기식 프로세서 코어 기능 및 성능 개선
  - 비동기식 프로세서 코어 기능확장 설계 및 구현
  - 비동기식 프로세서 코어 성능 개선 : 40MHz 목표
  - 비동기식 프로세서 코어에 대한 명령어셋 검증 : 100% 검증
  - 비동기식 프로세서 성능 검증환경 구축

- 비동기식 외부 인터페이스 통합 설계
  - 비동기식 외부 버스 인터페이스 설계 및 검증 (RTL code)
  - 비동기식 프로세서 코어와 캐시의 연동 시험

- 비동기식 회로 설계를 위한 요소기술 분석 및 개발
  - 동기-비동기식 접속장치 기술 연구
  - 비동기식 회로의 FPGA Prototyping 기술 연구
  - 비동기식 회로 합성기술 분석 및 적용
  - 비동기식 회로 설계환경(CAD) 분석 및 구축 연구

나. 연구수행 일정

당해년도의 주요 세부 연구개발 목표별 일정은 다음의 표에 나타낸 바와 같다.

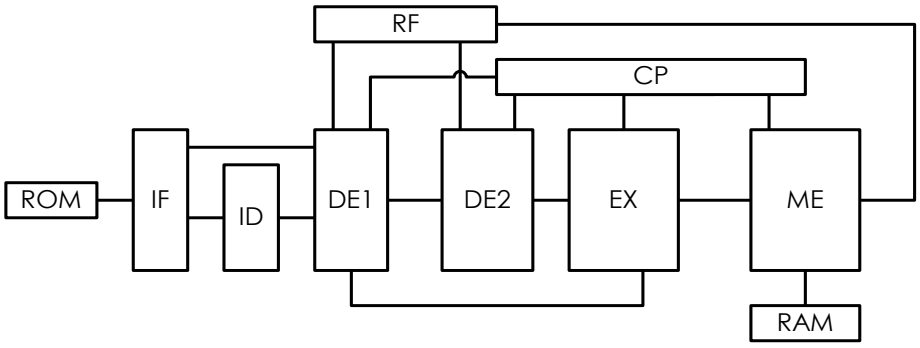
당해년도 연구개발 일정	1	2	3	4	5	6	7	8	9	10	11	12
o 32 비트 비동기식 프로세서 개발												
-프로세서 코어 분석 및 개선안 도출	◀	―	―	▶								
-프로세서 코어 기능/성능 개선 구현					◀	―	―	―	―	▶		
-코어 통합/ 기능 및 성능 검증											◀	▶
o 비동기식 설계 요소 기술 개발												
-AFSM 기반 비동기식 설계환경 구축 연구	◀	―	―	―	―	▶						
-비동기식 회로의 FPGA 프로토타이핑 기술 연구							◀	―	―	―	―	▶
-비동기식 외부 인터페이스 기술 연구		◀	―	―	―	―	―	―	―	―	▶	
o 비동기식 설계 프레임 워크 개발												
-그래프 기반 비동기식 회로 합성 기술 분석		◀	―	―	―	―	―	―	―	―	▶	

제 4절 당해년도 연구내용

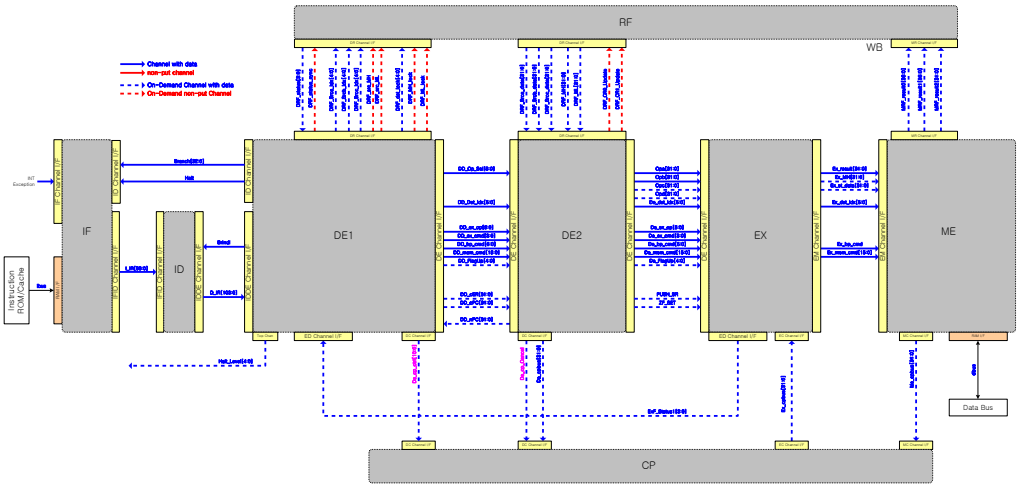
본 절에서는 초저전력 비동기 프로세서(코드명: ALTHEA) 기술 개발의 4 차년도 연구 개발 수행 내용 및 결과에 대하여 설명한다.

1. ALTHEA 프로세서 구조

당해년도는 전체 연구 과제 수행 기간 중 4 차년도로서, 1 차년도는 비동기식 설계 기술에 대한 전반적인 연구를 수행하였으며, 2 차년도에는 이를 바탕으로 비동기식 프로세서의 규격 정의와 함께 요소 블록의 설계를 진행 하였다. 3 차년도에는 설계된 비동기식 프로세서의 개별 요소 블록을 통합하고 기능 및 성능을 검증하였다. 4 차년도에는 3 차년도에 설계된 마이크로아키텍쳐를 개선하여 성능을 높였다. 4 차년도에 개발된 ALTHEA 마이크로아키텍쳐는 <그림 6> 과 같다.



(a) ALTHEA 마이크로아키텍쳐 개념도



(b) 상세 ALTHEA 마이크로아키텍처

<그림 6> ALTHEA 마이크로아키텍처

본 연구 과제를 통하여 개발하고 있는 저전력 비동기식 프로세서인 ALTHEA의 마이크로아키텍처는 앞에서 기술한 바와 같이 프로세서의 기능과 성능의 향상을 위하여 단계적으로 기능 모듈의 개발과 통합, 검증을 통한 개선 과정을 수행하였다. ALTHEA 는 당해년도(4 차년도, 2010 년) 추진 전략과 계획에 따라 ALTHEA 프로세서의 마이크로아키텍처를 개선하였다.

당해년도 연구 개발에 따른 ALTHEA 프로세서 마이크로아키텍처의 특징은 다음과 같다.

- 7 단 비동기식 파이프라인 구조
  - 주 파이프라인을 구성 유닛
    . IF(Instruction Fetcher) : 명령어 인출 및 pre-decode
    . ID(IF-DE1) : 명령어 버퍼 및 Non-RISC 명령어 분해
    . DE1(Decoder1) : 명령어 decode 및 RF 억세스, 포워딩 검출

. DE2(Decoder2) : 오퍼랜드 생성

. EX(Execution) : 명령어 실행

. ME(Memory and Write Back) : 메모리 억세스 및 결과 write-back

- 기타 기능 유닛인 레지스터파일 유닛 RF(Register File)와 코프로세서 유닛 CP(Co-Processor)

- On-demand 비동기식 채널 커뮤니케이션
  - 명령어의 실행시 사용되는(혹은 데이터의 전달이 필요한) 비동기식 커뮤티케이션 채널을 선택적으로 활성화
- Non-RISC 명령어의 복수 단일 명령으로의 분해
  - RISC 형식이 아닌 PUSH/POP 명령어를 복수개의 단일 PUSH/POP 명령으로 분해하여 처리.

2. ALTHEA 프로세서 코어 설계

가. Instruction Fetch (IF) 블록

IF 블록은 메모리의 명령어들을 읽어온 다음, 이 명령어들을 다음 파이프라인 단계인 Decoder(DE) 블록에 전달하는 역할을 기본으로 한다. 그 밖에도 DE 블록의 기능 일부를 가져와 미리 인스트럭션을 분류, 가공하여 DE 블록에서 보다 빠른 처리를 수행할 수 있도록 도와주는 역할도 한다. 명령어 메모리(ROM)를 통해 가져온 명령어는 종류에 따라 분류 및 가공 되고, 즉치값의 유무에 따라 몇 개의 명령어들이 묶어져 다음 단계로 전달되게 된다.

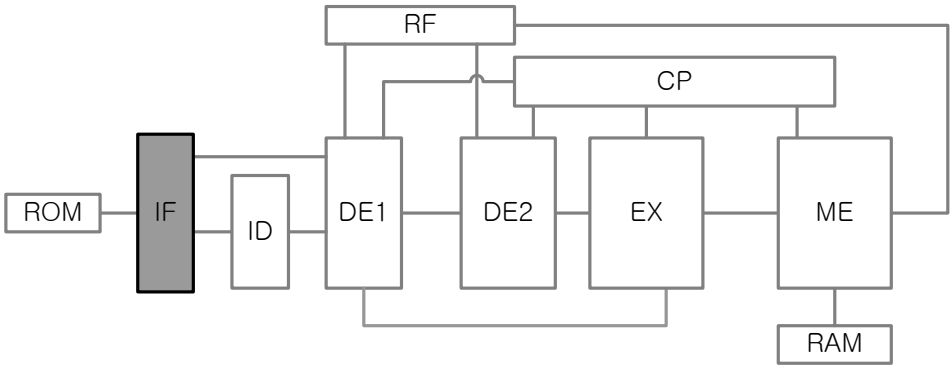
IF 단계는 프로세서 내부의 다른 단계들과 독립적(Decoupled)으로 동작한다. 본 프로세서는 이종으로 구성된 명령어들을 지연 없이 효율적으로 처리하기 위하여, 명령어 패치와 재가공을 담당하는 전위(Front-end) 파이프라인과 명령어의 본격적인 처리를 위한 후위(Back-end) 파이프라인으로 나뉘는데, IF 단계는 이

중 전위 파이프라인에 해당한다.

ALTHEA 프로세서의 IF 블록은 다음의 <표 2>와 같은 기능과 규격을 만족하여야 한다. <그림 7>은 IF 블록과 기타 블록과의 연결관계를 나타낸 그림이다.

<표 2> ALTHEA IF 블록의 주요 기능 및 규격

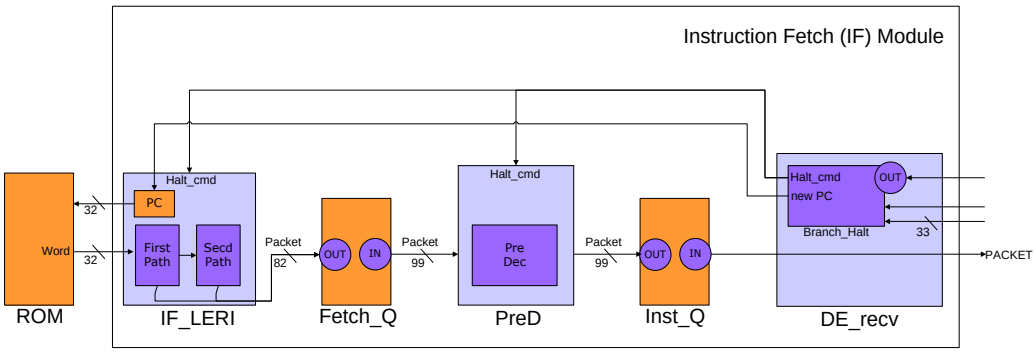
기능	규격
명령어셋 지원	DSP 명령을 제외한 AE32000c 프로세서 명령어 100 종
기능	명령어인출 및 선해독
동작 방식	Clockless 비동기식 Handshaking
인터페이스 블록	ID, DE1



<그림 7> IF 블록과 기타 블록과의 관계

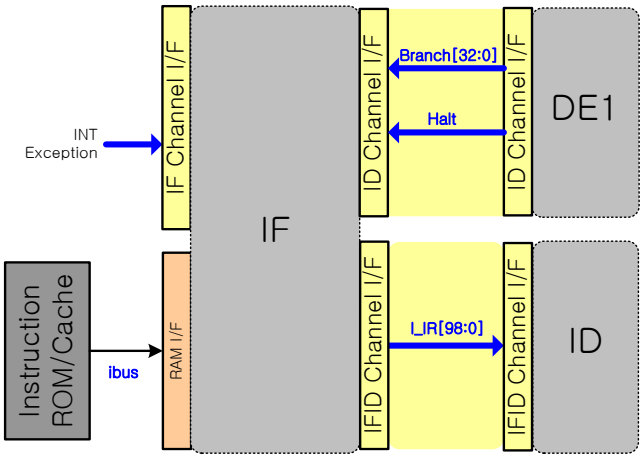
◆ **IF 블록의 구조:** <그림 8>은 IF 블록의 여러 기능 블록을 개념적으로 정리한 블록도 이다. IF 블록은 내부에 개념적으로 다음과 같은 기능 모듈로 구성된다.

- IF_LERI 패치 및 LERI폴딩 모듈: 명령어 처리 순서에 따라 명령어 인출과 LERI 폴딩
- Fetch_Q 패치큐 모듈: 임시 명령어 패킷을 저장
- PreD 선 해독기 모듈: 임시 명령어를 불러와 선 해독 하여 최종 패킷을 생성
- Inst_Q 명령어 큐 모듈: Decoder 파이프라인에 의해 인출
- De_recv 분기 처리 모듈: Decoder 파이프라인으로부터 분기 명령 등을 컨트롤



<그림 8> IF 블록 내부 구성도

<그림 9>는 IF 블록의 외부 채널 인터페이스에 대한 그림이다.



<그림 9> IF 블록 외부 인터페이스

다음 <표 3>은 IF 단계의 각 기능 블록 별 설명과 필요한 입출력 신호들을 정리한 것이다. 필요 출력은 각 기능들을 수행하기 위해 생성되어야 하는 최소한의 출력 물들로, 다음 단계로 전달되는 외부 출력과 내부의 자료구조 혹은 다른 기능블록으로 전달되는 내부출력으로 구분된다.

<표 3> 각 기능 별 설명과 필요 입출력

기능 블록	설명	필요출력	구분	출력설명
명령어 패치	명령어를 메모리로부터 읽어옴	Address Out	외부	명령어의 주소
		read	외부	read 신호
		new PC	내부	업데이트할 PC 값
명령어 가공	전처리를 통해 명령어를 타입에 따라 분류, 가공	Instruction	내부	명령어
		ER	내부	즉치값
		PC	내부	현재 명령어의 주소
		Control	내부	기타 컨트롤 정보

큐 삽입	가공된 패킷을 큐에 저장	Packet	내부	명령어패킷 – Instruction, ER, PC, Control 정보를 묶는 패킷 생성
		Queue Push	내부	생성된 패킷을 큐에 삽입
명령어 패킷 전달	만들어진 Packet 을 다음 단계인 후위 파이프라인의 Decode (DE) 블록으로 전달	packet	외부	다음 단계에 전달할 패킷
		q_fetch	내부	큐에 대한 패치 신호
분기 및 인터럽트 처리	Decode 블록으로부터 분기 관련 제어이나 외부 모듈로부터 인터럽트가 발생시	Flush	내부	큐에 대한 Flush 신호
		Branch Target Address	내부	Branch taken 되었을 때 PC 를 업데이트 하기 위한 출력

◆ **IF & LERI Folding (IF_LERI)**: 외부 메모리(ROM)으로부터 읽어오는 데이터는 32 비트 폭을 가지며 두 개의 명령어를 포함하고 있다. 따라서 <그림 8>의 IF_LERI 블록의 내부에서 보는 바와 같이, 명령어 쌍의 첫 번째 명령어를 처리하는 블록과 두 번째 명령어를 처리하는 블록이 따로 필요 (First Path, Second Path)하고 이들은 의존성을 가지기 때문에 순차적으로 연결하는 것이 편리하다. 다만, 순차적 처리 시 속도가 느려지는 단점이 있으므로 차후 이 블록의 성능 개선이 필요할 시, 병렬 처리가 가능하도록 고안할 수 있을 것이다. 이 경우 의존성 해소를 위한 부가적인 하드웨어가 필수 불가결하다.

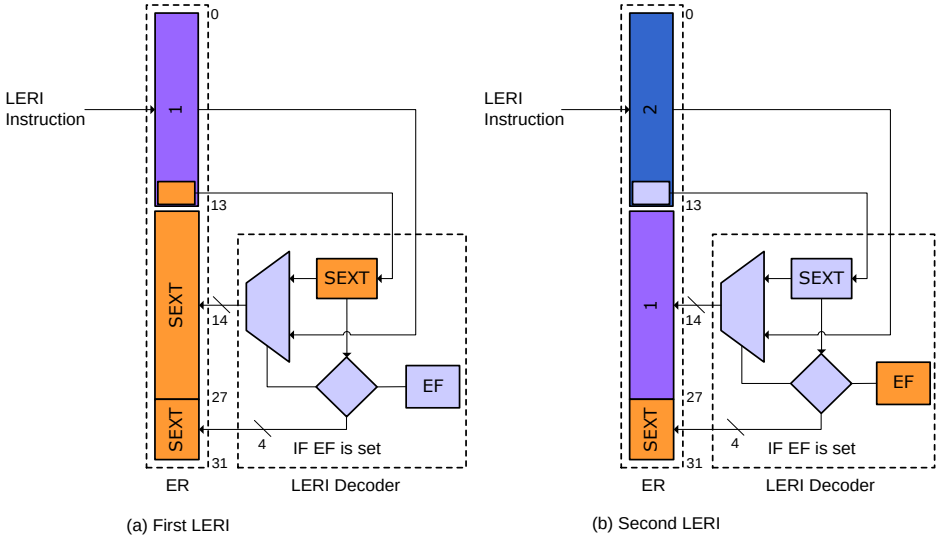
ROM 으로부터 읽어 오는 명령어는 크게 일반 명령어와 LERI 명령어 두 가지 타입으로 나뉠 수 있다.. 따라서 IF_LERI 모듈 내의 각각의 명령어 처리 블록은 일반 명령어와 LERI 명령어를 구별하여 처리 하는 기능을 가지고 있다.

일반 명령어를 처리 하는 것은 상대적으로 간단하다. 명령어를 그대로 패치 큐에 저장하면 되기 때문이다. 다만, 문맥 상 선행해서 들어온 LERI 명령어를 포함하게 되므로, 만약 먼저 들어온 LERI 명령어가 존재한다면, LERI 명령어를 쌓아둔 ER 레지스터 값을 함께 큐에 저장하여야 한다.

만약 ROM 을 통해 들어온 명령어가 LERI 명령어 라면 LERI 명령어를 해석하여 ER 에 저장한다. LERI 명령어에는 헤더 2 비트를 제외한 14 비트의 데이터가 있고 두 개의 LERI 명령어가 연속으로 들어올 수 있다. 첫 번째 LERI 명령어가 들어오면 ER 에 그냥 저장되지만 연이어 두 번째 LERI 명령어가 들어올 경우 ER 을 14 비트 밀어낸 후 두 번째 명령어의 14 비트를 저장한다.

