論文2001-38SP-3-7

IMT-2000 음성부호화 알고리즘의 실시간 DSP 구현

(Real-Time DSP Implementation of IMT-2000 Speech Coding Algorithm)

徐 正 旭 *, 權 洪 錫 *, 朴 萬 鎬 **, 裵 建 星 *

(Jeong-Wook Seo, Hong-Seok Kwon, Man-Ho Park, and Keun-Sung Bae)

요 약

본 논문에서는 3GPP와 ETSI에서 IMT-2000의 음성부호화 방식 표준안으로 채택한 AMR 음성부호화 알고리즘을 분석하고 C 컴파일러와 어셈블리 언어를 이용한 최적화 과정을 거친 후, 고정 소수점 DSP 칩인 TMS320C6201을 이용하여 실시간 구현하였다. 구현된 codec의 프로그램 메모리는 약 31.06 kWords, 데이터 RAM 메모리는 약 9.75 kWords, 그리고 데이터 ROM 메모리는 약 19.89 kWords 정도를 가지며, 한 프레임(20 ms)을 처리하는데 약 4.38 ms가 소요되어 TMS320C6201 DSP 칩의 전체 가용한 clock의 21.94%만 사용하여도 충분히 실시간으로 동작 가능함을 확인하였다. 또한, DSP 보드상에서 구현한 결과가 ETSI에서 공개한 ANSI C 소스 프로그램의 수행 결과와 일치함을 검증하였고, 구현된 AMR 음성부호화기를 sound I/O 모듈과 결합하여 실험한 결과, 어떠한 음질의 왜곡이나 지연 없이 실시간으로 충분히 동작함을 확인하였다. 마지막으로, Host I/O와 LAN 케이블을 이용하여 AMR 음성부호화 알고리즘을 통한 쌍방간 실시간 통신을 full-duplex 모드로 확인하였다.

Abstract

In this paper, we performed the real-time implementation of AMR(Adaptive Multi-Rate) speech coding algorithm which is adopted for IMT-2000 service using TMS320C6201, i.e., a Texas Instrument's fixed-point DSP. With the ANSI C source code released from ETSI, optimization is performed to make it run in real-time with memory as small as possible using the C compiler and assembly language. Implemented AMR speech codec has the size of 32.06 kWords program memory, 9.75 kWords data RAM memory, and 19.89 kWords data ROM memory. And, The time required for processing one frame of 20 ms length speech data is about 4.38 ms, and it is short enough for real-time operation. It is verified that the decoded result of the implemented speech codec on the DSP is identical with the PC simulation result using ANSI C code for test sequences. Also, actual sound input/output test using microphone and speaker demonstrates its proper real-time operation without distortions or delays.

* 正會員, 慶北大學校 電子電氣工學部
(School of Electronics and Electrical Engineering, Kyungpook National Univ.)
** 正會員, 韓國電子通信硏究員
(Electronics and Telecommunications Research Institute)
接受日字:2000年5月16日, 수정완료일:2001年1月31日

Ⅰ. 서 론

현재의 유선통신망이 PSTN과 N-ISDN을 거쳐 B-ISDN으로 진화함에 따라 무선통신 분야도 디지털 셀룰러와 PCS를 거쳐 3세대 IMT-2000으로 발전하고 있다. IMT-2000에서는 기존의 디지털 셀룰러 및 PCS에서 제공하는 음성서비스의 품질을 유선 전화급으로

향상시키는 동시에 사용자의 선택적 가입을 통해 영상 정보를 주요한 요소로 하는 무선 멀티미디어 서비스의 제공을 목표로 하고 있다. 유럽의 ETSI 및 3GPP에서는 IMT-2000 통신서비스를 위한 표준화 작업을 주도하고 있는데, 음성부호화의 경우 무선채널의 환경변화에도 toll quality의 음성품질을 유지할 수 있도록 8개의 전송모드를 갖는 AMR(Adaptive Multi-Rate) 음성부호화 방식을 표준안으로 선정하여 그에 관련된 작업을 1999년 12월에 완료하였다[1]. 본 연구에서는 광대역 이동 멀티미디어 서비스를 위한 핵심요소 기술개발의 일환으로 IMT-2000에서 채택된 음성부호화 방식의 핵심기술인 AMR 음성부호화 알고리즘을 연구하고 분석하여 고정 소수점 DSP 칩인 TMS320C6201[2]을 장착한 EVM 보드에서 실시간 구현하였다. DSP로 구현된 시스템의 결과는 ETSI에서 공개된 ANSI C 소스 프로그램[3]의 수행결과와 일치함을 검증하였으며, 마이크 및 스피커를 장착한 2 대의 부호화기를 LAN 케이블로 연결하여 full- duplex 모드로 실시간 동작함을 확인하였다. 본 논문의 구성은 다음과 같다. 먼저 II장에서는 AMR 음성부호화기의 알고리즘에 대해 간략히 설명하고, III장에서는 TMS320C6201 DSP 칩의 특징과 성능을 서술하고 외부 디바이스를 이용한 음성 입/출력 방법에 대해 설명한다. 그리고, IV장에서는 구현된 알고리즘의 최적화 과정을 설명하고, V장에서 실험결과 및 검토사항을 제시하며, 마지막으로 VI장에서 결론을 맺는다.

II. AMR 음성부호화 알고리즘의 개요

AMR 음성부호화기는 multi-rate 음성부호화기[1], 입력 신호의 음성/비음성 여부를 판단하는 VAD(Voice Activity Detector)[4], 비음성 구간에서 전송률을 낮추기 위해 comfort noise generation 시스템을 포함하는 SCR(Source Controlled Rate)[5] 부분, 그리고 전송 에러나 패킷 손실의 영향을 극복하기 위한 에러 복구 메카니즘[6] 등으로 구성된다. Multi-rate 음성부호화기는 4.75 kbits/s에서 12.2 kbits/s 사이의 8 가지 source rate와 low rate background noise 인코딩 모드(1.8 kbits/s)를 가진 하나의 통합된 음성부호화기다[1]. 이 음성부호화기는 한 프레임마다 무선채널 환경에 따라 codec 모드를 변환함으로써 전송률을 바꿀 수 있다. Codec 모드에 따른 전송율은 표 1과 같다.

표 1. AMR 음성부호화기의 codec 모드와 전송율

Table 1. codec mode and bit-rate of AMR codec.

Codec 모드	Source codec bit-rate
AMR_12.20	12.20 kbits/s (GSM EFR)
AMR_10.20	10.20 kbits/s
AMR_7.95	7.95 kbits/s
AMR_7.40	7.40 kbits/s (IS-641)
AMR_6.70	6.70 kbits/s (PDC-EFR)
AMR_5.90	5.90 kbits/s
AMR_5.15	5.15 kbits/s
AMR_4.75	4.75 kbits/s
AMR_SID*	1.80 kbits/s *

(*) SID 프레임이 연속적으로 전송된다고 가정

1. AMR 인코더의 구성 [1]

AMR에서 사용되는 8 가지 codec 모드의 음성부호화기는 CELP(Code Excited Linear Predictive) 모델에 기본을 두고 있는데, 일반적으로 CELP 모델은 LP(Linear Prediction) 분석 과정 외에 adaptive 코드북과 fixed(innovative) 코드북으로 구성된 여기신호를 추정하는 과정을 포함하고 있다[7]. AMR 음성부호화기의 인코더 구조를 그림 1에 나타내었다. 8 kHz로 샘플링된 음성 프레임(20 ms/프레임)에 대해서 CELP 모델을 사용하여 세 가지 파라미터, 즉 LP 필터 계수, 코드북 이득과 인덱스를 추출하게 된다. 이 때, 8 개의 codec 모드에 따라 각 파라미터에 할당되는 비트 수가 달라지게 되는데, 한 프레임당 244 비트(12.2 kbits/s 모드)에서 95 비트(4.75 kbits/s 모드) 사이로 양자화되어 전송된다.

