한국음향학회지 제20권 제7호 pp. 63~68 (2001)

ARM9 Thumb® 프로세서 코어를 이용한 G.729A의 실시간 구현

Real-time Implementation of the G.729 Annex A Using ARM9 Thumb® Processor Core

성 호 상*, 이 동 원**
(Ho sang Sung*, Dong won Lee**)

*한국 전자통신연구원, **삼성전자 통신연구소
(접수일자: 2001년 6월 19일; 채택일자: 2001년 8월 6일)

본 논문에서는 국제 통신 표준화기구인 ITU-T의 SG15에서 채택된 G.729 Annex A (이하 G.729A) 음성 부호화기를 ARM9 Thumb® 프로세서 코어에 적용 가능하도록 전체 모듈을 다양한 최적화 방법을 이용하여 어셈블리어로 실시간 구현하였다. G.729A는 8 kbit/s의 전송률을 갖는 ITU-T 표준 음성 부호화기이며, 입력신호는 8 kHz로 샘플링되며 샘플당 16 비트로 양자화된 PCM신호이다. G.729A는 앞서 표준화된 G.729와 비트단위로 상호호환 가능하며 계산량을 대폭 감소시킨 버전이다. 구현된 G.729A 음성 부호화기는 부호화기와 복호화기 부분이 각각 약 35 MIPS 및 8 MIPS의 복잡도를 나타내며, 사용된 메모리양은 프로그램 ROM 36.5 kBytes, RAM 6.3 kBytes이다. 구현된 G.729A 음성 부호화기는 ITU-T에서 제공하는 9개의 테스트 벡터를 모두 통과하였다.
핵심용어: 음성 부호화기, G.729A, ARM9, 실시간 구현
투고분야: 음성처리 분야 (2.2)

This paper describes the details of ITU-T SG15 G.729A speech coder implementation using ARM9 Thumb® processor core and various techniques used in the optimization process. ITU-T G.729 speech coder is the standard of the toll quality 8 kbit/s speech coding. The input to the speech encoder is assumed to be a 16 bits PCM signal at a sampling rate of 8000 samples per second. G.729A is reduced complexity version of the G.729 coder. This version is bit stream interoperable with the full version. The implemented coder requires 34.8 MIPS for the encoder and 8.1 MIPS for the decoder, 36.5 kBytes of program ROM and 6.3 kBytes of data RAM, respectively. The implemented coder is tested against the set of 9 test vectors provided by ITU-T for bit exact implementation.
Keywords: *Speech coder, G.729A, ARM9, Real-time implementation*
ASK subject classification: *Speech signal processing (2.2)*

I. 서 론

유선망 수준의 톨 음질을 제공하는 저속의 음성 부호화기는 높은 주파수 효율로 인해 이동통신 및 인터넷 텔레포니 등에서 새로운 서비스를 가능하게 하였다. 특히 인터넷 망에서의 VoIP (voice over internet protocol)는 저렴한 전화요금으로 인해 기하급수적으로 확산이 되고 있는 상태이나 그리 좋은 평가는 받고 있지 못하다. 하지만 이러한 확산에 걸림돌로 작용하는 인터넷 망에서의 낮은 음성 품질과 긴 지연으로 인해 생기는 서비스 저하를 다양한 방법으로 극복해 보려는 시도가 계속되고 있다. 이런 시도의 일환으로 VoIP 시스템에서 G.729와 같은 짧은 지연과 고음질을 가지는 음성 부호화기를 사용

책임저자: 성호상 (hssung@etri.re.kr)
305-350 대전시 유성구 가정동 161번지 한국전자통신연구원
네트워크기술연구소 휴먼인터페이스연구부 멀티미디어통신팀
(전화: 042-860-1246; 팩스: 042-861-1342)

