Adaptive 알고리즘을 이용한 192-tap Echo Canceller 설계

이 창 덕, 송 인 채, 이 찬 호
숭실대학교 전자공학과
전화 : (02) 816-6073 / 팩스 : (02) 821-7653

Design of a 192-Tap Echo Canceller Using Adaptive Algorithm

Changduck Lee, Inchae Song, Chanho Lee
Department of Electronic Engineering, Soongsil University
E-mail : rwb@hanul.ssu.ac.kr

Abstract

In this paper, we designed a 192-tap echo canceller for a modem using VHDL. To evaluate errors, we used adaptive algorithm. We adopted pipeline technique and realized delay-taps with RAMs. We simulated this design using Altera MAX+PLUS II.

Ⅰ. 서 론

디지털 통신기인 모뎀에는 balancing network의 동작 에러로 인해 두 회선간의 impedance 부정합이 발생하게 되는데 이로 인해 송신 신호가 반향 되어 송신단으로 되돌아오는 echo가 발생하게 된다. 디지털 데이터 통신에서 이 echo가 error의 주원인이 되므로 반드시 제거해야 한다. 디지털 모뎀에서 주로 사용하는 echo제거 방법은 adaptive algorithm을 이용한 echo canceller를 수신단에 사용하는 것이다. 이 방법은 adaptive filter를 이용하여 echo 신호의 복제 신호를 만들어 수신단에 들어온 echo 신호를 상쇄시킨다. echo canceller를 사용하면 양방향(full duplex) 전송이 가능하게 된다. 여기서는 ITU-T V.32(International Telecommunication Unit Telecommunication Sector)[1] 규격의 모뎀에 기초하여 약 20ms정도의 delay를 갖는 echo를 제거할 수 있도록 192-tap echo canceller를 VHDL을 이용하여 설계하였다.

동작속도를 높이기 위해서 파이프라인(pipeline) 구조를 사용하였다. 하드웨어의 복잡도를 줄이기 위해 비교적 간단한 LMS(least-mean-square) algorithm을 사용하였다. 반도체설계교육센터(IDEC)에서 지원해준 Altera Tool을 이용하여 simulation을 수행하였다.

Ⅱ. LMS adaptive algorithm

그림 1은 Processor부분과 error를 계산하는 부분을 보여주고 있다. 입력은 총 L개인데 하나의 입력을 delay시켜서 L개를 생성할 수도 있다. L개의 입력을 받아서 각각 weight $w_{l,k}$를 곱해준다. 입력과 weight와의 곱들의 합 y와 desired response d사이의 error를 수식으로 표현하면 식(1)과 같이 표현된다[2].

$$\varepsilon_k = d_k - y_k \quad (k : \text{time index}) \qquad (1)$$

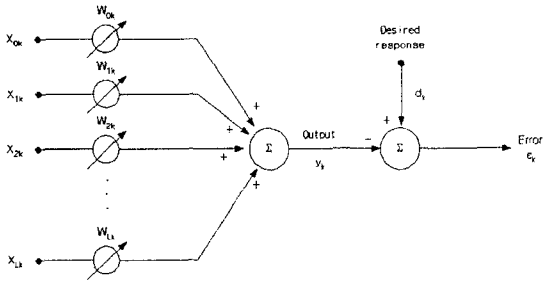

그림 1. Adaptive Linear Combiner

여기서 y_k를 입력 vector X_k와 weight vector W_k로 표시하면 error는 식 (2)와 같이 표현된다.

$$X_k = [\ x_{0k}\ x_{1k}\ x_{2k}\ \cdots\ x_{Lk}\ ]^T$$
$$W_k = [\ w_{0k}\ w_{1k}\ w_{2k}\ \cdots\ w_{Lk}\ ]^T \qquad (2)$$
$$\varepsilon_k = d_k - X_k^T W_k$$

MSE(mean square error)의 gradient(▽)는 식 (3)과 같이 표현할 수 있다[2].

$$\nabla_k = \frac{\partial \varepsilon_k^2}{\partial W} = 2\varepsilon \frac{\partial \varepsilon_k}{\partial W} = -2\varepsilon_k X_k \qquad (3)$$

식 (3)을 이용하여 새로운 Weight를 계산하면 식 (4)와 같이 주어진다.

$$W_{k+1} = W_k - \mu \nabla_k = W_k + 2\mu\varepsilon_k X_k \qquad (4)$$

여기서 μ 값은 damping ratio이다[2].

