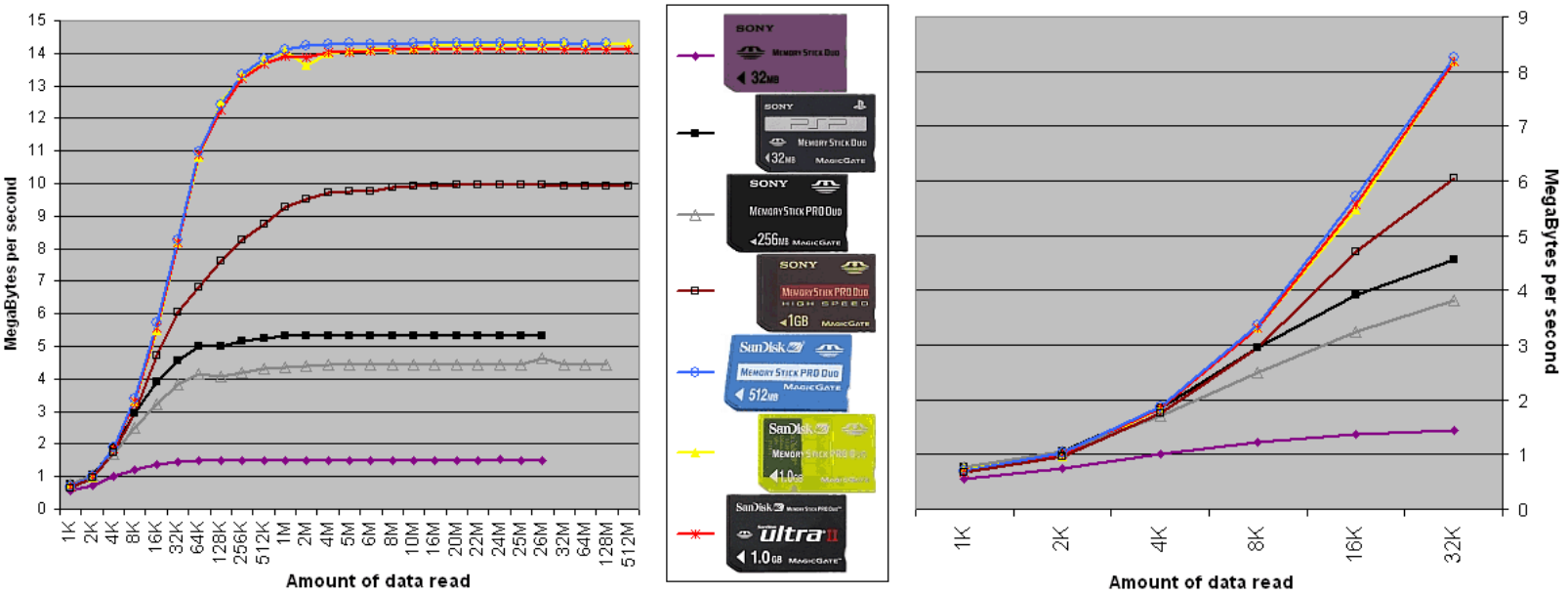
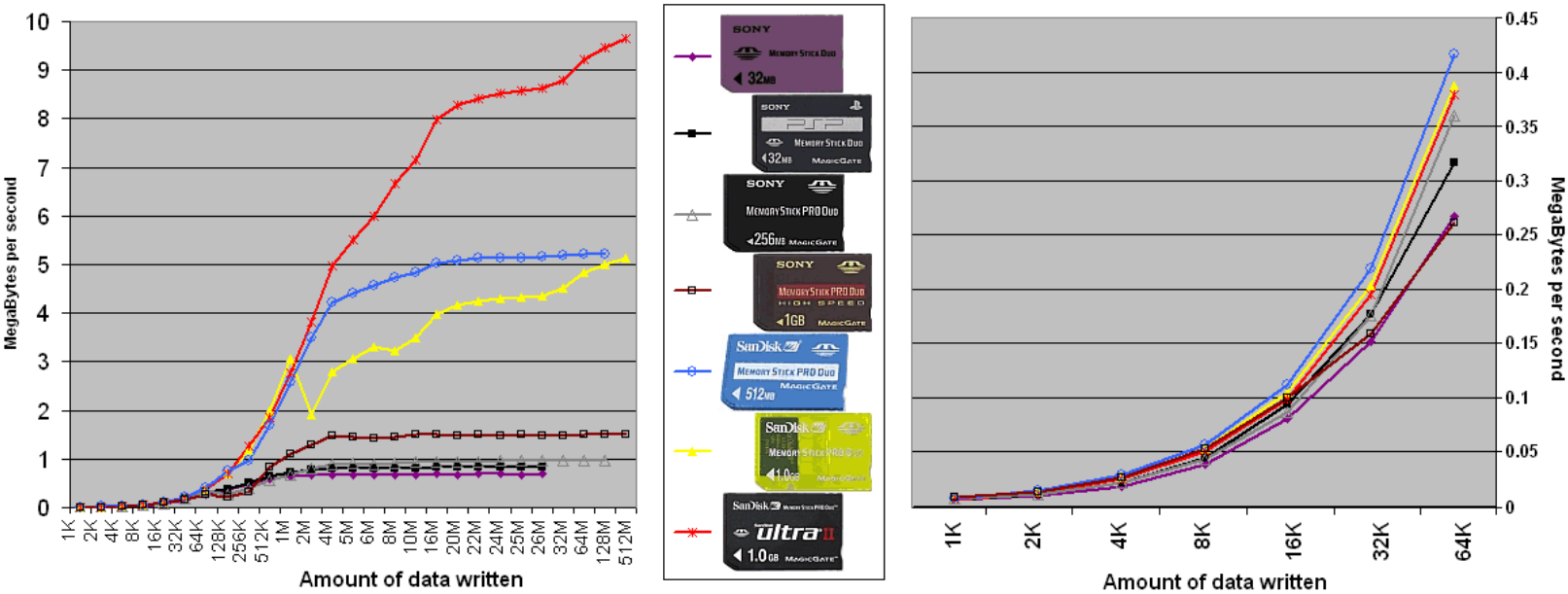
Memory Stick Duo - Read & Write Speed Comparisons

The reason for these measurements were the scarce availability of transfer speeds for the Memory Stick Duos. Since the primary interest was in how certain Memory Sticks behave with the PSP, all the measurements were done with the PSP, thus these results do not necessarily reflect the true transfer speeds of the Memory Stick, but the read and write speeds to and from the PSP. Nevertheless, interesting to note, that the Sony published 80Mbps (10MBps) value for the High Speed Memory Stick PRO Duo™ is quite accurately reproduced when reading 16MB+ of data.
The source data from which the graphs were derived from are listed after the figures and the code which was used for this measurements is attached after the raw data. "MB" is for MegaByte and "Mb" is for MegaBit.
Internal memory allocation peaked at 16MB, thus every transfer bigger than 16MB was in chunks.
Every read and write test run at least 100 times.
Note, that there is no information in this document about any other qualities of these products such as Power Consumption, Operating Life, MTBF etc.

Memory Stick Duo Read Speeds (measurements with PSP)



Memory Stick Duo Write Speeds (measurements with PSP)



(Over 500 tests were conducted on that 2MB anomaly for the SanDisk Yellow, but it had a 100% reproduction rate)

The Measured Raw Data :

	Sony Duo 32MB (Purple)				Sony PSP Duo S32				SanDisk 1GB Yellow Gaming				SanDisk 1GB Ultra II (Black)			
	Read	Write	R-MB/s	W-MB/s	Read	Write	R-MB/s	W-MB/s	Read	Write	R-MB/s	W-MB/s	Read	Write	R-MB/s	W-MB/s
1K	0.0018	0.1781	0.5580	0.0055	0.0013	0.1538	0.7750	0.0064	0.0013	0.1469	0.7512	0.0066	0.0015	0.1557	0.6689	0.0063
2K	0.0026	0.2133	0.7398	0.0092	0.0018	0.1845	1.0673	0.0106	0.0020	0.1473	0.9621	0.0133	0.0020	0.1567	1.0016	0.0125
4K	0.0039	0.2103	0.9990	0.0186	0.0021	0.1823	1.8426	0.0214	0.0021	0.1476	1.8426	0.0265	0.0021	0.1561	1.8426	0.0250
8K	0.0064	0.2049	1.2150	0.0381	0.0026	0.1773	2.9593	0.0441	0.0024	0.1479	3.3104	0.0528	0.0024	0.1562	3.3104	0.0500
