임베디드 RISC 코어의 성능 및 전력 개선

정홍균* · 류광기*

Performance and Power Consumption Improvement of Embedded RISC Core

Hong-kyun Jung* · Kwang-ki Ryoo*

이 논문은 IDEC의 CAD Tool 지원, 중소기업청의 산학협력실 지원사업 및
ETRI 시스템 반도체 산업 진흥센터의 IT SoC 핵심 설계인력 양성 사업의 연구결과임.

요 약

본 논문에서는 임베디드 RISC 코어의 성능 및 전력 소모 개선을 위해 동적 분기예측 구조, 4원 집합연관 캐쉬 구조, ODC 연산을 이용한 클록 게이팅 기법을 제시한다. 동적 분기 예측 구조는 분기 명령에 대해 다음에 실행될 명령에 대한 예측 주소를 저장하는 BTB (Branch Target Buffer)를 사용한다. 4원 집합연관 캐쉬는 네 개의 메모리 블록을 한 개의 캐쉬 블록에 사상되는 구조로서 직접사상 캐쉬에 비해 접근 실패율이 낮고 라인 교체 방식으로 Pseudo-LRU 방식을 채택하여 LRU 정보를 저장하는 비트 수를 감소시킨다. ODC를 이용한 클록게이팅 기법은 논리합성 개념인 무관조건의 입출력 ODC 조건을 찾아 클록 게이팅 로직을 삽입함으로써 동적 소비전력을 줄인다. 제시한 구조들을 임베디드 RISC 코어인 OpenRISC 코어에 적용하여 성능을 측정한 결과, 기존 OpenRISC 코어 대비 실행시간이 약 29% 향상 되었고, Chartered 0.18㎛ 라이브러리를 이용하여 동적 전력을 측정한 결과, 기존 OpenRISC 코어 대비 소비전력이 16% 이상 감소하였다.

ABSTRACT

This paper presents a branch prediction algorithm and a 4-way set-associative cache for performance improvement of embedded RISC core and a clock-gating algorithm using ODC (Observability Don't Care) operation to improve the power consumption of the core. The branch prediction algorithm has a structure using BTB(Branch Target Buffer) and 4-way set associative cache has lower miss rate than direct-mapped cache. Pseudo-LRU Policy, which is one of the Line Replacement Policies, is used for decreasing the number of bits that store LRU value. The clock gating algorithm reduces dynamic power consumption. As a result of estimation of performance and dynamic power, the performance of the OpenRISC core applied the proposed architecture is improved about 29% and dynamic power of the core using Chartered 0.18μm technology library is reduced by 16%.

키워드

OpenRISC, 집합연관 캐쉬, 분기예측 기법, 클록 게이팅, ODC연산, BTB

Key word

OpenRISC, Set-associative Cache, Branch Prediction Algorithm, Clock gating, ODC operation, BTB

* 한발대학교 정보통신전문대학원 정보통신공학과

접수일자 : 2009. 09. 20
심사완료일자 : 2009. 10. 12

I. 서 론

최근 수년간 VLSI 회로의 집적도 향상과 반도체 공정 기술의 발전으로, 임베디드 프로세서의 성능이 향상되었을 뿐 아니라 이를 응용한 네트워크, 가전, 이동통신 등 임베디드 시스템에 사용되고 있다. 이러한 시스템의 발전과정에서 다양한 기능과 성능이 시스템에 적용되어 왔으며, 이는 임베디드 프로세서의 발전에 의해서 가능해졌다.

그러나 이러한 프로세서 설계시 성능 향상과 이로 인한 전력 소모 문제에 대한 고려가 필수적이다. 대부분의 임베디드 프로세서는 정상적인 파이프라인 처리를 위하여 지연 분기를 사용 하지만, 지연 분기에 의한 파이프라인 처리 지연으로 인해 명령어 낭비가 발생하여 성능을 저해하는 중요 요인이 되고 있다. 시스템이 제공하는 물리적 요인에 제한된 메모리 대역폭에 만족하지 못하는 고성능 마이크로프로세서에서 캐쉬는 전체의 성능을 좌우하는 가장 기본적인, 그리고 가장 중요한 구성 요소이다. 또한 코어의 동적 소비전력의 대부분은 클록과 데이터패스 관련 회로에서 차지한다. 특히 신호선 가운데 항상 천이가 발생하고 많은 소자와 연결된 클록에 의한 동적 소비전력을 줄이는 것이 저전력 설계의 핵심이다.

임베디드 프로세서의 성능을 향상시키기 위해서 [1]은 2원 집합연관 캐쉬를 제시하였고, [2]는 BTB (Branch Target Buffer)를 이용한 분기예측 구조를 제시하였다. [1]에서 제시한 캐쉬 구조는 직접사상 캐쉬의 단점인 높은 미스율을 줄이지만 높은 차원의 집합연관 캐쉬에 비해서 미스율이 높다. [2]에서 제시한 구조는 지연분기의 단점인 불필요한 명령어의 삽입을 제거하여 임베디드 프로세서의 성능을 향상시켰으나 캐쉬를 사용하지 않은 임베디드 프로세서에 적용하였다.