<그림 10>은 LERI 디코더의 구조 및 동작을 나타낸 그림이다. <그림 10>의 (a)는 첫 번째 LERI 명령어가 들어왔을 경우를 나타낸 그림인데, 첫 번째 LERI 의 즉치값은 ER 의 하위 14 비트에 저장된다. 나머지 18 비트는 LERI 의 즉치값에 대한 부호 확장된 값이 채워지게 된다. 이때 채워지는 18 비트 중 하위 14 비트는 두 번째 LERI 가 들어올 때 첫 번째 값이 복사되는 자리이므로 multiplexer 를 이용하여 선택할 수 있도록 하였다.

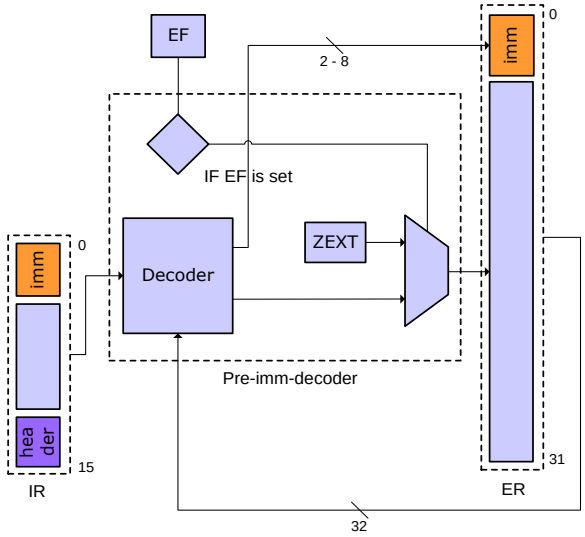
또한 선택의 기준인 LERI 명령어의 기 처리 여부를 저장하기 위해 별도의 플래그 비트가 필요한데 그림에서는 EF 로 표시되어 있다. 첫 번째 LERI 가 저장되고 부호 확장이 완료되면 EF 를 설정하여 두 번째 명령어가 올 때 부호 확장 된 값이 아닌 첫 번째에 들어왔던 14 비트가 복사될 수 있도록 구성되어 있다. <그림 10> (b)는 첫 번째 명령어가 저장된 후에 두 번째 명령어가 들어올 경우 일어나는 동작을 설명한 그림이다. EF 가 설정 되어 있으므로 처음 들어왔던 하위 14 비트는 다음 14 비트로 복사되고 최상위 4 비트는 값을 유지한다.



<그림 10> LERI 디코더

◆ **선 해독 (Pre-decoding)**: 선 해독기는 ER flag (EF)을 참고하여, 명령어의 IMM 을 읽어와 ER 을 완성하는 동작을 수행한다. ER flag 가 설정되어 있으면 일반 명령어의 전에 LERI 명령어가 읽혀졌음을 의미하므로 명령어 내의 IMM 의 크기에 따라 ER 을 시프트 한 후 ER 의 최하위 비트에 IMM 을 넣어주어 ER 을 완성시킨다. 반대로 ER flag 가 설정되어 있지 않으면 IMM 을 최 하위비트에 넣어주고 이외의 비트는 0 으로 채워준다.

<그림 11>은 선 해독기의 구성 및 동작을 보여주는 그림이다. 먼저 명령어의 IMM 을 추출하기 위해 내부의 Decoder 가 명령어를 해석한다. 추출된 IMM 의 크기에 따라 나머지 비트가 Shifter 나 ZEXT (Zero Extension)에 의해 채워지게 되고, IMM 은 그림의 맨 오른쪽 ER 의 최하위 비트에 쓰이게 된다. 한가지 주의할 점은 Shifting 하는 동작과 IMM 을 ER 에 써넣는 동작 사이의 순서관계 인데, Shifting 하는 동작이 IMM 을 쓰는 동작보다 빠르거나 같아야 ER 에 유효한 값이 저장된다.



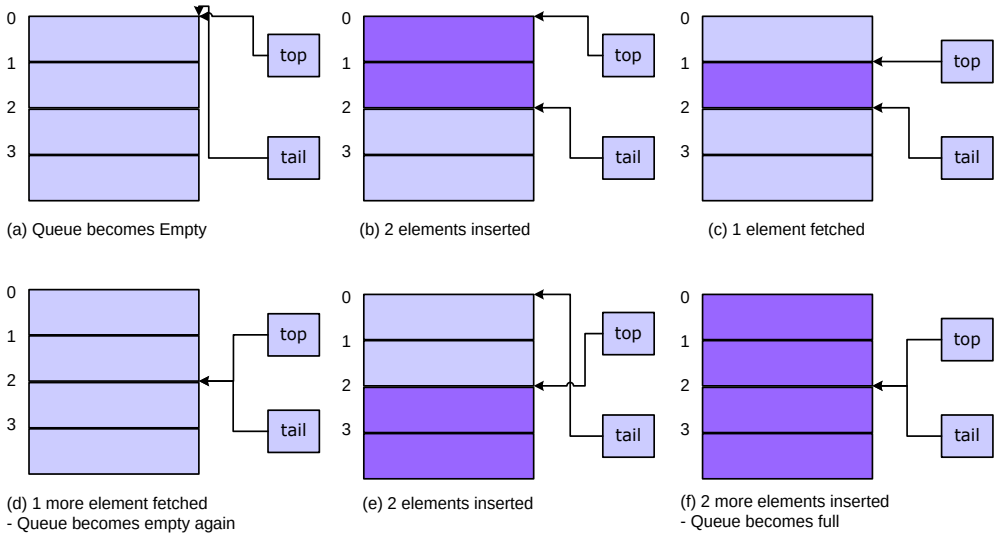
<그림 11> 선 해독기

명령어와 ER 은 명령어의 종류와 EF 비트에 따라, 선 해독기에 의해 여러 종류로 해독되어 ER 레지스터 값에 변경 저장되게 된다.

◆ **큐 관리:** IF 단계의 내부에 존재하는 큐는 명령어, ER, PC, Control 등을 다음 스테이지로 전달하는 과정에서 버퍼로서 기능한다. PC, IR 과 같은 다른 내부자료 구조와는 다르게 큐는 여러 가지 동작을 수행하게끔 요구 받으므로 자체적인 관리 메커니즘을 가지고 있어야 한다.

큐를 구성하는 방식은 FIFO, 순환큐와 같은 여러 가지 구현이 있을 수 있는데 본 프로세서에서는 순환큐를 채용한다. 순환큐는 큐에 대한 입출력이 일어날 때 shifting 이 일어나지 않고 포인터의 조작으로 큐의 입출력을 조절한다. Flush 와 같은 reset 동작도 큐를 비우는 실제적인 동작 없이 포인터를 초기화함으로써 해결할 수 있다. 순환큐의 읽기 동작과 쓰기 동작은 상호 비간섭적이어야 한다. 읽기와 쓰기 모두 동일한 포인터 값을 가감하기 때문에 쓰기와 읽기를 동시에 수행하게 될 경우 에러를 유발할 수 있다.

순환큐는 큐의 깊이가 늘어날수록 포인터관리의 복잡도가 증가하고 큐의 폭이 작을 경우 포인터 조작 자체가 오버헤드로 작용할 수 있다는 단점을 가지고 있으나, IF 단계에서 쓰일 큐는 깊이가 4 단정도로 작고 명령어, ER, PC, Control 등을 포함하는 큐의 폭이 상대적으로 넓기 때문에 순환큐의 단점은 어느 정도 커버된다. 더구나 큐의 개별 요소간의 shifting 이 일어나지 않으므로 파워 측면에서 유리하다고 할 수 있다. 다음의 <그림 12>는 순환큐의 자료구조를 나타낸다.



<그림 12> 순환큐 자료구조

◆ **명령어 패킷 전달 및 분기, 정지 명령 처리:** IF 모듈은 DE 모듈에 완성된 명령어 패킷을 전달해 주고, DE 모듈은 IF 모듈에 명령어를 해석하여 결과에 따라 분기 혹은 정지 명령을 내릴 수 있다.

먼저, DE 모듈에서 명령어 패킷을 요구할 경우, IF 모듈은 명령어 큐에 저장된 명령어 패킷을 인출하여 DE 모듈에 전달한다. 그리고 DE 모듈로부터 분기 명령을 받을 경우 De_recv 블록의 Halt_cmd 를 통해 큐를 포함한 각 모듈의 동작을 정지 시키고, 큐 등의 자료 구조를 초기화 한 후, New_PC 를 통해 분기 주소를 업

데이트 하고, 다시 모듈들의 동작을 재개시킨다. 정지 명령의 경우 모듈들의 동작을 재개시키지 않는 것만 제외하고 분기 명령을 처리할 때와 동작이 같다. 상기 두 기능은 의존성을 가지므로 상호 비간섭적으로 동작하여야 한다. 이 두 기능은 큐 등 동일한 자료구조를 공유하므로 동시에 수행하게 될 경우 큐가 제대로 초기화 되지 않는 등의 에러를 유발할 수 있다.

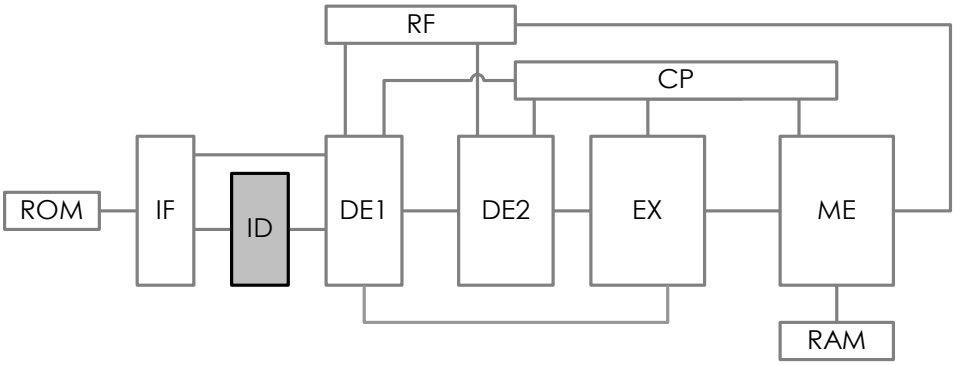
나. IF-DE1 (ID) 블록

ID 블록의 주요 기능 중 하나는 IF 블록을 통하여 인출된 명령어를 전달 받아, 이를 명령어의 해독을 수행하는 DE1 블록으로 전달하는 역할을 수행한다. ID 블록의 또 다른 기능은, 다중 사이클 명령어인 PUSH 와 POP 명령어의 분해 및 분해된 PUSH 와 POP 명령어와 관련된 제어 사이클을 DE1 블록으로 전달하는 일이다.

초기 설계의 디코더 블록은 다중 사이클 명령어인 PUSH 와 POP 명령어의 분해 및 실행을 디코더 내부에서 수행하였다. 이를 위하여 디코더에서는 여타 명령의 처리와 PUSH 와 POP 명령어의 다중 사이클 수행을 위한 상태머신을 관리함으로 인하여 디코더의 효율이 낮아지는 단점이 발생하였다. 이와 같은 문제점의 해결을 위해 디코더와 IF 블록의 중간에 버퍼 기능과 PUSH 와 POP 명령어의 분해 및 제어를 담당하는 ID 블록을 삽입하여 디코더에서의 성능 저하를 방지하였다. ALTHEA 프로세서는 명령어 인출과 해독 과정의 사이에 존재하는 버퍼로 인하여 de-coupling 되어있으므로 새로이 삽입된 ID 블록은 기존 버퍼의 depth 를 1 증가시키는 효과와 함께 다중 사이클 명령어를 효과적으로 처리하도록 설계한다. 다음의 <표 4>와 <그림 13>은 각각 ID 블록의 주요 기능과 기타 블록과의 관계를 나타낸 것이다.

<표 4> ALTHEA ID 블록의 주요 기능 및 규격

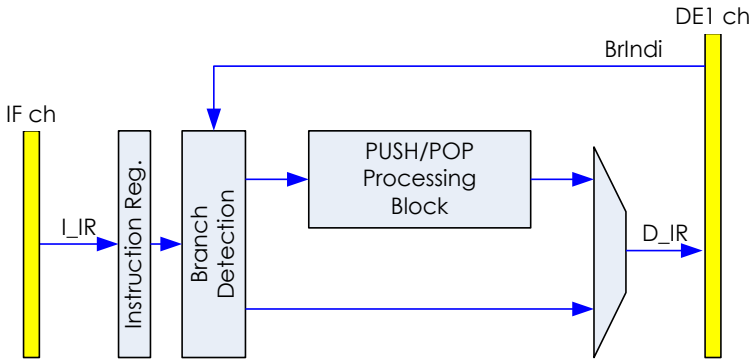
기능	규격
기능	IF 인출 명령어의 DE1 전달 및 다중 사이클 명령 분해
인터페이스 블록	IF, DE1 블록



<그림 13> ID 블록과 기타 블록과의 관계

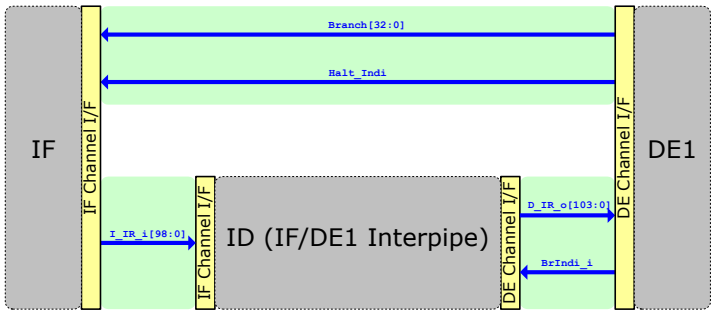
◆ **ID 블록의 구조:** <그림 14>는 ID 블록의 여러 기능 블록을 개념적으로 정리한 블록도이다. ID 블록은 내부에 개념적으로 다음과 같은 기능 모듈로 구성된다.

- 명령어 레지스터 : IF로부터 전달 받은 명령어의 저장
- 분기 검출 모듈 : ID 블록에서 전달한 이전 명령어가 분기가 발생할 경우를 검출
- PUSH/POP 처리 모듈 : IF로부터 전달 받은 명령어가 PUSH/POP 명령일 경우, 다중 사이클 처리. PUSH/POP의 개별 레지스터 리스트를 1개 단위의 PUSH/POP으로 분해하여 DE1으로 전송



<그림 15>는 ID 블록의 외부 채널 인터페이스에 대한 그림이다.

<그림 14> ID 블록 내부 구성도



<그림 15> ID 블록 외부 인터페이스

◆ **ID 블록의 동작:** ID 블록은 전술한 바와 같이 일반 명령의 단순 전달과 다중 사이클 PUSH/POP 의 분해를 통한 복수개의 단일 사이클 PUSH/POP 명령으로의 변환 및 제어 처리 기능이다. ID 블록에서의 명령어 처리과정은 다음 <그림 16>과 같은 절차에 따라 수행한다. 각 단계에 대한 동작 및 처리 절차는 다음의 순서와 같이 정의된다.

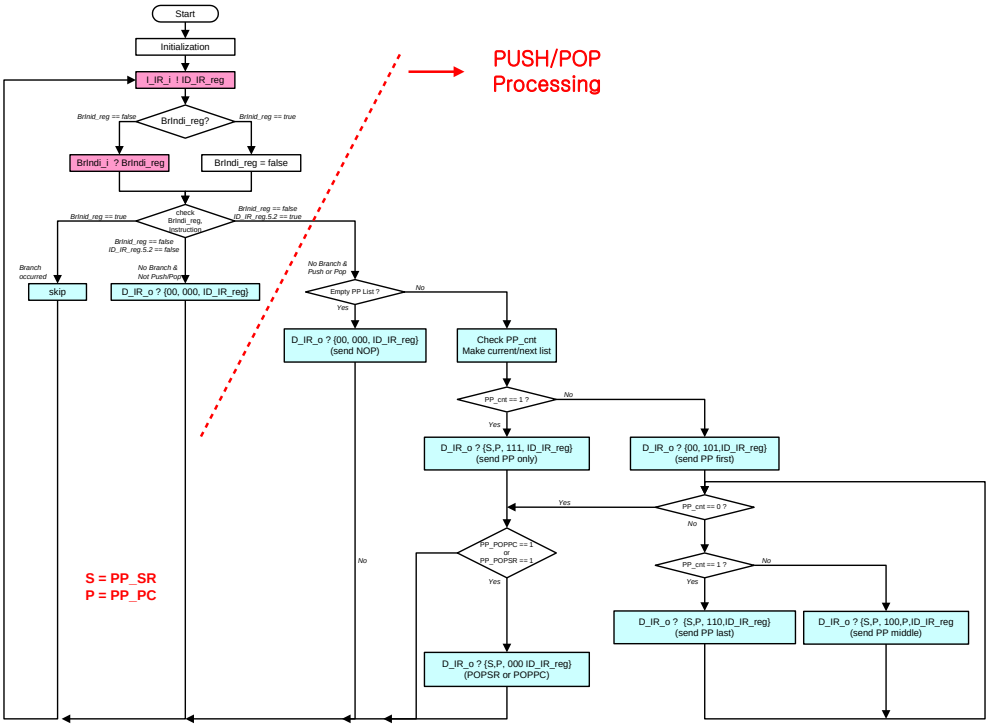
0. 리셋 및 채널 초기화
   A. 이 단계에서는 리셋 이후 내부 변수에 대한 초기화 과정을 수행한다.
1. I_IR 채널을 통한 명령어 인출
   A. IF로부터 명령어를 I_IR 채널을 통해 입력 받아, 내부 레지스터인 ID_IR_reg에 저장
2. 이전 명령에 의한 분기 상태 확인
   A. BrIndi_reg의 내용을 확인하여 이전에 분기 명령으로 인하여, 분기하였는지를 확인
   B. 분기가 발생 하였을 경우는 현재 명령이 분기 이후의 새로운 명령이므로, BrIndi_reg의 분기 기록을 갱신
   C. 분기가 발생하지 않았을 경우, 새로이 BrIndi 채널을 통하여 보내놓은 명령의 분기 수행여부를 확인
3. 분기 확인
   A. BrIndi_reg를 확인하여 분기가 발생하였는지를 확인
   B. 분기가 발생할 경우 현재 읽은 ID_IR_reg의 명령을 DE1으로 전달하지 않고 drop
   C. 분기가 발생하지 않았을 경우, 현재 읽은 명령에 대한 PUSH/POP 여부를 확인
      i. PUSH/POP 명령이 아닌 명령의 경우는 직접 DE1으로 명령을 전달
      ii. PUSH/POP 명령인 경우, PUSH/POP의 해독을 통한 다중 사이클을 분해를 수행
4. PUSH/POP 다중 사이클 분해
   A. PUSH/POP 명령의 경우 대상이 되는 Bank의 대상 레지스터 리스트를 검사
      i. 대상 레지스터의 개수, 대상 레지스터 종류 확인