16K	0.0115	0.1936	1.3622	0.0807	0.0040	0.1663	3.9160	0.0940	0.0029	0.1501	5.4825	0.1041	0.0028	0.1575	5.5605	0.0992
32K	0.0218	0.2068	1.4328	0.1511	0.0068	0.1771	4.5687	0.1765	0.0038	0.1538	8.1806	0.2032	0.0038	0.1599	8.1806	0.1954
64K	0.0422	0.2336	1.4810	0.2676	0.0125	0.1978	5.0201	0.3161	0.0058	0.1614	10.8131	0.3873	0.0057	0.1652	10.8885	0.3784
128K	0.0840	0.3234	1.4884	0.3865	0.0251	0.3351	4.9860	0.3730	0.0100	0.1778	12.4875	0.7030	0.0102	0.1763	12.2429	0.7091
256K	0.1669	0.5029	1.4976	0.4971	0.0485	0.4835	5.1589	0.5170	0.0188	0.2096	13.2979	1.1925	0.0189	0.1995	13.2415	1.2531
512K	0.3333	0.8621	1.5002	0.5800	0.0953	0.7795	5.2493	0.6414	0.0362	0.2507	13.8083	1.9948	0.0366	0.2686	13.6761	1.8615
1M	0.6657	1.5812	1.5023	0.6324	0.1884	1.3727	5.3073	0.7285	0.0710	0.3277	14.0924	3.0514	0.0719	0.3617	13.9179	2.7647
2M	1.3296	3.0195	1.5042	0.6624	0.3760	2.5539	5.3194	0.7831	0.1469	1.0518	13.6156	1.9014	0.1443	0.5237	13.8648	3.8193
4M	2.6592	5.9681	1.5042	0.6702	0.7500	4.9199	5.3333	0.8130	0.2861	1.4363	13.9797	2.7850	0.2853	0.8060	14.0223	4.9625
5M	3.3238	7.4076	1.5043	0.6750	0.9380	6.1758	5.3305	0.8096	0.3557	1.6348	14.0556	3.0585	0.3559	0.9080	14.0493	5.5069
6M	3.9887	8.8451	1.5043	0.6783	1.1250	7.3652	5.3333	0.8146	0.4256	1.8215	14.0968	3.2940	0.4262	1.0008	14.0795	5.9951
8M	5.3164	11.7969	1.5048	0.6781	1.4998	9.7590	5.3342	0.8198	0.5666	2.4777	14.1191	3.2288	0.5672	1.2020	14.1044	6.6556
10M	6.6471	14.6816	1.5044	0.6811	1.8741	12.2220	5.3359	0.8182	0.7061	2.8713	14.1633	3.4827	0.7083	1.4009	14.1177	7.1384
16M	10.6328	23.3262	1.5048	0.6859	2.9985	19.4082	5.3359	0.8244	1.1250	4.0160	14.2222	3.9841	1.1326	2.0045	14.1264	7.9822
20M	13.4526	29.2477	1.4867	0.6838	3.7492	24.1889	5.3344	0.8268	1.4044	4.7869	14.2409	4.1781	1.4173	2.4148	14.1116	8.2821
22M	14.6557	32.0169	1.5011	0.6871	4.1231	26.6585	5.3359	0.8253	1.5452	5.1751	14.2379	4.2512	1.5583	2.6113	14.1182	8.4248
24M	15.4863	34.5783	1.5498	0.6941	4.4969	29.0117	5.3370	0.8273	1.6860	5.5698	14.2345	4.3089	1.6973	2.8122	14.1404	8.5342
25M	16.6723	36.4738	1.4995	0.6854	4.6848	30.3706	5.3364	0.8232	1.7552	5.7723	14.2433	4.3310	1.7699	2.9109	14.1252	8.5883
