임베디드 응용을 위한 플래쉬 메모리와 하드디스크 파일 시스템의 성능 평가

김아람° 이인환
한양대학교 전자컴퓨터통신공학과
alkim°@csl.hanyang.ac.kr , ihlee@hanyang.ac.kr

Performance Evaluation of Flash Memory and Hard-Disk File Systems for Embedded Applications

A-Lam Kim° Inhwan Lee
Dept. of Electronics and Computer Engineering, Hanyang University

요 약

현재 임베디드 환경이 대두되면서 저장 매체로 플래쉬 메모리가 하드디스크를 대체하여 각광을 받고 있다. 이는 휴대폰과 같은 임베디드 환경의 이동성과 관련하여 플래쉬 메모리의 여러 물리적인 특징이 하드디스크보다 이런 환경에 적합하기 때문이다. 본 논문에서는 이런 부분은 배제하고 성능 측면만을 고려하여 하드디스크와 플래쉬 메모리를 비교해 보았다. 측정을 위해 2개의 보드를 사용하였다. 보드 1에서는 FAT 파일시스템 하드디스크와 FAT 플래쉬 메모리로 저장 매체에 따른 성능 측정을 위해 환경을 구축하였다. 보드 2는 FAT 플래쉬 메모리와 YAFFS 플래쉬 메모리로 플래쉬 메모리가 기존 파일시스템과 전용 파일시스템에 따라 얼마만큼의 성능 차이를 내는지 알아보기 위해 환경을 구축하였다.

1. 서 론

현재의 컴퓨터 환경에서 저장 매체는 아주 큰 비중을 차지한다. 그동안 많은 저장 매체가 있었지만 그중에서도 하드디스크[1]는 가장 널리 쓰이는 저장 매체로 자리 잡고 있다. 따라서 저장 매체의 파일을 관리하는 파일 시스템 역시 하드디스크에 기반을 두고 발전 되어 왔다. 오늘날의 하드디스크는 그 용량이 100GB를 넘어서게 되었으며, 속도 또한 예전에 비해 많이 개선이 되었다. 그러나 하드디스크는 물리적으로 정보를 저장하기 때문에 충격에 약하며, 속도가 개선 되었다 하더라도 컴퓨터 내부의 다른 장치들에 비해서는 아직 느린 장치에 속한다.

이에 반해 전기적인 저장 방식을 가지고 있는 플래쉬 메모리[2]는 비휘발성 메모리의 일종으로 그 속도가 하드디스크에 비해 빠르며, 전기적으로 정보를 저장하므로 충격에 있어서도 하드디스크보다 안전하다. 그러나 이런 플래쉬 메모리는 쓰기 횟수에 제한을 가짐으로 장치의 수명이 정해져 있으며 하드디스크에 비해 용량이 아주 작고 가격이 비싸다는 단점이 있다. 또한 플래쉬 메모리는 하드디스크처럼 정보가 있던 자리에 덮어 쓸 수 없고 반드시 정보를 지운 후에 그 자리에 정보를 저장할 수 있다.

이런 매체 특성을 고려하여 파일시스템도 이에 따른 여러 형태와 기능을 가지고 발전해 왔다. 그러나 대부분의 파일시스템이 하드디스크를 중심으로 발전해 왔으며 플래쉬 메모리에 대한 파일시스템 개발은 최근에 이루어 졌다. 따라서 플래쉬 메모리에서도 기존의 하드디스크의 파일시스템을 쓸 수 있도록 FTL(Flash Translation Layer)[3]을 이용하는 방법을 쓰기도 했으며 별도의 H/W로 플래쉬 메모리를 하드디스크처럼 보이게 하는 기술을 이용하기도 했다. 현재는 플래쉬 메모리 전용 파일시스템도 많이 개발이 되었긴 하지만 플래쉬 전용 파일시스템의 경우 다른 파일시스템과의 호환성에 문제를 일으킬수 있으므로 호환성에 중점을 둔 경우 전자와 같은 방법들을 아직 쓰고 있다.

위의 내용을 기반으로 저장 매체와 파일 시스템의 선택에 대해서 언급하자면, 임베디드 환경의 특성을 고려할 때 플래쉬 메모리를 사용하고, 호환성 측면을 생각해 볼 때 FTL이나 혹은 별도의 H/W를 쓸 수 있다고 하겠다. 휴대폰의 예를 들면, 윈도우 환경인 PC와 정보를 주고 받기 위해서는 FTL이나 H/W적인 장치가 없이 전용 파일시스템을 쓴다면 플래쉬 메모리 내의 파일이 인식 되지 않을 것이다.

