Ansteuerung von Compact Flash Karten im Memory Mapped Mode

von
Ralf Hochhausen

Inhaltsverzeichnis

1 Einleitung

Compact Flash Karten sind mittlerweile relativ preiswert und mit großen Speichern erhältlich. Deshalb werden sie auch im Hobbybereich immer interessanter. Zum Beispiel, wenn man große Datenmengen mit einem Mikrocontroller speichern will oder auch nur bereits gespeichertes wieder auslesen. Beliebte Anwendungen sind zum Beispiel Datenlogger oder MP3-Player. Hierzu findet man auch schon eine ganze Menge Informationen im Internet.

Compact Flash Karten können in drei verschiedenen Modi angesteuert werden. Das sind:

- True IDE Mode
- PC Card Mode
- Memory Mapped Mode.

Im folgenden wird nur die Ansteuerung im Memory Mapped Mode beschrieben, da sich die Karte in diesem Modus sehr gut an gängige Mikrocontroller anschließen lässt. In diesem Artikel wird dabei ein ATMEGA128 Mikrocontroller der Firma Atmel verwendet. Im Prinzip lässt sich jedoch auch jeder andere Controller mit einem externen Businterface auf ähnliche Weise an die Karte adaptieren. Weiterhin wurde der Mikrocontroller mit einem externen 32k SRAM Baustein im Speicher erweitert. Dieser Speicher ist ebenfalls am externen Bus angeschlossen. Die hier vorgestellte Schaltung ist vollständig im Anhang abgedruckt und wird auch in den folgenden Abschnitten genau beschrieben.

2 Bauteileauswahl und Schaltung

2.1 Controllerauswahl

Wie in der Einleitung schon kurz angesprochen, wird im folgenden ein ATMEGA128 verwendet. Dieser Controller ist der derzeit größte aus der MEGA-Reihe von Atmel und eigentlich viel zu groß gewählt für die hier vorgestellte Anwendung. Dieser Controller wurde jedoch ausgewählt um sich zunächst keine Sorgen um einen zu kleinen Speicherplatz machen zu müssen und zudem noch eine ganze Menge andere Features bietet, welche evtl. später noch nützlich sein könnten. Da die Controller der MEGA-Reihe aus Softwaresicht relativ kompatibel zueinander sind ist es später auch ohne größere Probleme noch möglich, auf einen kleineren Controller umzusteigen. Dabei ist im wesentlichen eigentlich nur zu beachten, dass der Controller eine externes Businterface besitzt, da dieses in der Software zur Anwendung kommt. Bezugsquellen zu diesem Controller sind im Anhang zu finden.

Der ATMEGA128 kann mit Quarzfrequenzen von 0 bis 16MHz betrieben werden. Hierbei ist jedoch zu beachten, dass man auch den richtigen Typ verwendet. Atmel bietet zwei verschiedene Versionen dieses Controllers an, den ATMEGA128 und den ATMEGA128L. Der Unterschied zwischen diesen beiden Typen liegt vor allem bei den möglichen Quarzfrequenzen und Spannungen. Der ATMEGA128 kann mit Spannungen von 4.5V bis 5.5V und Quarzfrequenzen bis zu 16MHz getaktet werden, wohingegen die L-Version mit Spannungen von 2.7V bis 5.5V zurecht kommt und mit maximal 8MHz getaktet werden kann. In der hier vorgestellten Schaltung wird die L-Version verwendet. Die Betriebsspannung der Schaltung beträgt 5V und der Controller läuft auf den maximalen 8MHz. Hieraus ergibt sich somit eine Befehlsausführungszeit von minimal 125ns.

2.2 Externes SRAM

Der Mikrocontroller wurde hier über das externe Businterface um zusätzliche 32kB externes SRAM erweitert. Bei der Auswahl dieses Bausteins ist vor allem die Betriebsspannung, sowie die

Zugriffszeit wichtig. Weiterhin ist die notwendige Daten und Adressbusbreite zu beachten. Bei der hier verwendeten Quarzfrequenz und der daraus resultierenden Zykluszeit von 125ns ist diese Zeit jedoch auch eher unkritisch. Der Adressbus sollte maximal 16Bit breit sein, da der MEGA128 solche Bausteine selbständig ansprechen kann. Die Datenbusbreite beträgt 8Bit. Prinzipiell könnten auch größere Speicher angeschlossen werden, dann wird jedoch der Aufwand größer. Man könnte den Speicher dann z.B. in unterschiedliche Pages einteilen, welche über zusätzliche Pins ausgewählt werden.

Der hier verwendete SRAM-Baustein hat die Typenbezeichnung M68AF031A und wird von der Firma ST produziert. Die Zugriffszeit dieses ICs liegt bei 70ns und die mögliche Betriebsspannung beträgt 4.5V bis 5.5V.

Will man diesen Speicher durch einen anderen ersetzen, so ist vor allem auf das Pinning zu achten, da dieses leichte unterschiede im vergleich zu anderen Speicherbausteinen aufweisen könnte. Von der Funktion können jedoch auch andere SRAM-Bausteine verwendet werden.

2.3 Schaltung

Die Schaltung entspricht im Prinzip einer ganz normalen Mikrocontrollerschaltung mit externem Businterface, wie sie auch im Datenblatt des Mikrocontrollers prinzipiell dargestellt ist. Die vollständige Schaltung ist hier im Anhang dargestellt.

Der Mikrocontroller besitzt ein externes Businterface, welches über die beiden Ports A und C nach außen geführt sind. Zusätzlich sind für die Ansteuerung der externen Bausteine noch drei Steuerleitungen notwendig. Das sind:

- ALE (Address Latch Enable)
- RD (Read)
- WR (Write)

Diese Leitungen sind ebenfalls am Controller herausgeführt.

Was nun noch auffällt ist, dass der Controller nur zwei Ports für ein externes Businterface zur Verfügung stellt. D.h. es stehen nur 16 Leitungen zur Verfügung. Der Controller bietet jedoch die Möglichkeit, den Speicher um 64kB zu erweitern. Dazu werden jedoch alleine 16 Leitungen für die Adressierung benötigt. Hinzu kommen weitere acht Leitungen, auf die, die zu lesenden oder schreibenden Daten gelegt werden. Damit fehlen insgesamt acht Leitungen. Die Lösung des Problems liegt beim Multiplexen des Adress- und Datenbusses. Dabei werden die unteren acht Leitungen des Adressbusses sowohl für die Adresscodierung, als auch für die Übertragung der Datenbytes genutzt. Die genaue Funktion wird im folgenden beschrieben.