Ⅲ. Echo Canceller의 구성과 동작

Echo canceller에는 입력을 저장하기 위한 192 word의 memory block과 weight를 even과 odd로 나누어 저장할 192 word의 memory block으로 구성된다. Input data는 14 bit로 구성되므로 1 word가 14 bit가 되고, weight는 1 word가 16 bit로 구성된다. 따라서 input memory block이 총 2688 bit가 되고, weight memory block은 3072 bit가 된다[1].

Weight 값은 weight를 갱신할 때 read와 write가 연속으로 일어나야 하므로 하나의 memory block으로 구성할 경우 2 clock cycle이 필요하게 된다. 하지만 even 과 odd로 나누게 되면, 한 memory block이 read 될 때 다른 memory block은 이전 clock cycle에서 read되어 계산된 결과를 write 할 수 있으므로 시간을 절약할 수 있다. 그림 2는 echo canceller의 구조를 보여주는 블록도 이다.

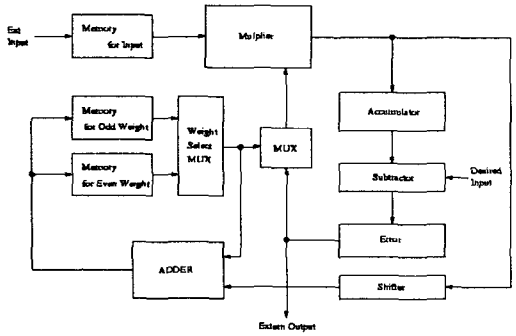

그림 2. Echo Canceller 전체 블록도

Multiplier는 echo canceller의 입력과 weight를 곱하고, weight를 업데이트하기 위해서 입력과 앞에서 계산된 error값과 곱하는 일을 한다. 그러므로 두가지 일을 하나의 multiplier로 수행하기 위해서 weight memory block을 선택하는 mux와 weight 와 error를 선택하는 mux, 두개가 필요하다. 또 현재의 weight는 adder의 또 다른 입력으로 들어가야 한다.

Multiplier는 signed fixed point연산을 한다. fixed point연산은 integer multiplier를 그대로 사용하고 결과만을 shift시킴으로써 쉽게 연산이 가능하다. Accumulator는 입력과 weight의 곱을 누적하는 역할을 한다. Multiplier의 출력이 30 bit이고 191번 더하기 연산이 수행되므로 8 bit만 추가하면 모든 값을 누적할 수 있다. 모든 입력과 weight를 누적하게 되면 desired response와 뺄셈을 수행하여 error를 구하게 된다. Accumulator 와 desired response와의 차이는 echo canceller의 출력이 되며 새로운 weight를 계산하기 위해 multiplier에서 입력과 곱해진다.

새로운 weight를 계산하기 위해서는 multiplier에서 곱해진 error와 입력의 곱에 2μ를 곱하게 된다. 이 값은 1보다 작은 상수이다. 그리고 전송 channel이 유선망이므로 channel의 상태가 비교적 안정되므로 μ값은 작은 값을 갖게 된다[3]. 또한 simulation을 통해 설계자가 선택하게 되므로 8이나 16과 같은 2의 지수승의 나눗셈으로 표현할 수 있다. 2의 지수승의 나눗셈은 shifter를 이용해서 쉽게 구현 할 수 있다. 특히 상수 일 경우는 hard wired shifter를 사용하게 되면 routing만으로 연산을 할 수 있다.