따라서 본 논문에서는 임베디드 RISC 코어인 OpenRISC 코어의 성능 및 전력소모의 개선을 위해 BTB를 이용하여 파이프라인 지연을 줄이고 성능을 개선시키는 동적 분기 예측 구조와 직접사상 캐쉬와 2원 집합연관 캐쉬보다 미스율이 낮은 4원 집합연관 캐쉬, 무관조건을 찾는 ODC 연산을 이용하여 클록 게이팅 로직을 삽입하는 저전력 설계방법[3]을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 OpenRISC 코어에 대해 설명하고, 3장에서는 4원 집합연관 캐쉬의 구조에 대해 설명한다. 4장에서는 BTB를 이용한 동적 분기예측 기법에 대해 설명하고, 5장에서는 ODC 연산을 이용한 클록 게이팅 기법에 대해 설명하고, 6장에서는 제시한 기법을 적용한 OpenRISC 코어의 성능 및 전력소모에 대해 논한다. 마지막으로 7장에서는 본 연구의 결론을 도출한다.

II. OpenRISC 코어

OpenRISC 코어는 명령어/데이터 버스 및 메모리가 분리된 하버드 구조로 된 MIPS 기반 RISC 코어이다. RISC 코어의 특징에 맞게 5단 파이프라인 구조를 채택하였으며, 임베디드 시스템을 타깃으로 실시간 운영체제 지원을 위한 메모리 관리 장치를 지원하고 코어 내에 곱셈 및 누산기 유닛을 통해 기본적인 DSP 기능을 지원하고 있다. 또한 코어 내부의 온칩 메모리를 제외한 로직들의 게이트 수가 비교적 작고, 저전력을 위한 파워 관리 블록과 외부 시스템과의 인터페이스를 통한 쉽고 빠른 디버깅 환경, 프로그램 가능한 인터럽트 인식 및 처리, WISHBONE 표준 인터페이스를 통해 명령어 및 데이터 인터페이스를 구성한다[4-5]. 그림 1은 OpenRISC 코어의 구조이다.

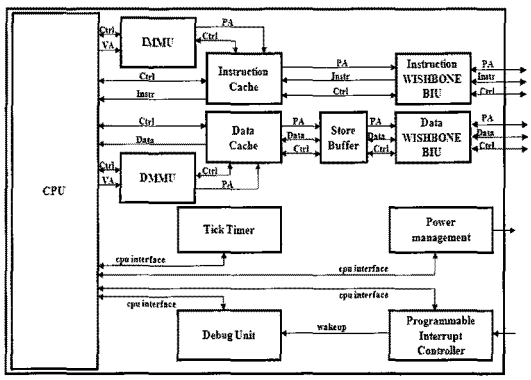

그림 1. OpenRISC 코어 구조
Fig. 1 The architecture of OpenRISC core

Ⅲ. 4원 집합연관 캐쉬

OpenRISC 코어에서 사용한 기존의 직접 사상 방식의 단점, 즉 한 개의 캐쉬 블록에 여러 개의 메모리 블록이 사상될 때의 높은 접근 실패율을 보완하기 위하여 4원 집합 연관 캐쉬를 제시한다. 4원 집합 연관 캐쉬는 사실상 병렬적으로 동작하는 네 개의 직접 사상 캐쉬이다. 캐쉬에 전달되는 주소는 네 개의 캐쉬 중 하나에서 그 데이터를 찾으며, 각 메모리 주소는 네 장소 중 한 곳에 저장된다.

제시한 4원 집합연관 캐쉬는 크기가 8KB이고 그림 2에서 보는 바와 같이 128개의 라인과 태그 비교기, 라인 디코더, 워드 멀티플렉서, 최근 참조된 Way를 저장하는 Pseudo-LRU 구현기로 구성되어 있다.