2.3.1 Funktion des Daten- und Adressbusses

Wie oben schon angedeutet, verwendet der Atmel Controller einen Daten- und Adressbus im Multiplexbetrieb. Das bedeutet, dass verschiedene Leitungen doppelt verwendet werden. Diese Leitungen sind die unteren acht Adressleitungen des Mikrocontrollers. Diese entsprechen dem PORTA des MEGA128. Dieser Port wird sowohl für die Codierung der unteren acht Adressbits, als auch für die Übertragung der Datenbytes verwendet. Da ein externer Speicherbaustein jedoch eine fest anliegende Adresse benötigt um ihn richtig auslesen oder beschreiben zu können, ist es notwendig, vor einem Zugriff die untersten acht Adressbits zwischen zu speichern. Diese Funktion erfüllt in der hiesigen Schaltung der 74HC573. Hierbei handelt es sich um ein Latch, welches zur Speicherung dieser Bits dient. Dieses IC besitzt acht Eingänge, welche mit PORTA des Controllers verbunden werden. Die entsprechenden acht Ausgänge werden mit den unteren acht Adressleitungen der externen Speicher verbunden. Zusätzlich besitzt dieses IC noch zwei Steuerleitungen. Das ist einmal ein Output Enable Eingang (OE), mit dem die Ausgänge aktiv oder

in den hochohmigen Zustand geschaltet werden können, sowie ein Latch Enable Eingang, mit dem die Übernahme der Eingangsdaten gesteuert wird. Legt man am OE-Pin einen low-Pegel an, so werden die Ausgänge aktiv geschaltet. Aus diesem Grund ist dieser Pin dauerhaft auf low gelegt um das IC dauerhaft zu aktivieren. Durch eine high-low-Flanke am LE-Pin werden die an den Eingängen anliegenden Daten durch das Latch gespeichert und an den Ausgängen ausgegeben (abhängig von OE).

Die Ansteuerung des externen Speichers durch den Atmel Controller erfolgt nun auf die folgende Art und Weise:

Zunächst legt der Controller die Adresse auf die beiden Adressports (PORTA und PORTC). Anschließend erzeugt er auf seinem ALE-Pin einen Impuls. Dieser Pin ist mit dem LE-Eingang des 74HC573 verbunden. Bei der fallenden Flanke an ALE werden die an PORTA anliegenden acht Bits (die untersten acht Adressbits A0…A7) vom Adresslatch übernommen und an den Ausgängen ausgegeben. Somit liegt nun die vollständige Adresse an den Speicherbausteinen an und es kann nun dieser Speicherplatz gelesen oder beschrieben werden. Da die untersten acht Adressbits nun durch das Latch gespeichert sind, kann der Controller seinen PORTA wieder anderweitig verwenden. Nun verwendet er diesen Port als Datenport. Sollen nun Daten in den Speicher geschrieben werden, so legt er im folgenden die zu schreibenden Daten auf den Port. Bei einem Lesezugriff konfiguriert er diesen Port als Eingang um und wartet auf Daten. Sind diese Vorgänge abgeschlossen, so erzeugt der Controller eine fallende Flanke auf einer der beiden Leitungen RD oder WR, abhängig davon, ob gelesen (RD) oder geschrieben (WR) werden soll. Hierdurch wird das externe Speicher-IC angewiesen, Daten von der angegebenen Adresse auf seinen Datenport zu legen oder die auf dem Datenbus liegenden Daten auf die entsprechende Adresse zu übernehmen.

Bei der bisherigen Beschreibung wurde noch eins vergessen: Woher weiß ein Speicher-IC, dass es gerade angesprochen wurde? Hierfür werden Chip Select Signale (CS) verwendet. D.h. dass jeder Speicher noch einen zusätzlichen Eingang besitzt, über den er mittels eines low-Pegels ausgewählt werden kann. Hierdurch ist es zum Beispiel auch möglich, einen gesamten Adressbereich in unterschiedliche Blöcke einzuteilen, was auch hier mit dem SRAM und der CF-Karte gemacht wird.

2.3.2 Erzeugung der Chip Select Signale

Die hier vorgestellte Schaltung verwendet zwei verschiedene Speicher. Das ist zum einen ein externes SRAM und zum anderen die Compact Flash Karte. Das bedeutet, dass zwei Chip Select Signale erzeugt werden müssen. Hierzu wird in der Schaltung ein 74HC138 IC verwendet. Dieses IC besitzt drei Eingänge und acht Ausgänge und legt entsprechend des an den Eingängen anliegenden Codes einen seiner Ausgänge auf low. Diese Ausgänge können nun als CS-Signale verwendet werden. Die Eingänge dieses ICs sind mit den obersten drei Adressleitungen des Controllers verbunden. Hierdurch wird der gesamte 64kB umfassende Adressbereich des ATMEGA128 in mögliche acht verschiedene Bereiche eingeteilt. Jeder dieser Bereiche umfasst dabei 8kB. Da hier nur zwei verschiedene Speicher vorhanden sind, werden hier jedoch nicht alle Signale verwendet. Weiterhin wird das CS-Signal des SRAMs leicht anders generiert. Das dieser Speicher 32kB groß ist (halber Speicherbereich des MEGA128), kann hier die Adressleitung A15 als CS-Signal verwendet werden. Dabei wird dass externe SRAM immer dann angesprochen, wenn die Leitung A15 auf low liegt. Somit belegt es den Speicherbereich von 0x0000 bis 0x7FFF. Die untersten 4kB dieses Bereiches können jedoch nicht angesprochen werden, da sie vom internen Speicher des MEGA128 belegt werden. Dies ist also ein kleiner Nachteil aber nicht unbedingt so schlimm, da der Speicher auch so völlig ausreichend ist. Würde man die Adressleitung nicht verwenden, so könnte man auch mehrere Ausgänge des 74HC138 über ein weiteres Gatter zusammenfassen, was jedoch aufwendiger ist und deshalb meistens nicht gemacht wird. Als letztes wird für die CF-Karte noch ein entsprechendes Signal benötig, damit auch diese Angesteuert werden kann. Da das SRAM schon den Speicherbereich bis 0x7FFF belegt muss diese in den

oberen 32kB umfassenden Bereich gelegt werden. Hier wird Ausgang Y7 des 74HC138 als CS-Signal der CF-Karte verwendet. Damit platziert sie sich auf dem Speicherbereich von 0xE000 bis 0xFFFF, wobei sie eigentlich nur 16Adressen dieses Bereiches wirklich benötigt, aber dazu später mehr.

