Memory-mapped mode on a Compact-Flash RAM card

I finally got a working interface between the 68HC912 SBC and a compact flash card. This is how I did it.

COMPACT FLASH CARDS

Compact Flash (CF) cards are produced by several manufacturers to a standard published by the CompactFlash Association. This standard covers both the hardware (connector specification and pinout) and the software (use of the pins) associated with these cards. Several manufacturers (SanDisk, Silicon Systems) publish a spec for these cards. Unfortunately, none of these specs contain any description of how you actually operate the CF, they merely describe the registers and command structures and leave it to you to figure out what combination in what order will do what you want.

I found an application note (from Silicon Storage Technology) describing the use of a CF card with a 8051-compatible single board computer (SBC) that they also offer. The hardware interface and software I came up with resulted from study of both this app note and the CF specs from the companies noted above.

CF cards are often used as static hard drives and must interface with a PC as though they were ATA hard drives, with a File Allocation Table, cylinders, sectors, and heads. In order to permit standard file command structures, each CF card contains a processor to interface the host computer to the card's actual RAM. Note that the disk structure form of usage requires data be stored in discrete chunks -- 512 bytes per sector in this case.

There is also a "Memory Mapped" mode available, wherein one can address the card as a large linear memory. This mode is much simpler to use than the ATA disk mode. It has the same limitation on sector sizes, however. You must read or write 512 byte chunks at a time.

The CF communicates with the host via a set of memory-mapped registers. The memory-mapped mode register set consists of 8 sequential registers:

\CE2	\CE1	\REG	A3	A2	A1	A0	READ	WRITE
1	0	1	0	0	0	0	DATA	DATA
1	0	1	0	0	0	1	ERROR	FEATURE
1	0	1	0	0	1	0	SECTOR CNT	SECTOR CNT
1	0	1	0	0	1	1	SECTOR NUM	SECTOR NUM
1	0	1	0	1	0	0	CYLINDER LO	CYLINDER LO
1	0	1	0	1	0	1	CYLINDER HI	CYLINDER HI
1	0	1	0	1	1	0	DRIVE HEAD	DRIVE HEAD
1	0	1	0	1	1	1	STATUS	COMMAND

These registers are 'addressed' by a combination of address lines (A3-0) and special input signals (\CE1, \CE2, and \REG). The proper combination of these lines will permit reading or writing any of these registers.

Data is read and written through the DATA register -- 512 bytes at at time. Once started, the correct number of bytes must be read or written. The location on the CF of the data is set by the Sector number and the Cylinder (HI-MID-LO) value. One can setup a linear addressing mode (Logical Block Addressing, or LBA) that uses the last 4 bits of the DRIVE HEAD register, the two bytes of the CYLINDER registers, and the SECTOR NUMber as a 28-bit address (LBA27-0). Writing $Ex to DRIVE HEAD register enables the LBA addressing mode, sets the DRIVE # to 0, and sets the addresses LBA27-24 to x.

The SECTOR CouNT tells the CF card how many sectors you plan to read or write. The simplest choice is 1 but some (all?) CF cards have a buffer size that is most efficient for writing data.

One can read/write data as 16-bit words or as 8-bit bytes. Since the 68HC912 SBC has an 8-bit data bus, I have elected to use the 8-bit mode. The FEATURE register (write-only) is used to enable 8-bit transfers by writing it with $01.

The COMMAND register is used to cause the internal CF processor to execute commands. For example, one would write $EF to COMMAND after setting the FEATURES register.

The COMMAND register is also used to tell the CF internal processor that you are about to read or write a sector of data. The last thing before reading or writing is to send $20 (read) or $30 (write) to the COMMAND register.

STATUS is used to determine if the CF is busy (bit 7 set) or if an error has occurred (bit 0 set). This register should be read after every command; wait in a loop until bit 7 is cleared.

Writing a $0C and then a $08 to DEVice ConTRoL register causes a soft reset of the CF card. This may not be necessary every time, but it is a sure way to start with the card in a known condition.