B. 대상 레지스터의 개수 만큼 PUSH/POP 명령을 분할 하여 개별 대상 레지스터에 대한 PUSH/POP 수행 형태로 변경
   i. 이때 다중 사이클의 경우, PUSH/POP의 시작, 중간, 종료의 확인을 위하여 D_IR 채널 출력에 s_ena, s_fst, s_lst 비트 필드를 설정
   ii. 명령어와 함께 PUSH/POP의 정보를 D_IR 채널을 통하여 DE1으로 전달

C. 개별 대상 레지스터에 대한 PUSH/POP 명령으로 분해시, %SR과 %PC에 대한 POP이 발생하였는지를 확인
   i. POP %SR과 POP %PC의 경우는 각각 새로운 플래그 갱신, PC 값의 갱신에 따른 분기 처리를 필요로 하므로, POP 명령의 처리가 종료된 후 이들에 대한 특별한 처리 사이클을 발생하여 오동작을 방지
   ii. POP %SR의 처리를 위하여 D_IR 채널 출력의 PP_SR 필드와 POP %PC의 처리를 위하여 PP_PC 필드를 설정하며, 이때 전송하는 명령은 NOP 명령어로 설정하여 DE1으로 전송

D. 마지막 단계는 읽어온 RF의 데이터를 해당 오퍼랜드 값으로 매핑 및 EX전달 명령어의 최종 생성을 수행

5. 모든 명령의 처리가 종료되면 1의 단계부터 처리를 재개



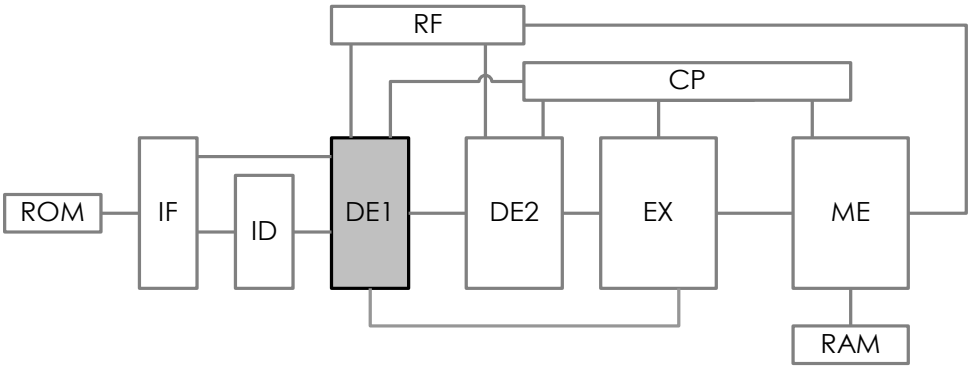
<그림 16> ID 블록 내부 처리 동작 흐름

다. Decoder1 (DE1) 블록

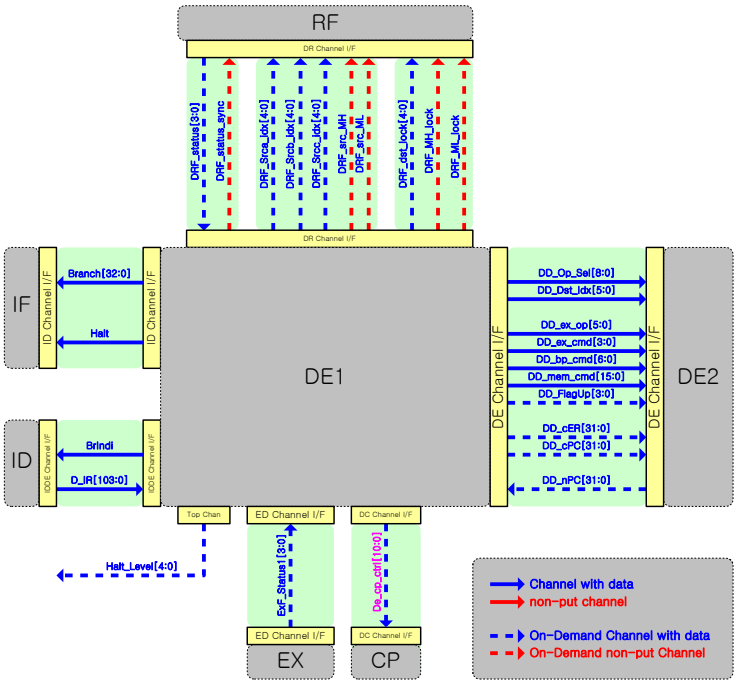
DE1 블록은 IF/ID 블록을 통하여 인출된 명령어를 전달 받아, 이를 실제 수행을 위하여 명령어에 대한 해독과 필요한 리소스(operand)를 인출 정보를 생성하여 이를 DE2 및 이후의 EX/ME 블록으로 전달하는 역할을 수행한다. DE1 은 ADChips 사의 AE32000c 프로세서 ISA 에 정의된 명령어 중 DSP 지원 명령어를 제외한 총 100 개의 명령어를 지원하여야 한다. ALTHEA 프로세서의 DE1 블록은 다음의 <표 5>와 같은 기능과 규격을 만족하여야 한다. <그림 17>은 DE1 과 기타 블록과의 연결관계를 나타낸 그림이다. <그림 18>은 DE1 블록의 외부 채널 인터페이스를 나타낸다.

<표 5> ALTHEA DE1 블록의 주요 기능 및 규격

기능	규격
명령어셋 지원	DSP 명령을 제외한 AE32000c 프로세서 명령어 100 종
기능	명령어셋의 해독
동작 방식	Clockless 비동기식 Handshaking
인터페이스 블록	IF, ID, DE2, EX, RF, CP 블록



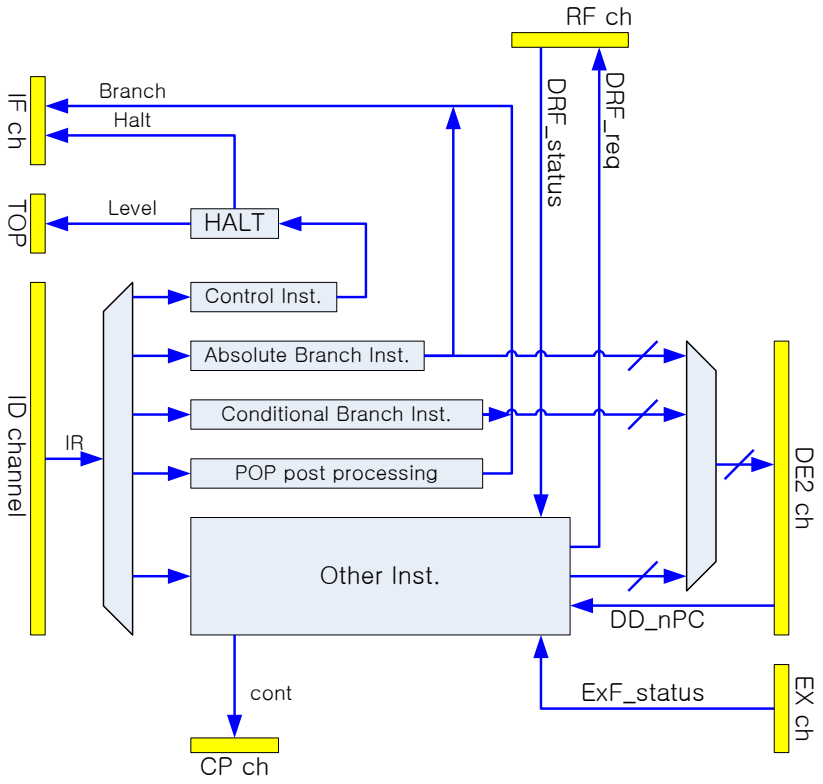
<그림 17> DE1 블록과 기타 블록과의 관계



<그림 18> DE1 블록 외부 인터페이스

<그림 19>는 DE1 블록의 여러 기능 블록을 개념적으로 정리한 블록도 이다. DE1 블록은 내부에 개념적으로 다음과 같은 기능 모듈로 구성된다.

- 제어명령 처리 모듈 : NOP, HALT등 제어 명령에 대한 해독 및 처리
- 절대 분기 처리 모듈 : 절대 분기 명령에 대한 분기 처리
- 조건 분기 처리 모듈 : 조건 분기 명령에 대한 분기 처리
- POP 후처리 모듈 : POP %SR/%PC의 후처리 모듈
- 기타 명령 처리 모듈 : 나머지 명령의 해독 및 처리 모듈
- 포워딩 검출 및 관리 모듈 : 데이터 포워딩 처리 및 테이블 관리



<그림 19> DE1 블록 내부 구성도

◆ **Decode1 모듈:** ID 채널로부터 명령어(IR)와 부분 디코드 정보(cont_signal)를 받아 DE1 블록 내부 및 외부 제어 신호를 생성한다. Branch 모듈로는 목적지 주소값 계산을 위해 offset 정보를 전달하고, Cp_if 모듈로는 코프로세서 레지스터 인덱스(idx), 코프로세서 번호(num), 각종 제어 신호(con)를 전달한다. HALT 명령어 처리를 위해서는 level 신호를 발생시킨다.

오퍼랜드 페치에 관련되어 최대 2 개의 소스 레지스터 인덱스(idx0, idx1)와 목적지 레지스터 인덱스(dst_idx)를 생성한다. WB 단계에서 쓰기 동작 수행 시 dst_idx 신호로 특정 레지스터에 lock 를 걸 수 있으며(dlock), 마찬가지로 status

비트를 변경하거나, ML, MH 레지스터에 쓰기 동작을 수행하는 명령어를 위해서 slock, mlock 신호로 잠금 기능을 수행한다.

이외에도 EX 블록에서 사용될 신호를 생성한다. 즉, Status 비트를 사용하는 연산 명령어를 위한 제어 신호(flag)와, 연산 종류(opcode)와 연산에 필요한 제어 신호(ex_cmd), 쉬프트 량(sht_cnt), EX, ME 단계에서 bypass 되는 명령어를 위한 bp_cmd, 메모리 단계의 수행을 위해 인코딩된 정보를 담은 mem_cmd 신호를 발생시킨다.

<표 6>에는 레지스터 파일을 접근할 때, 인덱스를 사용하는 레지스터와 개별 접근하는 레지스터를 요약하였다.

<표 6> 레지스터 접근 방법

접근 방법	인덱스 번호	레지스터
인덱스	0 ~ 15	R0 - R15
	17	CP0
	18	CP1
	20	SP
	23	SR
개별 접근	MH, ML, Flag	

- 제어명령 처리 모듈 : NOP, HALT등 제어 명령에 대한 해독 및 처리
- 절대 분기 처리 모듈 : 절대 분기 명령에 대한 분기 처리
- 조건 분기 처리 모듈 : 조건 분기 명령에 대한 분기 처리
- POP 후처리 모듈 : POP %SR/%PC의 후처리 모듈
- 기타 명령 처리 모듈 : 나머지 명령의 해독 및 처리 모듈

◆ **포워딩 검출 및 관리 모듈:** 오퍼랜드 페치 및 분기 명령어의 taken 판단 시에

소스 레지스터나 status 값을 WB 단계에서 lock 이 풀린 이후에 접근하는 것은 성능을 저하시킬 수 있다. 그러므로, 7 단 파이프라인 구조에서 EX 블록과 ME 블록에서의 연산 결과 값과 bypass 되는 MH, status 레지스터 값을 미리 사용할 수 있게 감지하는 기능을 수행한다.

포워딩의 검출은 목적 레지스터의 히스토리의 관리를 통하여, EX 단의 결과 데이터가 어느 목적 레지스터로 저장되는 지를 관리한다. DE2 에서는 총 4 개의 오퍼랜드의 생성이 가능하나, 모든 오퍼랜드에 대하여 히스토리를 관리할 경우 테이블의 크기와 검출알고리즘의 복잡도가 증가하여 효과적인 데이터 포워딩의 관리가 어려워 진다. ALTHEA 에서는 데이터의 사용 빈도가 가장 높은 Operand A 에 대하여 데이터 포워딩을 수행하도록 설계함으로써, 성능 향상과 구조의 복잡도를 최소화 하였다.

포워딩 검출 및 관리 모듈에서는 다음과 같은 자료구조를 관리한다.

- Token_Info : Source A에 대한 포워딩 여부를 결정하는 자료구조. 데이터를 포워딩 받거나 RF를 억세스 하기 위한 선택 데잍터를 저장
- FWDT_TKN_Rxx : 레지스터 파일의 각 레지스터에 대한 토큰 정보를 저장하는 자료구조, 해당되는 레지스터의 Token_Info를 저장
- FWDT_D2_Rxx : DE2 단에서의 포워딩 처리를 위한 포워딩 테이블. DE2에서 실행되고 있는 명령어의 해당 레지스터가 목적 레지스터로 사용되고 있는지의 여부를 저장(xx는 레지스터 번호)
- FWDT_EX_Rxx : EX 단에서의 포워딩 처리를 위한 포워딩 테이블. EX에서 실행되고 있는 명령어의 해당 레지스터가 목적 레지스터로 사용되고 있는지의 여부를 저장

◆ **절대 분기 처리 모듈:** 절대 분기 처리 모듈은 분기의 조건 판단이 필요치 않은 분기명령을 처리한다. 분기 명령어 처리시 DE2, EX 로 부터의 파이프라인 및 명령 처리 결과 상태를 필요치 않으므로 파이프라인의 동작 중지 없이 즉각적으로 분기 명령의 처리가 가능하다.

또한, 절대 분기 명령의 처리에 필요한 모든 동작이 DE1 안에서 처리 가능함으로써, DE2 이후의 파이프라인으로 불필요한 명령처리를 전달하지 않음으로써(on-demand 채널동작으로 구현), 파이프라인의 성능을 향상 시킬 수 있다.

◆ **조건 분기 처리 모듈:** 조건 분기의 처리는 EX 단의 연산 결과의 상태 플래그를 기반으로 분기의 여부를 결정하여야 한다. 따라서 조건 분기 명령어를 인출하였을 경우 이전 명령어가 EX 단으로 진행하여 연산 처리가 완료 되어야 한다. 이때 이전 명령어는 DE2 단에서 오퍼랜드 인출을 수행하고 있으므로, DE2 단으로 NOP 처리를 지시하여 DE2 의 명령을 EX 단으로 전달하도록 하며, DE1 단계에서 명령의 처리를 대기한다. EX 단으로 전달된 명령의 처리가 완료되면, 상태 플래그값을 포워딩 받아 이를 바탕으로 분기 명령을 실행한다.

◆ **제어명령 처리 모듈 모듈:** 제어명령 처리 모듈은 프로세서 코어의 기능을 정지시키는 HALT 명령어 혹은 SYNC, NOP 등과 같은 명령의 수행을 처리하는 모듈이다. HALT 의 경우는 HALT 레벨에 따라 프로세서의 동작을 정지시키고 HALT 레벨 값을 외부로 출력하는 동작을 수행하며, 기타 제어명령의 경우는 해당되는 처리 명령을 DE2 로 전달한다.

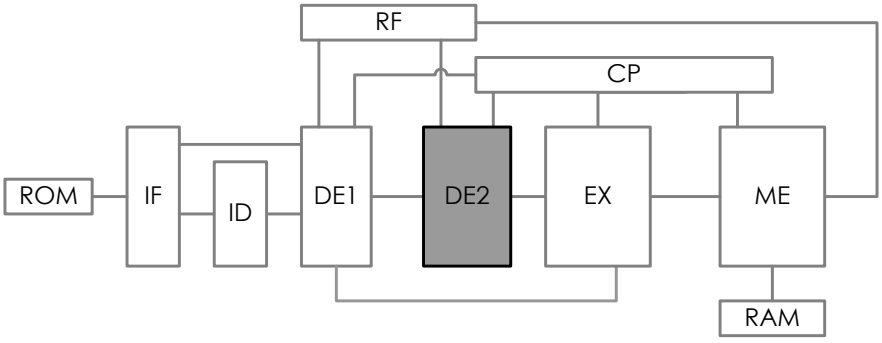
◆ **POP 후처리 모듈:** POP 후처리 모듈은 ID 단계에서 다중 PUSH/POP 명령어가 복수개의 개별 PUSH/POP 으로 분할 되어 전달 될 때 POP SR 혹은 POP PC 명

령어의 후처리를 위한 모듈이다. 일반적인 POP 명령과 달리 POP PC/SR 의 경우는 PC 의 변경(즉 새로운 분기처리, SR 의 변경(즉 RF 모듈에 대한 억세스)을 수반하게 됨으로 일반 명령의 처리와 달리, 해당되는 레지스터 값의 변경에 따른 후속 처리를 담당하게 된다.