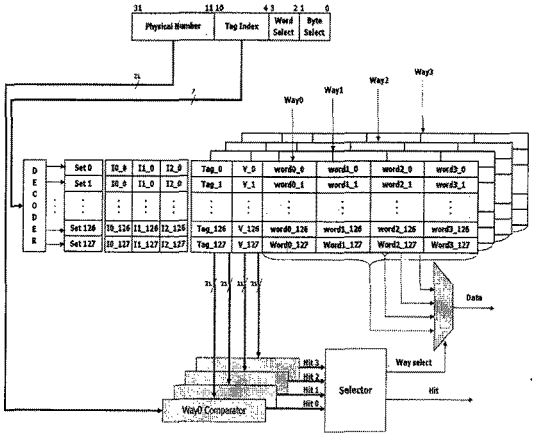

그림 2. 4원 집합연관 캐쉬의 구조
Fig. 2 The architecture of proposed cache

CPU로부터 메모리 참조가 발생하면 4원 집합연관 캐쉬로의 접근이 발생한다. 캐쉬는 CPU로부터 메모리의 접근목적이 읽기요구인지 쓰기 요구인지 판단한다. 만약 읽기 요구이면 4원 집합연관 캐쉬는 해당 라인의 네 개의 주소 태그 램에 저장되어 있는 각각의 21비트의 태그들과 32비트 물리주소의 물리적 영역 주소를 비교한다. 적중이 발생하면 워드 선택 비트에 의해 적중된 집합의 해당라인에 저장되어 있는 4개의 워드 중 한 개를 선택하여 CPU로 전달함과 동시에 참조된 집합을 해당라인의 LRU비트를 갱신한다. 만약 네 개의 주소 태그들과 물리적 영역 주소가 모두 일치하지 않으면 캐쉬 미스가 발생한다. 캐쉬 미스가 발생하면 라인 교체 알고리즘에 의해 결정된 교체 집합의 해당 라인 데이터들을 갱신한다. 그림 3은 CPU로부터 메모리 참조가 발생하였을 때 캐쉬의 동작을 흐름도로 나타낸 것이다.

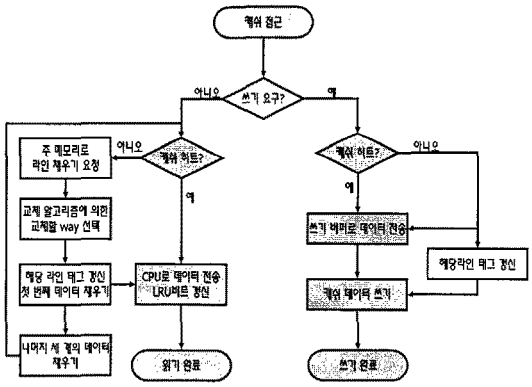

그림 3. 캐쉬 동작 흐름도
Fig. 3 Flow chart of cache operation

제시한 4원 집합연관 캐쉬는 라인 교체 알고리즘으로 Pseudo-LRU 알고리즘을 사용하였다. Pseudo-LRU 알고리즘은 보다 적은 비용과 면적으로 True-LRU 알고리즘의 성능을 구현하도록 고안된 알고리즘으로 다중 집합 구조의 캐쉬에서 사용되는 것으로, 각 집합별로 LRU비트를 유지하되, 몇 개의 way를 하나의 군으로 묶어 관리하는 방법이다[6-7].

4원 집합연관 캐쉬는 4개의 집합으로 구성되어 있으므로 Pseudo-LRU 알고리즘을 적용하기 위해서 Way0와 Way1을 하나의 군, Way2와 Way3을 하나의 군으로 나누어 관리하고 각 라인의 LRU는 3비트로 구현하였다. LRU의 두 번째 비트(LRU[1])는 Way0, Way1의 군과 Way2, Way3의 군을 구별하고, 첫 번째 비트(LRU[0])는 Way0와 Way1을 구별하고, 세 번째 비트(LRU[2])는 Way2와 Way3을 구별한다.

4원 집합연관 캐쉬에서의 LRU비트 갱신 방법은 표 1에서 보는 바와 같이 현재 참조된 way가 Way0나 Way1이면 LRU[1]이 1로 세팅되고 Way2와 Way3이면 LRU[1]이 0으로 세팅된다. LRU[0]은 Way0가 참조되면 1로 세팅되고 Way1이 참조되면 0으로 세팅된다. LRU[2]는 Way2가 참조되면 1로 세팅되고 Way3이 참조되면 0으로 세팅된다.

표 1에서 '-'는 LRU비트가 갱신되기 전의 값을 유지한다는 의미이다.

표 1. LRU비트 갱신 방법
Table. 1 The method of LRU bit update

참조된 way	LRU[2]	LRU[1]	LRU[0]
Way0	-	1	1
Way1	-	1	0
Way2	1	0	-
Way3	0	0	-

Ⅳ. BTB를 이용한 동적 분기예측 기법

지연 분기의 단점인 불필요한 명령어 삽입을 개선하기 위해 제시한 분기예측 기법은 BTB를 사용한 동적 분기예측 기법이다. BTB는 파이프라인의 IF(Instruction Fetch) 단계와 연관있는 캐쉬 메모리로서 이전에 실행한 분기 명령어의 주소와 분기의 목적지 주소를 저장한다. IF단계에서 BTB는 펫치된 명령어의 주소와 BTB 내부에 저장된 이전에 분기된 명령어 주소를 비교한다. 만약 두 주소가 일치하면 분기의 발생여부를 예측한다. 만약 분기가 발생한다고 예측되면 분기 목적지 주소는 다음 PC(Program Counter)로 사용되어 펫치된 명령어를 디코딩할 때 분기 목적지 주소의 명령어를 펫치한다. 따라서 실제로 분기가 실행되었을 때 ID 단계에서 분기 목적지 주소의 명령어를 디코딩하고 있기 때문에 분기에 따른 파이프라인의 지연없이 분기가 실행되는 장점이 있다 [8].