HARDWARE

A 74HC688 8-bit comparator is used to memory-map the CF card to address $7EA0-7EAF. I chose this address (in RAM space) because I use a DS1244 RTC-NVRAM for RAM and the clock registers are in the $7FFxx space. This puts all of my memory-mapped stuff in one place in RAM. I copied the memory-mapped peripheral method from New Micros where the upper 8 address bits are compared to a fixed number in a 74HC688 comparator. This lets me select any 256-byte section of memory to use for a memory-mapped peripheral such as the CF card. When any address in this range is addressed, the '688 output goes low. This signal is diode-ORed to \MEMDIS, to signal the CPU that an external device (not RAM) has been selected.

The output of the '688 is also used, with E-clock and R/W, to generate the \READ and \WRITE strobes to the CF card. Reading an address in the $7Exx space generates a \READ to the CF card, writing an address in the $7Exx space generates a \WRITE.

I connected address lines as follows (also see the schematic):

68HC912	Compact Flash
A7	\CE2
A6	\CE1
A5	\REG
GND	A10-A15
A4	A9
GND	A4-A8
A0-A3	A0-A3

With this arrangement, the DATA registers for the CF occur at $7EA0-7EAF and the control pins (\CE1, \CE2, \REG) are set appropriately for the memory-mapped mode.

Address lines A10 and A8-4 on the CF are not used and are grounded. Data lines D7-0 are connected to the CPU data lines.

\RDY can be used to verify the CF is ready to accept commands by sensing it on an ADC channel; the STATUS register can also be used and this is the mode employed in my code.

\CD1 can be used to sense the presence of a CF card. This line goes low when a card is installed in the socket. This flag is provided since CF cards can be hot-swapped in most applications. My application doesn't require this and so I don't bother to sense this line. The schematic shows it connected to the ADC, however.

SOFTWARE:

This is the tricky part, of course, I ended up copying the code from an application note by SST, Inc. that showed how to control a CF card in Memory Mode from an 8051 microprocessor. After considerable study (since I don't know the assembly code mnemonics for this processor), I doped out the following sequence:

1. Initalize the CF card by doing a soft reset.
2. Setup the CF by enabling 8-bit transfers and setting Drive # = 0.

Then, to read (write) to the CF, which must be done in sectors of 512 bytes at a time,

3. Setup the address registers (sector/cylinder/head or LBA address).
4. Send a read (write) command to the command register.
5. Read (write) 512 bytes to (from) a RAM buffer.

This sequence is included in the test code, attached.