Damit ist der Speicherbereich für diese Anwendung eingeteilt. Für Erweiterungen stehen hier noch drei Bereiche von 0x8000 bis 0xDFFF zur Verfügung. Hier könnte man zum Beispiel noch andere Speicher wie EEPROMS usw. anschließen.

2.3.3 Serielle Schnittstelle

Neben den oben beschriebenen Schaltungsteilen ist weiterhin eine serielle Schnittstelle eingefügt. Diese entspricht der Standardschaltung mit einem IC von Maxim, welches für die Umsetzung der Pegel notwendig ist. Diese Schnittstelle ist besonders für Debugzwecke sehr hilfreich, da man die Schaltung so einfach an einen PC anschließen kann. Dieser Teil soll jedoch hier nicht weiter beschrieben werden. Die einzige Besonderheit ist hier, dass ein MAX203 verwendet wurde, der ohne externe Kapazitäten für die Spannungswandlung auskommt.

2.3.4 Restliche Schaltungsteile

Im großen und ganzen ist nun die Schaltung vollständig beschrieben. Die restlichen Schaltungsteile bestehen nun nur noch aus Pfostenleisten, über die alle Pins des Controllers nach außen herausgeführt wurden. Die Schaltung und die daraus entwickelte Platine kann somit auch als eine Art Experimentierplatine verwendet werden. Die Belegungen der einzelnen Steckverbinder entsprechen hierbei denen des STK500, was ein einfaches Verbinden dieser beiden Schaltungen ermöglicht. Es ist somit z.B. möglich die LEDs oder Taster des STK500 zu verwenden. Weiterhin ist die Belegung des Programmiersteckers mit dem des STK500 identisch.

2.3.5 Schlussbemerkung zu Schaltung und Layout

In diesem Artikel wird nur die Schaltung veröffentlicht, da zu Beginn noch kleine Fehler in der Schaltung und somit auch im Layout vorhanden waren, welche bisher nur in der Schaltung korrigiert wurden. Weiterhin war das Layout teilweise mit 8mil Strukturen geroutet, was zu erhöhtem Aufwand bei der Platinenerstellung führt.

3 Ansteuerung der CF-Karte und Software

Im folgenden wird die Ansteuerung von Compact Flash Karten im Memory Mapped Mode beschrieben. Die anderen Modi der Karten werden hier nicht betrachtet. Im folgenden werden Funktionen zum Lesen und Schreiben von Sektoren in ihrer Funktionsweise vorgestellt.

3.1 Timing

Da das externe SRAM, als auch die CF-Karte am externen Adress- und Datenbus des Controllers betrieben werden, ist zunächst das Timing dieses Busses zu konfigurieren. Dazu sind in den Datenblättern der einzelnen Bausteine entsprechende Timingdiagramme zu finden. Eine Darstellung solcher Diagramme für Lese- und Schreibzugriffe ist in den folgenden Abbildungen zu sehen. Es wurden nicht alle Diagramme eingefügt, da sie sich gleichen.

Abbildung 3.1-1: Timingdiagramm bei Lesezugriff

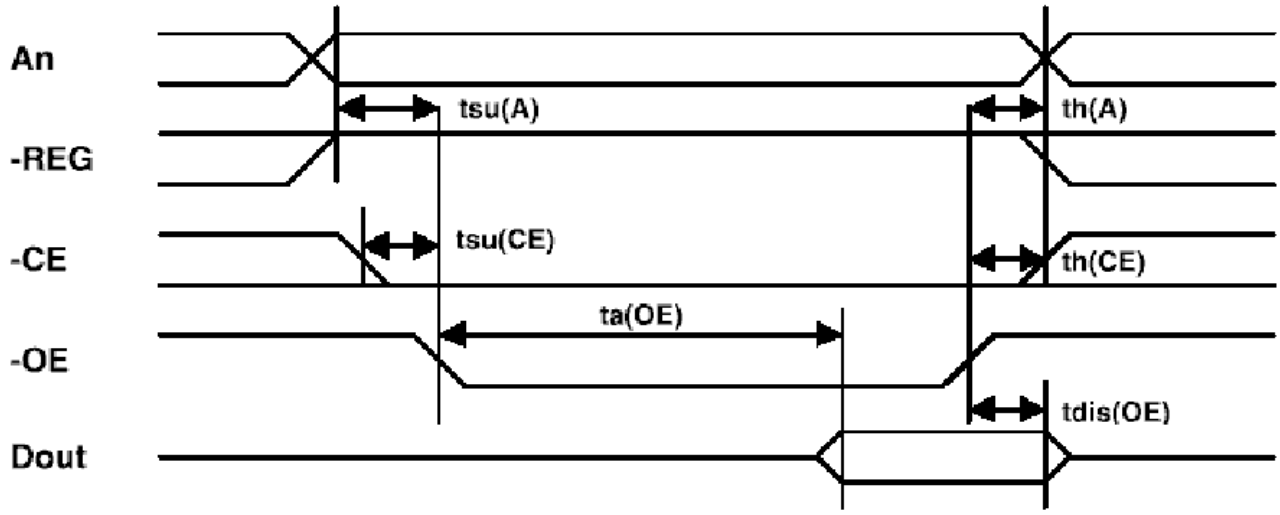
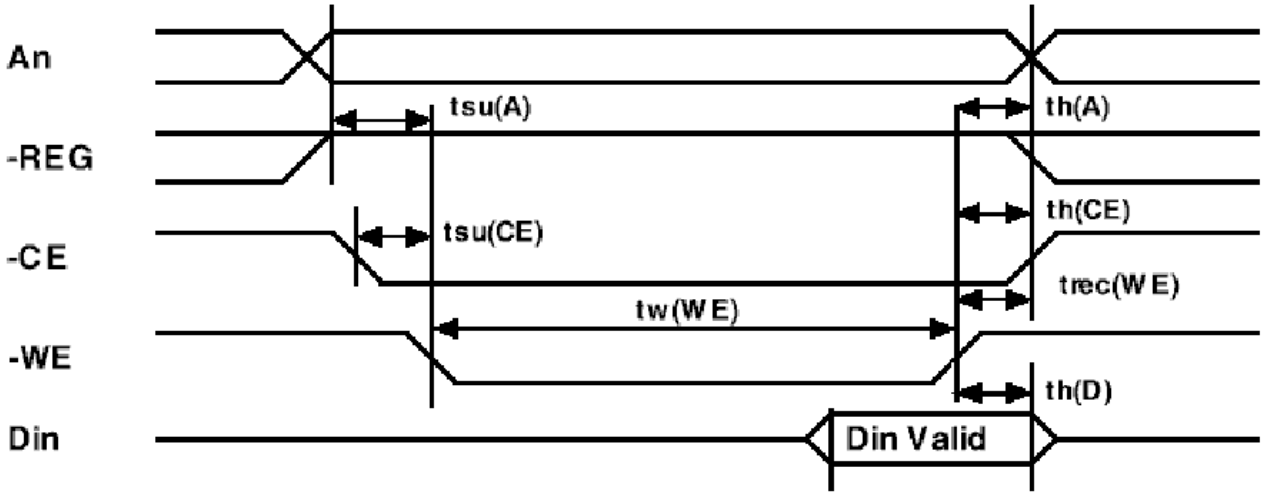


