TMS320VC5402 DSP를 이용한 G.723.1A 음성부호화기의 실시간 구현

Real-time Implementation of G.723.1A Speech Coder Using a TMS320VC5402 DSP

이 송 찬* · 정 익 주**
Song-chan Lee · Ikjoo Chung

ABSTRACT

This paper describes the issues associated with the real-time implementation of G.723.1A dual-rate speech coder on a TMS320VC5402 DSP. Firstly, the main features of the G.723.1A speech coder and the procedure involved in the implementation using assembly and C languages are discussed. Various real-time implementation issues such as memory/MIPS tradeoffs are also presented. For fixed-point implementation, we converted the ITU-T fixed-point ANSI C code into TMS320VC5402 code in the bit-exact way through verification using the test vectors. Finally, as the result of implementation, we present the MIPS and memory requirement for the real-time operation.

Keywords: G.723.1A, Speech Coder, TMS320VC5402

1. 서 론

디지털 음성 통신은 아날로그 음성 통신에 비하여 큰 대역폭을 필요로 한다. 그러나 이러한 단점은 음성 압축 기술을 이용하여 해결할 수 있게 되었다. 디지털 음성 통신은 신호처리의 용이함과 유연성, 보안 및 다른 시스템과의 연동이 용이하다는 점과 같은 장점 등으로 현재 음성 통신 방식의 주류로 떠오를 수 있었다. 음성 압축 방식은 크게 파형 부호화, 보코더, 혼성부호화의 세 가지 부류로 나눌 수 있다[1]. 이중 파형부호화는 가능한 파형 자체의 원형을 보전하면서 산술적으로 압축하는 방식으로서 음질은 좋은 반면, 그 압축 정도에 한계가 있다. 이와는 달리 보코더 방식은 파형 자체보다는 그 음성을 생성하는 발성 기관을 모델링하고 이 모델을 표현하는 파라메터를 분석, 추출하여 전송하는 방식으로 파형부호화 방식에 비해 압축률은 높으나 음질이 떨어진다. 주로 4.8 kbps 이하의 압축률을 가지며 LPC-10 보코더가 대표적이다. 마지막으로 혼성부호화는 위의 두 가지 압축 방식을 모두 활용하여 좀더 효율적인 압축이 가능하게 하는 것으로 현재의 음성 압축은 대부분 이 방식을 사용한다. 혼성부호화기는 합성에 의한 분석(analysis-by-synthesis, AbS) 기법을 이용하여, 압축하는 과정에서 음성 신호를 합성하여 실제 입력 신호와의 차이를 최소가 되도록 하는 여기신호 파라메터를 벡터 양자화하여

* 강원대학교 전자공학과 대학원
** 강원대학교 전기전자 정보통신공학부

전송 또는 저장하는 방식이다. CELP 계열의 많은 음성부호화기가 혼성부호화기에 해당하며 본 논문에서 구현하는 G.723.1A도 대표적인 혼성부호화기이다. 혼성부호화기는 AbS과정에서 필연적으로 동반되는 많은 수치 연산으로 인하여 DSP (digital signal processor)를 이용하여 효과적으로 구현될 수 있다.

본 논문에서는 ITU-T에서 표준화한 5.3 kbps와 6.3 kbps의 전송속도를 지원하는 G.723.1A 음성부호화기를 DSP를 이용하여 실시간 구현하는 방법에 대하여 기술하였다. 본 논문의 구성은 2 장에서 G.723.1A의 기본 구조와 특성을 살펴보고 G.723.1A의 실시간 구현에 대하여 논하며, 3 장에서는 구현된 G.723.1A 음성부호화기의 성능을 평가하고 4장에서 결론을 맺도록 하겠다.