그림4는 OpenRISC 코어에 적용한 BTB의 구조를 나타낸다. BTB는 주소 태그 메모리, 분기 목적지 주소 메모리, 라인 디코더, 태그 비교기로 구성된다. 주소 태그 램은 이전에 분기를 실행했던 분기 명령어의 주소를 저장하고 분기 목적지 주소 램은 분기 주소를 저장한다. 라인 디코더는 펫치된 명령어의 주소와 비교할 BTB의 라인을 선택한다. 태그 비교기는 PC의 태그 필드와 BTB의 해당 라인의 주소태그와 비교한다. BTB는 PC의 라인 인덱스 필드에 해당하는 6비트를 디코딩하여 라인을 선택한다.

선택된 라인의 주소태그와 PC의 태그 필드를 비교하여 일치하면 해당 라인의 분기 목적지 주소를 출력한다.

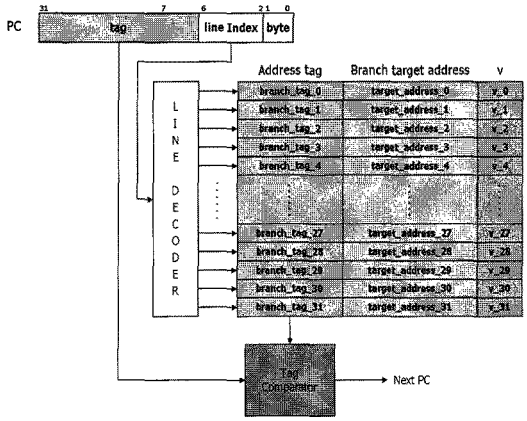

그림 4. BTB의 구조
Fig. 4 The architecture of Branch Target Buffer

그림 5는 BTB의 동작 흐름도를 보여준다. CPU는 명령어를 펫치하기 위해 PC를 명령어 캐쉬와 BTB에 전달한다. 파이프라인의 IF단계에서 BTB는 PC의 태그 필드와 버퍼 내부의 해당 라인의 주소 태그와 비교하여 일치하면 해당 라인의 분기 목적지 주소를 CPU로 전달한다. 펫치된 명령어가 실제로 분기하면 OpenRISC 코어는 분기에 의한 파이프라인의 지연없이 분기를 실행하고, 만약 명령어가 실제로 분기하지 않는다면 파이프 라인에서 펫치된 분기 목적지 주소의 명령어를 지우고 새로운 명령어를 펫치한다.

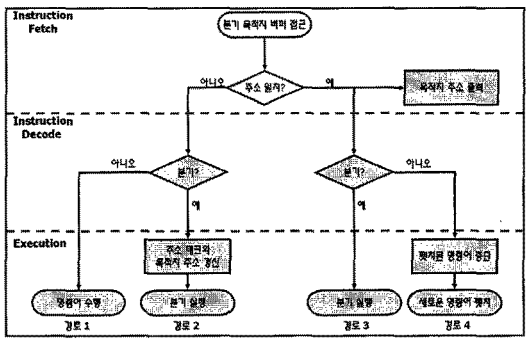

그림 5. BTB 동작 흐름도
Fig. 5 Flow chart of Branch Target Buffer operation

IF단계에서 PC의 태그 필드와 해당 라인의 주소태그가 일치하지 않으면 ID(Instruction Decode)단계에서 분기여부를 검사한다. 만약 분기를 실행하면 BTB는 해당 라인의 주소태그와 분기 목적지 주소를 갱신한다. 만약 분기가 실행되지 않으면 디코딩된 명령어를 수행한다[4].

표 2는 그림 5의 경로들에 대한 분기 패널티를 나타낸다. 경로 1은 분기가 실행되지 않기 때문에 분기 패널티가 0이고, 경로 2와 경로 3은 분기가 실행되었을 때 분기 목적지 주소의 명령어를 펫치하기 때문에 분기 패널티가 2가 된고, 경로 4는 분기가 실행되기 전에 분기 목적지 주소의 명령어를 펫치했기 때문에 분기 패널티가 0이다.

표 2. 분기 목표 버퍼에 의한 경로들의 분기 패널티
Table. 2 Branch panalties of the paths of BTB

경로	히트/미스	예측	분기조건	분기 패널티
1	미스	-	불일치	0
2	미스	-	일치	2
3	히트	예측	불일치	2
4	히트	예측	일치	0

V. ODC 연산을 이용한 클록 게이팅 기법

일반적인 순차회로에서의 레지스터는 클록 천이에 의해 출력을 내보내게 된다. 그러나 레지스터의 출력이 조합회로를 통과하여 출력으로 전해지는 과정에서 레지스터의 출력이 무의미해지게 되는 경우가 무관조건이 된다.