Abbildung 3.1-2: Timingdiagramm bei Schreibzugriff



Die oben dargestellten Timingdiagramme sind aus dem Datenblatt der Compact Flash Karte entnommen. Hierbei ist zu beachten, dass das REG-Signal in der Schaltung statisch auf high-Pegel gelegt ist, da es für die Ansteuerung, wie sie hier durchgeführt wird, nicht notwendig ist. Die entsprechenden Zeitwerte sind den Datenblättern der einzelnen ICs zu entnehmen. Bei der gewählten Quarzfrequenz von 8MHz für den Mikrocontroller und der daraus resultierenden Zykluszeit ergibt sich, dass alle weiteren Speicher schnell genug sind um ohne sog. Waitstates angesteuert zu werden. Ein Waitstate ist dabei die Anzahl der Takte, die der Mikrocontroller das RD- oder WR-Signal auf low zieht, bis der die Signale wieder auf den high-Zustand legt. Diese Funktion des Controllers ist dann hilfreich, wenn langsamere Speicher verwendet werden sollen. Der Controller bietet den Speichern damit die Möglichkeit, sich gewissermaßen auf den Datentransfer vorzubereiten. Dies ist hier jedoch nicht notwendig. Der ATMEGA128 bietet zudem noch die Sonderfunktion, dass unterschiedliche Speicherbereiche mit unterschiedlichen Waitstate-Konfigurationen betrieben werden können. Damit ist es möglich, schnelle und langsame Speicher in einer Schaltung zu mischen.
Die Konfiguration des Mikrocontrollers für die Ansteuerung der externen Speicher sieht im Quellcode wie folgt aus:

```
//Initialization of the external RAM area
XMCRA = 0x00;
XMCRB = 0x00;
MCUCR = 0x80;
```

Hierbei wird zunächst festgelegt, dass keine Speichersegmentierung verwendet wird und dass keine Waitstates notwendig sind. Anschließend wird im MCU Contol Register der externe Speicher freigeschaltet, indem das SRE-Bit gesetzt wird. Nachdem diese Konfiguration im Programm vorgenommen wurde, stellt sich aus Sicht der Software ein Zugriff auf den externen Speicherbereich genauso dar, wie einer auf den internen.

3.1.1 Verwendung von Controllern ohne externes Businterface

Will man CF-Karten oder andere Speicher an einen Mikrocontroller ohne externes Businterface anschließen, so stellt sich die Ansteuerung etwas komplizierter dar, als beim ATMEGA128. Bei diesen Controllern ist man selber dafür verantwortlich, die entsprechenden Signalfolgen an den Ports zu erzeugen. Dies spiegelt sich dann vor allem in der Laufzeit der Routinen wieder, da für die Erzeugung mehrere Befehle notwendig sind.

Bei einem Schreibzugriff ist es laut Timingdiagramm zunächst notwendig, die entsprechende Adresse auf den Adressbus zu legen. Wenn man wie hier ein Adresslatch verwendet, um die untersten Adressleitungen auch als Datenbus zu verwenden, so muss man nun durch einen Impuls am ALE-Pin des Adresslatches die unteren Adressbits in das Latch übernehmen. Anschließend kann man dann die beiden Signale CE und WE auf low ziehen, um einmal den entsprechenden Speicher auszuwählen und ihm mitzuteilen, dass man Daten lesen will. Nun ist es notwendig, eine entsprechende Zeit zu warten, die der Speicher benötigt um Daten zu lesen (tw(WE)). In dieser Zeit können die zu schreibenden Daten auf den Datenbus gelegt werden. Mit der Anschließenden low-high-Flanke werden die Daten dann vom Speicher übernommen. Mit dem Anlegen des high-Pegels an CE ist der Schreibzugriff dann abgeschlossen.

Ein lesender Zugriff stellt sich auf ähnliche Art und Weise dar. Hierbei ist vor allem notwendig, dass die Zeit eingehalten wird, die der Speicher benötigt, um gültige Daten auf den Datenbus zu legen (ta(OE)). Nach der abschließenden low-high-Flanke gehen die Datenports des Speichers nach tdis(OE) wieder in den hochohmigen Zustand (Tristate).

3.2 Treibersoftware für die CF-Karte

3.2.1 Funktion des CF-Karten Interfaces

Obwohl die Compact Flash Karte, einen riesigen Speicher besitzt (hier z.B. 128MB), den ein ATMEGA gar nicht adressieren könnte, belegt das Interface zur CF-Karte nur 16Adressen im externen Speicherbereich des Mikrocontrollers. Das Lesen und Beschreiben des Speichers der Karte erfolgt dabei über Steuerregister. Der Zugriff erfolgt dabei immer sektor- (512 Byte) und nicht byteweise. Die Register der CF-Karte sind, wie oben schon angegeben, ab Adresse 0xE000 im externen Speicher des Controllers angeordnet und wie folgt gegliedert:

-REG	A10	A9 to A4	A3	A2	A1	A0	Offset	-OE=L	-WE=L
1	0	x	0	0	0	0	0H	Data register	Data register
1	0	x	0	0	0	1	1H	Error register	Feature register
1	0	x	0	0	1	0	2H	Sector count register	Sector count register
1	0	x	0	0	1	1	3H	Sector number register	Sector number register
1	0	x	0	1	0	0	4H	Cylinder low register	Cylinder low register
1	0	x	0	1	0	1	5H	Cylinder high register	Cylinder high register
1	0	x	0	1	1	0	6H	Drive head register	Drive head register
1	0	x	0	1	1	1	7H	Status register	Command register
1	0	x	1	0	0	0	8H	Dup. even data register	Dup. even data register
1	0	x	1	0	0	1	9H	Dup. odd data register	Dup. odd data register
1	0	x	1	1	0	1	DH	Dup. error register	Dup. feature register
1	0	x	1	1	1	0	EH	Alt. status register	Device control register
1	0	x	1	1	1	1	FH	Drive address register	Reserved
1	1	x	x	x	x	0	8H	Even data register	Even data register
1	1	x	x	x	x	1	9H	Odd data register	Odd data register

Abbildung 3.2-3: Registerübersicht der CF-Karte

Wie oben zu erkennen ist, werden nur die Adressleitungen A0 bis A3 für die Ansteuerung benötigt. Die anderen Adressleitungen können fest beschaltet werden. Ebenso das Signal REG. Je nach Zugriffsart (lesend oder schreibend) ergibt sich bei verschiedenen Registern der Karte eine unterschiedliche Funktion für eine Registeradresse (siehe Error/Feature Register).
Da sich die Compact Flash Karte aus Sicht des Mikrocontrollers wie ein ganz normaler Speicher verhält ist es aus Softwaresicht nun relativ einfach auf die unterschiedlichen Register der Karte zuzugreifen. Der Zugriff unterscheidet sich nicht von einem normalen Zugriff auf ein internes Register des Controllers. Damit die einzelnen Register der CF-Karte in der Software namentlich verfügbar sind, ist es zunächst notwendig, alle Register zu definieren und mit der entsprechenden Adresse im externen Speicherbereich des Controllers zu verknüpfen. Dies sieht im Quelltext wie folgt aus:

```
#define CFC_DATA_REG     (*(volatile uchar*)(0xE000l)) //Data register
#define CFC_ERROR_REG    (*(volatile uchar*)(0xE001l)) //Error register (read)
#define CFC_FEATURE_REG (*(volatile uchar*)(0xE001l)) //Feature register (write)
#define CFC_SECTOR_COUNT_REG  (*(volatile uchar*)(0xE002l))//Sector count reg.
#define CFC_SECTOR_NUMBER_REG (*(volatile uchar*)(0xE003l))//Sector number
register    LBA07..LBA00
#define CFC_CYLINDER_LOW_REG  (*(volatile uchar*)(0xE004l))//Cylinder low
```

```
register    LBA15..LBA08
#define CFC_CYLINDER_HIGH_REG (*(volatile uchar*)(0xE005l))//Cylinder high
register    LBA23..LBA16
#define CFC_DRIVE_HEAD_REG    (*(volatile uchar*)(0xE006l))//Drive head register
LBA27..LBA24
#define CFC_STATUS_REG  (*(volatile uchar*)(0xE007l))//Status register (read)
#define CFC_COMMAND_REG (*(volatile uchar*)(0xE007l))//Command register (write)
```

Mit diesen Definitionen können die einzelnen Register der CF-Karte nun einfach mit Hilfe von einfachen Zuweisungen beschrieben und gelesen werden.
Wie sieht aber nun ein Zugriff auf ein Register in der Praxis aus? Im folgenden ist als Beispiel der Zugriff auf das Kommandoregister der Karte dargestellt.

```
CFC_COMMAND_REG = CFC_CMD_READ_SECTOR;
```

Man kann eigentlich gar nicht erkennen, dass hier auf ein Register der CF-Karte zugegriffen wird. Dies liegt an den oben beschriebenen Umständen im Zusammenhang mit der Verwendung des externen Businterface. Nun könnte man sagen, dass hier viel Assemblercode durch die Verwendung der Programmiersprache C verloren geht. Wenn man sich jedoch das vom Compiler erzeugte Listing ansieht, so stellt man fest, dass das ganze im Assembler auch nicht aufwendiger aussieht (rot markierter Code):

```
201:CF_Card_IF.c  ****  CFC_COMMAND_REG = CFC_CMD_READ_SECTOR;
 208                          .stabn 68,0,201,.LM24-CFC_vReadSector
 209                    .LM24:
 210 0074 80E2                ldi r24,lo8(32)
 211 0076 8093 07E0           sts -8185,r24
```

Nun aber zur Funktionsweise dieses Codes. Bei dem Ausdruck , `CFC_CMD_READ_SECTOR`' handelt es sich um ein Makro, welches über ein #define definiert wurde. Hierbei handelt es sich schlicht um den Wert 0x20, also 32 im Dezimalformat. Dieser Wert wird nun zunächst in das Register r24 der Controllers geladen. Anschließend wird dieser Wert über einen direkten Speicherzugriff auf die Adresse –8185 geschrieben. Diese Adresse –8185 entspricht im Hexadezimalformat der Adresse 0xE007, also der Adresse des Kommandoregisters der CF-Karte, wie man an den obigen Definitionen erkennen kann. Die Adresse dieses Registers wird nun wie folgt zusammengesetzt:

Wie bei der Beschreibung der Hardware schon angesprochen, dienen die oberen drei Bits der Adresse der Erzeugung der Chip-Select-Signale. Diese sind hier alle gleich ‚1'. Damit wird vom 74HC138-IC das Chip-Select-Signal für die Compact Flash Karte erzeugt. Die weiteren Adressbits von A12 bis A4 haben für die Ansteuerung der Karte keine Bedeutung. Die entsprechenden Leitungen sind auch nicht mit der Karte verbunden. Erst die Bits A3 bis A0 sind wieder relevant. Hierüber wird ein entsprechendes Register ausgewählt. Welche Bits für welches Register gesetzt oder gelöscht werden muss, ist aus der obigen Tabelle zu entnehmen. Für das Kommandoregister ergibt sich für diese Adressbits die Binärkombination b'0111', also 0x7. In Verbindung mit einem schreibenden Zugriff ergibt sich damit, dass das Kommandoregister der Karte angesprochen wird. Würde man lesend auf diese Adresse zugreifen, so würde nicht etwa der Inhalt des Kommandoregisters von der Karte gelesen, stattdessen liefert die Karte den Inhalt des Statusregisters. Es ist also immer auch die Art des Zugriffs (lesend oder schreibend) zu beachten.

