파이프라인 구조를 도입한 고성능 8 비트 마이크로 컨트롤러의 설계

이용석, 강형주, 박인철
한국과학기술원 전자전산학과 전기 및 전자공학 전공
대전광역시 유성구 구성동 373-1
전화 : 042-869-4406 Fax : 042-869-4040

요 약

8051은 오랫동안 사용되어져 왔고 현재도 많은 활용 가치를 가지고 있는 대표적은 8 비트 마이크로 컨트롤러이다. 본 논문은 8051과 명령어 집합 수준에서 호환하면서 파이프라인 구조를 채택하여 87 MIPS (mega instructions per second), 즉 기존의 구조보다 40 배 이상의 성능을 가지는 마이크로 컨트롤러의 설계를 다룬다.

1. 서론

8 비트 마이크로 컨트롤러는 이미 널리 사용되어져 많은 응용 프로그램이 개발되어 있으나 복잡한 기능보다는 간단한 제어 분야에 주로 사용되어 왔다. 하지만 SOC 기반의 시스템 설계 방식이 부각되면서 이전보다 마이크로 컨트롤러는 더욱 복잡하고 다양한 작업을 처리하게 되었다. 이런 상황에서 명령어 집합 수준에서 완벽한 호환성을 유지하는 것은 기존의 소프트웨어 개발 환경을 그대로 사용할 수 있다는 것을 의미하므로 새로운 시스템의 개발 시간을 단축할 수 있다.

현재 생산되는 대부분의 8 비트 마이크로 컨트롤러 칩들은 설계 당시 고가였던 메모리를 최대한 효율적으로 사용하기 위하여 코어 자체는 매우 비효율적으로 설계되어 있다. 즉 한 명령어를 처리하는데 12 클럭 사이클이 소요되며 동작 주파수 또한 최대 24 MHz에 그친다.[1] 그러나 기존의 구조를 파이프라인 구조로 재설계하는 것만으로도 2-3 배의 성능 향상을 가져올 수 있고, 명령어 수행에 필요한 단계 수를 줄이고 발달된 공정 기술과 향상된 설계 기법을 적용하면 그 성능은 수십 배로 높일 수 있다. 본 논문에서는 기존의 8051 마이크로 컨트롤러와 명령어 집합 수준에서 호환성을 가지면서 기존보다 성능을 높인 컨트롤러를 설계하였다.

본 논문의 2 절은 8051의 구조적인 특징과 명령어 집합을 간단하게 소개하고 이어 3 절에서는 파이프라인을 도입한 구조를 설명한다. 4 절은 RTL 수준의 설계를 C 모델과의 비교를 통해 검증한 방법을 다루고 5 절은 합성 결과에 따른 성능 예측을 제시하고 6 절에서 결론을 맺는다.

2. 8051의 특징

8051은 크게 CPU와 peripheral, 그리고 메모리로 이루어져 있으며 이외에도 인터럽트 컨트롤러와 입출력 포트 등이 있다. 이 중에서 가장 특징적인 부분은 메모리 구조이다.

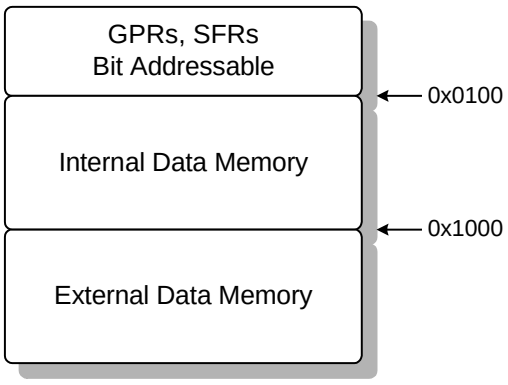

그림 1. 데이터 메모리의 주소 분할

메모리는 프로그램 메모리와 데이터 메모리가 분리되어 있는데, 그림 1과 같이 데이터 메모리의 하위 주소 부분은 범용 레지스터 (GPR)과 특수 기능 레지스터 (SFR)들은 위한 영역이다. 즉 일반적인 프로세서에서 레지스터로 구현되어 있는 부분들이 8051에서는 메모리에 합쳐져 있으며 이 중 일부는 비트 단위의 접근이

가능하다. 이 중 프로세서의 여러 가지 동작 방식을 정의하는 SFR들은 임의의 시간에 접근이 가능해야 하므로 실제로 메모리로 구현되어 있지는 않으며 다만 메모리 주소 공간에 대응되어 있다.[1]

8051의 명령어는 1 바이트, 2 바이트, 또는 3 바이트의 길이를 가지며 기능에 따라 산술 (arithmetic), 논리 (logical), 이동 (transfer), 부울 (Boolean), 분기 (branch) 명령어로 나눌 수 있다.