하면 음질 및 지연 측면에서 많은 이득을 얻을 수 있다. 1995년 11월에 표준화되어 G.729로 알려진 8 kbit/s CS-ACELP (conjugate structure algebraic CELP)[1]는 저속이면서 낮은 지연을 갖는 고음질의 음성 부호화기이다. G.729 음성 부호화기는 ACELP (algebraic code-excited linear prediction)와 컬레 구조 (conjugate structure) VQ (vector quantization)를 기본으로 한 부호화기로서, 피치 및 코드북의 효율적인 탐색구조를 첨가하여 계산량을 줄이면서 성능을 극대화한 결과 잡음/비잡음 환경에서 32 kbit/s ADPCM (G.726)과 비교할 때 그 이상의 음질을 나타내는 것으로 알려져 있다[2]. 또한 10 ms의 프레임을 사용하여 알고리즘 지연이 15 ms 이내로 좋은 성능을 나타내고 있다. 이러한 G.729와 상호호환 가능하면서 거의 동일한 음질에 50 %정도 복잡도가 낮은 G.729A[3]는 1996년에 표준화가 되었으며, 현재 V.70 (DSVD) 및 Frame Relay Forum에서는 기본 음성 부호화기로 사용되며, IMTC의 VoIP Forum에서는 선택적인 음성 부호화기로 규정되어 있으므로 VoIP 프로토콜인 H.323, SIP (session initiation protocol)에서도 널리 사용되고 있다. ARM9 Thumb® 프로세서 코어[4]는 ARM사에서 개발한 16/32 비트 범용 RISC 프로세서 코어이며 낮은 전력소모와 높은 성능을 가진다. 이 ARM9 Thumb® 프로세서 코어는 0.25 ㎛의 CMOS 공정으로 제작된 경우 130 MHz에 148 MIPS의 성능을 가지며, 이를 이용한 코어로는 캐쉬를 보유하고 있는 ARM920T와 ARM940T[4]코어가 있는데 두 코어는 캐쉬 크기의 차이와 MMU (memory management unit) 보유 유무 이외에는 모든 성능이 동일하며 동일한 명령어를 사용한다.

본 논문의 구성은 2장에서 G.729A의 기본구조와 특성에 대하여 살펴보고, 3장에서는 실시간 구현에 관해 논하며, 마지막으로 4장에서 결론을 맺도록 하겠다.

II. G.729A 음성 부호화기

2.1. 부호화기

G.729A는 기본적으로 분석/합성 방식의 CELP구조[5]를 이용하며 기본 개념도는 그림 1과 같다. 입력 음성신호는 8 kHz로 샘플링된 PCM 신호이며, 10 ms의 프레임과 5 ms의 부프레임을 갖는다. 선형예측 (LP: linear prediction) 계수 추출을 위해 총 240샘플 중 40샘플을 이후의 프레임에서 취하는 비대칭 창을 사용하므로써 총 알고리즘 지연

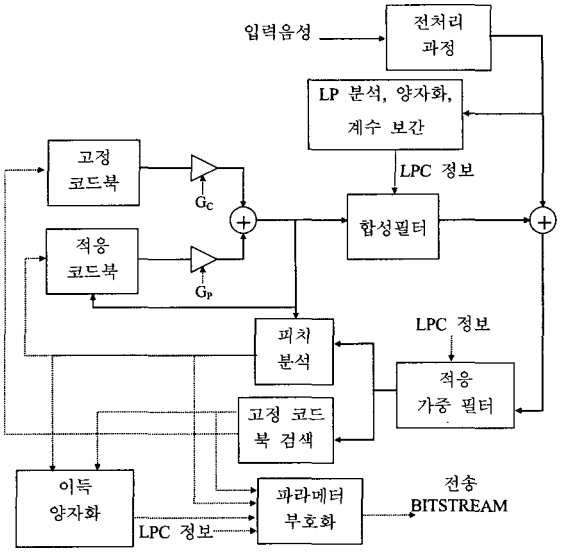


그림 1. ITU-T G.729A 부호화기의 기본구조
Fig. 1. Principle of the ITU-T G.729A encoder.

시간이 15 ms가 된다.