본 논문에서는 위에서 언급한 저장 매체의 물리적인 특성과 호환성이 아닌 성능만을 기준으로 했을 때 어떤 저장 매체에 어떤 파일시스템을 쓰는 것이 좋은지에 대한 성능 측정 자료를 위해 측정 환경을 구축하였다. 측정 대상은 2가지를 선택하였다. 첫째는 저장 매체들 간의 성능을 비교해 보는 것이고 둘째는 같은 저장 매체에서 파일시스템의 영향을 알아보는 것이다. 측정을 위해 보드 2개를 이용하였다. 보드 1은 하드디스크와 플

표 1 IDE 인터페이스 핀 구성

Pin	Signal	IO	Name	Pin	Signal	IO	Name
1	nRESET	I	RESET	2	GND		GROUND
3	DD7	I/O	DATA BUS BIT 7	4	DD8	I/O	DATA BUS BIT 8
5	DD6	I/O	DATA BUS BIT 6	6	DD9	I/O	DATA BUS BIT 9
7	DD5	I/O	DATA BUS BIT 5	8	DD10	I/O	DATA BUS BIT 10
9	DD4	I/O	DATA BUS BIT 4	10	DD11	I/O	DATA BUS BIT 11
11	DD3	I/O	DATA BUS BIT 3	12	DD12	I/O	DATA BUS BIT 12
13	DD2	I/O	DATA BUS BIT 2	13	DD13	I/O	DATA BUS BIT 13
15	DD1	I/O	DATA BUS BIT 1	16	DD14	I/O	DATA BUS BIT 14
17	DD0	I/O	DATA BUS BIT 0	18	DD15	I/O	DATA BUS BIT 15
19	GND		GROUND	20	N.C.		Not Connected
21	DMARQ	O	DMA REQUEST	22	GND		GROUND
23	nDIOW	I	I/O WRITE	24	GND		GROUND
25	nDIOR	I	I/O READ	26	GND		GROUND
27	IORDY	O	I/O CH. READY	28	CSEL		CABLE SELECT
29	nDMACK	I	DMA ACK.	30	GND		GROUND
31	INTRQ	O	INT. REQUEST	32	IOCS16	O	16 BIT I/O
33	DA1	I	ADDRESS BIT 1	34	PDIAG		PASSED DIAG.
35	DA0	I	ADDRESS BIT 0	36	DA2	I	ADDRESS BIT 2
37	nCS0	I	CHIP SELECT 0	38	nCS1	I	CHIP SELECT 1
39	nDASP	O	DRIVE ACTIVE	40	GND		GROUND

래쉬 메모리간의 저장 매체 성능을 비교해 보기 위해 사용하였으며, 파일시스템은 FAT를 선택하였다. 플래쉬 메모리의 경우 FAT 파일시스템을 쓰기위해 CF-카드 NAND 플래쉬 메모리를 사용하였다. 보드 2는 플래쉬 메모리에서 파일시스템에 따른 성능을 알아보기 위해 사용하였으며, 보드2에 내장된 플래쉬 메모리와 CF-카드 NAND 플래쉬 메모리를 사용하였다. 내장된 플래쉬 메모리는 플래쉬 전용 파일시스템인 YAFFS[4,5]를 적용하였으며, CF-카드는 FAT 파일시스템을 적용하였다.

2. 하드디스크 구조와 제어

하드디스크는 정보를 저장하거나 읽기 위해 기본적으로 섹터(sector) 단위를 사용한다. 한 섹터는 보통 512 Byte이며 섹터들이 모여서 트랙(track)을 이루고 트랙이 모여서 하나의 플래터(platter)를 이루게 된다. 하드디스크는 섹터 단위로 정보를 저장하지만 OS에 따라서는 하드디스크의 대역폭을 효율적으로 쓰기 위해 몇개 섹터의 묶음인 클러스터나 블록 단위로 정보의 입출력을 관리 하기도 한다. 하드디스크의 정보 저장은 기본적으로 플래터 위에 암(arm)이 직접 움직여서 물리적으로 정보를 저장하게 된다.