AMR의 부호화 과정을 각 단계별로 개괄적으로 살펴보면 다음과 같다. 먼저, 전처리 과정에서는 불필요한 저주파 성분을 제거하기 위한 고역통과 필터링과 고정 소수점 구현시 발생할 수 있는 overflow를 줄이기 위한 down-scaling 과정이 수행된다. LP 분석은 codec 모드에 따라서 수행 횟수가 달라지게 되는데, 12.2 kbits/s 모드에서는 한 프레임마다 두 개의 비대칭 창함수를 이용하여 두 번 수행된다. 이때 lookahead는 이용되지 않고, Levinson-Durbin 알고리즘을 이용해서 LP 계수를 구한 후 양자화와 interpolation을 쉽게 할 목적으로

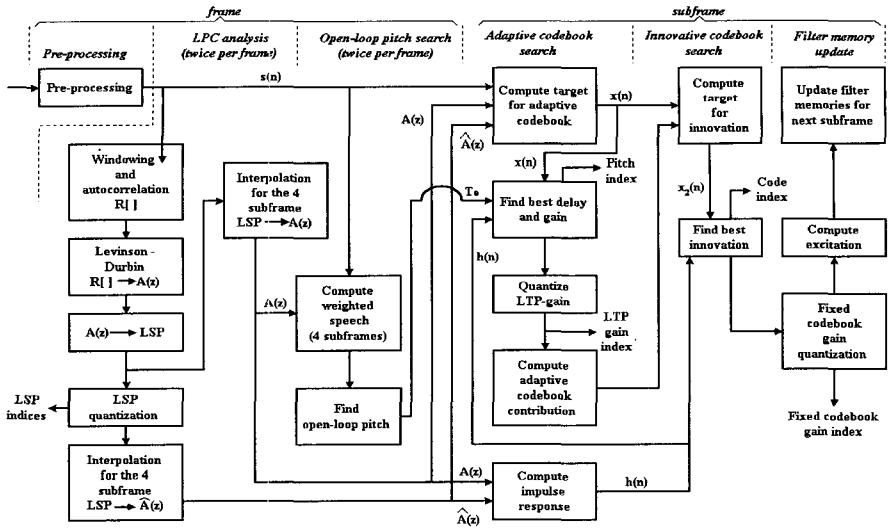


그림 1. AMR 음성부호화기의 인코더 구조
Fig. 1. Block diagram of AMR encoder.

LP 계수를 LSP(Line Spectrum Pair) 영역으로 변환시킨다. 나머지 7 개의 모드에서는 LP 분석이 한번만 수행되며, 5 ms의 lookahead가 이용된다. 나머지 과정은 12.2 kbits/s 모드에서와 동일하다. AMR 음성부호화기에서는 LP 분석 과정을 거친 후에 한 프레임을 네 개의 서브프레임(5 ms)으로 나누어 분석하게 되는데, 이를 위해 현재 프레임과 인접한 프레임의 계수들을 interpolation하여 사용한다.

Open-loop 피치 분석은 피치 분석을 간단하게 하고 closed-loop 피치 탐색을 추정치 부근의 작은 범위 이내로 제한하기 위해서 한 프레임마다 두 번씩 수행된다. 12.2, 7.95, 7.40, 6.70, 5.90 kbits/s 모드에서는 [18, 35], [36, 71], [72, 143] 등의 세 가지 범위 내에서 식 (1)과 같이 correlation을 구해 각각의 최대치를 정규화하고, 최대치에 해당하는 delay 중에서 가장 낮은 범위에 있는 값을 최적의 open-loop delay T_{op}로 선택한다. 단, 10.2 kbits/s 모드의 경우에는 correlation을 구할 때 따로 정의된 가중(weighting) 함수를 곱하여 계산한다. Delay 범위를 세 가지로 나누고 낮은 범위에 있는 값을 선택하는 것은 피치의 multiples를 선택하는 것을 막기 위해서이다. 5.75, 4.9 kbit/s 모드에서는 open-loop 분석을 프레임당 한 번씩 수행하는 것을 제외하고는 위와 똑같은 과정을 수행한다.

$$O_k = \sum_{n=0}^{79} s_w(n)\, s_w(n-k) \qquad (1)$$

여기서, $s_w(n)$는 가중 필터를 통과한 음성신호이다.

이런 과정을 거친 후, adaptive 코드북과 fixed 코드북 추정을 하기 위해서 서브프레임 단위로 가중 합성필터의 impulse response $h(n)$을 구한다. Adaptive 코드북 탐색은 서브프레임 단위로 수행되고, closed-loop 피치 탐색과 adaptive 코드벡터 계산과정이 수행된다. 모든 codec 모드에서 closed-loop 피치 탐색은 원음성신호와 합성된 음성 신호사이의 mean- squared weighted 에러를 최소화시키는 값을 선택한다.

Closed-loop 피치를 추정하는 과정은 각 codec 모드에 따라 달라지는데, 일반적으로 계산범위와 해상도에서 차이를 가진다. 즉, 12.2 kbits/s 모드의 경우 fractional 피치 delay를 1/6 해상도 단위로 구하는 반면, 나머지 모드들은 1/3 해상도로 구한다. 또, 12.2 kbits/s의 경우 피치 계산범위가 넓은 반면, codec 모드가 낮아질수록 피치 계산범위는 줄어들게 된다. 이런 과정을 거쳐 최적화된 전수 피치 delay가 결정되면 1/6의 간격으로 정수 피치의 [-3/6, +3/6]의 범위에서 fraction 값을 구하여 최종적인 Fractional 피치 lag값을 결정한다. 그리고, algebraic 코드북을 찾는데 이용되는 adaptive 코드벡터 $v(n)$은 주어진 정수 delay k와 fraction t에서의 여기신호 $u(n)$을 interpolation해서 구한다.