Weight의 변화량과 과거의 weight를 더하면 새로운 weight가 생성된다. 이 연산은 odd weight block이 reading을 하고 있으면 이전 clock cycle에서 연산이 끝난 even 값을 even weight block이 writing을 함으로써 한 cycle에 두 가지 일을 동시에 처리한다. 이렇게 해서 clock cycle의 수를 줄일 수 있다.

Ⅳ. Echo Canceller의 설계

4-1. Timing Chart

Echo canceller가 처음 동작할 때 weight memory block에는 입력이나 designed response에 상관없는 초기 reset 값이 저장되어 있다.

표 1은 첫 번째 error를 구하는 과정에 대한 clock cycle을 보여주고 있다. 첫 번째 clock cycle에서 입력을 받고, 두 번째 clock cycle에서부터 error를 구하기 위한 연산을 한다. 두 번째 clock cycle부터 input memory block은 매 clock cycle마다 읽기 동작을 하고, weight memory block은 교대로 읽기 동작을 수행하여 두 값을 곱한다. Multiplier에서 곱해진 값은 accumulator에서 누적된다. 이러한 동작을 3단계로 나누어 pipeline방식으로 동작시키므로, 입력은 4 clock cycle뒤에 accumulator에 누적되게 된다. 결국 모든 입력과 weight를 곱해서 누적시키기 위해서 194 clock cycle이 걸린다.

표 1. 첫 번째 error를 구하는 과정

Clock	State	Input M	Odd Weight	Even Weight	Mul	Acc	Sub	Error	Add
1	rising	ready write							
	high	write							
	falling	write end							
	low								
2	rising	ready read	ready read		multiply				
	high	read	read						
	falling	read end shift	read end shift		output latch				
	low								add
3	rising	ready read	ready write	ready read	multiply	add			latch
	high	read	write	read					
	falling	read end shift	write end shift	read end shift	output latch	latch			
	low								add
4	rising	ready read	ready read	ready write	multiply	add			latch
	high	read	read	write					
	falling	read end shift	read end shift	write end shift	output latch	latch			
	low								add
5	rising	ready read	ready write	ready read	multiply	add			latch
	high	read	write	read					
	falling	read end shift	write end shift	read end shift	output latch	latch			
	low				multiply	add			add

표 2는 error를 구하는 연산과 새로운 weight를 구하는 연산이 전환되는 과정을 보여주고 있다. 실제로 두 번째 입력이 들어온 다음부터 구한 새로운 weight가 이 echo canceller에서 의미를 갖는 weight가 된다. 그리고 두 번째 출력부터가 실제 echo canceller의 출력이 된다.

표 2. error와 새로운 weight를 구하는 연산이 전환되는 과정

Clock	State	Input M	Odd Weight	Even Weight	Mul	Acc	Sub	Error	Add
193	rising	ready write	ready write	ready read	multiply	add			latch
	high	write	write	read					
	falling	write end	write end shift	read end shift	output latch	latch			
	low								add
194	rising	ready read	ready read	ready write	multiply	add			latch
	high	read	read	write					
	falling	read end shift	read end shift	write end shift	output latch	latch			
	low						subtract		add
195	rising	ready read	ready write	ready read	multiply	add	latch	Output	latch
	high	read	write	read					
	falling	read end shift	write end shift	read end shift	output latch	latch			
	low								add
196	rising	ready read	ready read	ready write	multiply	add			latch
	high	read	read	write					
	falling	read end shift	read end shift	write end shift	output latch	latch			
	low								add
197	rising	ready read	ready write	ready read	multiply	add			latch
	high	read	write	read					
	falling	read end shift	write end shift	read end shift	output latch	latch			
	low								add
198	rising	ready read	ready read	ready write	multiply	add			latch
	high	read	read	write					
	falling	read end shift	read end shift	write end shift	output latch	latch			
	low				multiply	add			add