3.2.2 Das Statusregister der CF-Karte

Das Statusregister der CF-Karte nimmt eine besondere Rolle bei der Ansteuerung der Karte ein. Über dieses Register teilt die Karte mit, ob sie bereit ist, Kommandos zu verarbeiten oder ob sogar ein Fehler bei einer Aktion aufgetreten ist. Zur Abfrage der Betriebsbereitschaft der Karte dienen die beiden Statusbits 6 und 7 (BSY = Busy und DRVY = Drive Ready). Ein Fehlerzustand wird über das Bit Nr. 0 angezeigt (Errorflag).

3.2.3 Initialisierung

Nach dem anlegen der Spannung an die Schaltung benötigt die Compact Flash Karte eine gewisse Zeit um sich selbst zu initialisieren. Diese Zeit lag bei den Versuchen mit einer Toshiba CF-Karte in einem Bereich von ca. 200ms bis 400ms. Damit also keine Fehlfunktionen auftreten, ist es zunächst notwendig, die Zeit abzuwarten, bis die Karte bereit ist. Dies geschieht durch ständiges Lesen des Statusregisters und Überprüfen des Busy Flags der CF-Karte. Im Quelltext sieht das ganze wie folgt aus:

```
//*****************************************************************************
//***
//*** Function:         void CFC_vInit(void)
//***
//*** Return Type:      void
//***
//*** Parameters:       none
//***
//*** Function is used to initialize the comapct flash card. It waites until the
//*** busy flag of the card is cleared
//***
//*****************************************************************************
void CFC_vInit(void)
{
      while(CFC_STATUS_REG & CFC_STATUS_BSY){}        //Wait for CF card ready
}
```

Wie man erkennen kann, wird in dieser Funktion immer das Statusregister gelesen und mit `CFC_STATUS_BSY` bitweise ‚und'-verknüpft. Bei , `CFC_STATUS_BSY`' handelt es sich um ein Makro, welches mit 0x80 ersetzt wird. Hiermit wird also immer das Busy-Flag der Karte herausmaskiert. Ist das Bit gesetzt, ist die Karte also noch beschäftigt, so wird das Register wieder gelesen und erneut geprüft, ob die Karte bereit ist. Hierbei ist zu beachten, dass dieser Code nicht ganz ungefährlich ist. Es kann nämlich passieren, dass sich der Controller an dieser Stelle aufhängt, wenn z.B. keine Karte eingelegt ist oder die Karte defekt ist und somit das Busy-Flag immer mit ‚1' gelesen wird. Eine saubere Lösung würde nach einer bestimmten Zeit die Initialisierung abbrechen und eine Fehlermeldung generieren.

3.2.4 Lesen eines Sektors der CF-Karte

Wie oben schon kurz erwähnt, erfolgt der Zugriff auf eine Compact Flash Karte immer sektorweise. Das bedeutet, dass es nicht möglich ist, auf ein einzelnes Byte auf der Karte direkt zuzugreifen. Es müssen also immer direkt 512Byte von der Karte gelesen werden. Ebenso stellt sich auch der schreibende Zugriff auf die Karte dar.

Um einen bestimmten Sektor auf der Karte zu lesen, ist es zunächst nötig, die zu lesende Adresse festzulegen. Hierbei gibt es zwei verschiedene Möglichkeiten, den Speicher der Compact Flash Karte zu adressieren. Die Adressierungsarten entsprechen dabei denen, welche auch bei Festplatten für PCs verwendet werden. Bei der ersten Möglichkeit wird die Adresse über die Angabe des Lesekopfes, der Zylinder-, sowie der Sektornummer festgelegt. Diese Adressierungsart lässt sich vom Aufbau normaler Festplatten ableiten. Hierbei sind normalerweise mehrere einzelne Platten übereinander angeordnet, welche über verschiedene Leseköpfe gelesen werden. Da diese Platten rund sind, lassen sich kreisförmige Bahnen auf den einzelnen Platten in sog. Zylinder aufteilen.

Schließlich lässt sich jede kreisförmige Spur noch in mehrere Sektoren á 512Byte einteilen. Damit folgt aus diesem Aufbau auch die Adressierung der Festplatte. Obwohl die Compact Flash Karte diese Elemente gar nicht besitzt, sieht die Adressierung genau so aus. Die endgültige Umsetzung auf eine wirkliche Adresse der CF-Karte übernimmt anschließend der IDE-Controller, der die Schnittstelle zwischen dem Mikrocontroller und der CF-Karte bildet. Dieser Controller befindet sich mit auf der Karte. Da man mit dieser Adressierungsart nicht genug Speicher adressieren kann, ist es weiterhin möglich, den Speicher im sog. LBA-Modus zu adressieren. LBA steht hierbei für Logical Block Address. Der Unterschied zur vorherigen Adressierung ist, dass keine Unterscheidung zwischen Kopf/Zylinder/Sektor mehr gemacht wird. Statt dessen werden die ganzen Sektoren hintereinander linear durchnumeriert. Hieraus ergibt sich dann für jeden Sektor eine 28Bit breite Adresse. Dieser Modus wird im folgenden auch für die Adressierung der Karte verwendet.

Um einen Sektor der Karte zu lesen, ist es also zunächst notwendig der CF-Karte mitzuteilen, welche Adresse überhaupt gelesen werden soll. Dies wird durch das schreiben der folgenden vier Register durchgeführt, wodurch dann die LBA zusammengesetzt wird:

- Sector Number Register LBA07…LBA00
- Cylinder Low Register LBA15…LBA08
- Cylinder High Register LBA23…LBA16
- Drive Head Register LBA27…LBA24

Die Wertigkeit der einzelnen Registerbits ist oben mit angegeben. Die Reihenfolge, in der die Register beschrieben werden, ist nicht relevant. Was jedoch beachtet werden muss, ist dass nur vier Bits des Drive Head Registers für die logische Adresse verwendet werden. Die restlichen Bits dieses Registers dienen teilweise als Steuerbits. Die Belegung des Registers lautet wie folgt:

- Bit7 ‚1'
- Bit6 LBA
- Bit5 ‚1'
- Bit4 DRV
- Bit3…Bit0 LBA27…LBA24

Durch das Setzen des LBA-Bits kann hierbei der LBA Adressierungsmodus aktiviert werden. Das DRV-Bit dient der Auswahl einer bestimmten Karte, falls mehrere CF-Karten an einem Interface betrieben werden. Dieses Bit kann jedoch meistens auf ‚0' gesetzt werden. So auch hier.