AMR 음성부호화기에서 algebraic 코드북 구조는 ISPP(Interleaved Single-Pulse Permutation) 구조에 기

반하고 있으며, 서브프레임 단위로 구성되어 있다. 각 codec 모드에 따라 코드북은 달라지는데, 코드북 내의 각 innovation 벡터는 +1 혹은 -1의 크기를 가진 2 ~ 8 개(codec 모드에 따라)의 non-zero 펄스를 가지고 있으며 하나의 서브프레임을 구성하는 40개의 펄스 위치는 2 ~ 5 개(codec 모드에 따라)의 트랙으로 나누어진다. 각 트랙에 포함된 펄스는 codec 모드에 따라 2 ~ 10 개가 선택되고, 그 펄스들의 위치와 크기를 부호화한다. Algebraic 코드북 탐색은 가중 필터를 거친 음성신호와 가중 필터와 합성 필터를 거친 음성신호 사이의 mean square 에러를 최소화하여 수행한다. 즉, c_k를 인덱스 k에서의 algebraic 코드벡터라고 할 때, closed-loop 피치 탐색에 사용된 타깃 신호에서 식 (2)와 같이 adaptive 코드북 기여도를 빼서 값을 갱신한 $x_2(n)$과 c_k사이의 correlation이 최대가 되는 인덱스를 찾아서 수행하게 된다.

$$x_2(n)=x(n)-\hat{g}_p y(n),\quad n=0,\ldots,39 \qquad (2)$$

여기서, $y(n)=v(n)*h(n)$: filtered adaptive codebook $\hat{g}_p$: quantified adaptive codebook gain이다.

Agebraic 코드북 이득의 양자화는 fixed 변수를 가진 MA prediction을 이용해서 수행한다. 4차의 MA prediction을 수행하여 innovation 에너지와 predicted 에너지를 구한 후, 에너지의 차가 가장 작은 값을 찾아 각 codec 모드마다 정의된 상수들을 이용하여 양자화하게 된다.

한 프레임의 분석이 끝난 후, 다음 프레임에서의 타깃 신호를 구하기 위해서 합성 필터와 가중 필터의 상태를 갱신해 주어야 한다[1]. Adaptive 코드북과 algebraic 코드북의 이득들을 구하면 현재 서브프레임에서의 여기 신호를 식 (3)과 같이 구한다. 가중 필터 상태는 입력신호에서 adaptive 코드북 탐색에서 구해진 adaptive 코드북 이득과 코드벡터의 곱, 그리고 algebraic 코드북 이득과 코드벡터의 곱을 각각 뺀 나머지를 계산함으로써 갱신된다.

$$u(n) = \hat{g}_p v(n) + \hat{g}_c c(n)\ n=0,\ldots,39 \qquad (3)$$

여기서, $\hat{g}_p$: adaptive codebook gain $\hat{g}_c$: fixed codebook gain

$v(n)$: adaptive codebook vector $c(n)$: fixed codebook vector 이다.

2. AMR 디코더의 구성 [1]

AMR 디코더는 수신된 파라미터(LP 파라미터, adaptive 코드북 벡터, adaptive 코드북 이득, fixed 코드북 벡터, fixed 코드북 이득)를 디코딩하고, 합성음을 얻기 위해 합성 과정을 수행하게 된다. 이렇게 만들어진 합성음은 post-filtering과 up-scaling 과정을 거친 후 최종적인 합성음이 만들어지게 된다. 그림 2는 AMR 음성부호화기의 디코더 구조를 나타내고 있다.

3. SCR 및 VAD의 개요 [5]

SCR(Source Controlled Rate) operation은 AMR 음성부호화기에서 음성의 비활성을 고려하여 가능하면 더 낮은 전송율로 음성을 인코딩하는 메카니즘이다. SCR의 목적은 사용자 단말기의 전력소모를 줄이고 네트워크에서의 전체적인 load를 줄이는데 있다. SCR의 전체적인 링크는 그림 3과 같다. TX(Transmit) 쪽의 SCR 핸들러는 framing unit에 TX_TYPE(2 비트, 4 개의 모드)에 의해서 표시된 트래픽 프레임을 전송하게 된다. 각각의 프레임은 정보를 포함한 비트 영역와

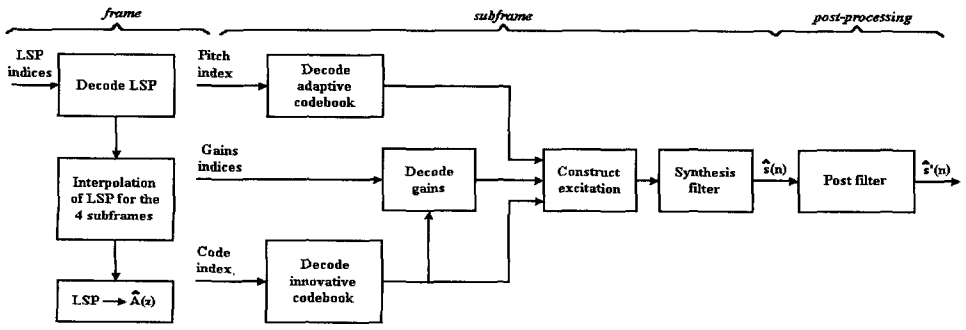


그림 2. AMR 음성부호화기의 디코더 구조
Fig. 2. Block diagram of AMR Decoder.

codec 모드, 그리고 TX_TYPE으로 구성되며 TX_TYPE에 따라 그 프레임의 contexts(특징 파라미터, comfort noise 혹은 none)가 조절된다. RX(Receive) 쪽에서는 수신된 트래픽 프레임에서 TX_TYPE을 RX_TYPE(3 비트: 8 개의 모드)으로 변환한 후, 그 값에 따라 AMR 디코더로 contexts를 전달하여 합성신호를 만들거나, 에러 복구 과정[6]을 수행하기도 하고, 혹은 comfort noise generation을 수행하게 된다.

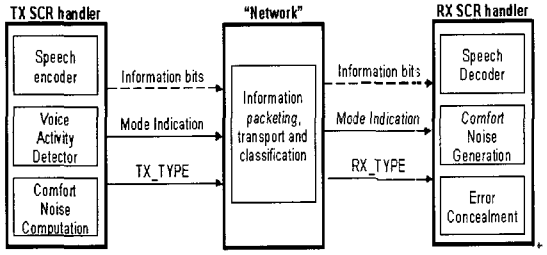


그림 3. One link SCR operation의 구조
Fig. 3. Block diagram of one link SCR operation.

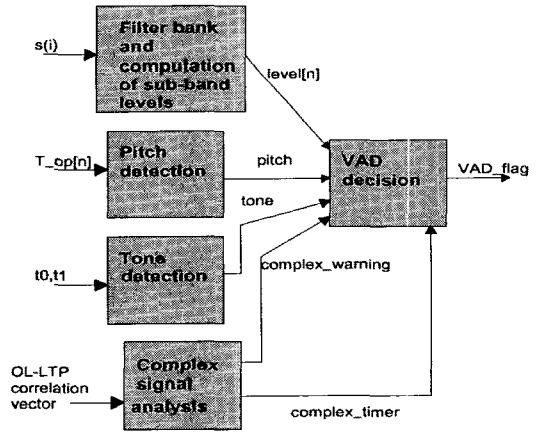


그림 4. VAD 알고리즘(option 1)의 전체 구조도
Fig. 4. Block diagram of VAD option 1.

VAD 알고리즘은 입력된 20 ms 길이의 프레임이 음성, 음악 등 전송해야 하는 신호인지의 여부를 결정하여 Boolean flag로 표시해 주는 기능을 한다[4]. VAD option 1 알고리즘은 그림 4에 나타나듯이 인코더의 파라미터를 이용하여 출력 Boolean VAD flag(VAD_flag)를 만든다. 입력 샘플들은 서브밴드별로 나누어지고, 각 밴드신호에 대한 레벨(level [n])이 계산되며, 인코더의 open-loop lags(T_op[n]), autocorrelation 값들(t0, t1), 그리고 open-loop correlation 벡터 값을 이용하여 피치 검출, 톤 검출, 그리고 Complex 신호 검출 과정을 거쳐 현재 입력 프레임 신호의 음성 여부를 결정한다. 최종적인 VAD 판단은 구해진 flag 값과 hangover를 비교하여 결정한다[6].