2. G.723.1A의 실시간 구현

2.1 ITU-T G.723.1A 음성부호화 알고리즘[2]

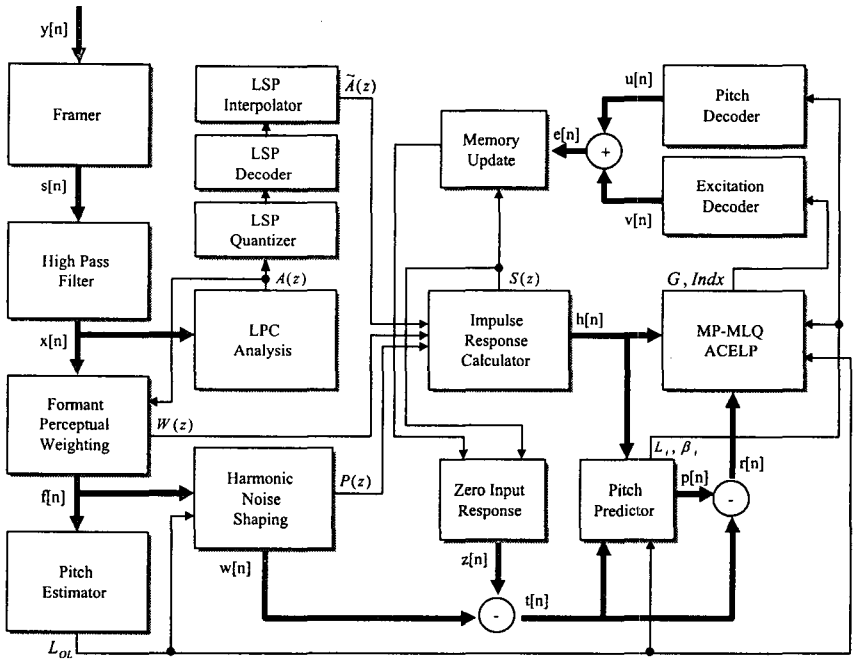

그림 1. G.723.1A 음성부호화기의 블록도

그림 1은 G.723.1A 음성부호화기의 블록도이다. 굵은 선은 신호의 흐름을 의미하며 가는 선은 파라메터의 흐름을 의미한다. G.723.1A 음성부호화기는 저전송률에서 고음질을 유지하기 위해서 analysis-by-synthesis(AbS) 구조를 가지며, 저전송률에도 불구하고 6.3 kbps의 경우 MOS가 3.90으로 'toll' 음질을 나타낸다[3,4]. 그러나 반복적인 합성과 비교 과정을 거침으로써 연산량이 많아지는 단점을 갖고 있다. G.723.1A 부호화기에 입력된 신호는 프레임 단위

로 처리가 되며 한 프레임은 240 샘플(30 ms)이다. 프레임은 다시 60 샘플(7.5 ms)의 서브 프레임 단위로 세분화되어 처리된다. 각 서브 프레임 마다 LPC 분석이 이루어지며 마지막 서브 프레임에 대한 파라메터만 LSP (line spectral pair)로 변환한 후 PSVQ 방식으로 양자화한다. 개루프 피치정보의 추출은 두 개의 서브 프레임인 120 샘플마다 이루어지며, 찾아진 피치는 하모닉 잡음 필터와 피치예측, 그리고 여기신호 부호화에 이용되어진다. 여기신호 부호화는 60 샘플의 서브 프레임마다 실시한다. 5.3 kbps의 경우 ACELP 방식으로, 6.3 kbps의 경우 MP-MLQ 방식을 이용하여 부호화된다. 표 1은 G.723.1A의 부호화 파라메터에 대한 전송률별 비트 할당이다.

표 1. G.723.1A 부호화기 파라메터의 비트 할당

파라메터	5.3 kbps	6.3 kpbs
LPC 인덱스	24	24
적응코드북 지연	18	18
코드북 이득	48	48
펄스 위치	48	73
펄스 부호	16	22
그리드 인덱스	4	4
합 계	158	189