산술 명령어는 기본적인 사칙 연산과 BCD 포맷을 맞추기 위한 명령어 (DA)가 포함되어 있고, Boolean 명령어는 피연산자를 비트 단위로 취하여 논리 연산이나 분기 동작을 한다. 분기 명령어는 가까운 분기 (short jump)와 먼 분기 (long jump)가 분리되어 있다.

명령어가 취할 수 있는 피연산자 중 많이 사용되는 것에는 accumulator, 레지스터, 직접 어드레싱, 간접 어드레싱, 상수 (immediate)가 있다. Accumulator (A)는 SFR에 있는 레지스터이고, 레지스터는 GPR을 가리키며 4 개의 뱅크 중 하나에 있는 8 개의 레지스터 중 하나를 가리킨다. 직접 어드레싱은 피연산자의 메모리 주소를 명령어에서 직접 공급하는 경우이고 간접 어드레싱은 피연산자의 메모리 주소가 레지스터에 있는 경우로서 R0나 R1만이 사용된다. 상수는 피연산자의 값을 명령어에서 직접 공급하는 경우이다.

데이터 포인터 (DPTR), B 레지스터, 상대 어드레싱 (relative addressing) 및 external은 비교적 사용 빈도가 적은 피연산자 형태이다. DPTR과 B 레지스터는 SFR에 포함되어 있고, 상대 어드레싱은 피연산자의 메모리 주소를 DPTR+A, 또는 DPTR+PC가 가리키는 경우이다. External은 DPTR로 접근하며 반드시 외부 메모리를 사용한다.

이외에도 비트 단위로 접근할 수 있는 피연산자가 있다.[2]

3. 파이프라인 구조의 설계

3.1. 메모리 사이클

파이프라인 구조를 결정함에 있어 메모리 사이클은 중요한 역할을 한다.[3] 본 설계에서는 동기식 메모리를 사용하였다. 즉, CPU와 메모리가 동일한 클럭에 동기되어 동작하는 것이다. 이 경우 선택할 수 있는 방식은 두 가지가 있다.

CPU와 메모리가 같은 에지 (edge)에서 동작하는 경우:

CPU에서 주소 및 제어 신호를 낸 사이클에서는 데이터를 읽어올 수 없고 다음 사이클에서 읽어야 한다. 그러므로 데이터를 읽어오는데 두 사이클이 소요되고, 다시 데이터를 메모리에 쓰는데 두 사이클이 소요된다. 이는 그림 2에 나타나 있다.

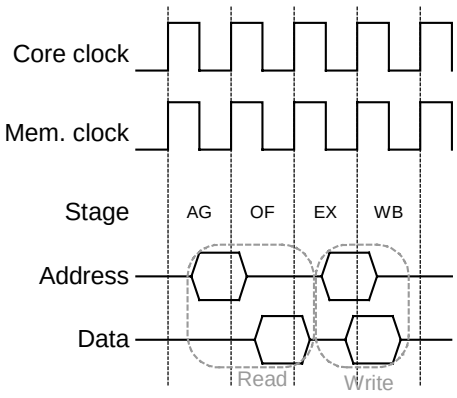

그림 2. CPU와 메모리가 클럭의 같은 edge를 사용할 경우

이 경우에 파이프라인은 6 단계가 된다: Instruction Fetch(IF) → Instruction Decode(ID) → Address Generation (AG) → Operand Fetch (OF) → Execution (EX) → Write Back (WB).

분기 명령어의 비율이 대략 33%라고 보았을 때 이 경우에 있어 한 명령이 수행하는데 소요되는 사이클의 수는 1 x 0.66 + 6 x 0.33 = 2.66 사이클이므로 대략 43MHz 정도로 16MIPS의 성능을 만족시킬 수 있다. 그러나 어떤 명령어가 A 레지스터를 갱신하고 다음 명령어가 동일 레지스터를 주소 계산에 사용한다면 포워딩 경로를 사용하더라도 stall을 피할 수 없다.