Ⅲ. TMS320C6201 DSP의 구성 및 Sound I/O 구현

1. TMS320C6201 DSP의 구성 [8]

TMS320C6201 DSP(Digital Signal Processor)는 Texas Instrument사의 TMS320C62xx 고정소수점 DSP 제품군의 하나로서, TMS320C6xx EVM 보드는 그림 5와 같이 CPU, 메모리, 주변 장치 등으로 구성되며, 각각의 구성요소들은 program bus, data bus, peripheral bus 등을 통해 연결된다. TMS320C6201 CPU는 하나의 clock 사이클 당 8 개의 32 비트 instruction을 8 개의 functional unit을 이용하여 동시에 수행할 수 있는, Texas Instrument사에서 개발된 VLIW(VelociTI advanced Very-Long Instruction Words) 구조에 기반을 두고 있다. 이를 이용하면 instruction packing을 통해서 프로그램 크기를 감소시킬 수 있고, 100% conditional instruction을 써서 빠른 실행속도를 얻을 수 있다. 또한 CPU는 200 MHz의 클럭율에서 최대 성능 1600 MIPS(Million Instructions Per Second)의 성능을 발휘할 수가 있다. TMS320C62xx EVM 보드에는 내부 메모리와 외부 메모리가 포함되어 있는데, 내부 메모리는 프로그램 메모리와 데이터 메모리가 각각 64 kbytes씩 구성되며, 외부메모리는 동기식 메모리인 SBSRAM 및 SDRAM과 비동기식 메모리인 SRAM과 EPROM으로 나눌 수 있다. 또한, CPU 내에는 DMA controller가 있어서 CPU와는 독립적으로 memory-mapping된 위치의 데이터를 다른 위치로 옮겨 줄 수 있다. 그리고, TMS320C6xx EVM에서는 PCI 버스를 이용해서 DSP HPI에 접근할 수가 있어서 host 소프트웨어가 DSP의 모든 메모리 영역을 읽고 쓸 수가 있다. 이 밖의 주변장치로는 telecommunication, 오디오 및 다른 직렬 장치 등과 직접 통신할 수 있는 두 개의 McBSP(Multi-channel Buffered Serial Port), 두 개의 32 비트 타이머, PLL(Phase Locked Loop) 클럭 발생기 등이 있다.

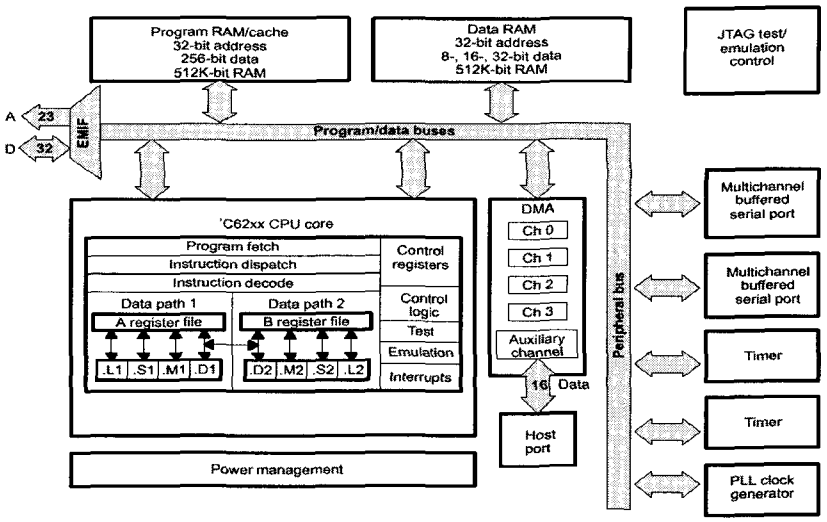


그림 5. EVM 보드의 구조도
Fig. 5. Block diagram of EVM board.

2. 실시간 처리를 위한 Sound I/O 구성 및 Host I/O의 개요 [9]

McBSP 드라이버, audio codec library, 그리고 board support library로 구성되어 있는 DSP support 소프트웨어는 TMS320C6201 EVM 보드를 제어하는데 필요한 함수 및 드라이버를 제공한다. 따라서 이러한 library를 이용함으로써 Sound I/O 제어 및 Host(PC)와의 통신을 쉽게 구현할 수가 있다. EVM 보드에서 사운드 출력은 McBSP를 통하여 이루어지는데, 보드에서 사운드의 실제 출력에 관여하는 장치는 stereo audio codec(CS4231A)으로서 16 비트 스테레오 사운드와 5.5 kHz에서 48 kHz까지의 샘플링율을 제공하고, 입력은 마이크와 라인 단자가 있으며 출력은 라인단자가 있다. 그리고 8 또는 16 비트의 선형 양자화 데이터형과 μ-law 또는 A-law 데이터형도 제공한다. 본 논문에서는 마이크로부터 8 kHz로 샘플링된 음성 데이터를 입력받아서 AMR codec을 거친 후, 스피커를 통해 실시간으로 출력하도록 구현하였다. 이때, 모든 수행동작은 DSP 보드에서 이루어지며, 실제 실시간 음성 입출력을 수행하고자 할 때는 sound I/O에 관련한 함수를 포함하고 있는 dev6x.lib와 drv6x.lib를 link해야한다. 그리고, 외부 디바이스(마이크, 스피커)로부터 데이터를 입출력하기 위해 비동기함수인 mcbsp_async_receive()와 mcbsp_async_send()를 사용하였으며, double buffering을 통해 첫 번째 버퍼가 입력 데이터를 받는 동안 두 번째 버퍼에 저장된 데이터를 처리하여 스피커로 출력하도록 구현하였다.

Host I/O는 EVM 보드와 PC가 데이터를 주고받고자 할 때 사용하는 모듈이다. Host와 보드 사이에 대량의 데이터를 주고받을 필요가 있을 때 사용하는 방법으로 본 논문에서는 EVM 보드와 PC를 연결시켜주는 PCI 컨트롤러내의 FIFO 레지스터를 통하는 방법을 이용하였다. EVM 보드에서 PCI 버스를 제어하는 장치인 PCI 컨트롤러는 그 내부에 FIFO 레지스터가 있어 PC와 CPU 사이의 데이터 통신을 가능하게 한다. PCI 컨트롤러 내에는 읽기, 쓰기가 가능한 FIFO 레지스터가 각각 8 개씩 존재하고 한 쪽에서 레지스터에 데이터를 쓰면 다른 쪽에서는 그 레지스터를 읽어들임으로써 데이터 전송을 가능하게 한다.

Ⅳ. AMR 음성부호화 알고리즘의 최적화 과정

AMR 음성 부호화기의 최적화 작업은 ETSI에서 제공하는 ANSI C 소스 프로그램[3]을 기반으로 C 컴파일러와 어셈블러를 이용하여 수행하였는데, 그 과정을 간략히 요약하면 다음과 같다.