새로운 weight를 구하기 위해서는 error값이 먼저 구해져야 하므로 latency가 존재한다. 일단 accumulator의 값에서 desired response를 빼서 error를 구하고 이 error를 echo canceller 다음 단에 출력으로 보낸다. 그리고 error와 input memory block의 값을 곱하고 shift시킨 뒤 과거의 weight에 더해서 새로운 weight를 구한다. 과거의 weight를 메모리에서 읽어올 때 두개의 메모리 블록에서 번갈아 read하고 다음 clock cycle에서 계산된 새로운 weight를 메모리에 저장한다.

표 3은 weight 갱신의 마지막 부분을 보여주고 있다. 여기서 보면 Input memory block과 Odd weight block은 먼저 작업이 종료하게 된다. 따라서 다음 입력을 받아 error를 계산하는 부분과 2 clock cycle을 병행하여 처리할 수 있다.

표 3 Weight 갱신의 마지막 부분

Clock	State	Input M	Odd Weight	Even Weight	Mul	Acc	Sub	Error	Add
384	rising	ready read	ready write	ready read	multiply	add			latch
	high	read	write	read					
	falling	read end	write end shift	read end shift	output latch	latch			
	low								add
385	rising			ready write		add			latch
	high			write					
	falling			write end shift		latch			
	low						subtract		
386	rising						latch	Output	
	high								
	falling								
	low								

입력을 하나 받아서 error를 구하고, 새로운 weight를 구하는데 필요한 clock은 error를 구하는데 194 clock cycle, 새로운 weight를 계산하는데 192 clock cycle이 필요하다.

Adaptive filter의 특성상 처음 입력을 받고 새로운 weight를 계산해서 memory에 저장하는 동작을 하는 처음 194 clock cycle 후의 첫 번째 출력은 실제로 원하는 동작과는 다른 의미 없는 값이다. 이것은 최초 reset 후 동작 때 memory에 저장된 Weight가 0으로 reset되기 때문이다. 그러므로 위의 Timing Chart에서 386 clock cycle에 출력되는 두 번째 출력이 Designed response에 대해서 echo가 제거되기 시작하는 출력이 된다.

4-2. Echo Canceller의 설계

전체 구조는 control block, input memory, weight memory, 각 memory의 address generator, MAC (multiplier accumulator), subtractor, adder, mux, barrel shifter, decoder로 구성되어 있다.

그림 3은 Altera사의 MAX+PLUSII에서 각 블록을 코딩한 뒤 schimetic editor를 이용해서 합성한 echo canceller의 전체 회로도이다.

모든 블록은 동작속도를 높이기 위해서 파이프라인 (pipeline) 구조를 사용하였다.