2.2 TMS320VC5402 DSP[7]

본 논문에서 실시간 구현을 위하여 사용한 TMS320VC5402는 Texas Instruments (TI)사의 16 bit 고정소수점(fixed-point) DSP로 100 MIPS의 속도로 동작하면서도 저가이고 전력소모가 적어 모바일 응용에 주로 사용된다. G.723.1A 음성 부호화기 알고리즘의 연산 요구량이 20~25 MIPS 정도로 알려져 있으므로 하나의 TMS320VC5402로 최대 5 채널 구현이 가능하다[5,6]. DSP의 내부 구조는 향상된 Harvard Architecture 구조로 되어 있으며 내부 메모리는 이중 액세스(dual access)가 가능하여 2 개의 데이터를 읽어와 연산한 후에 결과 데이터 1 개를 메모리에 기록하는 동작을 하나의 명령어로 수행 가능하다. DSP는 주어진 시간에 필요한 연산을 모두 끝내야 하는 실시간 처리를 주목적으로 하고 있기 때문에 CPU의 도움 없이도 주변기기로부터 연속적인 데이터 공급을 위하여 DMA를 활용하게 된다. DMA는 설정된 이벤트 발생 시 스스로 데이터를 옮기는 동작을 수행하게 되는데 실시간 동작에서 중요한 역할을 담당하게 된다. 그리고 CPU가 신호처리 연산에만 집중할 수 있도록 어드레스 생성 유닛(address generation unit)을 갖고 있다. TMS320VC5402의 경우 2 종류(데이터, 어드레스)의 어드레스 버스에 필요한 어드레스를 생성해 주는 2 개의 어드레스 생성 유닛을 갖고 있다. 표 2는 TMS320VC5402 DSP의 주요 사양이다.

표 2. TMS320VC5402의 주요 사양

	사 양
성능/실행시간	100 MIPS (10ns/cycle)
메모리	16 K * 16 bit Dual Access On-Chip RAM
내장 주변장치	On-Chip PLL(phase-Locked Loop) 2 개의 McBSP (Multichannel Buffered Serial Ports) HPI-8 (Enhanced 8 bit Host-Port Interface) 2 개의 16 bit Timer, 6 Channel DMA
주요 연산 유닛 및 구조	1 M * 16 bit External Address Space Advanced Multibus Architecture - 4 개의 Data Bus, 4 개의 Address Bus 40 bit ALU, 40 bit Barrel Shifter 2 개의 40 bit Accumulator 17 bit * 17 bit MAC CSSU (Compare, Select and Store Unit) Exponent Encoder 32 bit Long Word Operand 지원

2.3 개발 단계 및 실시간 구현

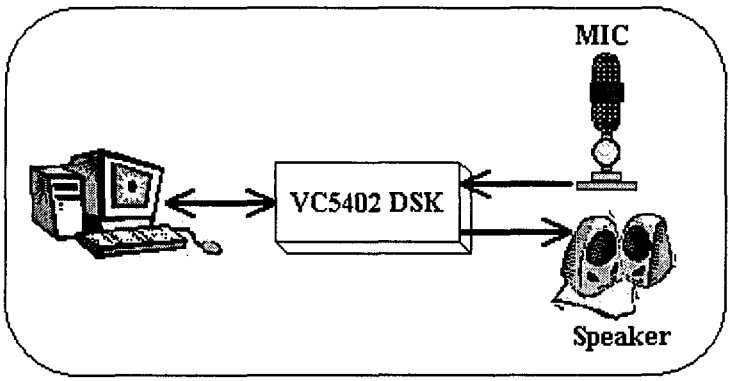

그림 2. 개발 환경

개발을 위하여 TMS320VC5402 DSK에 포함된 하드웨어와 Code Composer Studio (CCS) 버전 2.1을 사용하였다. 개발 환경은 그림 2와 같이 PC에서 CCS를 이용하여 작성된 코드를 DSK 하드웨어로 다운로드한 후 실시간 실행을 하였다. 고정소수점 코드를 개발하기 위하여 다음과 같이 2 단계의 과정을 거쳤다.