전처리 과정으로 차단주파수 140 Hz의 고대역 통과 필터를 사용하여 입력신호의 DC성분을 제거하고 오버플로우를 방지하기 위해 전체적인 스케일을 반으로 줄인다. 10차의 선형 예측 필터를 통해 얻어진 선형예측 계수는 안정도 특성을 가지는 LSP (line spectrum pair) 계수[6]로 변환되며, 이 값과 4차의 MA (moving average) 예측기를 통해 예측된 값과의 차이는 각각 128개와 32개의 코드워드를 갖는 2단계의 VQ 및 분할 VQ[7]를 거쳐서 총 18비트의 코드워드로 부호화된다. 이 때 서로 다른 계수값을 가지는 2종류의 MA 예측기가 사용되며 그 중 각각의 예측기에 의해 구해진 예측값과 LSP 계수값 사이의 WMSE (weighted mean squared error)가 최소화되는 것을 선택한다. 이렇게 양자화된 LSP 계수는 두번째 부프레임을 위한 것이며 첫번째 부프레임의 LSP계수는 이전 프레임에서 얻어진 LSP계수와의 보간을 통해 구해진다. 피치정보를 위한 적응 코드북과 여기신호를 얻기 위한 고정 코드북 파라메터들은 각 부프레임마다 전송된다. 최적 피치 지연값을 찾기 위한 절차는 다음과 같다. 먼저 개루프를 통한 피치검색은 적응 가중필터를 통과한 음성 신호에 대하여 매 프레임당 한번씩 수행된다. 각 부프레임에서는 폐루프 검색을 수행하여 미리 결정된 개루프 피치 지연값을 기준으로 1/3의 정밀도를 갖는 최적 피치 지연값을 얻는다. 최적 피치 지연값을 찾아내면 음성의 여기신호 추출을 위한 고정 코드북 검색이 이루어진다. 고정 코드북 검색을 위한 목적 신호는 피치검색에서 사용된 목적 신호

표 1. G.729A 고정 코드북의 구조
Table 1. Structure of fixed codebook.

펄스	부호	위치
I0	*S0* : ± 1	*m0* : 0, 5, 10, 15, 20, 25, 30, 35
I1	*S1* : ± 1	*m1* : 1, 6, 11, 16, 21, 26, 31, 36
I2	*S2* : ± 1	*m2* : 2, 7, 12, 17, 22, 27, 32, 37
I3	*S3* : ± 1	*m3* : 3, 8, 13, 18, 23, 28, 33, 38 4, 9, 14, 19, 24, 29, 34, 39

에서 적응 코드북에 의한 기여분을 제거함으로써 얻어진다. 표 1은 최적 여기신호 검출을 위해 G.729A에서 채택한 고정 코드북의 구조를 나타낸 것이다. 기존의 음성 부호화 방식과는 달리 음성의 여기신호벡터의 요소값으로서 각 부프레임당 4개의 펄스만이 지정된 위치에서 +1 또는 −1의 값을 갖게 되어 있다. 이러한 대수적 (algebraic) 코드북 구조에서 4번의 검색루프를 통해 연쇄적으로 최적 펄스의 위치를 찾는 G.729에서의 방식과 달리 depth-first 방식을 사용함으로써 탐색 횟수를 약 1/4로 줄였다.

적응 코드북과 고정 코드북의 이득값 양자화에는 검색의 효율을 위하여 컬레 구조를 갖는 2개의 코드북을 사용한다. 각각의 코드북은 8개와 16개의 요소값을 가지며 구해진 이득값들에 의해 미리 선택된 4개와 8개의 요소값들에 대해서만 검색이 이루어진다. 결국 32 (=4×8) 가지의 조합된 경우들에 대해서만 검색을 수행하므로써 적응 및 고정 코드북에 대한 최적 이득값들을 찾게 되며, 이러한 방법은 각각 8개와 16개의 요소값들을 가지는 코드북의 검색과 비교할 때 상당한 계산량의 감소를 얻게 한다. 부호화의 마지막 단계에서는 다음 프레임의 목적 신호를 구하기 위해 합성 및 가중필터의 메모리 갱신을 수행한다. 표 2는 각 파라메터에 할당된 비트를 나타낸다.

표 2. 8 kbit/s G.729A의 비트할당
Table 2. Bit allocation of the 8 kbit/s G.729A.

