Guarded Operation을 이용한 명령어 어드레싱 방법 및 구현

이세환, 곽승호, 이문기
연세대학교 전기전자공학과
전화 : 02-2123-4731 / 핸드폰 : 011-9026-1683

Instruction addressing method and implemetation for low power system by using guarded operation

Se Hwan Lee, Sung Ho Kwak, Moon Key Lee
Dept. of Electrical and Electronic, Yonsei University
E-mail : geni@spark.yonsei.ac.kr

Abstract

In this paper, we present a effective low-power technique which can reduce significantly the switching activity in instruction address bus, pipeline and I-cache. Using this method, named *Guarded Operation*, we has implemented address register, address bus architecture without complex hardware and designed loop buffer without tag. These architectures reduce 67% of switching activity with little overhead and also increase instruction-fetch performance.

I. 서론

최근에 프로세서의 저전력 특성은 급속히 각광받고 있는 휴대용 통신, 컴퓨팅 시스템뿐만이 아니라, 고성능을 위주로 하는 하이엔드 프로세서 시장에서도 중요한 성능 지표로 평가받고 있다. 전체 전력 소모의 90% 이상을 차지하는 switching activity를 줄이기 위한 방법으로 특정 블록에 불필요한 클럭 천이가 발생하지 않도록 하는 clock gating방법이 있다. 그리고, 신호의 천이 횟수를 줄이기 위해 데이터를 특정한 알고리즘으로 인코딩하는 Gray Code, Bus-Invert, T0 [1]같은 방법이 있다. 또한, 전력 소모의 많은 양이 외부 메모리 액세스에서 발생하므로 메모리 액세스를 줄이기 위해 Filter Cache나 Loop Cache [2]같은 작은 버퍼를 추가하거나 Code Compression을 통한 연구가 이루어지고 있다.

본 논문에서는 Guarded Operation을 위한 명령어 어드레스 레지스터와 파이프라인 구조를 제안하고 합성 가능한 HDL언어로 구현한 후 Guarded Operation에 기반한 어드레스 인코딩 기법과 명령어 루프 버퍼의 설계를 통해 좀 더 효과적인 저전력 기법을 제안한다.

II. Guarded Operation

2.1 기본 구조

대부분의 전력 소모가 신호의 동적 천이에 의해 발생하는 CMOS 로직에서 불필요한 신호 천이의 횟수를 줄이는 것이 저전력 설계를 위해 중요하다. 신호 천이의 횟수를 줄이는 가장 효과적인 방법은 입력 데이터의 상호 연관성(correlation)을 높이는 것으로 operation binding, loop pipelining, operand sharing, operand reordering, guarded evaluation등이 있는데, 일반적인 guarded evaluation이 산술연산 로직 블록에 대한 연구에 집중되어 왔는데, 본 논문에서는 명령어 어드레스와 파이프라인 구조에 적용하였다. 아래의 그림. 1에 기본적인 구조를 도식하였다.

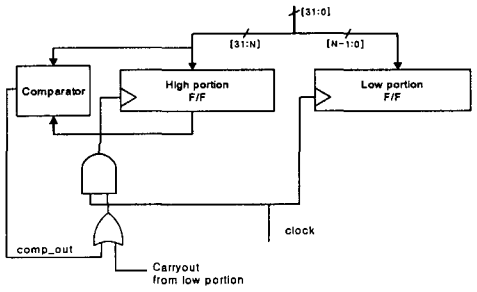

그림 1 guarded operation 레지스터 기본 구조

명령어 어드레스의 변화는 어드레스 인크리멘터에 의해 규칙적으로 변화하는 경우와 분기 명령 등을 통해 다른 명령어 코드 지역으로 이동하는 경우가 있게 된다. 첫 번째 경우는 덧셈 연산에 의한 신호의 천이가 발생한 것이므로 하위 블록의 캐리 아웃을 검사하여 상위 블록의 데이터 전달 여부를 결정하도록 하였으며, 두 번째 경우는 기존의 값과 새로 입력되는 값을 비교하여 다른 경우에만 상위 블록 값을 전달하도록 해서 불필요한 신호 전달을 제거하였다.