26M	17.4581	37.6083	1.4893	0.6913	4.8734	31.4023	5.3351	0.8280	1.8254	5.9619	14.2435	4.3610	1.8393	3.0127	14.1361	8.6301
32M									2.2445	7.0996	14.2573	4.5073	2.2671	3.6431	14.1148	8.7836
64M									4.4821	13.2172	14.2790	4.8422	4.5343	6.9359	14.1146	9.2273
128M									8.9594	25.6038	14.2867	4.9993	9.0644	13.5165	14.1212	9.4699
512M									35.8019	99.9596	14.3009	5.1221	36.2853	53.0371	14.1104	9.6536

	Sony Pro Duo 256MB (Black)				SanDisk Pro Duo 512MB (Blue)				Sony Pro Duo High Speed 1GB			
	Read	Write	R-MB/s	W-MB/s	Read	Write	R-MB/s	W-MB/s	Read	Write	R-MB/s	W-MB/s
1K	0.0013	0.1378	0.7750	0.0071	0.0014	0.1378	0.6877	0.0071	0.0015	0.1156	0.6643	0.0084
2K	0.0018	0.1817	1.0673	0.0108	0.0019	0.1361	1.0226	0.0143	0.0020	0.1514	0.9621	0.0129
4K	0.0023	0.1814	1.7133	0.0215	0.0021	0.1355	1.8780	0.0288	0.0022	0.1475	1.7439	0.0265
8K	0.0031	0.1812	2.4960	0.0431	0.0023	0.1372	3.3675	0.0569	0.0027	0.1474	2.9481	0.0530
16K	0.0048	0.1800	3.2283	0.0868	0.0027	0.1389	5.7234	0.1125	0.0033	0.1571	4.7063	0.0995
32K	0.0082	0.1774	3.8203	0.1761	0.0038	0.1428	8.2672	0.2189	0.0052	0.1964	6.0445	0.1591
64K	0.0151	0.1738	4.1391	0.3595	0.0057	0.1501	10.9649	0.4165	0.0092	0.2391	6.8157	0.2614
128K	0.0308	0.4523	4.0532	0.2764	0.0101	0.1669	12.4378	0.7489	0.0164	0.6040	7.6173	0.2069
256K	0.0597	0.6253	4.1911	0.3998	0.0187	0.2604	13.3547	0.9601	0.0302	0.7877	8.2727	0.3174
512K	0.1162	0.8760	4.3026	0.5708	0.0361	0.2930	13.8389	1.7067	0.0572	0.6064	8.7353	0.8245
1M	0.2291	1.4729	4.3655	0.6790	0.0709	0.3871	14.1004	2.5834	0.1079	0.9075	9.2670	1.1019
2M	0.4540	2.4688	4.4054	0.8101	0.1405	0.5722	14.2389	3.4953	0.2098	1.5502	9.5329	1.2902
4M	0.9037	4.4785	4.4262	0.8932	0.2803	0.9491	14.2699	4.2143	0.4114	2.7134	9.7225	1.4742
5M	1.1291	5.5313	4.4282	0.9040	0.3494	1.1353	14.3098	4.4043	0.5118	3.4627	9.7704	1.4440
6M	1.3557	6.7383	4.4259	0.8904	0.4207	1.3102	14.2606	4.5796	0.6139	4.1861	9.7733	1.4333
8M	1.8056	8.7344	4.4307	0.9159	0.5599	1.6906	14.2893	4.7320	0.8102	5.5415	9.8737	1.4437
10M	2.2548	10.7422	4.4350	0.9309	0.6992	2.0632	14.3017	4.8470	1.0100	6.6387	9.9010	1.5063