파라메터	코스워드		Bits/frame
	부프레임 1	부프레임 2	
LSP	L0[1], L1[7], L2[5], L3[5]		18
적응코드북지연	P1[8]	P2[5]	13
피치지연 parity	P0[1]		1
고정코드북 인덱스	C1[13]	C2[13]	26
고정코드북 부호	S1[4]	S2[4]	8
코드북 이득 (stage 1)	GA1[3]	GA2[3]	6
코드북 이득 (stage 2)	GB1[4]	GB2[4]	8
Total			80

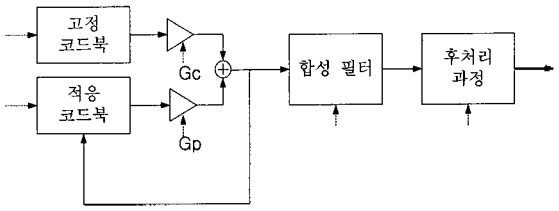


그림 2. G.729A 복호화기의 기본구조
Fig. 2. Principle of the G.729A decoder.

2.2. 복호화기

여기신호가 복원되기 전에 적응 코드북의 인덱스로부터 패리티 비트를 구하여 수신된 패리티 비트와 일치하는지를 확인하여 전송 중에 에러가 발생하였는가를 검사한다. 에러가 발생되었다고 판단되면 첫 번째 부프레임의 피치 지연값으로 과거의 두 번째 부프레임에서 결정된 값의 정수값을 사용한다.

추출된 각각의 파라메터들로부터 각 부프레임에 적용될 여기신호와 이득값, 피치 지연값과 이득값 그리고 LSP 계수값들이 정해지며 이를 사용하여 합성 음성을 복원한다. 복원된 합성 음성은 보다 나은 음질을 위해 후처리 과정을 거치게 된다. 후처리 과정에서는 다음의 기능들이 수행된다.

(1) 적응 후처리 필터링

적응 후처리필터의 계수는 매 부프레임마다 갱신되며 장구간 후처리필터링, 단구간 후처리필터링, 기울기 (tilt) 보상 필터를 단계적으로 거치게 된다.

(2) 고대역 통과 필터링과 신호의 신호비율 확대과정

고대역 통과 필터를 통해 저주파 성분의 잡음을 제거하고, 부호화기에서 오버플로우방지를 위해 신호비율 출소과정 (down-scaling)을 수행하였으므로 복호화의 끝단에서는 신호비율 확대과정 (up-scaling)을 수행한다.

G.729A는 G.729의 계산량을 감소시킨 버전으로, 부호화된 비트 스트림은 G.729와 상호호환 가능하다. G.729A에서 계산량이 감소된 부분은 LPC 해석, 피치 탐색, 고정 코드북 탐색 및 후처리 필터로서, G.729와 거의 같은 수준의 음질을 보이면서 약 50%의 복잡도를 감소시켰다.

계산량 감소를 위해 사용한 방식들을 구체적으로 살펴보면,부호변경구간을 찾는 체비쉐프방식으로 극점 (pole)을 구하는 LPC해석에서 구간폭을 1/60 에서 1/50로 늘임으로서 계산량을 줄였고, 피치 탐색부분에서는 개루프 탐색구간의 생략과 폐루프 탐색시, 적응 코드북으로부터 합성되는 신호 중에서 1/3정밀도 탐색에 필요한 부분만

합성하므로써 계산량을 줄였다. 또한 고정 코드북 탐색에서는 부프레임 당 약 1440번을 탐색하는 전체 탐색방식을 사용한 G.729와 달리, depth-first 방식을 사용하여 탐색횟수를 320번으로 줄였다. 마지막으로 후처리 필터에서는 1/8정밀도로 피치 지연을 찾는 부분을 제거하여 부가적인 계산량을 크게 줄였다.