2.2 상위 블록의 크기 결정

레지스터를 2개 이상의 블록으로 나누게 되는데, 블록의 개수와 크기를 결정하는 것이 가장 중요한 문제가 된다. 본 논문에서는 SimpleScalar라는 시뮬레이터를 이용해서 명령어 어드레스의 profiling을 통해 결정하도록 하였다. 또한, 명령어 파이프라인 블록을 SimpleScalar를 이용해 C 언어로 모델링하고 HDL로 합성가능하도록 작성하여 측정된 전력 소모비와 추가된 면적과 지연시간에 대한 데이터를 얻어서 trade-off 관계를 통해 적당한 크기를 결정하였다. 분석한 벤치마크 프로그램과 명령어 개수는 아래의 표. 1과 같다.

benchmark	# of insts.	benchmark	# of insts.
math	216084	parser	8111328
fmath	54189	gcc	19679240
llong	30243	vpr	410669
printf	1868429	mesa	1143245
lswlr	8771	bzip2	4100000000
gzip	4200000000	perlbmk	1414399925

표 1 벤치마크 프로그램과 명령어 개수

(1) 명령어 어드레스 profile

그림. 2는 각 명령어 어드레스의 비트당 천이 횟수를 분석해서 전체 천이 횟수에 대한 누적 비율로 나타낸 그림이다.

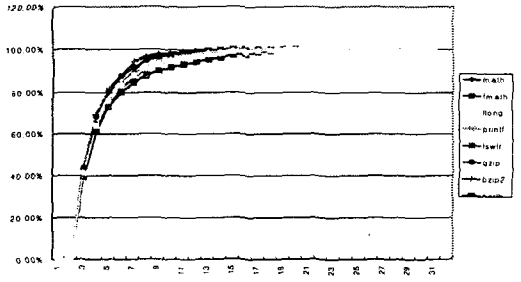

그림 2 전체 천이 횟수에 대한 비트별 천이 횟수의 누적 백분율

그림. 3은 두 개의 블록으로 나눌 때 Guarded Operation에 의해서 상위 블록의 신호 천이에 의해서만 상위 블록이 동작하게 되므로 각 비트에 대해서 한 사이클당 상위 블록 동작 비율의 평균값을 나타낸 그림이다.

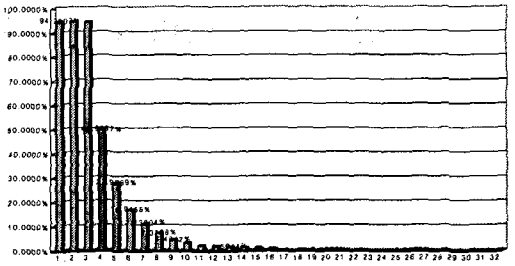

그림 3 한 사이클당 상위 블록 동작 백분율

위 그림에서 만약 상위 블록을 [31:9]으로 결정하면 상위 블록의 동작 비율은 각 명령어당 4.48%가 된다. 즉, 100개의 명령어가 진행되는 동안 상위 블록은 4~5번만 동작하게 되는 것이므로 guarded operation을 이용하면 나머지 95~96번 동안의 필요 없는 신호 천이를 줄일 수 있게 된다.

(2) 전력과 지연시간 Trade-off

그림. 4는 실제 저전력 32비트 마이크로 프로세서 설계에 적용할 명령어 어드레스 파이프라인 구조를 간단히 도식한 것이다.

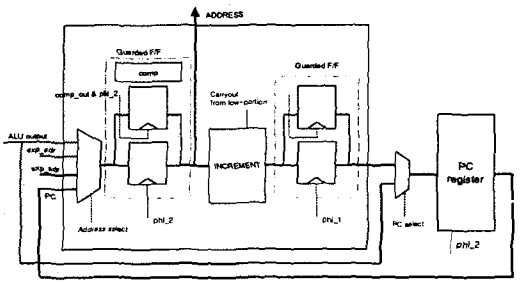

그림 4 명령어 어드레스 레지스터 구조

위와 같은 구조를 HDL 언어로 합성 가능하도록 설계한 후 하위 블록의 크기를 {N-1:0}으로 하고, 상위 블록의 크기를 [31:N]으로 하여 N의 크기가 달라짐에 따라 추가된 면적과 지연시간을 나타내었다.(표. 2)