- AMR 음성 부호화 알고리즘을 충분히 분석한 후, 우선적으로 C 프로그램 상에서 가능한 최적화 작업을 먼저 수행하고 어셈블러 변환시 더 나은 최적화가 이루어질 수 있는 것은 어셈블러로 변환하여 작업을 수행한다.

- 기본적인 연산을 수행하는 함수는 'C6x 컴파일러가 제공하는 intrinsic 함수로 대치한다.
- 반복 횟수가 같은 여러 개의 for 루프가 존재할 경우 알고리즘에 위배되지 않는 범위 내에서 적당한 개수만큼을 하나로 묶는다.
- 유사한 구조를 가지는 함수의 경우 알고리즘에 위배되지 않는 범위 내에서 임의의 flag 값을 이용하여 하나의 함수로 묶는다.
- 16 비트 연산으로 이루어진 함수는 32 비트 연산으로 수정한다.
- Sotfware pipeline을 이용하여 어셈블러로 함수를 구현한다.
- If 문 안의 조건문은 sub( ) 등과 같은 함수를 사용하지 않고, 단순히 대수기호 <, >, ==를 이용하여 나타낸다.

1. Intrinsic 함수로의 변환

'C6x 컴파일러는 'C6x 명령어에 직접 대응 되도록 미리 만들어둔 함수인 intrinsic을 제공하고 있다[10]. ANSI C 프로그램에서 사용하고 있는 함수들 중에서, 아래의 표 2와 같이 두 개의 16 비트 값을 곱해서 32 비트 값으로 리턴 해주는 L_mult( ) 함수를 _smpy( ) 라는 intrinsic 함수로 나타낼 수 있는데, L_mult( )가 많은 명령어를 필요로 하는데 반해 _smpy( )는 하나의 명령어로 똑같은 기능을 수행할 수 있다. 본 논문에서는 AMR 음성부호화기의 실시간 구현 과정에서 사용되는 약 35 개 정도의 기본연산함수를 intrinsic 함수로 대체하였는데, 이 과정이 프로그램 크기와 수행속도를 줄이는데 결정적인 기여를 하였다. 일반적으로 intrinsic을 적용할 경우, 알고리즘에 따라 차이가 있겠지만, 수행속도를 7~10배 정도로 줄일 수 있어서 가장 막강한 최적화 방법이라고 볼 수 있다.

표 2. L_mult( ) 함수의 변환
Table 2. Conversion of L_mult( ) function.

수정 전의 함수	Intrinsic으로 수정 후의 함수

Word32 L_mult
(Word16 var1, Word16 var2)
{
Word32 L_var_out;
L_var_out = (Word32)var1 *
(Word32)var2;
if (L_var_out != (Word32)
0x40000000L)
{
L_var_out *= 2;
}
else
{
Overflow = 1;
L_var_out = MAX_32;
}
return (L_var_out);
}
	#define L_mult(a,b)
	(_smpy(a,b))

2. 16 비트 연산을 32 비트 연산으로 변환

ETSI에서 제공되는 ANSI C 프로그램은 16 비트 연산으로 만들어졌으나 TMS320C6201는 32 비트 연산을 지원하므로[2], 굳이 32 비트 변수를 16 비트로 나누어 연산하고 다시 합칠 필요가 없다. 예를 들어, 아래 표 3에서 나타나듯이 Mpy_32_16(hi,lo,n)은 32 비트 변수와 16 비트 변수를 곱하고, 그 결과를 32 비트 변수에 저장한 후 리턴하는 함수로써 hi는 32 비트 변수의 상위 16 비트, lo는 32 비트 변수의 하위 16 비트를 나타내고, n은 16 비트의 변수를 나타낸다. 이를 32 비트 연산으로 바꾸면 Mpy_32_16_(a, b) 함수로 만들 수 있다. 즉 a는 32 비트의 변수이고 b는 16 비트의 변수가 된다. 16 비트 연산을 32 비트 연산으로 바꾸는 예는 pre-processing 과정, autocorrelation 계산과정, post-processing 과정 등 프로그램 내의 모든 부분에서 적용된다.

표 3. 16 비트 연산을 32 비트 연산으로 변환한 예
Table 3. Conversion example of 16 bits to 32 bits process.

16 비트 연산의 Mpy_32_16()	32 비트 연산의 Mpy_32_16_()

static inline int
Mpy_32_16(short hi,
short lo, short n)
{
int L_32;
L_32 = L_mult(hi,n);
L_32 = L_mac(L_32,
mult(lo,n),1);
return (L_32);
}
	#define Mpy_32_16_(a,b)
	(_sadd(_smpyhl(a,b),
	(_mpyus(a,b)>>16)<<1))

3. 반복문의 통합을 통한 최적화 과정

피치 추정 과정에서 필터의 에너지를 계산하기 위한 calc_filt_energies( ) 함수는 표 4와 같이 처음 세 개의 for 루프를 하나의 루프로 묶을 수 있고 네 번째의 for 루프와 다섯 번째의 for 루프를 하나로 묶을 수 있다. 한꺼번에 많은 for 루프를 묶는다고 해서 최적화가 되는 것은 아니므로 알고리즘에 위배되지 않는 범위 내

표 4. calc_filt_energies( ) 함수의 변환
Table 4. Conversion of calc_filt_energies( ) function.

수정 전의 전체적인 for 루프 구조	수정 후의 for 루프 구조

for(i=0; i<L_SUBFR; i++) {
y2[i]=shr(Y2[i],3); }
for ⇓
compute scalar product <y2[], y2[]>
for ⇓
compute scalar product -2*<xn[],y2>
for ⇓
compute scalar product 2*<y1,y2>
for ⇓
compute scalar product <y1, y2>
	for(i=0; i<L_SUBFR; i++) {
	y2[i]=shr(Y2[i],3);
	compute scalar product <y2[], y2[]>
	compute scalar product -2*<xn[],y2>
	}
	for ⇓
	compute scalar product 2*<y1,y2>
	compute scalar product <y1, y2>

에서 적당한 루프를 묶어야한다. 이렇게 함으로써 프로그램 크기와 수행속도를 줄일 수 있다. 그 이유는 각각의 for 루프는 counter와 compare, 그리고 branch를 가지고 있는데, for 루프를 줄임으로써 이러한 명령어가 한번만 쓰이므로 최적화가 가능하다. 이러한 과정은 calc_unfilt_energies( ) 함수, pitch_ol( ) 함수 등 프로그램 전반에 걸쳐 수행되었다.

4. 유사한 함수의 통합

AMR 음성부호화기에서 LSP 값을 양자화하는 과정은 크게 12.2 kbits/s와 나머지 전송모드로 나눌 수 있다[1]. 12.2 kbits/s를 제외한 나머지 전송모드의 양자화하는 과정은 LSP 값의 위치와 codesize에 따라 함수를 따로 구현되어 있지만, 이 함수사이에 많은 유사한 과정이 포함되어 있어 그림 6과 같이 임의의 flag 값을 사용하여 하나의 함수로 통합할 수 있다. 이와 유사하게 12.2 kbit/s 전송모드에서도 양자화 과정을 하나의 함수로 통합하였다. 이런 과정을 통해 LSF 양자화 프로그램 크기를 약 2/3로 줄일 수 있다. 또한, LSP 파라미터를 interpolation하는 과정에서, 양자화된 LP 파라미터와 양자화하지 않은 LP 파라미터를 interpolation하는 함수는 매우 유사하므로 flag 값을 첨가하여 하나의 함수로 통합할 수 있다. 이러한 과정을 통해 LP 파라미터를 interpolation하는 프로그램 크기를 1/2로 줄일 수 있다.