Structural hazard는 OF 단계 내부에서 두 개의 피연산자가 동일한 메모리 블록에 위치한 경우 stall이 발생하며, OF 단계와 WB 단계 사이에도 읽으려는 데이터와 쓰려는 데이터가 같은 주소는 아니면서 같은 메모리 블록에 위치할 경우 stall이 발생한다.

CPU와 메모리가 서로 다른 에지(edge)를 사용할 경우:

CPU에서 주소 및 제어신호를 메모리가 인식하기 전에 내어 주고, 메모리가 주소 및 제어 신호를 가져간 뒤 다음 CPU 동작 에지 전에 데이터를 내줌으로써 한 사이클에 메모리 접근을 마칠 수 있다. 이는 그림 3에 표현되어 있다. 이 경우는 메모리의 클럭을 CPU의 클럭보다 2배 높게 사용하는 것과 같다고 할 수 있다.

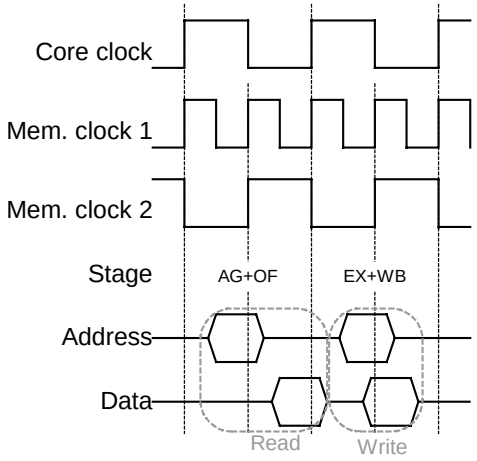

그림 3. CPU와 메모리가 클럭의 다른 edge를 사용할 경우

이 경우에는 앞서 같은 에지를 사용하는 경우에서 AG와 OF, EX와 WB를 하나의 단계로 합하여 다음과 같이 4 단계로 파이프라인을 구성할 수 있다: IF→ID→(AG+OF)→(EX+WB).

이렇게 하면 한 명령어를 수행하는데 1 x 0.66 + 4 x 0.33 = 1.99 사이클 정도가 소요되므로 32MHz의 동작 주파수에서 16 MIPS의 성능을 만족시킬 수 있다. 데이터 디펜던시에 의한 stall은 모두 포워딩으로 해결할 수 있으며 structural hazard는 AG+OF 단계에서 두 개의 피연산자가 동일한 메모리 블록에 위치한 경우 stall이 발생하며, AG+OF 단계와 EX+WB 단계 사이에도 읽는 데이터와 쓰는 데이터가 같은 주소는 아니면서 같은 메모리 블록에 위치하면 stall이 발생한다

위의 분석을 토대로 하면 같은 에지를 사용하게 되면 다른 에지인 경우에 비해 하나의 단계를 두 단계로 나눠 사용하므로 대략 두 배의 클럭이 가능하다고 볼 수 있다. 한 명령 당 사이클의 수가 2.7:2로써 대략 1.33배 이므로 같은 에지를 사용하면 2/1.33=1.5 배의 성능을 얻을 수 있다. 그러나 같은 에지를 사용하게 되면 데이터 디펜던시에 의한 stall이 발생하며, 포워딩 경로도 복잡하게 되어 제어가 어려워지고 인터럽트를 감안하면 동작의 완전성이 떨어지므로, 좀더 간단하고 안정성이 높은 컨트롤러를 만들기 위하여 메모리와 CPU가 서로 다른 에지를 사용하는 4단계의 파이프라인을 채택하였다.

3.2. 포워딩 경로 및 데이터 디펜던시에 의한 stall

4 단계 파이프라인 구조일 경우 포워딩을 사용하여 모든 데이터 디펜던시를 해결할 수 있다. 그러나 EX 단계에서 연산된 결과가 바로 AG 단계에 피연산자로 인가됨에 따라 critical path가 길어지게 되므로 이러한 critical path가 길어지는 포워딩 경로를 제거함으로써 성능 향상을 꾀했다.

n번째 명령어에서 계산된 결과가 다음 명령어의 피연산자로 사용될 경우에는 포워딩으로 해결한다. 그러나 n번째 명령어에서 계산된 결과가 다음 명령어의 주소 발생에서 사용된다면 stall을 사용하여 그 디펜던시를 해결한다. 주소 발생에 필요한 레지스터에는 R0, R1, PSW, SP, A, DPTR이 있다. 이들을 포워딩 경로로 연결할 경우 주소 계산에 너무 많은 시간이 소요되므로 모두 stall이 걸리도록 하였다.