The sample code does the card initialization. Then a buffer in RAM is filled with a bit pattern and a byte from the buffer displayed. This buffer is written to the card. The buffer is zeroed and then the same sector on the CF card is written into the buffer. Then a byte from the buffer is displayed and the contents compared against the original bit pattern. The sector address is incremented by one and this process repeats 44 more times.

```
cold

exram                   ( enable external RAM

hex

1000 dp !               ( load code into external RAM

( FILE TESTCF1.4th -- code to test CF-RAM on TAPS-NG CPU/IO card
( 19 May 2006

( Runs from RAM

( Port use is as follows:
(
(    PP0   is RUN/STOP - enables transmissions or noise datafiles
(    PP1   is SHTDWN   - controls power to CPU card peripherals
(                        high = off
(    PP2   is \ADC2    - MAX111 ADC
(    PP3   is \ADC1    - MAX1067 ADC
(    PP4   is \DAC     - AD5300BRT DAC
(    PP5   is FSELECT  - to select between F1 and F2 in DDS
(    PP6   is \DDS     - frames 16-bit transfers to DDS
(    PP7   is \XGATE   - transmit gate pulse

(    PT0   is FREQ1    - Frequency input to timer
(    PT1   is FREQ2    - Frequency input to timer
(    PT2-4 is MUX      - M0-M2 selects acoustic channels
```

```
(    PT5-7 is GAIN     - G0-G2 selects post-IF gains

(    PDLC5 is TRANS    - controls power to Transmitters
(    PDLC6 is INSTS    - controls power to External Instruments

( -------------------------- CONSTANTS -------------------------------
hex

006f constant portad              ( on-board ADC data register
0056 constant portp               ( Port P data register
0057 constant ddrp                ( Port P data direction register
00ae constant portt               ( Port T data register
00af constant ddrt                ( Port T data direction register
00fe constant pdlc                ( Port DLC Data register
00ff constant ddrdlc              ( Port DLC data direction register

0080 constant tios                ( Timer I/O select register
0084 constant tcnt                ( Timer counter register
0086 constant tscr                ( Timer system control register
008b constant tctl4               ( Timer control register 4
008d constant tmsk2               ( Timer interrupt mask register 2
008e constant tflg1               ( Timer interrupt flag register 1
008f constant tflg2               ( Timer interrupt flag register 2
0090 constant tc0                 ( Timer input compare register 1
0091 constant tc1                 ( Timer input compare register 2

00d0 constant spocr1              ( spi 1 control register
00d1 constant spocr2              ( spi 2 control register
00d2 constant spobr               ( spi baud rate register
00d3 constant sposr               ( spi status register
00d5 constant spodr               ( spi data register

00d6 constant ports               ( Port S data register
00d7 constant ddrs                ( Port S data direction register

(      Addresses decoded to R/W to CF: $7EA0 - 7EAF
7ea0 constant cfdata      ( byte-port to flash
7ea1 constant cferror     ( read-only
7ea1 constant features    ( write only
7ea2 constant sctrcnt     ( sector count register
7ea3 constant LBA0        ( LBA0:7   = Sector Number
7ea4 constant LBA1        ( LBA8:15  = Cylinder low
7ea5 constant LBA2        ( LBA16:23 = Cylinder high
7ea6 constant LBA3        ( LBA24:27 = head count
7ea7 constant status      ( read only
7ea7 constant command     ( write only
7eae constant dev-ctrl    ( write only
7e90 constant config      ( r/w config register
7e96 constant socket      ( r/w socket & copy register

-1   constant true
0    constant false

( ------------------------ VARIABLES -------------------------------

variable errorcount       ( holds # of errors in compare
variable testbyte         ( holds byte to write to CF
2variable fadr            ( long address of sector in FLASH
2variable tfadr           ( hold starting flash address

(  REGISTER + 0 = errors
(  REGISTER + 1 = sector count
(  REGISTER + 2 = sector #     or LBA  0:7
(  REGISTER + 3 = cylinder low or LBA  8:15
(  REGISTER + 4 = cylinder hi  or LBA 16:23
```

```
(  REGISTER + 5 = head count   or LBA 24:27
(  REGISTER + 6 = status

create registers          ( place to download CF reg's
7 allot

create buff
100 allot                 ( Text Input Buffer

create buffer
512 allot                 ( compact flash sector buffer

hex
( ---------------------- COMPACT FLASH RAM ROUTINES --------------------------


code-sub write-cf
3b c,               ( pshd   ; save D accumulator
34 c,               ( pshx   ; save X register
35 c,               ( pshy

( ;   Setup Sector Address values

4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
8601 ,              ( ldaa #01