그 외에 pre_process( ) 함수와 post_process( ) 함수의 구조는 단지 사용하는 필터계수와 변수 하나의 값만이 다를 뿐, 모든 과정이 동일하다. 따라서, 두 함수를 하나의 함수로 통합한 후, 필터계수 값만 따로 정의해주면 프로그램 크기를 줄일 수 있다.

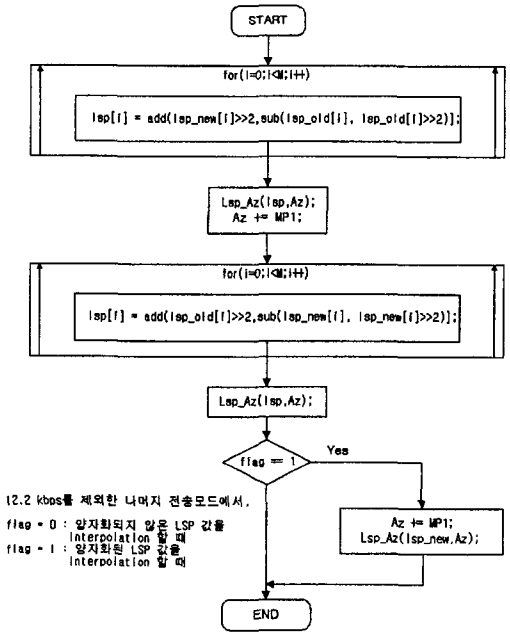


그림 6. 통합된 LSP 양자화 과정의 흐름도(12.2 kbits/s 이외의 전송모드)
Fig. 6. Flow chart of combined LSP quantization procedure(except for 12.2 kbits/s mode).

5. VAD(option 1) 과정의 최적화

VAD(option 1) 과정에서 가장 많은 연산이 필요한 부분은 각 주파수 대역별로 에너지를 구하기 위한 세 번의 5차 필터 뱅크인 filter5( ) 함수와 다섯 번의 3차

표 5. AMR 음성부호화기의 메모리 크기 및 수행속도 비교
Table 5. Comparison of memory size and processing clock.

		음성부호화기 A	음성부호화기 B	음성부호화기 C
프로그램 메모리		52.42 kWords (209.66 kbytes)	38.48 kWords (153.91 kbytes)	31.06 kWords (124.25 kbytes)
데이터 RAM (stack 포함)		25.74 kWords (102.97 kbytes)	24.72 kWords (98.87 kbytes)	9.75 kWords (39.00 kbytes)
데이터 ROM (heap 포함)		23.31 kWords (93.23 kbytes)	21.12 kWords (84.49 kbytes)	19.89 kWords (79.52 kbytes)
1 프레임 (20msec) 처리속도 [clock 수]	Average	18,714,071	2,138,737	876,976
	Maximum	21,634,254	3,838,948	1,517,828
	Minimum	18,229,134	1,648,137	435,917

필터 뱅크인 filter3( ) 함수를 수행하는 부분인데[4], 프로그램 속도 향상과 크기 감소를 위해 이 함수들에서 filter5( ) 함수와 기능이 동일한 first_filter_stage( ) 함수를 제거하였다. 그리고, filter_bank( ) 함수 내에서 각각의 반복 횟수만큼 filter5( ) 함수나 filter3( ) 함수를 호출하여 수행할 경우 반복문 내에서 함수 호출로 인한 NOP(No OPeration)가 발생하여 수행속도가 저하되는 문제점이 있는데, 함수의 원형에서 반복 횟수를 인자로 전달하여 함수 내에서 반복문을 수행하면 부가적으로 필터링과정에서 발생되는 필터 메모리 값을 마지막 반복 과정에서만 저장하게되어 Load와 Store로 인한 명령어를 줄이고 지연시간도 줄일 수 있다. 또한, level_calculation( ) 함수에 대하여서는 반복문을 수행하는 과정을 assembly로 작성함으로써 수행 속도를 향상시켰다.

V. 실험 및 결과

지금까지 언급한 최적화 과정을 통하여 AMR 음성부호화기를 TMS320C6201 EVM 보드에 porting하여 음성신호의 실시간 처리를 구현하였다. 구현된 codec의 성능을 평가하기 위해서 프로그램 메모리와 데이터 메모리의 크기, 그리고 프레임당 수행 시간에 대한 클럭을 측정하고 비교하였다. 먼저, AMR 음성부호화기를 동작시키기 위한 DSP 보드의 메모리 할당은 다음과 같다. 데이터 메모리의 경우, 많이 사용되는 .bss, .far, .stack section을 내부 메모리로 할당했는데, AMR 음성부호화기의 프로그램 크기가 너무 커서 전체를 내부 프로그램 메모리에 넣을 수 없기 때문에, 호출되는 횟수가 많은 함수를 실험적으로 선별하여 내부 메모리에 할당하였다. 또한, 인코더보다 수행속도가 많이 걸리지 않는 디코더와 호출 횟수가 적은 인코더의 함수들은 외부 메모리인 SBSRAM에 할당하였다. 또한, 프로그램의 원할한 수행을 위해 stack의 크기는 0x2500 (약 9.47 kbytes), heap의 크기를 0x1000(약 4.1 kbytes)으로 두고 사용하였다. 구현된 AMR codec의 성능평가는 수치적인 계산을 위해 파일 입출력을 통하여 측정하였다. 파일 입출력은 code composer studio에서 제공하는 파일 입출력 기능[11]을 이용하였으며, 실험에 사용된 입력 음성 파일은 ETSI에서 제공한 test sequence를 사용하였다.

표 5는 아래와 같이 정의된 음성부호화기에서 8 개의 codec 모드를 프레임마다 한번씩 차례로 변환시키면서 처리했을때(All modes)의 처리속도를 비교한 것이다.

- 음성부호화기 A : ETSI에서 제공하는 original source로 구현된 음성부호화기
- 음성부호화기 B : 음성부호화기 A에 intrinsic을 적용하여 구현한 음성부호화기
- 음성부호화기 C : 음성부호화기 B에 최적화 과정을 적용하여 구현한 최종 음성부호화기

표 5에서 볼 수 있듯이 최종적으로 최적화된 음성부호화기인 음성부호화기 C의 크기는 프로그램 메모리와 데이터 메모리 면에서 모두 ETSI에서 제공받은 음성부호화기 A에 비해 상당히 줄어든 것을 확인할 수 있다. 음성부호화기 C의 경우에 모든 데이터와 함수 호출을

표 6. DTX 설정 유무와 모드별 프레임당 소요되는 clock 수 [clock]
Table 6. Comparison of processing clock between DTX mode and none.