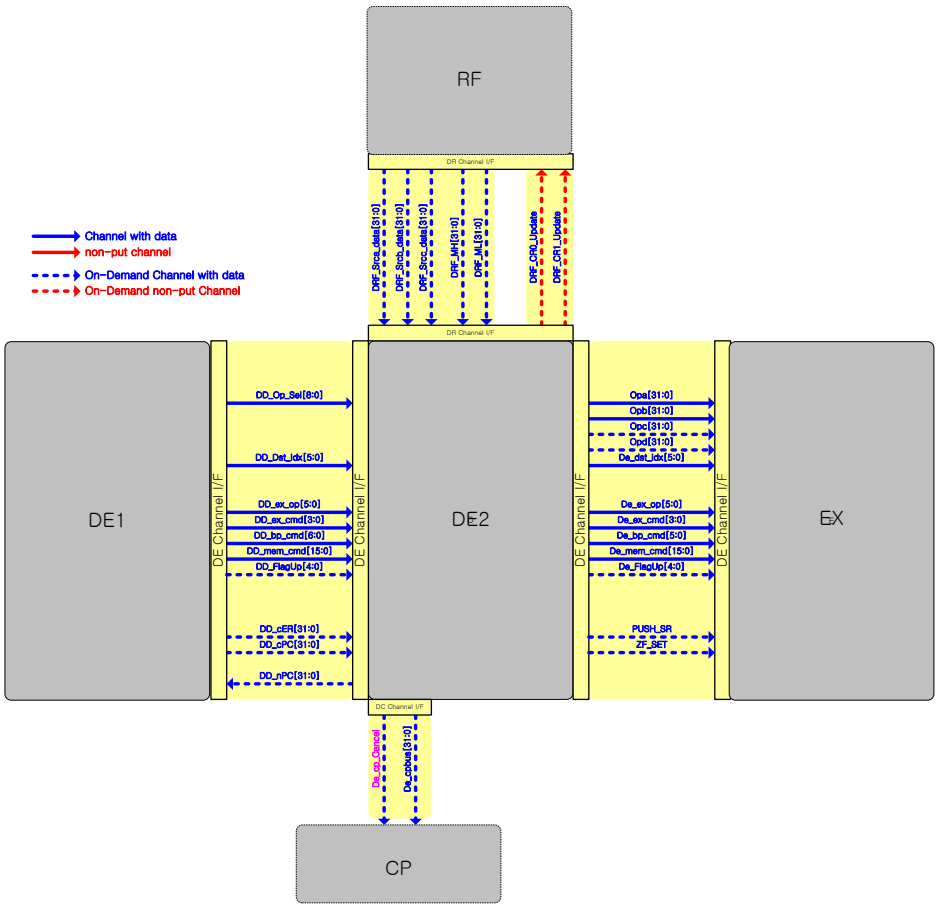
◆ **기타 명령 처리 모듈:** 상기 모듈에서 처리되는 명령어를 제외한 모든 일반 명령어는 기타 명령 처리 모듈에서 처리된다. 이 모듈에서는 일반 명령어의 처리를 한 후속 파이프라인의 동작 제어 명령어, 오퍼랜드 생성 등의 정보를 생성하여 DE2 단으로 전달한다.

라. Decoder2 (DE2) 블록

DE2 블록은 주로 RF 블록 혹은 DE1 블록으로부터의 데이터를 입력 받아 DE1 블록에서의 선택 제어 신호에 의해 EX 블록 수행에 필요한 4 개의 소스 오퍼랜드를 결정한다. 또한 분기 명령어 중에서 레지스터를 참조하는 경우에 분기 명령어의 분기 주소를 DE1 에 전달하기도 하며, CP 블록에 필요한 데이터도 생성한다. <그림 20>은 DE2 블록과 다른 블록과의 관계를 나타내며, <그림 21>은 DE1 블록의 입력 출력 채널을 보여주고 있다.



<그림 20> DE2 블록과 다른 블록과의 관계



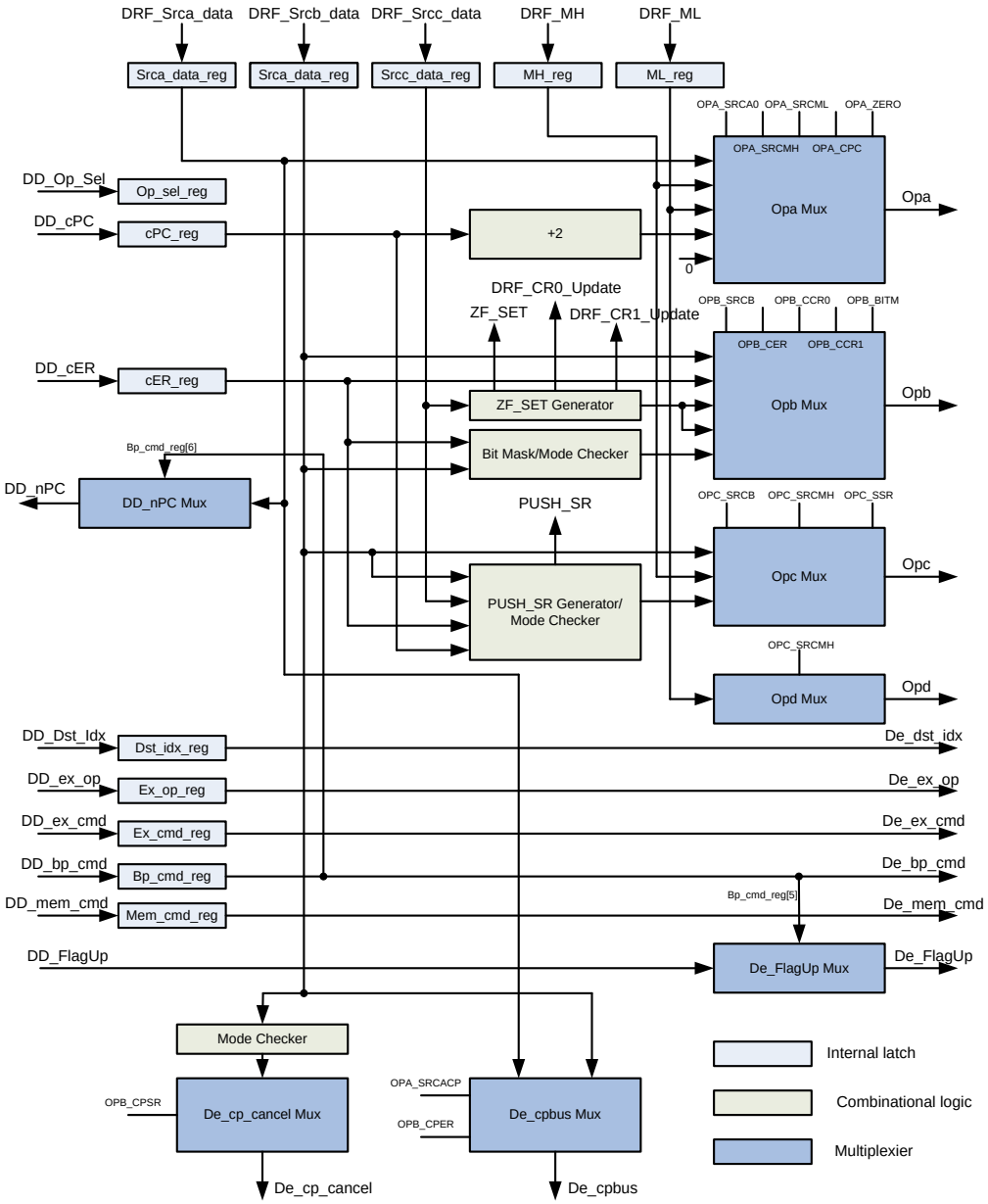
<그림 21> DE2 블록의 입출력 채널

각 component 의 기능은 <그림 22>의 DE2 내부 블록도와 같이 나타낼 수 있다. 입력 채널을 저장하기 위한 "_reg"로 표기된 내부 래치와 "_Mux"로 표기된 출력 채널의 선택 및 on-demand 출력을 제어하는 multiplexer, 그리고 내부 연산을 위한 각종 조합 회로로 구성된다.

DE2 블록의 채널 중 파이프라인 수행, 즉, EX, ME 블록에서 사용할 제어 신호 정보를 담은 De_ex_op, De_dst_idx, De_ex_cmd, De_bp_cmd, De_mem_cmd 는 DE1 블록의 입력이 거의 그대로 전달되며, 나머지 출력 채널들은 on-demand 방식의 출력을 위해 입력 제어 신호 (DD_Op_Sel, DD_bp_cmd) 의 영향을 받는다.

먼저 on-demand 특성이 always 인 DD_Op_Sel 채널 신호를 입력 받아 그림 2 에서 제시된 값에 따라 각 Mux 블록에서 출력해야할 입력 값을 선택한다. 이때에는 RF (DRF_Srca_data, DRF_Srcb_data, DRF_Srdc_data, DRF_MH, DRF_ML) 혹은 DE1 (DD_cPC, DD_cER) 채널을 on-demand 형식으로 입력받고, 필요한 경우에는 조합 회로의 결과를 출력한다. 다음은 각 조합 회로에 대한 설명이다.

- +2 : 무조건 분기 명령어 중 LR 에 리턴 주소 값을 저장하는 경우에 현재 PC 값에 "+2" 연산을 수행한다.
- ZF_SET_Generator : LADU, STAU 명령어 처리 시에 필요한 CR0, CR1 레지스터의 변경을 위해 DRF_CR0_Update, DRF_CR1_Update 채널 출력을 담당하고, DRF_Srcc_data로부터 입력받은 CR0, CR1 레지스터의 비트 필드를 확인하여 zero flag 변경 여부 신호 (ZF_SET)를 출력한다. 또한, 오퍼랜드 B를 통해 CR0, CR1 레지스터의 CNT 정보를 EX 블록으로 전달한다.
- Bit Mask/Mode Checker : SET, CLR 명령어 처리를 위해 DRF_Srcb_data 로 입력 받은 SR의 모드 비트를 확인하여 user 모드의 경우 오퍼랜드 B로 ER의 하위 8 비트를, 다른 경우는 ER의 모든 비트를 EX 블록에 전달한다.



<그림 22> DE2 내부 블록도

- PUSH SR Generator/Mode Checker : PUSH 명령어인 경우 DRF_Srcb_data로 메모리에 저장할 데이터를 입력 받는다. 일단 DRF_Srcc_data의 값, 즉, SR의 모드 비트를 확인하고 special 레지스터의 경우, user 모드, supervisor 모드에 따라 오퍼랜드 C로 출력한다. PUSH SR 의 경우에는 PUSH_SR 채널을 출력 시킨다.
- Mode Checker : SR의 모드를 체크하여 supervisor 모드인 경우 DE1에서 발생시킨 CP 관련 명령어를 취소시키는 채널 신호 (De_cp_Cancel)를 발생시킨다.

다음에서 DE2 블록의 핵심 제어채널인 DD_Op_Sel 과 DD_bp_cmd 를 설명한다. DE2 에서는 이들 두 채널의 bit 정보를 디코딩하여 모든 operation 을 결정한다.

<표 7> DD_bp_cmd 채널의 구성

bit	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	BR	POPSR	CBT	BPC	BPB	BPA	MBP

<표 7>은 DD_bp_cmd 채널의 구성을 나타내고 있고, 이들 component 중 BR, POPSR component 가 DE2 블록에서 사용된다. 다음에서 두 component 의 의미를 설명한다.

- BR은 레지스터 indirect branch를 의미하며 세트시에 DD_nPC로 RF에서의 데이터를 전송하여 PC 값을 저장한다.
- POPSR은 POP SR 명령어 처리시에 flag를 update 해야 하므로 이때 새 flag 값을 DD_FlagUp 채널을 통해 DE1 블록으로부터 입력 받아 EX 블록에 전달한다.

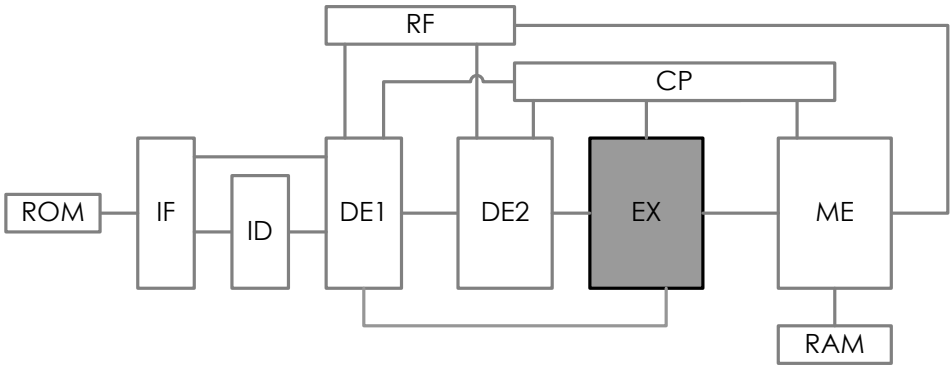
<표 8> DD_Op_Sel 채널의 구성

bit	[8:6]	[5:3]	[2:1]	[0]
	Opa_sel	Opb_sel	Opc_sel	Opd_sel

<표 8>은 DD_Op_Sel 채널을 나타내고 있다. 각 component 에 할당된 값은 DE2 블록의 모든 출력 채널을 제어하는 신호로 사용된다. 다음 표에는 명령어별로 DD_Op_sel 과 DD_bp_cmd 채널의 비트할당과 DE2 블록에서의 동작을 의사코드로 표현하였다.

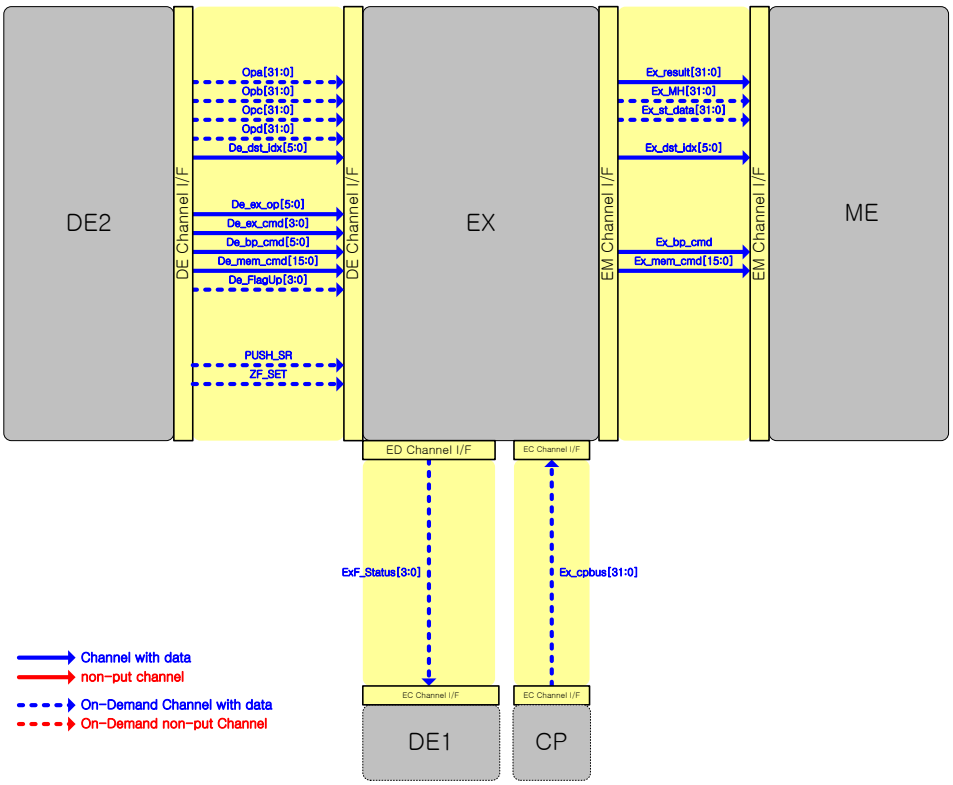
마. Execution (EX)블록

EX 블록은 7 단 파이프라인 중 EX 단을 실행시키는 블록으로 주로 DE2 블록으로부터 소스 오퍼랜드를 입력 받아 연산을 수행하고 결과값을 ME 블록으로 전달하는 역할을 수행한다. 다만, DE2 블록 이외에 CP 블록에서 데이터를 입력 받을 수 있고 연산 수행 결과를 ME 블록 이외에도 DE1 블록으로 전달할 수도 있다. <그림 23>은 EX 블록과 기타 다른 블록과의 관계를 나타낸다.



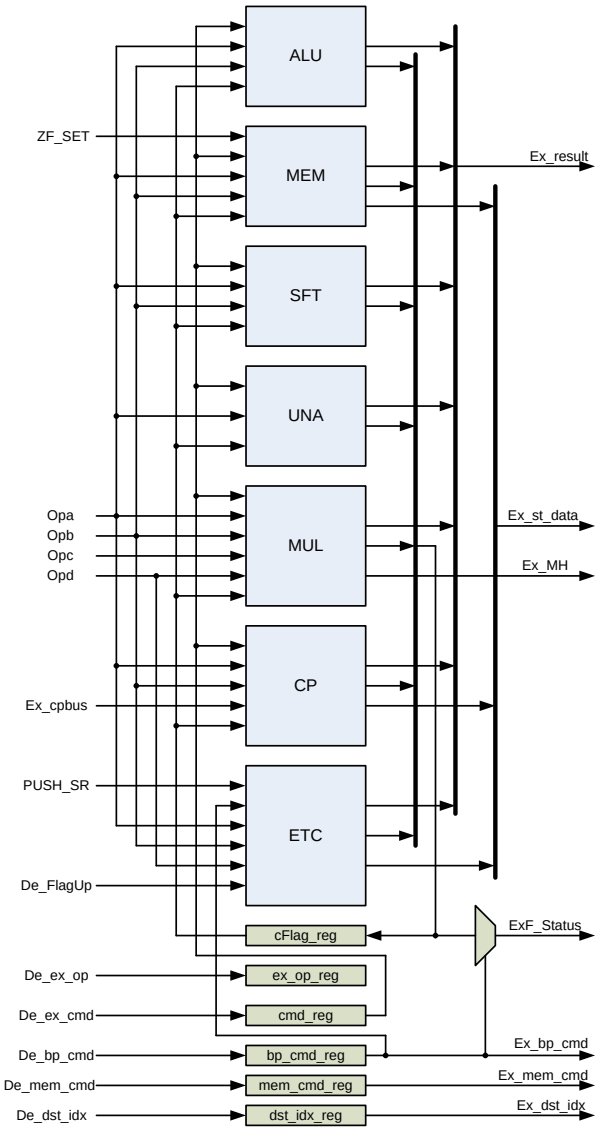
<그림 23> EX 블록과 다른 블록과의 관계

<그림 24>는 EX 블록의 입력 출력 채널을 보여주고 있다.



<그림 24> EX 블록의 입출력 채널

De_ex_op 채널은 EX 블록에서 수행되는 명령어들을 크게 ALU, memory, shift, unary, multiply, co-processor, 그리고 연산이 아닌 bypass 기능을 수행하는 7 가지 그룹으로 나뉘어 수행한다. 이들 명령어 그룹들은 수행되는 연산 종류가 다를 뿐만 아니라, 명령어 그룹에 따라 연산 후 발생되는 3 가지 결과 신호 Ex_result, Ex_st_data, Ex_MH 채널의 출력 저장 형태도 달라진다.



<그림 25> EX 블록 구조도

<그림 25>는 EX 내부 블록도를 나타낸다. TOP 수준에서의 입력 채널의 latch를 “_reg” 이름으로 표시하였고 앞서 설명한 모듈들은 ex_op_reg, 즉 De_ex_op 채널로 오직 한 개의 모듈이 선택된다. 각 모듈들은 cmd_reg,

bp_cmd_reg 의 제어 신호에 따라 기능 수행에 필요한 입력들을 on-demand 방식으로 입력 채널로부터 전달 받는다. 연산 결과 생기는 flag 값들은 현재 flag 를 저장하는 cFlag_reg 에 저장된다. De_bp_cmd, De_mem_cmd, De_dst_idx 채널은 ME 블록으로 bypass 되며, 현재 flag 를 ExF_Status 를 통해 DE1 블록으로 전달할 수도 있다.