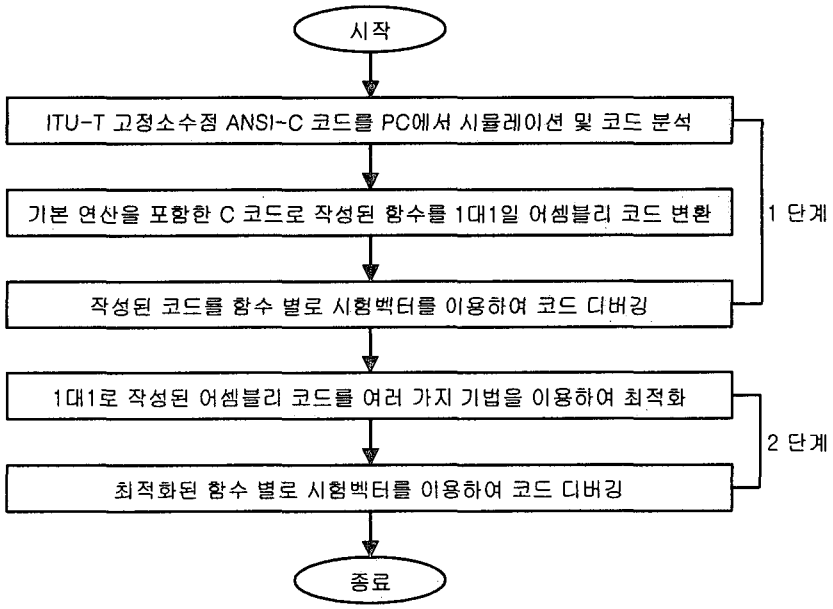

그림 3. 고정소수점 코드 개발 과정

1 단계에서는 ITU-T에서 제공된 고정소수점 ANSI-C 코드를 시뮬레이션해 보고, PC 상에서 C 소스 코드 분석을 통하여 불필요한 코드의 정리 및 제거, 루프 밖에서 미리 계산하여 놓을 수 있는 변수들의 루프 밖으로의 재배치 등과 같은 C 코드 수준에서 할 수 있는 최적화를 수행한 후, 기본 연산(basic operation)을 포함하여 함수 별로 어셈블리 코드로 변환하였다. 이 단계에서는 어셈블리 코드 변환 시 코드 최적화는 고려하지 않았다. 기본 연산 함수는 대부분 DSP의 어셈블리 명령어로 대치가 가능하며 배정수 연산과 관련이 있는 함수만 인라인(in-line) 어셈블리 함수로 작성하였다. 작성된 코드를 함수 별로 시험 벡터를 이용하여 검증을 하였다. 따라서 최종적으로 부호화된 결과는 비트-동일(bit-exact)을 보장한다. 2 단계에서는 다음 절에서 설명할 여러 가지 코드 최적화 기법을 적용하여 1 단계에서 작성된 어셈블리 코드를 최적화하였다. 1 단계와 마찬가지로 최적화된 코드는 함수 별로 시험 벡터를 이용하여 검증하였다. 2 단계를 거쳐 개발된 코드는 다음과 같이 전이중(full-duplex) 방식으로 실시간 동작을 확인하였다. 마이크로 음성을 입력하면 음성은 TLC320AD50 코덱을 통해 A/D 변환된 후에 직렬포트를 통해 DSP로 입력된다. DSP는 직렬포트를 통해 입력된 PCM 데이터를 G.723.1A 부호화기로 실시간 부호화를 하게 된다. 이렇게 부호화된 비트 스트림은 복호화기로 입력되어 다시 PCM 데이터로 변환되어 TLC320AD50 코덱을 통해 D/A 변환된 후 스피커로 출력된다.

2.4 코드 최적화

코드 최적화는 크게 연산속도 최적화와 메모리 사용량의 최적화로 나눌 수 있다. 우선 연산속도의 최적화는 다음 세 가지를 수행하는 것으로 요약된다.