모드	DTX 설정			DTX 비설정		
	Average	Maximum	Minimum	Average	Maximum	Minimum
MR122	936,596.6	1,294,370	479,035	1,241,540.0	1,255,925	1,225,253
MR102	952,742.5	1,291,509	540,813	1,213,505.8	1,245,152	1,203,229
MR795	946,483.6	1,374,153	455,104	1,255,804.2	1,330,889	1,201,278
MR740	905,557.4	1,262,510	454,244	1,199,685.6	1,217,298	1,182,665
MR670	1,040,135.2	1,542,005	453,963	1,426,599.4	1,497,884	1,380,153
MR590	811,741.3	1,120,366	453,644	1,034,479.7	1,075,066	991,346
MR515	673,724.3	894,699	450,232	787,419.0	840,381	744,967
MR475	822,091.0	1,149,453	449,752	1,047,432.4	1,106,378	999,247
All modes	876,976.0	1,517,828	435,917	1,150,231.5	1,482,054	746,125

far로 설정함으로써 데이터 RAM이 많이 줄어들게 되었다. TMS320C6201 DSP 칩은 한 클럭이 5 ns이므로 한 프레임인 20 ms를 처리하는데 가용할 수 있는 클럭 수는 4,000,000 클럭이 된다. 음성부호화기 A의 경우는 한 프레임을 인코딩과 디코딩을 처리하는데 걸리는 클럭 수가 평균 18,714,071이므로 실시간 처리가 불가능하지만, 음성부호화기 C의 경우에는 클럭 수가 평균 876,976이기 때문에 DSP 칩의 전체 가용한 클럭의 21.94%만 사용해서도 충분히 실시간으로 동작 가능함을 알 수 있다. 즉, 20 ms인 한 프레임을 처리하는데 약 4.38 ms가 소요되는데, 인코딩의 경우 전체 클럭의 약 70%가 소요되며, 디코딩의 경우 약 30% 정도가 소요된다. 또한 이 결과는 DTX 모드[12]로 동작한 결과이므로 입력 음성에 비음성 구간이 얼마나 포함되어 있는가에 따라서 한 프레임을 처리하는데 소요되는 평균 클럭이 제시된 결과와 차이가 생길 수도 있다. 본 논문에서 사용한 test sequence의 경우 음성과 비음성 구간이 약 53%와 47% 정도 포함되어 있다.

표 6은 DTX 모드를 설정한 경우와 설정하지 않은 경우에 대하여 각 모드에 따른 클럭 수를 측정하여 비교한 것이다. 6.7 kbits/s 모드인 경우에는 다른 모드에 비해서 상대적으로 많은 clock이 필요하며 5.15 kbits/s는 다른 모드에 비해 적은 클럭이 필요함을 알 수 있다. 같은 모드간의 DTX 설정 유무에 따른 최대값의 차이는 음성/비음성 구간을 검출하기 위한 클럭 수를 의미하는 것으로 전체 codec에 비해 많은 연산량을 필요로 하지는 않는다. 그러나, 최소값을 비교해 보면 DTX 모드로 동작하는 경우에는 상당히 많은 클럭이 줄어든 것을 알 수 있다. 따라서 DTX 모드를 사용함으로써 codec에 많은 부하를 주지 않으면서도 채널의 효율을 효과적으로 높일 수 있다.

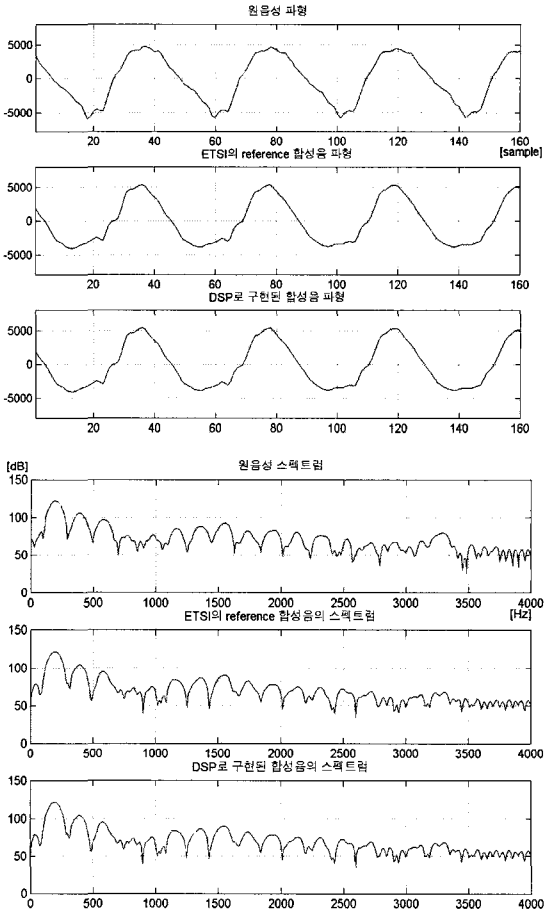


그림 7. 원음성과 합성음의 파형 및 스펙트럼 비교 (유성음 구간)
Fig. 7. Waveform and spectrum of original and synthesized speech(in voiced region).

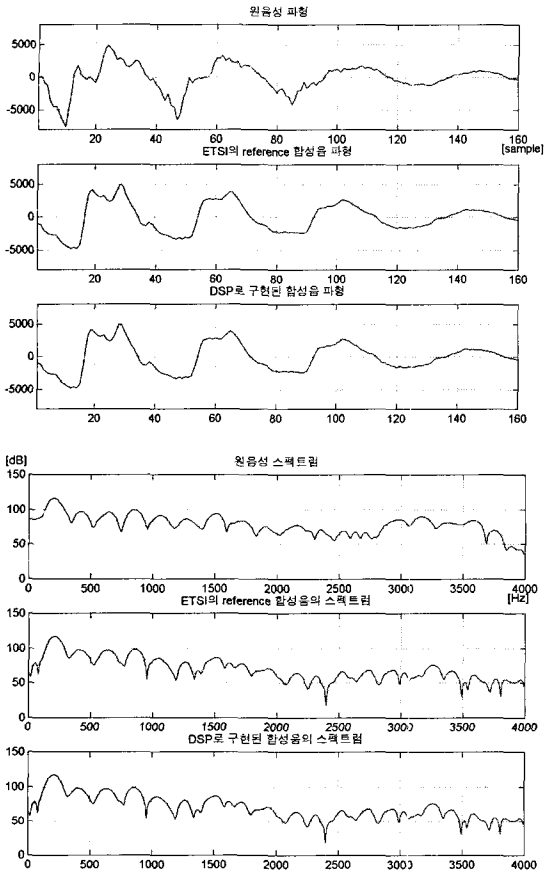


그림 8. 원음성과 합성음의 파형 및 스펙트럼 비교 (천이구간)

Fig. 8. Waveform and spectrum of original and synthesized speech(in transition region).

