Highspeed MMC-Interface für den Zeddy

Hallo Leute,
nachdem ich nun seit einiger Zeit MEFISDOS mit MMC und Ramdisk einsetze, habe ich mir öfters schon mal Gedanken gemacht, wie man den MMC-Zugriff schneller als bisher vornehmen könnte, da ich die MMC angesichts ihrer Größe gerne anstelle einer "richtigen Festplatte" als primäres Speichermedium einsetzen wollte (siehe dazu auch Artikel im TM 2/2005).

Die bisherige Lösung der Ansteuerung der MMC über eine PIO hatte den Charme, daß beim ZX96 dazu keine neue Hardware (bis auf die Pegelanpassung), sondern "***nur* Software**" (wie einer meiner Ex-Chefs öfters meinte) vonnöten war. Erkauft hat man sich diese unaufwändige Lösung allerdings durch einen relativ langsamen Datentransfer zwischen Zeddy und MMC, da die MMC ja über den SPI-Bus seriell angesteuert wird und der Zeddy (und dessen Software) die Datenwandlung parallel/seriell vornehmen und auch noch die Clock-Signale für die MMC generieren muß. Somit ergibt sich, daß zwischen MMC und Zeddy die Daten mit einer Datenrate von allenfalls einigen 100 KHz übertragen werden können.

Laut Datenblatt kann eine MMC aber mit Clockraten bis 10 MHz betrieben werden, was von einem Zeddy mit 3,25 MHz-Systemtakt durch eine Softwarelösung natürlich nie ausgenützt werden kann.
Es war mir also schnell klar, daß eine "richtig schnelle" MMC (wie sie ja auch in anderen Systemen üblich ist) nur durch entsprechende Hardwareunterstützung am Zeddy möglich sein würde. Dabei wollte ich den Hardware-Aufwand (sinclair-konform) möglichst gering halten, wobei ich "gering" im Vergleich zum Materialaufwand für ein Festplatteninterface sah.

Vorüberlegungen

Da die MMC über den SPI-Bus angesteuert wird, hatte ich mich erst mal nach Host-Controllern für den SPI-Bus umgesehen. Leider scheint es so was nicht zu geben, da die SPI-HOST-Funktionalität meist direkt im Microcontroller implementiert ist und nicht extern (durch weiteren Chip) realisiert werden muß. Leider hat unser Z80 diese Hardware nicht schon drinnen :-((
Also habe ich mich mal nach Schieberegistern umgesehen, damit dem Zeddy schon mal die parallel/seriell-Wandlung der Daten erspart bliebe. Ich habe mich dann für den 74HC299 entschieden, der als "PIPO-Schieberegister" (Parallel In und Parallel Out) serielle Daten senden/empfangen kann und Richtung Z80 parallel geschrieben/gelesen werden kann. Da man durch dessen äußere Beschaltung auch ein Links- bzw. Rechtsschieben der Bits ansteuern kann, ist dieser Chip ziemlich universell einsetzbar.

Nun mußten noch die Clock-Signale zum Schieben der Bits Richtung MMC generiert werden. Die Schwierigkeit liegt darin, daß genau 8 Clock-Signale zum Transfer eines Bytes Richtung MMC notwendig sind, und dann erst mal das nächste Byte von Zeddy bereitgestellt werden muß. Es muß der Zeddy also (parallel) 1 Byte in den PIPO schreiben und danach darf erst die Clock für den Bittransfer zur MMC loslaufen und muß nach 8 Takten stoppen, damit der Zeddy das nächste Byte bereitstellen kann.
Clockgenerator und Zeddy müssen also hart sychronisiert sein, damit der Datentransfer zur MMC klappt.
Erste Überlegungen waren also wieder, den Zeddy den Takt erzeugen zu lassen. Aber statt zweier OUT-Kommandos zum Erzeugen der fallenden und steigenden Flanke der MMC-Clock (OUT CLOCK,0 und dann OUT CLOCK,1) sollte ein Out-Befehl reichen: das dadurch vom Z80 generierte /WR-Signal könnte man als Takt für die MMC nehmen. Es wären dann zum Transfer eines Bytes noch 8 Out-Befehle für die 8 Clock-Takte notwendig.
Also:
OUT PIPO, A (1 Byte in PIPO schreiben)
OUT CLOCK,DUMMY (1. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (2. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (3. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (4. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (5. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (6. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (7. Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCK,DUMMY (8. Clock-Takt für MMC/Schieberegister erzeugen)