다음에서 EX 블록의 핵심 제어 채널인 De_ex_op 와 De_ex_cmd, De_bp_cmd 를 설명한다. EX 블록에서는 이들 채널의 bit 정보를 디코딩하여 모든 operation 을 결정한다.

<표 9> De_ex_op 채널의 구성

Bit	[5]	[4]	[3]	[2]	[1]	[0]
	CP	MUL	UNA	SFT	MEM	ALU

<표 9>는 De_ex_op 채널의 구성을 나타내며, 각 비트 할당에 따라 EX 블록에서 처리해야 할 명령어의 그룹핑 정보를 일차적으로 알 수 있다. 각 그룹별 설명은 다음과 같다

- ALU : shift 계열 명령어를 제외한 두 개의 오퍼랜드로 연산을 수행하는 명령어 그룹
- MEM : PUSH/POP을 제외한 메모리 접근 명령어와 LEA 명령어의 effective address 계산을 위한 명령어 그룹
- SFT : shift 계열 명령어
- UNA : 한 개의 오퍼랜드로 연산을 수행하는 명령어 그룹
- MUL : multiplier를 사용하는 명령어 그룹
- CP : 코프로세서 명령어 처리를 위한 그룹
- ETC: PUSH/POP을 위한 메모리 어드레스 생성 및 관리

◆ **ALU 모듈:** De_ex_op[0] 신호가 세팅되었을 때 구동되는 모듈이다. 두 소스 오퍼랜드 Opa, Opb 로 연산을 수행하며 연산의 종류는 De_ex_cmd 정보에 기반한다. 실제로 각 연산을 위해 가산기, 감산기 및 각종 논리 연산기가 구현되었다. 연산 시에 사용될 flag 값을 가진 cFlag_reg 정보가 입력된다. <표 10>은 ALU 모듈에서 명령어에 따라 반영되어야 할 flag 비트를 요약하고 있으며, 'C' 는 carry, 'Z' 는 zero, 'S' 는 sign, 'V' 는 overflow 를 의미한다.

<표 10> ALU 모듈의 명령어별 flag 반영

종류	ADD	ADC	SUB	SBC	ADDQ	NEG	CMPQ	AND	OR	SET	XOR	CLR	CMP	TST
flag	CZSV	CZSV	CZSV	CZSV	CZSV	-	ZS	ZS	ZS	해당 비트	ZS	해당 비트	CZSV	ZS

◆ **MEM 모듈:** De_ex_op[1] 신호에 의해 구동되며, 메모리 참조 명령어의 메모리 주소 값을 계산한다. 내부적으로는 두 소스 오퍼랜드인 Opa, Opb 를 이용한 32 비트 가산 연산이 이루어진다. 또한, 메모리 참조 명령어 중 LDAU, STAU 의 처리 시에 zero flag 반영 여부를 나타내는 ZF_SET 채널을 입력 받아 c_Flag_reg 에 최신의 flag 값을 반영한다.

◆ **SFT 모듈:** De_ex_op[2] 신호에 의해 구동되며, 쉬프트 명령어의 연산을 수행한다. 소스 오퍼랜드 Opa 의 값을 Opb[4:0] 만큼 쉬프트 시킨다. 연산 결과에 따라 overflow 를 제외한 C, Z, S 비트를 반영한다.

◆ **UNA 모듈:** 단일 소스 오퍼랜드를 사용한 연산을 수행하며, De_ex_op[3]에 의

해 구동된다. 연산 결과 및 flag 비트를 반영하며, <표 11>에는 UNA 명령어별로 반영되어야 할 flag 비트를 요약한다.

<표 11> UNA 모듈의 명령어별 flag 반영

종류	EXTB	EXTS	CNT0	CNT1	CVB	CVS	NOT
Flag	ZS	ZS	Z	Z	ZS	ZS	ZS

◆ **MUL 모듈:** 곱셈 연산과 MAC 연산을 수행하며, De_ex_op[4]에 의해 선택된다. Flag 비트 반영은 없으며, 그전 flag 값을 bypass 시킨다. 32 비트 곱셈 연산을 위해 Opa, Opb 를 사용하거나, 32 비트 MAC 연산을 위해 Opa, Opb, Opc, Opd 오퍼랜드를 사용한다. 64 비트 결과값은 상위 32 비트는 Ex_mh 에 하위 32 비트는 Ex_result 에 출력한다.

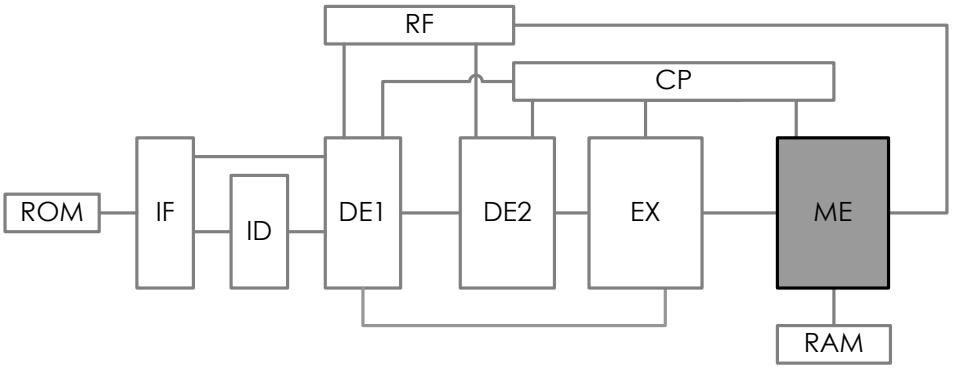
◆ **CP 모듈:** Coprocessor 관련 명령어 중에서 EX 블록에서 처리하는 명령어는 GETCn, MVFn, LDCn, STCn 이며, De_ex_op[5]에 의해 active 된다. CP 모듈은 이러한 명령어를 위한 데이터 패스를 제공하는데, 구체적으로 LDCn, STCn 을 위한 어드레스 계산, MVFn 을 위한 Ex_cpbus 채널 데이터의 Ex_result 로의 전달, 그리고 GETCn 을 위한 Ex_cpbus 데이터 값에 따른 Zero flag 변경 등을 수행한다. 이 모듈에서는 GETCn 명령어 처리 시에 Zero flag 를 반영하는 것을 제외하고 flag 값은 bypass 시킨다.

◆ **ETC 모듈:** LEA 명령어를 제외한 move 계열 명령어와 PUSH, POP 의 EX 단 처리를 위한 모듈로 De_ex_op[5:0]가 '0' 으로 세팅될 때 구동된다. Opa, Opb 데이터가 De_bp_cmd 값에 의해 선택되어 ex_result 로 출력 되며, flag 는 bypass 된다.

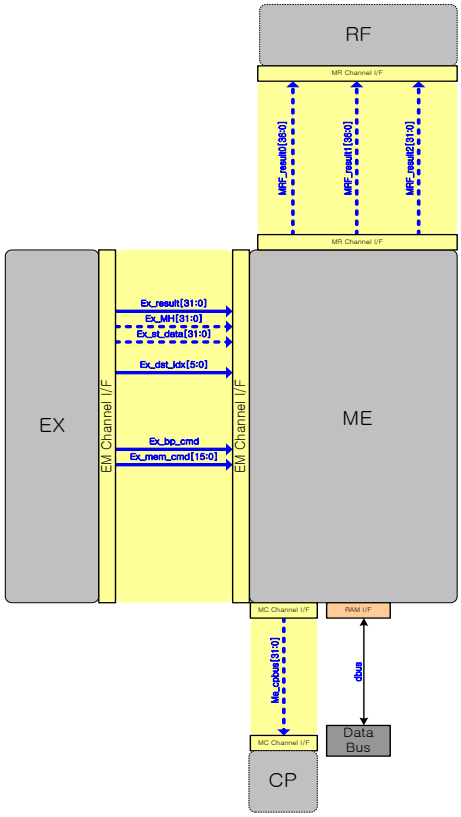
구체적으로 PUSH/POP 명령어의 first (초기 SP 값), 단순 move, JAL, JALR 명령어를 처리시 Opa 가 Ex_result 로 bypass 되고, LDI 명령어 처리 시에서는 Opb 가 Ex_result 로 전달된다. PUSH 명령어 처리 시에는 스택에 전달할 데이터를 PUSH_SR 채널을 검사하여 SR 일 경우, 현재 flag 값을 반영하여 Opc 의 채널 데이터를 가공한다. 또한 POP SR 명령어 처리 시에는 SR 의 flag 영역을 De_FlagUp 채널부터 제공받아 cFlag_reg 에 저장한다.

바. Memory (ME) 블록

ME 블록은 ALTHEA 의 7 단 파이프라인 구조에서 메모리 단계를 처리한다. 주로 메모리를 참조하는 명령어를 위해 메모리 읽기/쓰기 동작을 수행하여 결과값을 RF 블록에 전달한다. 이외에 EX 블록의 연산 결과를 RF 블록에 전달하는 기능, 메모리 참조 후 결과값을 CP 블록에 전달하는 기능, 파이프라인 토큰을 DE 블록에 전달하는 기능을 수행한다. <그림 26>과 <그림 27>은 각각 ALTHEA 프로세서에서 ME 블록의 위치와, ME 블록과 다른 블록 사이의 간단한 채널 인터페이스를 나타내고 있다.



<그림 26> ME 블록과 다른 블록과의 관계



<그림 27> ME 블록의 입출력 채널

ME 블록에서 제공하는 기능은 다음과 같다.

- 메모리 접근 및 메모리 데이터의 정렬
- PUSH/POP 명령어를 위한 SP 값 변경
- WB 단계를 위한 출력 포트 선택
- 코프로세서 입력 인가

7 단 파이프라인 구조에서 유일하게 메모리를 접근할 수 있는 기능을 제공한다. 메모리 접근 시에 필요한 메모리 주소 값을 EX 블록에서 전달받아 메모리 쓰기 및 읽기 기능을 수행한다. 메모리 읽기의 경우에는 데이터의 size 에 따라서 정렬 및 부호 확장 기능이 필요하다.

PUSH/POP 명령어 실행 시 변화하는 SP 값을 변경시키고 메모리 주소 값에 반영하면서, 최종 SP 값을 WB 단계에서 SP 레지스터에 저장시키는 기능을 제공한다. EX 블록의 연산 결과값, 메모리 읽기 경우의 메모리 데이터 값, 그리고, EX 블록에서 변경된 status 값, bypass 되는 MH 값의 WB 단계의 쓰기 동작을 위해 출력 포트를 선택한다. 마지막으로 코프로세서의 레지스터 파일에 쓰기 동작을 지원하기 위해 메모리에서 읽은 데이터를 코프로세서로 전달할 수 있다.

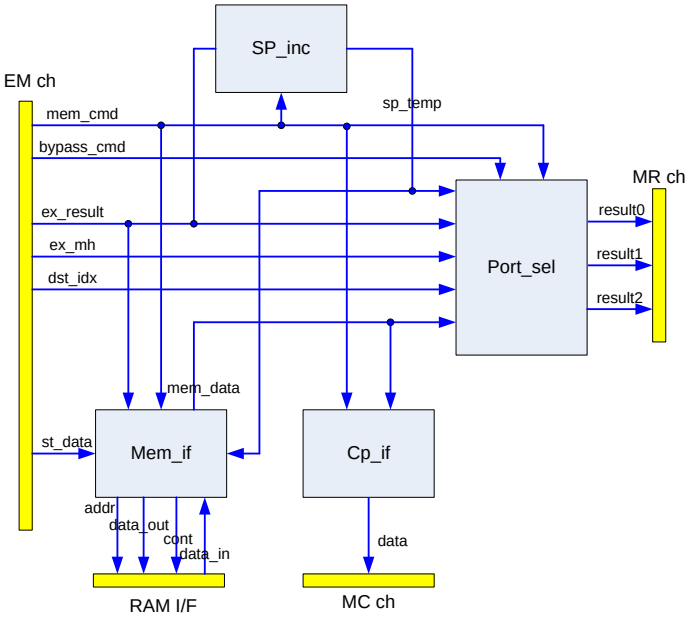
ME 블록은 <그림 28>과 같이 메모리 접근 및 데이터 align 기능을 수행하는 Mem_if 모듈, SP 값을 변경시키는 SP_inc 모듈, WB 블록으로 출력값을 선택하는 Port_sel 모듈, 그리고, 코프로세서 인터페이스를 위한 Cp_if 모듈로 구성된다. 다음에서 각 모듈별로 상세기능을 설명한다.

◆ **Mem_if 모듈:** Mem_if 모듈은 메모리 명령어인 경우 활성화되며 EX 블록의 결과값인 ex_result 를 주소 값으로 사용한다. mem_cmd 에 의해 메모리 쓰기/읽기 동작이 구분되며 쓰기 동작의 경우는 저장할 데이터(st_data)를 입력 받는다. 메모리와의 인터페이스는 주소(addr), 출력 데이터(data_out), 제어신호(cont), 입력 데이터(data_in)로 구성된다. 메모리 읽기의 동작의 경우 mem_cmd 에 정의된 size 정보로 정렬 및 부호확장 연산을 수행하여 결과값(mem_data)을 출력한다. PUSH/POP 명령어인 경우에는 ex_result 대신 SP 를 변경시킨 sp_temp 를 사용하여 주소 값을 생성시킨다.

◆ **SP_inc 모듈:** SP_inc 모듈에서는 초기 주소 값 즉, SP 값은 ex_result 로부터 입력되고, 처음 메모리 접근 동작을 수행한 후, 초기 SP 값을 4 씩 증가(POP 명령어), 혹은 4 씩 감소(PUSH 명령어) 연산을 수행한다. 이때, 입력되는 dst_idx

(POP 명령어), st_data (PUSH 명령어)와 동기화되어 다중 메모리 접근 동작을 수행한다. 이후, mem_cmd 로부터 최종 메모리 접근 동작임을 알리는 신호를 감지하여 최종 SP 값을 출력(sp_temp)한다.

◆ **Port_sel 모듈:** ALTEHA 코어의 레지스터 파일에는 총 3 개의 쓰기 포트가 존재한다. 이들 포트에 EX 블록의 결과값인 ex_result, 메모리 읽기 결과값인 mem_data, EX 블록에서 전달되는 MH 값(ex_mh), status 값(ex_flag)을 선택하는 기능을 수행한다. <표 12>에서 각 포트에 할당되는 입력 값을 요약하였다.



<그림 28> ME 블록 구조도

ALTHEA 코어의 레지스터 파일에는 총 3 개의 쓰기 포트가 존재하며 Port_sel 모듈에서 선택된다. 이들 포트에 EX 블록의 결과값인 ex_result, 메모리 읽기 결과값인 mem_data, EX 블록에서 전달되는 MH 값(ex_mh) 선택하는 기능을 수행한다. 각 포트에 할당되는 데이터는 <표 12>에 요약되어 있고, mem_cmd[15:13]에 의해서 구분된다.

<표 12> 포트에 할당되는 입력

포트	입력값
Result0 Mem_cmd[15]	EX 블록 연산 결과 값(ex_result) ME 블록 메모리 참조 값 (mem_data)
Result1 Mem_cmd[14]	EX 블록 곱셈 결과 상위 값(ex_MH) PUSH/POP 최종 SP 값(sp_temp)
Result2 Mem_cmd[13]	EX 블록 곱셈 결과 하위 값

Result0 의 입력 값은 모두 DE 블록에서 정의한 인덱스로 dst_idx 신호를 통하여 목적지 레지스터에 접근한다. DE 블록에서 읽기 기능 수행 시 개별 접근으로 분류되었던 ex_MH, ex_flag, 그리고 PUSH/POP 명령어의 최종 SP 값을 지닌 sp_temp 입력도 Result1 포트에서 구별되기 위해 Result1 채널에 레지스터 인덱스를같이 사용한다. <표 13>에는 Result1 포트의 각 입력에 해당되는 인덱스 값을 나타낸다.

<표 13> Result1 포트의 입력 인덱스

입력값	인덱스 (binary)
ex_MH	00
sp_temp	01
ex_flag	10

◆ **Cp_if 모듈:** Cp_if 모듈은 코프로세서 명령어 중 메모리의 값을 읽은 후 코프로세서 내부 레지스터에 쓰기 동작을 수행하는 명령어를 위해 mem_data 를 코프로세서에 전달한다.

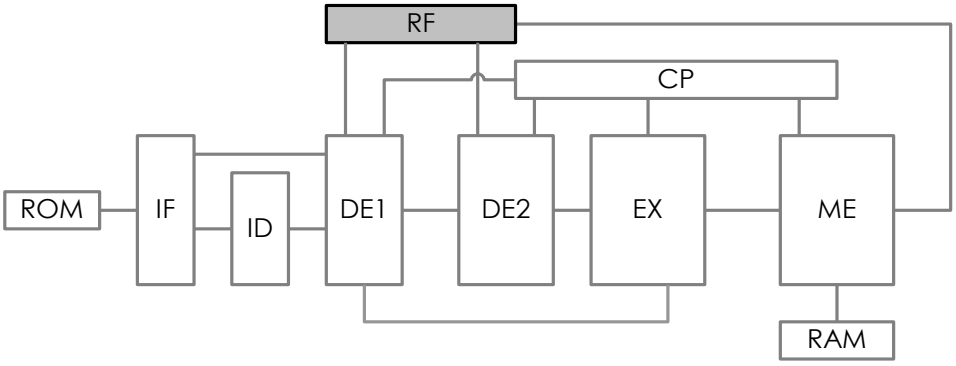
사. Register File (RF) 블록

RF 블록은 ALTHEA 프로세서의 연산 처리 및 결과값의 보관 등에 사용되는 GPR(범용 목적 레지스터)과 프로세서의 상태 및 정보를 위한 SPR(특수 목적 레지스터)로 구성 된다. RF 블록의 내부는 GRP 과 SPR 의 억세스(쓰기 및 읽기)외에도 명령어 간의 상호의존성을 해결하기 위한 메커니즘을 지원하여야 한다. ALTHEA 프로세서의 RF 블록은 다음의 <표 14>와 같은 기능과 규격을 지원한다.