N	전체 추가된 delay의 %	전체 추가된 area의 %	N	전체 추가된 delay의 %	전체 추가된 area의 %
4	29.210%	14.801%	13	19.821%	10.027%
5	23.696%	14.375%	14	18.927%	9.543%
6	26.080%	13.774%	15	18.927%	9.000%
7	27.273%	13.229%	16	18.629%	8.457%
8	24.590%	12.746%	17	18.331%	7.914%
9	24.888%	12.143%	18	18.331%	7.430%
10	23.696%	11.600%	19	18.927%	6.827%
11	23.398%	11.116%	20	18.331%	6.282%
12	23.994%	10.571%			

표 2 N값에 대한 추가된 면적과 지연시간

전력 측정은 각 입력포트를 앞 절에서 수행한 명령

어 어드레스 profiling을 이용해 얻은 값을 바탕으로 Synopsys의 Design Power의 토글 비율(toggle rate)을 이용한 확률적 모드로 산출했다. 클럭 주파수는 100MHz이고, 각 포트에 대한 node capacitance는 전체 마이크로 프로세서 디자인 안의 연결된 외부 블록을 통해 결정했으며 전력 측정에 필요한 parameter 정보가 포함되고 Design Power에 최적화된 0.25um 셀 라이브러리를 적용해서 구했다.

표. 3은 N값에 따라 측정된 전력과 guarded operation을 사용하지 않은 디자인과의 비교를 통해 감소된 전력 소비의 비를 제시했다.

N	Power	감소된 Power %	N	Power	감소된 Powcr %
4	82.53767	31.04%	13	78.82095	34.15%
5	74.83678	37.48%	14	79.77165	33.36%
6	76.09184	36.43%	15	81.95851	31.53%
7	74.92727	37.40%	16	82.74569	30.87%
8	75.37454	37.03%	17	83.15336	30.53%
9	75.65941	36.79%	18	83.59412	30.16%
10	74.31702	37.91%	19	86.21008	27.98%
11	76.65675	35.96%	20	86.58801	27.66%
12	78.54076	34.38%	org	119.6965	0.00%

표 3 N값에 대한 소비된 전력과 감소 백분율

(3) 결과 분석

그림. 5는 표. 2와 표. 3의 결과를 그래프로 나타낸 것으로 가장 많은 전력 감소율을 보이는 것이 N이 10일 때이지만, 지연시간과 면적을 고려해서 power-delay product를 계산하면, N이 14일 때가 가장 효율적인 것임을 알 수 있다.

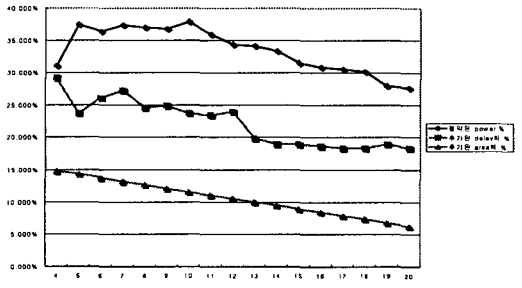

그림 5 감소된 전력과 추가된 면적과 지연시간

III. 어드레스 인코딩의 이용한 Guarded Operation

3.1 명령어 어드레스 인코딩

표. 4는 SimpleScalar 시뮬레이터에서 각 벤치마크 프로그램을 실행했을 때 연속적인 명령어 어드레스를 갖는 부분의 비를 나타낸 것으로 평균 84%의 명령어 어드레스가 워드 크기만큼 더해지는 연속적인 천이를 한다는 것을 알 수 있다.

benchmark	In-seq. addr.	비율
math	190477	88.15
fmath	47409	87.49
llong	26229	86.73
printf	1607893	86.06
lswlr	7764	88.52
gzip	3111862404	74.09
bzip2	2929086580	71.44
perlbmk	1202490453	85.02
mesa	1044904	81.40
parser	6413040	79.04
gcc	17365517	88.24
vpr	346169	84.29
평균	-	84.21