7a c, sctrcnt ,     ( staa sctrcnt

cd c, FADR ,        ( ldy #FADR           ; point Y at FLASH address
4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
a643 ,              ( ldaa 3,y            ; LB of address
7a c, lba0 ,        ( staa lba0

4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
a642 ,              ( ldaa 2,y            ; ML byte of address
7a c, lba1 ,        ( staa lba1

4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
a641 ,              ( ldaa 1,y            ; set Low byte of ADR
7a c, lba2 ,        ( staa lba2

4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
a640 ,              ( ldaa 0,y            ; set High byte of ADR
840f ,              ( anda #$0f           ; only use low 4 bits
8ae0 ,              ( ora #$e0            ; set CF1, LBA enabled
7a c, lba3 ,        ( staa lba3

4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
8630 ,              ( ldaa #write
7a c, command ,     ( staa command        ; set mode to WRITE

(   Write a block of 512 bytes into FLASH, checking RDY=1 before each write

ce c, buffer ,      ( ldx #buffer         ; X points at BUFFER
cd c, cfdata ,      ( ldy #cfdata         ; Y points at DATA port to FLASH
c600 ,              ( ldb #256            ; B counts words
( wordmove:
4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
180a , 3070 ,       ( movb 1,x+,1,y+      ; move a byte, post-increment
03 c,               ( dey                 ; adjust Y back to DATA
4f6f , 01fc ,       ( brclr portad,#$01,* ; ensure RDY is high
180a , 3070 ,       ( movb 1,x+,1,y+      ; move a byte, post-increment
03 c,               ( dey
04 c, 31eb ,        ( dbne b,wordmove     ; continue till all are moved

31 c,               ( puly
```

```
30 c,              ( pulx               ; recover registers
3a c,              ( puld               ; and accumulator
3d c,              ( rts
end-code

code-sub read-cf
3b c,              ( pshd   ; save D accumulator
34 c,              ( pshx   ; save X register
35 c,              ( pshy

( ;   Setup Sector Address values

4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
8601 ,             ( ldaa #01
7a c, sctrcnt ,    ( staa sctrcnt

cd c, FADR ,       ( ldy #FADR           ; point Y at FLASH address
4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
a643 ,             ( ldaa 3,y            ; LB of address
7a c, lba0 ,       ( staa lba0

4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
a642 ,             ( ldaa 2,y            ; ML byte of address
7a c, lba1 ,       ( staa lba1

4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
a641 ,             ( ldaa 1,y            ; set Low byte of ADR
7a c, lba2 ,       ( staa lba2

4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
a640 ,             ( ldaa 0,y            ; set High byte of ADR
840f ,             ( anda #$0f           ; only use low 4 bits
8ae0 ,             ( ora #$e0            ; set CF1, LBA enabled
7a c, lba3 ,       ( staa lba3           ; set high byte of ADR

4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
8620 ,             ( ldaa #read
7a c, command ,    ( staa command        ; set mode to READ

(   Read a block of 512 bytes into RAM, checking RDY=1 before each write

cd c, buffer ,     ( ldy #buffer         ; X points at BUFFER
c600 ,             ( ldb #256            ; B counts words
( wordmove:
4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
b6 c, cfdata ,     ( ldaa cfdata        ; read a byte
6a40 ,             ( staa 0,y           ; save to BUFFER
4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
b6 c, cfdata ,     ( ldaa cfdata
6a41 ,             ( staa 1,y
0202 ,             ( iny 2X
04 c, 31e9 ,       ( dbne b,wordmove     ; continue till all are moved

31 c,              ( puly
30 c,              ( pulx               ; recover registers
3a c,              ( puld               ; and accumulator
3d c,              ( rts
end-code

(  Read the ID sector on the compact flash card

code-sub read-cf-id
3b c,              ( pshd   ; save D accumulator
34 c,              ( pshx   ; save X register
35 c,              ( pshy
```

```
(  Send IDENTITY code

4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
86ec ,             ( ldaa #identify
7a c, command ,    ( staa command        ; set mode to READ

(   Read a block of 512 bytes into RAM, checking RDY=1 before each write

cd c, buffer ,     ( ldy #buffer         ; X points at BUFFER
c600 ,             ( ldb #256            ; B counts words
( wordmove:
4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
b6 c, cfdata ,     ( ldaa cfdata         ; read a byte
6a41 ,             ( staa 1,y            ; save to BUFFER
4f6f , 01fc ,      ( brclr portad,#$01,* ; ensure RDY is high
b6 c, cfdata ,     ( ldaa cfdata
6a40 ,             ( staa 0,y            ; note little-endian data format