<표 14> RF 블록의 주요 기능 및 규격

기능	규격
기능	GPR 읽기 및 쓰기 SPR 읽기 및 쓰기 레지스터 읽기/쓰기의 상호의존성 해결
동작 방식	Clockless 비동기식 Handshaking
인터페이스 블록	DE1, DE2, ME 블록

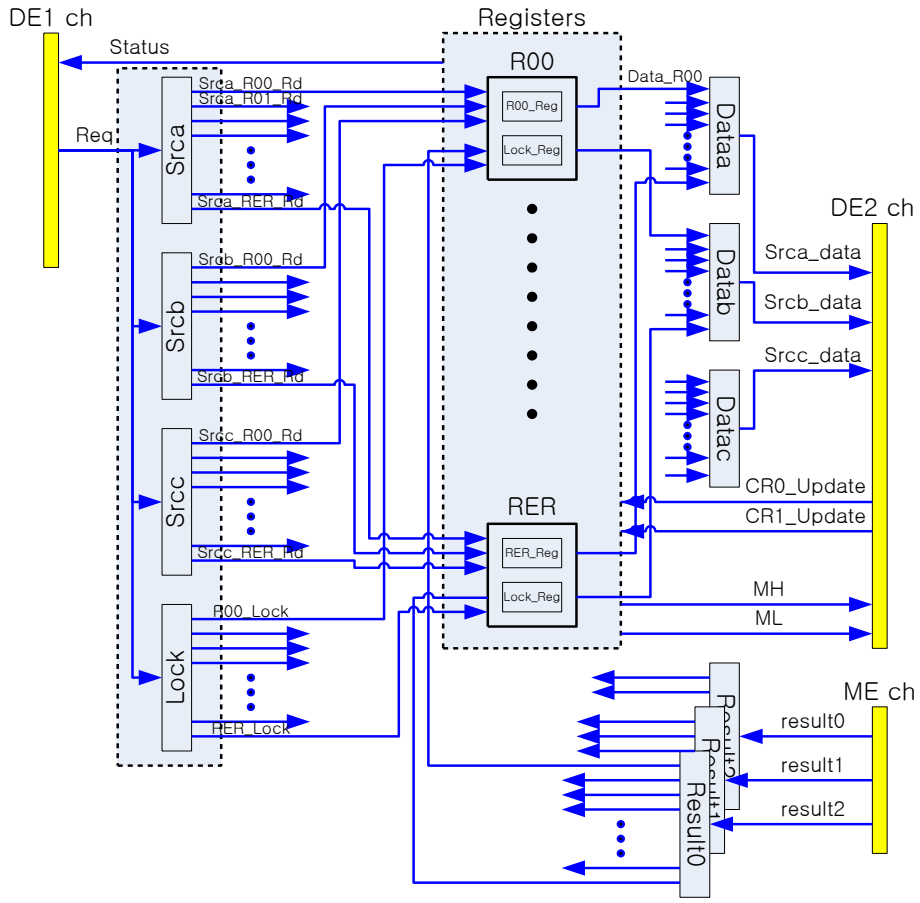
다음의 <그림 29>는 RF 블록과 기타 블록과의 연결관계를 나타낸 그림이다.



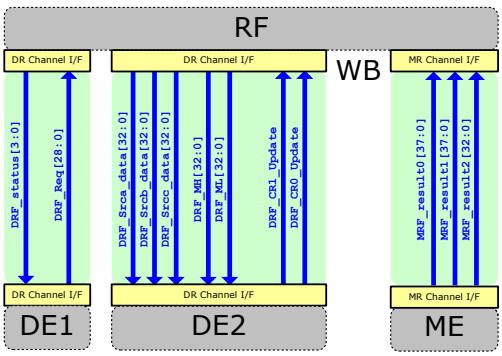
<그림 29> RF 블록과 기타 블록과의 관계

<그림 30>은 RF 블록의 여러 기능 블록을 개념적으로 정리한 블록도이며, <그림 31>은 RF 블록의 외부 채널 인터페이스에 대한 그림이다. RF 블록은 내부에 개념적으로 다음과 같은 기능 모듈로 구성된다.

- 개별 Register File 레지스터 모듈 : GPR과 SPR (총 26개)
- Source A/B/C 요청 처리 모듈 : DE1의 요청중 Srca/b/c/MH/ML 읽기 처리
- Lock 처리 모듈 : DE1의 요청중 Lock 처리 처리
- 데이터 A/B/C 읽기 모듈 : 개별 레지스터 중 읽기 요청에 따른 읽기 데이터 출력 처리
- 레지스터 쓰기 모듈 : Result 0/1/2 쓰기 요청에 따른 개별 레지스터에 대한 쓰기 처리



<그림 30> RF 블록 내부 구성도



<그림 31> RF 블록 외부 인터페이스

◆ **RF 블록 모듈의 상세 구조:** 레지스터 파일은 크게 3 가지 기능 모듈로 구성된다.

- GPR/SPR 모듈
  - 16개의 GPR 레지스터
  - SR, SP, CR0, CR1의 SPR 레지스터
- MHML 모듈
  - MH 및 ML의 SPR 레지스터
- 중재기 모듈
  - 읽기/쓰기 포트 중재기

GPR/SPR, MHML 모듈은 실제의 레지스터 모듈로서 각 모듈은 해당되는 레지스터와 각 레지스터의 잠금 및 잠금 해제 기능을 담당하는 Lock 모듈로 구성된다.

GPR/SPR 모듈은 다음과 같은 3 개의 내부 읽기 포트, 2 개의 내부 쓰기 포트와 2 개의 제어 포트를 가진다.

- 동작 요청 포트
  - 레지스터 오퍼랜드 인출 및 제어(lock)를 위한 동작 요청 포트
  - SR 레지스터 상태 출력 포트
- 읽기 포트
  - 3개의 데이터 포트 : Source A/B/C에 대한 오퍼랜드용 데이터 출력 포트
  - 2개의 데이터 포트 : MH:ML을 위한 데이터 출력 포트

- 쓰기 포트
  - 3개의 쓰기포트 : MRF_result0~2 입력에 연결되며 GPR/SPR 모듈의 모든 레지스터에 대하여 억세스
  - 2개의 CD ㄹ지스터 상태 변경 요청 포트 : CR0 및 CR1의 레지스터 값 변경을 위한 요청 포트

GPR/SPR 모듈에서 잠금 설정된 레지스터에 대한 잠금 해제는 쓰기 포트를 통하여 새로이 값이 갱신되는 레지스터에 대하여 잠금 해제를 수행한다.

MHML 모듈은 MH 와 ML 레지스터에 대한 2 개의 내부 읽기 및 쓰기 포트를 가진다. 또한 MHML 레지스터의 잠금을 위한 제어포트를 가진다.

중재기 모듈은 읽기 포트 중재기, 쓰기 포트 중재기의 두 개의 내부 모듈로 구성되며, 읽기 및 쓰기 중재기는 상호 연동되어 읽기 및 쓰기가 동시에 발생할 경우 해당 레지스터에 대한 중재를 수행한다.

아. Write Back (WB) 블록

WB 블록은 레지스터 파일 이외에 특별한 기능 모듈을 내장하지 않고 있으며, 앞에서 설명한 ME 블록의 최종 3 가지 결과 값(result0, result1, result2)을 레지스터 파일에 쓰기 동작을 수행한다. 레지스터 파일의 구조는 다음 절에서 설명한다.

3. 버스 인터페이스 유닛 설계

이 장에서는 ALTHEA 의 버스 인터페이스 유닛(BIU)의 구조와 기능, 그리고 인터페이스에 대하여 기술한다.

가. 버스 인터페이스 유닛 개요

ALTHEA 프로세서는 시스템 버스로서 여러 SoC 에서 폭넓게 받아들여지고 있는 ARM 사의 AMBA 버스 프로토콜을 사용한다. 이는 기존의 AMBA 기반으로 제작된 주변장치와 ALTHEA 프로세서를 이용하여 쉽게 SoC 를 구성할 수 있음을 의미한다. 또한, 프로세서 내부의 비동기식 데이터 전송 프로토콜과 외부의 동기식 AMBA 프로토콜을 에러 없이 동기화 시켜주는 기능도 수행한다.

나. 버스 인터페이스 유닛 기능

버스 인터페이스 유닛에서 제공되는 기능은 다음과 같다.

- 하바드 아키텍처 지원
- AMBA 버스 프로토콜 변환 (AMBA 2.0 or AMBA AXI)
- 비동기/동기 신호의 동기화
- Burst transaction 지원
- 예외 상황 처리 지원

프로세서 코어의 메모리 인터페이스가 명령어와 데이터 접근이 분리된 하바드 아키텍처를 지원하고, 외부 ABMA 시스템 버스도 명령어, 데이터 버스가 따로 존재하므로, 명령어 BIU (IBIU), 데이터 BIU (DBIU)에서 개별적으로 처리한다. 또한, 기존의 AMBA 버스 기반으로 설계된 주변 장치들을 쉽게 통합하기 위해서 내부의 버스 인터페이스를 AMBA 버스로 변환 혹은 AMBA 버스를 통한 외부 접근을 내부 버스로 변환하는 기능을 수행한다.

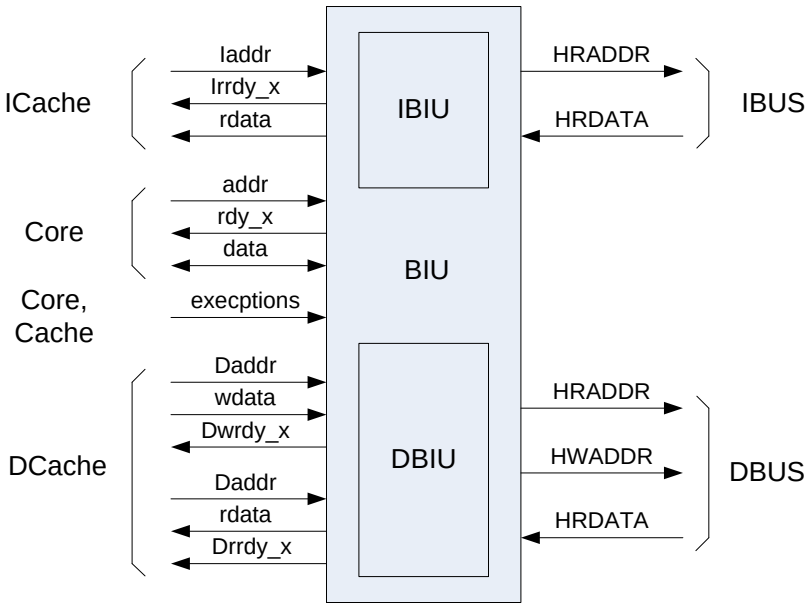
<그림 32>의 왼쪽 인터페이스들은 모두 비동기식 전송을 가정한 신호들이며, 오른쪽은 동기식으로 데이터를 전송한다. 즉, ALTHEA 의 버스 인터페이스 유닛은

AMBA 프로토콜 변환뿐만 아니라, 비동기식 도메인과 동기식 도메인 사이에서 타이밍 해저드 없이 데이터를 전달할 수 있는 동기화 기능을 제공한다. 아울러, 완벽한 AMBA 호환을 위해서는 burst 모드 전송을 처리할 수 있으며, cache 나 코어로부터의 각종 miss, violation 신호를 통해 데이터 흐름을 제어할 수 있는 기능을 제공한다.

다. 버스 인터페이스 유닛 입출력 신호

<그림 32>는 버스 인터페이스 유닛의 입출력 신호를 나타내고 있다. 프로세서 내부 인터페이스로 명령어 Cache, 데이터 Cache, 프로세서 코어와의 인터페이스가 존재한다. 또한, 코어와 Cache 에서 발생되는 예외상황들을 표현하는 신호도 같이 입력된다. 각각의 내부 인터페이스들은 어드레스와 데이터, 그리고 읽기, 쓰기가 종료됨을 알려주는 ready 신호로 구성된다.

외부와의 인터페이스로는 각각 명령어 버스와 데이터 버스로의 입출력 신호가 존재하는데 모두 AMBA 버스와 호환 가능한 신호들이다.

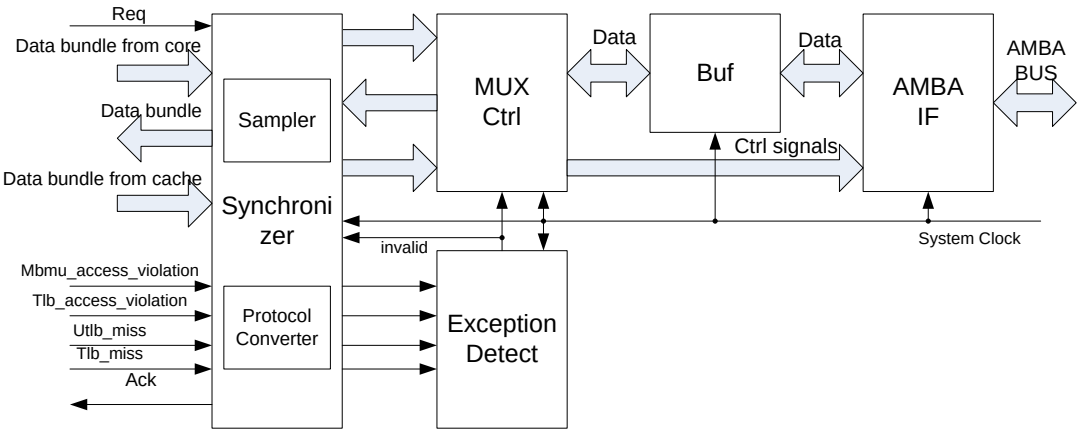


<그림 32> 버스 인터페이스 유닛 입출력

라. 버스 인터페이스 유닛 구조

DBIU 에서 데이터 쓰기 기능을 위한 신호들과 그 기능을 위한 블록들이 추가되는 것을 제외하고는 IBIU 와 DBIU 의 기능이 유사하므로 본 문서에는 DBIU 를 중심으로 설명한다. <그림 33>은 간략화된 DBIU 의 내부 구조를 나타내고 있다.

크게 동기화 모듈, 예외상황 감지 모듈, Mux 제어 모듈, 데이터 buffer 모듈, AMBA 인터페이스 모듈로 나눌 수 있으며 다음에서 각 모듈별 구조를 설명한다.



<그림 33> 버스 인터페이스 유닛 내부 구조

◆ **동기화 모듈:** 앞에서 언급한 것처럼 프로세서와의 내부 인터페이스에는 비동기/동기 도메인 사이의 동기화 장치가 존재한다. Sampler 는 데이터 전송이 비동기 도메인에서 동기 도메인으로 수행될 때 필요한 블록으로 시스템 클록을 사용하여 비동기식 제어 신호와 데이터를 샘플링 한다. 정지 가능한 내부 클록이 유효할 때는 metastable 상태에서 클록을 휴지시켜 데이터를 안정된 구간에서 저장한다.

반대로 동기 도메인에서 비동기 도메인으로 데이터가 전송되는 경우에는 프로토콜 변환 블록에서 프로세서 내부에서 사용되는 시그널링과 데이터 인코딩 방식에 맞게 시스템 클록에 동기화된 핸드쉐이크 제어 신호를 발생시킨다.

◆ **예외상황 감지 모듈:** 예외상황을 알리는 신호를 감지하여 데이터 흐름을 조절할 수 있는 신호를 발생시키고 결과를 프로세서에 전달하는 기능을 수행한다.

◆ **Mux 제어 모듈:** Bundle 된 데이터를 통해 Cache miss 처리를 위한 데이터 접

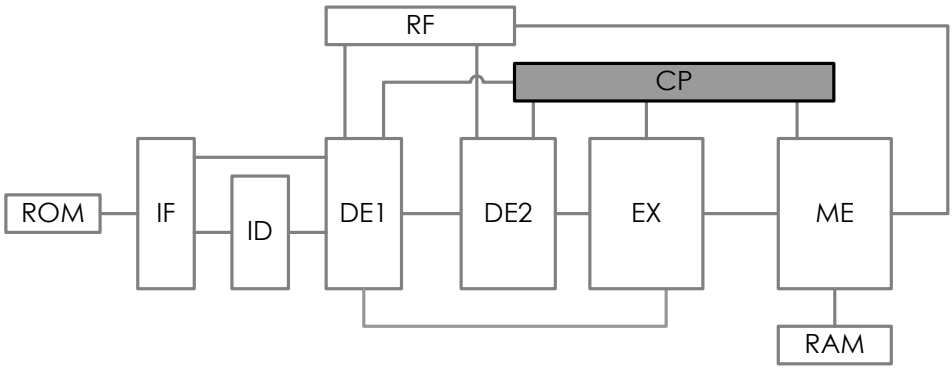
근인지 프로세서 코어에서 직접 AMBA 인터페이스를 접근하는 것인지를 판별한다. AMBA 버스로의 변환을 위한 제어 신호도 생성시킨다.

◆ **데이터 버퍼 모듈:** write-through 방식의 효과를 극대화하기 위해 Cache 모듈 이외에 버스 인터페이스 유닛 내부도 버퍼 회로를 구현한다. 그리고, 읽기 쓰기 동작의 Burst 모드 수행을 위해서도 데이터 버퍼 모듈이 필요하다.

◆ **AMBA 인터페이스 모듈:** AMBA 버스 프로토콜과 프로세서 내부 프로토콜의 변환을 수행한다.

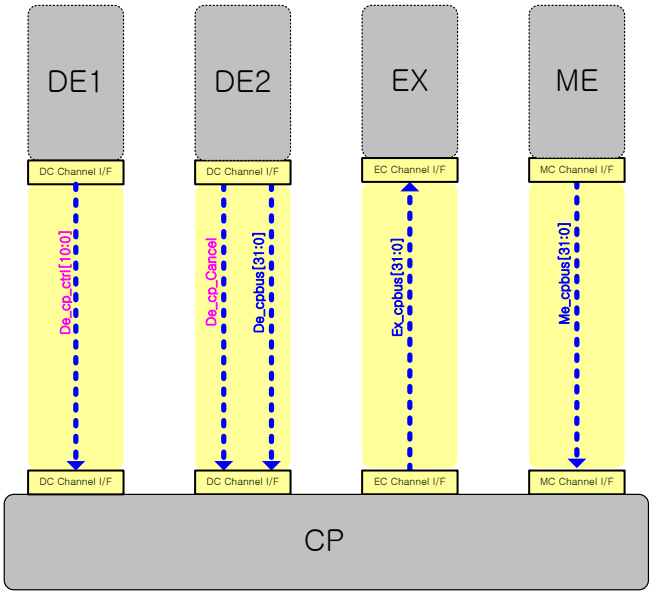
4. Co-Processor (CP) 유닛 설계

CP 블록은 ALTHEA 가 처리하는 명령어 가운데 코프로세서 관련 명령어를 처리한다. CP 블록내의 레지스터 파일의 읽기, 쓰기 기능을 수행하며 주변 블록과의 관계는 <그림 34>와 같다.



<그림 34> CP 블록과 다른 블록과의 관계

CP 블록의 모든 입출력 채널을 표시한 것은 <그림 35>와 같다.