III. 실시간 구현

3.1. ARM9 Thumb® 프로세서 코어

G.729A의 실시간 구현에 사용된 ARM9 Thumb® 프로세서 코어는 전형적인 RISC 프로세서이며 고정 소숫점 코어이다[8]. 부동 소수점 연산을 위한 VFP (Vector Floating Point) 명령이 제공되지만 고정 소수점 명령보다는 많은 연산량을 필요로 한다. 본 논문에서는 ARM9 Thumb® 프로세서 코어를 탑재한 ARM940T 코어를 이용하여 각종 실험을 하였다. ARM940T 코어는 MMU를 보유하고 있지 않으며, 4 kBytes의 데이터 캐쉬와 인스트럭션 캐쉬를 따로 가지는 하바드 구조를 채용하므로 사이클 당 1.1개의 명령어를 처리할 수 있다. 명령어로는 메모리를 위한 로드/스토어 (load/store)명령과 레지스터를 위한 다양한 연산명령이 있다. 총 37개의 레지스터를 제공하고 있으며 이는 30개의 일반적인 레지스터와 pc, CPSR, 5개의 SPSR로 이루어져 있다. 여기서 코딩시에 사용할 수 있는 레지스터는 15개 (r0 ~ r14)이다. 여기서 r13은 스택 포인터 (stack pointer)로 사용되므로 실제로 코딩에 사용가능한 레지스터는 14개이다. ARM9 Thumb® 프로세서 코어에서 제공하는 어셈블리어의 특징은 모든 연산이 조건부로 수행된다는 점, 다양한 어드레싱 모드를 위한 인라인 바렐 쉐프터를 제공한다는 것과 다중 로드/스토어 (load/store multiple) 명령을 이용하여 데이터 전송을 최대화할 수 있는 장점이 있다. 그 이외에도 32비트의 곱셈기를 제공하므로 충분한 정밀도의 연산도 가능하다.

3.2. G.729A 실시간 구현

ARM9 Thumb® 프로세서 코어를 이용한 G.729A의 구현과정은 다음과 같다. 먼저 C 소스코드를 크로스 컴파일한다. 이 과정을 거치면 어셈블리로 컴파일된 코드가 생기며 이 코드에는 각종 LABEL이 미리 정해져 있으므로 코딩을 더 빨리 할 수 있는 장점이 있으며 간단한 연산은 핸드코딩을 할 필요없이 바로 이용가능하다. 하지만 과잉 코드 (redundancy)가 많이 있으므로 대부분의 코드를 핸드코딩해 주면 많은 성능 향상을 이룰 수 있다. 주로 최적화가 필요한 부분은 루프문, 베이직 연산, 어드레싱 모드 등이다. 그리고 ARM940T 코어는 DSP (digital signal processor)와 달리, 성능에 많은 영향을 주는 특수한 명령어를 제공하지 않고, 포화 (saturation) 기능도 없으며, MAC (multiplication and accumulation) 연산 및 메모리로부터 데이터 로딩시 많은 연산량이 요구되므로 비트단위로 정확한 결과를 내기 위해서는 DSP보다 많은 클럭을 사용해야 한다. 이러한 제한하에서 최대한의 최적화를 위해 다양한 방식의 최적화과정을 시도하였다.

첫번째 최적화방식은 루프 전개 (unrolling) 방법이다. 특히 음성 부호화기는 디지털 신호를 처리하는 과정이므로 루프문을 상당히 많이 사용한다. 그러므로 이 방식을 사용하면 최소 3 사이클을 요구하는 분기 (branch)와 추가적인 카운터와 비교문의 사용을 줄일 수 있으므로 계산량을 많이 줄일 수 있다. 두 번째는 어드레싱 모드를 이용하는 방식이다. ARM940T 코어의 장점이라고 할 수 있는 다양한 어드레싱 모드는 연산의 시작부분에 대한 포인터만 지정해주면 데이터를 쉽게 로드/스토어를 할 수 있으며 이 과정에서 계산량을 줄일 수 있다. 세 번째는 베이직 연산의 처리이다. 고정 소수점 G.729A의 C 소스코드는 베이직 연산을 DSP의 상황을 고려해서 구현되었으므로 오버플로우나 언더플로우, 그리고 포화를 항상 고려하므로 추가적인 계산량이 많이 포함되어 있다. 하지만 이러한 베이직 연산을 DSP에서는 쉽게 하나의 명령어로 대치시킬 수 있지만 ARM940T 코어에서는 오버플로우나 언더플로우를 고려하지 않으므로 비트단위로 정확한 구현을 하려면 코딩시에 많은 주의가 필요하다. 그래서 성능저하없이 basic 연산을 잘 처리하면 많은 계산량을 줄일 수 있다. 네 번째는 조건부 (conditional field)의 적절한 사용이다. 이는 코드의 조건문에 대한 알고리즘을 최적화할 때 사용되는 기법으로 계산량이 많은 분기 명령을 대치함으로써 추가적으로 계산량을 줄일 수 있다.