0202 ,             ( iny 2X
04 c, 31e9 ,       ( dbne b,wordmove     ; continue till all are moved

31 c,              ( puly
30 c,              ( pulx                ; recover registers
3a c,              ( puld                ; and accumulator
3d c,              ( rts
end-code
( ---------------------- SPI INIT SUBROUTINE -----------------------------

code-sub spi-init        ( code to setup spi i/o
  4cd6 , 80 c,           ( bset ports,#$80  ; set SS line high
  180b , e0 c, ddrs ,    ( movb #$e0,ddrs   ; Configure PORT S ddr
  180b , 02 c, spobr ,   ( movb #$02,spobr  ; set SCLK rate = 1 MHz
  180b , 12 c, spocr1 ,  ( movb #$12,spocr1 ; MSTR=1, CPOL=CPHA=0
  180b , 08 c, spocr2 ,  ( movb #$08,spocr2 ; SPI 2 outputs, active pullups
  96d3 ,                 ( ldaa sposr       ; 1st step to clear SPIF flag
  96d5 ,                 ( ldaa spodr       ; 2nd step to clear SPIF flag
  4cd0 , 40 c,           ( bset spocr1,#$40 ; enable spi
  3d c,                  ( rts
end-code

code-sub spi-off
  4dd0 , 40 c,           ( bclr spocr1,#$40 ; disable spi
  3d c,
end-code

( ---------------------- UTILITY ROUTINES --------------------------------

code-sub power-on
  4cfe , 20 c,           ( bset pdlc,#$20   ; enable board power
  3d c,                  ( rts
end-code

code-sub power-off
  4dfe , 20 c,           ( bclr pdlc,#$20   ; disable board power
  3d c,                  ( rts
end-code

code-sub insts-on
  4cfe , 40 c,           ( bset pdlc,#$40   ; enable insts power
  3d c,                  ( rts
end-code

code-sub insts-off
  4dfe , 40 c,           ( bclr pdlc,#$40   ; disable insts power
  3d c,                  ( rts
```

```
end-code

code-sub cpu-off
  180b , 0a c, portp ,   ( movb #$0a,portp   ; power off, bits = 0
  3d c,                  ( rts
end-code

code-sub cpu-on
  180b , fd c, portp ,   ( movb #$fd,portp   ; power on, bits = 1
  3d c,                  ( rts
end-code

( ---------------------- FORTH CODE ---------------------------------

: bs
  20 0 do 08 emit loop
;
: tic-init
     fc tios c!              ( Timer bits 0-1 as input capture
     33 tmsk2 c!             ( set timer to 1 uS ticks
     80 tscr c!              ( enable timer
;
decimal
: wait-on
    20000 0 do
       3 0 do I drop loop
    loop
;
hex
: busy? ( - )
  begin
    status c@              ( read status register
    80 and                 ( mask bit 7
    0=                     ( is it clear?
  until
;
: get-regs                 ( read CF registers
  busy?
  cferror c@ registers c!
  sctrcnt c@ registers 1+ c!
  LBA3 c@ registers 2 + c!
  LBA2 c@ registers 3 + c!
  LBA1 c@ registers 4 + c!
  LBA0 c@ registers 5 + c!
  status c@ registers 6 + c!
;
: disp-regs
   hex
   ."    error code    = " cferror c@ . cr
   ."    sector count  = " sctrcnt c@ . cr
   ."    Hi  address   = " LBA3 c@ . cr
   ."    Mid address   = " LBA2 c@ . cr
   ."    Mid address   = " LBA1 c@ . cr
   ."    lo  address   = " LBA0 c@ . cr
   ."    status        = " status c@ . cr decimal
;
hex
: setup-CF
  cr ." Setting up CF card ...  "
  busy?
  01 features c!          ( enable 8-bit data transfer
  busy?
  fadr c@ LBA0 c!         ( install current sector address
  busy?
  fadr 1+ c@ LBA1 c!
```

```
  busy?
  fadr 2 + c@ LBA2 c!
  busy?
  fadr 3 + c@ 0f and
  e0 or LBA3 c!           ( Drive 0, LBA addressing
  busy?
  ef command c!           ( execute commands just loaded
  busy?
  ."  done" cr
;
decimal
: disp-status
    cr ."         BAE SYSTEMS " cr
       ."     TAPS New Generation " cr
       ."    CF-RAM TEST CODE v1.0" cr cr
;
( -----------------------  TEST ROUTINES ---------------------------------
decimal
: fill-buff ( N - )
   512 0 do dup buffer i + c! loop
   drop
;
: show-buff
  buffer 40 dump
;
: compare ( n - flg)
  0 errorcount !
  512 0 do
    dup buffer I + c@
    = not if
      1 errorcount +!
    then
  loop drop
  errorcount @ 0=              ( set flg=TRUE if no errors
;
: write45
  cr hex
  45 0 do                      ( write 45 sectors of data
     testbyte c@ fill-buff     ( fill buffer with byte pattern
     buffer 18 + c@ .
     write-cf                  ( write a sector to CF card
     fadr 2@ 1. d+ fadr 2!     ( increment sector address
  loop
  cr decimal
;
: read45
  cr hex
  45 0 do                      ( read 45 sectors of data
     0 fill-buff               ( clear buffer
     read-cf                   ( read a sector from CF card
     buffer 37 + c@ .
     testbyte c@ compare
     drop
(    if 46 emit else 42 emit then
     fadr 2@ 1. d+ fadr 2!     ( increment sector address
  loop
  cr decimal
;
: ud. ( d - )               ( print unsigned double word
    <# #s #>  type
;
: cfram-test ( n - n )
    cr ." Testing Compact Flash RAM card " cr
    decimal
    read-cf-id 4 spaces
```