3.3. Structural hazard

AG+OF 단계에서 읽어오려는 데이터와 EX+WB 단계에서 쓰려는 데이터가 동일한 주소는 아니면서 동일한 메모리 블록에 위치할 경우 stall을 발생시킨다. 그러나 AG+OF 단계에서 읽어오려는 두 개의

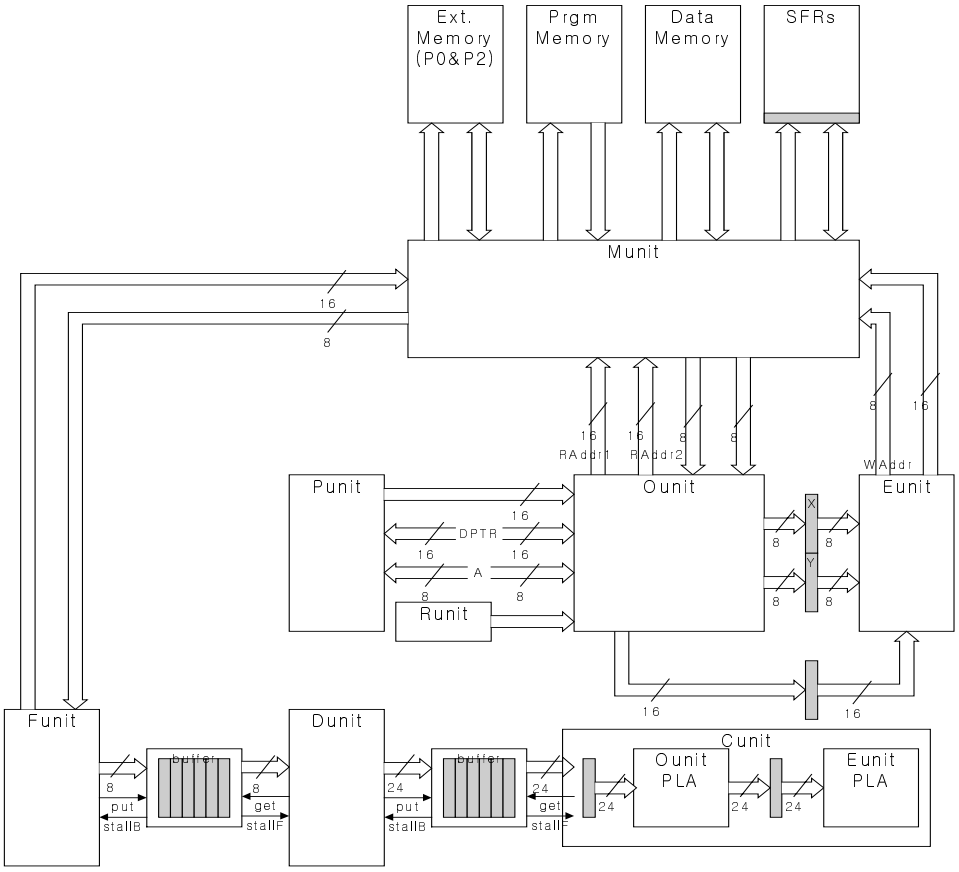

그림 4. 8 비트 마이크로 컨트롤러 CPU의 구조

피연산자가 같은 메모리 블록에 위치하는 경우에는 한 피연산자가 A레지스터이고 다른 하나의 피연산자가 SFR인 경우이다. 이 경우에는 A 레지스터를 메모리 통해서가 아니라 직접 가져오게 함으로써 stall없이 진행되도록 하였다.

3.4. 전체 구조

그림 4는 설계한 마이크로 컨트롤러의 블록도이다.

Funit은 IF 단계를 담당하며 메모리에서 프로그램 코드 바이트를 가져오는 제어 신호 및 주소를 생성한다. Funit은 자신의 출력단 버퍼를 보면서 그 버퍼가 비어 있으면 항상 메모리를 요청하여 버퍼를 채우려고 한다. 주소는 기본적으로 Punit에서 나오는 PC 값을 사용하나 AG+OF 단계에서 분기 명령어가 발견되면 메모리 접근을 중지하고 분기할 주소가 정해질 때까지 기다린다.