그림 6은 데이터 패스의 예제회로이다. 3상태 버퍼의 제어신호 ENB가 0이 되면 3상태 버퍼의 입력에 상관없이 출력인 OutData는 하이 임피던스 값을 가지므로 3상태 버퍼의 입력인 덧셈기(Adder)의 연산 결과는 무관조건이 된다. 덧셈기의 출력이 무관조건이면 덧셈기의 입력인 DataA, DataB, S의 값도 무의미하다. 따라서 ENB가 0이 되면 DataA, DataB, S의 값이 무의미하므로 ENB가 0인 경우에는 REG1과 REG2에 클록을 인가할 필요가 없다. 멀티플렉서(MUX)의 선택신호인 Mux_sel이 0이면 DataA가 선택되므로 DataB는 무관조건이고, Mux_sel이 1이면 DataA가 무관조건이다. 따라서 Mux_sel이 0인 경우 REG2에 클록을 인가할 필요가 없고, Mux_sel이 1인 경우에는 REG1에 클록을 인가할 필요가 없다. 위와 같이 입력 데이터의 무관조건을 연산하여 클럭 게이팅 로직을 설계하는 방법이 ODC연산을 이용한 클록 게이팅 기법이다. 이 기법은 클럭을 게이팅함으로써 입력 데이터의 천이를 방지하여 입력 데이터와 연결된 로직들의 천이도 방지하므로 전력 소모를 감소시킨다[9].

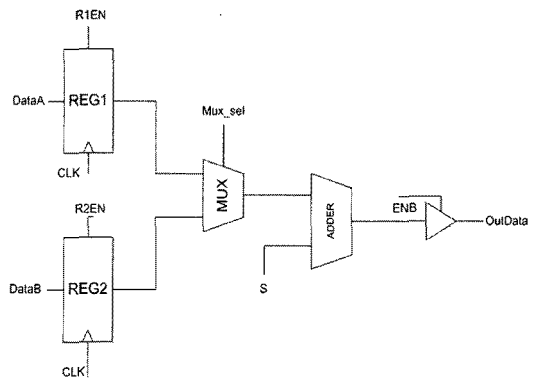

그림 6. 데이터 패스 회로
Fig. 6 Circuit of data path

식 1과 식 2는 그림 6의 입력데이터인 DataA와 DataB의 ODC 조건을 나타낸다. DataA는 ENB가 0이거나 MUX_sel이 1이거나 R1EN이 0인 경우에 무관조건이다. DataB는 ENB가 0이거나 MUX_sel이 0이거나 R2EN이 0인 경우에 무관조건이다.

$$ODC(DataA) = R1EN' + MUX\_sel' + ENB' \quad (1)$$

$$ODC(DataB) = R2EN' + MUX\_sel + ENB' \quad (2)$$

그림 7은 그림 6의 데이터 패스 회로에 ODC 연산 회로와 클록 게이팅 로직을 추가한 회로이다. ODC연산 회로는 입력 데이터 DataA와 DataB의 무관조건을 검사하고 클록 게이팅 로직은 ODC연산 호리의 제어 신호에 의해 REG1과 REG2 클록의 천이를 제어한다.

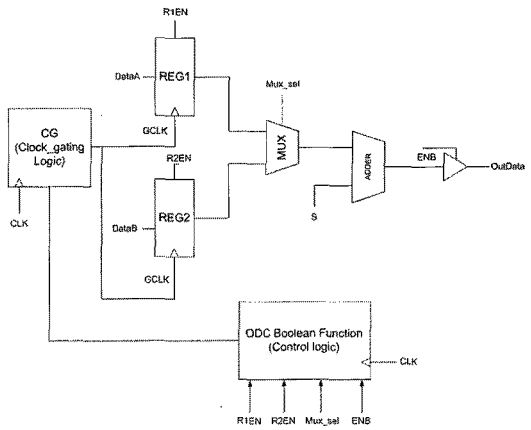

그림 7. ODC연산을 적용한 데이터 패스 회로
Fig. 7 Data path circuit applied ODC operation

ODC연산을 이용한 클록 게이팅 기법을 이용하여 OpenRISC 코어의 전력 소모를 줄이기 위해 OpenRISC 코어의 오퍼랜드 멀티플렉서의 입력 데이터에 ODC 연산을 적용하였다. 오퍼랜드 멀티플렉서는 레지스터 파일에서 두 개의 오퍼랜드를 전달받아 산술논리장치(ALU), 로드-스토어 장치(LSU) 등의 모듈에 전달하는 역할을 한다. 그림 8은 오퍼랜드 멀티플렉서와 입력데이터인 레지스터 파일의 블록 다이어그램을 나타낸다.