실험에서 사용한 ETSI에서 제공된 test sequence는 전체 데이터에서 음성구간이 5%정도 포함되어 있다. 그림 7과 8은 일부 유성음 구간과 천이구간에서 원음성과 PC상에서의 음성부호화기 A를 이용한 합성음, DSP 보드에서의 음성부호화기 C를 이용한 합성음 파형과 스펙트럼을 비교한 것이다. 원음성의 파형과 음성부호화기로 합성한 파형이 서로 다른 것을 볼 수 있는데, 이는 AMR 음성부호화기가 PCM과 같은 파형부호화 방식이 아니기 때문이다. 하지만 각 구간에 대한 스펙트럼의 궤적이 거의 일치하고 있으며 기본 주파수와 하모닉 성분이 원음성과 마찬가지로 음성부호화기의 합성음에서도 모두 잘 나타남을 알 수 있다. PC 상에서 C compiler를 이용하여 구해진 합성파형과 DSP 보드상에서의 출력파형들은 같은 결과를 나타내고 있음을 볼 수 있는데, 이는 C 소스 코드 최적화 및 DSP assembly 프로그래밍이 제대로 수행되었음을 의미한다.

지금까지 분석한 결과를 토대로 구현된 AMR 음성부호화기를 3.2절에서 서술한 Sound I/O 모듈과 결합하여 TMS320C6201 EVM 보드에 장착한 마이크로부터 음성을 입력받아 보드에서 AMR 알고리즘을 통해 인코딩과 디코딩을 거친 후, 다시 스피커를 통해 합성음을 출력해 본 결과, 어떠한 음질의 왜곡이나 지연 없이 실시간으로 충분히 동작함을 확인하였다. 또한, 3.2절에서 서술한 Sound I/O, Host I/O, 그리고 LAN 케이블을 이용하여 그림 9와 같은 시스템을 구성한 후, 본 연구에서 구현한 AMR 음성부호화 알고리즘을 이용한 쌍방간 실시간 음성 통신을 full-duplex 모드로 확인하였다.

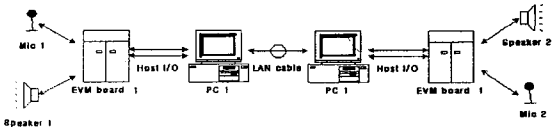


그림 9. AMR 음성부호화기의 full-duplex 모드 시스템 구성

Fig. 9. Block diagram of full-duplex mode system of AMR codec

Ⅵ. 결 론

본 논문에서는 IMT-2000에서 채택된 음성부호화 방식의 핵심기술인 AMR 음성부호화 알고리즘을 연구하고 분석하여 고정 소수점 DSP 칩인 TMS320C6201을 장착한 EVM 보드에서 실시간 구현하였다. 구현된 codec의 프로그램 메모리는 약 31.06 kWords, 데이터 RAM 메모리는 약 9.75 kWords, 그리고 데이터 ROM 메모리는 약 19.89 kWords 정도를 가지며, TMS320C6201 EVM 보드의 경우 내부 프로그램 메모리가 16 kWords이기 때문에 AMR codec 소스를 내부와 외부 메모리에 나누어 저장함으로써 수행속도가 상당히 저하되었다. 구현된 codec은 한 프레임인 20 ms를 처리하는데 걸리는 clock 수가 평균 876,976 정도로 나타나 TMS320C6201 DSP 칩의 전체 가용한 클럭의 21.94%만 사용해서도 충분히 실시간으로 동작 가능함을 확인하였다. 즉, 구현된 AMR codec은 20 ms인 한 프레임을 처리하는데 약 4.38 ms가 소요된다. DSP로 구현된 시스템의 결과는 ETSI에서 공개된 ANSI C 소스 프로그램의 수행결과와 일치함을 검증하였으며, 구현된 AMR 음성부호화기를 sound I/O 모듈과 결합하

여 TMS320C6201 EVM 보드에 장착한 마이크로부터 음성을 입력받아 DSP 보드에서 AMR 알고리즘을 통해 인코딩과 디코딩을 거친 후, 다시 스피커를 통해 합성음을 출력해 본 결과, 어떠한 음질의 왜곡이나 지연없이 실시간으로 충분히 동작함을 확인하였으며, Sound I/O, Host I/O, 그리고 LAN 케이블을 이용하여 AMR 음성부호화 알고리즘을 이용한 쌍방간 실시간 통신을 full-duplex 모드로 구현하였다.

본 논문에서 연구된 결과를 토대로 보다 최적화된 AMR codec을 구현하기 위해서 assembler를 이용한 작업이 조금 더 요구되어 진다.

참 고 문 헌

[1] ETSI Draft EN 301 704, Digital cellular telecommunication system(Phase 2+); Adaptive Multi-Rate(AMR) speech transcoding.

[2] Texas Instrument, TMS320C6201 : Fixed-Point Digital Signal Processor.

[3] ETSI Draft EN 301 712, Digital cellular telecommunication system(Phase 2+); Adaptive Multi-Rate(AMR) speech; ANSI-C code for the AMR speech codec.

[4] ETSI Draft EN 301 708, Digital cellular telecommunication system(Phase 2+); Voice Activity Detector(VAD) for Adaptive Multi-Rate(AMR) speech traffic channels.

[5] 3GPP, 3G TS 26.093 Mandatory Speech Codec speech processing functions : AMR Speech Codec; Source Controlled Rate operation.

[6] ETSI Draft EN 301 705, Digital cellular telecommunication system(Phase 2+); Substitution and muting of lost frames for Adaptive Multi-Rate(AMR) speech traffic channels.

[7] A. M. Kondoz, *Digital Speech*, John Wiley & Sons, 1994.

[8] Texas Instrument, TMS320C6201/6701 Evaluation Module User's Guide.

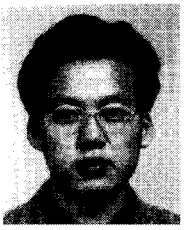
[9] Texas Instrument, TMS320C6201/6701 Evaluation Module Technical Reference.

[10] Texas Instrument, TMS320C62xx Programmer's Guide.

[11] Texas Instrument, Code Composer Studio User's Guide.

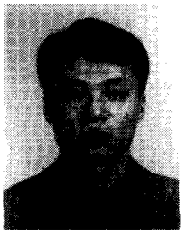
[12] ETSI Draft EN 301 707, Digital cellular telecommunication system(Phase 2+); Discontinuous Transmission(DTX) for Adaptive Multi-Rate(AMR) speech traffic channels.

저 자 소 개



徐 正 旭(正會員)

1995년 2월: 경북대학교 전자공학과 졸업. 1997년 2월: 경북대학교 전자공학과 석사. 1997년 3월~현재: 경북대학교 전자공학과 박사과정. <관심분야> 음성코딩, VoIP, 디지털신호처리



朴 萬 鎬(正會員)

1998년 2월: 경북대학교 전자공학과 졸업. 2000년 2월: 경북대학교 전자공학과 석사. 2000년 3월~현재: 한국전자통신연구원 무선방송기술연구소. <관심분야> 음성코딩, 디지털신호처리, 디지털통신

權 洪 錫(正會員)

1997년 2월: 경북대학교 전자공학과 졸업. 1999년 2월: 경북대학교 전자공학과 석사. 1999년 3월~현재: 경북대학교 전자공학과 박사과정. <관심분야> 적응신호처리, 디지털신호처리, 오디오코딩

裵 建 星(正會員)

1977년 2월: 서울대학교 전자공학과 졸업. 1979년 2월: 한국과학기술원 전기및전자공학과 석사. 1989년 5월: University of Florida 공학박사. 1979년 3월~현재: 경북대학교 전자·전기공학부 교수. <관심분야> 음성신호처리, 디지털신호처리, 디지털통신