Dunit은 그 버퍼에서 한 바이트씩 가져오며 명령어의 첫 바이트를 보고 그 명령어가 몇 바이트로 이루어져 있는지를 판별한 뒤 버퍼에 있는 바이트들로부터 하나의 명령어를 구성하고 다음 버퍼에 저장한다. 여기서 저장된 명령어는 Cunit에 입력되고 Cunit은 명령어에 따라 PLA 및 기타 논리 회로를 통해 다른 부분들에 공급될 제어 신호를 생성한다.

Punit은 현재의 program count를 계산하고 프로그램 메모리에 접근할 주소를 생성한다. 파이프라인 구조이기 때문에 한 시점에 여러 개의 명령어가 데이터패쓰에 있는데 여기에서는 AG+OF 단계에 있는 명령어의 다음 주소를 PC로 했다. Punit은 일반적인 PC의 증가 이외에 분기에 필요한 모든 프로그램 메모리의 주소를 생성한다. 그러므로 Punit은 데이터패쓰 내의 필요한 모든 부분들과 직접 연결되어 있다.

Ounit은 명령어에 대한 피연산자의 주소를 계산하고 데이터 메모리에서 읽어온 데이터를 선별하여 Eunit에 넘긴다.

Eunit은 실제로 연산을 행하는 부분이며 따라서 EX 단계를 담당한다. 연산의 종류에 따라 산술 파트, 논리 파트, 회전(rotation)/치환 (swap) 파트, 그리고 비트 연산 파트로 나뉘어 있다. 8x8 곱셈기를 가지고 있기 때문에 곱셈은 한 사이클에 수행할 수 있으며 나눗셈은 shift-and-subtract 알고리즘을 사용하여 8 사이클에 수행한다. 연산 결과는 일반적으로 Munit에게 메모리 요청을 하여 메모리에 저장하지만 C (carry), AC (auxiliary carry), OV (overflow) 등은 EX 단계에서 직접 갱신한다. 이 값들을 다음 명령어가 바로 사용할 수 있도록 하기 위함이다.

Munit은 각 unit들로부터 온 메모리 사용 요청과 주소를 보고 적절한 메모리와 연결하거나 stall을 삽입한다. Runit은 레지스터의 일부를 가지고 있으며 Sunit은 대부분의 SFR 레지스터들을 관리하는 부분이다.

4. RTL 모델의 동작 검증

본 설계는 Verilog HDL을 사용하여 RTL 수준에서 기술되었다. 이러한 RTL 모델이 규모가 작고 동작 중 가질 수 있는 상태(state)가 적을 때에는 가능한 모든 테스트 벡터를 인가하여 동작이 올바른지를 확인할 수 있다. 그러나 우리가 설계한 마이크로 컨트롤러와 같이 가능한 상태가 비교적 많은 경우에는 이러한 검증은 효율적이지 못하고 오류를 찾아낼 확률로 크지 않다.

우리는 RTL 모델을 개발하기 전에 C 언어를 사용하여 더 상위 수준에서 8051의 명령어 집합 시뮬레이터 (ISS: instruction set simulator)를 개발하였으며 RTL 모델이 정확하게 동작하는지를 검증하기 위하여 ISS의 시뮬레이션 엔진을 활용하였다. 물론 이 작업은 상위 수준 시뮬레이션이 정확하다는 것을 전제하고 있다..

RTL 모델과 C 모델의 동작이 일치함을 확인하기 위해서 두 모델이 같은 시뮬레이션 벡터를 수행하도록 하고 그 결과를 비교하였다. 이를 위하여 Verilog PLI (program language interface)를 사용하였다. 즉, RTL 모델이 한 개의 명령어를 수행하고 Verilog PLI를 사용하여 C 루틴으로 구현되어 있는 시뮬레이션 함수를 호출한다. 이 때 호출되는 C 함수는 8051의 동작 모델이며 동일한 명령어를 다시 수행한다. Verilog PLI는 이 동작이 끝난 후에 두 모델의 수행 결과를 비교하여 결과가 동일하면 다음 명령어로 넘어가고 그렇지 않을 경우에는 잘못된 부분을 알려주는 오류 메시지를 표시하며 수행을 중지한다. (그림 5 참조) 수행 결과를 비교하는 대상은 PC와 모든 SFR, 그리고 모든 범용 레지스터를 망라했다.