<그림 35> CP 블록의 입출력 채널

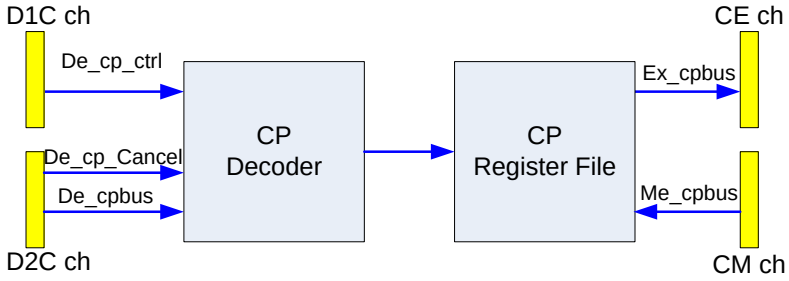
가. CP 블록의 구조

현재 구현된 CP 블록의 구조는 <그림 36>과 같이 내부에 16 개의 32 비트 레지스터를 갖는 레지스터 파일과 제어 신호를 해석하는 디코더로 구성되어 있다. DE1 블록과의 인터페이스로부터 입력 받은 제어 신호를 해석하여 DE2 블록으로부터의 데이터를 레지스터 파일에 저장할 수 있고, 레지스터 파일의 값을 EX 블록과의 인터페이스를 통해 출력, 혹은 ME 블록과의 인터페이스로 입력받아 레지스터 파일에 저장한다.

나. CP 블록의 동작

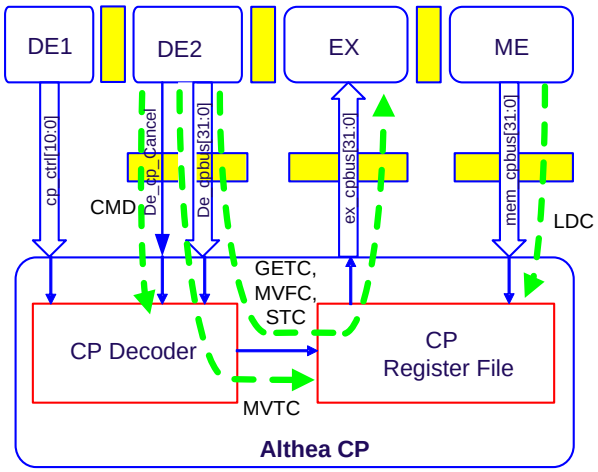
현재 Co-Processor 관련 명령어 중 구현된 것은 EXEC 를 제외한 CMD,

GETC, MVTC, MVF, LDC, STC 등이다. 이중 CP 블록 입장에서 GETC, MVFC, STC 는 CP 레지스터 파일의 값을 읽어서 EX 블록에 출력하는 동작을 수행한다. MVTC 명령어는 DE 블록과의 인터페이스를 통해서 레지스터 파일에 데이터를 저장할 수 있고, LDC 는 ME 블록과의 인터페이스를 통해 데이터를 저장한다.



<그림 36> CP 블록 구조도

<그림 37>은 각 명령어 별로 CP 블록 내부에서의 데이터 흐름을 나타낸다.



<그림 37> CP 블록 관련 명령어별 데이터 흐름도

5. ALTHEA 기능 및 성능 분석

가. 기능 분석

저전력 비동기식 프로세서인 ALTHEA 는 총 100 개의 명령어 셋을 지원하도록 설계되었으며, 당해년도에는 EXECn, SWI, BRKPT, SYNC, STEP 을 포함한 100 개의 명령어의 구현 및 기능 검증을 수행하였다. 다음 <표 15>는 ALTHEA 프로세서의 명령어 셋을 Class 와 Type 별로 분류한 것을 나타낸다.

<표 15> ALTHEA 명령어 분류

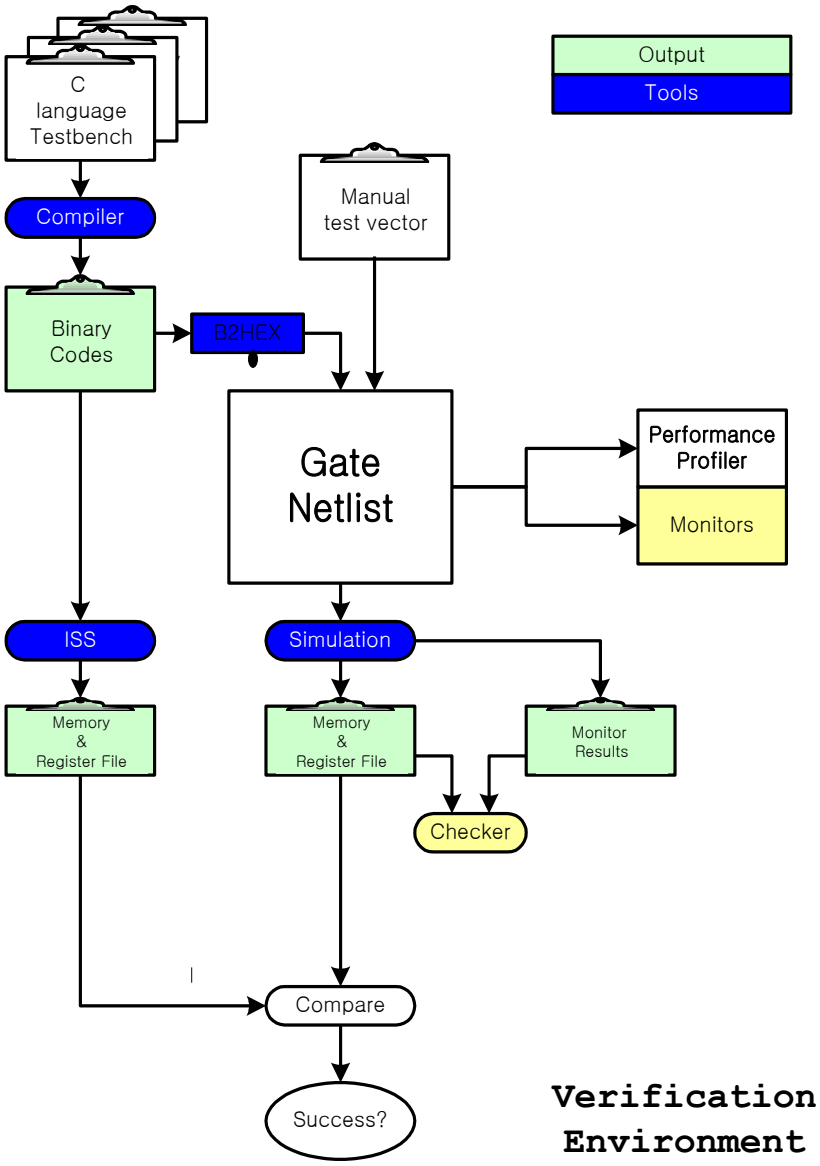
Class	type	Instruction
Load ER with immediate (1)		LERI
Memory Access (26)	Load	LD(2)/LDB(2)/LDBU(2)/LDS(2)/LDSU(2)/LDAU(2)/LEA(4)
	Store	ST(2)/STB(2)/STS(2)/STAU(2)
	multi	POP/PUSH
Data Move (9)	move	MTMH/MTML/MFMH/MFML/LDI/MTCR0/MTCR1/MFCR0/MFCR1
Branch (19)	Cond	JNV/JV/JP/JM/JNZ/JZ/JNC/JC/JGT/JLT/JGE/JLE/JHI/JLS
	Uncond	JMP/JAL/JR/JALR/JPLR
Arith & Logical (32)	ALU	ADD/ADC/SUB/SBC/AND/OR/XOR/CMP/TST/ADDQ/CMPQ/NEG/SET/CLR
	Shift	ASR(2)/LSR(2)/ASL(2)/SSL(2)
	Unary	EXTB/EXTS/CVB/CVS/CNT0/CNT1/NOT
	Multiply	MUL/MULU/MAC
Misc (13)	Cop	CPCMDn/GETCn/MVTCn/MVFCn/LDCn/STCn/EXECn

	Control	NOP/SYNC/HALT
	interrupt	SWI/BRKPT/STEP
● Total 100 instructions (in binary) ● Supports only AE32000C version ● Instructions are described in order		

1) 기능 검증을 위한 검증 환경 및 시뮬레이션을 통한 검증

설계 및 통합된 ALTHEA 프로세서의 기능 검증을 위하여 기능 검증을 위한 검증 환경을 구축하고 구축된 검증 환경을 바탕으로 개별 ALTHEA 명령어의 기능 동작을 수행하였다. <그림 38>은 ALTHEA 의 기능 및 동작 검증을 위한 환경을 나타낸 그림이다.

개발 환경은 2 차년도 연구수행을 통하여 확보한 C-언어를 이용한 개발 환경을 사용하여, 기능 테스트를 위한 테스트벡터를 생성하고 이를 합성된 게이트 네트리스트에 인가한 후 검증 환경의 모니터 코드 와 프로파일 코드를 통하여 기능의 동작 여부를 확인하였다.



<그림 38> ALTHEA 기능 검증환경

<표 16>은 이와 같은 기능 검증 환경을 통하여 구현된 명령어와 그에 따른 동작 검증 확인을 나타낸 표이다.

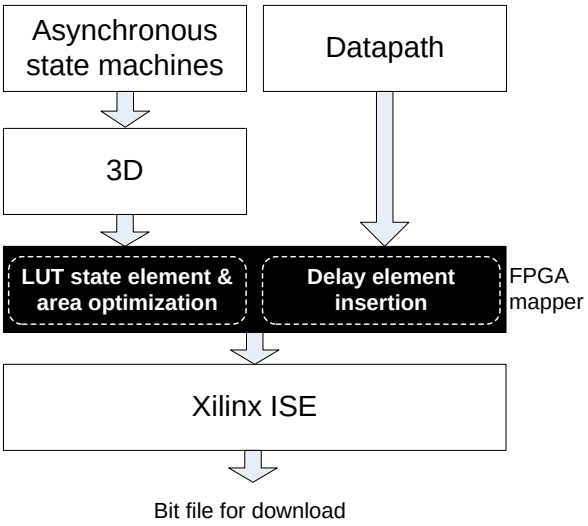
<표 16> ALTHEA 명령어 구현 및 동작 체크리스트

그룹	명령어	구현	동작검증	그룹	명령어	구현	동작검증
LERI	LERI	○	○	BRC	JLT	○	○
LD	LD	○	○		JGE	○	○
	LDB	○	○		JLE	○	○
	LDS	○	○		JHI	○	○
	LDBU	○	○		JLS	○	○
	LDSU	○	○	ALU	ADD	○	○
	LD2	○	○		ADC	○	○
	LEA(ASPO)	○	○		SUB	○	○
	LEA(LSEA)	○	○		SBC	○	○
	LEA(LESA)	○	○		AND	○	○
	LEA	○	○		OR	○	○
	LDB2	○	○		XOR	○	○
	LDS2	○	○		CMP	○	○
	LDBU2	○	○		ADDQ	○	○
	LDSU2	○	○		CMPQ	○	○
	LDAU	○	○		NEG	○	○
	LDAU	○	○		SET	○	○
ST	ST	○	○		CLR	○	○
	STB	○	○		TST	○	○
	STS	○	○	SFT	ASR	○	○
	ST2	○	○		ASR2	○	○
	STB2	○	○		LSR	○	○
	STS2	○	○		LSR2	○	○
	STAU	○	○		ASL	○	○
	STAU	○	○		ASL2	○	○
STK	PUSH	○	○		SSL	○	○
	POP	○	○		SSL2	○	○
MOV	LDI	○	○	UNA	EXTB	○	○
	MTML	○	○		EXTS	○	○
	MTMH	○	○		CVB	○	○
	MFML	○	○		CVS	○	○
	MFMH	○	○		NOT	○	○
	MTCR0	○	○		CNT0	○	○
	MTCR1	○	○		CNT1	○	○
	MFCR0	○	○	MUL	MUL	○	○
	MFCR1	○	○		MULU	○	○
BRA	JMP	○	○		MAC	○	○
	JAL	○	○	CP	CPCMD*n*	○	○
	JR	○	○		GETC*n*	○	○
	JALR	○	○		EXEC*n*	○	○
	JPLR	○	○		MVTC*n*	○	○
BRC	JNV	○	○		MVFC*n*	○	○
	JV	○	○		LDC*n*	○	○

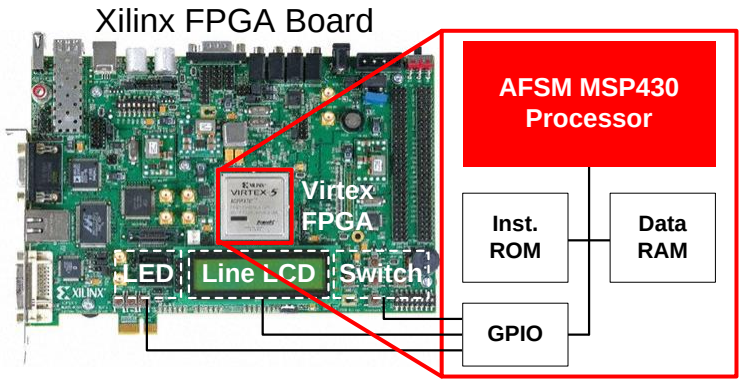
	JP	○	○		STC*n*	○	○
	JM	○	○	CTRL	NOP	○	○
	JNZ	○	○		SYNC	○	○
	JZ	○	○		HALT	○	○
	JNC	○	○	INT	SWI	○	○
	JC	○	○		STEP	○	○
	JGT	○	○		BRKPT	○	○

2) FPGA 구현을 통한 기능 검증

본 연구과제에서는 차년도에 개발될 ALTHEA 프로세서의 실증적인 검증을 위한 FPGA 를 이용한 기능 검증 방법을 개발하였다. 차년도 사업에서는 성능향상을 위하여 Asynchronous FSM 방식의 제어회로를 합성할 수 있는 CAD 툴인 3D 를 활용하여 ALTHEA 를 설계하게 된다. 본 연구 과제에서는 3D 로 설계된 비동기 프로세서를 Xilinx Virtex5 의 특성에 맞게 합성 및 매핑할 수 있는 방법을 개발하였다. <그림 39> (a)의 FPGA mapper 가 Xilinx Virtex5 에 맞게 최적화 및 회로의 변경을 가하게 된다. 당해 년도 연구과제에서는 시험적으로 간단한 구조의 MSP430 호환 비동기 프로세서를 설계하여 환경에 대한 검증을 마쳤다. 구현된 비동기 프로세서는 C-언어를 사용한 간단한 데모 프로그램을 작성하여 이를 FPGA 상에서 수행하였다. FPGA 를 통한 기능 검증을 위하여 16x2 line LCD 를 이용하여 화면 출력을 수행하도록 하였으며, <그림 39>는 이와 같은 FPGA 검증 환경과 형상을 나타낸 그림이다.



(a) FPGA 기능 검증 환경



(b) FPGA 기능 검증 형상

<그림 39> ALTHEA FPGA 기능 검증 구성

나. 성능 분석

<표 17>은 비동기식 프로세서 코어의 게이트 수준의 합성 후 면적 정보를 나타내고 있다. TiDE 의 합성 단계별, 또한 cell type 정보도 구분하였다. 최종적으로 DC 를 invoke 한 합성에서는 대략 110 만 게이트의 면적을 차지하였다.

<표 17> 비동기식 프로세서 코어의 면적 정보

<table>
<tr><th></th><th>Cell type</th><th>Map</th><th>Logopt</th><th>Post</th><th>sdf_b</th><th>sdf_n</th><th>sdf_w</th></tr>
<tr><td rowspan="5">gate eq.</td><td>Delay Matchers</td><td>10,262</td><td>10,650.7</td><td></td><td colspan="3" rowspan="4">CONTROL
115677.5/5.184
LBL
449471.8/5.184</td></tr>
<tr><td>Memory</td><td>47,902.3</td><td>47,902.3</td><td></td></tr>
<tr><td>C-element</td><td>8,817.3</td><td>8,817.3</td><td></td></tr>
<tr><td>Logic</td><td>76,481.3</td><td>67,803</td><td></td></tr>
<tr><td>Total</td><td>143,463.0</td><td>135,173.3</td><td></td><td colspan="3">109,017.9</td></tr>
</table>

Throughput 성능을 측정하기 위해 MiBench 프로그램 중 SHA 를 사용하였다. <표 18>은 각각의 벤치마크 프로그램에 대해 gate 수준의 코어를 sdf 파일을 발생시켜 실행시켰을 때의 타이밍 정보를 나타낸다. 실행된 명령어의 개수와 최종 시뮬레이션 종료 시간, DE1 에서의 명령어 issue 후 다음 명령어 issue 까지의 평균 시간, MIPS, 동기식 버전의 최대 클록 주파수(160 MHz)와의 환산 주파수를 수치화 하였다.

<표 18> SHA 실행시의 성능 지표

The # of executed Instruction	16,049,554
Simulation time (ns)	455,676,396
Average latency (ns)	28.39
MIPS	35.22
Equivalent Clock Frequency(MHz)	41.13

다. 전력 소모 분석

설계된 비동기식 프로세서 코어의 전력 소모 측정은 gate level 의 netlist 와 타이밍 시뮬레이션 결과 발생하는 데이터베이스를 vcd 파일로 변환한 입력 패턴을 사용하여 Primetime-PX 에서 이루어진다. 즉, dynamic power analysis 가 이루어지므로, 합성 결과 발생되는 static power analysis 보다는 의미 있는 결과 데이터 이지만, 실제 소모 전력과 측정된 전력이 상당 수준 연관성이 있는 것은 아니다. 그러나, 위와 같은 flow 로 동기식 버전의 전력 소모 양상은 상대 비교가 가능하다. <표 19>는 Qsort 실행 시의 전력 소모결과로서 mW/MHz 단위로도 환산하였다. 참고로 동기식 버전의 전력 소모량은 normal case 기준 12.18 mW 이다.

<표 19> Qsort 실행시의 전력 소모 지표

Type	best	normal	worst
Power (mW)	11.6	5.88	3.036
mW/MHz	0.335	0.237	0.204

라. 캐시 연동 모델의 성능 분석

다음에서 각 벤치마크 프로그램 별로 명령어 캐시와 데이터 캐시를 통합한 ALTHEA 의 성능측정 결과를 제시한다. 즉, 캐시가 있는 구조와 캐시가 없는 구조의 성능을 비교하며, 각 캐시들의 hit-ratio 를 계산하였다. 3-corner 중 normal case 의 결과이며, 현실적인 시뮬레이션을 위해 AMULET3 의 시뮬레이션 환경(메인 메모리 access time: 60 ns, 메인 메모리-캐시간 access time: 8 ns)을 참조하여 메인 메모리의 access time 을 50 ns 로 가정하였다.