Mir erschien das Out-Kommando mit 11 bzw. 12 Z80-Taktzyklen dann aber noch zu lang. Vielleicht könnte man das /M1 oder /RFSH-Signal des Z80 ja als Clock nehmen? Dann hätten z. B .8 NOP (nur je 4 Z80-Takte) für den Transfer eines Bytes gereicht. Allerdings benötigt man dann ein "GATE", mit dem man die permanent vorhandenen /M1-Signale freischalten oder unterdrücken kann. Die Software hätte dann etwa so ausgesehen:
OUT PIPO, A (1 Byte in PIPO schreiben und Gate öffnen)

NOP (1. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
NOP (2. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
NOP (3. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
NOP (4. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
NOP (5. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
NOP (6. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
NOP (7. /M1-Clock-Takt für MMC/Schieberegister erzeugen)
OUT CLOCKGATE,OFF (8. /M1-Clock-Takt, Gate schließen und weitere Takte unterbinden)

Durch weitere Überlegungen bin ich dann beim Systemtakt des Zeddys gelandet. Aber sie sollte man sicherstellen, daß bei der Sequenz
OUT PIPO, A (1 Byte in PIPO schreiben)
OUT CLOCK,AN (Clock Generator einschalten)
OUT CLOCK,AUS (Clock Generator ausschalten)
sicherstellen, daß genau 8 Zeddy-System-Takte als Clock zur MMC gelangen, wenn ein Out-Befehl selbst schon 11 Takte lang ist? Das erschien mir erstmal unmöglich!

Aber genau das ist die Lösung! Allerdings war dazu das intensive Studium der Z80 Timingdiagramme der In- und OUT-Zyklen und die Hinzuziehung zweier Flipflops vonnöten. Und nun sieht die Software zum Senden eines Bytes so aus:
OUT PIPO,A
IN A,PIPO

Das war's!
Und damit wären wir schon mittendrin in der

Schaltungsbeschreibung

Kern ist, wie gesagt, das PIPO-Schieberegister IC4 (74HC299), das vom Z80 parallel geschrieben/gelesen werden kann und das seriellen die Daten zur/von der MMC transferiert.

Beim OUT-Befehl wird das auszusendende Byte ins Schieberegister geschrieben und dort bei der steigenden Flanke des /WR-Signals des Z80 gespeichert. Diese Flanke bringt aber die Gate-Schaltung (IC5: 74HC74) in "Hab-Acht"-Stellung (erstes Flipflop IC5A wird gesetzt). Die nächsten steigende Flanke des Zeddy-Systemtakts öffnet das Gate (2 . Flipflop IC5B wird gesetzt). Der invertierende Ausgang des Flipflops gibt über das ODER-Gatter IC6A den Zeddy-Systmtakt als Clock für PIPO und MMC frei. Jeder weitere Z80-Takt ist nun Clocksignal des Schiebregisters und der MMC!

Während also die Clocksignale munter laufen und der PIPO die seriellen Daten über seinen Ausgang (PIN 17, DIN-MMC) zur MMC schickt, hat der Z80 mit der Decodierung des IN-Befehls begonnen. In dessen Ausführung wird dann irgendwann mal das /RD-Signal aktiviert, an dessen Ende die im Schieberegister liegenden Daten vom Z80 übernommen werden. Aber gleich zu Beginn des /RD-Signals wird das Gate über die Reset-Eingänge der Flipflops asynchron geschlossen. Und zu diesem Zeitpunkt sind genau 8 Takte zur MMC gelangt und genau 1 Byte wurde übertragen! Das war's also (ZILOG sei Dank)!

Die gleiche Befehlssequenz wird zum Einlesen der von der MMC kommenden Daten benutzt. Allerdings muß beim OUT kann ein für die MMC als "NOP"-Befehl geltendes Byte ausgegeben werden: $FF
Also wird mit
LD A,$FF
OUT PIPO,A
IN A,PIPO

der Wert $FF zur MMC geschickt. Gleichzeitig werden die von der MMC (DOUT-MMC) kommenden Bits im Schieberegister gespeichert und werden beim IN-Befehl (und damit deaktivierter Clock) aus dem Schieberegister eingelesen.