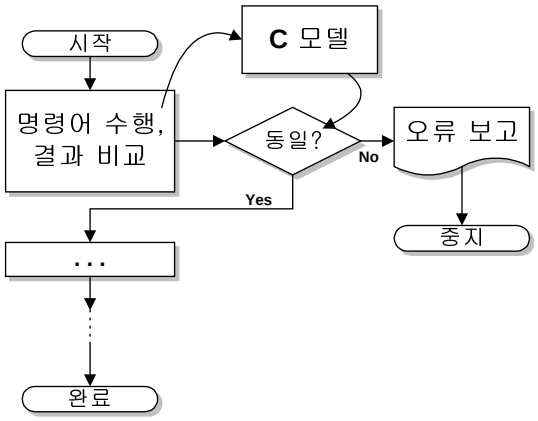

그림 5. C 모델과 RTL 모델의 비교 검증 과정

제대로 된 검증이 이루어지기 위해서는 가능한 모든 경우의 조합을 모두 시험해야 하기 때문에 2 KB의 메모리를 다 채우는 임의의 벡터를 여러 번 생성하여 검증에 사용했다. 기본적으로 8051의 명령어는 8 비트 단위로 이루어져 있지만 모든 조합이 올바른 명령어에 해당하는 것은 아니다. 그리고 임의의 벡터를 생성했을 경우 중간에 무한 반복이 생겨서 실제로 수행되는 코드는 극히 일부분에 지나지 않을 수 있기 때문에 분기 명령들에 대한 특별한 제어가 필요했다.

본 설계를 비교 검증 할 때에는 이렇게 하여 생성된 테스트 벡터 50 개를 사용하였으며 그러므로 약 10만 라인 이상의 임의 벡터가 테스트되었다. 여기에

추가로 일반적인 임의의 벡터로는 다루기가 곤란한 경우 (외부 포트 입출력 루틴 등)에 대하여 따로 테스트 프로그램을 제작하여 사용했다.

5. 성능 예측

완성된 RTL 코드를 합성할 때에는 Synopsys 社의 Design Compiler와 UMC 社의 0.25 um 표준 CMOS 공정을 사용하였다. 합성 결과 critical path delay는 5.78 ns였고 이는 회로의 최대 동작 주파수가 173 MHz임을 의미한다. Critical path delay의 대부분은 주소 생성 (address generation)과 메모리 셋업에서 발생했으며 hold time violation은 발견되지 않았다.

전체 게이트 수는 NAND 게이트 한 개의 크기로 환산한 결과 약 13,800 개였다. 각 부분별 게이트 수는 다음과 같다.

표 1. 합성된 회로의 부분별 게이트 수

부분	게이트 수
Funit + Dunit + Buffers	1,477
Cunit	2,080
Ounit	1,545
Eunit	1,410
Punit	715
Runit	712
Munit	1,440
Sunit	1,391
Peripherals	2,893
Timer	1,319
Serial port	1,075
Ports	499

한편, 동작 주파수를 낮추어서 회로의 크기를 줄였을 경우에는 100 MHz의 동작 주파수에서 11,513 게이트가 나왔다.

6. 결론

본 논문에서는 8 비트 마이크로 컨트롤러의 대명사격인 8051의 설계에 파이프라인 구조를 도입하여 성능 향상을 꾀하였다. 또한 RTL 모델을 C 모델과 비교 검증하여 RTL 모델의 정확도를 높였다.

합성 결과 최대 주파수는 0.25 um 표준 CMOS 공정에서 173 MHz 였으며, CPI (cycles per instruction)가 1.99이므로 87 MIPS의 성능을 보였다. 이는 현재 시판되고 있는 24 MHz에 동작하는 8051 계열 칩들에 비하여 40 배 이상의 성능 향상을 의미한다.

본 설계의 결과물은 게이트 수가 비교적 적기 때문에 독립적인 칩으로서 뿐만 아니라 다른 SOC를 제어하는 IP로서도 그 활용 가치가 높다.

참고 문헌

[1] Flash Microcontroller, Architectural Overview, Atmel, 1997.

[2] Microcontroller Instruction Set, Atmel, 1997.

[3] D. A. Patterson and J. L. Henessy, *Computer Architecture: A Quantitative Approach*, 2nd Ed., Morgan Kauffmann Publishers, Inc., San Francisco, CA, 1995.