그림 3은 어셈블리로 구현된 G.729A의 루프문이다. C code에서 회색 부분은 어셈블리 코드에서도 같은 색을 가진 부분을 의미한다. 어셈블리로 작성된 부분을 처음부터 살펴보면 다음과 같다. 라인 1에서 v1(=i)에 초기값을 2로 준 이유는 카운팅 기능을 하면서 y[j+i] 배열의 옵셋 기능을 하기 때문이다. y[j+i] 배열의 데이터 크기는 16 비트인데 ARM 어셈블리는 byte addressing을 하므로 2 byte씩 증가해야 한다. 그러므로 초기값을 2로 정하고 라인 18에서 2씩 증가를 한다. 라인 3에서 정의된 |L1,292|는 크로

C code in G.729A

```
for (i = 1; i <= m; i++)
{
   sum = 0;
   for(j=0; j<240-i; j++)
        sum = L_mac(sum, y[j], y[j+i]);
   sum = L_shl(sum, norm);
   L_Extract(sum, &r_h[i], &r_l[i]);
}
```

Assembly code

```
    MOV   v1,#2              ; v1=i=2                    ①
    MOV   v7,#10             ; v7=m=10                   ②
|L1.292|                     ; 1st "for" loop            ③
    MOV   v2,#0              ; v2=sum=0                  ④
    MOV   a4,#0              ; a4=j=0                    ⑤
    ADD   v6,v4,v1           ; v6=y[j+i], v4=y[j]        ⑥
    RSB   v8,v1,#0xf0*2      ; v8=L_WINDOW-i             ⑦
|loop|                       ; 2nd "for" loop            ⑧
    LDRSH  a1,[v4,a4]        ; a1=y[j] data              ⑨
    LDRSH  a2,[v6,a4]        ; a2=y[j+i] data            ⑩
    MLA    v2,a1,a2,v2       ; v2=sum                    ⑪
    ADD    a4,a4,#2          ; j++                       ⑫
    CMP    a4,v8             ; j<240-i ?                 ⑬
    BLT    |loop|            ; End of 2nd loop           ⑭
|L1.344|                                                 ⑮
    MOV   v2,v2,LSL v3       ; L_shl(),v3=norm+1         ⑯
    STR   v2,[v5,v1,LSL #1]  ; L_Extract( )              ⑰
    ADD   v1,v1,#2           ; i++                       ⑱
    CMP   v1,v7, LSL #1      ; i<=m ?                    ⑲
    BLE   |L1.292|           ; End of 1st loop           ⑳
```

그림 3. ARM 어셈블리로 구현된 G.729A의 루프문
Fig. 3. Implementation of the G.729A loop example using ARM assembly language.

스 컴파일을 할 때 정해지는 LABEL명이다.

그림 3에서 색칠된 부분을 잘 구현해야 많은 연산량을 줄일 수 있다. 현재 구현된 예는 가장 기본적으로 구현한 예이며 여기서 앞에서 언급한 각종 최적화 방식을 이용하면 많은 연산량을 줄일 수 있다. 첫째로 루프 전개 방식을 이용한다. 그림 3에서 색칠된 부분은 총 230~239회를 반복하므로 많은 계산량을 소모한다. 그러므로 어떤 경우라도 230회는 실행하므로 처음 시작부터 230회의 MAC연산은 그림 3의 루프 문에서 분리시켜 독립적으로 실행한다. 독립적으로 실행할 때 루프는 프로그램 메모리를 낭비하지 않는 한도 내에서 전개를 한다. 그리고 전개시 계산량을 줄이기 위해서 LDRSH 명령의 a4 offset 대신에, post-indexed 또는 pre-indexed에 대한 어드레싱 모드를 사용한다. 어드레싱 모드를 사용하면 a4 카운터를 이용할 필요가 없어지므로 계산량을 줄일 수 있다. 그리고 다음으로 주의해야 할 것은 MAC연산이다. C 소스 코드에서 MAC연산은 오버플로우를 체크하지만 색칠된 루프에서는 정규화 인수 (normalize factor)를 이용하는 부분이므로 오버플로우가 발생하지 않는다. 그러므로 이런 경우에는 오버플로우 체크를 할 필요가 없다. 그리고 L_mac() 함수의 출력은 MLA 명령의 출력의 2배이므로 v3값이 norm+1이 되어야 한다. 이런 값들은 미리 설정해 두어야 한다. 이와 같이 간단한 루프에서도 위에서 언급한 다양한 최적화 기능을 사용할 수 있다.