<표 20> 캐시 연동 모델의 성능 분석

Benchmark Program	Hit ratio %)		#of instr.	Completion time (ns)		Improvement (%)
	Instr. cache	Data cache		w/o cache	With cache	
QSORT	98	67	105,825	8,186,484	6,006,098	36.3
String Search	99	87	100,421	7,539,484	5,819,013	29.5
Bitcount	99	94	485,343	39,017,872	25,872,357	50.8
Dijkstra	99	57	6,708,414	546,461,527	346,221,838	57.8
Sha	99	79	16,049,554	1,016,735,518	661,997,622	53.5

Dhrystone	99	95	860,959	67,379,226	44,555,145	51.2
AVG						46.5

비교적 순차적인 명령어들을 패치하는 명령어 캐시의 hit-ratio 는 높으나, 데이터 캐시의 경우 프로그램 별로 hit-ratio 가 편차가 존재한다. 전반적으로 캐시를 사용하지 않는 구조에 비해, 캐시를 사용하게 되면, 46.6 %의 성능 향상을 기대할 수 있다.

제 5절 문제점 분석 및 차년도 연구 수행 방향

당해년도에는 지난 3 년 동안의 연구개발 결과를 이용하여 비동기식 프로세서 코어(ALTHEA)의 성능 보완 및 외부 버스 인터페이스 구현을 수행하였다. 구현된 비동기식 프로세서 코어는 벤치마크 프로그램을 통해 분석한 결과 현재 최대 41.1MHz 의 성능을 보이고 있으며, Qsort 를 수행했을때 전력소모(power)는 정상적인 동작환경(nornal case)에서 5.88mW 정도의 값을 보여주고 있고 합성시 약 110 만 게이트급의 규모로 설계되었다. 이러한 전력과 성능 측면의 결과치는 본 사업에서 추구하는 최종 목표에는 아직 미치지 못하고 있지만, 현재 비동기식 프로세서 설계에 적용하는 환경 측면을 고려해 보면 기존의 동기식 프로세서에 비하여는 전력소모 측면에서 상당한 이득이 있음을 알 수 있다.

프로세서 설계시에 전력소모는 반도체공정, 라이브러리, 게이트 레벨 회로설계 방식, 아키텍쳐 및 명령어셋 구조, 컴파일러 등의 소프트웨어, 그리고 CAD 툴의 특성등 다양한 요소에 의하여 영향을 받는다. 본 연구개발 수행시에는 소규모 연구개발 특성상 저전력화에 영향을 주는 다양한 요소에서 모두 저전력화를 추구하

는 것은 현실적으로 어려움이 있어서 주로 회로설계 방식 및 아키텍쳐 수준에서의 최적화를 이용한 비동기식 설계를 적용하여 저전력화를 추구하였기 때문에 이러한 정도의 결과를 얻은 것으로 판단된다.

성능 측면에서 살펴보면, 현재 ALTHEA 설계에 사용된 TiDE 기반의 top-down 방식의 설계 흐름은 설계자의 의도를 반영할 수 있는 것은 최상위 수준의 언어인 HASTE 가 유일하다. 다시 말하면, 설계자가 의도하는 시스템의 최적화는 HASTE 로 설계 스펙을 어떻게 기술하느냐에 따라서 달라진다. 그러나, 같은 비동기식 시스템을 여러 기술 방식으로 다양하게 표현하는 것은 한계가 있으며, 이는 top-down 방식이 회로 기술부터 구현까지의 짧은 turn around 시간을 갖는 장점에도 불구하고, 회로의 최적화 설계 측면에서 단점으로 지적되어 왔다. 데이터패스는 동기식의 것을 그대로 사용하고, 4-phase bundled data 방식의 프로토콜을 기본으로 제어패스를 그래프 방식에 기반한 툴에 의해 설계하는 이른바, bottom-up 방식의 설계 흐름은 앞서 설명한 top-down 방식의 설계 흐름의 단점을 보완할 수 있다. 즉, 제어패스 설계 시에 개별 프로토콜 신호 발생까지 설계자의 의도를 반영할 수 있으므로, 같은 시스템의 behavior 에 대해 다양한 구현 방식을 선택할 수 있다. 이는 top-down 방식의 설계 흐름보다 설계자가 시스템을 최적으로 구현할 수 있다.

따라서, 지금까지 설계된 ALTHEA 의 성능 향상을 위해서는 bottom-up 방식의 설계 흐롬을 도입하는 것이 필요하다. 이러한 측면에서 2011 년 연구개발에서는 비동기식 프로세서의 성능 개선을 위하여 비동기 설계 flow 를 변경하는 것을 포함하여 다각도로 새로운 방안을 모색할 예정이다. 이를 통하여 비동기식 프로세서 코어의 동작속도를 향상시키는 방향으로 연구개발을 수행할 계획이다.

제 3 장 결론

비동기식 설계(Asynchronous Design) 기술에 기반을 둔 설계 기법은 현재 반도체 설계 분야에서 클럭에 기반을 둔 동기식 기술(Synchronous Design)에 비하여 아직은 시장에서 활성화되지 못한 측면이 있으나, 현재 몇몇 응용분야에 적용되어 상용화가 진행되고 있으며 점차 관련 CAD 솔루션이 출시되는 등 점차 그 입지를 넓혀가고 있다. 비동기식 설계 기술은 글로벌 클럭(global clock) 대신에 핸드세이킹 기법(handshaking technique)을 통하여 동작하기 때문에 전력소모가 적고, 전자기파적인 특성이 우수하며, 초고집적 SoC 설계시에 다중클럭 도메인(multiple clock domain)의 구현시 클럭 스큐(clock skew) 등이 발생하지 않아 검증에 따른 비용이 절감되고, 모듈화 특성이 우수하며, 주변환경의 변화에도 강인한 동작 특성이 있어 미래 유망 기술로서 연구개발이 수행되는 분야이다. 이러한 비동기식 설계 기반 기술 및 제품에 대한 연구개발은 유럽이나 미국의 주요 기업이나 대학 등에서 지속적으로 연구개발이 수행되고 있으나 국내는 이 분야에 대한 저변이 취약하고 연구개발이 미진한 상황이다. 당 부서에서는 이러한 상황에서 본 연구개발을 통하여 비동기식 설계 기술에 기반을 둔 저전력 프로세서 기술과 관련 IPR을 확보하고 이를 통하여 국내 비동기 관련 기술의 활성화를 추진하였으며, 그간 소기의 성과를 달성할 수 있었다.

당초 2007 년에 전문연구사업으로 시작한 본 사업이 2008 년 말에 정부의 수탁사업 대과제화 방침에 따라 2009 년부터 ETRI 대구경북권의 ‘지능형자동차 전장용 임베디드 시스템 사업’과 통합되어 수행하였다. 그러다가 당해년도인 2010 년에는 ETRI 융합부품부서의 ‘융복합부품 핵심기술 사업’ 대과제로 소속이 변경되어 연구개발이 수행되었다. 당해년도에는 사업의 제 4 차년도에 해당되는 기간으로서 지난 3 년동안 얻은 연구개발 성과를 토대로 하여 32 비트급 초저전력 비동기 프로세서 코어의 기능보완을 통한 개선을 수행하고, 비동기 프로세서와 외부 시스템간의 인터페이스를 용이하게 하기 위하여 AMBA 기반의

외부 버스 인터페이스를 설계하였다. 또한, 부수적으로 본 연구개발에서는 비동기 설계 환경(CAD)과 검증환경에 대한 분석을 통하여 현재 적용중에 있는 TiDE 기반 비동기 설계 도구의 개선 포인트를 분석하고 이를 차기년도에 대체할 수 있는 비동기식 설계 플로우를 구축하고자 하였다. 이를 통하여 AFSM 기반의 비동기식 설계 방식이 현재 비동기 프로세서의 성능 개선 측면에서 효과적임을 도출할 수 있었다. 또한, 그간 비동기식 프로세서의 성능향상 방안으로서 개발을 한 비동기식 데이터/명령어 캐시(Asynchronous Data/Instruction Cache)와의 연동시험을 수행하였다. 비동기식 프로세서 코어(ALTHEA)와 데이터 캐시를 연동하고 이를 시뮬레이션을 통하여 분석한 결과 비동기식 프로세서 코어의 성능이 평균 46.5% 정도 개선됨을 확인하였다. 그리고 향후 비동기식 프로세서의 검증 및 성능개선의 일환으로서 간단한 시험회로를 AFSM 기반으로 설계하여 이를 FPGA 상에 적용하였고, 이를 기존의 TiDE 기반 설계결과와 비교함으로서 AFSM 기반 설계방식의 가능성을 확인하였다.

본 연구개발을 통하여 확보한 초저전력 32 비트급 프로세서와 비동기 저전력 설계 기술은 향후 산업체에서 다양하게 활용 가능할 것으로 기대된다. 이러한 기술은 다양한 저전력 기반 제품의 개발시에 활용이 가능하며, 이를 통하여 그간 외국 주요 기업들에 의해 주도되어 온 프로세서 기술 분야에서 경쟁력을 확보하는데 기여할 수 있을 것으로 기대된다. 또한 그간 대부분을 외국의 프로세서 기술을 도입하여 사용하던 것을 일부 대체할 수 있을 것으로 기대되며, 이를 통하여 국내의 관련 산업의 발전과 활성화에 크게 기여할 수 있을 것으로 전망된다.

본 사업은 당초 소규모의 예산과 인력을 이용하여 중장기적으로 원천 요소기술에 대한 연구개발을 수행하고자 하는 목적에서 시작된 전문연구사업으로서, 연구개발 수행 시에 예산 및 인력 면에서 여러가지 제약이 따른다. 또한, 이러한 사업상의 제약 요소뿐만 아니라 우리나라와 같이 비동기 설계 분야나 저전력 프로세서 개발 관련 저변이 폭넓지 않고 지원이 활성화되지 않은 경우에는

이러한 연구개발의 수행하는데 있어 어려움이 더욱 크다. 당 부서에서 2007 년부터 2010 년까지 본 연구개발을 수행함에 있어서 본 사업이 내포하고 있는 이러한 한계를 극복하기 위하여 다각적인 노력을 기울였다. 이러한 대내외 관련 기업 및 기관과의 협력연구를 통하여 당 부서에서는 비동기 저전력 프로세서 설계에 필요한 주변기술을 확보하고 이를 비동기 설계 기반 저전력 프로세서 설계에 적용 함으로서 본 연구개발을 성공적으로 수행할 수 있었고, 이들과의 적극적인 관계 구축을 통하여 상호 부족한 점을 보완하면서 장점을 공유 함으로서 향후 시너지를 창출 할 수 있는 기반을 마련하였다. 또한 국내 대학과도 위탁과제 수행을 통하여 비동기 설계 분야에 대하여 중장기적인 연구개발을 수행하게 함으로서 인력육성 및 향후 비동기 기술의 확산을 위해 노력하였다. 이러한 관계 설정 및 상호 협력은 향후 본 연구개발의 성과를 상용화하고 적용하는데 있어 아주 유용하게 활용할 수 있을 것으로 기대된다.

2011 년에는 지금까지의 연구수행 결과를 토대로 하여 비동기식 프로세서의 안정화와 통합검증, 그리고 프로세서 코어 자체의 성능향상을 위한 방안을 다각도로 시도할 예정이다. 이를 위하여 지금까지의 비동기식 프로세서 설계 결과에 대하여 다시한번 검토해 보고, 이를 개선할 수 있는 방안을 도출할 계획이다. 한가지 예로서 현재 비동기식 프로세서의 설계에 적용하고 있는 TiDE 설계 플로우를 대신할 수 있는 설계 방법을 적용하여 이를 적용해 볼 예정이다. 또한 설계한 비동기식 프로세서의 검증을 위하여 FPGA 기반 검증 방법에 대하여도 지속적으로 연구해 나갈 예정이다.

당해년도에 본 연구개발을 수행하면서 느꼈던 애로사항은 다음과 같다.

우선 본 연구개발을 수행하면서 가장 어려웠던 점은 본 연구개발사업의 목표 및 성격에 비추어 연구비 규모가 소규모이고, 연도별 연구개발 예산이 확정되어 있지 않아 사업수행 계획을 수립하고 추진하는데 어려웠다는 점을 들 수 있다. 본 연구개발을 효과적으로 수행하기 위해서는 전체 연구개발 기간동안 연구비의

안정적인 확보 및 지원이 매우 중요하며, 이에 따라 전체 연구개발 로드맵을 확정하고 연구개발을 수립하고 수행할 수 있지만, 본 연구개발 사업은 이러한 부분에서 어려움이 있었다고 판단되며, 이러한 점이 아쉬움으로 남는다.

두번째는 본 연구개발사업의 성격에 대한 인식 차이로 기인한다. 당초 본 사업은 중장기적인 차원에서 핵심 요소기술 개발을 위한 전문연구사업으로 시작하였는데, 2009 년부터 정부에서 대과제화 정책을 시행하여 대과제에 통합됨으로서 연구개발 수행 목표 설정에 어려움을 겪은 점이다. 당초 본 연구개발사업에서 지향하는 연구개발 목표와 대과제로 통합된 이후에 대과제에서 지향하는 상용화 목표 간에는 상호 차이가 있었는데, 대과제로 통합하면서 이를 효과적으로 잘 반영하기 어려웠다. 또한, 2009 년에 이어 2010 년에는 본 사업의 소속 대과제가 또 변경되어서 이에 따른 새로운 목표설정 및 수립등으로 인한 행정적인 어려움이 가중되어 연구개발 이외에도 부수적인 어려움이 있었다.

세번째는 정부출연금 사업의 다양한 연구개발 특성을 반영하지 않은 평가 기준이 어려움으로 작용하였다. 당초 본 사업은 소규모의 예산을 활용하여 소규모 인력이 중장기적으로 비동기식 설계 기반 핵심 요소기술을 확보하는 것을 목표로 하였는데, 과제수행 평가시에는 상용화나 기술이전 등 상용화 측면의 요구사항이 많았기 때문에 이를 반영하는데 어려웠다. 또한, 사업의 예산 및 투입인력 규모에 비추어 볼 때 비동기식 프로세서 코어의 반도체 제작을 통한 검증은 예산상 반영되지 않았음에도 불구하고 평가시에는 이러한 점들이 이슈화되어 평가시에 부정적인 요소로 작용하는 등 어려움을 겪은 점을 들 수 있다. 이러한 점에 비추어 볼 때 향후에는 과제 성격에 맞는 평가지표 및 평가 기준의 개발이 매우 중요하다고 판단한다.

상기 기술한 몇 가지 사항이 본 연구개발을 수행하면서 느꼈던 어려운 점이라 판단되었으며, 향후 남은 연구개발년도 동안에는 이러한 점이 어느정도라도 개선이 되었으면 하는 바람이며, 이를 통하여 안정적이고 예측 가능한 환경에서 본 연구개발을 성공적으로 마무리 짓고 싶다

참고문헌

[1] “융복합부품 핵심기술 개발 대과제 사업수행계획서,” 한국전자통신연구원, 2010

[2] “초저전력 비동기 프로세서 기술 개발 사업수행계획서,” 한국전자통신연구원, 2010

[3] 오명훈 외, “비동기 회로 설계 요소기술 분석서,” 한국전자통신연구원 기술문서, TM20070357, 2007

[4] 김영우, 김성운, “비동기 설계의 기술 및 특허 동향,” 주간기술동향, 제 1301 호, 2007

[5] 김영우 외, “명령어셋(ISA) 정의서,” 한국전자통신연구원 기술문서, TM 200707713, 2007

[6] “ALTHEA 명령어 마이크로코드 상세분석서 V1.0,” 2008 년 11 월, ETRI 서버플랫폼팀

[7] 김성남 외, “비동기식 외부 버스 인터페이스 구조 설계서,” 한국전자통신연구원 기술문서, 2009

[8] 김성남 외, “ALTHEA 마이크로아키텍쳐 설계서 V3.0,” 한국전자통신연구원 기술문서, 2010

[9] 김성남 외, “ALTHEA 블록설계서 V3.0,” 한국전자통신연구원 기술문서, 2010

[10] Instruction Set Reference Manual for AE32000: An 32bit EISC microprocessor, ADChips Inc., 2008

[11] AE32000C(Lucifer): Hardware Reference Manual, ADChips Inc., 2007

[12] ETRI Platform Specification: Lucida Base, ADChips Inc., 2008

[13] EISC Software Developer Guide: Extendable Instruction Set Computer, ADChips Inc., 2008

[14] TiDE Manual, Handshake Solutions, 2009.

[15] HASTE Manual, Handshake Solutions, 2009.

약 어 표

AHB	AMBA High-Performance Bus
ALTHEA	Asynchronous Low power processor based on The EISC Architecture
AFSM	Asynchronous Finite State Machine
AMBA	Advanced Microcontroller Bus Architecture
ARM	Advanced Risc Machines
AXI	Advanced eXtensible Interface
BIU	Bus Interface Unit
BTA	Branch Target Address
CISC	Complex Instruction Set Computer
CU	Co-Processor Unit
DBIU	Data Bus Interface Unit
DE	DEcode stage
DSP	Digital Signal Processing
EISC	Extendable Instruction Set Computer
ER	Extension Register
EX	Execution stage
FPU	Floating Point Unit
GPR	General Purpose Register
IBIU	Instruction Bus Interface Unit
IF	Instruction Fetch stage
ISA	Instruction Set Architecture
ISP	OSI Mode Stack Pointer
JTAG	Joint Test Action Group

LERI	Load Extension Register with Immediate
MAC	Multiply and ACcumulation
MBMU	Memory Bank Management Unit
MMU	Memory Management Unit
MRE	Multiply Result Extend
NMI	Non-Maskable Interrupt
OSI	On-Silicon In-circuit emulator
PC	Program Counter
PCU	Processor Core Unit
RAM	Random Access Memory
RF	Register File
PM	Power Manager / Power Management
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
SIMD	Single Instruction Multiple Data
SoC	System on Chip
SPR	Special Purpose Register
SRAM	Static Random Access Memory
SSP	Supervisor mode Stack Pointer
SWI	SoftWare Interrupt
TCM	Tightly Coupled Memory
TLB	Translate Look-aside Buffer
UART	Universal Asynchronous Receiver/Transmitter
UML	Unified Modeling Language
USP	User mode Stack Pointer
WB	Write Back stage

주 의

1. 이 연구보고서는 한국전자통신연구원의 주요사업으로 수행한 연구 결과 입니다.

2. 이 보고서의 내용을 발표할 때에는 반드시 한국전자통신연구원 에서 수행한 주요사업 결과임을 밝혀야 합니다.