- 연산 자체의 최소화
- 데이터 메모리 액세스의 규칙화
- 분기의 최소화

2.4.1 연산 자체의 최소화

연산 자체의 최소화는 코드 전반에 걸쳐 이루어져야 하지만 대부분의 경우 루프 내의 연산을 최적화하는 것으로 집약된다. 특히 다중 루프의 경우 가장 안쪽 루프의 코드를 최적화할 경우 연산량이 많이 감소하며 최적화를 통해 얻어지는 대부분 연산량 감소는 여기에서 기인하는 것이라고 할 수 있다. 뿐만 아니라 대부분의 DSP들이 지원하는 zero-over loop 기능은 어셈블리 코드로 작성될 경우 루프 자체에서 발생하는 분기로 인한 오버헤드를 최소화하게 된다. 반면 C 컴파일러로 컴파일된 코드의 경우 루프가 가능한 곳에서도 상당 부분 분기 명령으로 처리함으로서 이러한 장점을 활용하지 못한다. 루프 밖에서 미리 연산 가능한 변수들이나 메모리 인덱스들은 C 코드 분석 과정에서 이미 어느 정도 이루어졌다. 루프 내의 조건문 사용을 최소화하고, 2.4.2 절에서 설명할 변수들의 규칙적인 배열을 통한 어드레스 생성 유닛의 활용, 배열의 대칭적 배치를 통한 일부 수학적 연산의 제거, 필터링 부분에서의 원형 버퍼의 적절한 사용 등으로 루프 내의 실행 클럭 수를 최소화하였다. 특히 루프의 반복 횟수가 많은 경우, 횟수에 비례하여 연산량 감소의 이득이 있는 만큼 해당 부분의 코드 최적화에 집중하였다. 연산량이 많은 부분은 코드북 탐색 부분인데 MP-MLQ 고정코드북 탐색의 경우 가장 안쪽 루프가 총 1,320 회 반복이 되는데 위에서 설명한 방법들을 적절히 적용하여 가장 안쪽 루프내의 실행 클럭 수를 절반 가까이 줄임으로서 상당한 연산량 감소를 얻을 수 있었다.

2.4.2 데이터 메모리 액세스의 규칙화

C와 어셈블리를 혼합해서 구현할 때 스택의 동작 방식을 이해하고 스택 내의 변수나 파라메터의 위치를 파악해야 효율적인 코딩이 가능하다. 일반적으로 연속해서 메모리에 어떤 값을 읽고 쓰기를 행할 경우에 해당 메모리 주소 값을 갱신해 주어야 한다. 이 때 스택 내의 변수나 메모리 내의 배열들이 규칙적으로 배열되어 있을 경우에 보조레지스터(auxiliary register)를 이용한 다양한 어드레싱 방법을 이용하여 CPU를 통한 어드레스 갱신을 최소화함으로서 연산속도를 증가시킬 수 있다. 특히 G.723.1A의 경우 많은 테이블을 사용하는데 이러한 테이블의 데이터를 효과적으로 읽는데 유용하다.

표 3. 복잡한 인덱스의 변화를 필요로 하는 메모리 액세스 예

```
for ( i = 1 ; i < CosineTableSize/2 ; i++ ) {     // CosineTableSize = 512
  for ( j = 0 ; j <= LpcOrder/2 ; j ++ )
      CurrVal = L_mac( CurrVal, Spq[LpcOrder - 2 * j +k],
                        CosineTable[i * j % CosineTableSize] );
  ...
}
```

위의 C 코드에서 테이블 CosineTable의 인덱스 계산에 모듈로(%) 연산자를 사용하고 있다. C 컴파일러로 컴파일할 경우 이런 모듈로 연산은 CPU의 ALU가 담당을 하게 된다. 그러나 수작업 어셈블리 변환에서는 어드레스 생성 유닛과 보조레지스터를 이용하여 메모리 인덱스를 갱신함으로서 CPU의 연산 부담을 줄인다. 특히 이 기법은 루프 내의 배열을 처리할 때 실행 클럭 수를 줄이는데도 중요한 역할을 하였다.