16M	3.6042	16.8262	4.4393	0.9509	1.1187	3.1859	14.3021	5.0222	1.6133	10.6572	9.9173	1.5013
20M	4.5070	21.0332	4.4375	0.9509	1.3982	3.9333	14.3045	5.0848	2.0104	13.5845	9.9484	1.4723
22M	4.9564	23.0566	4.4387	0.9542	1.5384	4.2910	14.3005	5.1270	2.2112	14.6479	9.9492	1.5019
24M	5.4070	25.0898	4.4387	0.9566	1.6780	4.6729	14.3029	5.1361	2.4143	16.1573	9.9410	1.4854
25M	5.6318	26.1016	4.4390	0.9578	1.7475	4.8657	14.3065	5.1380	2.5120	16.6430	9.9523	1.5021
26M	5.6025	26.8058	4.6408	0.9699	1.8180	5.0410	14.3017	5.1577	2.6080	17.6278	9.9693	1.4749
32M	7.2051	33.2490	4.4413	0.9624	2.2368	6.1602	14.3060	5.1947	3.2316	21.6314	9.9021	1.4793
64M	14.4023	66.0273	4.4437	0.9693	4.4780	12.2793	14.2920	5.2120	6.4528	42.6329	9.9181	1.5012
128M	28.8441	132.198	4.4377	0.9682	8.9531	24.5518	14.2967	5.2135	12.9165	85.7036	9.9098	1.4935
512M									51.5907	342.342	9.9243	1.4956

The simplified code snippet for the measurements :

```
#define TESTFILENAME    "MS0:/test.data"

void RunTestSeries(int TransferSize, int NumberOfTests)         // NumberOfTests default was 100
{
        ScreenClear();
        printf ("\nTest file size = %d MB / %d KB\n",(int)(TransferSize>>20),(int)(TransferSize>>10));
        printf("\nWrite : \n");
        ctr=0;
        while(NumberOfTests>ctr)
        {
                a=WriteTest(TransferSize);
                if(a==0)                        // Write success, record result
                {
                        Results[ctr]=MeasuredTime;
                        printf("%1.5f ",Results[ctr]);
                        MesSec+=Results[ctr];
                        ++ctr;
                }
        }
        sum=(MesSec/(float)ctr);        // Tried omitting min and max values, but the measurements were quite consistant
        printf("\nTime = %1.5f",sum);

        printf("\nRead  :\n");
        ctr=0;
        while(NumberOfTests>ctr)
        {
                a=ReadTest(TransferSize);
                if(a==0)                        // Read success, record result
                {
                        Results[ctr]=MeasuredTime;
                        printf("%1.5f ",Results[ctr]);
                        MesSec+=Results[ctr];