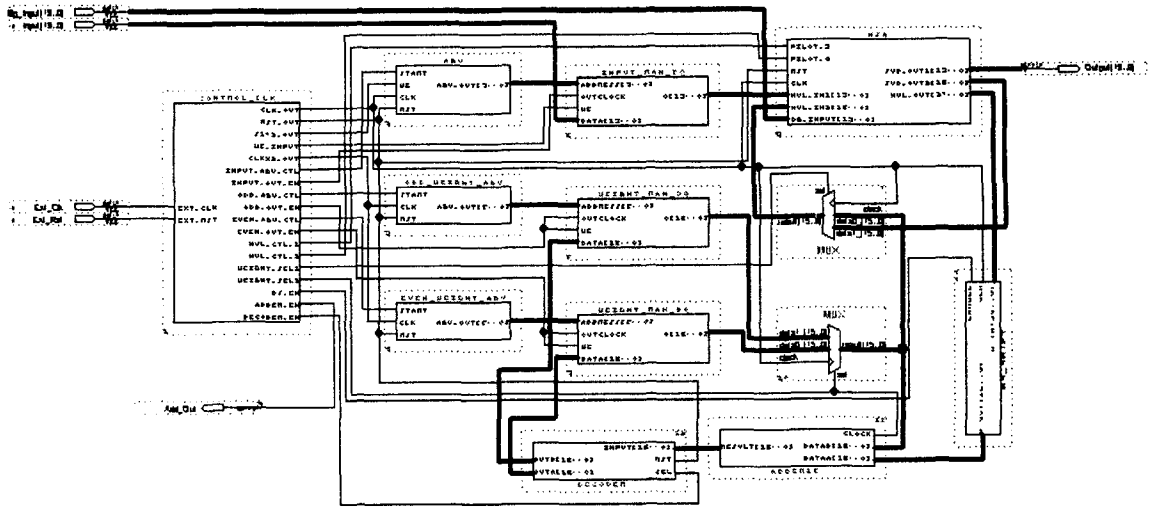

그림 3. Echo Canceller의 전체 회로도

V. Simulation

그림 4는 echo canceller의 전체 simulation 출력 파형을 보여주고 있다. 입력값(Input)은 계속 증가하도록 하였고, designed response(DG_Input)는 "0001"에서 "0111"로 바꾸었을 때, 출력값은 "0001"→"0000"→"0110"→"010F"로 변하면서 "010F"로 수렴하는 것을 확인할 수 있다.

Altera사의 MAX+PLUS II에서 EPF10K50BC356-3 Device를 채택하였을 때, 1141개의 logic cell과 7680 bit의 memory를 사용하였다. 동작 주파수는 50MHz를 사용하였다.

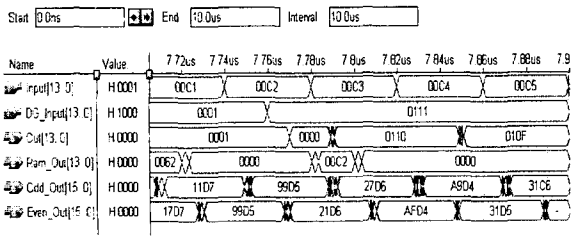

그림 4. Echo Canceller의 전체 Simulation

VI. 결 론

ITU에서 규격으로 정하는 모뎀에서 사용할 수 있는 192-tap echo canceller를 설계하였다. 동작 속도를 향상시키기 위해서 파이프라인구조를 사용하여 각 블록이 한 클럭에 출력값을 출력할 수 있도록 하였다.

Weight memory을 odd와 even으로 나누어 odd weight memory이 reading을 하고 있으면, 이전 clock cycle에서 연산이 끝난 값을 even weight memory에 writing을 함으로써 한 cycle에 두 가지 일을 동시에 처리한다. 이렇게 해서 clock cycle의 수를 줄일 수 있도록 하였다. Simulation에 사용된 동작 주파수는 50MHz이고, 이 경우 초당 약 26K개의 sample을 처리 할 수 있다.

참고문헌

[1] ITU-T V.32(International Telecommunication Unit Telecommunication Sector), ITU, 1993.

[2] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice Hall, 1985.

[3] HAYKIN, *Adaptive Filter Theory*, 3rd ed, Prentice Hall, 1996.

[4] J. M. Cioffi "A fast echo canceller initialization method for the CCITT V.32 modem," *IEEE Trans. Commun.*, vol. 38, no. 5, pp. 629-638, May. 1990.

[5] A. Dembo and J. Salz, "On the leat squares tap adjustment algorithm in adaptive digital echo cancellers," *IEEE Trans. Comm*, vol. 38, no. 5, pp. 622-628, May. 1990.

[6] V. Friedman, J. M. Khoury, M. heobald, and V. P. Gopal, "The implementation of digital echo cancellation in codecs," *IEEE J. Solid-State Circuits*, vol. 25, no. 4, pp. 979-986, Aug. 1990.