처음 G.729A를 크로스 컴파일했을 때 부호화기와 복호화기의 전체 계산량은 약 260 MIPS였으나, 위에서 언급한 다양한 최적화 방법을 이용하여 구현된 결과는 평균 42.9 MIPS로 나타났으며, 크로스 컴파일된 계산량에 비해 약 1/6로 최적화되었음을 알 수 있다.

3.3. 실시간 구현 결과

본 논문에서는 프로그램의 전 과정이 어셈블리어로 이루어졌으며 프로그램된 각각의 모듈은 ITU-T에서 제공된 테스트 벡터를 이용해 검증하였다. ITU-T에서 제공하는 테스트 벡터는 총 9개로서, 부호화기와 복호화기에 공통으로 사용하는 6개의 벡터와 복호화기에만 사용하는 3개의 벡터가 있다. 구현된 G.729A 어셈블리 프로그램은 테스트 벡터와 비트단위로 정확하게 일치하였다. 그리고 구현된 G.729A의 성능을 비교하기 위해 TMS320C54x칩과 OakDSPCore®에서 구현된 결과[9]를 비교하였다. 먼저 사용 메모리를 표 3에 나타내었다. 프로그램에 사용된 ROM의 크기에서 약 2배의 차이를 보이는 것은 구현에 사용된 ARM940T 코어의 명령어는 모두 32비트 명령이기 때문이다. 표 4는 구현한 결과의 평균 복잡도 비교를 나타내고 있다. 본 논문에서는 부호화기와 복호화기 각각 34.8, 8.1 MIPS의 계산량을 가지는데, 3.2절에서 언급했듯이 DSP와 ARM940T 코어의 구조적인 차이로 인해 대부분을 어셈블리로 구현했음에도 불구하고 약 3배의

표 3. G.729 A의 사용 메모리 (kBytes)
Table 3. Memory requirements (kBytes) for G.729A.

	ROM	RAM	
		Static Memory	Stack Memory
ARM940T core	36.5	3.4	2.9
OakDSPCore®	16.8	6.2	
TMS320C54x	23.4	8.0	

표 4. 상용 DSP와의 복잡도 비교 (MIPS)
Table 4. Complexity comparison against commercial DSP.

	Encoder	Decoder	Full duplex
ARM940T core	34.8	8.1	42.9
OakDSPCore®	11.4	2.5	13.9
TMS320C54x	10.2	2.3	12.5

계산량 차이가 나는 것을 알 수 있다.

최종적으로 시뮬레이션상에서 구현된 프로그램은 실시간 동작을 위해 하드웨어와 소프트웨어를 연결하는 작업이 필요하다. ARM940T 코어를 하드웨어에 올려서 실행하기 위해 ARM 코어의 emulator tool kit인 Multi-ICE를 통해서 PC에서 이미지를 테스트 보드에 다운로드한 후 결과를 확인하였다. Multi-ICE를 이용하면 시뮬레이터와 달리 실제 시스템의 메모리를 사용하므로 제한된 크기의 메모리에 사용 메모리의 속도까지 포함된 결과를 확인할 수 있다. 이를 위해서는 ARM사에서 제공하는 ARM Integrator/AP 보드와 ARM Integrator/CM940T 보드를 이용하여 시스템을 구성하였다. ARM Integrator/AP 보드는 마더 보드로서 다양한 코어보드를 장착할 수 있다. 이것은 독립형 (standalone)으로도 사용가능하며 Multi-ICE로도 사용 가능하다. 여기에 코어 보드인 ARM Integrator/CM940T 보드를 장착하면 ARM940T 코어를 시험할 수 있게 된다. 이렇게 구성한 시스템에 microHAL API를 이용하여 G.729A를 포팅한다. microHAL API는 OS나 어플리케이션을 포팅할 수 있게 해주는 저급함수세트 (low-level function set)이다[10]. 포팅된 G.729A를 동작시켰을 때 최소 50 MHz의 코어 클럭과 20 MHz의 시스템 클럭에서도 실시간으로 동작하는 것이 확인되었다. 이는 130 MHz의 코어인 경우 CPU의 점유율이 약 40% 이하가 됨을 의미한다.