초기 하드디스크는 버스 컨트롤러가 별도로 있었으나 이는 하드디스크 교체시 호환성에 문제를 유발할수 있기 때문에 IDE 표준[6] 인터페이스(ATA)가 나오면서 하드디스크 내부로 들어가게 되었다. IDE는 보통 IDE 버스로 명명되고 있으나 보는 관점에 따라서는 단지 하드디스크에 달린 버스를 제어하는 전자 부품들의 모임으로 볼 수 있다.

표 1은 40핀 IDE 인터페이스의 핀 설명도이며, 실험 환경을 위해 사용한 44핀 IDE 또한 1-40번 핀이 이와 동일하다. 나머지 4핀은 41번 5V(logic), 42번 5V(motor)로 내부 회로와 하드디스크 모터를 위한 전압이며, 43번은 접지, 44번은 TYPE-(0-ATA)로 할당이 되어 있다. 하나의 IDE에는 두개의 하드디스크가 달릴 수 있는데, 두개의 칩셀렉트(nCS0, nCS1)로 이를 선택

표 2 컨트롤 블록 어드레스

어드레스					기능	
/CS0	/CS1	A0	A1	A2	READ	WRITE
0	1	1	1	0	Alternated Status	Device Control
0	1	1	1	1	Drive Address	사용 안 함

표 3 커맨드 블록 레지스터

어드레스					기능	
/CS0	/CS1	A0	A1	A2	READ	WRITE
1	0	0	0	0	Data	Data
1	0	0	0	1	Error	Feature
1	0	0	1	0	Sector Count	Sector Count
1	0	0	1	1	Sector Number (LBA bit 0-7)	Sector Number (LBA bit 0-7)
1	0	1	0	0	Cylinder Low (LBA bit 8-15)	Cylinder Low (LBA bit 8-15)
1	0	1	0	1	Cylinder High (LBA bit 16-23)	Cylinder High (LBA bit 16-23)
1	0	1	1	0	Drive/Head (LBA bit 24-27)	Drive/Head (LBA bit 24-27)
1	0	1	1	1	Status	Command

할 수 있도록 되어 있고, 16비트 데이터 전송, 3개의 어드레스 라인을 가지고 있다. 또한 하드디스크는 인터럽트 방식(31번 핀)을 사용하며, 리눅스를 쓸 경우 PIO모드를 기본으로 사용하기 때문에 DMA핀은 연결하지 않는다.

IDE 하드디스크는 표 2의 컨트롤 블록 어드레스와 표 3의 8개 레지스터들에 제어 명령을 씀으로 동작을 하게 되는데, 같은 레지스터지만 읽기와 쓰기에 따라서 기능이 달라진다는 점에 주의를 해야한다. CS0와 CS1은 표 1의 칩셀렉트를 말하며, A0 ~ A2는 표 1의 어드레스(DA0 ~ DA2)를 뜻한다. 즉 칩셀렉트와 어드레스 라인으로 하드디스크를 제어하는 레지스터중 하나를 선택하게 되는 것이다.

3. NAND 플래쉬 메모리 구조와 제어

NAND 플래쉬 메모리는 바이트(byte)단위의 접근을 허용하지 않고 항상 페이지(page)단위로 정보를 저장하고 읽을 수 있도록 구성이 되어 있다. 그러나 기종에 따라서는 페이지를 반으로 나누어 앞의 반 페이지와 뒤의 반 페이지를 각각 읽을 수 있는 것도 있다. 페이지 내부에는 정보의 저장 영역이외에 페이지의 상태값을 저장할 수 있는 스페어 영역(spare area)이 있어서 해당 페이지의 ECC, bad block 정보와 같은 값들이 저장 된다. 또한 정보를 지우기 위해서는 페이지들의 묶음인 블록(block)단위로 수행을 하게 된다. 페이지와 스페어 영역, 블록의 크기는 전체 플래쉬 메모리의 용량에 따라 달라