Das Setzen der Adresse übernimmt in der Software die folgende Funktion:

```
void vSetLBAAddress(lba_address lbaAddress_)
```

Sie ist im Quelltext wie folgt realisiert:

```
//*****************************************************************************
//*** Function:         void vSetLBAAddress(lba_address ulAddress_)
//***
//*** Return Type:      void
//***
//*** Parameters:       lba_address ulAddress_:      LBA Address
//***
//*** Write the LBA address into the CF card registers
//*****************************************************************************
```

```
void vSetLBAAddress(lba_address lbaAddress_)
{
      while(!(CFC_STATUS_REG & CFC_STATUS_BSY)){}
      CFC_SECTOR_NUMBER_REG  = lbaAddress_.byte[0];
      CFC_CYLINDER_LOW_REG   = lbaAddress_.byte[1];
      CFC_CYLINDER_HIGH_REG  = lbaAddress_.byte[2];
      CFC_DRIVE_HEAD_REG = ((lbaAddress_.byte[3]&0x0F)|0xA0| CFC_DRV_HEAD_LBA);
      return;
}
```

Wie man erkennen kann, übernimmt die Funktion nur das Beschreiben der Adressregister. Im Drive Head Register wird zusätzlich noch das LBA-Bit gesetzt. Eine kleine Besonderheit ergibt sich aus der ersten while-Schleife. Hier wird das Busy-Flag des Statusregisters so lange gelesen, bis die Karte bereit ist, die Adressregister zu beschreiben. Die zu Setzende Adresse wird der Funktion als Übergabeparameter mitgeteilt. Hierbei handelt es sich um eine Union, welche es erlaubt, wahlweise als 32Bit long Wert oder byteweise auf die Adresse zuzugreifen.

Nachdem nun die zu lesende Adresse in die entsprechenden Register der CF-Karte eingetragen wurde, ist es notwendig, der Karte mitzuteilen, wie viele Sektoren gelesen werden sollen und dass man überhaupt Daten lesen will. Das bedeutet, dass man in das Sektor Number Register die Anzahl der zu lesenden Sektoren einträgt (hier also 1) und anschließend das Kommando ‚Sektor Lesen' in das Command Register schreibt. Dieses Kommando entspricht der Zahl 0x20 oder 0x21. Mit dem Schreiben des Kommandos in das Command Register werden die zuvor geschriebenen Werte gültig und die Karte beginnt mit dem Lesevorgang. Das Abrufen der 512 Datenbytes erfolgt nun durch lesen des Datenregisters (Adresse 0xE000). Dabei wird bei jedem Lesenden Zugriff auf das Datenregister ein neues Datenbyte des Sektors auf den Datenbus gelegt. Der Lesevorgang ist abgeschlossen, wenn 512 mal das Datenregister gelesen wurde.

Die gesamte Lesefunktion ist im Quelltext wie folgt realisiert:

```
//*****************************************************************************
//***
//*** Function:         void CFC_vReadSector(lba_address lbaSectorAddress,
//                                           uchar* pucBuffer)
//***
//*** Return Type:      void
//***
//*** Parameters:       lba_address lbaSectorAddress:
//                      Address of the Sector that has to be read
//***                   uchar* pucBuffer: Pointer to the memory location,
//***                   where the data has to be stored
//***
//*** Function to read one sector of the compact flash card
//***
//*****************************************************************************
void CFC_vReadSector(lba_address lbaSectorAddress_, uchar* pucBuffer_)
{
      uint uiByteCounter_ = 0;
```

```
	vSetLBAAddress(lbaSectorAddress_);  //Write LBA address into registers

	CFC_SECTOR_COUNT_REG = 0x01;        //Read one sector

	CFC_COMMAND_REG = CFC_CMD_READ_SECTOR;//Write command for reading sector

	while(CFC_STATUS_REG != 0x58){}     //Wait for CF card ready

	for(uiByteCounter_ = 0; uiByteCounter_ < 512; uiByteCounter_++)
	{
		*pucBuffer_ = CFC_DATA_REG;
		pucBuffer_++;
	}
	return;
}
```

Am Quelltext ist zu erkennen, dass zunächst die entsprechende Adresse in die oben genannten Register geschrieben werden. Dann wird über das Sector Count Register festgelegt, dass nur ein Sektor gelesen werden soll. Will man mehr Sektoren auf einmal lesen, so wird hier die entsprechende Anzahl eingetragen. Anschließend wird der Karte über das Kommando ‚CFC_CMD_READ_SECTOR' mitgeteilt, dass ein Sektor gelesen werden soll. Hierbei handelt es sich wiederum um ein Makro, welches für der Hexadezimalwert 0x20 steht. Die Anschließende while-Schleife wartet wiederum, bis die Karte bereit ist, die Daten über das Datenregister auszugeben. Der eigentliche Lesevorgang findet dann in der anschließenden for-Schleife statt.

3.3 Schreiben eines Sektors auf die CF-Karte

Der Schreibvorgang auf eine CF-Karte präsentiert sich auf ähnliche Art und Weise, wie der Lesevorgang. Auch hier kann wieder nur ein kompletter Sektor auf einmal geschrieben werden. Der einzige Unterschied in der Schreibfunktion besteht darin, dass in das Command-Register der Befehl zum schreiben eingetragen wird und anschließend 512 mal schreibend auf das Datenregister zugegriffen wird. Hierbei wird vom Mikrocontroller immer ein neues Datenbyte aus dem Puffer gelesen und in das Datenregister der CF-Karte geschrieben. Der Quellcode sieht wie folgt aus:

```
//*****************************************************************************
//***
//*** Function:
//**** void CFC_vWriteSector(lba_address lbaSectorAddress, uchar* pucBuffer)
//***
//*** Return Type: void
//***
//*** Parameters: lba_address lbaSectorAddress: Address of the Sector that has
//***                                           to be written
//***             uchar* pucBuffer:             Pointer to the memory location
//***                                           with data,that has to be stored
//***
```

```
//*** Function to write one sector on the compact flash card
//***
//*****************************************************************************
void CFC_vWriteSector(lba_address lbaSectorAddress_, uchar* pucBuffer_)
{
      uint uiByteCounter_ = 0;

      vSetLBAAddress(lbaSectorAddress_);  //Write LBA address into registers

      CFC_SECTOR_COUNT_REG = 0x01;        //Write one sector

      CFC_COMMAND_REG = CFC_CMD_WRITE_SECTOR; //Write command for writing sector

      while(CFC_STATUS_REG != 0x58){}           //Wait for CF card ready

      for(uiByteCounter_ = 0; uiByteCounter_ < 512; uiByteCounter_++)
      {
            CFC_DATA_REG = *pucBuffer_;
            pucBuffer_++;
      }
      return;
}
```

3.4 Testsoftware

Um den Treiber für die Compact Flash Karte testen zu können wurde ein Textprogramm geschrieben, mit dem beliebige Sektoren gelesen werden können und anschleißend über die serielle Schnittstelle an einen PC übertragen werden. Die Software ist in der ZIP-Datei CF_TEST.ZIP zusammengefasst. Die zu lesende Adresse kann über eine Variable in der Quelltextdatei ‚system.c' festgelegt werden.