Damit sich immer nur eine MMC (maximal 4 habe ich vorgesehen) für die Daten interessiert, muß deren Card-Select-Eingang auf LOW liegen. Dafür ist das 4-Bit-Latch (IC7: 74HC175) vorgesehen. Nach einem Zeddy-Reset sind dessen invertierende Ausgänge (die als Card-Select dienen) HIGH und alles MMCs deselektiert. Schreibt man über ein OUT-Kommando dort die Werte 1, 2, 4 oder 8 rein, geht der entsprechende /CS-MMCx-Ausgang auf LOW und selektiert eine der MMCs.

Die Signale für die MMC-Karten) sind auf 3 10-polige Steckverbindungen (SV2-4) geführt, von denen über Flachbandkabel die TTL-Level-Signale zu einem externen Treiberplatinchen (ähnlich wie beim PIO-MMC-Interface, seihe TM 4/2004) gelangen und dort auf 3V-Level für die MMC umgesetzt werden müssen. Für eine MMC (an /CS-MMC0) ist der Pegelumsetzer (IC8, 74HC4050) gleich onboard vorgesehen (dies ist dann meine "Festplatte C").

Schließlich braucht man noch einen 08/15-Adreßdecoder, der im wesentlichen aus IC1 (74HC138) besteht. Dieser decodiert die bei mir noch unbenutzten I/O-Adressen $0F (MMC-Card-Select Register) und $1F (PIPO-Schieberegister).

Und die unbenutzten Gatter habe ich dann als Puffer für Z80-Signale (/RD, /WR, SYS-CLK) genutzt. Einen Bustreiber zur Pufferung des Datenbusses habe ich nicht vorgesehen, da der PIPO-Baustein "Starkstrom"-Ausgänge (15 LS-TTL-Lasten) besitzt und direkt meinen Bus mit 6 Slots treiben kann. Das reicht wohl auch noch für einen 10-Slot-Bus des ZX96.
Die Z80-Signale kommen bei mir von einer VG64-Steckerleiste (ST1) **mit ECB-BUS-Belegung!** Beim **ZX96** liegen diese Signale auf **anderen Pins**!

Nachdem ich die Schaltung in 3 Nachmittagen auf einer Punktraster-Platine aufgebaut hatte (der eigentliche Logik-Teil paßt auf eine halbe Europakarte, im freien Teil habe ich dann nachträglich noch die Onboard-MMC untergebracht), konnte ich mit meinem OSZI überprüfen, ob die Schaltung sich an die Datenblätter hält: sie tat es.

Und nachdem ich auch die MEFISDOS-MMC-Treiber auf das neue MMC-Interface umgeschrieben hatte, konnte ich bei ersten Tests die Geschwindigkeitssteigerung effektiv sehen: ein "großes" Basic-Programm wird nun in weniger als 1 Sekunde von der MMC geladen!

Da mit diesem MMC-Interface das Timing zur MMC hin fest an Ablauf der OUT- und IN- Befehle geknüpft ist, dürfen diese 2 Befehle nicht unterbrochen oder verzögert werden (also kein /NMI, /INT, /WAIT oder /BUSRQ erlaubt) und der Zeddy muß in dieser Phase nun zwangsläufig im FAST-Modus laufen. Damit das immer sichergestellt ist, auch wenn andere Programme (wie FILEFIND oder FORMAT) nur Teilprogramme aus MEFISDOS aufrufen, habe ich die entsprechenden FAST-Aufrufe in die Basis-MEFISDOS-I/O-Routinen eingebaut (es gibt nun also eine neue MEFISDOS-Version 2.1).

Schließlich habe ich mit einem Basic-Programm (siehe Listing) noch ein paar Messungen der Zeiten für Speichern und Laden (je 20 mal) von 16K großen Dateien in der 6. MEFISDOS-Unterverzeichnisebene auf MMC vorgenommen (siehe Programmlisting am Ende).
Die durchschnittlichen Zeiten pro Load/Save-Aufruf sind:

Speichern (16K): 1,4 Sekunden
Laden (16K): 0,85 Sekunden

Ich glaube, damit kann sich eine MMC mit diesem Interface gegenüber einer Festplatte durchaus sehen lassen!