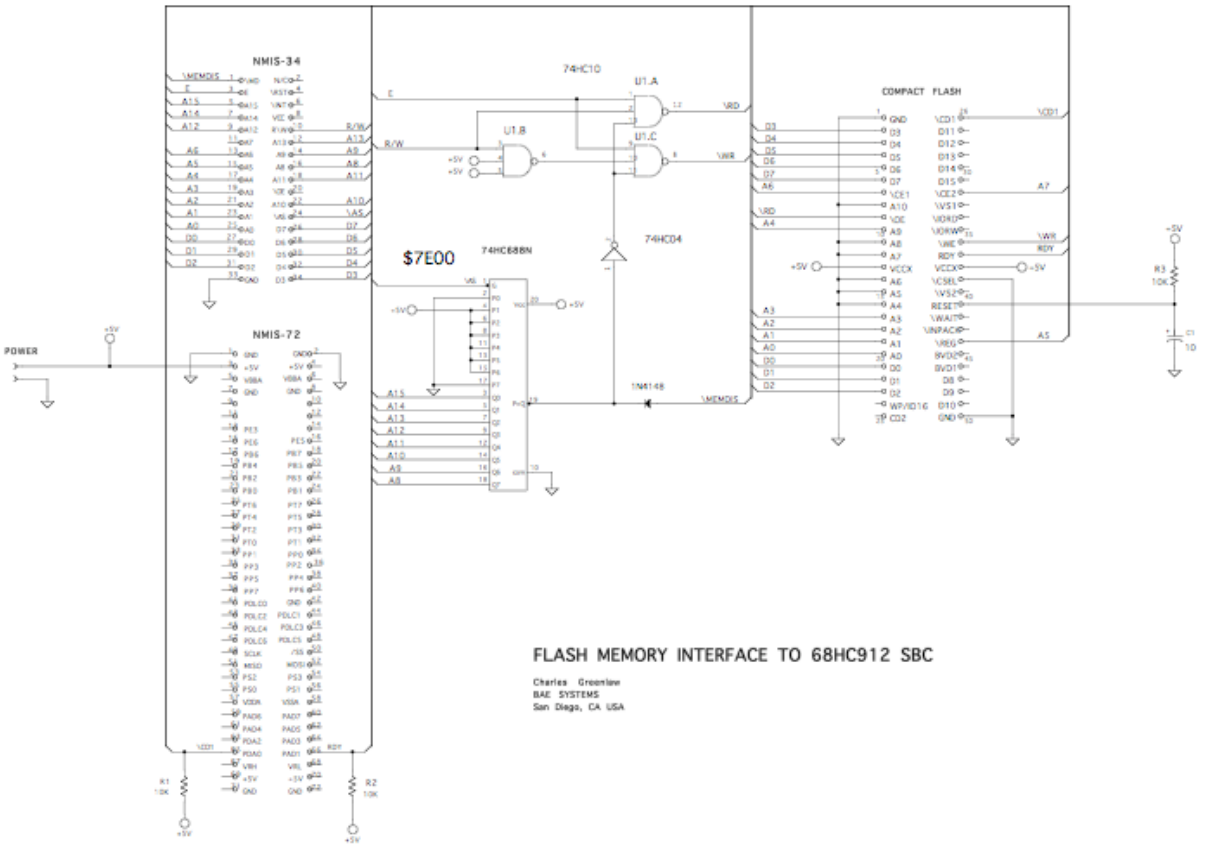
```
    buffer  2 + @ dup . ." cylinders " cr 4 spaces
    buffer  6 + @ dup . ." heads " cr 4 spaces
    *
    buffer 10 + @ dup . ." bytes/sector " cr 4 spaces
    buffer 12 + @ dup . ." sectors/track " cr 4 spaces
    * um* d. ."  bytes available " cr 4 spaces
    buffer 42 + @ . ." sectors/buffer " cr cr

    get-regs
    disp-regs

    fadr 2@ tfadr 2!                ( save in temp loc'n
    setup-cf
    ." Starting sector address for write = " fadr 2@ d.
    write45
    ." Ending sector address = " fadr 2@ d. cr
    tfadr 2@ fadr 2!                ( put starting LBA in FADR
    setup-cf
    ." Starting sector address for read = " fadr 2@ d.
    read45
    ." Ending sector address = " fadr 2@ d. cr
    ." Press any key to continue " key drop cr
;
: do-choice ( n - flag )
    dup 1 = if cfram-test then
        0 = if true else false then
;
: get-choice ( - n )
    cr ."  Choice = " key dup emit
    48 -
;
: disp-menu
    ."        TEST MENU" cr
    ."       testcf1.4th" cr
    ."  0             END" cr
    ."  1             CF RAM test" cr cr
;
hex
: init
    exram                   ( hopefully redundant
    001a 00c0 !             ( set baud rate to 19200
    0a portp c!             ( set ADC1 high, CPU peripherals off
    00 pdlc c!              ( set external power off
    00 portt c!             ( set MUX to CH 0, GAIN to 0
    ff ddrp c!              ( set Port P DDR
    ff ddrdlc c!            ( set Port DLC DDR
    fc ddrt c!              ( set Port T DDR; bits 0-1 are inputs