4 Anhang

4.1 Informationsquellen

Wichtige Informationen zu Compact Flash Karten und deren Ansteuerung sind unter den folgenden Adressen zu finden:

www.sandisk.com

www.glyn.de

www.mikrocontroller.net

4.2 Bezugquellen

Damit man CF-Karten vernünftig anschließen kann, ist es sinnvoll einen fertigen Sockel zu verwenden. Mögliche Quellen sind dabei:

www.segor.de	(Yamaichi-Sockel)
www.avxcorp.com	(leifert evtl. Samples von CF-Sockeln)

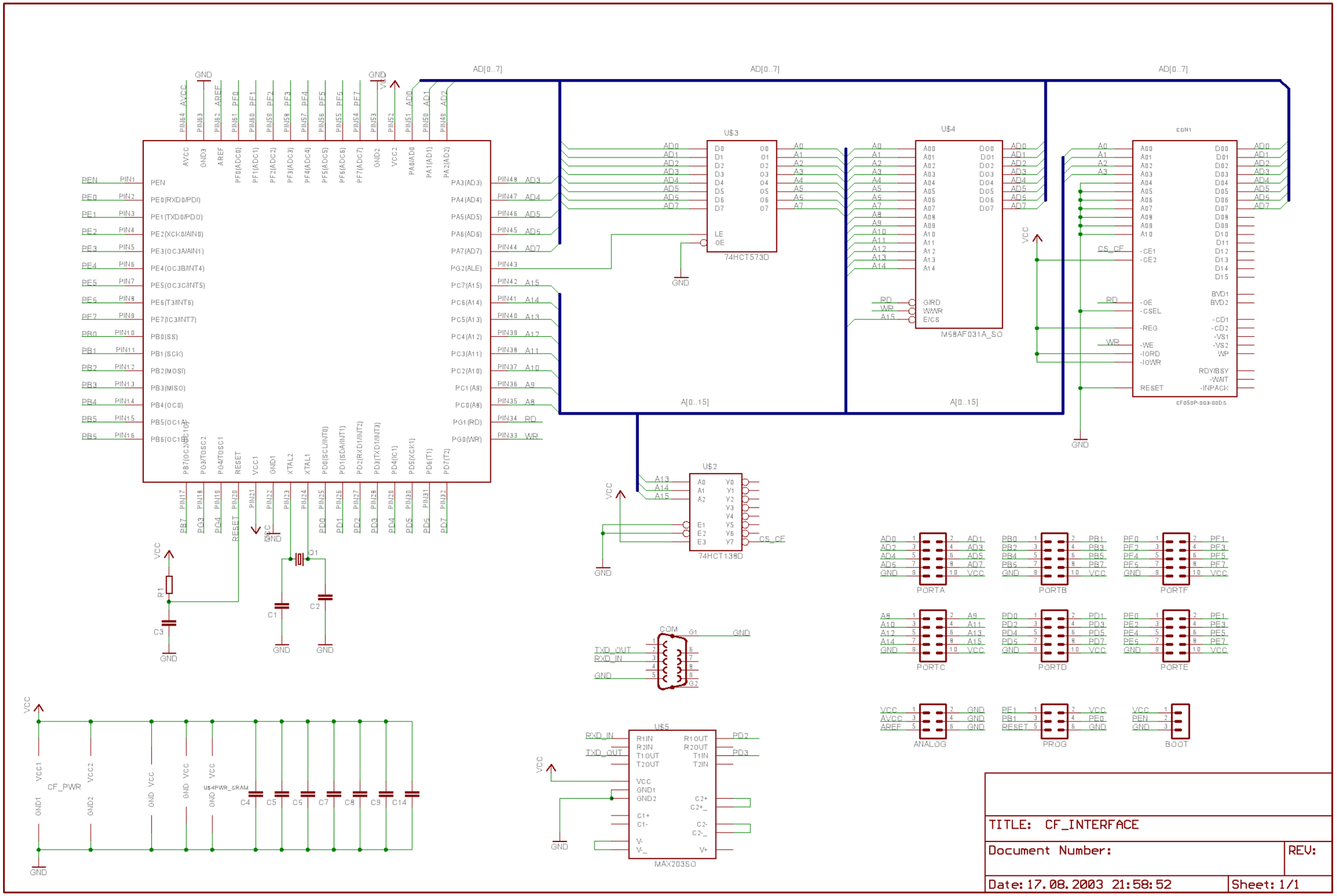
Bei der hier vorgestellten Schaltung wurde ein Sockel der Firma Yamaichi verwendet. Dieser Sockel besitzt keinen Auswurfhebel. Er ist bei der Firma Segor erhältlich und kostet ca. 6,50Euro. Als Bastellösung bieten sich auch Pfostenleisten mit dem entsprechenden Rastermaß an. Diese sind jedoch auch nur schlecht erhältlich und ein Anschluss mit diesen Leisten ist relativ fehleranfällig. Der Nachteil des Yamaichi-Sockels ist das kleine Rastermaß. Es ist deshalb notwendig eine Platine fertigen zu lassen.

Die restlichen Bauelementen (bis auf den Speicher) sind relativ leicht über verschiedene Elektronikhändler zu beziehen. Mögliche Quellen sind z.B.:

www.reichelt.de

www.kessler-elektronik.de

Das Speicher-IC ist leider nicht so einfach erhältlich. Es kann jedoch durch ein gängiges anderes Speicher-IC vom Typ 62256 ersetzt werden. Hierbei ist jedoch ein evtl. anderes Pinning zu beachten!



4.3 Schaltplan