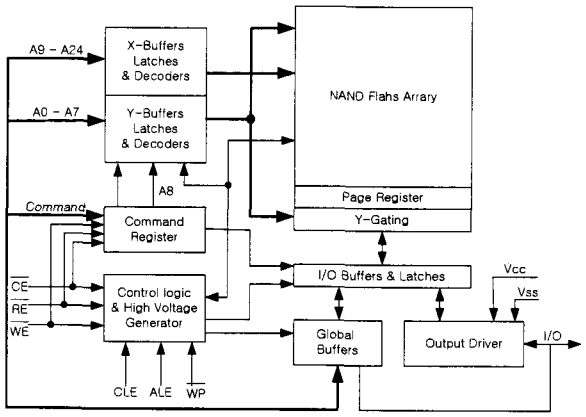

그림 1 NAND 플래쉬 메모리 블록도[2]

표 4 NAND 플래쉬 제어 핀

Pin Name	Pin Function	
I/O	Data inputs/outputs	8, 16비트 데이타 입출력
CLE	Command Latch Enable	명령 레지스터 선택
ALE	Address Latch Enable	주소 레지스터 선택
$\overline{CE}$	Chip Enable	플래쉬 메모리 선택
$\overline{RE}$	Read Enable	I/O의 데이터 읽기
$\overline{WE}$	Write Enable	I/O의 데이터 쓰기
$\overline{WP}$	Write Protect	쓰기 방지

표 5 플래쉬 제어 명령어

Function	1st Cycle	2nd Cycle
Read 1	00h/01h(1)	-
Read 2	50h	-
Read ID	90h	-
Reset	FFh	-
Page Program	80h	10h
Copy-Back Program	00h	8Ah
Block Erase	60h	D0h
Read Status	70h	-

지지만 정보의 읽기, 쓰기, 삭제는 항상 이들 단위로 이루어지며 커맨드 레지스터(그림 1) 명령으로 플래쉬 메모리를 제어한다. Y-Buffer(A0-A7)는 플래쉬 메모리의 행에 해당하는 주소를 지정하며, X-Buffer(A9-A24)는 열(페이지)에 해당하는 주소를 지정한다.

플래쉬 메모리의 제어는 기본적으로 표 4의 핀 선택에 의해 이루어지며, 제어 명령어는 표 5와 같다. 명령어 마다 필요한 싸이클이 다르며, 2싸이클이 필요한 경우 첫번째 싸이클에는 명령을 수행하기 위한 설정을 하고 두번째 싸이클에 실행을 하게 된다. 표 4, 5의 핀과

표 6 실험 환경

공통	Linux 2.6.13
	Arm-linux-gcc 3.3.2
	Intel® Pentium4 CPU 2.4GHz
	Bonnie 벤치마크
보드 1	PXA270 (520MHz)
	64MB MDOC flash
	128MB SDRAM
	SanDisk 2GB CompactFlash
	2.5" 44핀 하드디스크(4GB)
보드 2	ARM2440 (400MHz)
	1MB Nor Flash
	64MB Nand Flash
	64MB SDRAM
	SanDisk 2GB CompactFlash

명령어들은 호스트에 플래쉬 메모리를 직접 이식할 때 필요하다. YAFFS 플래쉬 파일시스템의 경우는 MTD에 제어 과정이 모두 포함되어 있으므로 YAFFS를 사용할 시 는 YAFFS 인터페이스를 사용하여 리눅스와 파일시스템을 연결하게 된다.

4. 실험 환경과 구현

4.1 실험 환경

실험 환경(표 6) 구축을 위해 2개의 보드를 사용하였다. 보드 1에 하드디스크는 44핀 IDE 인터페이스로 연결하였으며, CF-카드 NAND 플래쉬 메모리는 PCMCIA에 연결하였다. 또한 보드의 운영체제인 linux 2.6.13에서 하드디스크와 CF-카드 플래쉬 메모리가 작동 되도록 디바이스 드라이버를 작성하여 설치하였다. 초기에 파일시스템은 두 매체 모두 FAT16을 쓸 계획이였으나, FAT16의 경우 최대 2GB까지 밖에 인식하지 못하는 관계로 하드디스크(4GB)의 경우 Ext2 파일시스템을 이용하였다. 보드 2는 내장되어 있는 64 MB NAND 플래쉬 메모리와 CF-카드 플래쉬 메모리를 사용하였다. 내장 되어있는 플래쉬 메모리에 YAFFS 파일시스템을 적용하였으며, CF-카드 플래쉬 메모리에는 FAT16 파일시스템을 적용하였다.