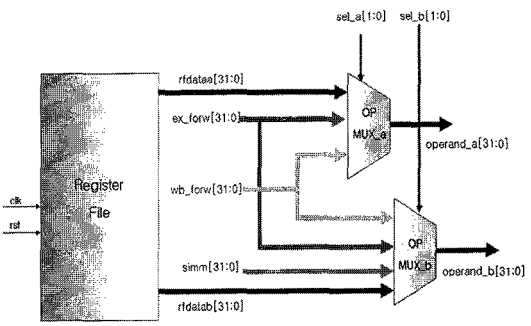

그림 8. 오퍼랜드 멀티플렉서의 블록 다이어그램
Fig. 8 Block diagram of the Operand Multiplexor

오퍼랜드 멀티플렉서는 선택 신호인 sel_a와 sel_b의 상태에 따라서 실행단계에서 포워딩된 데이터(ex_forw), 후기입(write back) 단계에서 포워딩된 데이터(wb_forw), 즉시 데이터(simm), 레지스터 파일에서 전송되는 데이터(rfdataa, rfdatab)중 하나를 선택하여 각각 오퍼랜드 a, 오퍼랜드 b로 출력한다.

표 3은 오퍼랜드 멀티플렉서의 선택신호의 상태에 의해 오퍼랜드 a와 오퍼랜드 b로 출력되는 입력 데이터를 보여준다. 오퍼랜드 멀티플렉서 a의 선택신호 값이 "00"과 "01"인 경우 레지스터 파일의 데이터 a가 선택되고, 오퍼랜드 멀티플렉서 b의 선택신호 값이 "00"인 경우 레지스터 파일의 데이터 b가 선택된다.

표 3. 선택신호에 의한 오퍼랜드
Table. 3 Operand for select signals

sel_a[1:0]	operand a	sel_b[1:0]	operand b
00	rfdataa	00	rfdatab
01	rfdataa	01	simm
10	ex_forw	10	ex_forw
11	wb_forw	11	wb_forw

OpenRISC 코어의 오퍼랜드 멀티플렉서에 ODC연산을 적용하기 위해서 오퍼랜드 멀티플렉서를 원-핫 멀티플렉서로 수정하였다. 원-핫 멀티플렉서란 선택신호의 비트 수를 입력 데이터의 개수와 같게 하여 선택신호와 입력데이터를 일대일 대응시켜 한 비트가 활성화되면 그에 대응하는 입력데이터를 출력하는 기능을 수행하는 데이터 선택 논리회로이다. 원-핫 멀티플렉서로 수정된 오퍼랜드 멀티플렉서는 선택신호가 2비트에서 4비트로 증가한다. 표 4는 수정된 오퍼랜드 멀티플렉서에서 선택신호에 따라 출력되는 입력데이터를 보여준다.

표 4. 수정된 멀티플렉서의 선택신호에 대한 오퍼랜드
Table. 4 Operand for select signals of modified multiplexor

sel_a	operand a	sel_b	operand b
0001	rfdataa	0001	rfdatab
0010	rfdataa	0010	simm
0100	ex_forw	0100	ex_forw
1000	wb_forw	1000	wb_forw

오퍼랜드 멀티플렉서 a의 선택신호가 "0001"이거나 "0010"일 경우 레지스터 파일의 데이터a가 선택되고 오퍼랜드 멀티플렉서 b의 선택신호가 "0001"인 경우 레지스터 파일의 데이터 b가 선택된다. 따라서 식 3, 4에서 보는 바와 같이 레지스터 파일의 데이터 a의 ODC조건은 오퍼랜드 멀티플렉서 a의 선택신호의 첫 번째 비트가 0이거나 두 번째 비트가 0인 경우이고 레지스터 파일의 데이터 b의 ODC조건은 오퍼랜드 멀티플렉서 b의 선택신호의 첫 번째 비트가 0인 경우이다.

$$ODC(rfdataa) = sel\_a[0]' + sel\_[1]' \quad (3)$$

$$ODC(rfdatab) = sel\_b[0]' \quad (4)$$

그림 9는 ODC 연산을 이용한 클록 게이팅 로직을 추가한 오퍼랜드 멀티플렉서와 레지스터 파일의 블록다이어그램을 나타낸다. 레지스터 파일의 클록 게이팅 로직은 ODC 조건에 의해 오퍼랜드 멀티플렉서의 선택신호의 첫 번째 비트로 제어된다.

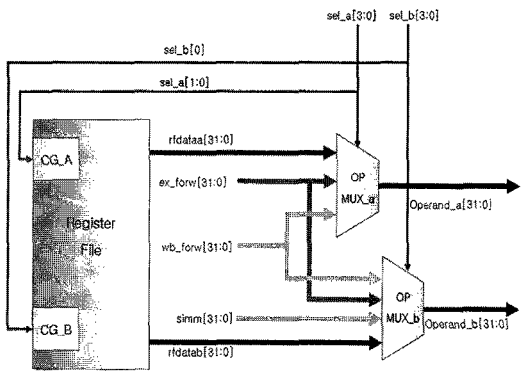

그림 9. 클록 게이팅 로직을 삽입한 오퍼랜드 멀티플렉서의 블록 다이어그램
Fig. 9 Block diagram of operand multiplexor added clock-gating logic