표 4 연속적인 어드레스의 개수와 그 비율

이와 같은 사실로부터 명령어 어드레스 인코딩을 통한 저전력 방법을 Guarded Operation에도 적용할 수 있다. 즉, 모든 비트에 어드레스 인코딩을 적용하는 것이 아니라, 하위 블록이 연속적인 어드레스 천이가 더 많은 확률로 발생하므로 하위 블록에만 어드레스 인코딩을 적용하는 것이다. 이것은 비트 수가 늘어남에 따라 증가하는 인코딩/디코딩 하드웨어의 오버헤드를 상당 부분 줄이면서, 어드레스 인코딩의 효과를 최대한 얻을 수 있게 된다.

3.2 인코딩 방법

연속적인 신호일 경우 가장 좋은 방법으로 알려진 것은 Gray code 인코딩 방법이다. 순차적인 값을 가질 때 Gray code는 오직 한 비트의 천이만을 가지므로 최소한의 신호 천이를 갖게 된다. 그러나 비트 수가 많아지면 명령어 인코딩과 디코딩 회로도 비례해서 커지게 되고 이 회로는 critical path에 놓이게 되므로 전체적인 성능 저하를 가져온다.

본 논문에서는 Gray code대신 어드레스를 받는 메모리에 덧셈기를 추가하여 연속적인 어드레스가 들어올 때는 추가적인 제어 신호를 통해 덧셈기를 동작하게 하도록 하고 실제 전달되는 어드레스 신호는 전의 값을 유지하도록 해서 신호 천이가 일어나지 않도록 하는 Guarded Operation을 적용했다. 이 구조는 그림. 4의 구조를 다음과 같이 변형함으로써 구현된다.

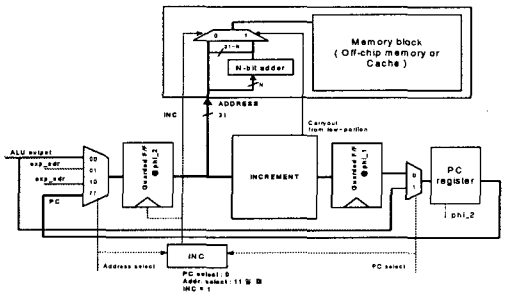

그림 6 어드레스 인코딩을 적용한 어드레스 레지스터 구조

위의 그림과 같이 연속적인 명령어 어드레스가 발생할 때 간단한 로직을 통해 INC 신호를 발생시키므로

따로 복잡한 인코딩 회로가 필요 없다. 그리고 그 신호에 따라 순차적으로 증가하는 어드레스일 경우에는 메모리 로직에 N-비트 덧셈기를 두어서 N-비트의 하위 블록의 어드레스 값만 증가시킨다. 만약 연속적인 어드레스이지만, 상위 블록의 값을 변경하게 되는 경우(하위 블록의 캐리아웃발생) 단순한 구조를 위해 이러한 인코딩 방법을 포기하고 완전한 어드레스 변화를 가지도록 한다.

3.3 인코딩 후 결과 비교

하위 블록의 순차적인 천이를 제거하는 Guarded Operation을 C 언어를 이용해 SimpleScalar 시뮬레이터로 모델링하였다. 인코딩을 하지 않은 경우와의 신호 천이 횟수와 매 명령어당 천이 하는 백분율이 표. 5와 그림. 7에 제시되었다.

benchmark	인코딩전 전체 천이 횟수	인코딩후 전체 천이 횟수	감소 비율
math	483910	155334	67.90%
fmath	126181	40837	67.64%
llong	68479	23223	66.09%
printf	4243309	1639333	61.37%
lswlr	20276	4678	76.93%
gzip	9088180500	2713468185	70.14%
bzip2	8921508354	2172602535	75.65%
perlbmk	3356650872	1636633425	51.24%
mesa	2472272	398124	83.90%
parser	18276502	6454430	64.68%
gcc	44627537	14602087	67.28%
vpr	906125	383721	57.65%
평균	-	-	67.54%