    fc tios c!              ( set Port T TIOS same
    86 c@ 80 and 86 c!      ( set TEN bit of TSCR, start timer
    01 80 c!                ( set output compare Timer 1
    power-off               ( disable ext power
    cpu-off                 ( disable cpu peripherals
    0c dev-ctrl c!          ( soft reset of CF
    08 dev-ctrl c!
    30000 0 do I drop loop  ( short delay
    get-regs
    setup-CF
    2 places
    decimal
;
decimal
: main
    init spi-init spi-off cr disp-status cr
    power-on cpu-on wait-on
    0002. fadr 2!                   ( put starting LBA in FADR
```

```
    139 testbyte c!
    setup-cf
    begin
      disp-menu
      get-choice
      do-choice
    until cr
    power-off cpu-off
;

(  TESTCF1.4th - TAPS NG CPU TEST CODE - 19 May 2006

here u. cr
decimal
```

SCHEMATIC:

The schematic below shows the interface I used to drive a compact-flash RAM card as a memory-mapped peripheral using the code above. J12 is the I/O connector on both the 68HC11 and 68HC12 cards from New Micros Inc. The code shown is for a 68HC12 card but there is no reason that a 68HC11 CPU card can't drive a CF-flash card.

Separate /READ and /WRITE lines are de-coded from R\W and the E clock lines and NAND'ed with the output of the comparator. Thus a simple read or write operation, within the address limits imposed by the comparator, can be used to send data to and from the CF-flash card.


NMIS-34
74HC10
U1.A
U1.B
U1.C
74HC04
74HC688N
$7E00
1N4148
COMPACT FLASH
NMIS-72
POWER
FLASH MEMORY INTERFACE TO 68HC912 SBC
Charles Greenlaw
BAE SYSTEMS
San Diego, CA USA