VI. 실험 및 고찰

기존의 OpenRISC 코어와 제안된 OpenRISC 코어의 성능을 비교하기 위해 H.264 복호기의 역변환 및 역양자화 기능 블록을 테스트프로그램으로 사용하였다. H.264 복호기의 역변환 및 역양자화 기능 블록은 16x16 크기의 매크로 블록을 처리하는 동안 역양자화 및 역변환을 순차적으로 26번 수행한다. 그림 10은 역양자화를 수행한 후 역변환을 수행하기 전까지 실행시간을 기준으로 기존의 OpenRISC 코어와 제안된 OpenRISC 코어의 실행 사이클 수를 나타낸다. 첫 번째 역양자화를 수행하고 역변환을 수행하는 동안 기존의 코어와 제안된 코어의 실행 사이클 수는 260개로 동일하였으나, 역양자화 및 역변환을 반복적으로 수행한 경우 제안된 코어의 실행 사이클 수는 115개로 감소하였고, 기존 코어의 실행 사이클 수는 206개로 감소하였다. 또한 18번 째 역양자화를 수행한 후 Hadamard 역변환을 수행한 후 기존 코어의 실행 사이클 수는 97개로 일정하였고, 제안된 코어의 실행 사이클 수는 97개에서 37개로 감소하였다. 따라서 제안된 코어의 성능이 기존의 코어에 비해 46%이상 향상되었다.

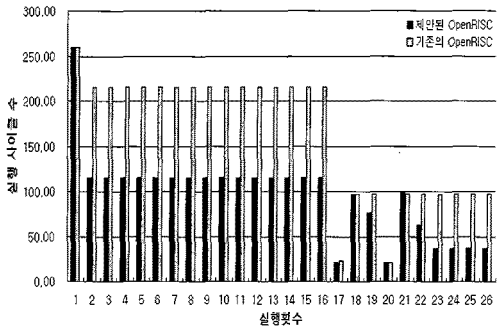

그림 10. 역양자화 후 역변환에 대한 실행 사이클 수 비교
Fig. 10. The comparison of the number of execution cycle for the inverse transform after inverse quantization

그림 11은 역변환을 수행한 후 역양자화를 수행하기 전까지 코어가 실행한 사이클 수를 나타낸다. 역변환 후 역양자화를 반복해서 수행하는 동안 제안된 코어의 실행 사이클 수는 1,108개이고, 기존 코어의 실행 사이클 수는 1,581개이다. 또한 Hadamard 역변환을 수행한 후 기존 코어의 실행 사이클 수는 1,061개이고 제안된 코어의 실행 사이클 수는 701개이다. 따라서 제안된 코어의 성능이 기존의 코어보다 29%이상 향상되었다.

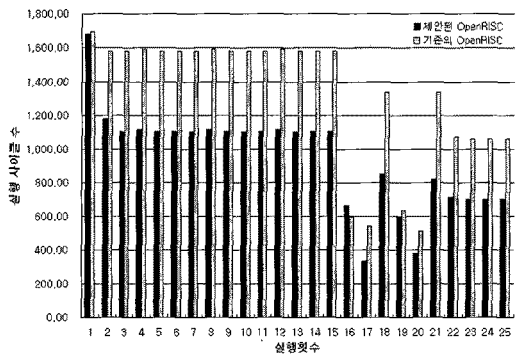

그림 11. 역변환 후 역양자화에 대한 실행 사이클 수 비교
Fig. 11. The comparison of the number of execution cycle for the inverse quantization after inverse transform

그림 12는 역양자화를 수행한 후 역변환을 수행하기 전까지의 실행시간을 기준으로 기존 코어와 제안된 코어의 미스율을 나타낸다. 첫 번째 실행에서 기존 코어와 제안한 코어의 미스율은 20.37%로 동일하였으나, 반복 횟수가 증가할수록 제안한 코어의 미스율은 4.5%로 감소하였고, 기존 코어는 21.15%로 증가하였다. 또한 Hadamard 역변환을 수행하였을 때 기존 코어와 제안한 코어의 미스율은 24%로 동일하였으나, 횟수가 증가하면서 제안한 코어의 미스율은 5%로 감소하였고, 기존 코어의 미스율은 24%로 증가하였다. 따라서 제안한 코어의 미스율은 기존 코어의 미스율보다 15%이상 감소했다.

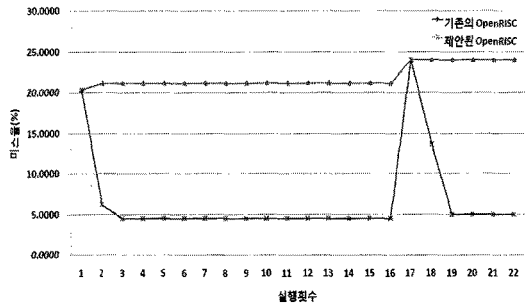

그림 12. 테스트 프로그램에 대한 미스율 비교
Fig. 12. The comparison of the miss ratio for test program

제시한 저전력 설계의 동적 전력을 비교하기 위해서 Chartered 0.18㎛ 셀 라이브러리를 이용하여 Synopsys 사의 Design Compiler와 Power Compiler를 이용하여 각각 로직 합성을 수행하고 전력소모를 측정하였다. 표 5에서 보는 바와 같이 ODC 연산을 이용한 클록 게이팅 설계기법을 적용한 OpenRISC의 동적 전력이 저전력을 고려하지 않은 OpenRISC에 비해 16.2% 감소한 것을 보여준다.