파일시스템 벤치마크는 Bonnie[7]를 사용하였다. Bonnie 벤치마크는 putc()와 getc()를 이용하여 문자 단위의 입출력 속도를 측정하고, read(), write()를 이용하여 블록 단위의 입출력 속도 또한 측정하는 간단한 벤치마크 프로그램이다. Bonnie의 결과표에 나오는 CPU 퍼센트는 위에 언급한 함수를 수행하는 시간과 이에 동반된 OS에서 파일 영역 할당 시간을 읽거나 쓰는 총 시간에 대비해서 백분율로 표시한 것이다.

표 7 측정 결과 1

	Sequential Output (write)					
	Per Character		Block		Rewrite	
	K / sec	% CPU	K / sec	% CPU	K / sec	% CPU
flash	1364	0	3154	0	1078400404	0
HDD	1887	0	12783	0	1079498192	0
	Sequential Input (read)				Random	
	Per Character		Block		Seek	
	K / sec	% CPU	K / sec	% CPU	K / sec	% CPU
flash	1043	0	2288	0	97.2	51.8
HDD	1863	0	72753	0	99.5	100.2

표 8 측정 결과 2

	Sequential Output (write)					
	Per Character		Block		Rewrite	
	K / sec	% CPU	K / sec	% CPU	K / sec	% CPU
YAFFS	638	100	834	100	681	99.9
FAT	1628	81.7	2089	12.1	461	4.8
	Sequential Input (read)				Random	
	Per Character		Block		Seek	
	K / sec	% CPU	K / sec	% CPU	K / sec	% CPU
YAFFS	1843	100	45040	100	384.3	100
FAT	1725	96.7	43408	99.1	2007.2	93.1

4.2 구 현

Linux 2.6.13을 쓰는 보드 1에 IDE 하드디스크를 적용하기 위해서는 linux IDE 디바이스 드라이버를 보드 1에 맞게 변경해 주는 작업이 필요하다. Linux 소스 코드에서 IDE 관련 소스는 /drivers/ide/에 위치해 있다. 또한 /drivers/ide/arm 디렉토리는 IDE관련 디렉토리와 의존성을 갖는 위치로, 보드에 맞게 IDE 디바이스 드라이버를 포팅하기 위해서 이 부분에 코드를 작성하여 부팅시에 모듈로 포함되도록 하였다.

IDE를 인식시키기 위해서는 첫째로 표 3의 IDE 관련 레지스터 8개를 주 메모리에 매핑을 해야하며, 둘째로 하드디스크 인터럽트를 설정해 주어야 한다. Linux의 경우 PIO모드를 기본적으로 사용하기 때문에 DMA 연결은 하지 않았다.

보드 1의 메모리는 0x10000000의 물리 주소가 칩셀렉트 0번, 0x10000010이 칩셀렉트 1번으로 할당되어 있으며, MMU를 사용하기 때문에 이 주소중 칩셀렉트 0번을 가상 메모리 0xf1000000에 연결시켜 주었다. 이 같은 메모리 매핑과 관련된 사항은 /arch/arm/mach-pxa/generic.c에 map_desc{}에서 설정하였다.

가상 메모리 설정후 이 주소값을 표 3의 8개 레지스터에 할당했으며, GPIO(General Purpose IO)선들중에 하나를 인터럽트 선으로 사용했다. 보드 1은 하드웨어적으로 A0-A2(표 3)중 가장 낮은 비트인 A0가 프로세서의 주소선 26개(MA0 - 25)중에 1번(MA1)에 연결이 되어 있다. 따라서 레지스터 주소값 증가는 16비트 단위로 이루어 진다. 인터럽트의 경우 GPIO중 인터럽트 용도로 설계된 별도의 입력선이 있어서 이를 사용했다.

만들어진 모듈은 커널 소스에 첨부하여 부팅 시에 동작 되도록 하였다. 커널 소스에 포함하는 방법은 Kbuild 수정으로 간단하게 포함 시킬 수 있다. YAFFS와 관련된 소스와 포팅은 모두 Aleph One Company의 YAFFS 사이트[4]를 참조하였다.