표 4. 표 3을 수작업 어셈블리로 변환한 예

```
00  asm(" STM    #512, BK");
    for ( i = 1 ; i < CosineTableSize/2 ; i++ ) { // CosineTableSize = 512
01          asm(" MVMM     AR5, AR2");
02          asm(" STM      #5, BRC");
            ...
11          asm(" RPTB     LABEL_EvaluateSelPoly-1");
12              asm(" MAC  *AR2-, *AR3+0%, B");
13              asm(" MAR  *AR2-");
14          asm("LABEL_EvaluateSelPoly:");
15          asm(" RSBX     FRCT");
             ...
    }
```

위의 코드는 표 3을 수작업 어셈블리로 변환한 예이다. 바깥쪽 루프는 "RPTB" 명령어가 중첩을 허용하지 않는 관계로 어셈블리로 변경을 하여도 zero-over head의 장점을 얻을 수 없어서 C언어를 그대로 사용하였다. 표 4의 "MAC" 명령어를 사용하는 구문을 보면 "*AR3+0%"과 같이 원형 어드레싱(Circular addressing)을 사용하고 있다. 이 원형 어드레싱과 관련된 모듈로 연산은 어드레스 생성 유닛에서 담당을 한다.

2.4.3 분기의 최소화

C5000 계열의 DSP는 6 레벨 깊이의 파이프라인을 갖는다. 명령어 수준에서의 분기(branch) 명령어나 함수 호출(call) 명령어 수행 시 파이프라인의 연속성이 유지되지 않음으로 인하여 연산에 필요한 실행 클럭이 증가하게 된다. 따라서, 루프가 가능한 코드는 파이프라인의 연속성이 보장되는 루프 명령어를 이용하였다. 또한 지연된 분기나 지연된 복귀(return) 명령어들을 최대한 활용함으로서 파이프라인의 연속성이 깨짐으로 인한 연산속도의 감소를 최소화하였다. 한편, 함수 수준에서의 분기는 분기로 인한 오버헤드 이외의 스택 연산이 추가되므로 적절한 기능의 함수는 인라인으로 처리하거나 함수를 호출하지 않고 직접 삽입하였다. 예를 들어 기본 연산 함수들은 인라인으로 처리하였으며, 스트림 패킹(stream packing)에서 사용되는 간단하면서도 루프 내에서 호출되는 함수들은 함수가 호출되는 지점에서 직접 호출되어지는 함수를 어셈블리로 작성하여 삽입하였다. 많은 경우는 아니지만 루프의 횟수가 3 회 이하이면서 코드의 크기가 작은 경우에는 루프를 사용하지 않고 횟수만큼 직접 코드를 삽입하였다.

2.4.4 메모리 최적화

연산 속도 최적화와는 달리 메모리 최적화의 여지는 별로 많지 않다. 사용되는 메모리의 용도는 코드와 데이터를 위한 메모리, 스택 메모리, 테이블을 위한 메모리로 나누어진다. 코드를 위한 메모리의 최적화는 연산 속도를 위한 최적화와 어떤 의미에서는 상충된다. 코드의 크기는 속도 향상을 위하여 인라인이나 함수의 삽입, 적은 횟수의 루프의 경우 루프 대신 해당 횟수만큼의 코드를 직접 삽입하는 등의 방식을 사용하므로 코드의 크기가 늘어나게 된다. 그러나 코드의 경우 메모리 사용량의 최소화보다는 속도 향상이 우선되므로 메모리 최적화 입장에서는 크게 고려하지 않았다. 한편 다중 채널 구현을 고려하지 않을 경우, 데이터를 위한 메모리와 스택 메모리의 합이 최소화되도록 하였다. C5000 계열의 DSP는 저전력 소모로 인하여 단말기에 많이 사용되므로 단일 채널로 동작하는 경우를 가정한 메모리 최소화 역시 의미가 있다. G.723.1A에서 사용되어지는 각 함수마다 많은 지역 변수들을 사용하고 있다. 스택 메모리의 최대 사용량을 구해보면 1,500 words 정도 차지했다. 전체적인 메모리 사용량을 줄이기 위해서 지역 변수들 중에서 서로 관련이 없는 변수들만을 합하면 600 words 정도가 되는데 이러한 변수만을 지역 변수로 사용함으로서 스택 메모리를 600 words로 줄일 수 있었다. 그리고 나머지 변수들은 전역 변수로 460 words 정도의 공용버퍼를 잡아 사용하였다. 따라서 처음 1,500 words 사용되던 스택 메모리를 600 words의 스택 메모리와 전역 변수를 위한 데이터 메모리로 460 words를 사용함으로써 약 440 words의 메모리 양을 줄일 수 있었다. 그러나 다중 채널의 경우, 다중 스레드 방식으로 동작을 하는데 이 경우는 각 스레드마다의 지역 변수들이 독립적으로 유지되어야 한다. 따라서 지역 변수를 임의로 전역 변수화할 수 없으므로 위의 방법을 적용할 수 없다. 한편 C언어에서 int형 변수(16 bits)와 long형 변수(32 bits)를 함께 사용할 경우에도 워드 경계(boundary)와 더블워드 경계를 맞추어 선언함으로서 적은 양이지만 더블워드 배치 시 경계의 불일치에서 발생하는 메모리 낭비를 줄일 수 있었다.