표 5. 소비전력의 비교
Table. 5 Comparison of power consumption

소비전력	제안된 OpenRISC	OpenRISC	차이
셀	36.7583㎽	42.5833㎽	15.8%
넷 스위칭	3.9256㎽	4.6862㎽	19.4%
합계	40.6839㎽	47.2694㎽	16.2%

Ⅶ. 결론

본 논문에서는 임베디드 RISC 코어의 성능 향상 및 전력 소모 개선을 위해 4원 집합연관 캐쉬, BTB를 이용한 분기예측 기법 및 ODC연산을 이용한 클록 게이팅 회로를 제시한다. 4원 집합연관 캐쉬는 한 개의 블록에 네 개의 메모리 블록이 사상되는 구조로 되어있고 Pseudo-LRU 알고리즘을 사용하여 임베디드 RISC 코어인 OpenRISC 코어의 캐쉬의 높은 접근실패율과 교체 알고리즘의 회로의 크기를 감소시켰다. BTB를 이용한 분기예측 기법은 분기 주소와 분기 목적지 주소를 저장하는 BTB를 사용하여 분기 발생시 발생하는 지연 사이클 수를 감소시켰다. ODC연산을 이용한 클록게이팅 로직은 입력의 무관 조건을 찾아 입력 레지스터의 클록을 게이팅하여 OpenRISC 코어의 동적소비전력을 감소시켰다. 테스트 프로그램을 이용하여 성능 및 동적 소비전력을 측정한 결과, 기존의 OpenRISC 코어 대비 제시한 OpenRISC 코어의 성능이 29% 이상 향상되었고, 동적 소비 전력은 약 16% 감소하였다. 따라서 제시한 구조를 적용한 OpenRISC 코어가 기존의 OpenRISC 코어에 비해 성능 및 전력소모가 개선된다는 결론을 이끌었다.

참고문헌

[1] Hongkyun Jung and Kwangki Ryoo, "The Design of Cache Architecture in 32-bit RISC for the Performance Improvement," *ITC-CSCC 2007*, vol. 2, pp. 308-309, 2007

[2] 김형준, 강광명, 류광기, "OpenRISC 프로세서의 저전력 설계와 성능 개선," 제4회 국방정보 및 제어기술 학술대회, pp. 79-81, 2008

[3] Pietro Babighian, Luca Benini and Enrico Macii, "A Scalable Algorithm for RTL Insertion of Gated Clocks Based on ODCs Computation," *IEEE Trans. Computer Aided Design of Integrated Circuits and System*, vol. 24, no. 1, pp. 29-42, 2005

[4] Damjan Lampret, *OpenRISC1200 IP Core Specification Revision 0.7*, 2001

[5] Damjan Lampret, *OpenRISC1000 Architecture Manual*, 2003

[6] 이종익, 손승일, 이문기, "캐쉬 메모리에서 True-LRU 알고리즘과 Pseudo-LRU 알고리즘의 성능 비교," 정보과학회논문지 제23권 제11호, pp. 1148-1160, 1996

[7] Nikitas Alexandridis, *Design of Microprocessor based systems*, Prentice Hall, 1993

[8] Lee, J.K.F. and Smith, A.J., "Branch Prediction Strategies and Branch Target Buffer Design," *IEEE Computer Magazine*, vol.17, no.1, pp. 6-22, Jan. 1984

[9] Christian Piguet, *Low-Power CMOS Circuits*, CRC Press, 2006

저자소개

정홍균(Hongkyun Jung)

2007년 한밭대학교 정보통신공학과 공학사
2009년 한밭대학교 정보통신공학과 공학석사
2009년~현재 한밭대학교 정보통신공학과 박사과정
※관심분야: 임베디드 프로세서, SoC 플랫폼 설계, 하드웨어/소프트웨어 통합설계, 멀티미디어 코덱 설계

류광기(Kwangki Ryoo)

1986년 한양대학교 공과대학 전자공학과 공학사
1988년 한양대학교 대학원 전자공학과 공학석사
2000년 한양대학교 대학원 전자공학과 공학박사
1991년~1994년 육군사관학교 교수부 전자공학과 전임강사
2000년~2002년 한국전자통신연구원 시스템 IC 설계팀 선임연구원
2003년~현재 한밭대학교 정보통신공학과 부교수
※관심분야: SoC 플랫폼 설계 및 검증, 하드웨어/소프트웨어 통합설계 및 통합검증, 멀티미디어 코덱 설계