                        ++ctr;
                }
        }
        sum=(MesSec/(float)ctr);        // Tried omitting min and max values, but the measurements were quite consistant
        printf("\nTime = %1.5f",sum);

        RemoveFile(TESTFILENAME);
        CleanUpMemory();
}

int WriteTest(int TransferSize)
{
        if(Buffer==NULL) AllocateMemory(MEMSIZE);
        RemoveFileIfExists(TESTFILENAME);
        if((fd=OpenFile(TESTFILENAME, WRONLY|CREAT, 0777))<0) return(1);
        LseekFile(fd,0,SEEK_SET);
        remaining=TransferSize;
        StartTimer();
        while(remaining>0)
        {
                if(remaining>BufferSize)
                        a=WriteToFile(fd,Buffer,BufferSize);
                else
                        a=WriteToFile(fd,Buffer,remaining);
                remaining-=a;
        }
        SyncFile("MS0:");
        CloseFile(fd);
        EndTimer();     // This puts the measured time into "MeasuredTime"
return(0);
}

int ReadTest(int TransferSize)
{
        if((fd=OpenFile(TESTFILENAME, RDONLY|CREAT, 0777))<0) return(1);
        FSize=LseekFile(fd,0,SEEK_END);
        LseekFile(fd,0,SEEK_SET);
        if(FSize!=TransferSize) return(1);      // WriteTest ran before read and left the file
        remaining=FSize;
        StartTimer();
        while(remaining>0)
        {
                if(remaining>BufferSize)
                        a=ReadFromFile(fd,Buffer,BufferSize);   // WriteTest ran before read and took care of memory
                else
                        a=ReadFromFile(fd,Buffer,remaining);
                remaining-=a;
        }
        CloseFile(fd);
        EndTimer();     // This puts the measured time into "MeasuredTime"
return(0);
}

void StartTimer(void)
{
struct timeval tval;

        gettimeofday(&tval,0);          //      tval.tv_usec = 0~999.999
        StartTimeStamp=(float)((float)tval.tv_sec+((float)tval.tv_usec/(float)1000000));
}


void EndTimer(void)
{
struct timeval tval;

        gettimeofday(&tval,0);
        MeasuredTime=(float)((float)tval.tv_sec+((float)tval.tv_usec/(float)1000000))-StartTimeStamp;
}
```