3. 실험 및 결과

실험은 TI사의 TMS320VC5402 DSK의 하드웨어 플랫폼을 사용하여 PC에서 Code Composer Studio 통합 개발 환경을 통해서 실험을 하였다. 기존의 다른 구현 결과와 비교를 위하여 ITU-T에서 제공되는 시험 벡터를 입력 데이터로 사용하였다. 표 5는 2 절에서 언급한 각 단계에서의 성능을 나타낸다. 6.3 kpbs의 경우 최적화를 통하여 45%의 실행 클럭을 줄일 수 있었으며, 5.3 kbps의 경우 37%의 실행 클럭을 줄일 수 있었다.

표 5. 각 단계의 실행 결과

단 계	전송률	최대 (cycle)
1 단계 (C 코드를 특별한 최적화 없이 변환된 어셈블리 코드)	6.3 kpbs	950,030
	5.3 kbps	984,696
2 단계 (최적화된 어셈블리 코드)	6.3 kpbs	522,587
	5.3 kbps	620,630

표 6은 본 논문에서 구현된 부호화기의 연산량을 나타내고 있다.

표 6. 구현된 G.723.1A 음성부호화기의 연산량

		최대(cycle)	평균(cycle)	MIPS (full duplex)
6.3 kbps	Encoder	522,587	471,425	20.0 MIPS
	Decoder	76,639	67,197	
5.3 kbps	Encoder	620,630	558,567	23.14 MIPS
	Decoder	73,609	66,845	

6.3 kbps의 경우 전이중 동작을 위해서는 599,226의 실행 클럭이 필요하며 이를 초당 연산수로 환산하면 (599,226 cycle) /(30 msec) = 20.0 MIPS가 된다. 그리고 5.3 kbps의 경우에는 23.14 MIPS가 된다. 선행 연구 결과를 살펴보면 6.3 kpbs와 5.3 kbps의 연산량이 비슷하다.[5,6] 그러나 본 연구 개발에서는 개발된 결과물의 활용이 음질이 좋은 6.3 kbps에 초점이 맞추어져 있었으므로 6.3 kbps의 최적화에 중점을 둔 관계로 5.3 kbps의 MIPS가 다소 높게 나왔다. 표 7은 본 논문과 동일한 환경에서 개발된 최근의 연구결과[6]와 비교한 것이다.

표 7. G.723.1A 인코더 연산량 비교

	본 논문의 결과 (최대 cycle)	함정표 외[6] (최대 cycle)
6.3 kbps Encoder	522,587	540,074
5.3 kbps Encoder	620,630	548,744

표 7은 인코더의 성능을 비교한 것인데 6.3 kbps의 경우 [6]보다 약간 좋은 성능을 보이나 5.3 kbps의 경우 추가 최적화의 여지가 있음을 알 수 있다. 한편, 다음은 인코더의 주요 모듈 별 연산량을 나타낸 것이다.

표 8. 주요 모듈 별 연산량

	6.3kpbs 인코더(MIPS)	5.3kpbs 인코더(MIPS)
LPC 분석	0.88	0.88
LSP 양자화	1.52	1.52
개루프 피치 검색 & 적응 코드북 검색	7.10	7.10
MP-MLQ	8.48	N/A
ACELP	N/A	11.51
기 타	2.10	2.10
합 계	20	23.1

다음은 본 실험에 사용된 메모리 소요량을 나타낸다.