5. 실험 결과 및 분석

측정 보드1에서 bonnie 벤치마크를 적용하여 100MB 입력을 줬을 때 표 7과 같은 결과가 나왔다. 하드디스크의 용량 때문에 같은 파일시스템을 적용하지는 못했지만, 측정 결과를 보면 기존 하드디스크 파일시스템을 적용했을 때 블록 단위의 읽기와 쓰기 부분에서 하드디스크가 CF-카드 플래쉬 메모리에 비해서 약3 ~ 4배정도 빠른 속도를 내고 있음을 알 수 있다. 한가지 더 주목할 부분은 랜덤하게 정보를 찾는 부분에서 기존의 파일시스템을 그대로 적용했지만 플래쉬 메모리와 하드디스크간의 속도차가 거의 나지 않는다는 것이다. 이는 하드디스크의 seek time 때문이라 생각한다. 그 외 문자 단위의 입출력 속도 부분은 크지 않은 차이지만 하드디스크가 플래쉬 메모리에 비해서 조금 더 빠른 성능을 냄을 알 수 있다.

측정 보드 2에서 bonnie 벤치마크를 적용하여 40MB 입력을 줬을 때 표 8과 같은 결과가 나왔다. 여기서 읽기 부분에서 YAFFS가 문자 단위로 읽는 속도와 블록 단위로 읽는 속도가 모두 FAT 파일 시스템을 사용하는 CF-카드 플래쉬 메모리보다 빠르다는 것을 알 수 있다. 그러나 쓰기 부분에서는 YAFFS와 FAT의 현저한 속도차를 볼 수 있는데, 이는 %CPU가 높다는 것과 관련이 있다. 실험 보드 2에 사용한 NAND 플래쉬 메모리는 총 용량이 64MB로 커널과 다른 프로그램을 제외하고 41MB의 저장 공간이 남아 있었고, CF-카드 플래쉬 메모리는 2GB 영역이 모두 사용 가능한 영역이였다. YAFFS의 경우 정책에 따라 다르긴 하지만 일정 용량 이상 정보가 저장되면 가비지 콜렉션을 수행하게 된다. 따라서 YAFFS 파일시스템 측정 과정에서 가비지 콜렉션이 발생하여 쓰는 속도에 대한 측정 결과가 낮게 나왔다고 생각한다.

6. 결 론

본 논문에서는 플래쉬 메모리에서 파일시스템에 의한 성능을 측정과 하드디스크, 플래쉬 메모리로 저장 매체

에 따른 성능 측정을 2개의 보드를 사용하여 해 보았다. 이와 같은 환경외에 FTL을 사용한 플래쉬 메모리에 대해서도 측정을 하고자 하였으나, 아직 FTL을 구현하지 못하여 측정하지 못하였다.

플래쉬 메모리에서 기존의 파일 시스템을 쓰는 방법에 소프트웨어적(FTL)인 방법과 하드웨어적인 방법이 있다고 하였는데, FTL을 구현하여 하드웨어로 구현된 방법과 성능을 비교해 본다면 cost/performance 차원에서 좋은 자료로 사용될 수 있으리라 생각하며 이에 대한 측정 환경을 차후에 구현 해 보고자 한다.

또한 벤치마크도 다중 프로세서를 지원하는 프로그램으로 측정을 해 보고자 한다. Bonnie 벤치마크의 경우 하나의 프로세서에서 하나의 파일을 쓰고 읽음으로 측정을 하지만, 현재의 임베디드 환경이 다중 프로세서를 지원하는 추세이기 때문에 이와 같은 환경에서 측정해 볼 수 있는 벤치마크를 사용하여 측정해 보는 것 역시 의미가 있으리라 생각한다.

7. 참고 문헌

[1] Storage Review at http://www.storagereview.com/guide2000/ref/hdd/index.html

[2] Samsung Flash memory at http://www.samsung.com/Products/Semiconductor/Flash/

[3] Understanding the Flash Translation Layer (FTL) Specification, Intel® company

[4] Yet Another Flash Filing System(YAFFS) at http://www.yaffs.net/, Aleph One Company at http://www.aleph1.co.uk/

[5] Mendel Rosenblum and John K. Ousterhout "The Design and Implementation of a Log-Structured File System" Proc. Of the 13th ACM Symposium on Operating Systems Principles and the February 1992 ACM Transactions on Computer Systems.

[6] http://www.ele.uri.edu/courses/ele408/s2001/projects/roland_ide/IDE.pdf or Technical Committee T13 AT Attachment at http://www.t13.org

[7] Bonnie at http://www.textuality.com/bonnie/