IV. 결 론

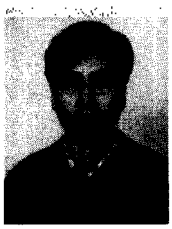
본 논문에서는 16/32비트 고정 소숫점 프로세서 코어인 ARM940T 코어를 이용하여 G.729A 음성 부호화기를 실시간 구현하였다. 구현된 G.729A 코덱은 부호화기와 복호화기 부분이 각각 약 34.8 MIPS 및 8.1 MIPS의 복잡도를 나타내며, 사용된 메모리양은 프로그램 ROM 36.5 kBytes, RAM 6.3 kBytes이다. 그리고 상용 DSP와 계산량을 비교해 볼 때 ARM940T 코어는 약 3배의 계산량을 가지는 것을 알 수 있다. 이는 구조적인 문제에 기인하는 것이므로 ARM940T 코어를 채택할 때 중요한 판단 기준이 된다. 본 논문에서 구현된 G.729A는 ARM940T 코어의 저전력 소비, ASIC구현의 용이함 등의 장점을 이용하여 VoIP 또는 음성저장 장치 등 다양한 음성 압축/재생 응용장치의 핵심모듈을 저가의 SOC (system on a chip)로 구현할 수 있게 해준다.

참 고 문 헌

1. ITU-T Recommendation G.729, "Coding of speech at 8 kb/s using conjugate-structure algebraic code-excited linear prediction (CS-ACELP)," June 1995.
2. P. Usai et al., "Subjective performance of the proposed ITU-T 8 kb/s speech coding standard," *IEEE Speech Coding Workshop*, Annapolis, Sept. 1995.
3. ITU-T Recommendation G.729 Annex A, "Reduced complexity 8 kbit/s CS-ACELP speech codec," Nov. 1996.
4. http://www.arm.com
5. B. S. Atal and M. R. Schroeder, "Stochastic coding of speech signals at very low bit rates," *Conf. Rec. Int. Conf. Commun.*, pp. 1610-1613, May 1984.
6. N. Sugamura and N. Farvardin, "Quantizer design in LSP speech analysis-synthesis," *IEEE J. Select. Areas in Commun.*, vol. **6**, pp. 432-440, Feb. 1988.
7. K. K. Paliwal and B. S. Atal, "Efficient Vector Quantization of LPC parameters at 24 bits/frame," *IEEE Trans. speech and audio proc.* vol. **1**, pp. 3-14, Jan. 1993.
8. ARM Ltd., ARM Developer Suite v.1.0.1: Developer Guide, March 2000.
9. 이동원, 김승훈, 강상원, 성유나, 심민규, "Real-time implementation of the G.729 and G.729A using OakDSPCore," *KSPC00*, vol. 13, No. 1, pp. 807-810, Oct. 1999.
10. ARM Ltd., ARM Firmware Suite v.1.1: Reference Guide, Feb. 2000.

저자 약력

● **성 호 상 (Ho sang Sung)**



1996년 2월: 한양대학교 제어계측공학과 졸업 (학사)
1999년 2월: 한양대학교 제어계측공학과 졸업 (석사)
1999년 6월~현재: 한국전자통신연구원 멀티미디어 통신팀 연구원
※ 주관심 분야: 음성 신호처리

● **이 동 원 (Dong won Lee)**

1997년 2월: 한양대학교 제어계측공학과 졸업 (학사)
1999년 8월: 한양대학교 제어계측공학과 대학원 졸업 (석사)
1999년 9월: 한양대학교 전자전기제어계측공학부 대학원 입학 (박사)
2001년 2월~현재: 삼성전자 통신연구소 IMT-2000 단말연구팀 연구원
※ 주관심 분야: 음성 신호처리, 모뎀