표 5 어드레스 인코딩 후 줄어든 천이 횟수와 백분율

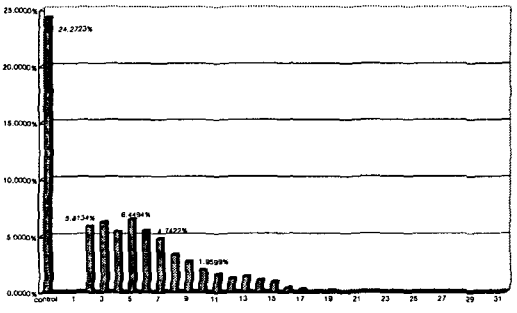

그림 7 인코딩후 32비트 어드레스와 2비트의 제어신호의 매 명령어당 천이 확률의 평균값

그림에서 알 수 있듯이 INC와 carryout 제어 신호의 천이 확률이 24%가 되지만, 하위 블록 비트들의 확률은 10%이하로 대폭적으로 줄어들어서 전체적으로 switching activity가 평균 67.54%가 감소했음을 확인했다.

IV. 명령어 루프 버퍼

프로세서 코어와 L1 캐쉬 사이에 작은 명령어 버퍼를 두는 filter cache, loop cache등의 방법은 외부 메모리나 캐쉬 억세스 횟수를 줄이므로 전력 소모를 줄이는 데 효과적이지만, 캐쉬 미스에 의한 성능 저하가 존재한다. 제안하는 루프 버퍼는 태그가 필요치 않으며 다음 어드레스가 버퍼에 히트될지 않을 지를 간단한 제어 로직을 통해 한 사이클 전에 정확히 알 수 있으므로 미스가 발생하지 않아 성능저하가 없다.

또한, 어드레스 파이프라인의 Guarded Operation으로 하위 블록만의 천이와 앞 절에서 설명한 어드레스 인코딩을 통한 천이 횟수의 감소로 루프 버퍼 자체의 전력 소모도 줄어들게 된다.

아래의 그림. 8은 명령어 루프 버퍼의 기본적인 구조를 나타낸다.

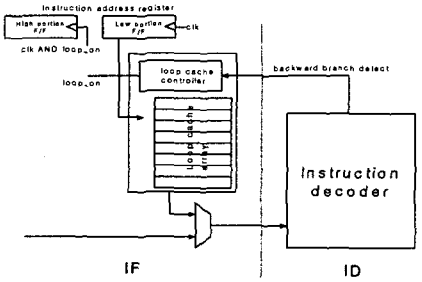

그림 8 명령어 루프 버퍼의 기본 구조

V. 결론

본 논문에서 제안한 Guarded Operation을 명령어 어드레스에 적용함으로써 30%~70%의 switching activity의 감소를 얻을 수 있고, 루프 버퍼에의 적용을 통해 명령어 페치의 성능향상 뿐이니라 더 간단한 하드웨어로 더 많은 전력을 절약할 수 있음을 확인했다.

Guarded Operation의 이점은 상당한 양의 switching activity의 감소가 성능저하 없이 아주 적은 하드웨어의 추가만으로 가능하다는 점이다. 이 방법은 저전력 저비용의 프로세서뿐 아니라 고성능의 하이엔드 프로세서에도 성공적으로 적용할 수 있을 것으로 본다.

참고문헌(또는 Reference)

[1] Benini, L.; De Micheli, G.; Macii, E.; Sciuto, D.; Silvano, C., "Address bus encoding techniques for system-level power optimization", Proceedings , 1998 p.861-866

[2] Anderson, T.; Agarwala, S, "Effective hardware-based two-way loop cache for high performance low power processors", Proc.. 2000 International Conference on , 2000, p403-407

[3] Canal, R., Gonzalez, A. and Smith, J.E., "Very low power pipelines using significance compression", Proc. 33rd Annual IEEE/ACM International Symposium on , 2000, p.181-190

[4] Junghwan Choi, Jinhwan Jeon and Kiyoung Choi "Power minimization of functional units by partially guarded computation", Proc. of the 2000 International Symposium on , 2000, p.131-136