Und vielleicht kann jemand von Euch die einfachen UND/ ODER/VIELLEICHT-Gatter in GALs oder PALs einbauen und damit den Bauteile-Aufwand noch minimieren?

Ich betreibe mit diesem Interface derzeit maximal 3 MMCs: eine ist immer gesteckt (onboard) und meine "Festplatte C". Eine weitere MMC ist über ein ca. 12 cm langes Flachbandkabel (Kabel so kurz wie möglich halten!) und Pegelwandlerplatinchen (an der Frontseite meines Zeddy montiert) von außen zugänglich und steckbar (meine "Wechselplatte D"). Eine weitere MMC (mit Pegelwandlerplatinchen) kann ich bei Bedarf noch freifliegend" an einen freien Stecker hängen ("Wechselplatte E"). Somit kann ich nun auch einfach Datensicherung von einer MMC auf eine andere vornehmen.

Ach übrigens: die MMC wird sich auch mit dieser Anbindung an den Zeddy zumeist noch langweilen: denn erstens wird sie nur mit ca. 3,25 MHz getaktet, obwohl sie 10 MHz verdauen könnte. Und zweitens, weil zwischen den von/zur MMC übertragenen Bytes noch diverse Pausen bei inaktiver MMC-Clock eingestreut sind, weil der Zeddy inzwischen das Byte in/aus dem Speicher holen muß und seine Register zum Zählen der Bytes und für die Adresse zum Abspeichern aktualisieren muß. Es könnte also noch viiiiiiiiiiiiiiel schneller gehen ...

Dennoch: Viel Spaß beim Nachbasteln, Laden und Speichern auf MMC mit Hochgeschwindigkeit,
wünscht

Siggi

(Siegfried Engel)

Programmlisting zur Zeitermittlung:

```
 10 FAST
 20 FOR F=1 TO 20
 30 SAVE "D:1/2/3/4/5/6/TEST ,50000,15500 -J"
 40 NEXT F
 50 INPUT G$
100 FOR F=1 TO 20
110 LOAD "D:1/2/3/4/5/6/TEST"
120 NEXT F
```

ECB-BUS

ST1

MAC64L

Puffer Kondensatoren

C2 100N
C3 47
C4 100N

3V-Power fuer MMC

D1 1N4004
D2 1N4004
D3 1N4004
C1 47
R3 1K0
C5 100N

ADRESS-DECODER

IC1 74138

/BUSCS
/IORQ
/WR
/RD
SYS-CLK
/RESET
D1

/CS0 ($0F)
/CS1 ($1F)

PIPO-Schieberegister

IC4 74299

PIPO-CLK
DIN-MMC
DOUT-MMC
CLOCK-MMC
SYS-CLK*

MMC-Kartenanschlüsse

FE05-2 SV2
FE05-2 SV3
FE05-2 SV4

VCC-MMC
CLCK-MMC
DIN-MMC
GND
CS-MMC1
CS-MMC2
CS-MMC3

R1 1K0

D7 AA119
D4 AA119
D6 AA119
D5 AA119

DOUTMMC3
DOUTMMC2
DOUTMMC1
DOUTMMC0

Buffer (freie Gatter)

IC6A IC6B IC6C IC6D 7432

/RD*
/PIPO-RD
/PIPOWR
/WR*
/CS0
/L-CLK
SYS-CLK*

Gate und Synchronisation Systemtakt

IC2C IC2D 7408

IC5A IC5B 7474

CLKGATE

MMC-CS-Register

IC7 74175

/CS-MMC0
/CS-MMC1
/CS-MMC2
/CS-MMC3

Power

IC1P IC2P IC3P IC4P IC5P IC6P IC7P IC8P

MMC-Karte

GND
MMC-PIN3
MMC-PIN6
VCC-MMC
MMC-PIN4

R4 56-100K

IC8F 4050

MMC-PIN7

/CS-MMC0
DIN-MMC
CLCK-MMC

IC8A IC8B IC8C IC8D IC8E 4050

MMC-PIN1
MMC-PIN2
MMC-PIN5

R2 470R

Pegelumsetzung 5V/3V