표 9. 메모리 소요량

	사용된 words (2 bytes)
코 드	11797 words
데이터	1414 words (2134 words)
스 택	600 words
테이블	9450 words

코드 메모리의 양은 순수 알고리즘과 실시간 구현을 위한 주변기기의 설정과 관련된 부분까지 포함된 메모리 량이다. 데이터 메모리의 경우 괄호 안의 메모리 소요량은 인코더, 디코더, VAD/CNG 등의 상태를 담고 있는 구조체 전역변수를 포함한 통상적인 데이터 메모리 이외에 입출력을 위한 실시간 음성 버퍼링을 위해서 3 개의 버퍼 할당에 필요한 메모리(240 * 3 = 720 words)까지 포함한 메모리의 크기이다. 스택 메모리는 앞서 3 절의 메모리 최적화 부분에서 설명한 바와 같이 데이터 메모리 영역에 460 words의 공용 버퍼를 잡아 사용함으로써 600 words 크기로 줄일 수 있었다.

4. 결 론

본 논문에서는 ITU-T G.723.1A 음성부호화기를 TMS320VC5402 고정 소수점 DSP를 이용하여 실시간 구현하는 방법과 최적화 기법을 제시하였다. ITU-T에서 제공하는 ANSI C 코드를 1 단계로 어셈블리 코드로 변환한 후, 여러 가지 코드 최적화 기법을 이용하여 최적화하였다. 최적화 결과 최적화를 적용하지 않은 어셈블리 코드와 비교하여 6.3 kbps의 경우 45%, 5.3 kpbs의 경우 37%의 연산량을 감소시킬 수 있었다. 개발된 코드는 각 단계에서 ITU-T가 제공하는 시험 벡터 이용하여 검증함으로서 비트 동일(bit-exact) 방식으로 개발되었다.

메모리 소요량의 경우 스택 메모리로 사용되는 변수들을 분석하여 상당량의 지역변수들을 공용 데이터 메모리에 할당하여 사용하므로 스택 메모리와 데이터 메모리를 합한 메모리의 양을 29% 줄일 수 있었다. 또한, TMS320VC5402 DSK의 하드웨어를 이용하여 전이중 방식으로 실시간 동작을 확인하였다.

참 고 문 헌

[1] Kondoz, A. M. 1994. *Digital Speech Coding for Low Bit Rate Communications Systems*. Wiley.

[2] ITU-T. 1996. *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s*.

[3] Pritzker, Z. 1995. "ITU G.723.1: A new standard for low bit rate speech coding in visual telephony." *Proceedings of the 1995 DSPx Technical Program*, 227-232.

[4] Huang, X., A. Acero & H. Hon. 2001. *Spoken Language Processing*. Prentice Hall PTR.

[5] Pradhyumnan. 1998. "Real-time implementation of ITU-T G.723.1 dual rate speech coder on a TI TMS320C54x Chip." *Proceedings of ICSPAT*, Vol. 2, 1267-1271.
[6] 함정표, 최용수, 조성범, 강태익. 2002. "DSP를 이용한 실시간 ITU-T G.723.1A의 효율적인 구현." *제15회 신호처리 합동 학술대회 논문집*, 제15권 1호, 134.
[7] Texas Instruments. 2001. *TMS320C54x DSP Reference Set, Volume 1 CPU and Peripherals.*
[8] Texas Instruments. 2001. *TMS320C54x Assembly Language Tools User's Guide.*

접수일자: 2003. 2. 19.
게재결정: 2003. 5. 14.

▲ 이송찬
강원도 춘천시 효자2동 (우: 200-701)
강원대학교 전자공학과
Tel: +82-33-250-6322
E-mail: songchan@dsplab.kangwon.ac.kr

▲ 정익주
강원도 춘천시 효자2동 (우: 200-701)
강원대학교 전기전자 정보통신공학부
Tel: +82-33-250-6322
E-mail: ijchung@cc.kangwon.ac.kr